Unit 1

1. Ravi has to make sure that this application should return the grade of each students according to the marks obtained where Grade A (marks &gt;=85) /Grade B (marks &gt;=65) /Grade C (marks&gt;=45) / Grade F (marks &lt;45).Ravi is warned with the instruction that he has to validate the score value so that it should not be negative and not greater than 100.

   Code:-

```
import java.util.Scanner;
public class Grade
{
   public static void main(String args[])
   {   int mark;
       Scanner sc = new Scanner(System.in);
         System.out.print("Enter Marks ");
         mark = sc.nextInt();
       System.out.print("The student Grade is: ");
       if(mark>100 || mark<0)
       {
          System.out.print("The entered marks are not valid ");
       }

       else if(mark>=85 && mark<=100)
       {
          System.out.print("A");
       }
       else if(mark>=65 && mark<85)
       {
         System.out.print("B");
       }
       else if(mark>=45 && mark<65)
       {
          System.out.print("C");
       }
       else
       {
          System.out.print("D");
       }
   }
}
```

2. Analyze the concept of Constructor in java and discuss the various types of it. Write a program create a class ‘simpleobject‘. Using constructor display the message.

Ans:-Analyze the concept of Constructor in java and discuss the various types of it. Write a program create a class ‘simpleobject‘. Using constructor display the message.

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 constructors

There are two types of constructors in Java:

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

Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

1. <class_name>(){}

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Code:-

```
class simpleobject{
simpleobject(){
   System.out.println("constructer is created");}
public static void main(String args[]){
 simpleobject b=new simpleobject();
}
}
```

3. Develop a program to create a class named shape. In this class we have three sub classes circle, triangle and square each class has two-member function named draw () and erase () .Create these using polymorphism concepts.

   Code:-

```
class Shape {
    void draw() {
        System.out.println("Drawing Shape");
    }
    void erase() {
        System.out.println("Erasing Shape");
    }
```

```
 }
class Circle extends Shape {

      void draw() {
          System.out.println("Drawing Circle");        }

      void erase() {
          System.out.println("Erasing Circle");
      }
 }
class Triangle extends Shape {

     void draw() {
           System.out.println("Drawing Triangle");
     }

      void erase() {
           System.out.println("Erasing Triangle");
      }
 }
class Square extends Shape {

      void draw() {
           System.out.println("Drawing Square");
      }

      void erase() {
           System.out.println("Erasing Square");
      }
 }
 public class Solution {
      public static void main(String[] args) {

           Shape c = new Circle();
           Shape t = new Triangle();
           Shape s = new Square();
           c.draw();
           c.erase();
           t.draw();
           t.erase();
           s.draw();
           s.erase();
      }
}
```

4. Elaborate /Explain /Summarised/Illustrate the Features of Java Programming.

Answer: list All the feature of java . Explain them in details.

Major Features of Java Programming Language

- Simple.
  Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Object-Oriented.
  Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.
- Platform Independent.
  Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.
- Portable.
  Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.
- Secure.
  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

  Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.
- 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. Threads are important for multi-media, Web applications, etc.
- Robust
  The English mining of Robust is strong. Java is robust because:
  It uses strong memory management.
  There is a lack of pointers that avoids security problems.

5. **Elaborate /Explain /Summarised/Illustrate the Object oriented programing or Java Programming.**

Answer: list All the concept of oop . Explain them in details.

OOPS concepts are as follows:

1. Class
   Collection of objects **is called class. It is a logical entity.**
   A class can also be defined as a blueprint from which you can create an individual object.
   Class doesn't consume any space.

2. Object
   An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

- Abstraction
  ***Hiding internal details and showing functionality* is known as abstraction. For example phone call, we don't know the internal processing.**
  In Java, we use abstract class and interface to achieve abstraction.
- Encapsulation
  ***Binding (or wrapping) code and data together into a single unit are known as encapsulation*. For example, a capsule, it is wrapped with different medicines.**
  A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

- Inheritance
  ***When one object acquires all the properties and behaviors of a parent object*, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.**
- Polymorphism
  If *one task is performed in different ways*, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.
  In Java, we use method overloading and method overriding to achieve polymorphism.

  Compile-time polymorphism
  In this, the methods are overloaded by changing the number of parameters in the argument. The first add method includes two parameters (a and b) and it returns the addition of two numbers given in the argument whereas the second method of add includes three parameters (a, b, and c)
  Runtime polymorphism
  Runtime polymorphism is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

6. Analyse or Explain the Concept of method overloading and the concept of method overriding in java.
Ans:-

Overriding	Overloading
Implements "runtime polymorphism"	Implements "compile time polymorphism"
The method call is determined at runtime based on the object type	The method call is determined at compile time

Overriding	Overloading
Occurs between superclass and subclass	Occurs between the methods in the same class
Have the same signature (name and method arguments)	Have the same name, but the parameters are different
On error, the effect will be visible at runtime	On error, it can be caught at compile time

Example: Method Overriding

```
import java.io.*;

class Animal {

    void eat()
    {
        System.out.println("eat() method of base class");
        System.out.println("eating.");
    }
}

class Dog extends Animal {

    void eat()
    {
        System.out.println("eat() method of derived class");
        System.out.println("Dog is eating.");
    }
}

class MethodOverridingEx {

    public static void main(String args[])
    {
        Dog d1 = new Dog();
        Animal a1 = new Animal();

        d1.eat();
        a1.eat();

        Animal animal = new Dog();
        // eat() method of animal class is overridden by
        // base class eat()
        animal.eat();
    }
}
```

Example of Method Overloading:

```
import java.io.*;

class MethodOverloadingEx {

    static int add(int a, int b)
    {
      return a + b;
    }

    static int add(int a, int b, int c)
    {
        return a + b + c;
    }

    public static void main(String args[])
    {
          System.out.println("add() with 2 parameters");
        System.out.println(add(4, 6));

          System.out.println("add() with 3 parameters");
        System.out.println(add(4, 6, 7));
    }
}
```

7. **Develop a complete Java application to prompt the user for the double radius of a sphere, and call method SphereVolume to calculate and display the volume of the sphere. Use the following statement to calculate the volume:**

Answer:

```
import java.util.Scanner;
class Sphere
{

   public static void main(String args[])
    {

                 Scanner s= new Scanner(System.in);
          System.out.println("Enter the radius of sphere:");
          double r=s.nextDouble();


              double volume= Sphere. SphereVolume (r);

           System.out.println("Volume Of Sphere is:" +volume);
    }

   public static double SphereVolume (double r)
  {

          double volume= (4*22*r*r*r)/(3*7);

           return volume;
    }
```

8. A class called circle is designed as shown in the following class diagram. It contains:
• Two private instance variables: radius (of the type double) and color (of the type String), with default value of 1.0 and "red", respectively.
• Two constructors - a default constructor with no argument and a constructor which takes a double argument for radius.
Two public methods: getRadius() and getArea(),which return the radius and area of this instance, respectively

