22CS302-PROBLEM SOLVING TECHNIQUES III

QUESTION BANK

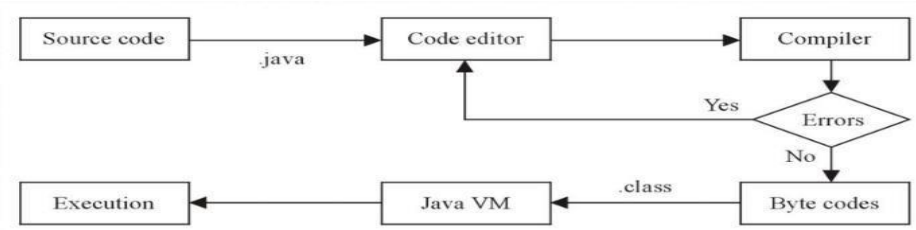
UNIT I

PART A

1.Identify any 3 features of OOP

- Inheritance : extending features from other class
- Data Abstraction : hiding internal implementation details and exposing onlythe required functionalities
- Encapsulation : Combining things under a single block

2. Illustrate the process of JAVA compilation with the significance of JVM



- Java Compiler compiles the java program to byte code which is platformindependent.
- The JVM is specific to platforms. Different environments has different JVM's and hence
- JVM converts the platform independent bytecode to platform specificexecutables.

And Hence the java program written in one environment runs in allenvironments

3 .Summarize the dynamic nature of JAVA arrays and give the declaration syntax

- When an array is declared, only a reference of an array is created.
  - To create or give memory to the array, array should be created like,
  - Type var-name = new type [size];
- E.g.:
  - int marks[]=new int[10];
  - 10*4 bytes of memory is allocated for marks array.

4.Write a JAVA program to give the average of 5 floating point values of an array sized '5'.

```
import java.util.Scanner;

public class AverageCalculator {


public static void main(String[] args) {
// Create an array to store 5 floating-point valuesfloat[] values = new float[5];
```

```
// Scanner to read input from the user Scanner scanner = new
Scanner(System.in);

// Read 5 floating-point values from the userfor (int i = 0; i < 5; i++) {
    System.out.print("Enter value " + (i + 1) + ": ");values[i] = scanner.nextFloat();
}

// Calculate the sum of valuesfloat sum = 0;
for (float value : values) {sum += value;
}

// Calculate the averagefloat average = sum / 5;

// Print the average
System.out.println("Average of the 5 values: " + average);

// Close the scannerscanner.close();
}
}
```

5.What is command line argument.Give an example.

- 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.

```
class CommandLineExample{
public static void main(String args[]){
System.out.println("Your argument is: "+args[0]);
}
}
```

compile by > javac CommandLineExample.java

run by > java CommandLineExample Welcome

6.State the types of constructors

- Default (no-argument) constructor, and
- Parameterized constructor.

7.Briefly discuss the types of Access Specifiers in Java.

- Default : When no access modifier is specified, then it is said to have the default access modifier.
- It cannot be accessed from outside the package and is only accessible within that package.
- Public [**public** keyword] : This has the widest scope among access modifiers and this can be accessed anywhere within the program like within the class, outside the class, within the package and

- outside the package.
- Private [private keyword] : Methods or variables defined as private can be accessed within that class
- and not outside it.

- Protected [protected keyword] : Methods or classes defined as protected can be accessed within that package and outside package by subclasses.

8.Is Java a fully object Oriented Programming Language?Justify your answer.

- No. In Java, we have predefined types as non-objects (primitive types) e.g.: **int, long, bool, float, char**
- which violates OO feature.
- When we declare a class as static then it can be invoked without the use of an object in Java. If we are using static function or static variable then we can't call that function using object which violates OO feature.

9.Differentiate Static Method and Instance Method

- Static methods are defined with a keyword static
- They are accessed without the need of creating an object but with class name

- Instance methods in Java are attached to the objects of a class, rather than the class itself.
- Here, the method belongs to a class whose object must be created to call the function.

