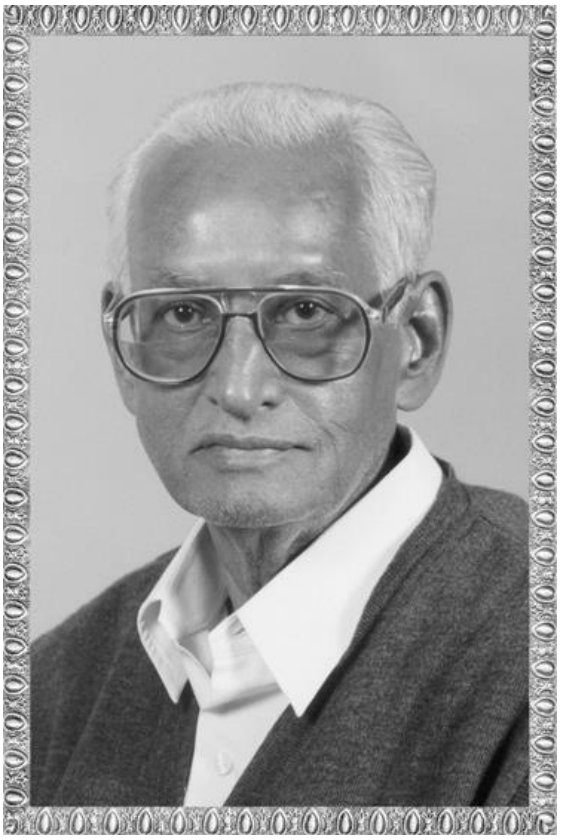
SHREE H. N. SHUKLA GROUP OF COLLEGES

(AffiliatedtoSaurashtraUniversity&GujaratTechnologicalUniversity)



Lt. Shree Chimanbhai Shukla

BCA/ BSCIT SEM - 4

Programming With Java

ShreeH.N.ShuklaCollegeCampus,
StreetNo.2,VaishaliNagar,
Nr.AmrapaliRailwayUnder
Bridge,
RaiyaRoad,Rajkot.
Ph.(0281)2440478,2472590

ShreeH.N.ShuklaCollegeCampus,
StreetNo.2,VaishaliNagar,
Nr.AmrapaliRailwayUnder Bridge,
RaiyaRoad,Rajkot.
Ph.(0281)2471645

Website:https://hnsgroupofcolleges.org/

CS-07: ADVANCE C AND DATA STRUCUTRE				
Ch No.	**Topic**	**Details**	**Marks**	
1	History, Introduction and Language Basics	• History and features of java • Java editions • JDK,JVM,JRE • JDK Tools • Compiling and executing basic java program • Java IDE(netbeans and eclipse) • Datatypes • Java Tokens • Operators • Type casting • Decision Statements • Looping Statements • Jumping Statements • Array • Comand Line Argument Array		
	classes and objects	• OOP Concept (class,objects,encapsulation,inheritance,polymorphism) • Creating and using class with members • Constructor • Finalize() method • Static and non static members • Overloading(constructor & method) • VarArgs		

Q-1 Write a short note on history of java

Ans

- Java is an object oriented programming language developed by James Gosling in early 1990s.
- Initially this language was intended to develop digital devices like setup box, television etc. Originally C++ was considered to be used in this project but the idea was rejected for several reasons (like c++ requires more memory, supports pointers etc)
- James Gosling endeavored to alter and expand C++. This project was named as “Green talk” by James gosling and his team and later became to be known as “Oak”.
- The name Oak was used by Gosling after an Oak tree but they had to later rename it as "Java" “s Oak was already a trademark by Oak Technologies.
- The name Java originates from espresso bean.
- Java was created on the principles like portable, robust, platform independent, high performance, multithread etc.
- Currently, java is used in internet programming, mobile devices, games, business solutions etc.
- Many different versions of java came like: JdkBeta, JDK 1.0, JDK 1.1, JSE 1.3, JSE 1.4 etc.

Q-2 Explain features of Java

Ans:

Features:

1) Simple:
   - Java inherits the C/C++ syntax and many of the object oriented features of C++,
   - Java supports OOP and does not support pointer which makes it simpler.

2) Security:
   - Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:
     - **No explicit pointer**
     - **Java Programs run inside a virtual machine sandbox**

3) Portable:
   - Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation. It can be implemented in any OS.

4) Object-Oriented:

- Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.
- Basic concepts of OOPs are:

    1) Object
    2) Class
    3) Inheritance
    4) Polymorphism
    5) Abstraction
    6) Encapsulation

5) Robust (Healthy, Strong):

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.

6) Multi-threaded:

- A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area.

7) Architecture-neutral:

- Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.
- In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

8) Interpreted:

- Usually a computer language is either compiled or interpreted. Java combines these approaches thus making java a two-stage system.
- Java compiler translates source code into byte code instructions. Byte codes are not machine instructions and so java interpreter generates machine code that can be directly executed by the machine that is running the java program.

9) High Performance: .

- Java does not support pointer which increases the performance. It is portable, secure.

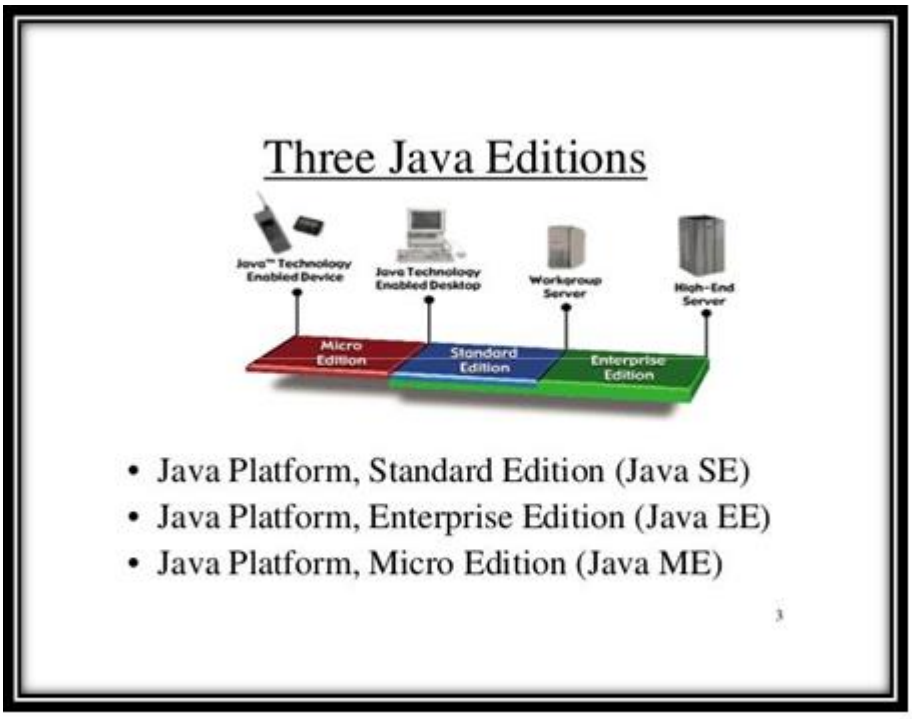
10) Dynamic:

- Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand

MCQ		
Sr No.	**Question**	**Answer**
1	**Which type of `architecture java supports?**	**Neutral**
2	**Robust means…….**	**Strong/healthy**
3	**Java is ……programming language**	**OOP**
4	**Class and object are the concepts of …….**	**OOP**

Q-2 Explain editions of Java

Ans:



1) Java Platform, Micro Edition (J2ME):

- JEME stands for Java 2 platform Micro Edition
- **Java ME**, is designed for mobile phones (especially feature phones) and set-top boxes.
- Java ME was designed by Sun Microsystems

2) Java Platform, Enterprise Edition (J2EE):

- J2EE stands for **Java 2 Platform, Enterprise Edition**. J2EE is the standard platform for developing applications in the enterprise and is designed for enterprise applications that run on servers.

3) Java Platform, Standard Edition (J2SE):

- It has concepts for developing software for Desktop based (standalone) CUI (command user interface) and GUI (graphical user interface) applications, applets.

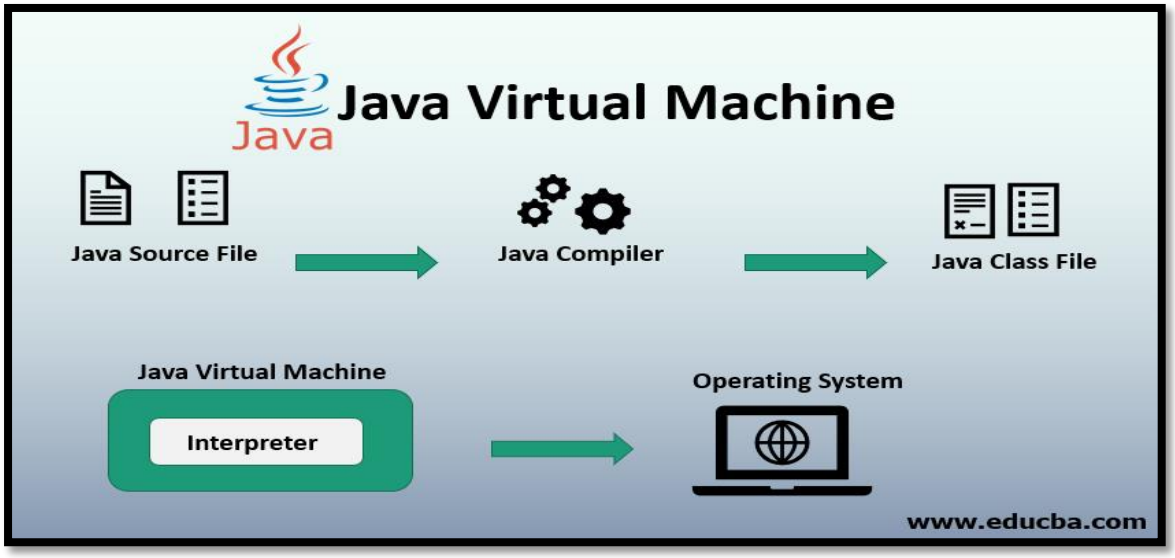
MCQ		
Sr No.	**Question**	**Answer**
1	**J2ME stands for ……**	**Java 2 platform Micro Edition**
2	**J2EE stands for ……**	**Java 2 platform Enterprise Edition**
3	**J2SE stands for ……**	**Java 2 Platform Standard Edition**

Q-3 Write a short note on JVM, JRE and JDK

Ans:

JVM:

- JVM stands for **Java Virtual Machine.**
- JVM loads, verifies and runs the java byte code.
- JVM is the interpreter or the core of the java programing language because it runs java programming
- JVM helps in executing java bytecode.'
- Java compiler compiles source code into class file(byte code). Byte code is not machine code. So JVM (Java Interpreter) converts byte code into machine code and execute



JRE:

- The Java Runtime Environment (JRE), also known as Java Runtime, is part of the Java Development Kit (JDK), a set of programming tools for developing Java applications.
- It is a software that java programs require to run correctly.
- JRE consists of JVM, core classes and supporting files.

JDK:

- JDK stands for **Java Development Kit**.
- It is a collection of tools/components which are used for developing and running the java program.

Components:-

1) **Applet viewer** :- it used to view the java applets without using the browser.
2) **Javac** :- the java compiler translate source code to byte code.
3) **Java** :- java interpreter runs the applets and application by reading the byte code.
4) **Javadoc**: - it creates html format documentation from java source code.
5) **Javah** :- it produce header file.
6) **Javap** :- it enables to convert byte code into program description.
7) **Jdb** :- it is java debugger used for find errors from the programs.

MCQ		
Sr No.	**Question**	**Answer**
1	**JVM stands for ……**	**Java Virtual Machine**
2	**JRE stands for ……**	**Java Runtime Environment**

3	JDK stands for ……	Java Development Kit
4	What is javac?	Compiler of java
5	Which component of jdk produce header file?	javah
6	Which component of jdk is used to execute or run the application?	java
7	Which component of jdk is used to find the errors from the program	jdb

Q-4 Write a short note on compiling and executing basic java program.

Ans:

```
class hello
{
public static void main(String args[])
{
        System.out.println("Hello");
        }
}
```

Steps:

Step 1: Save the program with .java extension (Note: Name of the program must be same as the name of the class)

Step 2: Open the command prompt and compile your program

javac hello.java (where hello.java is the name of your file)

Step 3: Execute/run the program

java hello

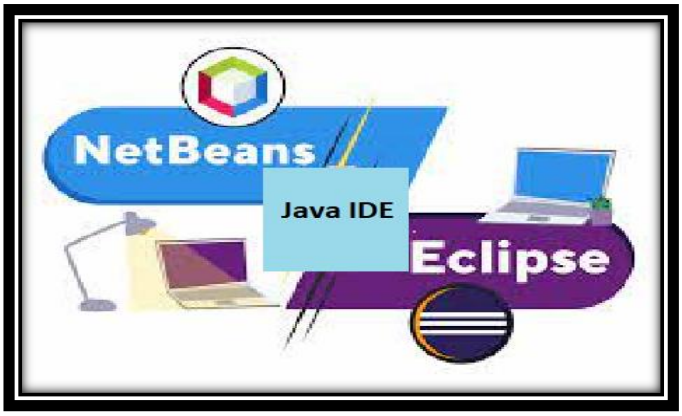
Output: hello

MCQ		
Sr No.	**Question**	**Answer**
1	**The program of java should have the same name as …….**	**Program name**
2	**Java program should be saved with …….extension**	**.java**

Q-5 Write a short note on java IDE

Ans:

IDE stands for Integrated Development Environment



Netbeans:

- This IDE for java can be used for developing applications in other languages like PHP, C, C++
- Netbeans uses javac compiler present in JDK installed on client machine for compiling of java source code.
- Developing enterprise applications in java is easy with netbeans as compared to eclipse
- It has built-in drivers for mysql and oracle
- Netbeans takes more time to load IDE compared with eclipse.
- Netbeans can be more beginner friendly than eclipse. It has simple user interface

Eclipse:

- It is open source java development platform for java, C, C++, PHP, Python, javascript etc
- Eclipse uses other compiler than javac compiler, thereby allows running broken code with unresolved errors and display more warning and errors as compared to javac compiler
- To develop a enterprise applications, Eclipse IDE requires to install enterprise edition tools.
- It has different windows for designing or developing various applications,

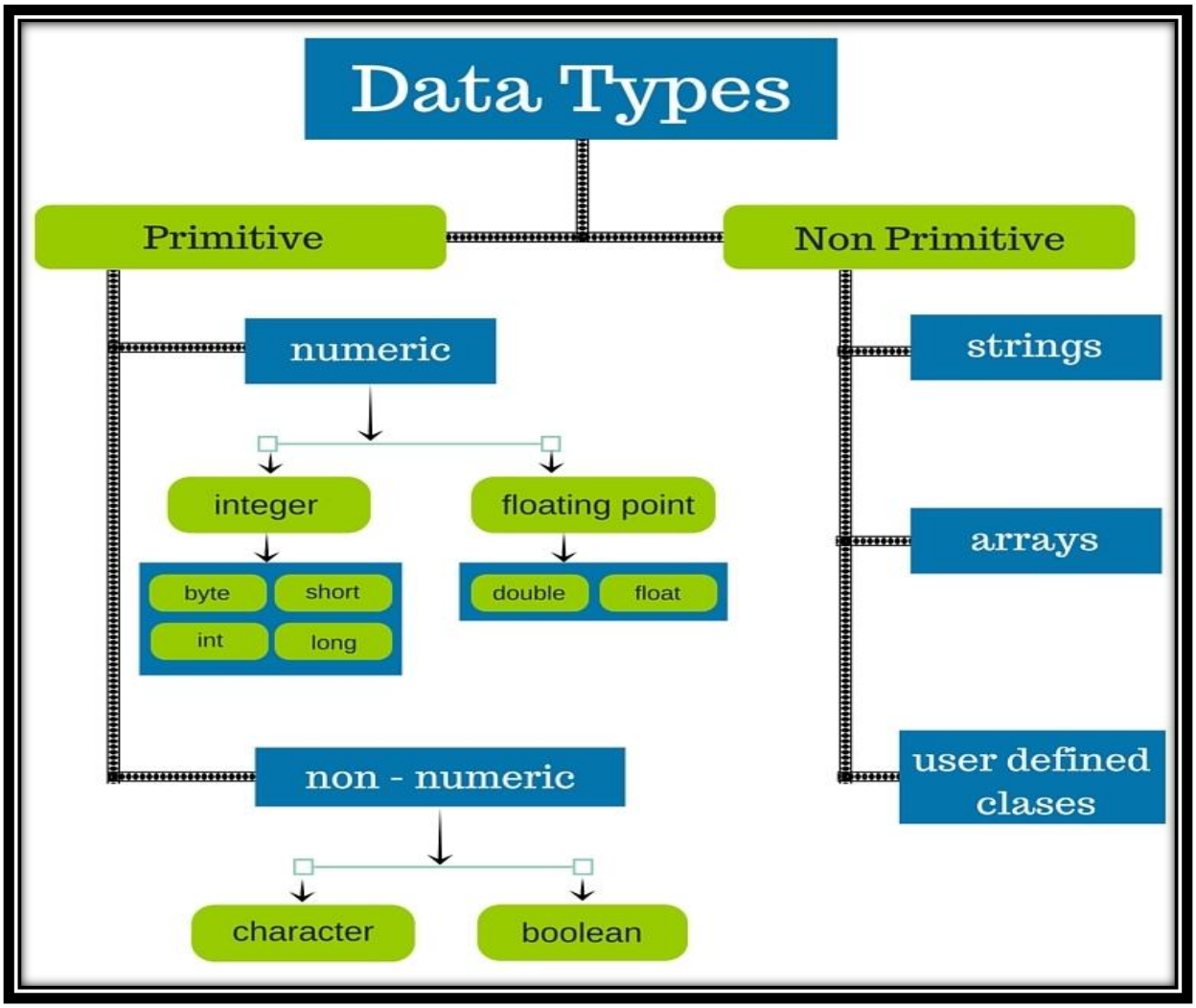
- It has JDBC driver support but needs to configure them before working with database
- Eclipse IDE takes less time opening itself compared to Netbeans IDE.
- Eclipse is not more beginner friendly compared to Netbeans.

MCQ		
Sr No.	**Question**	**Answer**
1	**IDE stands for**	**Integrated Development Environment**
2	**Which editors or platform are provided by java?**	**NetBeans and Eclipse**

Q-6 Write a short note on data types in java

Ans:

- Data type means the type of data.



Integer

- Java provides four integer types: byte, short, int, long.
- All of these are signed, positive and negative values.

Type	Size/bits	Range
Byte	8	-128 to 127
Short	16	-32,768 to 32,767
Int	32	-2,147,483,648 to 2,147,483,647

Long	64	-9,223,372,036,854,775,808 9,223,372,036,854,775,807

Floating-Point Types

- Floating-Point numbers, also known as real numbers, are used when evaluating expressions that require fractional precision.
- For example, calculations such as square root, or transcendental such as sine and cosine, result in a value whose precision requires a floating-point type.
- **There are two kinds of floating-point types, float and double, which represent single and double-precision numbers, respectively.**

Type	Size/bytes	Range
Float	32	1.4e – 045 to 3.4e + 038
Double	64	4.9e - 324 to 1.8e + 308

Character

- The data type used to store characters is **char**.

Type	Size/bytes	Range
Char	16	0 to 65,536

Boolean:

- Java has primitive type, called boolean, for logical values.
- It can have only one of two possible values, true or false.

Type	Size/bytes	Range
Boolean	1	True/False , Yes/No , 0/1

MCQ		
Sr No.	**Question**	**Answer**
1	**short requires …….bits**	**16 bits(2 byte)**
2	**int requires ……bits**	**32 bits (4 bytes)**

3	**Boolean data type can only the value either …….or …….**	**true, false**
4	**char requires …….**	**16 bits (2 bytes)**

Q-7 what is ADT?

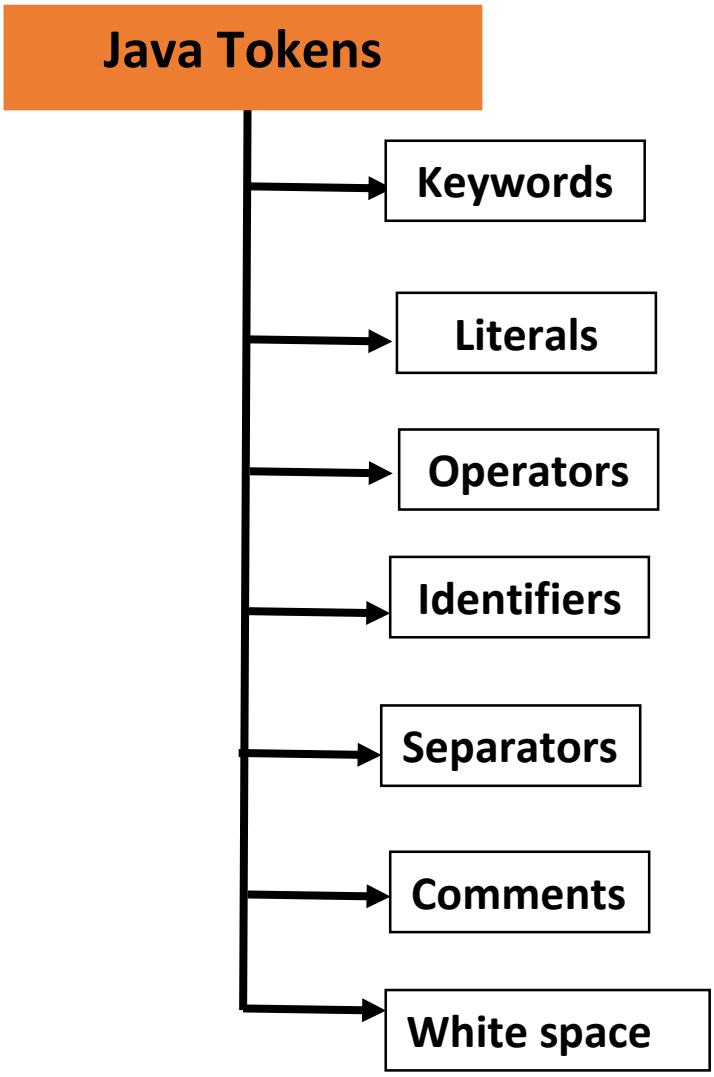
Ans:

- **ADT stands for Abstract Data type**
- **It is a model that defines operations and behaviours for a data type without specifying how these are implemented or how data is stored.**
- The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations.
- It is called “abstract” because it provides an implementation-independent view.
- Examples of ADT are: List, Stack, Queue
- ADT are a way of encapsulating data and operations on that data into a single unit.

Q-8 Write a short note on java tokens

Ans:

- **Token is the smallest individual unit of a program.**
- Tokens are the various Java program elements which are identified by the compiler
- Tokens supported in Java include keywords, operators, identifiers, literals, separators, comments, white space



1) Keywords: Keywords are the reserved or predefined words whose meaning cannot be changed. Examples of keywords are: assert, strictfp,enum etc.
2) Literals: This are the constant values which can not be modified.
    a. Integer Literal : 20, 30
    b. Floating point literal: 2.5, 6.1
    c. String literal : “hello”
3) Operators: The symbol which is used to perform various mathematical operations. Examples: +, -, *, /
4) Identifiers: Identifiers means naming of variable, function or class
5) Separators: It separates different part of programs such as commas, semicolon, parenthesis( ), braces { }, brackets [ ]

6) Comments: Comments are the information or code that will not be displayed in the output. There are 2 types of comments:
    a. Line Oriented (//): It is used to give single line comments
       Example: // this is single line comment
    b. Block Oriented (/*…….*/): It is used to give multi line comments
       Example: /* This is multiline comment
                statement1
                statement */
7) Whitespace: Whitespace characters are the characters that are used to give space, tab or new line

MCQ		
Sr No.	**Question**	**Answer**
1	**The smallest individual unit of program is known as ….**	**Token**
2	**…….are the reserved words**	**keyword**
3	**Literals means the ……..**	**value**

Q-9 Write a short note on Operators in java

Ans:

- An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.

- Following are the types of operators in java:
    1) Arithmetic
    2) Relational
    3) Logical
    4) Assignment
    5) Conditional
    6) Instance Of
    7) Sizeof
    8) Bitwise
    9) Increment/Decrement

1) Arithmetic Operators:

Operators	Result
+	Addition of two numbers
-	Subtraction of two numbers
*	Multiplication of two numbers
/	Division of two numbers
%	(Modulus Operator)Divides two numbers and returns the remainder

Example:

```
import java.util.*;
class arithmetic
{
public static void main(String args[])
{
int a,b;
Scanner sc=new Scanner(System.in);
System.out.println("Enter a");
a=sc.nextInt();
System.out.println("Enter b");
b=sc.nextInt();
System.out.println("Addition=" +(a+b));
System.out.println("Sub is" +(a-b));
System.out.println("Mul is" +(a*b));
System.out.println("Div is"+(a/b));
System.out.println("Modulas is" +(a%b));
}
}
```

2) Relational Operators:

Operators	Operations
==	Equal to
!=	Not Equal to
>	Greater than
>=	Greater than equal to
<	Less than
<=	Less than equal to

Example:

```
import java.util.*;
class arithmetic
{
public static void main(String args[])
{
int a,b;
Scanner sc=new Scanner(System.in);
System.out.println("Enter a");
a=sc.nextInt();
System.out.println("Enter b");
b=sc.nextInt();
System.out.println("> than" +(a>b));
System.out.println("< than" +(a<b));
System.out.println(">= is" +(a>=b));
System.out.println("<="+(a<=b));
System.out.println("== is" +(a==b));
System.out.println("!= is" +(a!=b));
}
}
```

3) Logical Operators:

Logical Operator	Java Operator
AND	&&
OR	||
NOT	!

Example:

```
import java.util.*;
class arithmetic
{
public static void main(String args[])
{
int a,b,c;
Scanner sc=new Scanner(System.in);
System.out.println("Enter a");
a=sc.nextInt();
System.out.println("Enter b");
b=sc.nextInt();
System.out.println("Enter c");
c=sc.nextInt();
if(a>b && a>c)
        System.out.println("a is max");
else if(b>c && b>a)
        System.out.println("b is max");
else
        System.out.println("c is max");
}
}
```

4) Assignment Operators:

- **Assignment operator is indicated by =**

Case 1: (Copy the value to the variable)

a=5

In the above case, 5 is assigned the value to a

Case 2: (Copy the variable value to another variable)

a=4,b=5

a=b

In the above case, the value of b is assigned to a

Case 3: (Copy the expression to the variable)

a=5,b=4

c=a+b

In the above case, the value of a+b is assigned to c.

Note: Assignment operator is also known as shorthand operator because it represent shorthand ways to represent the variable.

Example:

a=a+5	**a+=5**
a=a-4	**a-=4**
a=a*2	**a*=2**

5) Conditional Operators:

- **Conditional operator is indicated by ?**
- **It is also known as ternary operator because it requires 3 parts:**

Syntax:

Condition? True part: false part

Example:

(a>b) ? "a is max" : "b is max"

6) Instance of Operator:

- **It is used to test whether the object is an instance of the specified type**
- **It is also known as type comparison operator because it compares instance with the class type.**

Syntax:

objectname=classname

Example:

test t1=new test();

The above example checks whether the t1 is object of the class test or not

7) size of Operator:

- **It is used to find the size of the variable or the type.**

Syntax:

Datatype (SIZE)

Example:

Float(SIZE);

8) Bitwise Operator:

- **Bitwise operator operates on the bits(0 and 1)**
- **Following are the types of bitwise operators.**
  - **Bitwise AND (&)**
  - **Bitwise OR(|)**
  - **Bitwise Exclusive OR(^)**
  - **Bitwise Complement (~)**
  - **Bitwise Shift operators**

Truth Table of Bitwise AND, Bitwise OR, Bitwise Exclusive OR,

a	b	a&b	a|b	a^b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

1) Bitwise AND (&):

- **It is the binary operator which returns true only if both the bits are 1 or else it returns 0.**

Example:

int x=8, y=9

The binary value of 8 is 1000 and binary value of 9 is 1001.

So x & y will give 8 (1000)

2) Bitwise OR (|):

- **It is the binary operator which returns true if any of both the bits are 1 or else it returns 0.**

Example:

int x=8, y=9

The binary value of 8 is 1000 and binary value of 9 is 1001.

So x | y will give 9 (1001)

x	~x
0	1
1	0

3) Bitwise Exclusive OR (^):

- **It is the binary operator and it returns 0 if both the bits are 1**

Example:

int x=8, y=9

The binary value of 8 is 1000 and binary value of 9 is 1001.