9. Write a Java program using Objects and Classes concepts in Object Oriented Programming. Define the Rectangle class that will have: Two member variables width and height , A no-arg constructor that will create the default rectangle with width = 1 and height =1. ♣ A parameterized constructor that will create a rectangle with the specified x, y, height and width. ♣ A method getArea() that will return the area of the rectangle. ♣ A method getPerimeter() that will return the perimeter of the rectangle.

```
class Rectangle{

            double length, width;
            Rectangle()
            {
                length = 1;
                width = 1;
            }
            Rectangle(double length, double width)
            {
                this.length = length;
                this.width  = width;
            }
            double getArea()
            {
                return (length * width);
            }
            double getPerimeter()
            {
                return (2 * (length + width));
            }
        }
        public class TestRectangle {
            public static void main(String[] args) {

                Rectangle rect1 = new Rectangle();
                Rectangle rect2= new Rectangle(15.0,8.0);

                System.out.println("Area of first object="+rect1.getArea());
                System.out.println("Perimeter of first 
object="+rect1.getPerimeter());
                System.out.println("Area of second object="+rect2.getArea());
                System.out.println("Perimeter of second 
object="+rect2.getPerimeter());
           }

        }
```
10. Identify class and object from the list And also write down the relationship between class and objects with a minimum 4 attributes:
1)CAR, Zen, Suzuki, Duster, Swift
2)Flower, Tree, Rose, jasmine
3)Dog, Animal, Horse, Tiger
4)Samsung, Phone, Nokia

11. Constructors are essential in OOP. Analyze what would happen if constructors are not there? Mention its types and characteristics.
ANS:-
Need of Constructor
Think of a Box. If we talk about a box class then it will have some class variables (say length, breadth, and height). But when it comes to creating its object(i.e Box will now exist in the computer's memory), then can a box be there with no value defined for its dimensions? The answer is No.
So constructors are used to assign values to the class variables at the time of object creation, either explicitly done by the programmer or by Java itself (default constructor).
Types of Constructors in Java
the types of the constructor, so primarily there are two types of constructors in java:

No-argument constructor
Parameterized Constructor
Default Constructor

CHARACTERISTICS:-
Constructors must have the same name as the class within which it is defined it is not necessary for the method in Java.
Constructors do not return any type while method(s) have the return type or void if does not return any value.
Constructors are called only once at the time of Object creation while method(s) can be called any number of times.

12 Sum Of Two Integer Numbers Using Command Line Arguments In Java
13. To find loan using cmd
14. Create a class Rectangle. The class has attributes length and width, each of which defaults to 1. It has methods that calculate the perimeter and the area of the rectangle. It has set and get methods for both length and width. The set methods should verify that length and width are each floating-point numbers larger than 0.0 and less than 20.0.

Unit 2

1. **Discuss the concept of String Buffer class and String Builder Class and StringTokanizer class in Java with all methods available in it. (Explain with separate code.)**

1. **5.** Discuss the concept of String Buffer class and String Builder Class in Java with all methods available in it.(Explain withseparate code.)

Ans:-

Java StringBuffer Class

Java StringBuffer class is used to create mutable (modifiable) String objects. The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed.

A String that can be modified or changed is known as mutable String. StringBuffer and StringBuilder classes are used for creating mutable strings.

1) StringBuffer Class append() Method
2) StringBuffer insert() Method
3) StringBuffer replace() Method
4) StringBuffer delete() Method
5) StringBuffer reverse() Method

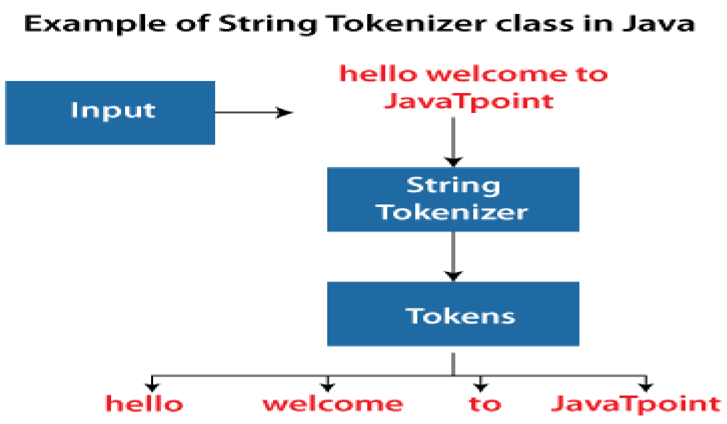
```
class Stringbuf{
public static void main(String args[]){
StringBuffer sb=new StringBuffer("Hello");
System.out.println(sb.append("Java"));
System.out.println(sb.insert(1,"Java"));
System.out.println(sb.replace(1,3,"Java"));
System.out.println(sb.delete(1,3));
System.out.println(sb.reverse());
}
}
```

OUTPUT:-

```
HelloJava
HJavaelloJava
HJavavaelloJava
HvavaelloJava
avaJolleavavH
```

StringTokenizer in Java

The **java.util.StringTokenizer** class allows you to break a String into tokens. It is simple way to break a String. It is a legacy class of Java.


Example of String Tokenizer class in Java
Input
hello welcome to JavaTpoint
String Tokenizer
Tokens
hello
welcome
to
JavaTpoint

```
import java.util.StringTokenizer;
public class Simple{
 public static void main(String args[]){
   StringTokenizer st = new StringTokenizer("my name is khan"," ");
     while (st.hasMoreTokens()) {
         System.out.println(st.nextToken());
     }
            while (st.hasMoreTokens())
              {
                  System.out.println(st.nextElement());
              }

     System.out.println("Total number of Tokens: "+st.countTokens());
   }
}
```

Output:

```
my
name
is
khan
```

2. Discuss the different ways to create string objects.

Java String

In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string.

Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class is used to create a string object.

How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

1) String Literal

Java String literal is created by using double quotes. For Example:

```
String s="welcome";
```

Why Java uses the concept of String literal?

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

2) By new keyword

```
String s=new String("Welcome");
public class StringExample{
public static void main(String args[]){
String s1="java";
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);
String s3=new String("example");
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```

3. Assume a string containing n numbers of words. The task is to reverses all the characters of the string using StringBuilder class. Input: "Sweets are so Sweet" Output: "teewS os era stews”

```
class StringBuilderExample5{
public static void main(String args[]){
StringBuilder sb=new StringBuilder("Sweets are so Sweet");
System.out.println(sb.reverse());
}
}
```

4. Illustrate different Approaches of importing Package in java.

There are three ways to access the package from outside the package.

- import package.*;
- import package.classname;
- fully qualified name.

*1) Using packagename.**

If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.*;

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

```
Output:Hello
```

2) Using packagename.classname

If you import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

```
//save by A.java

package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.A;

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

```
Output:Hello
```

3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.
It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

```java
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
class B{
  public static void main(String args[]){
   pack.A obj = new pack.A();//using fully qualified name
   obj.msg();
  }
}
```

```
Output:Hello
```

7. Given a string str and a positive integer k, the task is to develop a Java program to print the first k characters of the string. If the length of the string is less than k then print the string as it is.