10.Predict the output

```
class  cexample{
public static void main(String args[])
{
   for (int i = 0; i < 10; i++) {
      // If the number is 2
      // skip and continue
      if (i == 2)
         continue;

      System.out.print(i + " ");
   }
}
      }
```

Output: 0 1 3 4 5 6 7 8 9

PART B

1.Explain in detail about the OOPS concepts

- Classes
- Objects
- Data Abstraction : hiding internal implementation details and exposing only the required functionalities
- Encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding
- Message Passing

2.Explain about method and constructor with appropriate examples.

Methods:
In Java, a method is a block of code that performs a specific task. Methods are used to organize code into reusable blocks, making the code more modular and easier to maintain. They are defined within a class and can be called to execute their functionality.

Example Code:

```
public class ExampleClass { public void printMessage() {
    System.out.println("Hello, this is a simple method!");
  }
  public static void main(String[] args) {
    // Create an instance of ExampleClass ExampleClass example = new ExampleClass();
    example.printMessage();
  }
}
```

Constructors:
A constructor in Java is a special type of method that is invoked when an object of a class is created. It has the same name as the class and doesn't have a return type. Constructors are used to initialize the state of an object.

Example Code:

```
public class Person { String name;
  int age;
   public Person(String initialName, int initialAge) { name = initialName;
    age = initialAge;
  }
   public void displayInfo() { System.out.println("Name: " + name); System.out.println("Age:
    " + age);
  }
  public static void main(String[] args) {
    // Create an instance of Person using the constructor Person person1 = new Person("John", 25);
    person1.displayInfo();
  }
 }
```

3.Write a Java program to create a class called Employee ,with methods set details where all the details of the Employee are given and another method set details where the details are displayed

```
import java.util.Scanner;

 class Employee {
 int emp_id; String name; String dept; float salary;
 void add_info (int id, String n, String d, float sal)
 {
 emp_id = id;
 name = n;
 dept = d;
 salary = sal;
 }
 void display() {
 System.out.println("Employee id: " + emp_id );
 System.out.println("Employee name: " + name );
 System.out.println("Employee department: " + dept );
 System.out.println("Employee salary: " + salary );
 }
 }
```

```
public class EmployeeClass {
    public static void main(String[] args) {
    Employee e1 = new Employee();
    Employee e2 = new Employee();
    e1.add_info (101, "Naman", "Salesforce", 45000);
    e2.add_info (102, "Riya", "Tax", 25000);
    e1.display();
    e2.display();
    }
    }
```

4. Write a JAVA program to perform binary search for an integer array of size 10.

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

  {
   Scanner scanner = new Scanner(System.in);

   // Creating a sorted array of integers
   int[] array = {1, 5, 10, 15, 20, 25, 30, 35, 40, 45};



   // Getting the search key from the user System.out.print("Enter the number to search:
   ");int key = scanner.nextInt();

   // Performing binary search


   int index = binarySearch(array, key);

   // Displaying the resultif (index != -1) {
      System.out.println("Element " + key + " found at index " + index);


   } else {
      System.out.println("Element " + key + " not found in the array");
   }

   scanner.close();
}

// Binary search method
private static int binarySearch(int[] array, int key) {int low = 0;
   int high = array.length - 1;

   while (low <= high) {
      int mid = (low + high) / 2;

      if (array[mid] == key) { return mid; // Key found
    } else if (array[mid] < key)
    {
                low = mid + 1; // Search in the right half
                } else {

                high = mid - 1; // Search in the left half
```

```
            }

            }

            return -1; // Key not found

          }

        }
```

5.Expalin in detail bout the operators in java

- Arithmetic Operators

Unary Operators

- Assignment Operator
- Relational Operators
- Logical Operators
- Ternary Operator
- Bitwise Operators

Shift Operators

6.Write a JAVA program to read one integer as input and to print the integer and its reversed interger

Input : 12345
Output : 12345 54321