So x ^ y will give 1 (0001)

4) Bitwise Complement (~):

- **It is the unary operator and it returns inverse or opposite of the bit**

5) Bitwise Shift operators

- **Shift operators are used to shift the bits of a number left or right, thereby multiplying or dividing number by 2**

Example: a=256, b=4

a>>4 gives a/2^4 $256/2^4 = 256/16 = 16$

a<<4 gives a*2^4 $256*2^4 = 256 * 16 = 4096$

6) Increment/Decrement Operator:

- **This operator is used to increment or decrement the value by 1.**
- **There are 2 types of increment and decrement:**
  - **pre-increment (++a)**
  - **Post-increment(a++)**
  - **pre-decrement(--a)**
  - **post-decrement(a--)**

Pre-increment	Post-Increment
1) In this, value is increment first and then it is assigned to the variable	**1) In this, value is first assigned to the variable and then it is incremented.**
2) ++a	**2) a++**

Example:

```
class inc
{
public static void main(String args[])
{
        int i=0,j=0;
        System.out.println(i++);  //post increment
        System.out.println(++j); // pre-increment
}
```

2) In this, value is decremented first and then it is assigned to the variable	**3) In this, value is first assigned to the variable and then it is decremented.**
4) --a	**2) a--**

Example:

```
class dec
{
public static void main(String args[])
{
        int i=0,j=0;
        System.out.println(i- -);  //post increment
        System.out.println(- -j); // pre-increment
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**Assignment Operator is also known as …..**	**Shorthand Operator**
2	**Conditional Operator is also known as ….**	**Ternary Operator**
3	**Which operator is used to test whether the object is an instance of the specified type?**	**instance of**
4	**Relational operator is also known as ……..operator**	**Comparison**

Q-10 Write a short note on Decision Control structures

Ans:

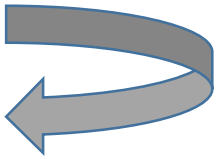
- **Following are the types of decision control structures:**
  - **if**
  - **if-else**
  - **Nested if**
  - **else-if ladder**
  - **switch case**

1) if:

Description: This decision control structure only deals with the true part of the condition and it works only with one condition.

Syntax:

```
if(condition)
{
        statements;
}
```



True part

Example:

```
import java.util.*;
class dc
{
public static void main(String args[])
{
        int a,b;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter a");
        a=sc.nextInt();
        System.out.println("Enter b");
        b=sc.nextInt();
        if(a==b)
                System.out.println("Equal");
}
}
```

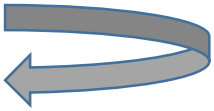
2) if-else:

Description: This decision control structure deals with the true and false part of the condition and it works only with one condition.

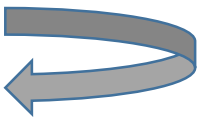
If the condition is true then statements inside if are executed and if the condition is false then statements inside false are executed.

Syntax:

```
if(condition)
{
        statement1;
}
else
{
        statement 2;
}
```



True part



False part

Example:

```
import java.util.*;
class dc
{
public static void main(String args[])
{
        int a,b;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter a");
        a=sc.nextInt();
        System.out.println("Enter b");
        b=sc.nextInt();
        if(a==b)
                System.out.println("Equal");
        else
                System.out,println("Not Equal");
}
}
```

3) Nested if:

Description:

- **Nested if means one if inside another if. It is used when you have more than 1 condition.**
- **In Nested if, there are two if-else, one is inner if-else and other is outer if-else**

Syntax:

```
if(condition1)
{
     if(condition2)
     {
          Statement1;
     }
     else
     {
          Statement2;
     }
 }
else
{
     Statement 3;
}
```

Outerif-else

Inner if-else

Example:

```
import java.util.*;
class dc
{
public static void main(String args[])
{
        int a,b,c;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter a");
        a=sc.nextInt();
        System.out.println("Enter b");
        b=sc.nextInt();
        System.out.println("Enter c");
        c=sc.nextInt();

        if(a>b)
        {
                if(a>c)
                System.out.println("a is max");
        else
                System.out,println("c is max");
        }
        else
        {
                if(b>c)
                        System.out.println("b is max");
                else
                        System.out.println("c is max");
        }
}
}
```

4) else-if ladder:

Description:

- **If-elseif-elseif-elseif….else is known as else-if ladder.**
- **It is used when you have more than 2 conditions.**
- **In else-if ladder, when all the conditions are false then else part is executed**

Syntax:

```
if(condition1)
{
        statement1;
}
elseif(condition2)
{
        statement2;
}
elseif(condition3)
{
        statement3;
}
….
….
else
{
        statement4;
}
```

Example:

```
import java.util.*;
class dc
{
public static void main(String args[])
{
        int a,b,c;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter a");
        a=sc.nextInt();
        System.out.println("Enter b");
        b=sc.nextInt();
        System.out.println("Enter c");
        c=sc.nextInt();

        if(a>b && a>c)
                System.out.println("a is max");
        else if(b>c && b>a)
                System.out.println("b is max");
        else
                System.out.println("c is max");

}
}
```

5) switch case:

Description:

- switch case is used with multiple options.
- In switch case, the value of the variable is passed which is compared with the different cases, the case which matches the value is executed.
- switch case have 4 keywords: switch, case, default, break
- break keyword is used to exit from the particular case
- When no case is executed, at that time default case is executed

Syntax:

```
switch(variable)
{
        case 1: statement1;
                break;
        case 2: statement2;
                break;
        case 3: statement3;
                break;
        case4: statement4;
                break;
        default: statement5;
                break;
}
```

Example:

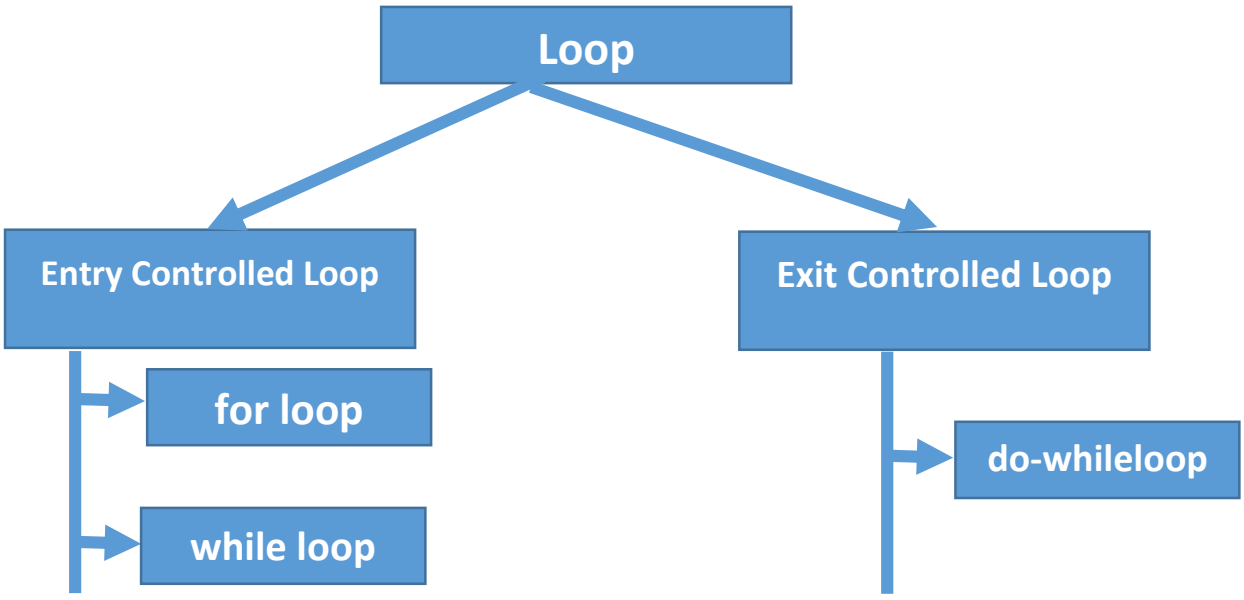
```
import java.util.*;
class dc
{
public static void main(String args[])
{
        int a,b,ch;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter a");
        a=sc.nextInt();
        System.out.println("Enter b");
        b=sc.nextInt();
        System.out.println("Enter 1.add 2.sub 3.mul 4.div");
        ch=sc.nextInt();
        switch(ch)
        {
                case 1: System.out.println(a+b);
                        break;

                case 2: System.out.println(a-b);
                        break;

                case 3: System.out.println(a*b);
                        break;

                case 4: System.out.println(a/b);
                        break;

                default: System.out.println("Invalid");
                        break;
        }

}
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**Which decision control structure is used when we have multiple options?**	**Switch case**
2	**In switch case, there are …….keywords**	**4**
3	**break in switch case is used to …….**	**Exit from the particular case**
4	**if inside another if is known as ……**	**Nested if**
5	**Which decision control sturucture only deals with true part**	**if**

Q-11 what is loop? List out types of looping structures and explain in detail

Ans:

- **When same task is to be performed multiple times, then in that case loop is used.**
- **Following are the types of loop**



1) Entry Controlled Loop:

- ❖ **When the condition is checked in the starting of the loop, it is known as entry controlled loop.**
- ❖ **for loop and while loop are known as entry controlled loop.**

*** for loop:**

Syntax:

for(initialization ;condition ;increment/decrement)

{
Statements;

}

Example:

```
class loop
{
public static void main(String args[])
{
        int i;
        for(i=1;i<=5;i++)
        {
                System.out.println(i);
        }
}
```

*** while loop:**

Syntax:

Initialization;
while(condition)

{
Statements;
Increment/decrement;

}

Example:

```
class loop
{
public static void main(String args[])
{
        int i=1;
        while(i<=5)
        {
                System.out.println(i);
                i++;
        }
}
```

2) Exit Controlled Loop:

- **When the condition is checked in the end of the loop, it is known as exit controlled loop.**
- **do-while loop is example of exit controlled loop**
- **In do-while loop, statements are executed atleast once without checking the condition.**

Syntax:

```
Initialization;
do
{
        Statements;
        Increment/decrement;
} while (condition);
```

Example:

```
class loop
{
public static void main(String args[])
{
        int i=1;
        do
        {
                System.out.println(i);
                i++;
        }while(i<=5);
}
```

*** Difference between while and do-while loop

while loop	do-while loop
It is known as entry controlled loop	It is known as exit controlled loop
If the condition is true then only statements are executed	The statements are executed atleast once and after that the condition is checked.
While loop does not have semicolon	do-while loop have terminating semicolon

MCQ		
Sr No.	Question	Answer
1	Which 3 things all the loops have?	initialization, condition, increment/decrement
2	for loop is known as …….	Entry controlled loop
3	do-while loop is known as ……	Exit controlled loop
4	Which loop have terminating semi-colon at the end?	do-while
5	In which loop, without checking the condition the statements are executed at least once?	do-while

Q-12 Write a short note on jumping statements in java

Ans:

- **Jumping statements transfers the control from one location to another location.**
- **Following are the types of jumping statements:**
  - **break**
  - **continue**

1) break statement:

- **This statement is used to exit immediately from the loop or program**
- **When the break statement is encountered in the program, the control directly moves to the end of the program.**

Syntax:

```
for(; ;)
{
        if(condition)
        break;
}
```

Example:

```
class jm
{
public static void main(String args[])
{
        int i;
        for(i=1;i<=5;i++)
        {
                if(i==3)
                break;
                System.out.println(i);
        }
}
```

2) continue statement:

- **This statement is used to continue back to the re-evaluation of the condition.**
- **Using continue statement, certain statements are bypassed.**

Syntax:

```
for(; ;)
{
        if(condition)
        continue;
}
```

Example:

```
class jm
{
public static void main(String args[])
{
        int i;
        for(i=1;i<=5;i++)
        {
                if(i==3)
                continue;
                System.out.println(i);
        }
}
```

***** Difference between break and continue statement**

break	continue
it is used to exit immediately from the loop	It transfers the cursor back to the re-evaluation of the condition
Using break, certain statements cannot be bypassed	Using continue statement, certain statements can be bypassed.

3) return statement:

- **This statement helps you to transfer the control from one method to the method that is called**
- **Since, the control jumps from one part of program to another, return is also a jumping statement**

Example:

```
class jm
{
        public static int add(int a,int b)
        {
                int sum=a+b;
                return sum;
        }
public static void main(String args[])
{
        int result= add(2,3);
        System.out.println(result);
}
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**Which are the jumping statements in java?**	**break,continue**
2	**In which type of jumping statement, certain statements are bypassed**	**continue**
3	**Which jumping statement is used to immediately exit from the loop or program?**	**break**

Q-13 Write a short note on typecasting

Ans:

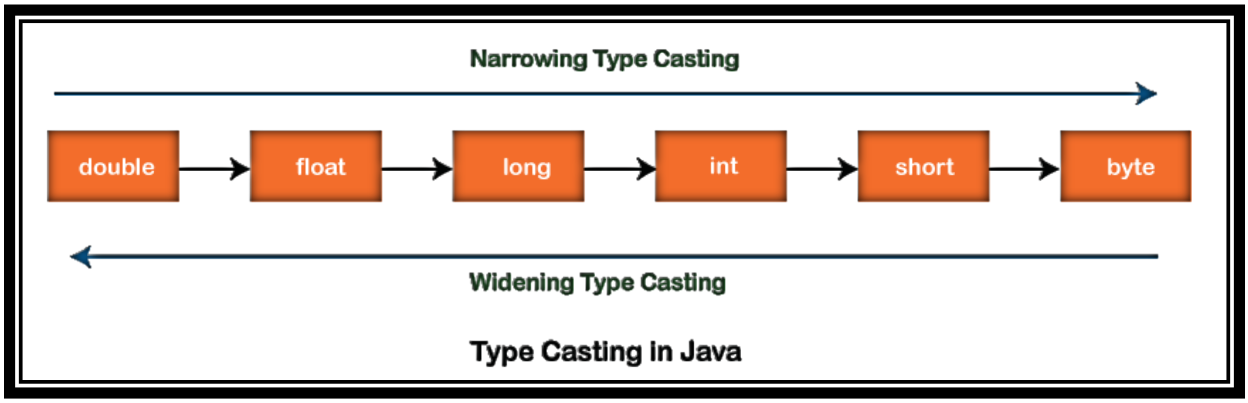
- **Typecasting means converting one data type into another datatype.**
- **Type casting is required when there are different data types in same statement.**
- **There are 2 types of type casting in java.**

1) Widening type casting/Casting down

- **It is also known as implicit/ automatic type casting**
- **In widening type casting, smaller data type is automatically converted in to large data type.**
- **Converting a lower data type into a higher one is** called **widening**
- **No data loss occurs in widening type casting**

Example:

```
class type
{
        public static void main(String args[])
        {
                int x=10;
                float  y=x;
                System.out.println(y);
        }
}
```



1) Narrowing type casting/Casting up

- **It is also known as Explicit/user type casting**
- **In narrowing type casting, large data type is manually converted into small data type**
- Converting a large data type into a small one is called **widening**
- In this type of typecasting, user manually coverts large data type into small data type using typecast / cast operator

- The main disadvantage of narrowing type casting is that: Data loss occurs as we convert large data type into smaller data type.

Example:

```
class exp1
{
        public static void main(String args[])
        {
                double x=2.5;
                int y=(int) x;
                System.out.println(y);
        }
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**Converting one data type into another data type is known as …..**	**Type casting**
2	**In which type of typecasting, smaller data type is automatically type casted to larger data type?**	**Widening**
3	**Narrowing data type is known as …….**	**Explicit type casting**

Q-14 Write a short note on Arrays

Ans:

- Array is the collection of elements that have same data type. All the elements of array share same array name
- The main concept of array is index

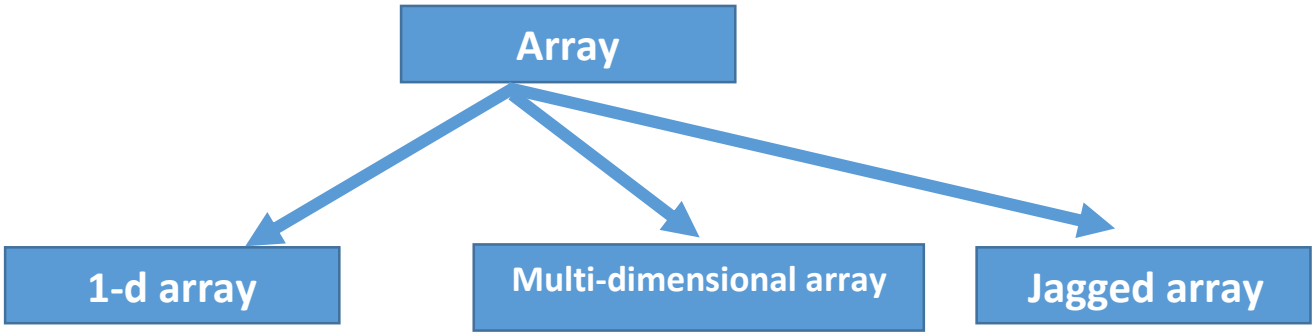
Advantages:

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.

- **Random access:** We can get any data located at an index position.

Disadvantages:

Size Limit: We can store only the fixed size of elements in the array. It doesn't grow its size at runtime.



1) 1-d array (One Dimensional Array):

- **The array having only one dimension (row size) is known as 1-d array.**

Syntax:

Declaration of array:

datatype[] arrayname OR
datatype arrayname[] OR

Initialization of array:

datatype arrayname[]=new datatype[size];

Example:

```
import java.util.*;
class arr
{
	public static void main(String args[])
	{
		int i;
		int a[]=new int[5];
		Scanner sc=new Scanner(System.in);
		for(i=0;i<5;i++)
		{
			System.out.println("Enter array elements");
			a[i]=sc.nextInt();
		}
		for(i=0;i<5;i++)
		{
			System.out.println("Array ele" +a[i]);
		}
	}
}
```

2) 2-d array/Multi-Dimensional array/rectangular array

- **In this array, the data is stored in row and column based index (also known as matrix form)**
- **Total number of elements in 2-d array is row size * column size**

Syntax:

Declaration of array:

datatype[][] arrayname OR
datatype arrayname[][]

Initialization of array:

datatype arrayname[][]=new datatype[rowsize][columnsize];

Example:

```
import java.util.*;
class arr
{
	public static void main(String args[])
	{
		int i,j;
		int a[][]=new int[2][3];
		Scanner sc=new Scanner(System.in);
		for(i=0;i<2;i++)
		{
			for(j=0;j<3;j++)
			{
			System.out.println("Enter array elements");
			a[i][j]=sc.nextInt();
			}

		}
		for(i=0;i<2;i++)
		{
			for(j=0;j<3;j++)
			{
			System.out.println("Array ele" +a[i][j]);
			}
		}
	}
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**The collection of elements having same data type and same array name is known as ……**	**Array**
2	**Which type of array have both row and column size?**	**Multi-dimensional array**
3	**In which type of array, data is stored in matrix form?**	**Multi-dimensional array**

3) Jagged array:

- **Array of array is known as jagged array**
- **The main feature of jagged array is that row size of jagged array is fixed but the size of column will vary according to row**
- **In jagged array, each row can have different columns**

Example:

```
int a[ ][ ]=new int [2][ ];
a[0]=new int [3];
a[1]=new int[2];
```

```
import java.util.*;
class arr
{
	public static void main(String args[])
	{
		int i,j;
		int a[][]=new int[2][ ];
		Scanner sc=new Scanner(System.in);
		for(i=0;i<2;i++)
		{
			for(j=0;j<a[i].length;j++)
			{
			System.out.println("Enter array elements");
			a[i][j]=sc.nextInt();
			}
		}
		for(i=0;i<2;i++)
		{
			for(j=0;j<a[i].length;j++)
			{
			System.out.println("Array ele" +a[i][j]);
			}
		}
	}
}
```

Q-14 Write a short note on Command Line arguments

Ans:

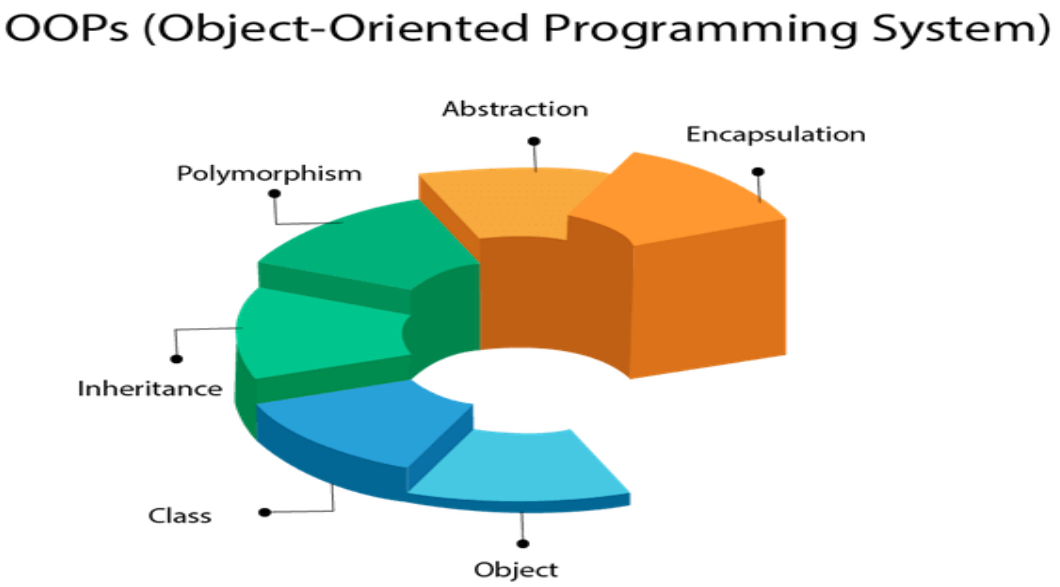
- **The java command-line argument is an argument i.e. passed at the time of running the java program.**
- **The arguments passed from the console can be received in the java program and it can be used as an input.**

Example 1:

```
class cmd1
{
        public static void main(String args[])
        {
                System.out.println(args[0]);
        }
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**The arguments passed at runtime of program is known as……**	**Command line arguments**
2	**Passing of arguments in command line arguments starts with ……**	**args[0]**

Unit 1 –Class Fundas (Part 2)

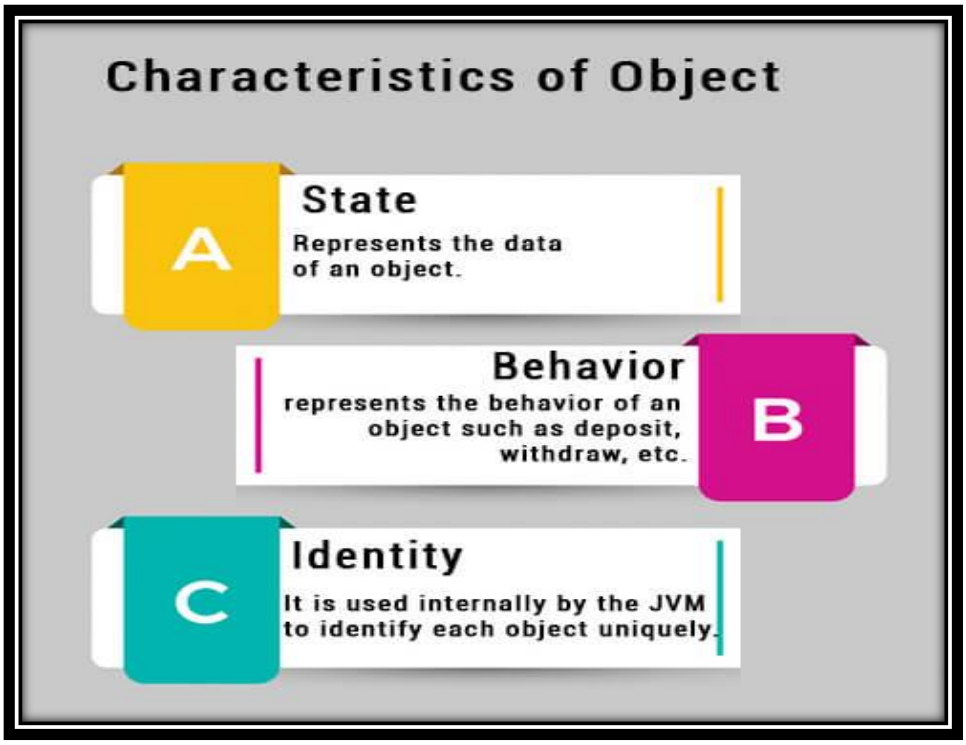


Q-1 Explain class and object

Ans:

- A class is a **template from which objects are created**. That is objects are instance of a class
- **When you create a class, you are creating a new data-type**. You can use this type to declare objects of that type.
- An entity that has state and behaviour is known as an object e.g., chair, bike, marker, pen, table, car, etc.
- **An object is an instance of a class.**
- Class **defines structure and behaviour (data & code)** that will be shared by a set of objects
- Each object contains its own copy of each variable defined by the class
- A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

An object has three characteristics:



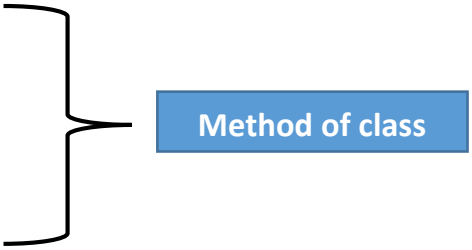
- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

- For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

.

Syntax:

```
class ClassName
{
        type instance variable1;
        type instance variable2;

                type methodname1 (parameter list)
                {
                        body of method;
                }
                type methodname2 (parameter list)
                {
                        body of method;
                }

}
```



Example:

```
class student
{
        int a=10;
        int b=20;
        public void add( )
        {
                System.out.println(a+b);
        }
        public static void main(String args[])
        {
                student s=new student();
                s.add();
        }
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**Which are the features or characteristics of object?**	**state,behavior,identity**
2	**A ……. is a template from which objects are created**	**class**
3	**……..and ……..are the main concept of OOP**	**class, object**

Q-2 Explain Encapsulation

Ans:

- **Encapsulation is one of the fundamental concept of OOP**
- Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.
- In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.

To achieve encapsulation in Java:

- Declare the variables of a class as private.
- Provide public setter and getter methods to modify and view the variables values.
- **To achieve encapsulation in Java :**

Advantages of Encapsulation:

- The fields of a class can be made read-only or write-only.
- A class can have total control over what is stored in its fields.

Example:

```
class test
{
        private int roll;
        public void get(int r)
        {
                roll=r;
        }
        public void set( )
        {
                System.out.println(roll);
        }
}

class enc
{
        public static void main(String args[])
        {
                test t=new test();
                t.get(1);
                t.set();
        }
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**Wrapping of variables and methods in single unit is known as……**	**Encapsulation**
2	**The main concept of encapsulation is …….**	**Data Hiding**
3	**To achieve encapsulation, the variables have to be declared …….**	**Private**
4	**To access the private variable of the class which methods are used?**	**get() and set()**

Q-3 Write a short note on Inheritance

Ans:

- **Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviours of a parent object. It is an important part of OOPs (Object Oriented programming system).

Terms used in inheritance:

1) **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
2) **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
3) **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class

Advantage of Inheritance:

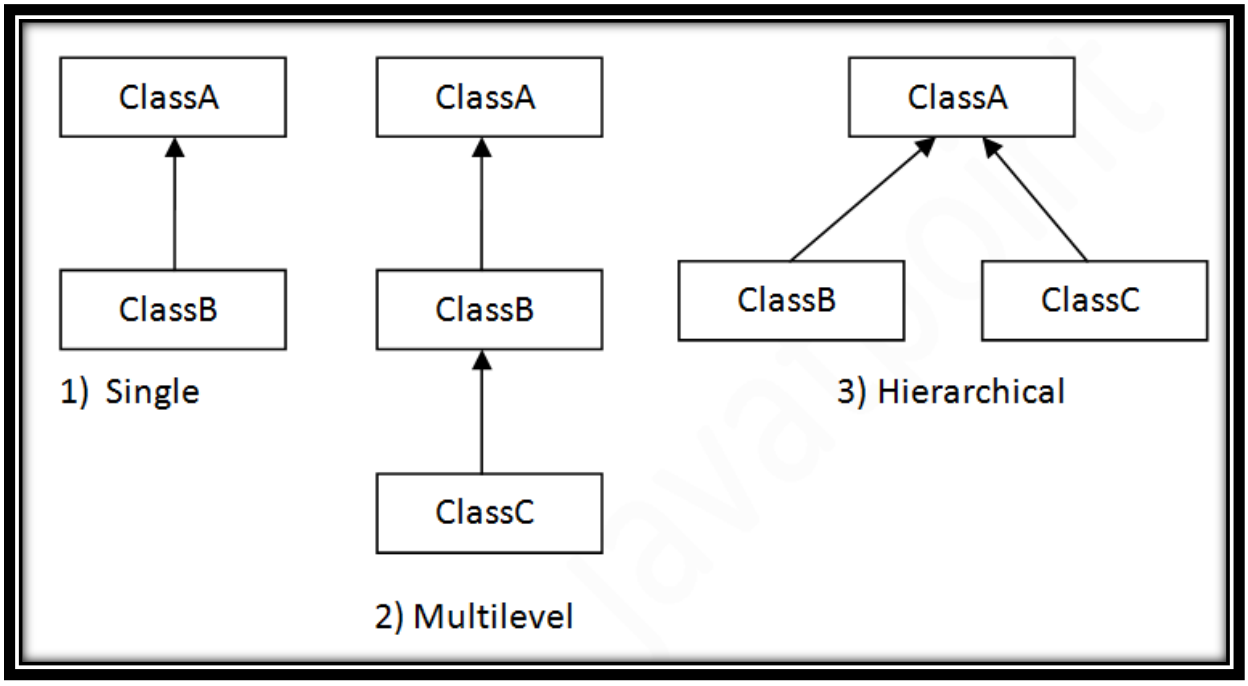
Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

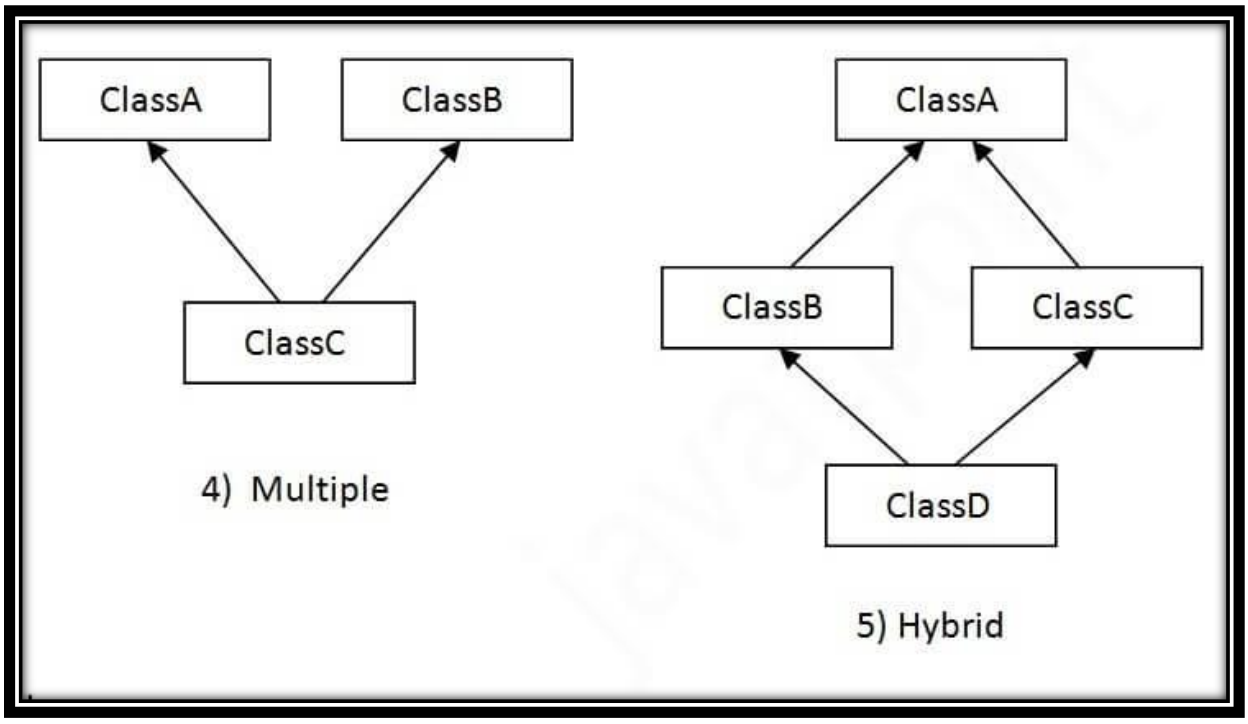
Syntax:

```
class subclass extends superclass
{
        //methods and fields
}
```

The **extends keyword** indicates that you are making a new class that derives from an existing class.