   CODE:-

```java
class GFG {
    public static String
      firstKCharacters(String str, int k){
        if (str == null || str.isEmpty())
            return null;
        if (str.length() > k)
            return str.substring(0, k);

        else
            return str;
    }
    public static void main(String args[])
    {
        String str = "GeeksForGeeks";
        int k = 5;
        System.out.println(firstKCharacters(str,k));
    }
}
```

Input: *str = “GeeksForGeeks”, k = 5*
Output: *Geeks*

7. Assume a string containing n numbers of words. The task is to swap the corner words of the string and reverses all the middle characters of the string. Input: "Hello this is the GFG user" Output: "user GFG eth si siht Hello"

```java
import java.io.*;
import java.util.*;
public class fg {
    static void swap(String m, int length)
    {
        String msg[] = m.split(" ");
        String temp = msg[0];
        msg[0] = msg[msg.length - 1];
        msg[msg.length - 1] = temp;
        String mid = "";

        for (int i = msg.length - 2; i >= 1; --i) {
            String temp_s = msg[i];
            for (int j = temp_s.length() - 1; j >= 0; --j) {
                mid += temp_s.charAt(j);
            }
            mid += " ";
        }
        System.out.print(msg[0] + " " + mid + " "
                         + msg[msg.length - 1]);
    }
    public static void main(String[] args)
    {
        String m = "Hello this is the GFG user";
        int length = m.length();
        swap(m, length);
    }
}
```

8. Develop a program that takes your full name as input and displays the abbreviations of the first and middle names except the last name which is displayed as it is. For example, if your name is Robert Brett Roser, then the output should be R.B.Roser

```java
import java.util.*;
class Ans{
  public static void main(String[] args){
    Scanner s = new Scanner(System.in);
    System.out.println("Enter your full name");
    String st = s.nextLine();
    String sr = "";
```

```
        sr = sr+st.charAt(0);
        sr = sr+". ";
        for (int i = 0; i<st.length();i++){
          if(st.charAt(i) == ' ' && st.charAt(i+1)!=' ' && i+1<st.length()){
            sr = (sr+st.charAt(i+1)).toUpperCase();
            sr = sr+". ";
          }
        }
        String last_wrd = "";
        //to get the last word
        for(int i = st.length()-1;i>=0;i--){
          if(st.charAt(i) == ' '){

            last_wrd = st.substring(i+2);
            break;
          }
        }
        //to remove last ". "
        sr = sr.substring(0,sr.length()-2);
        sr = sr+last_wrd;
        System.out.println(sr);
      }
    }
```

9. Consider the strings s1 and s2 given as follows: StringBuilder s1 = new StringBuilder("CSE"); StringBuilder s2 = new StringBuilder("ML"); Show the value of s1 after each of the following statements. Assume that the statements are independent. Write the code for each function separately. a. s1.append(" is fun"); b. s1.append(s2); c. s2.charAt(int 2); d. s2.subString(2,4,"department");

```
public class String {
    public static void main(String []args){
        StringBuilder s1 = new StringBuilder("CSE");
        StringBuilder s2 = new StringBuilder("ML");
        System.out.println(s1.append("is fun"));
        System.out.println(s1.append(s2));
        System.out.println(s2.charAt(2));
        System.out.println(s2.substring(2,4,"department"));

    }

}
Output:-
```

10. You have created a class with two instance variables. You need to initialize the variables automatically when a class is initialized. Identify the method that you will use you to achieve this. In addition, describe the characteristics of this method.

11. Elaborate the given statement "String is immutable"

- This is how a String works:
- `String str = "knowledge";`
- This, as usual, creates a string containing "knowledge" and assigns it to reference str. Simple enough? Let us perform some more functions:
- `// assigns a new reference to the`
- `// same string "knowledge"`
- `String s = str;`

These are some more reasons for making String immutable in Java. These are:

- The String pool cannot be possible if String is not immutable in Java. A lot of heap space is saved by JRE. The same string variable can be referred to by more than one string variable in the pool. String interning can also not be possible if the String would not be immutable.
- If we don't make the String immutable, it will pose a serious security threat to the application. For example, database usernames, passwords are passed as strings to receive database connections. The socket programming host and port descriptions are also passed as strings. The String is immutable, so its value cannot be changed. If the String doesn't remain immutable, any hacker can cause a security issue in the application by changing the reference value.
- The String is safe for multithreading because of its immutableness. Different threads can access a single "String instance". It removes the synchronization for thread safety because we make strings thread-safe implicitly.
- Immutability gives the security of loading the correct class by Classloader. For example, suppose we have an instance where we try to load java.sql.Connection class but the changes in the referenced value to the myhacked.Connection class does unwanted things to our database.

12. Construct the code using given statement Create a class First in learnjava package that access it in another class Second by using import keyword.

13. Logic for reverse string

14. Interpretation for strring compare,append,length

Unit – 4

Unit 4

1. **Difference between Abstract class and interface class.**

Abstract class	Interface
1) Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
2) Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
3) Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
4) Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
5) The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
6) An abstract class can extend another Java class and implement multiple Java interfaces.	An interface can extend another Java interface only.

7) An abstract class can be extended using keyword "extends".	**An interface can be implemented using keyword "implements".**
8) A Java abstract class can have class members like private, protected, etc.	**Members of a Java interface are public by default.**

9)Example:
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

2. Analyze and differentiate the various methods used to implement Thread in java. Justify your answer with sample code.

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

Commonly used methods of Thread class:

1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.

3. **public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long miliseconds):** waits for a thread to die for the specified miliseconds.
6. **public int getPriority():** returns the priority of the thread.
7. **public int setPriority(int priority):** changes the priority of the thread.
8. **public String getName():** returns the name of the thread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread currentThread():** returns the reference of currently executing thread.
11. **public int getId():** returns the id of the thread.
12. **public Thread.State getState():** returns the state of the thread.
13. **public boolean isAlive():** tests if the thread is alive.
14. **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
15. **public void suspend():** is used to suspend the thread(depricated).
16. **public void resume():** is used to resume the suspended thread(depricated).
17. **public void stop():** is used to stop the thread(depricated).
18. **public boolean isDaemon():** tests if the thread is a daemon thread.
19. **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
20. **public void interrupt():** interrupts the thread.
21. **public boolean isInterrupted():** tests if the thread has been interrupted.
22. **public static boolean interrupted():** tests if the current thread has been interrupted.

Runnable interface:

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

1. **public void run():** is used to perform action for a thread.

Starting a thread:

The **start() method** of Thread class is used to start a newly created thread. It performs the following tasks:

- A new thread starts(with new callstack).
- The thread moves from New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

1) Java Thread Example by extending Thread class

FileName: Multi.java

```
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 }
}
```

Output:

```
thread is running...
```

2) Java Thread Example by implementing Runnable interface

FileName: Multi3.java

```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);   // Using the constructor Thread(Runnable r)
t1.start();
 }
}
```

Output:

```
thread is running...
```

3. Analyze the mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. Explain methods available to handle this situation.

Inter-thread communication in Java is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.

It is implemented by following methods of **Object class**:

- wait()
- notify()
- notifyAll()

1) wait() method

The wait() method causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

Method	Description
public final void wait()throws InterruptedException	It waits until object is notified.
public final void wait(long timeout)throws InterruptedException	It waits for the specified amount of time.

2) notify() method

The notify() method wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation.