```
import java.util.Scanner;
public class ReverseIntegerWithoutBuiltInFunction
{
 public static void main(String[] args)
{
      Scanner scanner = new Scanner(System.in); System.out.print("Enter an integer: ");
       int originalNumber = scanner.nextInt(); scanner.close();
       System.out.println("Original Integer: " + originalNumber);
        int reversedNumber = 0;
      int temp = originalNumber; while (temp != 0)




     {
        int digit = temp % 10;
        reversedNumber = reversedNumber * 10 + digit; temp /= 10;
     }
      System.out.println("Reversed Integer: " + reversedNumber);
  }
 }
```

UNIT II

PART A

1.List the rules for method overloading

The overloaded method must change the **argument list** (number of parameters,data type, or sequence of parameters).

The overloaded method can change the **return type**.
The overloaded method can change the **access modifier** (the signature of thefunction should be different).

2.Write a JAVA program to create a class called Bird which has a method named flyable () receiving its own instance as input.

```
public class Bird {
   public void flyable(Bird bird) {
      System.out.println("I am a bird, and I can fly!");
      System.out.println("My name is: " + bird.getClass().getSimpleName());
   }
   public static void main(String[] args) {
      Bird myBird = new Bird();
      myBird.flyable(myBird);
   }
}
```

3.How static nested classes are different from normal inner classes?

- In the case of normal or regular inner classes, without an outer class objectexisting, there cannot be an inner class object.
- i.e., an object of the inner class is always strongly associated with an outer classobject
- But in the case of static nested class, Without an outer class object existing, theremay be a static
- nested class object.
- i.e., an object of a static nested class is not strongly associated with the outerclass object.
- OuterClass.StaticNestedClass nestedObject = new OuterClass.StaticNestedClass();

4.Predict the Output of the following code snippet and justify the answer.

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


aa:
for(int i=1;i<=3;i++){
bb:
for(int j=1;j<=3;j++){
if(i==2&&j==2){
break aa;
```

```
} System.out.println(i+" "+j);
}}}}
```

1 1
1 2
1 3
2 1

5.Does java support multiple inheritance? Justify your answer.

No. Because Diamond Ambiguity problem arises when the child class overrides themethod present in parents class. And a class can extend from one class only.

6.Identify the need for inheritance and Differentiate super class and sub class.

- Need for inheritance is to enhance reusability.

Sub class :

- A subclass, also known as **child class** or **derived class**, is the class that inherits theproperties and behaviors of another class.
- So, if A and B are two classes and if B class inherits A class, then the B class iscalled the subclass.

Super class :

- A superclass, also known as **parent class** or **base class**, is the class whoseproperties and behaviors are inherited by the subclass.

So, if A and B are two classes and if B class inherits A class, then A class is calledthe superclass

7.How do you achieve run time polymorphism in Java?

- In Java, runtime polymorphism is achieved through method overriding. Runtime polymorphism
- allows you to use a class to represent different types of objects, andthe actual method that gets
- called is determined at runtime. This is often

accomplished using inheritance and overriding methods.

8.List the usage of 'this' keyword.

- this keyword can be used to refer current class instance variable.
- If there is ambiguity between the instance variables and parameters, this keywordresolves the
- problem of ambiguity.

9.Differentiate a class from interface

Class	**Interface**
The keywords are "class" and"extends"	The keywords are "interface" and"implements"
A class can be instantiated i.e., objectsof a class can be created.	An Interface cannot be instantiated i.e.objects cannot be created.
Classes do not support multipleinheritance.	The interface supportsmultiple inheritance.

It can be inherited from another class.	It cannot inherit a class.
It can contain constructors.	It cannot contain constructors.
It cannot contain abstract methods.	It contains abstract methods only.

10. Define Abstract class.

Java abstract class is a class that can not be initiated by itself, it needs to be subclassed by another class to use its properties. An abstract class is declared using the “abstract” keyword in its class definition.

PART B

1.Which OOP concept allows you to achieve Parent Child relationship between classes?Explain its types with appropriate example. (Any 2)

Inheritance :

- Inheritance is an object-oriented programming concept in which one classacquires/inherits the properties and behavior of another class.
- It represents a **parent-child relationship** between two classes.
- This parent-child relationship is also known as an **IS-A** relationship

Types of inheritance :

- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance
- Multiple Inheritance

2.Identify the use of super keyword in JAVA on variables, methods and constructors with appropriate examples.