Types of Inheritance:





Note: ***Multiple inheritance is not supported in Java through class.***

1) Single Inheritance :

- **When a class inherits another class, it is known as a *single inheritance*.**

Example:

In the below example, Dog is the subclass and Animal is the Parent Class. So Dog inherits the method of the Animal Class and also it have its own method

```
class Animal
{
	void eat()
	{
		System.out.println("Eating");
	}
}

class Dog extends Animal
{
	void bark()
	{
		System.out.println("Barking");
	}
}

class single
{
	public static void main(String args[])
	{
		Dog d=new Dog();
		d.bark();
		d.eat();
	}
}
```

2) Multilevel Inheritance :

- When there is a chain of inheritance, it is known as *multilevel inheritance*.
- In Multilevel inheritance, one class inherits from another class, another class inherits from next class and so on.

```
class Animal
{
        void eat()
        {
                System.out.println("Eating");
        }
}

class Dog extends Animal
{
        void bark()
        {
                System.out.println("Barking");
        }
}
class BabyDog extends Dog
{
        void sleep()
        {
                System.out.println(“Sleeping”);
        }
}

class single
{
        public static void main(String args[])
        {
                BabyDog bd=new BabyDog();
                d.bark();
                d.eat();
                d.sleep();
        }
}
```

3) Hierarchical Inheritance :

- When two or more classes inherits a single class, it is known as *hierarchical inheritance*.

```
class Animal
{
	void eat()
	{
		System.out.println("Eating");
	}
}

class Dog extends Animal
{
	void bark()
	{
		System.out.println("Barking");
	}
}
class cat extends Animal
{
	void sleep()
	{
		System.out.println(“Sleeping”);
	}
}

class single
{
	public static void main(String args[])
	{
		cat c=new cat();
		c.eat();
		c.sleep();
		Dog d=new Dog();
		d.bark();
		d.eat();

	}
`
```

4) Multiple Inheritance :

- Multiple inheritance is not supported in java by class. The reason is suppose A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Example:

```
class A
{
        void msg()
        {
                System.out.println("Hello");
        }
}
class B
{
        void msg()
        {
                System.out.println("Hi");
        }
}
class C extends A,B
{
        public static void main(String args[])
        {
                C c1=new C();
                c1.msg();//Which msg() method will be called?
        }
}
```

Will generate compile time error

5) **Hybrid Inheritance :**

- ❖ Hybrid inheritance is the combination of more than 1 type of inheritance.

Example:

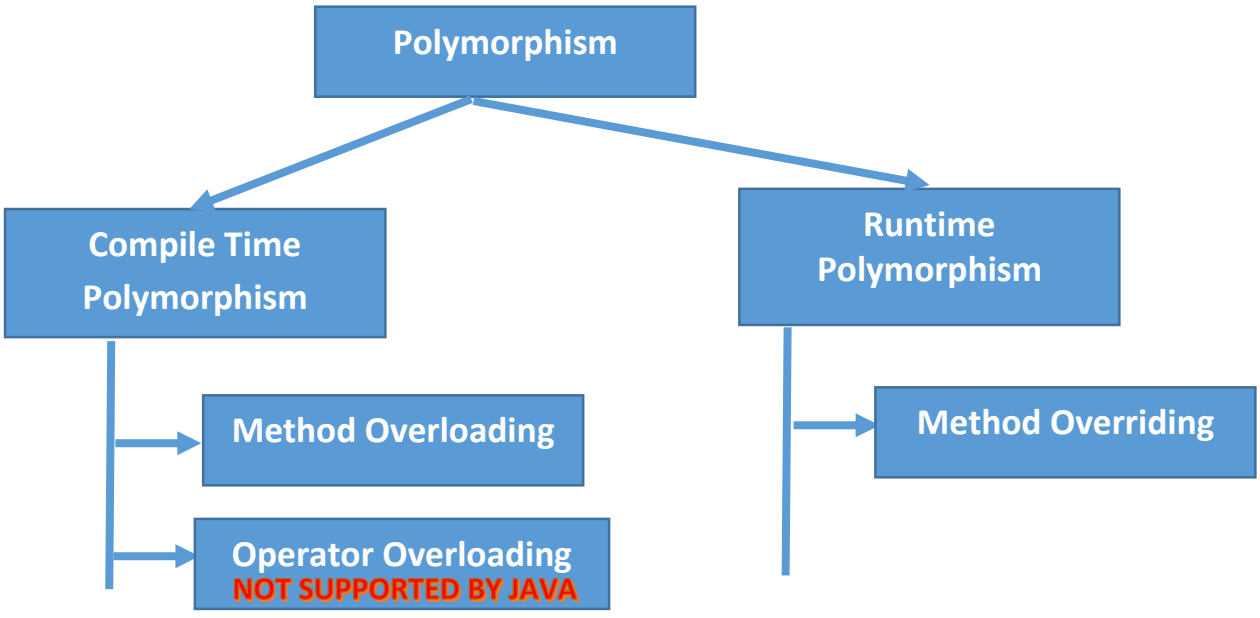
```
class Animal
{
        void eat()
        {
                System.out.println("Eating");
        }
}

class Dog extends Animal        //Single level inheritance
{
        void bark()
        {
                System.out.println("Barking");
        }
}
class cat extends Animal        //Multilevel inheritance
{
        void sleep()
        {
                System.out.println(“Sleeping”);
        }
}

class single
{
        public static void main(String args[])
        {
                cat c=new cat();
                c.eat();
                c.sleep();
                Dog d=new Dog();
                d.bark();
                d.eat();

        }
```

MCQ		
Sr No.	**Question**	**Answer**
1	**The parent child concept is known as ……**	**Inheritance**
2	**The main advantage of inheritance is …..**	**reusability**
3	**Which inheritance is not supported in java?**	**Multiple Inheritance**
4	**In ……inheritance, there is one parent class and multiple child class and all child class access same parent class**	**Hierarchical**
5	**Which inheritance is known as chain of inheritance?**	**Multilevel**
6	**……..inheritance is the combination of more than one type of inheritance**	**Hybrid**

Q-3 Write a short note on Polymorphism

Ans:

- Polymorphism **in Java** is a concept by which we can perform a *single action in different ways*.
- Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.



1) Compile time Polymorphism:

- It is also known as static polymorphism which is achieved by method overloading.
- Method overloading means methods with same name but different parameters.
- Method can be overloaded by change in number of arguments or change in type of arguments.

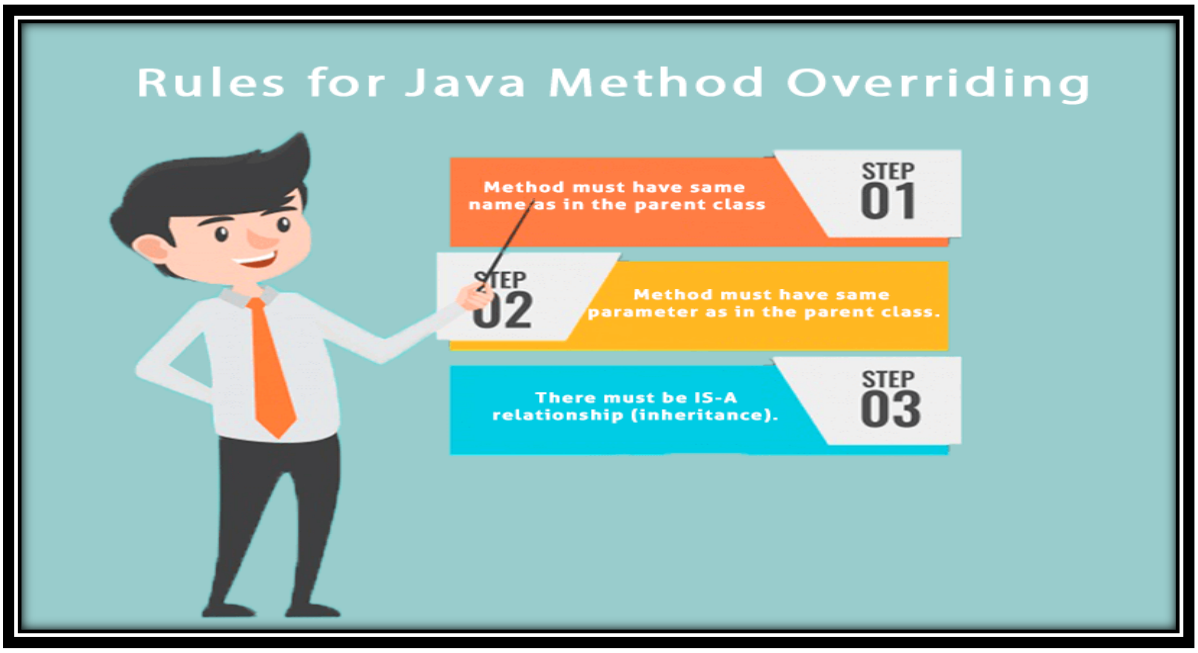
Example:

```
class overload
{
        void add(int a,int b,int c)
        {
                System.out.println(a+b+c);
        }
        void add(int a,int b)
        {
                System.out.println(a+b);
        }
}
class method1
{
        public static void main(String args[])
        {
                overload o1=new overload();
                o1.add(2,3,4);
                o1.add(1,2);
        }
}
```

2) **Run time Polymorphism:**

- It is also known as dynamic polymorphism which is achieved by method overriding.
- It is a process in which a function call to the overridden method is resolved at Runtime.
- In the below example, When an object of child class is created, then the method inside the child class is called. This is because the method in the parent class is overridden by the child class. Since the method is overridden, this method has more

priority than the parent method inside the child class. So, the body inside the child class is executed.



Example:

```
class A
{
	void run()
	{
		System.out.println("Hello");
	}
}

class B extends A
{
	void run()
	{
		System.out.println("Hi");
	}
}
class sam
{
	public static void main(String args[])
	{
		B b1=new B();
		b1.run();
	}
}
```

Compile Time Polymorphism	Runtime polymorphism
1) This type of polymorphism is known as Static binding or early binding	1) This type of polymorphism is known as dynamic binding or late binding
2) It is achieved by method overloading as operator overloading is not supported by java	2) It is achieved by method overriding
3) Inheritance is not used in this type of inheritance	3) Inheritance is used in this type of polymorphism to achieve method overrding.

MCQ		
Sr No.	Question	Answer
1	Compile time polymorphism is also known as …….	Static binding
2	Runtime polymorphism is also known as….	Dynamic Binding
3	Compile time polymorphism is achieved by …….	Method overloading
4	The methods with same name but different parameters is known as ……..	Method overloading
5	Operator overloading is supported by java? (T/F)	False. Not supported
6	Runtime polymorphism is achieved by …..	Method overriding
7	In ……., child class overwrites the method of parent class	Method overriding

Q-4 Write a short note on Constructor.

Ans:

- A constructor is a special method which have same name as class name
- It is automatically called when object of class is created.

Rules for creating Java constructor

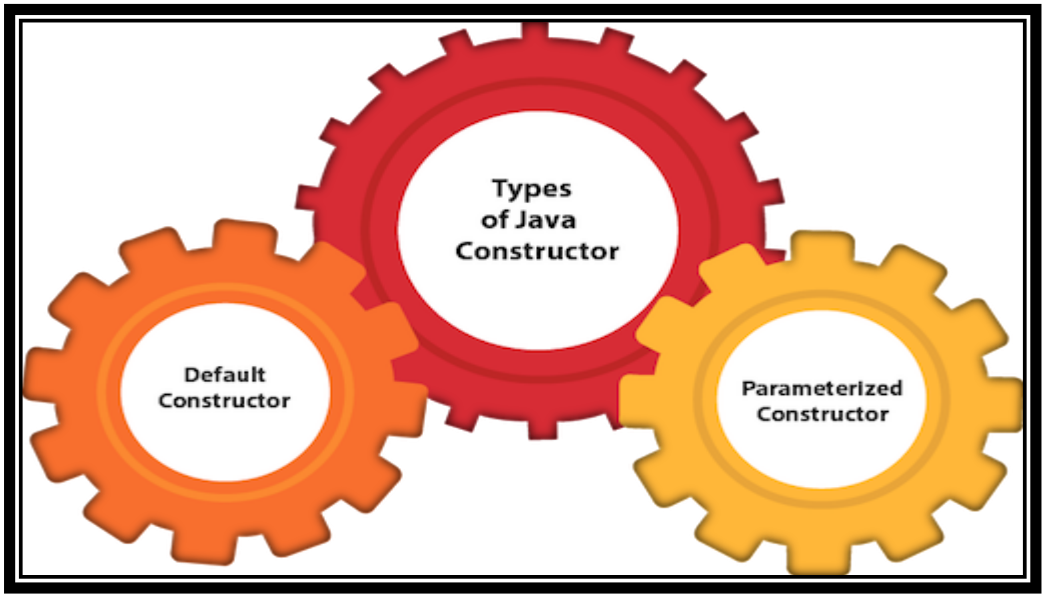
There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructor

- There are two types of constructors in Java:

    1) Default constructor (no-arg constructor)
    2) Parameterized constructor



1) Default Constructor:

- A constructor with zero parameter or no argument is known as default constructor.

Example:

```
class A
{
        A()
        {
                System.out.println("Default constructor");
        }
}

class inhe
{
        public static void main(String args[])
        {
                A a1=new A();
        }
}
```

2) Parameterised Constructor:

- A constructor with parameters is known as parameterised constructor.

Example:

```
class A
{
        int roll;
        A(int r)
        {
                roll=r;
        }
        void display()
        {
                System.out.println(roll);
        }
}

class inhe
{
        public static void main(String args[])
        {
                A a1=new A(10);
                a1.display();
        }
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**Which method have same name as class name**	**Constructor**
2	**Constructor does not have return type (T/F)**	**True**
3	**The constructor with no or 0 arguments is known as……**	**Default constructor**
4	**The constructor with parameters is known as ……..**	**Parameterized constructor**
5	**When constructor is called?**	**When object of class is created.**

Q-5 Write a short note on Constructor overloading

Ans:

- **A constructor with same name but different parameters is known as constructor overloading.**

Example:

```
class cons
{
	int a,b;
	cons()
	{
		System.out.println("Hello");
	}
	cons(int a1)
	{
		a=a1;
	}
	cons(int a.int b1)
	{
		a=a1;
		b=b1;
	}
	void display()
	{
		System.out.println(a);
		System.out.println(b);
	}
}
class consoverload
{
	public static void main(String args[])
	{
		cons c=new cons();
		cons c1=new cons(10);
		cons c2=new cons(10,11);
		c1.display();
		c2.display();
	}
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**Constructor with same name but different parameters is known as …..**	**Constructor overloading**

Q-6 Write a short note on static and non-static members in java

Ans:

Static Variables:

- When a variable is declared as static, then a single copy of the variable is created and shared among all objects at a class level. Static variables are, essentially, global variables. All instances of the class share the same static variable.
- We can create static variables at class-level only

Example:

```
class test
{
        static int a=20;
        void display()
        {

                a++;
                System.out.println(a);
        }

}
class sample
{
        public static void main(String args[])
        {
                test t1=new test();
                test t2=new test();
                test t3=new test();
                t1.display();
                t2.display();
                t3.display();
        }
}
```

Non-Static Variables:

- When a variable is declared as non-static, then individual copy of the variable is created for the different objects. Static variables are, essentially, local variables. All object of the class share the different non-static variable.

Example:

```
class test
{
        int a=20;
        void display()
        {

                a++;
                System.out.println(a);
        }

}
class sample
{
        public static void main(String args[])
        {
                test t1=new test();
                test t2=new test();
                test t3=new test();
                t1.display();
                t2.display();
                t3.display();
        }
}
```

Static Members	Non-static Members
1) Static members can be accessed directly using classname	1) Non-static members can be accessed using object of class
2) Static variables can be accessed by static and non-static methods both	2) Non-static variables cannot be accessed inside static method
3) All the objects of class share same static variables	3) All the objects of the class have its own non-static variable.
4) Static variables are like global variables	4) Non-static variables are like local variables.
5) static keyword is used to declare the static members.	5) No keyword is required to declare non-static members.

MCQ		
Sr No.	Question	Answer
1	In …….variable, only 1 copy of variable is shared by all the objects of the class	static
2	Which keyword is used to declare static variable?	static
3	In …..variable, all the objects have their own copy of variable	Non-static
4	Static variables can be accessed by both …….and …….methods	static and non-static
5	Static variables are like …….variables	global
6.	Non-static variables are like ……variables	local

Q-7 Write a short note on varargs

Ans:

- The varrags allows the method to accept zero or multiple arguments.
- If we don't know how many argument we will have to pass in the method, varargs is the better approach.
- The varargs uses ellipsis i.e. three dots after the data type. Syntax is as follows:

Syntax:

```
return_type method_name(data_type... variableName)
{
}
```

Example:

```
class varg
{
        public void display(int...values)
        {
                System.out.println("hello");

        for(int i:values)
        System.out.println(i);
        }
}
class B
{
        public static void main(String args[])
        {
                A a=new A();
                a.display();
                a.display(1,2,3,4);
                a.display(12,22);
        }
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**If we don't know how many argument we will have to pass in the method, …… is the better approach**	**varargs**
2	**varargs uses ……**	**ellipsis (…)**

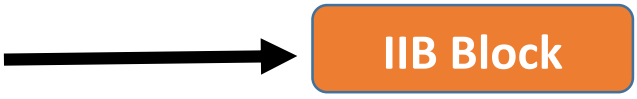
Q-8 Write a short note on IIB block in java

Ans:

- IIB stands for instance initializer block
- **Instance Initializer block** is used to initialize the instance data member
- It run each time when object of the class is created.

Example:

```
class iib
{
        int a;

        {
                a=10;
        }
        public void display()
        {
                System.out.println(a);
        }

}
class im
{
        public static void main(String args[])
        {
                iib i=new iib();
                i.display();
                iib i2=new iib();
                i2.display();
        }
}
```



MCQ		
Sr No.	**Question**	**Answer**
1	**IIB stands for ……**	**Instance Intializer Block**
2	**IIB runs each time the …….is created**	**object of class**

JAVA UNIT-2 MATERIAL

0Ch. No.	Topic	Detail+s	Marks
2	Inheritance and java packages	• Universal Class (Object Class) • Access Specifiers (public, private, protected, default, private protected) • Constructors in inheritance • Method Overriding • Interface, Object Cloning • Nested and Inner Class • Abstract and -Final Class • Normal import and Static Import • Introduction to Java API Packages and imp. Classes java.lang , java.util ,java.io , java.net, java.awt ,java.awt.event , java.applet , java.swing • java.lang Package Classes (Math, Wrapper Classes, String, String Buffer) • java.util Package Classes (Random, Date, GregorianCalendar • StringTokenizer, Collection in Java -Vector, HashTable, LinkedList, SortedSet, Stack, Queue, Map • Creating and Using UserDefined package and sub-package	14

Q-1 Explain Universal class

Ans:

- The Object class is the parent class of all the classes in java by default. In other words, it is the topmost class of java.
- Object class is present in java. Lang package. Every class in Java is directly or indirectly derived from the Object class.
- If a class does not extend any other class then it is a direct child class of Object and if extends another class then it is indirectly derived.
- Therefore the Object class methods are available to all Java classes.
- Hence object class acts as a root of inheritance hierarchy in any Java Program.
- The Object class is beneficial if you want to refer any object whose type you don't know.
- Following are the methods of object class.

Sr No.	Method Name	Description
1	toString()	It is used to convert an object to string Note: It is always recommended to override toString() method to get our own string representation.
2	hashcode()	It returns a hashvalue that is used to search the object in the collection Note: For every object, JVM generates unique number which is known as hascode.
3	equals	It compares given object to “this” object (the object on which method is called)
4	getClass()	It returns the class of the object and is used to get actual runtime class of the object.
5	finalize()	This method is called just before an object is garbage collected. It is called on an object when garbage collector determines that there are no references to the object. Note: finalize() is just called once on an object

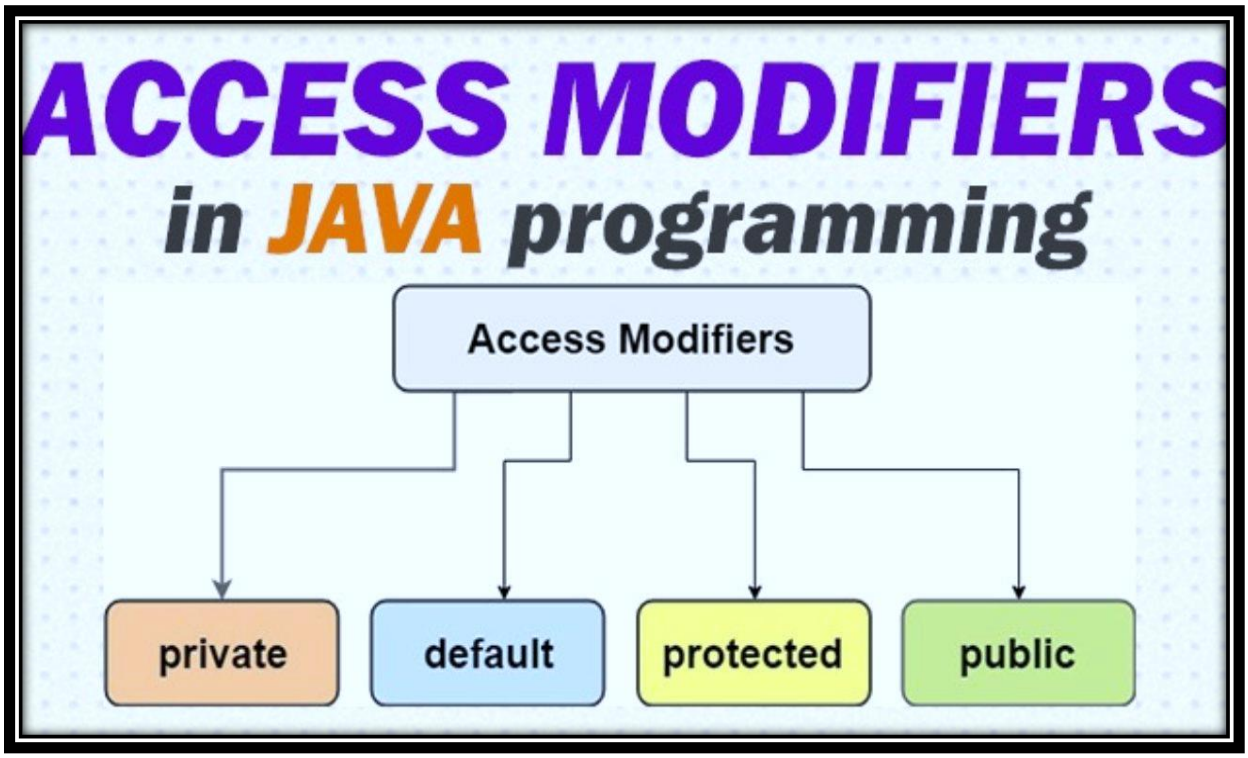
6	clone()	It returns the new object that is exactly same as this object
7	notify()	wakes up single thread, waiting on object's monitor
8	notifyall()	wakes up all thread, waiting on object's monitor
9	wait()	Causes the current thread to wait for the specified milliseconds until another thread notifies.

MCQ		
Sr No.	**Question**	**Answer**
1	**Which is the universal class in java?**	**Object**
2	**Object class is present in ……package**	**java.lang**
3	**Which method of object class is used to return a hash value that is used to search the object in collection?**	**hashcode()**
4	**Which method of object class generates exactly same object as this object?**	**clone()**
5	**Which method of object class is called just before the object is garbage collected?**	**finalize()**

Q-2 Write a short note on Access Specifier

Ans:

- The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.
- There are 4 types of java access modifiers



1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class. **A class cannot be private except nested class.**

```
class A
{
		private int data=40;
		private void msg()
		{
			System.out.println("Hello java");
		}
}
class Simple
{
	public static void main(String args[])
	{
		A obj=new A();
		System.out.println(obj.data);//Compile Time Error
		 obj.msg();//Compile Time Error
	}
}
```

2. Default:

- A default access modifier in java has no specific keyword
- If we do not explicitly specify any access modifier for classes, methods, variables, etc., then by default the default access modifier is considered

```
class A
{
        void display()
        {
                System.out.println("hello");
        }
}
. class Simple
{
.       public static void main(String args[])
        {
.               A obj=new A();
                obj.display();
        }
}
```

Note:

- **In the above example, we have a class and method inside it without any access Specifier. Hence both the class and method have default access**

3. **Protected**: The protected access Specifier allows access to entities through subclasses of the class in which the entity is declared.

```
class A
    {
                Protected void display()
                {
                        System.out.println(“hello”);
                }
    }
    class B extends A
    {

    }
class Simple
{
        public static void main(String args[])
        {
                 B obj=new B();
                obj.display();
        }
}
```

4. **Public**: A class or method or data field specified as public is accessible from any class.

```
class A
    {
                public void display()
                {
                        System.out.println("hello");
                }
}
class Simple
{
        public static void main(String args[])
        {
                A obj=new A();
                obj.display();
        }
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**……..modifier is accessible from everywhere**	**public**
2	**The access level of ……modifier is within the package and outside the package through inheritance**	**protected**
3	**Which is the default access modifier?**	**default**

Q-3 Write a short note on Constructor in inheritance

Ans:

- A constructor in Java is similar to a method with a few differences. Constructor has the same name as the class name. A constructor doesn't have a return type.
- A Java program will automatically create a constructor if it is not already defined in the program. It is executed when an instance of the class is created.

- A constructor cannot be static, abstract, final or synchronized. It cannot be overridden.
- When the constructor is used in inheritance, then the constructor of base class is executed first and then constructor of child class is executed.

Constructor in multi level inheritance

```
class parent
{
        parent()
        {
                System.out.println("Parent class constructor");
        }
}
class child extends parent
{
        child()
        {
                System.out.println("Child class constructor");
        }
}
class child1 extends child
{

        child1()
        {
                System.out.println("Child1 constructor");
        }
}
class sample
{
        public static void main(String args[])
        {
                child1 c=new child1();
        }
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**In case of constructor in inheritance, the order of execution of constructor will be ……**	**parent class constructor** **child class constructor**
2	**In case of constructor in multilevel inheritance, the order of execution of constructor will be…..**	**parent class constructor** **child class constructor** **child1 constructor**

Q-4 Write a short note on Interface

Ans:

- An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.
- Interface in java is used to achieve abstraction and multiple inheritance
- Interface can only have abstract methods that is methods without body
- "Implements" keyword is used when the class implements interface.

Syntax:

```
interface interfacename
{
    returntype methodname();
}
```

Example:

Multiple inheritance in java using interface:

- **Multiple inheritance means multiple parent class and one child class. The child class inherits from multiple parent class.**
- **Multiple inheritance is not supported by java. But through the interface multiple inheritance is achieved in java.**

Example:

```
interface a
{
        void display();
}
interface b
{
        void display1();
}
class mul implements a,b
{
        public void display()
        {
                System.out.println("hello");
        }
        public void display1()
        {
                System.out.println("hi");
        }
}

class multiple
{
        public static void main(String args[])
        {
                mul m=new mul();
                m.display();
                m.display1();
        }
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**class ……interface**	**implements**
2	**Which keyword is used to declare the interface?**	**interface**
3	**interface can have only ……methods**	**abstract**
4	**Multiple inheritance in java is achieved using ……..**	**interface**

Q-5 Write a short note on Object cloning

Ans:

- The object cloning is a way to create exact copy of an object.
- The clone() method of Object class is used to clone an object.
- The **clone() method** is defined in the Object class.
- Every class that implements clone() should call super.clone() to obtain the cloned object reference.
- The class must also implement java.lang.Cloneable interface whose object clone we want to create otherwise it will throw CloneNotSupportedException when clone method is called on that class's object.