Syntax:

```
public final void notify()
```

3) notifyAll() method

Wakes up all threads that are waiting on this object's monitor.

Syntax:

```
public final void notifyAll()
class Customer{
int amount=10000;

synchronized void withdraw(int amount){
System.out.println("going to withdraw...");

if(this.amount<amount){
System.out.println("Less balance; waiting for deposit...");
try{wait();}catch(Exception e){}
}
this.amount-=amount;
System.out.println("withdraw completed...");
}

synchronized void deposit(int amount){
System.out.println("going to deposit...");
this.amount+=amount;
System.out.println("deposit completed... ");
notify();
}
}

class Test{
public static void main(String args[]){
final Customer c=new Customer();
new Thread(){
public void run(){c.withdraw(15000);}
}.start();
new Thread(){
public void run(){c.deposit(10000);}
}.start();

}}
```

Output:

```
going to withdraw...
Less balance; waiting for deposit...
going to deposit...
deposit completed...
withdraw completed
```

4. Analyze the various types of Synchronization used to overcome the problem in Multithreading in java. Minimum two types should be explain in detail.

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.

Types of synchronization:

1. Method level synchronization
2. Block level synchronization
3. Static block synchronization

Method level synchronization

Synchronized methods enables a simple strategy for preventing the thread interference and memory consistency errors. If a Object is visible to more than one threads, all reads or writes to that Object's fields are done through the synchronized method.

It is not possible for two invocations for synchronized methods to interleave. If one thread is executing the synchronized method, all others thread that invoke synchronized method on the same Object will have to wait until first thread is done with the Object.

Example:

```
Class Table {
Public synchronized void printTable(int n)
{
   for(int i=1; i<=10;i++) {
      S.o.P(n*i);
     }
  }
}
Class thread1 extends Thread {
   Table t;
    thread1( Table t);
    {
        this.t= t;
}
Public void run() {
    t.printTable(5);
   }
}
Class thread2 extends Thread {
Table t;
thread2( Table t) {
this.t=t;
}
Public void run() {
   t.printTable(3);
}
}
Class main
{
p.s.v.m(String[] args) {
Table obj = new Table();
Thrad1 t1=new thead1(obj);
Thread2 t2=new thread2(obj);
t1.start();
t2.start();
}
}
```

Output:

```
5
10
15
20
25
30
35
40
45
50
3
6
9
12
15
18
```

```
21
24
27
30
```
Block level synchronization

If we only need to execute some subsequent lines of code not all lines (instructions) of code within a method, then we should synchronize only block of the code within which required instructions are exists.

For example, lets suppose there is a method that contains 100 lines of code but there are only 10 lines (one after one) of code which contain critical section of code i.e. these lines can modify (change) the Object's state. So we only need to synchronize these 10 lines of code method to avoid any modification in state of the Object and to ensure that other threads can execute rest of the lines within the same method without any interruption.

Example:

```
Class message {
Public void show(String name) {
  Synchronized (this) {
      For (int i=1; i<=3; i++) {
          S.O.P("how are you" +name);
           }
      }
  }
}
Class thread1 extends Thread {
     Message m;
     String name;
     thread1(message m,String name) {
this.m=m;
this.name=name;
}
Public void run() {
  m.show( name);
  }
}
Class synBlock {
   P.S.V.M (String[] args) {
        Message m = new message();
        Thread1 t1= new thread1(m," arun");
         Thread1 t2= new thread1(m," ajay");
           t1.start();
           t2.start();
    }
}
```
Output:
```
How are you arun
How are you arun
How are you arun
How are you ajay
How are you ajay
How are you ajay
```
5. Discuss in detail the life cycle of Thread.

or

14.Identify different phases of thread lifecycle and tell how transitions are done among the phases?

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

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.
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**.

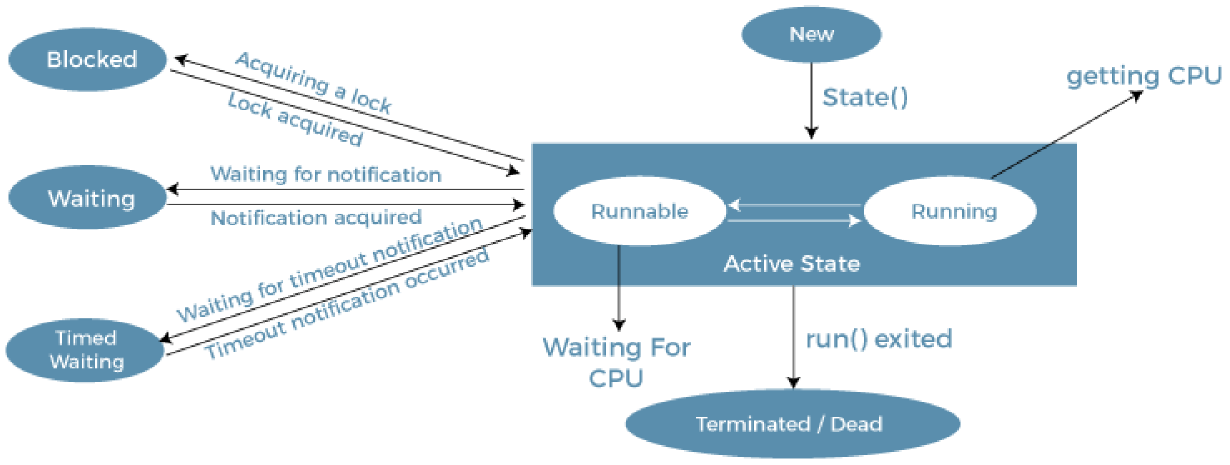
- **Runnable:** A thread, that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time. It is the duty of the thread scheduler to provide the thread time to run, i.e., moving the thread the running state.
  A program implementing multithreading acquires a fixed slice of time to each individual thread. Each and every thread runs for a short span of time and when that allocated time slice is over, the thread voluntarily gives up the CPU to the other thread, so that the other threads can also run for their slice of time. Whenever such a scenario occurs, all those threads that are willing to run, waiting for their turn to run, lie in the runnable state. In the runnable state, there is a queue where the threads lie.
- **Running:** When the thread gets the CPU, it moves from the runnable to the running state. Generally, the most common change in the state of a thread is from runnable to running and again back to runnable.

Blocked or 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. A thread in the blocked state is unable to perform any execution and thus never consume any cycle of the Central Processing Unit (CPU). Hence, we can say that thread A remains idle until the thread scheduler reactivates thread A, which is in the waiting or blocked state.
When the main thread invokes the join() method then, it is said that the main thread is in the waiting state. The main thread then waits for the child threads to complete their tasks. When the child threads complete their job, a notification is sent to the main thread, which again moves the thread from waiting to the active state.
If there are a lot of threads in the waiting or blocked state, then it is the duty of the thread scheduler to determine which thread to choose and which one to reject, and the chosen thread is then given the opportunity to run.
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.
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.

A terminated thread means the thread is no more in the system. In other words, the thread is dead, and there is no way one can respawn (active after kill) the dead thread.
The following diagram shows the different states involved in the life cycle of a thread.