The Java super keyword is used as a reference variable to access members from parent classes, allowing access to parent class constructors, members, and methods in the derived class.

It plays a crucial role in inheritance and polymorphism concepts.

- Use of super with variables
- Use of super with methods
- Use of super with constructors

Use of super with variables

```
class Vehicle {
int maxSpeed = 120;}
class Car extends Vehicle {
int maxSpeed = 180;
void display()
{System.out.println("Maximum Speed: "+ super.maxSpeed);
```

```
        System.out.println("Maximum Speed: "+ maxSpeed);}}
        }
        class Test {
        public static void main(String[] args)
        { Car small = new Car();
        small.display();


        }
 }
```

Use of super with methods

```
class Person {
void message()
{
System.out.println("This is person class\n");
}}
class Student extends Person {void message()
{
System.out.println("This is student class");
}
void display()
{ message(); super.message();
}}
class Test {
public static void main(String args[])
{ Student s = new Student();
// calling display() of Students.display();
}}
```

Use of super with constructors

```
class Person {Person()
{
System.out.println("Person class Constructor");
          }
}
class Student extends Person {Student()
{
super();
System.out.println("Student class Constructor");
}}
```

```
        class Test {
        public static void main(String[] args)
        {
        Student s = new Student();
        }}
```

3.Write a Java program to create an interface ‘Drawable’ with a method draw() that takesno arguments and returns void. Create three classes ‘Circle’, ‘Rectangle’, and ‘Triangle’that implement the ‘Drawable’ interface and override the draw() method to draw their respective shapes.

```
// Drawable interfaceinterface Drawable {
   void draw();
 }
```

```
// Circle class implementing the Drawable interface class Circle implements
Drawable {
   @Override
   public void draw() { System.out.println("Drawing a Circle");
   }
}

// Rectangle class implementing the Drawable interfaceclass Rectangle implements
Drawable {
   @Override


   public void draw() { System.out.println("Drawing a Rectangle");
   }
}

// Triangle class implementing the Drawable interfaceclass Triangle implements
Drawable {
   @Override
   public void draw() { System.out.println("Drawing a Triangle");
   }
}

// Main class to test the implementationspublic class Main {
   public static void main(String[] args) {
      // Creating objects of Circle, Rectangle, and TriangleCircle circle = new Circle();
      Rectangle rectangle = new Rectangle(); Triangle triangle = new
      Triangle();

      // Calling draw() method on each object circle.draw();          // Drawing
      a Circle rectangle.draw();  // Drawing a Rectangle
}}
```

4.Imagine you are designing a system for a library where books, CDs, and DVDs need to be managed. Each of these items has unique properties, and the library wants a unified system to handle them. How would you utilize Java interfaces to create a cohesive and extensible solution for managing different types of items in the library's inventory?

```
interface LibraryItem {
   String getTitle();
   void setTitle(String title);
   void displayInfo();
}

class Book implements LibraryItem {
   private String title;

   @Override
   public String getTitle() {
      return title;
   }
   @Override
   public void setTitle(String title) {
```

```
  this.title = title;
    }

    @Override
    public void displayInfo() {
        System.out.println("Book: " + title);


 }
}

class CD implements LibraryItem {
    private String title;

    @Override
    public String getTitle() {
        return title;
    }

    @Override
    public void setTitle(String title) {
        this.title = title;
    }

    @Override
    public void displayInfo() {
        System.out.println("CD: " + title);
    }
}

class Library {
    public static void main(String[] args) {
        // Create instances of different items
        Book book = new Book();
        book.setTitle("The Java Programming Language");

        CD cd = new CD();
        cd.setTitle("Java Soundtrack");

        DVD dvd = new DVD();
        dvd.setTitle("Java: The Movie");


        book.displayInfo();
        cd.displayInfo();
        dvd.displayInfo();
    }
}
```