Syntax:

public Object clone() **throws** CloneNotSupportedException

Example:

```
class stu implements Cloneable
{
        int x,y;

        stu
        {
                x=10;
                y=20;
        }
public Object clone()throws CloneNotSupportedException
{
        return super.clone();
}
}
class B
{
public static void main(String args[])throws CloneNotSupportedException
{

        A a1=new A();
        A a2=(A)a1.clone();

System.out.println(a1.x)
System.out.println(a1.y);
System.out.println(a2.x);
System.out.println(a2.y);
}
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**Which method is used to create exact copy of the object?**	**clone()**
2	**To use clone(), which interface class should implement?**	**Cloneable**
3	**Every class that implements clone() should call …….. to obtain the cloned object reference.**	**super.clone()**

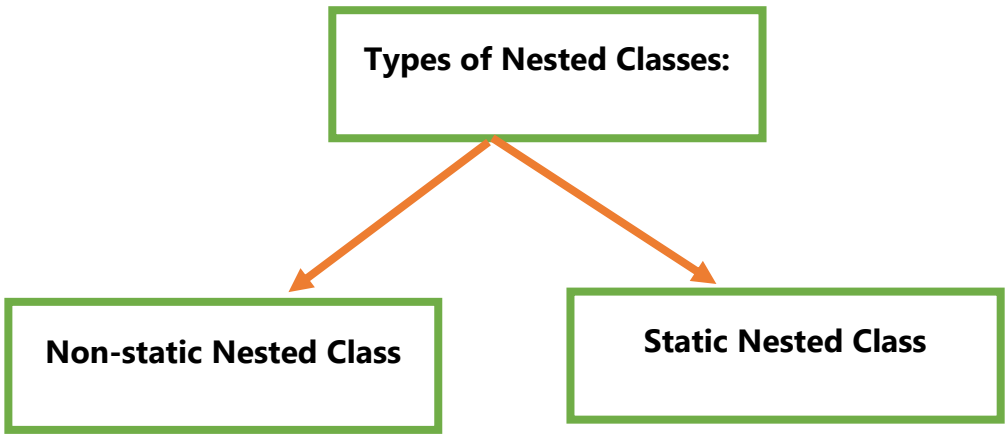
Q-6 Write a short note on Nested and Inner Class

Ans:

- **Java inner class** or nested class is a class that is declared inside the class or interface.
- We use inner classes to logically group classes and interfaces in one place to be more readable and maintainable.
- Additionally, it can access all the members of the outer class, including private data members and methods.

Syntax:

```
class Java_Outer_class
{
        //code
                class Java_Inner_class
                {
        //code
                }
}
```



1) Non-Static Nested class:

- A non-static nested class is a class within another class. It has access to members of the enclosing class (outer class). It is commonly known as inner class
- As inner class exists inside outer class, the object of outer class must be created in order to create object of inner class

Example:

```
class outer
{
        int x=10;
        class inner
        {
                int y=4;
                public int display()
                {
                        return x;
                }
        }
}

class nested
{
        public static void main(String args[])
        {
                outer o=new outer();
                outer.inner i=o.new inner();
                System.out.println(i.y);
                System.out.println(i.display());
        }
}
```

Note:

Dot operator(.) is used to create the object of inner class using outer class

2) Static Nested class:

- Static class inside another class is known as static nested class.
- Static nested class are not known as static inner class.
- The main difference between static nested class and inner class is that a static nested class cannot access member variables of the outer class.it is because the static nested class does not require you to create an object of outer class.

Example:

```
class outer
{
        int x=10;
        class inner
        {
                int y=4;
                public int display()
                {
                        return x;
                }
        }
}

class nested
{
        public static void main(String args[])
        {
                outer o=new outer();
                outer.inner i=new inner();
                System.out.println(i.y);
                System.out.println(i.display());
        }
}
```

****Difference between static and non-static nested class**

Non-static nested class	Static Nested Class
1) In non-static nested class, static keyword is not used	1) In static nested class, static keyword is used
2) Non-static nested class is known as inner class	2) Static nested class is not known as inner class
3) Non-static nested class requires object of outer class	3) Static nested class does not require object of outer class
4) Non-static nested class can access the members of outer class	4) Static nested class cannot access members of the outer class

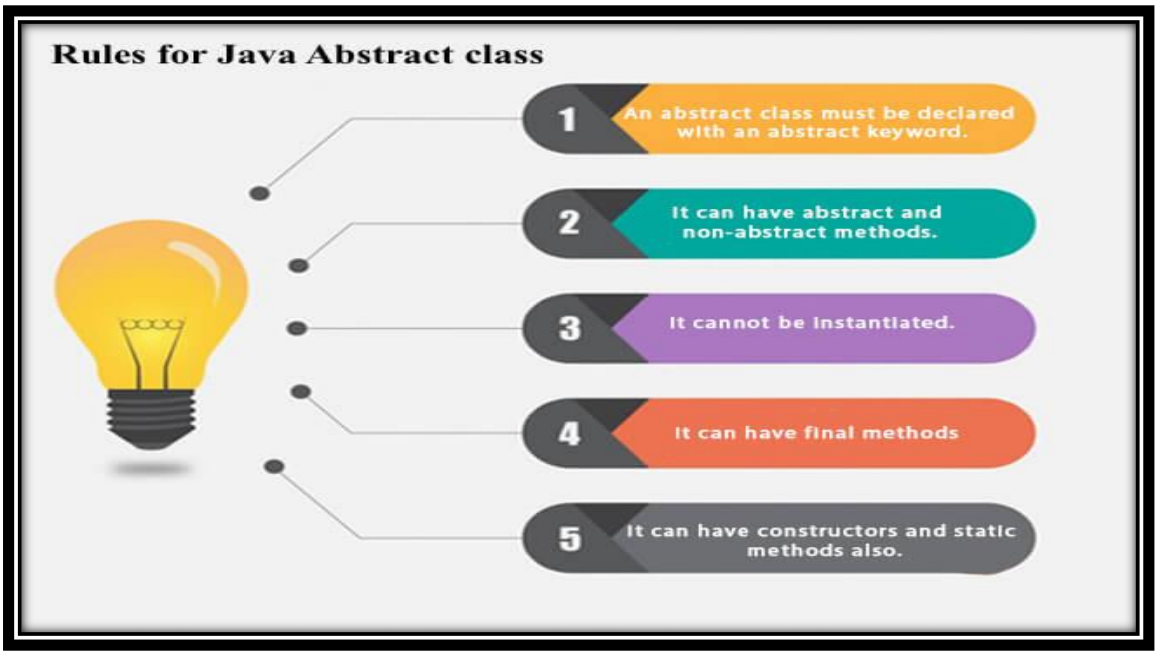
MCQ		
Sr No.	**Question**	**Answer**
1	**Which keyword is used to declare static nested class?**	**Static**
2	**In ……., object of outer class is required**	**non-static**
3	**……..can access members of outer class**	**Non-static nested class**

Q-6 Write a short note on abstract class and final class

Ans:

Abstract Class:

- A class which is declared as abstract is known as an **abstract class**.
- It can have abstract and non-abstract methods.
- It needs to be extended and its method implemented.
- It cannot be instantiated.

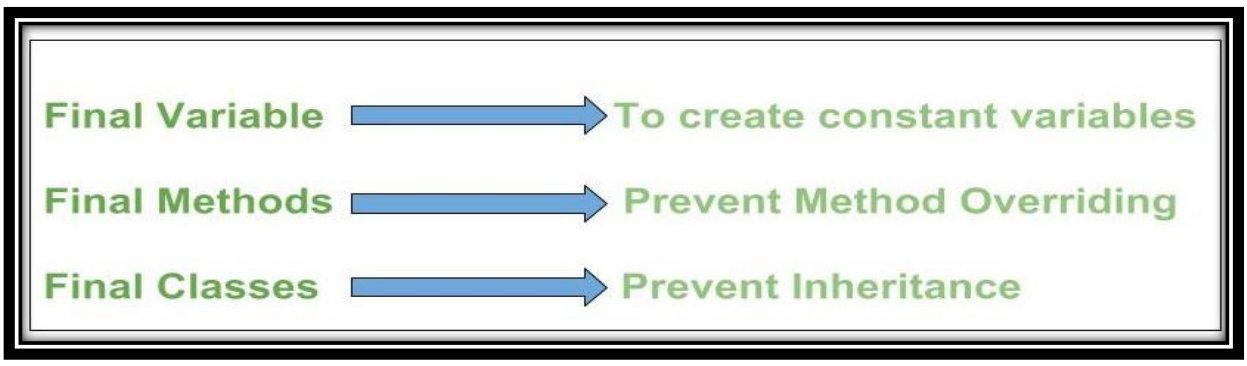


Example:

```
abstract class bike
{
        abstract void run();
        void display()
        {
                System.out.println("hello");
        }
}

class honda extends bike
{
        public void run()
        {

                System.out.println("hi");
        }
}

class sampe
{
        public static void main(String args[])
        {
                honda h=new honda();
                h.run();
                h.display();
        }
}
```

Final:

- The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:
    - Variable
    - Method
    - Class



- A class which cannot be inherited or extended is known as final class.

Example:

```
final class bike
{


}

class ja extends bike
{
        void display()
        {
                System.out.println("hi");
        }
        public static void main(String args[])
        {
                bike b=new bike();
                b.display();   //Compile time error
        }
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**Which keyword is used to declare the class as abstract?**	**abstract**
2	**Can we create the object of abstract class?**	**No**
3	**Abstract class can contain non-abstract methods?**	**True**
4	**When final keyword is applied to the class it prevents ……**	**Inheritance**

Q-7 Write a short note on normal import and static import

Ans:

Normal import:

- The import allows the java programmer to access classes of a package without package qualification
- The import provides accessibility to classes and interface

```
class ke
{
        public static void main(String args[])
        {
                System.out.println(Math.sqrt(25));
                System.out.println(Math.pow(2,2));
        }
}
```

static import:

- In Java, static import concept is introduced in 1.5 version.
- With the help of static import, we can access the static members of a class directly without class name or any object.
- For Example: we always use sqrt() method of Math class by using Math class i.e. **Math.sqrt()**, but by using static import we can access sqrt() method directly.

```
import static java.lang.Math.*;

class ke
{
        public static void main(String args[])
        {
                System.out.println(sqrt(25));
                System.out.println(pow(2,2));
        }
}
```

** Difference between normal import and static import

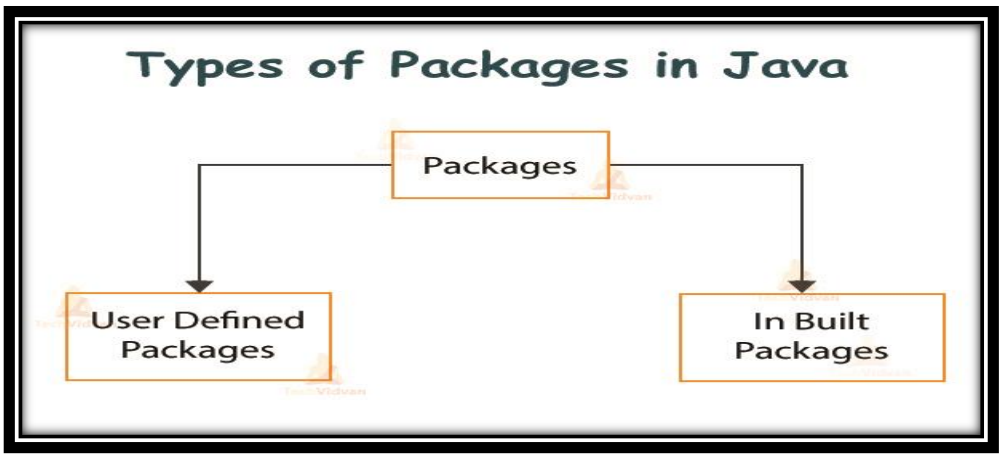
normal import	Static import
1) The normal import provides access to class and interface	1) static import provides access to the static members of the class.
2) The normal import allows to access the class of the package without package qualification	2) Static import feature allows to access the static members of a class without class qualification.

MCQ		
Sr No.	Question	Answer
1	…….import provides access to class and interface	normal
2	……..import provides access to the static members of the class	static

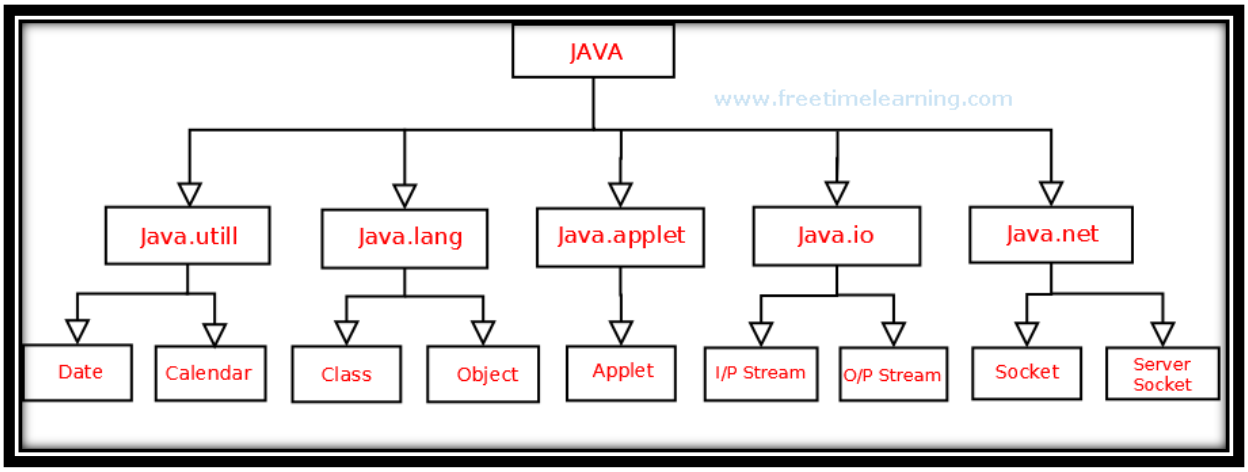
Q-8 what is package? List out the packages in java

Ans:

- Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces. Packages are used for: **Preventing naming conflicts**
- For example there can be two classes with name Employee in two packages, college.staff.cse.Employee and college.staff.ee.Employee



- Following are the types of in built packages:



Built-in Packages:

- These packages consist of a large number of classes which are a part of Java **API**.
- Some of the commonly used built-in packages are:
  - **java.lang:** Contains language support classes(e.g classed which defines primitive data types, math operations). This package is automatically imported.
  - **java.io:** Contains classed for supporting input / output operations.
  - **java.util:** Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.
  - **java.applet**: Contains classes for creating Applets.
  - **java.awt**: Contain the classes for implementing the components of graphical user in **java.applet** terfaces like button, menus etc.
  - **java.net**: Contain classes for supporting networking operations**.**

java. lang Package

- Provides classes that are fundamental to the design of the Java programming language.
- Following are few important classes of java.lang package
  1. Math
  2. Wrapper Classes
  3. String
  4. String Buffer

Math Class of java.lang package

- The **java.lang.Math** class contains various methods for performing basic numeric operations such as the logarithm, cube root, and trigonometric functions etc. The various java math methods are as follows:
- All the methods of the Math class are static so it can be called by class name
- Basic Math methods

Method	Description
Math.abs( )	It will return the Absolute value of the given value.
Math.max( )	It returns the Largest of two values.
Math.min( )	It is used to return the Smallest of two values.
Math.round( )	It is used to round of the decimal numbers to the nearest value.
Math.sqrt( )	It is used to return the square root of a number.
Math.cbrt( )	It is used to return the cube root of a number.
Math.pow( )	It returns the value of first argument raised to the power to second argument.
Math.ceil( )	It is used to find the smallest integer value that is greater than or equal to the argument or mathematical integer.
Math.floor( )	It is used to find the largest integer value which is less than or equal to the argument and is equal to the mathematical integer of a double value.
Math.random( )	It returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
Math.cos( )	Returns the trigonometric cosine of an angle.
Math.sin( )	Returns the trigonometric sine of an angle.
Math.tan( )	Returns the trigonometric tangent of an angle.
Math.log( )	Returns the natural logarithm (base e) of a double value.

```
import java.lang.*;

class mathfunctions
{
	public static void main(String args[])
	{
		System.out.println(“Absolute” +Math.abs(-2));
		System.out.println("maximum “ +Math.ceil(2.56));
		System.out.println("maximum “ +Math.floor(2.56));
		System.out.println("round “ +Math.round(2.56));
		System.out.println("random “ +Math.random());
		System.out.println(“power “ +Math.pow(2,3));
		System.out.println(“Minimum “+Math.min(24,56));
		System.out.println(“Maximum “+Math.max(24,56));
		System.out.println(“square “+Math.sqrt(25));
		System.out.println(“square “+Math.cbrt(25));
		System.out.println(“cos” +Math.cos(25));
	}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**Math.ceil gives …….**	**Maximum round value of floating point number**
2	**Math.sqrt() gives…..**	**Square root of number**
3	**Math class is included in ……**	**java.lang**
4	**…….is the default package**	**java.lang**

Wrapper class in java.lang package

- The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive*.
- **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

Primitive Type	Wrapper Class	Primitive Type	Wrapper Class
byte	Byte	float	Float
boolean	Boolean	int	Integer
char	Character	long	Long
double	Double	short	Short

Autoboxing:

- The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.

Example:

```
class wrapper1
{
        public static void main(String args[])
        {
        int a=20;
        Integer j=a; //Converting primitive data type to Integer wrapper class
                System.out.println(j);
        }
}
```

Unboxing:

- The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of unboxing.

```
class wrapper1
{
        public static void main(String args[])
        {
                Integer a=new Integer(3);
                int i=a; //Converting Wrapper class into primitive data type
                System.out.println(i);
        }
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**…….Provides the mechanism *to convert primitive into object and object into primitive*.**	**Wrapper Class**
2	**The process of converting primitive data type into object is known as ……**	**autoboxing**
3	**The process of converting object into primitive data type is known as ….**	**unboxing**

String class in java.lang package

- The **java.lang.String** class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.Strings are constant, their values cannot be changed after they are created.

Method	Description
char charAt(int index)	Returns char value for the particular index

int length()	Returns string length
String substring(int beginIndex)	Returns substring for given begin index.
String substring(int beginIndex, int endIndex)	Returns substring for given begin index and end index.
boolean contains(Char Sequences)	Returns true or false after matching the sequence of char value.
boolean equals(Object another)	Checks the equality of string with the given object.
Boolean isEmpty()	Checks if string is empty.
String concat(String str)	Concatenates the specified string.
String replace(char old, char new)	Replaces all occurrences of the specified char value.
static String equalsIgnoreCase(String another)	Compares another string. It doesn't check case.
int indexOf(int ch)	Returns the specified char value index.
String toLowerCase()	Returns a string in lowercase.
String toUpperCase()	Returns a string in uppercase.
String trim()	Removes beginning and ending spaces of this string.
int compareTo(String)	Compares this String to another specified String if match then return zero(0) otherwise not zero(0).
int compareToIgnoreCase(String)	Compares two strings lexicographically, ignoring case differences.

Example:

```
class stringfunctions
{
	public static void main(String args[])
	{
		String s="hello";
		String s1="hello";
		String s3="karishma";
		String s4="rupani";
		String s5="hi";
		String s6="Hi";
		System.out.println("Charat" +s.charAt(1));
		System.out.println("length" +s.length());
		System.out.println("substring" +s.substring(1));
		System.out.println("substring" +s.substring(1,3));
		System.out.println("index" +s.contains("el"));
		System.out.println("Equality" +s.equals(s1));
		System.out.println("Empty" +s.isEmpty());
		System.out.println("Concate"+s3.concat(s4));
		System.out.println("replace"+s.replace('l','k'));
		System.out.println("Uppercase" +s3.toUpperCase());
		System.out.println("Lowercase" +s3.toLowerCase());
		System.out.println("IndexOf" +s.indexOf('o'));
		System.out.println("ignorecase" +s5.equalsIgnoreCase(s6));

	}
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**Which class contains the string functions?**	**String class**
2	**substring(startind,endind) gives …….**	**Substring from start index to end index.**
3	**Which function of string class is used to merge 2 strings?**	**concat()**

4	**Which function returns the index of the given character**	**indexOf()**

StringBuffer class in java.lang package

- **StringBuffer** is a peer class of **String** that provides much of the functionality of strings.
- The string represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences
- **StringBuffer** may have characters and substrings inserted in the middle or appended to the end.
- It will automatically grow to make room for such additions and often has more characters preallocated than are actually needed, to allow room for growth.

Method	Description
append(String s)	Is used to append the specified string with this string.
insert(int offset, String s)	Is used to insert the specified string with this string at the specified position.
replace(intstartIndex, intendIndex, String str)	Is used to replace the string from specified startindex and endindex.
delete(int startIndex, int endIndex)	Is used to delete the string from specified startindex and endindex.
reverse()	Is used to reverse the string.
capacity()	Is used to return the current capacity.
charAt(int index)	Is used to return the character at the specified position.
length()	Is used to return the length of the string i.e. Total number of characters.
substring(int beginIndex)	Is used to return the substring from the specified begin index.
int indexOf(String str)	returns the index within this string of the first occurrence of the specified substring.

Example:

```
class stringfunctions
{
        public static void main(String args[])
        {
                StringBuffer s=new StringBuffer("hello");
                System.out.println("Length" +s.length());
                System.out.println("Specified" +s.insert(2,"hi"));
                System.out.println("Replace" +s.replace(1,3,"ee"));
                System.out.println("Deleting" +s.delete(1,3));
                System.out.println("Reversing" +s.reverse());
                System.out.println("Capacity" +s.capacity());
                System.out.println("Charat" +s.charAt(2));
                System.out.println("Substring" +s.substring(1));

        }

}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**……..is the peer class of the String class**	**StringBuffer**
2	**……..is the growable sequence of characters**	**StringBuffer**
3	**Which method is used to add insert the string in the specified string?**	**insert()**

java.util Package

- The **java.util** packages provide support for the event model, collections framework, date and time facilities, and contain various utility classes.
- Following are the classes in java.util package
  - Random
  - Date
  - GregorianCalendar
  - Vector

- HashTable
- StringTokenizer
- Collections in Java : Linked List, SortedSet, Stack, Queue, Map

Random class in java.util package

- Java Random class is used to generate a stream of pseudorandom numbers
- This class provides various method calls to generate different random data types such as float, double, int.

Methods	Description
doubles()	Returns an unlimited stream of pseudorandom double values.
ints()	Returns an unlimited stream of pseudorandom int values.
longs()	Returns an unlimited stream of pseudorandom long values.
next()	Generates the next pseudorandom number.
nextBoolean()	Returns the next uniformly distributed pseudorandom boolean value from the random number generator's sequence
nextByte()	Generates random bytes and puts them into a specified byte array.
nextDouble()	Returns the next pseudorandom Double value between 0.0 and 1.0 from the random number generator's sequence
nextFloat()	Returns the next uniformly distributed pseudorandom Float value between 0.0 and 1.0 from this random number generator's sequence
nextInt()	Returns a uniformly distributed pseudorandom int value generated from this random number generator's sequence
nextLong()	Returns the next uniformly distributed pseudorandom long value from the random number generator's sequence.

Example:

```
import java.util.*;
class randomn
{
        public static void main(String args[])
        {
                Random r=new Random();
                System.out.println("Integer val" +r.nextInt());
                System.out.println("Next decimal" +r.nextDouble());
        }
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**Random class comes in which package?**	**Java.util**
2	**Which class generates pseudorandom numbers?**	**Random()**

Date class in java.lang package

- The java.util.Date class represents date and time in java.

Method	Description
boolean after(Date date)	Tests if current date is after the given date.
boolean before(Date date)	Tests if current date is before the given date.
int compareTo(Date date)	Compares current date with given date.
boolean equals(Date date)	Compares current date with given date for equality.
long getTime()	Returns the time represented by this date object.
void setTime(long time)	Changes the current date and time to given time.

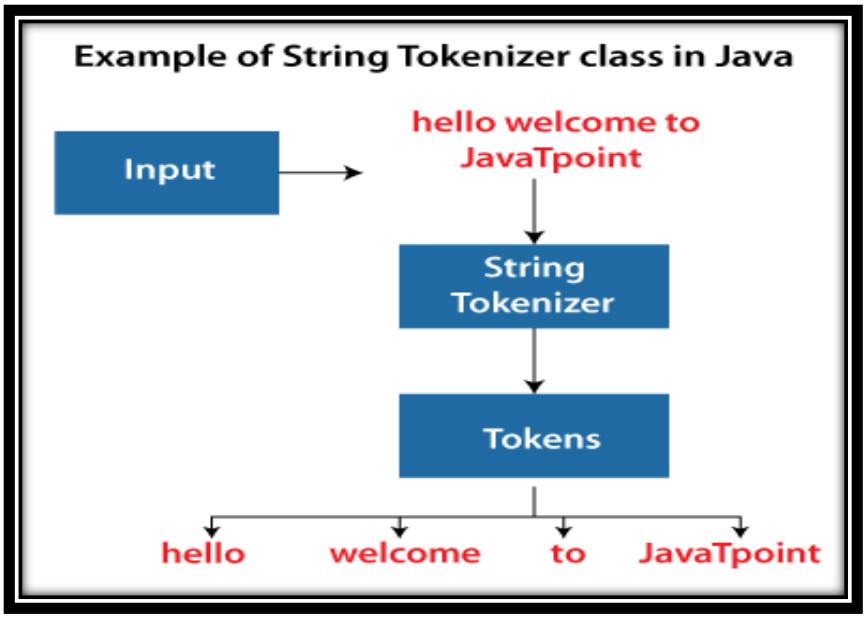
Example:

```
import java.util.*;

public class main1
{
   public static void main(String[] args)
   {
                Date d=new Date();
                Date d1=new Date(03-12-2022);
                Date d2=new Date(01-11-2022);

                System.out.println(d);
                System.out.println("checking "+d1.before(d2));
                System.out.println("checking " +d1.after(d2));
                System.out.println("Compare "+d1.compareTo(d2));
                System.out.println("Compare "+d1.equals(d2));
                System.out.println("Time "+d.getTime());
                System.out.println("Time" +d.hashCode());
        }
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**Which class of java.util package is used to give date and time**	**Date**
2	**Which method is used to check that current date is before given date or not?**	**before()**
3	**Which method of Date class is used to generate unique ID of Date object**	**hashcode**

StringTokenizer class in java.util package

- The java.util.StringTokenizer class allows you to break a string into tokens. It is simple way to break string.



Method	Description
booleanhasMoreTokens()	Checks if there is more tokens available.
String nextToken()	Returns the next token from the stringtokenizer object.
String nextToken(String delim)	Returns the next token based on the delimeter.
booleanhasMoreElements()	Same as hasmoretokens() method.
Object nextElement()	Same as nexttoken() but its return type is Object.
intcountTokens()	Returns the total number of tokens.

Example:

```
import java.util.*;

class main1
{
   public static void main(String[] args)
   {
         StringTokenizer st=new StringTokenizer("my name is karishma");
         System.out.println("Total number of Tokens: "+st.countTokens());
         while(st.hasMoreTokens())
         {
               System.out.println(st.nextToken());
         }

         }
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**Which class of java.util package allows you to break the string into tokens?**	**StringToeknizer**
2	**Which method is used to count total no. of tokens in the string?**	**countTokens()**
3	**Which method is used to print next token of the string?**	**nextToken()**

Gregorian calendar class in java.util package

- Gregorian calendar is a concrete subclass(one which has implementation of all of its inherited members either from interface or abstract class) of a **Calendar.**
- The major difference between Gregorian calendar and Calendar class are that the Calendar class being an abstract class cannot be instantiated.