Life Cycle of a Thread

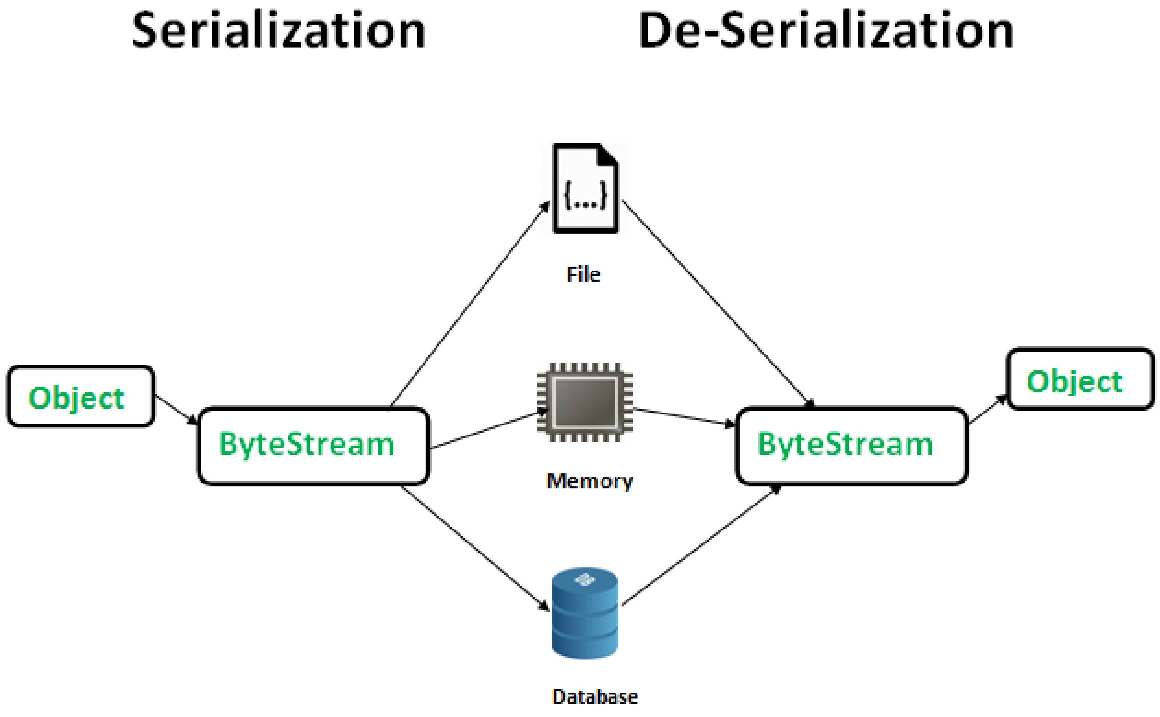
6. Create two ArrayList one is of string type having 5 element and another is of integer type having 5 elements. Sort both the ArrayList by using Collection . Sort() method. Develop a program for the same.

```
import java.util.*;
class GFG {
	// Main driver method
	public static void main(String[] args)
	{
		// Define an objects of ArrayList class
		ArrayList<String> list1 = new ArrayList<String>();
		ArrayList<Integer> list2 = new ArrayList<Integer>();
		// Adding elements to the ArrayList
		list1.add("India");
		list1.add("Pakistan");
		list1.add("Srilanka");
		list1.add("USA");
		list1.add("Japan");
	    list2.add(410);
	    list2.add(250);
	    list2.add(144);
	    list2.add(967);
	    list2.add(289);
```

```
                    System.out.println("Before Sorting : " + list1);
                    Collections.sort(list1);
                    System.out.println("After Sorting : " + list1);
                    System.out.println("Before Sorting : " + list2);
                    Collections.sort(list2);
                    System.out.println("After Sorting : " + list2);
          }
}
```

```
Output;
Before Sorting : [India, Pakistan, Srilanka, USA, Japan]
After Sorting : [India, Japan, Pakistan, Srilanka, USA]
Before Sorting : [410, 250, 144, 967, 289]
After Sorting : [967, 410, 289, 250, 144]
```

7. Discuss the Serialization and Deserialization with sample code.
Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.



```
Example:
//serialization
Import java.io.*;
Public class serialDemo
{
p.s.v.m(String [] args) throws Exception {
```

```
Save obj = new Save();
Try {
	Obj.name = “harry”;
	Obj.address = “nagpur”;
FileOutputStream fos = new FileOutputStream(“./obj.txt”);
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(obj);
s.o.p(“serialized data is stored in obj.txt file”);
//desirialization
FileInputStream fis= new Fileinputstream(“./obj.txt”);
ObjectOutputStream ois = new ObejctInputStream(fis);
Save obj1= (Save)ois.readObject();
sop(“value of obj1” +obj1.name);
sop(“value of obj1” +obj1.address);
}
Catch (Exception e) { }
}
}
Class Save implements Serialization
{
String name;
String address;
}
```

Output:

```
Serialized data is stored in obj.txt file
Value of obj1 harry
Value of obj1 nagpur
```

9 Savita is given an assignment where there is a class to be used To demonstrate multiple threads, rename each thread with new names, set and get the priority of each thread. How can you help Savita to develop a code in order to solve the given assignment?

```java
// Importing the required classes
import java.lang.*;

public class ThreadPriorityExample extends Thread
{

// Method 1
// Whenever the start() method is called by a thread
// the run() method is invoked
public void run()
{
// the print statement
System.out.println("Inside the run() method");
}

// the main method
public static void main(String argvs[])
{
// Creating threads with the help of ThreadPriorityExample class
```

```java
ThreadPriorityExample th1 = new ThreadPriorityExample();
ThreadPriorityExample th2 = new ThreadPriorityExample();
ThreadPriorityExample th3 = new ThreadPriorityExample();

// We did not mention the priority of the thread.
// Therefore, the priorities of the thread is 5, the default value

// 1st Thread
// Displaying the priority of the thread
// using the getPriority() method
System.out.println("Priority of the thread th1 is : " + th1.getPriority());

// 2nd Thread
// Display the priority of the thread
System.out.println("Priority of the thread th2 is : " + th2.getPriority());

// 3rd Thread
// // Display the priority of the thread
System.out.println("Priority of the thread th2 is : " + th2.getPriority());

// Setting priorities of above threads by
// passing integer arguments
th1.setPriority(6);
th2.setPriority(3);
th3.setPriority(9);

// 6
System.out.println("Priority of the thread th1 is : " + th1.getPriority());

// 3
System.out.println("Priority of the thread th2 is : " + th2.getPriority());

// 9
System.out.println("Priority of the thread th3 is : " + th3.getPriority());

// Main thread

// Displaying name of the currently executing thread
System.out.println("Currently Executing The Thread : " +
Thread.currentThread().getName());

System.out.println("Priority of the main thread is : " +
Thread.currentThread().getPriority());

// Priority of the main thread is 10 now
Thread.currentThread().setPriority(10);

System.out.println("Priority of the main thread is : " +
Thread.currentThread().getPriority());
```

```
}
}
```

Output:

```
Priority of the thread th1 is : 5
Priority of the thread th2 is : 5
Priority of the thread th2 is : 5
Priority of the thread th1 is : 6
Priority of the thread th2 is : 3
Priority of the thread th3 is : 9
Currently Executing The Thread : main
Priority of the main thread is : 5
Priority of the main thread is : 10
```

10 Suppose you have created a class BulletFight which contains a member variable bullets that is initialized to 30 and two methods shout() and reload(). The shout () method shout the number of bullets passed to it until the bullets become 0 and when bullets become 0 it invokes the wait() method which caused the calling thread to sleep and release the lock on the object while reload() method increased the bullets by 30 and invokes the notify() method which wakes up the waiting thread. Design a code to implement the above task.