5.Design a calculator with two different operations ,addition and multiplication organized into separate packages. Import the calculator.operation.add and calculator.operation.multiply package in the driver class

```
package com.calculator.operations.add;

public class Addition {
    public static int add(int a, int b) {
```

```
    return a + b;
    }
}
```

```
 package com.calculator;

import com.calculator.operations.add.Addition;
import com.calculator.operations.subtract.Subtraction;
public class Calculator {
    public static void main(String[] args) {
        // Sample usage of calculator operations



        int num1 = 10;
        int num2 = 5;

        // Addition
        int sum = Addition.add(num1, num2);
        System.out.println("Sum: " + sum);

        // Subtraction
        int difference = Subtraction.subtract(num1, num2);
        System.out.println("Difference: " + difference);


    }
}
```

6.Explain in detail about Inner classes in Java

Nested Inner Class: A class defined within another class. It can access the members, including private members, of the outer class.

Local Inner Class: A class defined within a method. It can access the members of the enclosing class and the final variables of the method.

Anonymous Inner Class: A class defined without a name. It is used for instantiating objects and implementing interfaces.

Static Nested Class: A static class defined within another class. It can access only static members of the outer class.

UNIT III

PART A

1. Define an exception in Java

An exception in Java is an event that disrupts the normal flow of the program during its execution. It occurs when an error or an unexpected condition is encountered.

2.What is the purpose of the try-catch block in Java?

The try-catch block in Java is used to handle exceptions. The code that might throw an exception is placed inside the try block, and if an exception occurs, it is caught and handled in the catch block.

3. Write the java program to print the exception information using getMessage() method

```
import java.io.*;
class test {
public static void main (String[] args) {
int a=5;
int b=0;
        try{
        System.out.println(a/b); }                }
catch(ArithmeticException e){
        System.out.println(e.getMessage()); }}}
```

4. Identify the need for 'finally' block in exception handling.

- finally block is associated with a try, catch block.
- It is executed every time irrespective of exception is thrown or not.
- finally block is used to execute important statements such as closing statement,release the resources, and release memory.

5. Differentiate throw and throws in Java.

throw is used to explicitly throw an exception within a method, while throws is used in the method signature to declare the exceptions that might be thrown by that method.

6.Can multiple catch blocks be used for a single try block?

Yes, multiple catch blocks can be used for a single try block. Each catch block handles a specific type of exception, allowing the program to take different actions based on the type of exception.

7.Differentiate between checked and unchecked exceptions.

Checked exceptions are checked at compile-time, and the programmer is required to handle them using try-catch or declare them in the method signature using throws. Unchecked exceptions are runtime exceptions that need not be explicitly handled.

8.Predict the output.

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

```
{ try
        { System.out.printf("1");
                int sum = 9 / 0;
                System.out.printf("2"); }
        catch(ArithmeticException e)
        { System.out.printf("3");       }
        catch(Exception e)
        { System.out.printf("4"); }             }
        finally
        {       System.out.printf("5");
        } }}
```

Ans:135.

9.How can the finally block be used without a catch block?

The finally block can be used without a catch block when it is necessary to ensure that certain code is executed regardless of whether an exception occurs or not. It is often used for cleanup operations.

10.Write a JAVA program to throw divide by zero exception.

```
class ExceptionExample {
public static void main(String args[]) {
  try {
    // Code that can raise exception
    int div = 509 / 0;
  }
catch (ArithmeticException e)
{
    System.out.println(e);
        System.out.println(“Pls give a non zero divisor”);
  }
  System.out.println("End of code");
}
}
```

PART B

1.Explain about the difference between throw and throws, final and finally with appropriate code whenever required.

```
public class Main {
static void checkAge(int age) throws
ArithmeticException {if (age < 18) {

throw new ArithmeticException("Access denied - You must be
at least 18 yearsold.");
} else {
System.out.println("Access granted - You are old enough!");
}
}
public static void
main(String[]
args) {
checkAge(15);
}
}
```