```
Calendar cal = Calendar.getInstance();
```

- **GregorianCalendar** Class being a concrete class, can be instantiated. So an object of the **GregorianCalendar** Class

```
GregorianCalendar gcal = new GregorianCalendar();
```

Example:

```
import java.util.*;
class greg
{
        public static void main(String args[])
        {
                Calendar cal=Calendar.getInstance();
                GregorianCalendar c=new GregorianCalendar();
                System.out.println("Calendar date:"+cal.getTime());
                System.out.println("Greg" +c.getTime());
        }
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**…….is the subclass of Calendar Class**	**Gregorian Calendar**
2	**We cannot create object of Calendar Class directly (T/F)?**	**True**
3	**Which method is used to create the object of calendar class?**	**getInstance()**

Collection class in java.util package

- The Collection interface is the root interface of the collections framework hierarchy. A Collection represents a single unit of objects, i.e., a group.
- Java does not provide direct implementations of the Collection interface but provides implementations of its sub interfaces like List, Set, and Queue.
- Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
- Following are the collections in java:
    - Vector
    - HashTable
    - Linked List
    - Sorted Set
    - Stack
    - Queue
    - Map

Vector class in java.util package

- **Vector** is like the *dynamic array* which can grow or shrink its size. Unlike array, we can store n-number of elements in it as there is no size limit.

void add(int index, Object element)	Inserts the element at the given position.
void add()	Adds the elements in vector
void clear( )	Removes all of the elements from this.
int lastElement()	Returns the last element of vector
int firstElement()	Returns the first element of vector
int indexOf(Object element)	Searches for the first occurrence of the given argument.
boolean remove(Object o)	Removes the first occurrence of the specified element.
void removeElementAt(int index)	Deletes the component at the specified index.
void clone()	Creates the clone of vetor.
int capacity()	Returns the capacity of vector.
int size( )	Returns the number of components in this vector.
boolean contains()	Checks whether the element is contained in vector or not.

```
import java.util.*;
class greg
{public static void main(String args[])
{
Vector<Integer>ve=new Vector<Integer>(5);ve.add(10);
ve.add(20);ve.add(30);
ve.addElement(40);
System.out.println("Elements of vector" +ve)
System.out.println("Clone" +ve.clone());
System.out.println("Capacity" +ve.capacity());
System.out.println("Size" +ve.size());
System.out.println("Firstelement" +ve.firstElement());
System.out.println("Firstelement" +ve.lastElement());
System.out.println("Index" +ve.indexOf(20));
System.out.println("Contains" +ve.contains(10));
ve.remove(0);
System.out.println("Elements after removal" +ve);
}
}
```

Hashtable class in java.util package

- The **Hashtable** class implements a hash table, which maps keys to values.

put(key,value)	Maps the specified key to the specified value in hashtable
remove(key)	Removes the element specified at key in the hashtable
size()	Returns the number of keys in the hashtable
replace(key,value)	Replaces the value at given key in the hashtable
Boolean containsKey(key)	Checks if hastable contains specified key or not
Boolean containsValue(value)	Checks if hashtable contains specified value or not

```
import java.util.*;
class hash1
{
	public static void main(String args[])
	{
		Hashtable<Integer,String> h=new Hashtable<Integer,String>();
		h.put(1,"abc");
		h.put(2,"xyz");
		System.out.println("Mapping" +h);
		h.put(2,"ddd");
		System.out.println("Mapping" +h);
		h.remove(2);
		System.out.println("Mapping" +h);
		System.out.println("Check key:" +h.containsKey(1));
		System.out.println("Check value: "+h.containsValue("abc"));
	}
}
```

Linked List class in java.util package

- Linked List is a part of the Collection framework present in java.util package.
- This class is an implementation of the LinkedList data structure which is a linear data structure where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part.
- The elements are linked using pointers and addresses.
- Each element is known as a node.

add(ele)	Adds the element in the linked list
addFirst(ele)	Adds the element in the first node of the linked list
addLast(ele)	Adds the element in the last node of the linked list
remove(ele)	Removes the given element from the linked list
removeFirst()	Removes the first element from the linked list
removeLast()	Removes the last element from the linked list

```
import java.util.*;
class linked
{
	public static void main(String args[])
	{
		LinkedList<String>s=new LinkedList<String>();
		s.add("A");
		s.add("B");
		s.add("C");
		s.addLast("D");
		s.addFirst("E");
		System.out.println(s);
		s.remove("C");
		s.removeFirst();
		s.removeLast();
		System.out.println(s);
	}
}
```

Stack class in java.util package

- Java Collection framework provides a Stack class that models and implements a **Stack data structure**.
- The class is based on the basic principle of last-in-first-out. In addition to the basic push and pop operations, the class provides three more functions of empty, search, and peek.

push(ele)	Insert the element in the stack
pop()	Removes the last element inserted in the stack
search(ele)	Searches the given element in the stack
peek()	Returns the first element of the stack
empty()	Returns the Boolean value after checking whether the stack is empty or not

```
import java.util.*;
class stack
{
	public static void main(String args[])
	{
		Stack<Integer> s=new Stack<Integer>();
		s.push(10);
		s.push(11);
		s.push(12);
		s.push(13);
		s.push(14);
		System.out.println(s);
		s.pop();
		System.out.println(s);
		System.out.println("Stack empty :"+s.empty());
		System.out.println("Search :"+s.search(11));
		System.out.println("Peek: "+s.peek());
	}
}
```

Queue class in java.util package

- Java Collection framework provides a Queue class that models and implements a **Queue data structure**.
- The class is based on the basic principle of First-in-first-out.

add(ele)	It is used to insert the element in the queue
remove()	Removes the head element from the queue
poll()	Used to retrieve and removes the head of queue
peek()	Returns the head element of the stack

```
import java.util.*;
class que
{
	public static void main(String args[])
	{
		PriorityQueue<String> p=new PriorityQueue<String>();
		p.add("abc");
		p.add("xyz");
		p.add("aaa");

		System.out.println("Queue ele: "+p);
		p.remove();
		System.out.println("After removal: "+p);
		System.out.println("Poll: "+p.poll());
		System.out.println("After poll:" +p);
		System.out.println("Peek: "+p.peek());
	}
}
```

MCQ		
Sr No.	**Question**	**Answer**
1	**…….class allows to insert and remove the elements unlike array**	**Vector**
2	**…….is the collection which represents element in key value form?**	**Hashtable**
3	**Which Class works on LIFO?**	**Stack**
4	**To create the queue, ……..is used**	**Priority Queue**

Sorted Set class in java.util package

- It is used to provide particular ordering on its element.
- The elements are ordered either by using natural ordering or by using comparator.

comparator()	Order the elements in given set
first()	Returns the first element of sorted set
last()	Returns the last element of sorted set
add()	add the elements in the sorted set
remove()	Removes or delete the element of sorted set

```
import java.util.*;
class que
{
        public static void main(String args[])
        {
                SortedSet s=new Treeset();
                s.add("abc");
                s.add("aaa");
                s.add("xyz");
                s.add("yyy");
                s.add("bbb");
                System.out.println(s);
                s.remove("aaa");
                System.out.println(s.first());
                System.out.println(s.last());
        }
}
```

Map class in java.util package

- It contains the values on the basis of the key that is key and value pair
- Each key and value pair is known as entry
- A map contains unique key

put()	add the elements in the Map
remove()	Removes or delete the element of Map

```
import java.util.*;
class que
{
        public static void main(String args[])
        {
                HashMap m=new HashMap();
                m.put(1,"abc");
                m.put(2,"aaa");
                System.out.println(m);
                m.remove(2);
                System.out.println(m);
        }
}
```

java.applet package

- This package provides classes needed to create applets and communicate with their applet.

java.awt package, java.awt.event

- AWT (Abstract Window Toolkit) package contains the classes for creating user interfaces and graphics. It also includes classes for painting images
- java.awt and jawa.awt.event packages are used for different events related to graphics and user interfaces

java.io package

- This package provides different classes for system input and output through data source, serialization and file system
- Different classes of java.io package are: BufferedReader, BufferedWriter, StreamReader, StreamWriter etc

java.net package

- This package provides the classes for implementing networking applications.
- It includes various classes that provide easy way to access network resource

java.swing package

- This package provides different swing components
- Swing is JFC (Java Foundation Class) and extension of AWT (Abstract Window Toolkit) and components of swing are lightweight components.

Q-8 what is User define package? How to create the package

Ans:

- ❖ The package which is created by the user is known as User Define Package.
- ❖ Following are the steps to create the package

Step 1: Create the notepad file

```
package mypackage;
public class Demo
{
        public void display()
        {
                System.out.println("hi");
        }
}
```

Step 2: Create the folder named “mypackage” and save the above file with Demo.java

E:\java\mypackage\Demo.java

Step 3: Create the file and import the created package

```
import mypackage.*;
public class sample
{
        public static void main(String args[])
        {
        Demo d=new Demo();
        d.display();
        }
`
```

Step 4: Save the file in the root directory outside the “mypackage” folder as sample.java

E:\java\mypackage\sample.java

Step 5: Compile Package (E\java\mypackage\javac Demo.java)

Step 6: Compile the file in which you imported package (E:\java\sample.java)

Step 7: Run the File in which you imported package (E:\java\java sample)

JAVA UNIT-3 MATERIAL

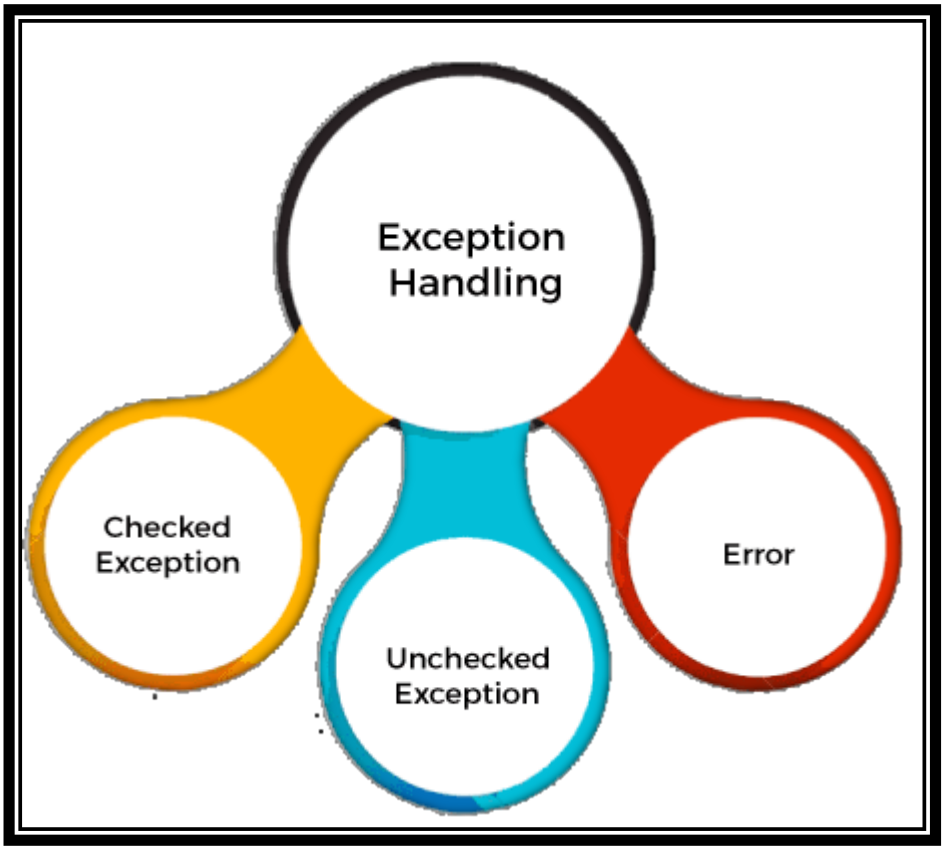
Ch. No.	Topic	Details	Marks
3	Exception Handling, Threading and Streams (Input and Output)	• Introduction to exception handling (try,catch, finally, throw, throws) • Creating user defined exception class • Thread and its life cycle (Thread States) • Thread class and its methods • Synchronization in multiple threads (Multithreading) • Deamon thread • Non-Daemon thread • Stream and its types (Input, Output, Character, Byte) • File and Random access file • Reading and writing through Character Stream classes (File Reader, Buffered Reader, File Writer, Buffered Writer) • Reading and writing through byte stream class (Inputstream, FileInputStream, DataInputStream, OutputStream, FileOutputStream, DataOutputStream) • StreamTokenizer Class • Piped Streams, Bridge Classes: InputStreamReader, OutputStreamWriter • ObjectInputStream, ObjectOutputStream	14

Q-1 what is Exception Handling?

Ans:

- The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.
- The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions.

Types of Java Exceptions



1) Checked Exception:

- The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) UnChecked Exception:

- The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error:

- Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

Java Exception Keywords

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

```
class exc
{
	public static void main(String args[])
	{
		try
		{
			int a=5/0;
		}catch(ArithmeticException e)
		{
			System.out.println(e);
		}
		finally
		{
			System.out.println("hi");
		}
	}
}
```

** Difference between checked and unchecked exception

Checked Exception	Unchecked Exception
1) Checked Exceptions are declared at compile time	1) Unchecked Exceptions are declared at runtime
2) This class inherits Throwable Class except Rutime Exception	2) This class inherits Runtime Exception class
3) Example: IO Exception	3) Example: Arithmetic Exception

** Difference between throw and throws in java

throw	throws
1) throw keyword is used to throw the exception explicitly in the code, inside the function or the block of code.	1) throws keyword is used to declare the exception in the method signature
2) It is followed by an instance of the exception to be thrown	2) It is followed by the class njgame of exception to be thrown
3) Only 1 exception can be thrown at a time	3) We can declare multiple exceptions using throws keyword.

MCQ	
1) Which is the superclass of all the errors and exceptions?	Throwable
2) Which keyword is used to write the code that contains error or exception?	try
3) Which keyword is used to write the code that must be compulsory executed?	finally
4) ……Exception is declared at compile time	Checked Exception

Q-2 what is User Defined Exception? How to create it

Ans:

- Creating our own Exception is known as custom exception or user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user need.
- This can be done by extending the class Exception

```
class userdefinedexception
{
	public static void main(String args[])
	{
		try
		{
			throw new myexception(400);
		}
		catch(myexception e)
		{
			System.out.println(e);
		}
	}
}

class myexception extends Exception
{
	int n1;
	myexception(int n2)
	{
		n1=n2;
	}
	public String toString()
	{
		return ("Status: " +n1);
	}
}
```

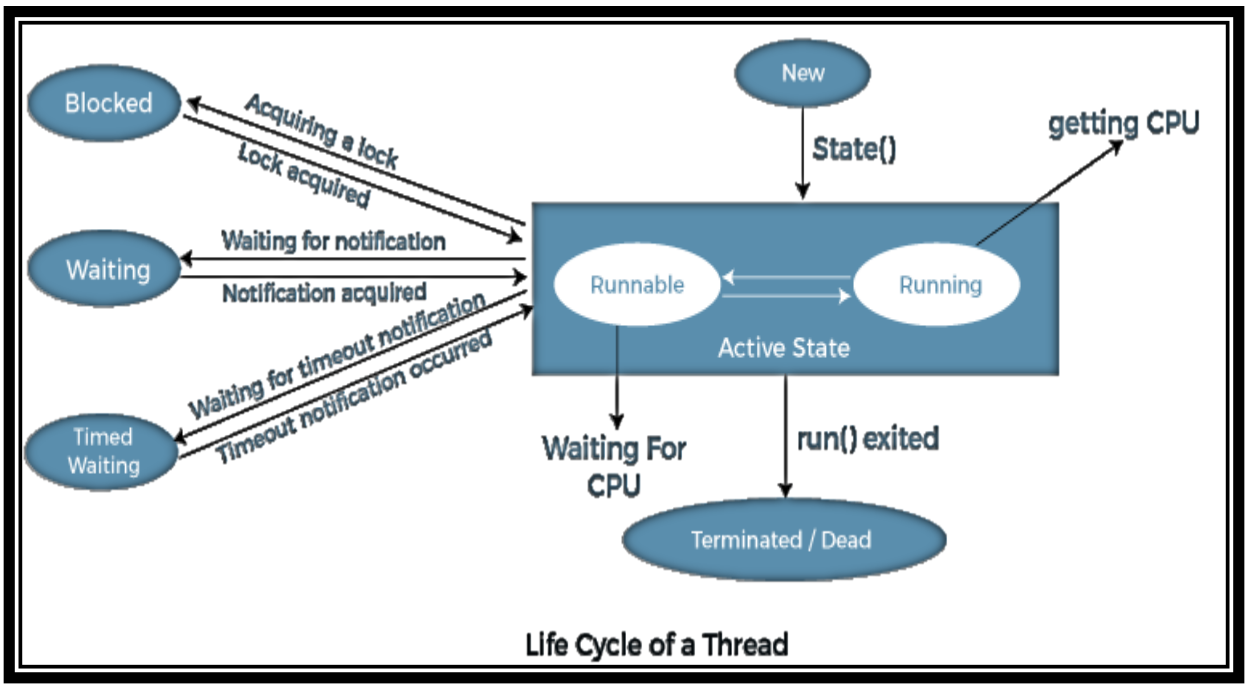
MCQ	
1) Creating the customized exception is known as ……..	User defined exception
2) User defined exception should extend ……Class	Exception

Q-3 what is thread? Explain life cycle of thread

Ans:

- A **thread** is a single sequential flow of control within a program. A process can have multiple **threads**, all executing at the same time.
- There can be more than one thread inside a process. Each thread of the same process makes use of a separate program counter
- In Java, a thread always exists in any one of the following states. These states are:

1. New
2. Active
3. Blocked / Waiting
4. Timed Waiting
5. Terminated



1) New:
   - ❖ Whenever a new thread is created, it is always in the new state. For a thread in the new state, the code has not been run yet and thus has not begun its execution.

2) Active:
   - ❖ When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is **runnable**, and the other is **running**.

Runnable:

- ❖ A thread that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time.
- ❖ It is the duty of the thread scheduler to provide the thread time to run, i.e., moving the thread the running state.

Running:

- ❖ When the thread gets the CPU, it moves from the runnable to the running state

3) Blocked/Waiting:
   - ❖ Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state.
   - ❖ For example, a thread (let's say its name is A) may want to print some data from the printer. However, at the same time, the other thread (let's say its name is B) is using the printer to print some data.
   - ❖ Therefore, thread A has to wait for thread B to use the printer. Thus, thread A is in the blocked state.

4) Timed Waiting:
   - ❖ Sometimes, waiting for leads to starvation. For example, a thread (its name is A) has entered the critical section of a code and is not willing to leave that critical section.
   - ❖ In such a scenario, another thread (its name is B) has to wait forever, which leads to starvation. To avoid such scenario, a timed waiting state is given to thread B.
   - ❖ Thus, thread lies in the waiting state for a specific span of time, and not forever.
   - ❖ A real example of timed waiting is when we invoke the sleep() method on a specific thread. The sleep() method puts the thread in the timed wait state.
   - ❖ After the time runs out, the thread wakes up and start its execution from when it has left earlier.

5) Terminated:

- A thread reaches the termination state because of the following reasons:
- When a thread has finished its job, then it exists or terminates normally.
- **Abnormal termination:** It occurs when some unusual events such as an unhandled exception or segmentation fault.

MCQ	
1) ……..is a sequential flow of the control within the program	Thread
2) There can be more than one thread inside the process (T/F)?	True
3) The active state of thread contains ……. and ……..states	Runnable, Running
4) Which process occurs when some unusual events such as an unhandled exception or segmentation fault?	Abnormal Termination

Q-4) what is Thread Class? Explain its methods.

Ans:

- Thread can be created using following methods:
  - Extending Thread Class
  - Implementing Runnable interface
  - Explicitly creating thread object using Thread Class

1) Extending Thread Class:

- Thread class provide constructors and methods to create and perform operations on a thread.

```
class m extends Thread
{
        public void run()
        {
                System.out.println("Running");
        }
        public static void main(String args[])
        {
                m m1=new m();
                m1.start();
        }
}
```

2) **Implementing Runnable Interface:**

- ❖ Create Runnable implementer which implements run()
- ❖ Create the object of Thread Class
- ❖ Pass the instance of class in Thread class
- ❖ Runnable interface have only one method named run().

```
class m extends implements Runnable
{
        public void run()
        {
                System.out.println("Running");
        }
        public static void main(String args[])
        {
                m m1=new m();
                Thread t1=new Thread(m1);
                t1.start();
        }
}
```

3) **Explicitly creating thread object using Thread Class:**

- ❖ If we don't extend the Thread class, then object of thread class is created explicitly.

```
class m extends
{
        public static void main(String args[])
        {
                Thread t1=new Thread("First Thread");
                t1.start();
                System.out.println("Thread Name: "+t1.getName());
        }
}
```

Methods of Thread Class:

Methods	Description
void run()	**Used to perform action for thread**
void start()	**Starts the execution of thread. JVM calls run() on thread**
public String getName()	**It is used to get the name of thread**
public void setName(String)	**It is used to change or set the name of the given thread.**
public int getPriority()	**It is used to get the priority of the thread**
public int setPriority()	**It is used to set the priority of the thread**
public void suspend()	**It is used to suspend the given thread**
public void resume()	**It is used to resume the suspended thread**
public void sleep(long milliseconds)	**Waits for a thread to die for the specified milliseconds**
public void yield()	**Causes currently executing thread object to temporarily pause and allow other threads to execute.**
public Boolean isDaemon()	**Tests if the thread is daemon thread or not**

```
class th
{

        public static void main(String args[])
        {
                Thread t1=new Thread("First Thread");
                System.out.println("Thread Priority: "+t1.getPriority());
                t1.setPriority(6);
                t1.start();
                t1.setName("My Thread");
                System.out.println("Thread Name: "+t1.getName());
                System.out.println("Thread Priority: "+t1.getPriority());
                System.out.println("Daemon Thread: "+t1.isDaemon());

        }
}
```

```
class th1 extends Thread
{
        public void run()
        {
                for(int i=1;i<=5;i++)
                {
                        try
                        {
                                Thread.sleep(1000);
                        }
                        catch(InterruptedException e)
                        {
                                System.out.println(e);
                        }
                        System.out.println(i);
                }
        }

        public static void main(String args[])
        {
                th1 t=new th1();
                t.start();


        }
}
```

MCQ	
1) Thread can be created by implementing ……interface	Runnable
2) Which method of thread class moves the thread from new state to active state?	start()
3) Which method of thread class is called on the start () of thread?	run()
4) Runnable interface have only 1 method. Which is that?	run()
5) Which method causes currently executing thread object to	yield()

temporarily pause and allow other threads to execute	

Q-5) Write a short note on Thread Synchronization? OR

Write a short note on Multithreading

Ans:

- Synchronization in Java is the capability to control the access of multiple threads to any shared resource.
- Java Synchronization is better option where we want to allow only one thread to access the shared resource.
- Synchronization can be achieved by:
  - Using Synchronized method
  - Using Synchronized block
  - Using Static Synchronization

Why use Synchronization?

- The synchronization is mainly used to

  1. To prevent thread interference.
  2. To prevent consistency problem.

- Below example shows the scenario where synchronization is not there. In the below example, thread t1 and thread t2 are not synchronized so thread t1 prints table of 5 and simultaneously thread t2 prints table of 10. So in this case, synchronization is required

```
class table
{
	void printtable(int n)
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println(n*i);
			try
			{
				Thread.sleep(500);
			}
			catch(InterruptedException e)
			{
			System.out.println(e);
			}
		}
	}
}

class mythread1 extends Thread
{
	table t;
	mythread1(table t)
	{
		this.t=t;
	}
	public void run()
	{
		t.printtable(5);
	}
}
class mythread2 extends Thread
{
	table t;
	mythread2(table t)
	{
		this.t=t;
	}
	public void run()
	{
		t.printtable(10);
	}
}
```

```
class synch
{
	public static void main(String args[])
	{
		table o=new table();
		mythread1 t1=new mythread1(o);
		mythread2 t2=new mythread2(o);
		t1.start();
		t2.start();
	}
}
```

Example with synchronization:

```
class table
{
	synchronized void printtable(int n)
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println(n*i);
			try
			{
				Thread.sleep(500);
			}
			catch(InterruptedException e)
			{
			System.out.println(e);
			}
		}
	}
}

class mythread1 extends Thread
{
	table t;
	mythread1(table t)
	{
		this.t=t;
	}
	public void run()
	{
		t.printtable(5);
	}
}
```

```
class mythread2 extends Thread
{
        table t;
        mythread2(table t)
        {
                this.t=t;
        }
        public void run()
        {
                t.printtable(5);
        }
}


class synch
{
        public static void main(String args[])
        {
                table o=new table();
                mythread1 t1=new mythread1(o);
                mythread2 t2=new mythread2(o);
                t1.start();
                t2.start();
        }
}
```

MCQ	
1) ……..is the capability to control the access of multiple threads to any shared resource.	Synchronization
2) What are the advantages of Synchronization?	1) To reduce conflicts among threads 2) To avoid interference among threads

Q-6) Write a short note on Daemon and Non-Daemon Thread

Ans:

Daemon Thread:

- Damon thread in java is a low-priority thread that runs in the background to perform tasks such as garbage collection.
- Daemon thread in java is also a service provider thread that provides services to the user thread.

Non-Daemon Thread:

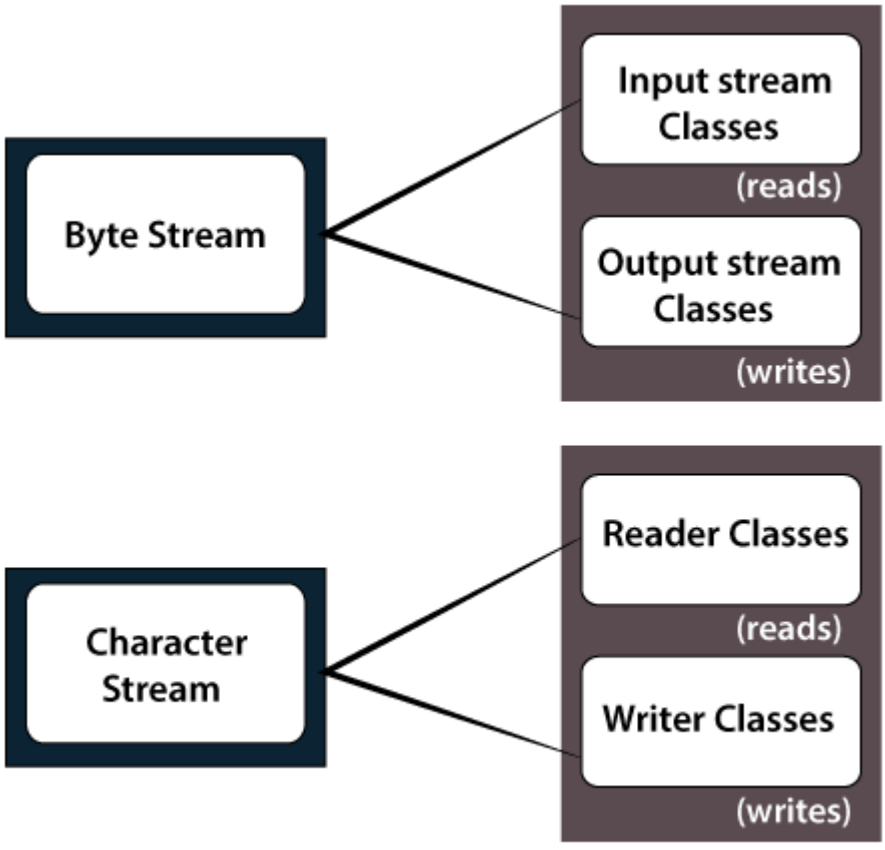
- Non-Damon thread in java is a thread that does not run in the background.
- Non-daemon thread in java are not service provider

Daemon Threads	User Threads (Non-daemon)
Daemon threads are created by JVM	User threads are created by an application itself
JVM does not wait for its execution	JVM waits until the execution completes
Low Priority threads	High priority threads
Used for background tasks(not critical)	Used for foreground tasks(critical)
Life is dependent on user threads	Life is independent

Q-7) what is Stream? Explain types

Ans:

- A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream
- Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.
- In general, a Stream will be an input stream or, an output stream.
    - **InputStream** − this is used to read data from a source.
    - **OutputStream** − this is used to write data to a destination.
- Based on the data they handle there are two types of streams −
    - **Byte Streams** − these handle data in bytes (8 bits) i.e., the byte stream classes read/write data of 8 bits. Using these you can store characters, videos, audios, images etc.
    - **Character Streams** − these handle data in 16 bit Unicode. Using these you can read and write text data only.



Brief classification of I/O streams

**** Difference between byte stream and character stream:**

Byte Stream	Character Stream
Java Byte streams are used to perform input and output of 8-bit bytes	Java Character streams are used to perform input and output for 16-bit Unicode.
Byte stream are not tied to any data type. Data of any data type can be read	Character stream are tied to data type. Only string or character data type can be read.
Byte stream reads byte by byte	Character stream reads character by character
Byte stream are known as data streams-Data input stream and Data Output Stream	Character streams are also known as reader and writer streams.

MCQ	
1) …….is sequence of data	**Stream**
2) Which package contains all the class for input and output operations?	**java.io package**
3) Which streams are used to perform input and output for 8-bit bytes	**Byte Stream**
4) Which stream are tied to data type?	**Character Stream**

Q-8) what is File Class?

Ans:

- The File class is an abstract representation of file and directory pathname.
- A pathname can be either absolute or relative.
- The File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc.

Example:

```
import java.io.*;
class filedemo
{
        public static void main(String args[])
        {
                File f=new File("d:\\th123.java");
                if(f.isFile())
                        System.out.println("\n File Exists");
                else
                        System.out.println("\n File does not exists");
        }
}
```

Creating a new file:

- **Create a File** operation is performed to create a new file.
- We use the **createNewFile()** method of file.
- The **createNewFile()** method returns true when it successfully creates a new file and returns false when the file already exists.

Example:

```
import java.io.*;
import java.io.IOException;

class crf
{
	public static void main(String args[])
	{
		try
		{
		File f=new File("D:\\hel.txt");
		if(f.createNewFile())
		System.out.println("Created");
		else
		System.out.println("Not Created");
		}
		catch(IOException e)
		{
			System.out.println(e);
		}
	}

}
```

Getting the file information:

- The operation is performed to get the file information. We use several methods to get the information about the file like name, absolute path, is readable, is writable and length.

Example:

```
import java.io.*;
import java.io.IOException;

class crf
{
	public static void main(String args[])
	{

		File f=new File("D:\\hel.txt");
		if(f.exists())
		{
			System.out.println("Name of file: "+f.getName());
			System.out.println("Path of file: "+f.getAbsolutePath());
			System.out.println("Path of file: "+f.length());
		}
		else
		{
			System.out.println("Does not exists");
		}

	}

}
```

Write into file:

- In order to write data into a file, we will use the **FileWriter** class and its **write()** method together. We need to close the stream using the **close()** method to retrieve the allocated resources.

Example:

```
import java.io.*;
import java.io.IOException;

class crf
{
	public static void main(String args[])
	{
		try
		{
		FileWriter w=new FileWriter("d:\\hel.txt");
		w.write("Hello");
		w.close();
		}
		catch(IOException e)
		{
			System.out.println(e);
		}


	}

}
```

Read from the file:

- In order to read the data from file, we will use the **Scanner** class. Here, we need to close the stream using the **close()** method.
- We will create an instance of the Scanner class and use the **hasNextLine()** method **nextLine()** method to get data from the file.