11 Differentiate Multithreading from Multitasking

Features	**Multitasking**	**Multithreading**
Definition	When a single processor does many jobs (program, threads, process, task) at the same time, it is referred to as multitasking.	Multitasking occurs when the CPU does many tasks, such as a program, process, task, or thread.
Basic	A CPU may perform multiple tasks at once by using the multitasking method.	Multithreading allows a CPU to generate numerous threads from a job and process them all at the same time.

Resources and Memory	The system must assign different resources and memory to separate programs that are running concurrently in multitasking.	The system assigns a single memory block to each process.
Switching	CPU switches between programs frequently.	CPU switches between the threads frequently.
Speed of Execution	It is comparatively slower in execution.	It is comparatively faster in execution.
Working	A user may easily run several jobs off of their CPU at once.	A CPU has the ability to split a single program into several threads to improve its functionality and efficiency.
Process Termination	The process of terminating a task takes comparatively more time.	It requires considerably less time to end a process.

12. How is Multithreading implemented in Java? Differentiate between the Thread class and Runnable interface for creating a Thread?

Same as que2

Thread

- It is a class.
- It can be used to create a thread.
- It has multiple methods such as ‘start’ and ‘run’.
- It requires more memory space.
- Since multiple inheritance is not allowed in Java, hence, after a class extends the Thread class, it can’t extend to any other class.
- Every thread creates a unique object and associates with it.

Runnable

- It is a functional interface.
- It can be used to create a thread.
- It has a single abstract method 'run'.
- It requires less memory space.
- When a class implements the 'runnable' interface, the class can extend to other classes.
- Multiple threads can share the same objects.

13.What would happen if we call the start method more than once on the same thread object?

15.Create a class that demonstrates the execution of two or more threads in a strict sequence. The code developed is expected to print some numbers. Do the necessary use of 'synchronized' keyword, so that, the program prints the output in the following order: -----------------OUTPUT------------------- 5 10 15 20 25 100 200 300 400 500

```
class Execute{
synchronized void print(int n){
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){
        System.out.println(e);
     }
   }
 }
 }
 class Thread1 extends Thread{
            Execute t;
            Thread1(Execute t){
                        this.t=t;
            }
            public void run(){
                        t.print(5);
            }
}

class Thread2 extends Thread{
            Execute t;
            Thread2(Execute t){
                        this.t=t;
            }
        public void run(){
                t.print(100);
        }
}
public class Question64{
    public static void main(String args[]){
                Execute obj = new Execute();//only one object
                Thread1 t1=new Thread1(obj);
                Thread2 t2=new Thread2(obj);
                t1.start();
                t2.start();
        }
}
```

16.Explain setpriority,getpriority,sleep(),wait(),IsAlive(),resume(),Suspend()

setPriority(int newPriority): java.lang.Thread.setPriority() method changes the priority of thread to the value newPriority. This method throws IllegalArgumentException if value of parameter newPriority goes beyond minimum(1) and maximum(10) limit.
getPriority(): java.lang.Thread.getPriority() method returns priority of given thread.

The sleep() Method Syntax:

Following are the syntax of the sleep() method.

1. **public static void** sleep(**long** mls) **throws** InterruptedException
2. **public static void** sleep(**long** mls, **int** n) **throws** InterruptedException

The method sleep() with the one parameter is the native method, and the implementation of the native method is accomplished in another programming language. The other methods having the two parameters are not the native method.

Parameters:

The following are the parameters used in the sleep() method.
mls: The time in milliseconds is represented by the parameter mls. The duration for which the thread will sleep is given by the method sleep().
n: It shows the additional time up to which the programmer or developer wants the thread to be in the sleeping state. The range of n is from 0 to 999999.
wait() method is a part of java.lang.Object class. When wait() method is called, the calling thread stops its execution until notify() or notifyAll() method is invoked by some other Thread.
Syntax:

```
public final void wait() throws InterruptedException
```

The **isAlive()** method of thread class tests if the thread is alive. A thread is considered alive when the start() method of thread class has been called and the thread is not yet dead. This method returns true if the thread is still running and not finished.

Syntax

1. **public final boolean** isAlive()

The **resume()** method of thread class is only used with suspend() method. This method is used to resume a thread which was suspended using suspend() method. This method allows the suspended thread to start again.

Syntax

```
1. public final void resume()
```

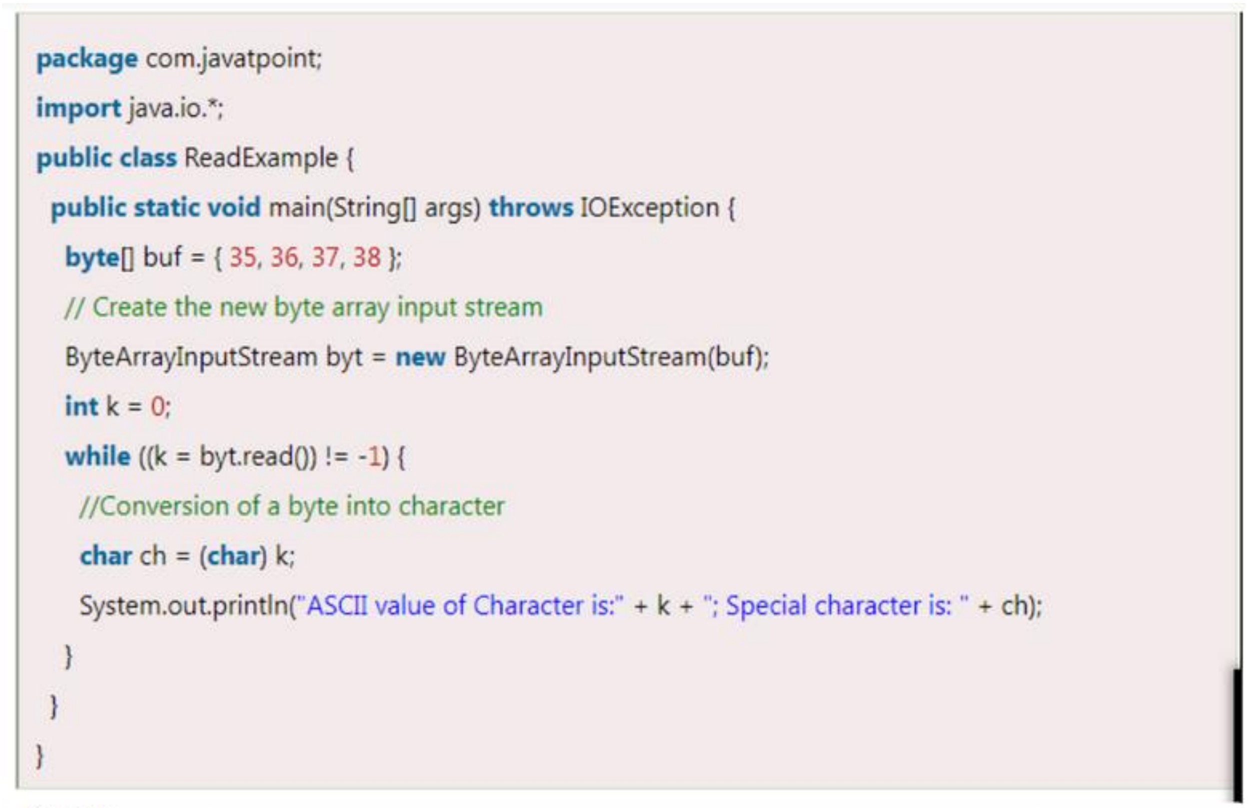
The **suspend()** method of thread class puts the thread from running to waiting state. This method is used if you want to stop the thread execution and start it again when a certain event occurs. This method allows a thread to temporarily cease execution. The suspended thread can be resumed using the resume() method.