2.Illustrate the working of nested try statement with an appropriate program.

```
class NestingDemo {
  public static void main(String args[]) {
   try {//main try-block1
     try {//try-block2
       try {//try-block3
          int arr[] = { 1, 2, 3, 4 };
            System.out.println(arr[10]);
       }
                    catch (ArithmeticException e) {
                       System.out.print("Arithmetic Exception");
                       System.out.println(" handled in try-block3");
                     }
                    }
catch (ArithmeticException e) {
       System.out.print("Arithmetic Exception");
       System.out.println(" handled in try-block2");
```

3.Write a JAVA program to explicitly throw ArrayIndexOutOfBounds Exception and to handle it with a message “Out of box thinking is needed! But out of box access is denied!”

```
public class ArrayExceptionExample {
    public static void main(String[] args) {
        try {
            // Simulating an array with a length of 5int[]
            myArray = new int[5];
```

```
            // Trying to access an index beyond the array's bounds
           Throw new ArrayIndexOutOfBoundsException(("Out of box thinking is needed!But
 out    of    box    access    is    denied!");
         } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println(e.getMessage());
         }
      }
}
```

4.Illustrate the Multiple Catch Blocks with an example

```
public class MultipleCatchBlock1 {
  public static void main(String[] args) {
    try {
      int a[] = new int[5];
      a[5] = 30 / 0;
    } catch (ArithmeticException e) {
      System.out.println("Arithmetic Exception occurs");
    } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("ArrayIndexOutOfBounds Exception occurs");
    } catch (Exception e) {
      System.out.println("Parent Exception occurs");
    }
    System.out.println("End of the code");
  }
}
```

5. Write a java program to throw the user defined exception, If the students' CGPA in first year is <75, then generate an exception stating a message like "Pls pass first year, First !" If the CGPA is >=75, allow the student for admission

```
class CGPAException extends Exception {

    public CGPAException(String message) {

        super(message);

    }}

// Student class represents a student with CGPA

class Student {

    private double cgpa;

    public Student(double cgpa) {
```

```
    this.cgpa = cgpa;
}
public void checkAdmissionEligibility() throws CGPAException {
    if (cgpa < 75) {
        throw new CGPAException("Please pass first year, First!");
    } else {
        System.out.println("Congratulations! Admission allowed.");
    }}}
public class AdmissionApplication {
public static void main(String[] args) {
    // Create a sample student with CGPA
    Student student = new Student(70);
    try {
        // Check admission eligibility
        student.checkAdmissionEligibility();
    } catch (CGPAException e) {
        System.out.println("Exception: " + e.getMessage());
    }}}
```

6.Explain the different blocks in Exceptional Handling with an example

Java uses **try-catch** blocks and other keywords like finally, throw, and throws to handle exceptions.
JVM(Java Virtual Machine) by default handles exceptions, when an exception is raised it will halt the execution of the program and throw the occurred exception.

UNIT IV

PART A

1.State thread in Java?

A thread in Java is the smallest unit of execution that represents an independent path of execution within a program.

2.Why is the start method used for in Java threads?

The start method is used to begin the execution of a thread. It internally calls the run method of the thread.

3.What is a daemon thread in Java?

A daemon thread is a thread that runs in the background and does not prevent the JVM from exiting when all non-daemon threads have completed.

4.Define different methods in creating a thread.

Threads can be created by using two mechanisms :

- Extending the Thread class
- Implementing the Runnable Interface

5. Write a JAVA code to print the priority of main thread.

```
class aa {
        public static void main(String[]
        args) {System.out.println("Hello,
        World!");
        System.out.println( "Main thread priority : "+Thread.currentThread().getPriority());
        }}
```

6. Predict the output:

```
public class Test extends Thread implements Runnable
{
        public void run()
        {
                System.out.printf("Hi");
        }
        public static void main(String[] args) throws InterruptedException
        {
```

```
            Test obj = new Test();
            obj.run();
            obj.start();
        }
}
```

Ans:Hi
Hi

7. What happens when threads have same priority?

   If two threads have the same priority then we can't expect which thread will execute first.
   It depends on the thread scheduler's algorithm(Round-Robin, First Come First Serve, etc)

8. Differentiate between run and start in thread.

start()	run()
Creates a new thread and the run() method is executed on the newly created thread.	No new thread is created and the run() method is executed on the calling thread itself.
Can't be invoked more than one time otherwise throws java.lang.IllegalStateException	Multiple invocation is possible
Defined in java.lang.Thread class.	Defined in java.lang.Runnable interface and must be overridden in the implementing class.