Example:

```
import java.io.*;
import java.io.FileNotFoundException;
import java.util.*;

class crf
{
	public static void main(String args[])
	{
		try
		{
		File f=new File("D:\\hel.txt");
		Scanner s=new Scanner(f);
		while(s.hasNextLine())
		{
			String fd=s.nextLine();
			System.out.println(fd);
		}
		s.close();
		}
		catch(FileNotFoundException e)
		{
			System.out.println(e);
		}

}
```

Delete from the file:

- In order to delete a file, we will use the **delete()** method of the file. We don't need to close the stream using the **close()** method because for deleting a file

Example:

```
import java.io.*;

class del
{
	public static void main(String args[])
	{
		File f=new File("D:\\hel.txt");
		if(f.delete())
			System.out.println("Deleted");
		else
			System.out.println("Not Deleted");
	}
}
```

MCQ	
1) Which method is used to create the file?	createNewFile()
2) createNewFile() must be used with ……..exception	IOException
3) In order to write the data into file, which class is used?	FileWriter
4) Which class is used to read the data from the file?	Scanner
5) hasnextLine() and nextLine() are the methods of ……..class	Scanner
6) …….method is used to delete the file	delete()

Q-9) what is Random Access File Class?

Ans:

- A sequential file structure is the common way where records are stored in order by the record key field.
- Random file access is the superimposed mechanism implemented through Java code to access individual records directly and quickly without searching through records in the file
- The RandomAccessFile class allows you to write programs that can seek to any location in a file and read or write data at that point.
- This type of functionality is very valuable in some programs.

Method	**Description**
Void close()	Close the file.
Long getFilePointer()	Returns the current position of the file pointer. This identifies the point at which the next byte is read or written.
Long length()	Returns the number of bytes in the file.
Int read()	Reads and returns a byte from the file. Waits until data is available.
Int read(byte buffer[], int index, int size)	Attempts to read size bytes from the file and places these in buffer starting at position index. Returns the number of bytes actually read. Waits until data is available.
Int read(byte buffer[])	Reads bytes form the file and places these in buffer. Returns the number of bytes read. Waits until data is available.
Void seek(long n)	Positions the file pointer at n bytes form the beginning of the file. The next read or write occurs at this position.
Int skipBytes(int n)	Adds n to the file pointer. Returns the actual number of bytes skipped. If n is negative, no bytes are skipped.

Example

```
import java.io.*;
class Random1
{
        public static void main(String args[])
        {
        try
        {
        RandomAccessFile f=new RandomAccessFile("test1.txt","rw");
        f.writeChar('k');
        f.writeInt(10);
        f.writeDouble(10.2);
        f.seek(0);

        System.out.println(f.readChar());
        System.out.println(f.readInt());
        System.out.println(f.readDouble());
        f.close();
        }

        catch(IOException e)
        {
                System.out.println(e);
        }
}
}
```

MCQ	
1) Random Access is used to perform ……and ……..operations both simultaneously	read,write
2) In random access file, the file can be moved to any location using ……..method	seek()

Q-10) Write a short note on character stream class

Ans:

- The java.io package provides character stream classes to overcome the limitations of byte stream classes which can only handle 8 bit and is not compatible to work directly with characters.
- Character stream classes are used to work with 16 -bit.
- Generally, character stream classes are used to read the characters from the source and write them into destination.

Classes of Character Stream	
Class	**Description**
Buffered Reader	**This class provides the methods to read characters from buffer**
FileReader	**This class provides the methods to read characters from the file**
BufferedWriter	**This class provides the methods to write the characters to the buffer**
FileWriter	**This class provides the methods to write the characters to the file.**

File Reader Class and File Writer Class:

- Java FileWriter and FileReader classes are used to write and read data from text files
- Unlike FileOutputStream class, you don't need to convert string into byte array because it provides method to write string directly.

Methods of File Reader and File Writer	
Methods	**Description**
int read()	**This method returns the integral representation of the next character present in the input.**
void write()	**This method is used to write the data to the output stream**
void flush()	**This method is used to flush the output stream**

Example of File Reader and File Writer

```
import java.io.*;
import java.io.IOException;

class ch1
{
	public static void main(String args[])
	{
		try
		{
			FileWriter f=new FileWriter("D:\\myf6.txt");
			f.write("Hello");
			f.close();
			FileReader f=new FileReader("D:\\myf6.txt");
			int i;
			while((i=f.read())!=-1)
			System.out.println((char)i);
			f.close();

		}
		catch(IOException e)
		{
			System.out.println(e);
		}
	}
}
```

Buffered Reader Class and Buffered Writer Class:

- The **"BufferedWriter"** class of java supports writing a chain of characters output stream (Text based) in an efficient way. The Chain-Of-Characters can be Arrays, Strings etc. The **"BufferedReader"** class is used to read stream of text from a character based input stream.
- The BufferedReader and BufferedWriter class provides support for writing and reading

Example of Buffered Reader and Buffered Writer:

```
import java.io.*;
import java.io.IOException;

class ch1
{
	public static void main(String args[])
	{
		try
		{
			FileWriter f=new FileWriter("D:\\myf7.txt");
			BufferedWriter b=new BufferedWriter(f);
			f.write("Hello");
			f.close();
			FileReader f=new FileReader("D:\\myf7.txt");
			BufferedReader b=new BufferedReader(f);
			int i;
			while(i=f.read())!=-1)
			{
				System.out.println((char)i);
			}
			b.close();


		}
		catch(IOException e)
		{
			System.out.println(e);
		}
	}
}
```

MCQ	
1) Character Stream class works on …….bits	**16**
2) BufferedReader inherits from ……	**FileReader**

Q-11) Write a short note on Byte stream class

Ans:

- Byte Stream classes are used to read bytes from the input stream and write bytes to the output stream. In other words, we can say that Byte Stream classes read/write the 8-bits.
- The Byte Stream classes are divided into two types of classes, i.e., Input Stream and Output Stream.

InputStreamClass:

- The Input Stream class provides methods to read bytes from a file, console or memory. It is an abstract class and can't be instantiated.
- The subclass of Input Stream class are:

SN	Class	Description
1	BufferedInputStream	This class provides methods to read bytes from the buffer.
2	ByteArrayInputStream	This class provides methods to read bytes from the byte array.
3	DataInputStream	This class provides methods to read Java primitive data types.
4	FileInputStream	This class provides methods to read bytes from a file.
5	FilterInputStream	This class contains methods to read bytes from the other input streams, which are used as the primary source of data.
6	ObjectInputStream	This class provides methods to read objects.

7	PipedInputStream	This class provides methods to read from a piped output stream to which the piped input stream must be connected.

Methods of File InputStream	
Methods	**Description**
int read()	It is s used to read the byte from the current stream
void flush()	**This method is used to flush the input stream**
void close()	**It is used to close the input stream**

OutputStreamClass:

- The **Java.io.OutputStream** class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

SN	Class	Description
1	BufferedOutputStream	This class provides methods to write bytes in the buffer.
2	ByteArrayOutputStream	This class provides methods to write bytes from the byte array.
3	DataOutputStream	This class provides methods to write Java primitive data types.
4	FileOutputStream	This class provides methods to write bytes in a file.
5	FilterOutputStream	This class contains methods to write bytes in the other input streams, which are used as the primary source of data.
6	ObjectOutputStream	This class provides methods to write objects.

7	PipedOutputStream	This class provides methods to write in a piped output stream to which the piped input stream must be connected.

Methods of File OutputStream	
Methods	**Description**
int write()	It is s used to write the byte in the current stream
void flush()	**This method is used to flush the output stream**
void close()	**It is used to close the output stream**

FileInputStreamClass

- Java FileInputStream class obtains input bytes from a file.
- It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc.

Methods of File Input Stream	
Methods	**Description**
int read()	It is s used to read the byte from the current stream
void flush()	**This method is used to flush the input stream**
void close()	**It is used to close the input stream**

FileOutputStreamClass

- FileOutputStream is an output stream used for writing data to a file.
- If you have to write primitive values into a file, use FileOutputStream class.
- You can write byte-oriented as well as character-oriented data through FileOutputStream class.

Methods of File Output Stream	
Methods	**Description**
int write()	It is s used to write the byte in the current stream
void flush()	**This method is used to flush the output stream**
void close()	**It is used to close the output stream**

Example of FileOutputStream and FileInputStream:

```
import java.io.*;

class inp
{
	public static void main(String args[])
	{
		try
		{
		FileOutputStream f=new FileOutputStream("D:\\myf9.txt");
		f.write(10);
		FileInputStream f1=new FileInputStream("D:\\myf9.txt");
		int i;
		while((i=f1.read())!=-1)
		{
			System.out.println(i);
		}
		}
		catch(IOException e)
		{
			System.out.println(e);
		}
	}
}
```

DataInputStreamClass and DataOutputStream Class:

- The DataInputStream class read primitive Java data types from an underlying input stream in a machine-independent way. While the DataOutputStream class write primitive Java data types to an output stream in a portable way.

Example of DataInputStream and DataOutputStream:

```
import java.io.*;
class da
{
	public static void main(String args[])throws IOException
	{
		FileOutputStream f=new FileOutputStream("D:\\myf10.txt");
		DataOutputStream f1=new DataOutputStream(f);
		f1.write(10);
		FileInputStream f2=new FileInputStream("D:\\myf10.txt");
		DataInputStream f3=new DataInputStream(f2);
		int i;
		while((i=f3.read())!=-1)
		{
			System.out.println(i);
		}

		f1.close();
		f3.close();
	}
}
```

Program: Write a program to copy the data of one file into another file

```
import java.io.*;
class cop
{
        public static void main(String args[])throws IOException
        {
                FileInputStream f=new FileInputStream("D:\\myf1.txt");
                FileOutputStream f1=new FileOutputStream("D:\\copy.txt");
                int i;
                while((i=f.read())!=-1)
                {
                        f1.write((char)i);
                }
                f.close();
                f1.close();
        }
}
```

MCQ	
1) Byte stream class works on ……bits	**8 ibts**
2) Which subclass of byte stream class is used to read and write the primitive data types	**DataInputStream and DataOutputStream**

Q-12) Write a short note on Stream Tokenizer Class

Ans:

- The **Java.io.StreamTokenizer** class takes an input stream and parses it into "tokens", allowing the tokens to be read one at a time.
- The stream tokenizer can recognize identifiers, numbers, quoted strings, and various comment styles.
- For StreamTokenizer, the source is a character stream, Reader.
- Following constant variables used to decide the type of the token:

int ttype	**When the nextToken() returns a token, this field can be used to decide the type of the token.**
int TT_EOF	**This field is used to know the end of file is reached.**
int TT_EOL	**This field is used to know the end of line is reached.**
int TT_NUMBER	**This field is used to decide the token returned by the nextToken() method is a number or not.**
int TT_WORD	**This field is used to decide the token returned by the nextToken() method is a word or not.**
String sval	**If the token is a word, this filed contains the word that can be used in programming.**
double nval	**If the token is a word, this filed contains the number that can be used in programming.**

```
import java.io.*;

class stto
{
	public static void main(String args[])throws IOException
	{
		FileReader r=new FileReader("D:\\myf1.txt");
		StreamTokenizer st=new StreamTokenizer(r);
		double sum=0;
		int n=0;
		while(st.nextToken()!=st.TT_EOF)
		{
			if(st.ttype==StreamTokenizer.TT_NUMBER)
			sum=sum+st.nval;
			else if(st.ttype==StreamTokenizer.TT_WORD)
			n++;
		}
		System.out.println("Sum:" +sum);
		System.out.println("Total Words: "+n);
	}
}
```

MCQ	
1) Which class takes an input stream and parses it in to tokens?	**Stream Tokenizer**
2) Which constant is used to indicate type of token?	**ttype**
3) TT_EOF indicates …….	**End of File**
4) Which constant is used to check whether the returned token is number or not?	**TT_NUMBER**
5) Which constant is used to check whether the returned token is word or not?	**TT_WORD**

Q-13) Write a short note on Piped Input and Output Stream

Ans:

- The PipedInputStream and PipedOutputStream classes can be used to read and write data simultaneously.
- Both streams are connected with each other using the connect() method of the PipedOutputStream class.

```
import java.io.*;
class pipe
{
        public static void main(String args[])
        {
                PipedOutputStream out=new PipedOutputStream();
                PipedInputStream in=new PipedInputStream();
                try
                {
                        in.connect(out);
                        out.write(23);
                        out.write(24);
                        for(int i=0;i<2;i++)

                        System.out.println(in.read());
                }
                catch(IOException e)
                {
                        System.out.println(e);
                }
}
```

MCQ	
1) Which method is used to connect piped input stream with piped output stream	**connect()**
2) What is main Feature of piped stream?	**Allows to read and write the data simultaneously**

Q-14) Write a short note on Bridge Class

Ans:

InputStreamReader:

- An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified charset.

```
import java.io.*;
class br
{
	public static void main(String args[])throws IOException
	{
		FileInputStream f=new FileInputStream("D:\\myf3.txt");
		InputStreamReader r=new InputStreamReader(f);
		int i;
		while((i=r.read())!=-1)
		{
			System.out.println((char)i);
		}
		r.close();
	}
}
```

OutputStreamWriter:

- OutputStreamWriter is a which is used to convert character stream to byte stream, the characters are encoded into byte using a specified charset. write() method calls the encoding converter which converts the character into bytes.

```
import java.io.*;
class br
{
        public static void main(String args[])throws IOException
        {
                FileOutputStream f=new
FileOutputStream("D:\\myf100.txt");
                OutputStreamWriter w=new OutputStreamWriter(f);
                w.write("hello world");
                w.close();
        }
}
}
```

ObjectInputStream and ObjectOutputStream:

- The objectinputstream class is mainly used to deserialize the primitive data and objects which are written by using ObjectOutputStream.

```
import java.io.*;
class br
{
        public static void main(String args[])throws IOException
        {
                int i=10;
                FileOutputStream f=new FileOutputStream("d:\\myfile101.txt");
                ObjectOutputStream o=new ObjectOutputStream(f);
                o.writeInt(i);


                FileInputStream f1=new FileInputStream("d:\\myfile101.txt");
                ObjectInputStream o1=new ObjectInputStream(f1);
                System.out.println("Integer:" +o1.readInt());
;
                f.close();
                f1.close();
        }
```

MCQ

1) ……..class is used to convert byte stream to character stream class?	**InputStreamReader**
2) Which class is used to convert character stream to byte stream class?	**OutputStreamWriter**

JAVA UNIT-4 and UNIT-5 MATERIAL

Ch. No.	Topic	Details	Marks
4 and 5	JavaFx basics and Event Driven Programming and animations	• Basic Structure of JavaFx Program • Panes • UI Controls and Shapes • Color and Font Class • Image and Image-View Class • Layout panes and shapes • Events and Events Sources • Registering Handlers and Handling events • Inner Classes, anonymous inner class handlers • Mouse and Key Events • Listeners for observable objects • Animation	14

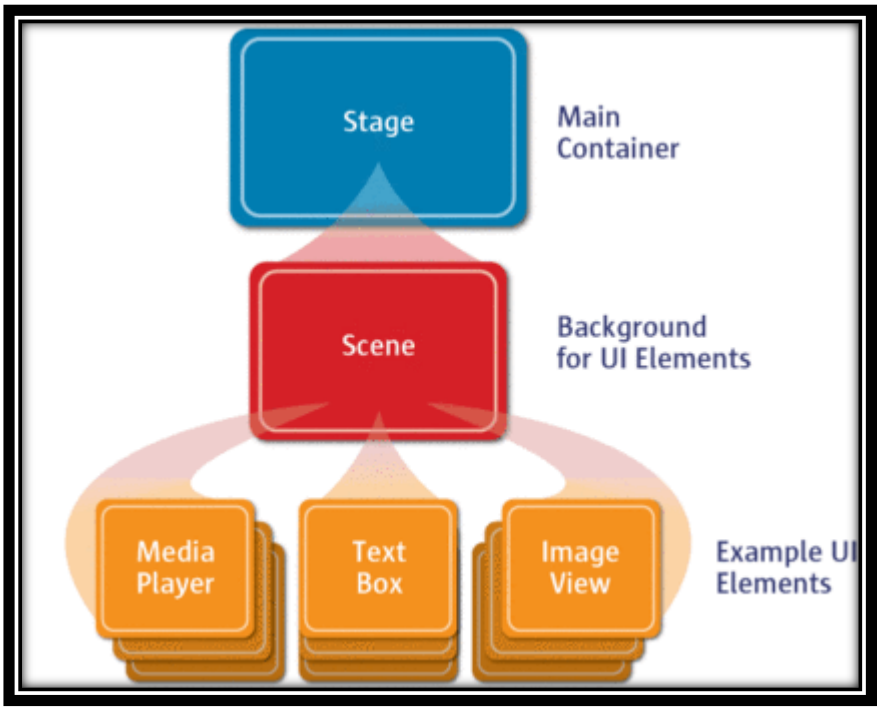
Q-1 what is JavaFx? Explain basic structure of JavaFx program

Ans:

- JavaFx is a java program which is a set of graphics and media packages that enables developers to design, create, test, debug and deploy rich client applications that operate consistently across different platforms.
- It is a powerful way for creating graphics applications in java

Basic Structure of JavaFx Program:

-



- JavaFx have 3 major components namely Stage, Scene and Node
- Each and every javaFx application must extend javaFx.application.Application package

1) Stage:

- It contains all the objects of javafx application. It is represented by Stage class of javafx.stage package.
- The primary stage is created by the platform itself. The created Stage object is passed to the start() as arguments.
- Start() is the method of application class
- Stage is divided in to content area and border
- Show() is called to display the content of the page

2) **Scene:**

- It represents the physical content of javaFx application
- It contains all the contents of Scene graph
- It is represented by Scene class of javaFx.scene package
- At one time, Scene object is added to only one stage.
- Scene can be created by instantiating Scene class

3) **Node:**

- Node contains the UI controls such as button, textbox etc

Different Methods:

1) start() : It is the entry point method of javaFx application where all code is written
2) init(): It is empty method that can be overridden. In this method, user can not create a stage or scene
3) stop(): It is also an empty method that can be overridden just like init(). In this method, user can write the code to halt the application
4) launch(): JavaFx implements a static method known as launch() which is used to launch the javaFx application. As launch is static, the user should call it from static method only. Generally, static method which calls launch() is main()

Example:

```
import javaFx.application.Application;
import javaFx.scene.Scene;
import javafx.stage.Stage;
import javafx.Scene.Control.Button;

public class A extends Application
{
        public void start(Stage ps)
        {
                Button b=new Button("Ok");
                Scene s=new Scene(b,200,250);
                ps.setTitle("My first prog");
                ps.setScene(s);
                ps.show();
        }
}
```

Q-2 Explain panes in JavaFx

Ans:

- **A pane is a UI element ("Node")** that contains other UI elements ("child nodes") and manages the layout of those nodes within the Pane.
- **Following are the types of pane in JavaFx class**
  - **Flow Pane**
  - **Stack Pane**
  - **Border Pane**
  - **Grid Pane**
  - **Tile Pane**
  - **Anchor Pane**

1) Flow Pane:

- Flow Pane lays all nodes one after another in the order they were added.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.Control.Button;
import javafx.scene.layout.FlowPane;

public class A extends Application
{
        public void start(Stage ps)
        {
                FlowPane p=new FlowPane();
                Button b1=new Button(“Ok”);
                Button b2=new Button(“Cancel”);
                p.getChildren.addAll(b2,b1);
                Scene s=new Scene(p,200,250);
                ps.setTitle(“Flow layout”);
                ps.setScene(s);
                ps.show();
}
```

2) Stack Pane:

- Stack Pane lays all nodes one on top of another

```
import javafx.application.Application;
import javaFx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.Control.Button;
import javafx.scene.layout.StackPane;

public class A extends Application
{
        public void start(Stage ps)
        {
                StackPane p=new StackPane();
                Button b1=new Button(“Ok”);
                Button b2=new Button(“Cancel”);
                p.getChildren.addAll(b2,b1);
                Scene s=new Scene(p,200,250);
                ps.setTitle(“Stack layout”);
                ps.setScene(s);
                ps.show();
}
```

3) Grid Pane:

- Grid Pane allows you to create flexible grid of rows and columns and position each node in exact place.

```
import java.application.Application;
import javaFx.scene.Scene;
import javafx.stage.Stage;
import javafx.Scene.Control.Button;
import javafx.scene.layout.GridPane;
import javafx.Scene.Control.Label;
import javafx.Scene.Control.TextField;


public class A extends Application
{
        public void start(Stage ps)
        {
                Lable l1=new Label(“Username”);
                Label l2=new Label(“Password”);
                TextField t1=new TextField();
                TextField t2=new TextField();
                Button b=new Button(“Submit”);
                GridPane g=new GridPane();
                g.addRow(0,l1,t1);
                g.addRow(1,l2,t2);
                g.addRow(2,b);
                Scene s=new Scene(g,200,250);
                ps.setTitle(“Grid layout”);
                ps.setScene(s);
                ps.show();
}
```

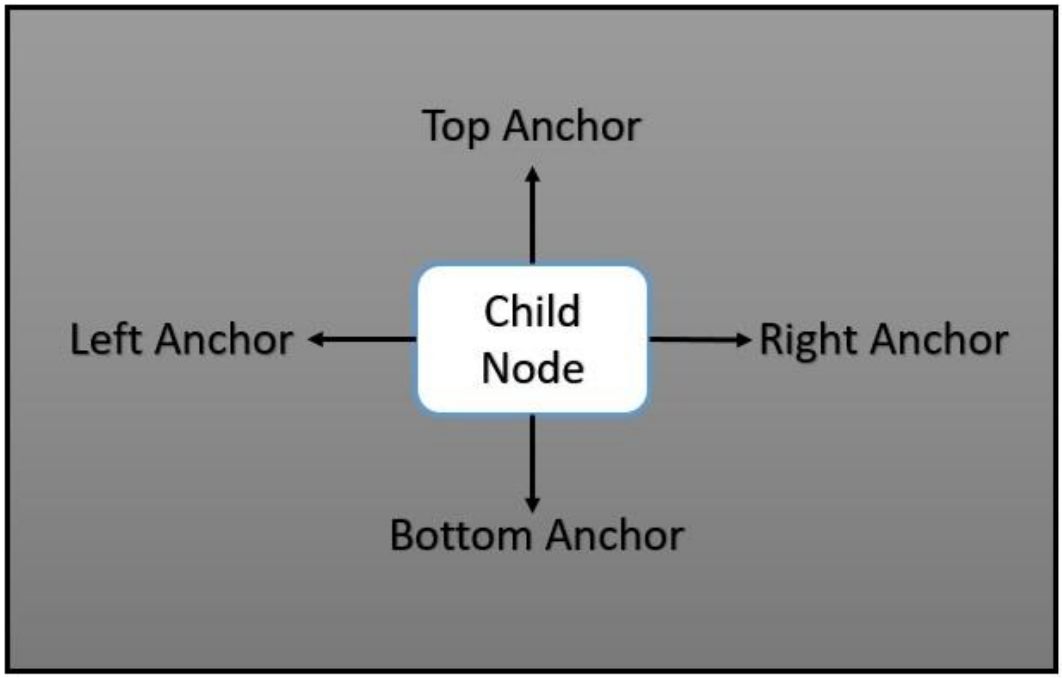
4) Border Pane:

- BorderPane splits the scene in five regions such as: top, bottom, left, right, and center. Where you can adjust added nodes. BorderPane also allows you to add different panes in each region

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.Scene.Control.Button;
import javafx.scene.layout.BorderPane;

public class A extends Application
{
        public void start(Stage ps)
        {
                Button b1=new Button(“Left Button”);
                Button b2=new Button(“Right Button”);
                Button b3=new Button(“Top Button”);
                Button b4=new Button(“Bottom Button”);
                Button b5=new Button(“Center Button”);
                BorderPane bp=new BorderPane();
                bp.setLeft(b1);
                bp.setRight(b2);
                bp.setTop(b3);
                bp.setBottom(b4);
                bp.setCenter(b5);
                Scene s1=new Scene(bp,200,250);
                ps.setScene(s1);
                ps.show();
        }
}
```

5) Anchor Pane:

- It is a layout component that allows the edges of child nodes to be anchored to an offset from edges of Anchor Pane



- Each child can have up to 4 anchors
    - Top Anchor
    - Bottom Anchor
    - Left Anchor
    - Right Anchor
- When you specify the constraints to one of the above mentioned 4 anchors, you specify the node and distance from pane to child node in pixel

Example:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.Scene.Control.Button;
import javafx.scene.layout.AnchorPane;

public class A extends Application
{
        public void start(Stage ps)
        {
                Button b=new Button(“ok”);
                TextArea t=new TextArea();
                AnchorPane a=new AnchorPane();
                a.getChildren().addAll(t,b);
                a.setTopAnchor(t,10.0);
                a.setBottomAnchor(t,10.0);
                a.setLeftAnchro(t,10.0);
                a.setRightAnchor(t,10.0);

                a.setLeftAnchor(b,20.0);
                a.setRightAnchor(b,20.0);
                a.setTopAnchor(b,20.0);
                a.setBottomAnchor(b,20.0);

                Scene s1=new Scene(a,200.250);
                ps.setScene(s1);
                ps.show();
                }
}
```

6) Tile Pane:

- It is a layout container which is so similar to Flow Pane
- It arranges the child components on a row and automatically pushes the components down to next line if the current row is filled up
- Tile Pane will arrange all its child nodes in same cell size while in Flow Pane the cell size differs depending upon button size.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.Scene.Control.Button;
import javafx.scene.layout.TilePane;

public class A extends Application
{
        public void start(Stage ps)
        {
                Button b1=new Button(“ok”);
                Button b2=new Button(“Cancel”);
                Button b3=new Button(“Retry”);
                Button b4=new Button(“hide”);
                Button b5=new button(“hhh”);

                b1.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);
                b2.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);
                b3.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);
                b4.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);
                b5.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);

                b2.setPrefSize(80,60);
                TilePane t=new TilePane();
                t.getChildren().addAll(b1,b2,b3,b4,b5);
                Scene s1=new Scene(t,200.250);
                ps.setScene(s1);
                ps.show();
                }
}
```

Border Pane	Anchor Pane
1) Anchor Pane allows the edges of child nodes to be anchored to an offset from the anchor pane's edges	1) Border Pane lays out children in top, left, right, bottom, and center positions.

Difference between Flow Pane and Tile Pane

Flow Pane	Tile Pane
1) The Tile Pane places all of the child nodes in a single row and pushes in the next line if row is completed but each cell have same size	1) The Tile Pane places all of the child nodes in a single row and pushes in the next line if row is completed but each cell have same size

Q-3 Explain UI controls and shapes

Ans:

UI Controls:

1) **Label:**

- A Label object is a component for placing text.

Example:

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.FlowPane;

public class A extends Application
{
        public void start(Stage s)
        {
                 Label l=new Label();
                FlowPane f=new FlowPane();
                f.getChildren().addAll(l);
                Scene s1=new Scene(f,200,250);
                s.setScene(s1);
                s.show();
        }
}
```

2) Labeled:

- Labeled is a more general class that Label extends. It is an abstract class that is the parent of Label and other controls like Button, CheckBox, and RadioButton. These controls can display text and/or graphics.
- While Label is a subclass of Labeled, other controls like Button also inherit from Labeled.

Label	Labeled
Label **is a specific subclass of Labeled designed for displaying text only.**	**Labeled** is an abstract class that provides a foundation for any control that can display both text and graphics (like Button, CheckBox, etc.).
Use Label **when you want to display a simple text element.**	Use **Labeled** when working with other controls like Button, CheckBox, etc., that can contain both text and graphics.

3) Button:

- This class creates a labeled button. This control is used to perform some action on its click event

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.FlowPane;
public class A extends Application
{
	public void start(Stage s)
	{
		Button b1=new Button("Ok");
		Button b2=new Button("Cancel");
		FlowPane f=new FlowPane();
		f.getChildren().addAll(b1,b2);
		Scene s1=new Scene(f,200,250);
		s.setScene(s1);
		s.show();
	}
}
```

4) TextField:

- A TextField object is a text component that allows for the editing of a single line of text.
- It is a control that allows the user to enter single line data

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.scene.layout.FlowPane;

public class A extends Application
{
        public void start(Stage s)
        {
                 TextField t=new TextField();
                FlowPane f=new FlowPane();
                f.getChildren().addAll(t);
                Scene s1=new Scene(f,200,250);
                s.setScene(s1);
                s.show();
        }
}
```

5) TextArea:

- A Text Area object is a text component that allows for the editing of a multi-line of text.
- It is a control that allows the user to enter multi- line data like address

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.TextArea;
import javafx.scene.layout.FlowPane;

public class A extends Application
{
        public void start(Stage s)
        {
                 TextField t=new TextArea();
                FlowPane f=new FlowPane();
                f.getChildren().addAll(t);
                Scene s1=new Scene(f,200,250);
                s.setScene(s1);
                s.show();
        }
}
```

6) **CheckBox:**

- A CheckBox is a graphical component that can be in either an on(true) or off (false) state.
- User can select multiple checkbox or single checkbox or zero checkbox depending upon requirement of user.