Syntax

```
1. public final void suspend()
```

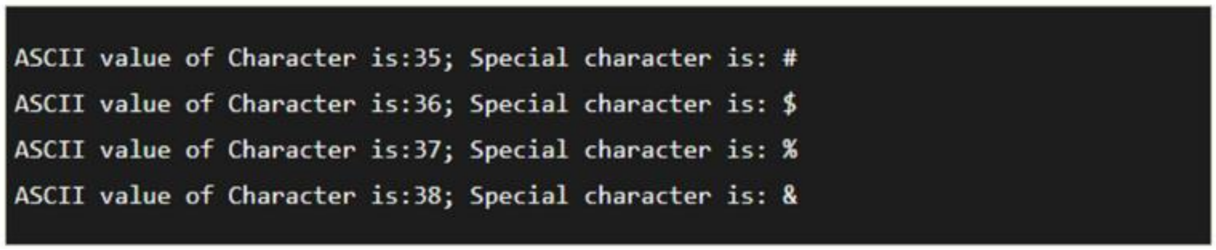
Unit-5

In an interview Roshni has been asked to evaluate the ASCII value of each character using the appropriate byte-oriented streams in java. An interviewer informed strictly to accept the inputs through the byte array input stream which will be an array of integers only. The interviewer also provided the expected output as – ASCII value of character is : 35 ; Special Character is : # ASCII value of character is : 36 ; Special Character is : $ ASCII value of character is : 37 ; Special Character is : % ASCII value of character is : 38 ; Special Character is : &



```java
package com.javatpoint;
import java.io.*;
public class ReadExample {
 public static void main(String[] args) throws IOException {
   byte[] buf = { 35, 36, 37, 38 };
   // Create the new byte array input stream
   ByteArrayInputStream byt = new ByteArrayInputStream(buf);
   int k = 0;
   while ((k = byt.read()) != -1) {
    //Conversion of a byte into character
    char ch = (char) k;
    System.out.println("ASCII value of Character is:" + k + "; Special character is: " + ch);
   }
 }
}
```

Output:



```
ASCII value of Character is:35; Special character is: #
ASCII value of Character is:36; Special character is: $
ASCII value of Character is:37; Special character is: %
ASCII value of Character is:38; Special character is: &
```

Select an appropriate byte-oriented stream java to read data “Welcome to Java Programming” from text file input.txt Select an appropriate

character-oriented stream in java to write this data into text files named as output.txt.

Ans:

```
import java.io.FileInputStream;
public class DataStreamExample {
public static void main(String args[]){
try{
FileInputStream fin=new FileInputStream("D:\\testout.txt");
FileOutputStream fout = new FileOutputStream(“output.txt”);
int i=0;
while((i=fin.read())!=-1){
fout.write(i);
System.out.print((char)i);
}
fin.close();
fout.close();
}catch(Exception e){System.out.println(e);}
}
}
```

Given a list of numbers, square them and filter the numbers which are greater 10000 and then find average of them (Java 8 APIs only).

Ans:

```
import java.util.*;
import java.util.stream.*;
public class Practical8 {
        public static void main(String[] args) {
Integer[] arr = new Integer[]{10,20,30,40,300};
List <Integer> list = Arrays.asList(arr);
OptionalDouble average = list.stream().mapToInt(n-> n*n).filter(n-
>n>1000).average();
List <Integer> a = list.stream().map(i-> i*i).collect(Collectors.toList());;
System.out.println(a);
System.out.println(average);

}
}
```

Design a sample code to explore the process of passing lambda expressions as arguments in Java?

```
import java.util.ArrayList;
import java.util.Arrays;

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

    // create an ArrayList
    ArrayList<String> languages = new ArrayList<>(Arrays.asList("java",
"python"));
    System.out.println("ArrayList: " + languages);

    // call the foEach() method
    // pass lambda as argument fo forEach()
    // reverse each element of ArrayList
    System.out.print("Reversed ArrayList: ");
    languages.forEach((e) -> {

      // body of lambda expression
      String result = "";
      for (int i = e.length()-1; i >= 0 ; i--)
      result += e.charAt(i);
      System.out.print(result + ", ");
    });

  }
}
```

Elaborate the statement: Can you convert an array to Stream? How?

Create two linked hash sets {"George","Jim", "John", "Blake", "Kevin", "Michael"} and {"George", "Katie", "Kevin", "Michelle", "Ryan"} and find their union, difference, and intersection. (You can clone the sets to preserve the original sets from being changed by these set methods.)

Summarize, what is the parallel Stream? How can you get a parallel stream from a List?

ANS:

Java Parallel Streams is a feature of Java 8 and higher, meant for utilizing multiple cores of the processor. Normally any java code has one stream of processing, where it is executed sequentially. Whereas by using parallel streams, we can divide the code into multiple streams that are executed in parallel on separate cores and the final result is the combination of the individual outcomes. The order of execution, however, is not under our control.
There are two ways we can create which are listed below and described later as follows:
Using parallel() method on a stream
Using parallelStream() on a Collection

Method 1: Using parallel() method on a stream
The **parallel() method** of the **BaseStream interface** returns an equivalent parallel stream. Let us explain how it would work with the help of an example. In the code given below, we create a file object which points to a pre-existent 'txt' file in the system. Then we create a Stream that reads from the text file one line at a time. Then we use the **parallel() method** to print the read file on the console. The order of execution is different for each run, you can observe this in the output. The two outputs given below have different orders of execution.
Example

Java

```
// Java Program to Illustrate Parallel Streams
// Using parallel() method on a Stream

// Importing required classes
```

```
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.util.stream.Stream;

// Main class
// ParallelStreamTest
public class GFG {
// Main driver method
public static void main(String[] args) throws
IOException {

// Creating a File object
File fileName = new
File("M:\\Documents\\Textfile.txt");

// Create a Stream of string type
// using the lines() method to
// read one line at a time from the text file
Stream<String> text = Files.lines(fileName.toPath());

// Creating parallel streams using parallel() method
// later using forEach() to print on console
text.parallel().forEach(System.out::println);
// Closing the Stream
// using close() method
text.close();
```

Method 2: Using parallelStream() on a Collection
The **parallelStream() method** of the Collection interface returns a possible parallel stream with the

collection as the source. Let us explain the working with the help of an example.