9.Why java is not purely object oriented programming?

Pure Object Oriented Language or Complete Object Oriented Language are Fully Object Oriented Language which supports or have features which treats everything inside program as objects. It doesn't support primitive datatype(like int, char, float, bool, etc.)Java supports primitive data types.

10.State the need for Wrapper classes in java.

- Wrapper class in Java is a class whose object wraps or contains primitive data types.
- When we create an object to a wrapper class, it contains a field and in this field, we can store primitive data types.
- In other words, we can wrap a primitive value into a wrapper class object.

- Wrapper classes provide a way to use primitive data types (int, boolean, etc..) as objects.

PART B

1.Explain about the need for Wrapper classes and give a JAVA code to perform Autoboxing and unboxing on an integer value.

A Wrapper class in Java is a class whose object wraps or contains primitive datatypes. When we create an object to a wrapper class, it contains a field and in this field,we can store primitive data types.
In other words, we can wrap a primitive value into a wrapper class object

```
public class WrapperClassExample {
public static void main(String[] args)
{
Integer intWrapper = Integer.valueOf(42);  // Integer wrapper for int
Double doubleWrapper = Double.valueOf(3.14); // Double wrapper for double
Boolean booleanWrapper = Boolean.valueOf(true); // Boolean wrapper for booleanint
intPrimitive = intWrapper.intValue();
double doublePrimitive = doubleWrapper.doubleValue(); boolean
booleanPrimitive = booleanWrapper.booleanValue();
System.out.println("Wrapper to Primitive:");
System.out.println("Integer: " + intPrimitive);
System.out.println("Double: " + doublePrimitive);
System.out.println("Boolean: " + booleanPrimitive);

// Autoboxing and Unboxing (Java automatically converts between primitives and wrappers)
Integer ageWrapper = 25; // Autoboxing (int to Integer)
int agePrimitive = ageWrapper; // Unboxing (Integer to int)
System.out.println("\nAutoboxing and Unboxing:");
System.out.println("Age (Wrapper): " + ageWrapper);
System.out.println("Age (Primitive): " + agePrimitive);
}}
```

2. Write a JAVA program to implement multithreading by extending the thread class.

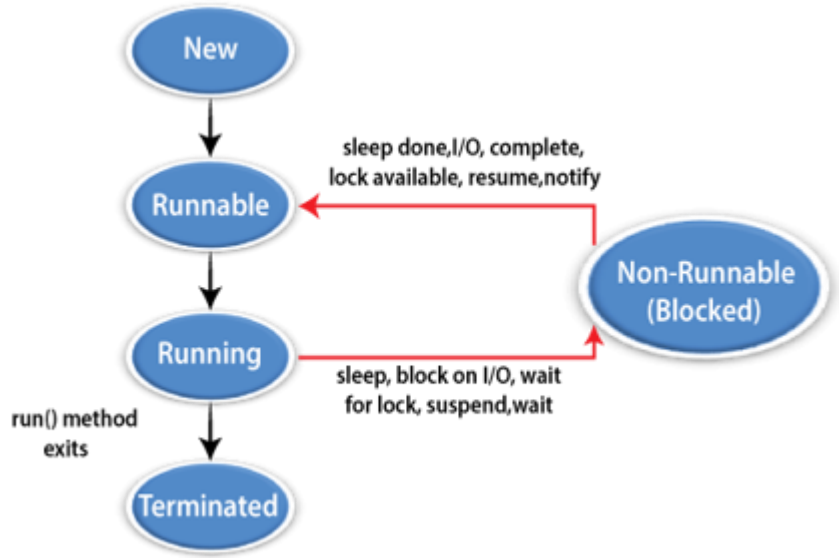
```
class MultithreadingDemo extends Thread {
public void run()
 {
try {
 System.out.println( "Thread " + Thread.currentThread().getId()
+ " is running");
      }
catch (Exception e) {
System.out.println("Exception is caught");
}}}
```

```
 public class Multithread {
 public static void main(String[] args){int n
  = 8; // Number of threads
for (int i = 0; i < n; i++) {
MultithreadingDemo object= new MultithreadingDemo(); object.start();

}}}
```