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.CheckBox;
import javafx.scene.layout.FlowPane;

public class A extends Application
{
        public void start(Stage s)
        {
                CheckBox c1=new CheckBox("Reading");
                CheckBox c2=new CheckBox("Dancing");
                CheckBox c3=new CheckBox("Singing");
                FlowPane f=new FlowPane();
                f.getChildren().addAll(c1,c2,c3);
                Scene s1=new Scene(f,200,250);
                s.setScene(s1);
                s.show();
        }
}
```

7) RadioButton:

- The Radio Button class is a graphical component, which can either be in a ON (true) or OFF (false) state in a group.
- Radio button are mutually exclusive that is user can select only one radio button at a time.

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.CheckBox;
import javafx.scene.layout.FlowPane;
import javafx.scene.control.ToggleGroup
public class A extends Application
{
	public void start(Stage s)
	{
		ToggleGroup t=new ToggleGroup();
		RadioButton r1=new RadioButton(“Male”);
		r1.setToggleGroup(t);
		RadioButton r2=new RadioButton(“Female”);
		r2.setToggleGroup(t);
		FlowPane f=new FlowPane();
		f.getChildren().addAll(r1,r2);
		Scene s1=new Scene(f,200,250);
		s.setScene(s1);
		s.show();
	}
}
```

8) **Combo Box:**

- A **JavaFX ComboBox** control enables the user to select an option from a predefined list of choices, or type in a value.
- The difference between combo box and listview is that in combobox, at load time only one data will be visible to user, rest of items will be visible when user will click that combobox and in listivew, at load time entire list or all the items are visible to user

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.ComboBox;
import javafx.scene.layout.FlowPane;
import jaafx.collections.FXCollections;
import javafx.collections.ObservableList;
public class A extends Application
{
        public void start(Stage s)
        {
ObservableList<String>
items=FXCollections.observableArrayList("M.B.A",M.Sc.IT", I.T");
                ComboBox c=new ComboBox(items);
                FlowPane f=new FlowPane();
                f.getChildren().addAll(c);
                Scene s1=new Scene(f,200,250);
                s.setScene(s1);
                s.show();
        }
}
```

9) ListView:

- A ListView component presents the user with a scrolling list of text items.
- The difference between combo box and listview is that in combobox, at load time only one data will be visible to user, rest of items will be visible when user will click that combobox and in listivew, at load time entire list or all the items are visible to user

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.ListView;
import javafx.scene.layout.FlowPane;
import jaafx.collections.FXCollections;
import javafx.collections.ObservableList;

public class A extends Application
{
        public void start(Stage s)
        {
ObservableList<String>
items=FXCollections.observableArrayList("M.B.A",M.Sc.IT", I.T");
LisView<String>l=new ListView<String>(items);
                FlowPane f=new FlowPane();
                f.getChildren().addAll(l);
                Scene s1=new Scene(f,200,250);
                s.setScene(s1);
                s.show();
        }
}
```

10) ProgressBar:

- As the task progresses towards completion, the progress bar displays the task's percentage of completion.

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.ProgressBar;
import javafx.scene.layout.FlowPane;

public class A extends Application
{
	public void start(Stage s)
	{
		ProgressBar p=new ProgressBar(0.6);
		ProgressIndicator pi=new ProgresIndicator(0.6);
		FlowPane f=new FlowPane();
		f.getChildren().addAll(p,pi);
		Scene s1=new Scene(f,200,250);
		s.setScene(s1);
		s.show();
	}
}
```

11) Slider:

- A Slider lets the user graphically select a value by sliding a knob within a bounded interval.

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Slider;
import javafx.scene.layout.FlowPane;

public class A extends Application
{
	public void start(Stage s)
	{
		Slider s=new Slider();
		s.setMin(0);
		s.setMax(100);
		s.setValue(50);
		FlowPane f=new FlowPane();
		f.getChildren().addAll(s);
		Scene s1=new Scene(f,200,250);
		s.setScene(s1);
		s.show();
	}
}
```

12) ColorPicker:

- A ColorPicker provides a pane of controls designed to allow a user to manipulate and select a color.

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.ColorPicker;
import javafx.scene.layout.FlowPane;
public class A extends Application
{
	public void start(Stage s)
	{
		ColorPicker c=new ColorPicker();
		FlowPane f=new FlowPane();
		f.getChildren().addAll(c);
		Scene s1=new Scene(f,200,250);
		s.setScene(s1);
		s.show();
	}
}
```

13) DatePicker:

- The DatePicker control **allows the user to enter a date as text or to select a date from a calendar popup**.

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.DatePicker;
import javafx.scene.layout.FlowPane;
public class A extends Application
{
	public void start(Stage s)
	{
		DatePicker d=new DatePicker();
		FlowPane f=new FlowPane();
		f.getChildren().addAll(d);
		Scene s1=new Scene(f,200,250);
		s.setScene(s1);
		s.show();
	}
}
```

14) ScrollBar:

- A Scrollbar control represents a scroll bar component in order to enable user to select from range of values.

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.ScrollBar;
import javafx.scene.layout.FlowPane;
public class A extends Application
{
        public void start(Stage s)
        {
                ScrollBar s=new ScrollBar();
                s.setMin(0);
                s.setMax(100);
                s.setValue(50);
                FlowPane f=new FlowPane();
                f.getChildren().addAll(s);
                Scene s1=new Scene(f,200,250);
                s.setScene(s1);
                s.show();
        }
}
```

15) Password Field:

- A PasswordField object is a text component specialized for password entry.

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.PasswordField;
import javafx.scene.layout.FlowPane;

public class A extends Application
{
        public void start(Stage s)
        {
                PasswordField p=new PasswordField();
                FlowPane f=new FlowPane();
                f.getChildren().addAll(p);
                Scene s1=new Scene(f,200,250);
                s.setScene(s1);
                s.show();
        }
}
```

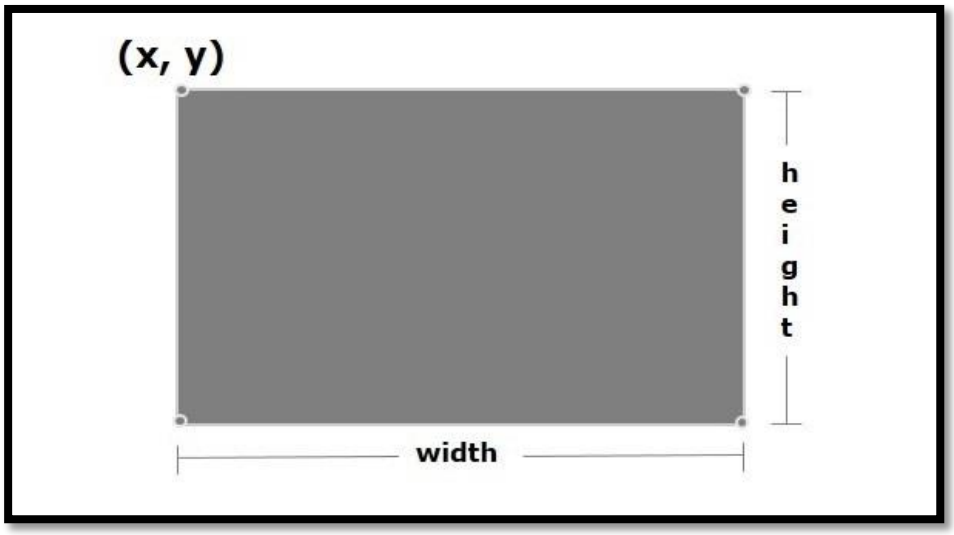
Shapes:

1) **Line:**
   - In JavaFX, a line, or a line segment, is represented by a class named Line
   - This class belongs to the package **javafx.scene.shape**
   - By instantiating this class, you can create a line node in JavaFX.
   - Line class has following 4 methods:
     - setStartX()
     - setStartY()
     - setEndX()
     - setEndY()

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.shape.Line;
import javafx.scene.Group;
public class A extends Application
{
	public void start(Stage s)
	{
		Line l=new Line();
		l.setStartX(100.0);
		l.setStartY(150.0);
		l.setEndX(500.0);
		l.setEndY(150.0);
		Group g=new Group(l);
		Scene s1=new Scene(g,200,250);
		s.setScene(s1);
		s.show();
	}
}
```

2) Rectangle:

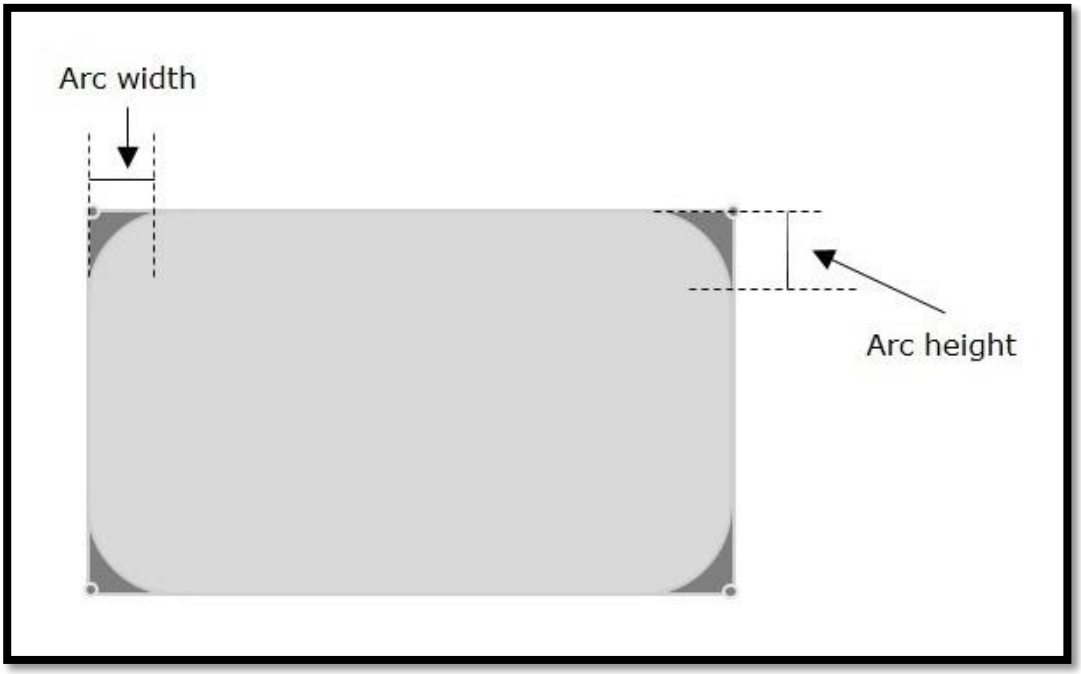
- In JavaFX, a Rectangle is represented by a class named Rectangle
- This class belongs to the package **javafx.scene.shape**
- Rectangle class has following 4 methods:
  - setX()
  - setY()
  - setWidth()
  - setHeight()




```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.shape.Rectangle;
import javafx.scene.Group;

public class A extends Application
{
	public void start(Stage s)
	{
		Rectangle r=new Rectangle();
		r.setX(150.0);
		r.setY(75.0);
		r.setWidth(400.0);
		r.setHeight(150.0);
		Group g=new Group(r);
		Scene s1=new Scene(g,200,250);
		s.setScene(s1);
		s.show();
	}
}
```

3) Rounded rectangle:

- In JavaFX, a rectangle with arched edges is known as rounded rectangle.
- It is represented by Rectangle class.
- This class belongs to the package **javafx.scene.shape**
- Rounded rectangle class has following 6 methods:
    - setX()
    - setY()
    - setWidth()
    - setHeight()
    - setArcHeight()
    - setArcWidth()

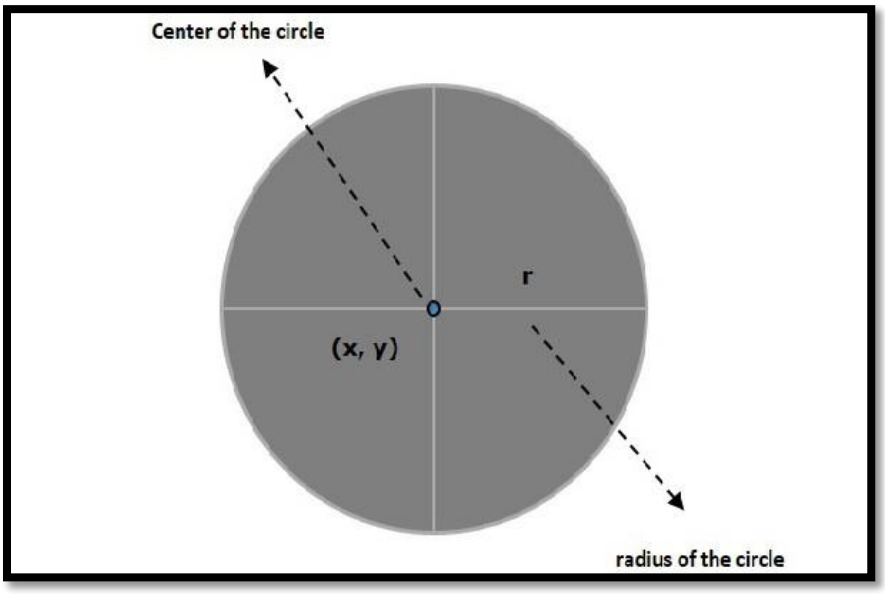


```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.shape.Rectangle;
import javafx.scene.Group;

public class A extends Application
{
	public void start(Stage s)
	{
		Rectangle r=new Rectangle();
		r.setX(150.0);
		r.setY(75.0);
		r.setWidth(400.0);
		r.setHeight(150.0);
		r.setArcWidth(30.0);
		r.setArcHeight(20.0);
		Group g=new Group(r);
		Scene s1=new Scene(g,200,250);
		s.setScene(s1);
		s.show();
	}
}
```

4) Circle:

- In JavaFX, a circle is represented by a class named Circle. A circle is the locus of all points at a fixed distance (radius of circle) from a fixed point (the centre of circle).
- This class belongs to the package **javafx.scene.shape**
- Circle class has following 3 methods:

- setCenterX()
- setCenterY()
- setRadius()




```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.shape.Circle;
import javafx.scene.Group;


public class A extends Application
{
	public void start(Stage s)
	{
		Circle c=new Circle();
		c.setCenterX(300.0);
		c.setCenterY(150.0);
		c.setRadius(50.0);
		Group g=new Group(c);
		Scene s1=new Scene(g,200,250);
		s.setScene(s1);
		s.show();
	}
}
```

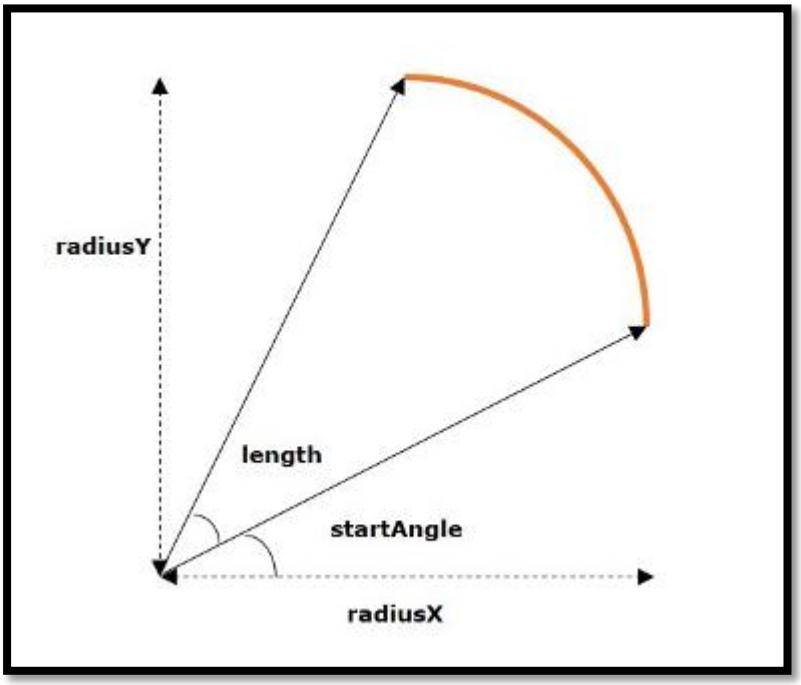
5) Ellipse:

- An Ellipse is defined by two points, each called a focus. If any point on the Ellipse is taken, the sum of the distances to the focus points is constant. The size of the Ellipse is determined by the sum of these two distances. The sum of these distances is equal to the length of the major axis (the longest diameter of the ellipse).
- In JavaFX, a circle is represented by a class named Ellipse.
- This class belongs to the package **javafx.scene.shape**
- Circle class has following 4 methods:
  - setCenterX()
  - setCenterY()
  - setRadiusX()
  - setRadiusY()

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.shape.Ellipse;
import javafx.scene.Group;

public class A extends Application
{
        public void start(Stage s)
        {
                Ellipse e=new Ellipse();
                e.setCenterX(300.0);
                e.setCenterY(150.0);
                e.setRadiusX(150.0);
                e.setRadiusY(75.0);
                Group g=new Group(e);
                Scene s1=new Scene(g,200,250);
                s.setScene(s1);
                s.show();
        }
}
```

6) Arc:

- An arc in simple geometry is defined as a portion of a circumference of an ellipse or a circle.
- In JavaFX, arc is represented by a class named Arc.
- This class belongs to the package **javafx.scene.shape**
- Arc class has following 6 methods:
    - setCenterX()
    - setCenterY()
    - setRadiusX()
    - setRadiusY()
    - setStartAngle()
    - setLength()

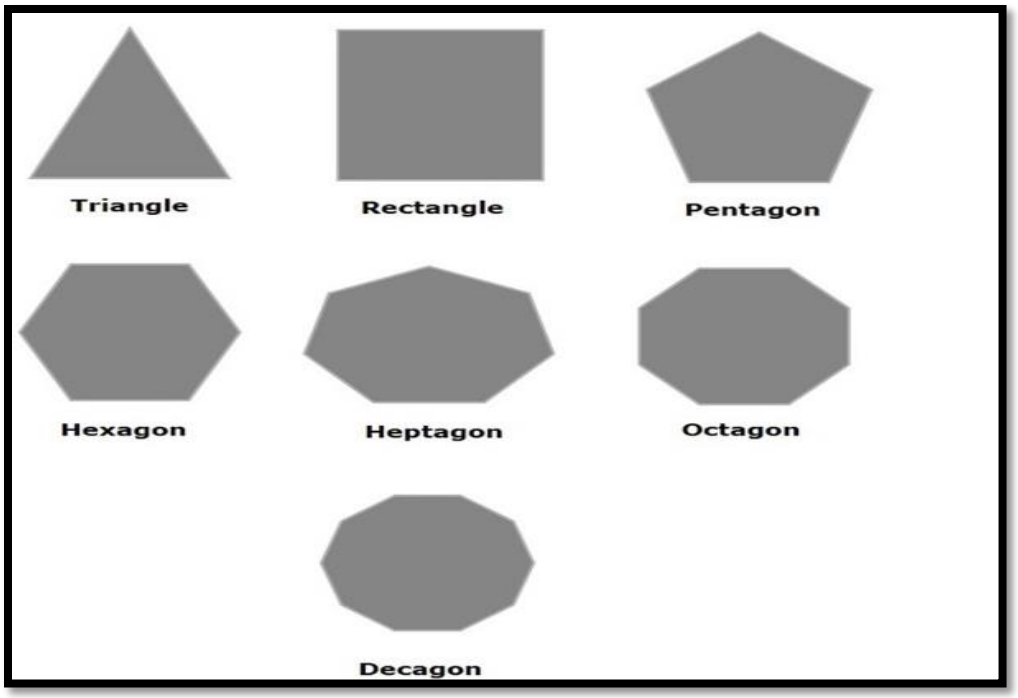


```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.shape.Arc;
import javafx.scene.Group;
public class A extends Application
{
        public void start(Stage s)
        {
        Arc a=new Arc();
        a.setCenterX(400.0);
        a.setCenterY(200.0);
        a.setRadiusX(50.0);
        a.setRadiusY(80.0);
        a.setStartAngle(30.0);
        a.setLength(70.0);
        Group g=new Group(a);
        Scene s1=new Scene(g,200,250);
        s.setScene(s1);
        s.show();
        }
}
```

6) Polygon:

- Polygon is geometrically defined as a closed shape formed by a number of coplanar line segments connected from end to end.
- There are various types of Polygons based on the numbers of sides and angles.

- If a polygon has three sides, then it is referred to as a triangle.
- If a polygon has four sides, then it is known as quadrilateral. Shapes like rectangles, squares, parallelogram etc., are all types of quadrilaterals.
- If a polygon has five sides, then it is known as a pentagon. Similarly, the polygon with six sides is called hexagon, seven sides is heptagon, eight sides is octagon etc.



- In JavaFX, Polygon is represented by a class named Polygon.
- This class belongs to the package **javafx.scene.shape**
- **Polygon has only 1 method:**
  - **getPoints().addAll(new Double[ ]){…….};**

Triangle:

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.shape.Polygon;
import javafx.scene.Group;

public class A extends Application
{
	public void start(Stage s)
	{
		Polygon p=new Polygon();
p.getPoints().addAll(new Double[ ]{200.0,200.0,300.0,100.0,400.0,200.0});
		Group g=new Group(p);
		Scene s1=new Scene(g,200,250);
		s.setScene(s1);
		s.show();
	}
}
```

Pentagon:

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.shape.Polygon;
import javafx.scene.Group;
public class A extends Application
{
        public void start(Stage s)
        {
                Polygon p=new Polygon();
p.getPoints().addAll(new Double[ ]
{400.0,300.0,350.0,250.0,450.0,150.0,500.0,250.0,470.0,300.0});
                Group g=new Group(p);
                Scene s1=new Scene(g,200,250);
                s.setScene(s1);
                s.show();
        }
}
```

Hexagon:

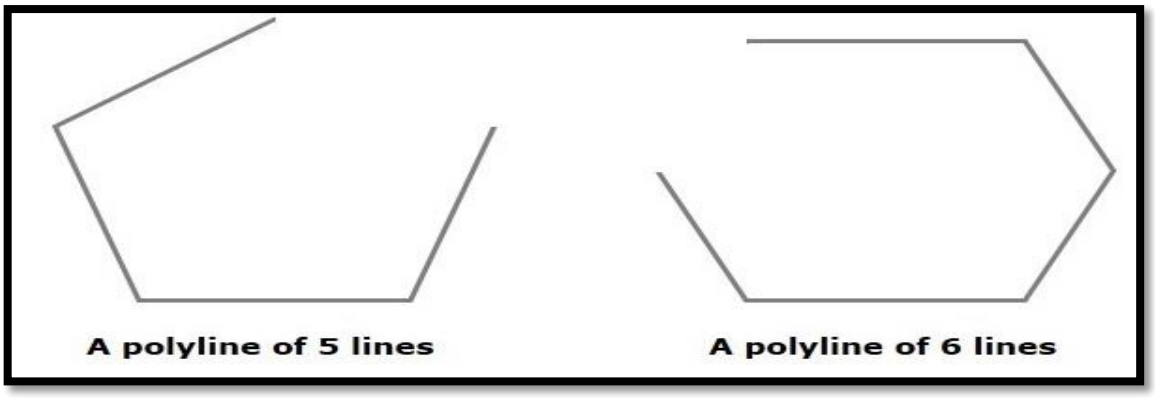
```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.shape.Polygon;
import javafx.scene.Group;


public class A extends Application
{
        public void start(Stage s)
        {
                Polygon p=new Polygon();
p.getPoints().addAll(new Double[ ]
{400.0,300.0,300.0,200.0,400.0,100.0,500.0,100.0,600.0,200.0,500.0,300.0});
                Group g=new Group(p);
                Scene s1=new Scene(g,200,250);
                s.setScene(s1);
                s.show();
        }
}
```

Octagon:

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.shape.Polygon;
import javafx.scene.Group;

public class A extends Application
{
        public void start(Stage s)
        {
                Polygon p=new Polygon();
p.getPoints().addAll(new Double[ ]
{400.0,300.0,300.0,200.0,300.0,100.0,400.0,50.0,500.0,50.0,600.0,100.0,600.0,
200.0,500.0,300.0});
                Group g=new Group(p);
                Scene s1=new Scene(g,200,250);
                s.setScene(s1);
                s.show();
        }
}
```

7) PolyLine:

- A Polyline is same as a polygon except that a polyline is not closed in the end.
- In JavaFX, Polyline is represented by a class named Polyline.
- This class belongs to the package **javafx.scene.shape**
- **Polyline has only 1 method:**
    - **getPoints().addAll(new Double[ ]){…….};**

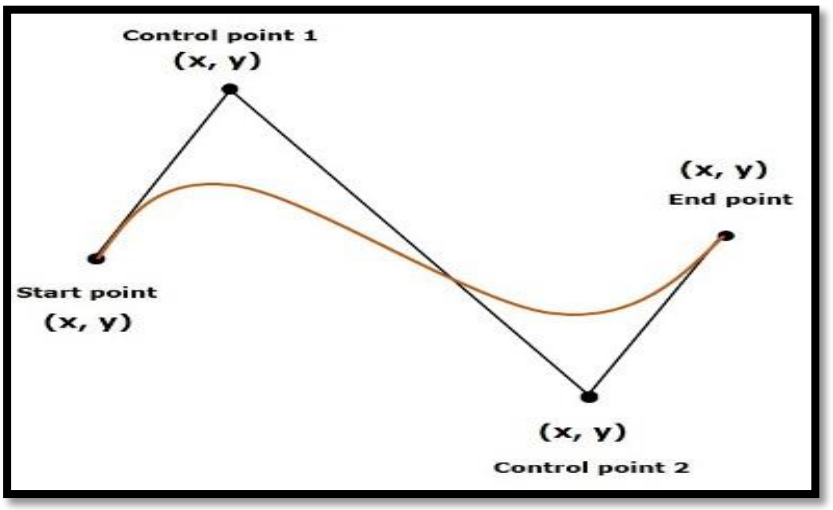


Triangle with PolyLine:

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.shape.PolyLine;
import javafx.scene.Group;
public class A extends Application
{
	public void start(Stage s)
	{
		PolyLine p=new PolyLine();
p.getPoints().addAll(new Double[ ]{200.0,200.0,300.0,100.0,400.0,200.0});
		Group g=new Group(p);
		Scene s1=new Scene(g,200,250);
		s.setScene(s1);
		s.show();
	}
}
```

8) CubicCurve:

- A cubic curve is a Bezier parametric curve in the XY plane is a curve of degree 3. It is drawn using four points – Start Point, End Point, Control Point1 and ControlPoint2.
- In JavaFX, Cubic Curve is represented by a class named Cubic Curve
- This class belongs to the package **javafx.scene.shape**
- Cubic Curve class has following 8 methods:
    - setStartX()
    - setStartY()
    - setControlX1()
    - setControlY1()
    - setControlX2()
    - setControlY2()
    - setEndX()
    - setEndY()



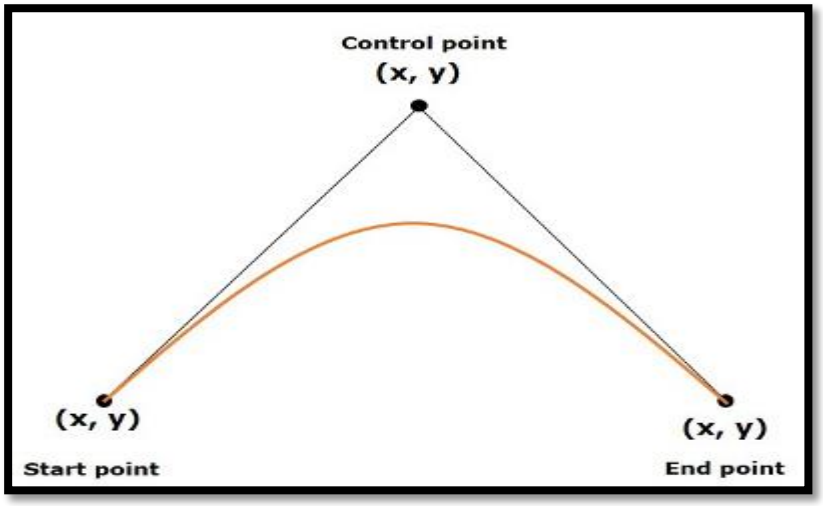
Example:

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.shape.CubicCurve;
import javafx.scene.Group;
public class A extends Application
{
        public void start(Stage s)
        {
        CubicCurve c=new CubicCurve();
        c.setStartX(200.0);
        c.setStartY(200.0);
        c.setControlX1(300.0);
        c.setControlY(100.0);
        c.setControlX2(400.0);
        c.setControlY2(300.0);
        c.setEndX(500.0);
        c.setEndY(200.0);
        Group g=new Group(c);
        Scene s1=new Scene(g,200,250);
        s.setScene(s1);
        s.show();
        }
}
```

9) QuadCurve:

- A quadratic curve is a Bezier parametric curve in the XY plane which is a curve of degree 2. It is drawn using three points. Start Point, End Point, Control Point.
- In JavaFX, Quad Curve is represented by a class named Quad Curve
- This class belongs to the package **javafx.scene.shape**
- Quad Curve class has following 8 methods:
    - setStartX()
    - setStartY()
    - setControlX()

- setControlY()
- setEndX()
- setEndY()



Example:

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.shape.QuadCurve;
import javafx.scene.Group;
public class A extends Application
{
        public void start(Stage s)
        {
        QuadCurve c=new QuadCurve();
        c.setStartX(200.0);
        c.setStartY(200.0);
        c.setControlX(300.0);
        c.setControlY(100.0);
        c.setEndX(500.0);
        c.setEndY(200.0);
        Group g=new Group(c);
        Scene s1=new Scene(g,200,250);
        s.setScene(s1);
        s.show();
        }
}
```

Q-4 Write a short note on Color class and Font class

Ans:

Color Class:

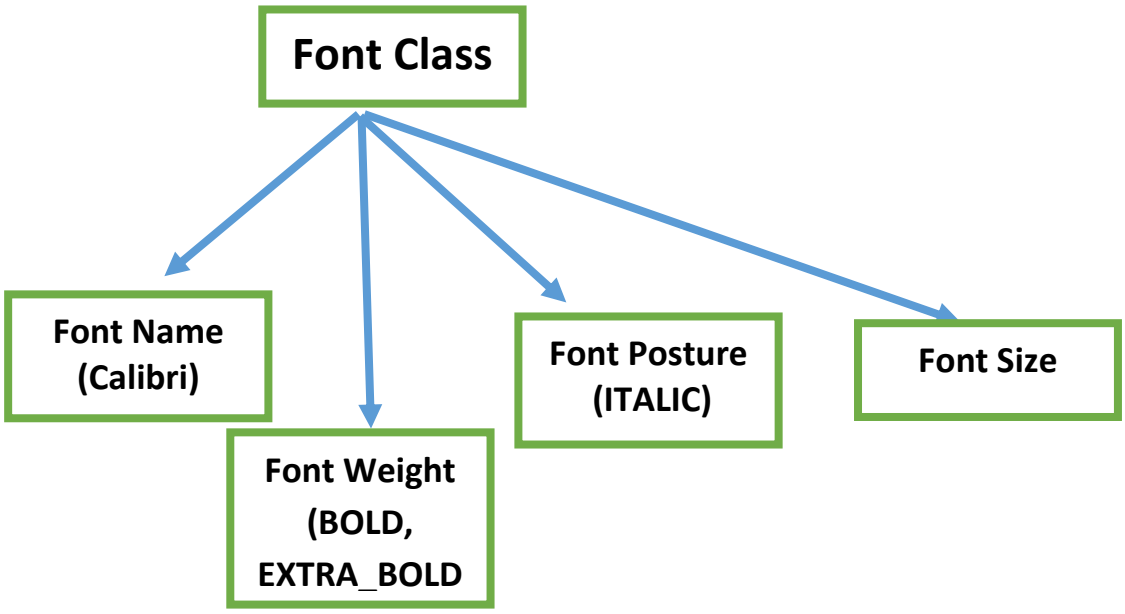
- When you draw a 2D shape in a JavaFX application, you might have observed that, by default, it is colored black. But, the color black is not always suitable for all types of applications a user creates. Hence, JavaFX allows you to change this default color into whichever color the user deems perfect for their application.
- To apply colors to an application, JavaFX provides various classes in the package javafx.scene.paint package. This package contains an abstract class named Paint and it is the base class of all the classes that are used to apply colors.
- All those node classes to which color can be applied such as shape, text have two methods: setFill() and setStroke()

Example:

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.shape.Circle;
import javafx.scene.Group;
import javafx.scene.paint.Color;
public class A extends Application
{
        public void start(Stage s)
        {
        Circle c=new Circle();
        c.setCenterX(300.0);
        c.setCenterY(150.0);
        c.setRadius(50.0);
        c.setFill(Color.RED);
        Group g=new Group(c);
        Scene s1=new Scene(g,200,250);
        s.setScene(s1);
        s.show();
        }
}
```

Font Class:

- A **Font** is defined by its name, weight, posture, and size.
- The **javafx.scene.text.Font** class is used to create fonts,



Example:

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.FlowPane;
import javafx.scene.text.Text;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.FontPosture;
public class A extends Application
{
        public void start(Stage s)
        {
        Text t=new Text("hello how are you");
        Font f= Font.   font("Calibri",FontWeight.EXTRA_BOLD,FontPosture.ITALIC,40);
t.setFont(f);
FlowPane f1=new FlowPane();
f1.getChildren().addAll(t);
Scene s1=new Scene(f1,200,250);
        s.setScene(s1);
        s.show();
        }
}
```

Q-5 Explain Property Binding

Ans

- Target object can be binded to source object. A change in source object will be automatically reflected in target object
- Property binding in javafx is a concept that enables a target object to be bound to source object. The target object is called a binding object or binding property and source object is called bindable object.
- Example: Circle is not centered after window is resized. In order to display the circle centered as window resizes, x and y coordinates of circle need to be reset to the center of the pane. This can be done by binding centerX with pane's width/2 and center with pane's height/2

Example:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.layout.Pane;
import javafx.scene.shape.Circle;

public class A extends Application
{
        public void start(Stage s)
        {
                Circle c=new Circle();
                Pane p=new Pane();
                c.centerXProperty().bind(p.widthProperty().divide(2));
                c.centerYProperty().bind(p.heightProperty().divide(2));
                c.setRadius(50.0);
                p.getChildren().addAll(c );
                Scene s1=new Scene(p,200,250);
                s.setScene(s1);
                s.show();
        }
}
```

Q-6 Explain Image and ImageView Class

Ans:

- ImageView is a class used for painting and loading images with Image class
- Image View class is allowing user to resize the displayed image by without affecting the aspect ratio and also without effecting image pixels
- In javafx, before creating the IamgeView class, we must create Image Class to load this object into ImageView class
- Following are the different methods of ImageView class
    - setX() : used to set x coordinates of image
    - setY(): used to set y coordinates of image
    - setFitWidth(): used to set width of image
    - setFitHeight(): used to set height of image
- Image class and ImageView class is available in:
    - import javafx.scene.image.Image;
    - import javafx.scene.image.ImageView

Example:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.Group;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import java.io.*;

public class A extends Application
{
        public void start(Stage s)throws Exception
        {
                FileInputStream f=new FileInputStream("E:\\images\\1.jpg");
                Image i=new Image(f);
                ImageView v=new ImageView(i);
                v.setX(200.0);
                v.setY(150.0);
                v.setFitWidth(400.0);
                v.setFitHeight(300.0);
                Group g=new Group(v);
                Scene s1=new Scene(g,200,250);
                s.setScene(s1);
                s.show();
}
```

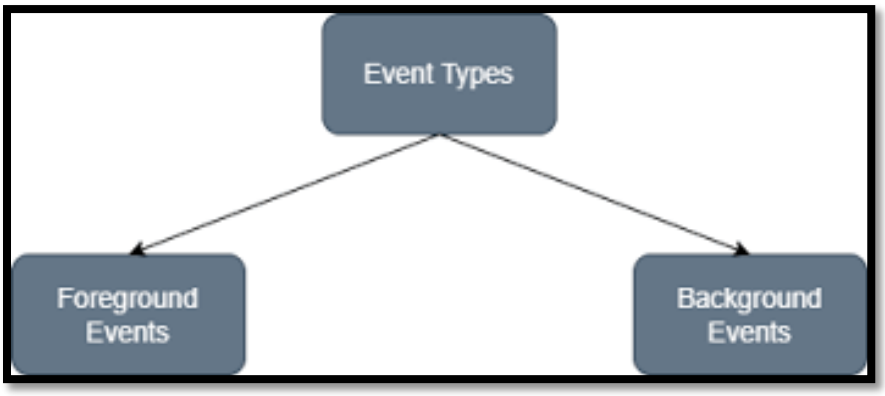
Q-7 Explain Event handling in javafx

Ans:

- In javafx applications, an event is a change in the state of an object triggered by some action such as clicking the button, scrolling the page etc. The package used for event handling is:

  ```
  import javafx.event.Event;
  import javafx.event.EventHandler;
  ```

- An event is occurred whenever user interacts with application nodes.
- There are2 types of events:

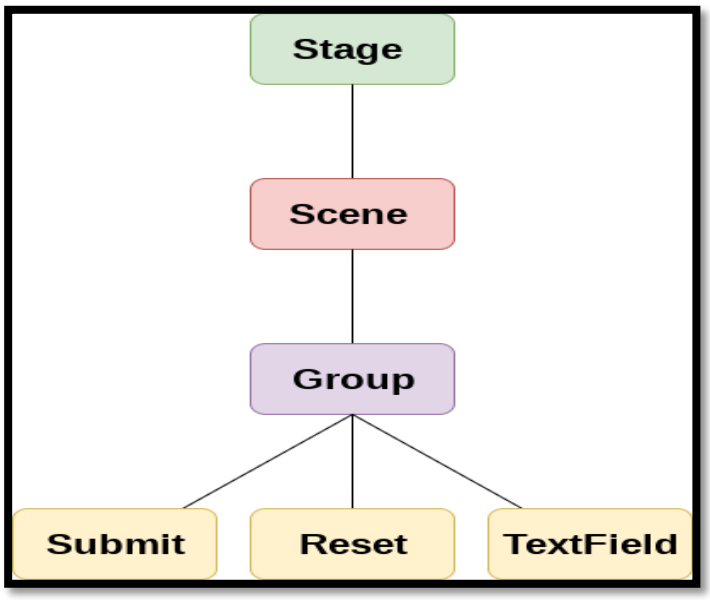


1) Foreground Events: The events which require user interaction with GUI application such as button click etc
2) Background Events: The events which does not require user's interaction with application. These events are mainly occurred due to operating system interrupts, failure, operation completion etc.

Every event has 3 components:

1) **Event Source**: The node or the controls which generate the event
   Example: Button
2) **Event Target**: The node or the controls on which event occurred
   Example: Scene, Window
3) **Type**: It indicates type of event
   Example: Button click event

Event Delivery Process:



1) **Route Construction:**
   - An event dispatch chain is created in order to determine the default route of the event, whenever it is generated
   - The event dispatch chain contains the path from stage to Node in which the event is generated

2) **Event Capturing Phase:**
   - Once the event dispatch chain is created, the event is dispatched from source node of event.
   - All the nodes are traversed by the event from top to bottom
   - If the event filter is registered with any of the nodes, it will be executed.
   - If any of the nodes are not registered with event filter then the event is transferred to target node. The target node processes the event.

3) **Event Bubbling:**
   - Event bubbling can be defined as a phase of Event propagation in which if an event occurs on a particular element will propagate or bubble up to the ancestor or the parent elements in the DOM hierarchy.
   - Once the event is processed by target node or by any of the registered filter, the event traverses all the nodes again from bottom to stage node.

4) **Event Handlers and Filters:**
   - It contains the application logic to process an event. Node can be registered to more than one event filter.

Event Classes in javafx:

1) **Action Event:**

- In JavaFX, the ActionEvent class, a subclass of Event, represents actions like button clicks and key presses, and is used to handle events in your GUI. The package used for ActionEvent class is **import javafx.event.ActionEvent.**
- Action Event class is used in conjunction with event handlers like setOnAction() for buttons to define what should happen when specific action occurs.

Example:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.Scene.Control.Button;
import javafx.scene.layout.GridPane;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.Scene.Control.Label;

public class A extends Application
{
        public void start(Stage ps)
        {
                Button b=new Button(“Ok”);
                GridPane g=new GirdPane();
                Label l=new Label();
EventHandler<ActionEvent> e=new EventHandler<ActionEvent>(){
public void handle(ActionEvent e1)
{
        l.setText(“Hello”);
}
};
b.setOnAction(e);
g.addRow(0,b);
g.addRow(0,l);
Scene s1=new Scene(g,200,250);
s.setScene(s1);
s.show();
}
}
```

2) Mouse Event:

- In JavaFX, the MouseEvent class belongs to following package:

  import javafx.scene.input.MouseEvent
- Mouse Event class represents events triggered by mouse interactions like clicking, dragging, entering, and exiting a node. You can use this class to handle these events and react accordingly in your application.
- Following are the methods of MouseEvent class
  1) MousePressed()
  2) MouseReleased()
  3) MouseDragged()
  4) MouseClicked()
  5) MouseEntered()
  6) MouseExited()

Example:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.layout.GridPane;
import javafx.event.MouseEvent;
import javafx.event.EventHandler;
import javafx.Scene.Control.Label;

public class A extends Application
{
        public void start(Stage ps)
        {
                GridPane g=new GirdPane();
                Label l=new Label();
EventHandler<MouseEvent> e1=new EventHandler<MouseEvent>(){
public void handle(MouseEvent e1)
{
        l.setText("Mouse Pressed");
}
};
EventHandler<MouseEvent> e2=new EventHandler<MouseEvent>(){
public void handle(MouseEvent e2)
{
        l.setText("Mouse Released");
}
};
EventHandler<MouseEvent> e3=new EventHandler<MouseEvent>(){
public void handle(MouseEvent e3)
{
        l.setText("Mouse Clicked");
}
};

EventHandler<MouseEvent> e4=new EventHandler<MouseEvent>(){
public void handle(MouseEvent e4)
{
        l.setText("Mouse Entered");
}
};
```

```
EventHandler<MouseEvent> e5=new EventHandler<MouseEvent>(){
public void handle(MouseEvent e5)
{
        l.setText("Mouse Exited");
}
};
EventHandler<MouseEvent> e6=new EventHandler<MouseEvent>(){
public void handle(MouseEvent e6)
{
        l.setText("Mouse Dragged");
}
};
g.getChidren().addAll(l);
Scene s1=new Scene(g,200,250);
s1.setOnMousePressed(e1);
s1.setOnMouseReleased(e2);
s1.setOnMouseClicked(e3);
s1.setOnMouseEntered(e4);
s1.setOnMouseExited(e5);
s1.setOnMouseDragged(e6);
ps.setScene(s1);
ps.show();
}
}
```

3) Key Event:

- In JavaFX, the KeyeEvent class belongs to following package:

  import javafx.scene.input.KeyEvent

- An event which indicates that a keystroke occurred in a Node . This event is generated when a key is pressed, released, or typed. Depending on the type of the event it is passed to onKeyPressed , onKeyTyped or onKeyReleased function.
- Following are the methods of MouseEvent class
  1) KeyPressed()
  2) KeyReleased()
  3) KeyTyped()

Example:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.layout.GridPane;
import javafx.event.KeyEvent;
import javafx.event.EventHandler;
import javafx.Scene.Control.Label;
public class A extends Application
{
	public void start(Stage ps)
	{
		GridPane g=new GirdPane();
		Label l=new Label();
EventHandler<KeyEvent> e1=new EventHandler<KeyEvent>(){
public void handle(KeyEvent e1)
{
	l.setText(“Key Pressed”);
}
};
EventHandler<KeyEvent> e2=new EventHandler<KeyEvent>(){
public void handle(KeyEvent e2)
{
	l.setText(“Key Released”);
}
};
EventHandler<KeyEvent> e3=new EventHandler<KeyEvent>(){
public void handle(KeyEvent e3)
{
	l.setText(“Key Typed”);
}
};

g.getChidren().addAll(l);
Scene s1=new Scene(g,200,250);
s1.setOnKeyPressed(e1);
s1.setOnKeyReleased(e2);
s1.setOnKeyTyped(e3);
ps.setScene(s1);
ps.show();
}
}
```

Q-8 Write a short note on nested and inner class in javafx

Ans:

- In Javafx , **nested classes** and **inner classes** are common ways to organize your code and improve the readability and structure of your application.
- The classes are defined within another class, but they differ in how they are structured and how they can access members of the enclosing class.

Types of Nested and Inner Classes:

1. **Static Nested Class**: A nested class that is marked as static. It can exist independently of an instance of the enclosing class and doesn't have access to the instance variables of the enclosing class.
2. **Non-static Inner Class**: A nested class that is not marked as static. It has access to all the members (including private members) of the enclosing class.
3. **Local Class**: A class defined within a method or a block of code, usually to provide a specific functionality to that method or block. Local classes can access local variables and parameters within the method they are defined.
4. **Anonymous Inner Class**: A class defined and instantiated in a single expression, usually to handle a specific action like event handling.

Example: Static Nested Class

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.control.Button;
import javafx.scene.layout.FlowPane;

public class A extends Application
{
	public void start(Stage s)
	{
		FlowPane f=new FlowPane();
		B b1=new B();
		b1.print();
		Scene s1=new Scene(f,200,250);
		s.setScene(s1);
		s.show();
	}

		static class B
		{
			public void print()
			{
				System.out.println("hello");
			}
		}
}
```

Example: Non-Static Nested Class

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.control.Button;
import javafx.scene.layout.FlowPane;

public class A extends Application
{
	private String s=”hello how are you”;
	public void start(Stage s)
	{
		FlowPane f=new FlowPane();
		B b1=new B();
		b1.print();
		Scene s1=new Scene(f,200,250);
		s.setScene(s1);
		s.show();
	}

		class B
		{
			public void print()
			{
				System.out.println(s);
			}
		}
}
```

Example: Local Nested Class

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.control.Button;
import javafx.scene.layout.FlowPane;

public class A extends Application
{
	public void start(Stage s)
	{

		FlowPane f=new FlowPane();
		class B
		{
			public void print()
			{
				System.out.println(“hello”);
			}
		}

		B b1=new B();
		b1.print();
		Scene s1=new Scene(f,200,250);
		s.setScene(s1);
		s.show();
	}

}
```

Example: Annonymous Nested class

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.control.Button;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.layout.FlowPane;

public class A extends Application
{
        public void start(Stage s)
        {
                Button b=new Button("ok");
                FlowPane f=new FlowPane();

b.setOnAction(new EventHandler<ActionEvent>(){
public void handle(ActionEvent e)
{
        System.out.println("button clicked");
}
});
f.getChildren().addAll(b);
Scene s1=new Scene(f,200,240);
s.setScene(s1);
s.show();
}
}
```

- In the above example, the btn.setOnAction() method takes an EventHandler<ActionEvent>. Instead of defining a separate class for the event handler, an anonymous inner class is used.
- The handle method is overridden to define the behavior when the button is clicked.

Q-9 Write a short note on Animation

Ans:

- Animations are used in an application to add certain special visual effects on elements like images, text, drawings, etc. You can specify the entry and exit effects on a text, fading an image in and out etc
- In JavaFX, a node can be animated by changing its property over time. JavaFX provides a package named javafx.animation.This package contains classes that are used to animate the nodes. Animation is the base class of all these classes.

Fade Transition:

- In JavaFX, the fade transition is used to transition a node's opacity from a starting value to an ending value over a specified duration. This can be done using the FadeTransition class belonging to the javafx. Animation package.

Example:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.layout.FlowPane;
import javafx.scene.shape.Circle;
import javafx.animation.FadeTransition;
import javafx.scene.paint.Color;
import javafx.util.Duration;
public class A extends Application
{
        public void start(Stage s)
        {
Circle c=new Circle();
c.setRadius(50.0);
c.setFill(Color.RED);
Flowpane f=new FlowPane();
FadeTransition f1=new FadeTransition(Duration.millis(1000));
f1.setNode(c);
f1.setFromValue(1.0);
f1.setToValue(0.3);
f1.setCycleCount(50);
f1.setAutoReverse(false);
f1.play();
f.getChildren().addAll( c);
Scene s1=new Scene(f,200,250);
s.setScene(s1);
s.show();
        }
}
```

2) Fill Transition:

- Fill transition is a type of transition that changes the color of a JavaFX node during a specified period. This transition is commonly used in applications to conduct quizzes: where the option turns "green" if the answer is correct and "red" otherwise

Example:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.layout.FlowPane;
import javafx.scene.shape.Circle;
import javafx.animation.FillTransition;
import javafx.scene.paint.Color;
import javafx.util.Duration;
public class A extends Application
{
        public void start(Stage s)
        {
Circle c=new Circle();
c.setRadius(50.0);
Flowpane f=new FlowPane();
FillTransition f1=new FillTransition(Duration.millis(1000));
f1.setShape(c);
f1.setFromValue(Color.BLUE);
f1.setToValue(Color.GREEN);
f1.setCycleCount(50);
f1.setAutoReverse(false);
f1.play();
f.getChildren().addAll( c);
Scene s1=new Scene(f,200,250);
s.setScene(s1);
s.show();
        }
}
```

3) Rotate Transition:

- Rotate Transition in JavaFX is applied using the RotateTransition class in the javafx.animation package. This is done by specifying the starting value, the ending value and the duration of the transition.
- Rotate transition is used to deal with an object's position by retaining its shape and properties.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.layout.FlowPane;
import javafx.scene.shape.Polygon;
import javafx.animation.RotateTransition;
import javafx.scene.paint.Color;
import javafx.util.Duration;
public class A extends Application
{
        public void start(Stage s)
        {
Polygon p=new Polygon();
p.getPoints().addAll(new Double[ ]{400.0,300.0,500.0,200.0,600.0,300.0});
Flowpane f=new FlowPane();
RotateTransition f1=new Rotate Transition(Duration.millis(1000));
f1.setNode(c);
f1.setByAngle(360);
f1.setCycleCount(50);
f1.setAutoReverse(false);
f1.play();
f.getChildren().addAll(p);
Scene s1=new Scene(f,200,250);
s.setScene(s1);
s.show();
        }
}
```

4) Scale Transition:

- Scaling refers to increasing or decreasing the size of an object. In computer graphics, by applying the scale transition on an object (or image), you can either increase or decrease its size, for a specified duration.

Example:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.layout.FlowPane;
import javafx.scene.shape.Circle;
import javafx.animation.ScaleTransition;
import javafx.scene.paint.Color;
import javafx.util.Duration;
import javafx.scene.Group;


public class A extends Application
{
        public void start(Stage s)
        {
Circle c=new Circle();
c.setRadius(50.0);
c.setCenterX(500.0);
c.setCenterY(200.0);
c.setFill(Color.RED);
Group g=new Group(c);


ScaleTransition f1=new ScaleTransition(Duration.millis(1000));
f1.setNode(c);
f1.setByX(3.5);
f1.setByY(3.5);
f1.setCycleCount(50);
f1.setAutoReverse(false);
f1.play();
Scene s1=new Scene(g,200,250);
s.setScene(s1);
s.show();
        }
}
```

5) Stroke Transition:

- Stroke transition is used to change the stroke color of a shape.
- A shape in JavaFX can consist of three types of strokes: inside, outside and centered; with different properties applied to it.
- JavaFX offers stroke transition using the StrokeTransition class which belongs to the javafx.animation package.

Example:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.layout.FlowPane;
import javafx.scene.shape.Circle;
import javafx.animation.StrokeTransition;
import javafx.scene.paint.Color;
import javafx.util.Duration;
import javafx.scene.Group;


public class A extends Application
{
        public void start(Stage s)
        {
Circle c=new Circle();
c.setRadius(50.0);
c.setCenterX(500.0);
c.setCenterY(200.0);
c.setFill(Color.RED);
c.setStrokeWidth(10.0);
Group g=new Group(c);


StrokeTransition f1=new StrokeTransition(Duration.millis(1000));
f1.setShape(c);
f1.setFromValue(Color.BLACK);
f1.setToValue(Color.GREEN);
f1.setCycleCount(50);
f1.setAutoReverse(true);
f1.play();
Scene s1=new Scene(g,200,250);
s.setScene(s1);
s.show();
        }
}
```

6) TranslateTransition:

- The Translate transition simply moves an object from one location to another on an application. This is usually done by specifying the new coordinate points or the distance this object needs to be moved to.
- **Example:**

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.layout.FlowPane;
import javafx.scene.shape.Circle;
import javafx.animation.TranslateTransition;
import javafx.scene.paint.Color;
import javafx.util.Duration;
import javafx.scene.Group;


public class A extends Application
{
        public void start(Stage s)
        {
Circle c=new Circle();
c.setRadius(50.0);
c.setCenterX(500.0);
c.setCenterY(200.0);
c.setFill(Color.RED);
Group g=new Group(c);


TranslateTransition f1=new TranslateTransition(Duration.millis(1000));
f1.setNode(c);
f1.setByX(600.0);
f1.setCycleCount(50);
f1.setAutoReverse(false);
f1.play();
Scene s1=new Scene(g,200,250);
s.setScene(s1);
s.show();
        }
}
```

7) Sequential Transition:

- Sequential transition is applied on a JavaFX node when you want to apply multiple transitions on a JavaFX node one after the other.
- SequentialTransition class is used to play multiple transitions on a JavaFX node in a sequential order.

Example:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.layout.FlowPane;
import javafx.scene.shape.Circle;
import javafx.animation.TranslateTransition;
import javafx.scene.paint.Color;
import javafx.util.Duration;
import javafx.scene.Group;
import javafx.animation.SequentialTransition;
import javafx.animation.FadeTransition;

public class A extends Application
{
        public void start(Stage s)
        {
Circle c=new Circle();
c.setRadius(50.0);
c.setCenterX(500.0);
c.setCenterY(200.0);
c.setFill(Color.RED);
FadeTransition ft=new FadeTransition(Duration.millis(1000));
ft.setNode(c);
ft.setFromValue(1.0);
ft.setToValue(0.3);
ft.setCycleCount(50);
ft.setAutoReverse(false);
TranslateTransition f1=new TranslateTransition(Duration.millis(1000));
f1.setNode(c);
f1.setByX(600.0);
f1.setCycleCount(50);
f1.setAutoReverse(false);
SequentialTransition st=new SequentialTransition(c, ft,f1);
st.play();
Group g=new Group(c);
Scene s1=new Scene(g,200,250);
s.setScene(s1);
s.show();
        }
}
```

8) Pause Transition:

- Pause transition is a type of transition offered by JavaFX, which pauses the animation for a while before resuming another transition in a sequential order.

Example:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.layout.FlowPane;
import javafx.scene.shape.Circle;
import javafx.animation.TranslateTransition;
import javafx.scene.paint.Color;
import javafx.util.Duration;
import javafx.scene.Group;
import javafx.animation.SequentialTransition;
import javafx.animation.FadeTransition;
import javafx.animation.PauseTransition;
public class A extends Application
{
        public void start(Stage s)
        {
Circle c=new Circle();
c.setRadius(50.0);
c.setCenterX(500.0);
c.setCenterY(200.0);
c.setFill(Color.RED);
FadeTransition ft=new FadeTransition(Duration.millis(1000));
ft.setNode(c);
ft.setFromValue(1.0);
ft.setToValue(0.3);
ft.setCycleCount(50);
ft.setAutoReverse(false);
TranslateTransition f1=new TranslateTransition(Duration.millis(1000));
f1.setNode(c);
f1.setByX(600.0);
f1.setCycleCount(50);
f1.setAutoReverse(false);
PauseTransition pt=new PauseTransition();
pt.setDuration(Duration.millis(1000));
SequentialTransition st=new SequentialTransition(c, ft,pt,f1);
st.play();
Group g=new Group(c);
Scene s1=new Scene(g,200,250);
s.setScene(s1);
s.show();
        }
`
```

9) Parallel Transition:

- Parallel Transition used to play multiple transitions parallelly on a JavaFX node.
- Javafx allows you to apply multiple transitions on a node. This can be done by applying one transition at a time (called sequential transitions) or applying multiple transitions at a time (called parallel transitions).

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.layout.FlowPane;
import javafx.scene.shape.Circle;
import javafx.animation.TranslateTransition;
import javafx.scene.paint.Color;
import javafx.util.Duration;
import javafx.scene.Group;
import javafx.animation.ParallelTransition;
import javafx.animation.FadeTransition;
import javafx.animation.PauseTransition;
public class A extends Application
{
        public void start(Stage s)
        {
Circle c=new Circle();
c.setRadius(50.0);
c.setCenterX(500.0);
c.setCenterY(200.0);
c.setFill(Color.RED);
FadeTransition ft=new FadeTransition(Duration.millis(1000));
ft.setNode(c);
ft.setFromValue(1.0);
ft.setToValue(0.3);
ft.setCycleCount(50);
ft.setAutoReverse(false);
TranslateTransition f1=new TranslateTransition(Duration.millis(1000));
f1.setNode(c);
f1.setByX(600.0);
f1.setCycleCount(50);
f1.setAutoReverse(false);
ParallelTransition st=new ParallelTransition(c, ft,f1);
st.play();
Group g=new Group(c);
Scene s1=new Scene(g,200,250);
s.setScene(s1);
s.show();
        }
}
```

Q-10 Write a short note on Video and Audio or Multimedia

Ans:

- The JavaFX media support, meaning JavaFX video and audio support, is provided by the JavaFX media classes Media, MediaPlayer, MediaView and AudioClip.
- In JavaFX, the `Media`, `MediaPlayer`, and `MediaView` classes work together to provide a robust mechanism for playing media files such as audio and video.
- The Media class represents a media file (either audio or video) and contains the details needed to load and play that file.
- he MediaPlayer class is responsible for controlling the playback of the media represented by the Media object. It allows you to start, pause, stop, and adjust various properties like volume, rate, and looping.
- The MediaView class is used to display the video content of a MediaPlayer.
- **m.toURI()**: Converts the object m (likely a File) to a URI.
- **.toURL()**: Converts the URI to a URL. This is valid for a URI object, but it's not always necessary for creating a Media object in JavaFX.
- **.toString()**: Converts the URL to a String.

Example:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.layout.FlowPane;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javafx.scene.media.MediaView;
import java.io.File;
import java.net.MalformedURLException;

public class A extends Application
{
        public void start(Stage s)throws MalformedURLException
        {
                File m=new File("C:/Users/HP/Downloads/2.mp4");
                Media m1 = new Media(m.toURI().toURL().toString());
                 MediaPlayer m2 = new MediaPlayer(m1);
                MediaView mv = new MediaView(m2);

FlowPane f=new FlowPane();
f.getChildren().addAll(mv);
Scene s1=new Scene(f,200,250);
s.setScene(s1);
s.show();
m2.play();
}
}
```

Q-11 Write a short note on Listener for observable object

Ans:

- In JavaFX, you can create listeners for changes in observable objects using the ChangeListener
- These listeners are particularly useful for responding to changes in properties or collections that are wrapped in Observable objects.

Change Listener:

- A ChangeListener is used when you need to listen to changes in a **single value**, such as a property of an object. It's commonly used for properties like StringProperty, IntegerProperty, BooleanProperty, etc.
- The ChangeListener interface has the following method:

  void changed(ObservableValue<? extends T> observable, T oldValue, T newValue);

Example:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.layout.FlowPane;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.beans.property.IntegerProperty;
import javafx.beans.property.SimpleIntegerProperty;

public class A extends Application
{
        public void start(Stage s)
        {
                IntegerProperty n = new SimpleIntegerProperty(10);
                n.addListener(new ChangeListener<Number>() {

       public void changed(ObservableValue<? extends Number> observable,
Number oldValue, Number newValue) {
          System.out.println("Old value: " + oldValue + ", New value: " + newValue);
        }
     });
FlowPane f=new FlowPane();
Scene s1=new Scene(f,200,250);
s.setScene(s1);
s.show();
}
}
```