Implementation:

In the code given below, we are again using parallel streams but here we are using a List to read from the text file. Therefore, we need the *parallelStream() method.*

```
// Java Program to Illustrate Parallel Streams
// using parallelStream() method on a Stream

// Importing required classes
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.util.*;
// Main class
// ParallelStreamsTest
public class GFG {
// Main driver method
public static void main(String[] args) throws
IOException
{

// Creating a File object
File fileName
= new File("M:\\Documents\\List_Textfile.txt");

// Reading the lines of the text file by
// create a List using readAllLines() method
```

```
List<String> text
= Files.readAllLines(fileName.toPath());

 // Java Program to Illustrate Parallel Streams
 // using parallelStream() method on a Stream

 // Importing required classes
 import java.io.File;
 import java.io.IOException;
 import java.nio.file.Files;
 import java.util.*;

 // Main class
 // ParallelStreamsTest
 public class GFG {

 // Main driver method
 public   static   void   main(String[]   args)throws
 IOException
 {
 // Creating a File object
 File fileName
 = new File("M:\\Documents\\List_Textfile.txt");

 // Reading the lines of the text file by
 // create a List using readAllLines() method
 List<String> text
 = Files.readAllLines(fileName.toPath())
 // Creating parallel streams by creating a List
 //  using readAllLines() method
 text.parallelStream().forEach(System.out::println);
 }
```

Develop a program to get the sum of all numbers present in a list. (Use feature of Java 8)

Ans:

```
import java.util.Arrays;
import java.util.List;

class Main {
   public static void main(String[] args) {
      List<Integer> nums = Arrays.asList(1, 2, 3, 4, 5);

      int sum = nums.stream().mapToInt(i -> i).sum();
      System.out.println(sum);        // 15
   }}
```

Unit 6

Given a list of numbers, square them and filter the numbers which are greater 10000 and then find average of them (Java 8 APIs only).

Ans:

```
import java.util.*;
import java.util.stream.*;
public class Practical8 {
	public static void main(String[] args) {
Integer[] arr = new Integer[]{10,20,30,40,300};
List <Integer> list = Arrays.asList(arr);
OptionalDouble average = list.stream().mapToInt(n-> n*n).filter(n-
>n>1000).average();
List <Integer> a = list.stream().map(i-> i*i).collect(Collectors.toList());;
System.out.println(a);
System.out.println(average);

}
}
```

Design a sample code to explore the process of passing lambda expressions as arguments in Java?

Ans:

```
import java.util.ArrayList;
import java.util.Arrays;

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

    // create an ArrayList
    ArrayList<String> languages = new ArrayList<>(Arrays.asList("java",
"python"));
    System.out.println("ArrayList: " + languages);

    // call the foEach() method
    // pass lambda as argument fo forEach()
    // reverse each element of ArrayList
    System.out.print("Reversed ArrayList: ");
    languages.forEach((e) -> {

      // body of lambda expression
      String result = "";
      for (int i = e.length()-1; i >= 0 ; i--)
      result += e.charAt(i);
      System.out.print(result + ", ");
    });

  }
}
```

Design the following methods that return a lambda expression performing a

specified action:

***The lambda expression must return a number whether it is an odd or even.**

***The lambda expression must return a number whether it is prime or composite.**

Ans:

```
import java.util.*;
interface PerformOperation {
 boolean check(int a);
}
class MyMath {
 public static boolean checker(PerformOperation p, int num) {
  return p.check(num);
 }
   PerformOperation isOdd()
   {
       PerformOperation po = (int a)-> a%2 == 0 ? false : true;
       return po;
   }
   PerformOperation isPrime()
   {
       PerformOperation po = (int a)->
       {
           if(a == 1) return true;
           else
```

```
      {
         for (int i =  2; i < Math.sqrt(a); i++)
             if(a%i == 0) return false;
          return true;
      }
   };
   return po;
}
 public class Solution {
  public static void main(String[] args) throws IOException {
   MyMath ob = new MyMath();
   BufferedReader br = new BufferedReader(new
 InputStreamReader(System.in));
   int T = Integer.parseInt(br.readLine());
   PerformOperation op;
   boolean ret = false;
   String ans = null;
   while (T--> 0) {
    String s = br.readLine().trim();
    StringTokenizer st = new StringTokenizer(s);
    int ch = Integer.parseInt(st.nextToken());
    int num = Integer.parseInt(st.nextToken());
    if (ch == 1) {
     op = ob.isOdd();
     ret = ob.checker(op, num);
```

```
        ans = (ret) ? "ODD" : "EVEN";
       } else if (ch == 2) {
        op = ob.isPrime();
        ret = ob.checker(op, num);
        ans = (ret) ? "PRIME" : "COMPOSITE";
    }
    System.out.println(ans);
    }
    }
```

Identify & Analyze functional interface in Java 8? How can You create a Functional Interface?

Elaborate main characteristics of the Lambda Function?

Ans:

The lambda expressions were introduced in Java 8 and facilitate functional programming. A lambda expression works nicely together only with functional interfaces and we cannot use lambda expressions with more than one abstract method.

Characteristics of Lambda Expression

- Optional Type Declaration − There is no need to declare the type of a parameter. The compiler inferences the same from the value of the parameter.
- Optional Parenthesis around Parameter − There is no need to declare a single parameter in parenthesis. For multiple parameters, parentheses are required.
- Optional Curly Braces − There is no need to use curly braces in the expression body if the body contains a single statement.
- Optional Return Keyword − The compiler automatically returns the value if the body has a single expression to return the value. Curly braces are required to indicate that expression returns a value.

Design a Java 8 program to iterate a Stream using the for Each method.

Ans

Java provides a new method forEach() to iterate the elements. It is defined in Iterable and Stream interface. It is a default method defined in the Iterable interface. Collection classes which extends Iterable interface can use forEach loop to iterate elements.

This method takes a single parameter which is a functional interface. So, you can pass lambda expression as an argument.

forEach() Signature in Iterable Interface

```
default void forEach(Consumer<super T>action)
```

```
import java.util.ArrayList;
import java.util.List;
public class ForEachExample {
    public static void main(String[] args) {
        List<String> gamesList = new ArrayList<String>();
        gamesList.add("Football");
        gamesList.add("Cricket");
        gamesList.add("Chess");
        gamesList.add("Hocky");
                System.out.println("------------Iterating by passing lambda expression--------------");
                gamesList.forEach(games -> System.out.println(games));

            }
        }
```

Develop a program to get the sum of all numbers present in a list. (Use feature of Java 8)

Ans:

```
import java.util.Arrays;
import java.util.List;

class Main {
   public static void main(String[] args) {
      List<Integer> nums = Arrays.asList(1, 2, 3, 4, 5);

      int sum = nums.stream().mapToInt(i -> i).sum();
      System.out.println(sum);        // 15
   }}
```

Explain Normal And Functional interface