3.Write a java program to to set and get priority of threads using thread methods with an example.

```
import java.lang.*;
class ThreadDemo extends Thread {
   public void run()
   {
      System.out.println("Inside run method");
   }
   public static void main(String[] args)
   {
      ThreadDemo t1 = new ThreadDemo();
      ThreadDemo t2 = new ThreadDemo();
      ThreadDemo t3 = new ThreadDemo();
      System.out.println("t1 thread priority : "
                  + t1.getPriority());
System.out.println("t2 thread priority : "
                  + t2.getPriority());
      System.out.println("t3 thread priority : "
                  + t3.getPriority());
 t1.setPriority(2);
      t2.setPriority(5);
      t3.setPriority(8);
      System.out.println("t1 thread priority : "
                  + t1.getPriority());
      System.out.println("t2 thread priority : "
                  + t2.getPriority());
      System.out.println("t3 thread priority : "
                  + t3.getPriority());
System.out.println("Currently Executing Thread : "
                     + Thread.currentThread().getName());
System.out.println("Main thread priority : "
                    + Thread.currentThread().getPriority());

Thread.currentThread().setPriority(10);
      System.out.println(
         "Main thread priority : "
         + Thread.currentThread().getPriority());
   }
}
```

4.Illustrate the Lifecycle of thread in detail.



5.Write a java program to implement the Runnable Interface in multithreading.

```
public class RunnableDemo {
        public static void main(String[] args)
        {
          System.out.println("Main thread is- "
                                          + Thread.currentThread().getName());
          Thread t1 = new Thread(new RunnableDemo().new RunnableImpl());
          t1.start();
        }
        private class RunnableImpl implements Runnable {
          public void run()
          {                          System.out.println(Thread.currentThread().getName()
                                                  + ", executing run() method!");
          }}}
```

6.Explain the need for Wrapper classes in java

There are certain needs for using the Wrapper class in Java as mentioned below:

1. They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).
2. The classes in java.util package handles only objects and hence wrapper classes help in this case also.
3. Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.
4. An object is needed to support synchronization in multithreading.

UNIT V

PART A

1.What is called Stream and different categories of Stream

A stream is a sequence of data. In Java, a stream is composed of bytes.
3 categories of classes in java.io package:

- Input Streams
- Output Streams
- Error Streams

2.State the different operations of file

- Create a File
- Read from a File
- Write to a File
- Delete a File

3.Why file handling is needed in Java

In Java, with the help of File Class, we can work with files.
File Class is inside the java.io package.
The File class can be used by creating an object of the class and then specifying the name of the file.
File Handling is an integral part of any programming language as file handling enables us to store the output of any particular program in a file and allows us to perform certain operations on it.

4.Write the syntax for creating a file in java

```
import java.io.File;  // Import the File class

File myObj = new File("filename.txt"); // Specify the filename
```

5.Define String Constant Pool.

The string constant pool in Java is a storage area in the heap memory that stores string literals.

When a string is created, the JVM checks if the same value exists in the string pool.
If it does, the reference to that existing object is returned.
Otherwise, a new string object is created and added to the string pool, and its reference is returned.

6.State the difference in creating a string using String literals and new keyword.

Using String literal

- String s1="Welcome"; //It doesn't create a new instance

Using new keyword

- This is the common way to create a String object in java.
- String str1= new String("Hello!");

7.Write a java program to returns a char value at the given index number

```
class Main {
  public static void main(String args[]){
  String name="codemithra";
   char ch=name.charAt(4);
   System.out.println(ch);
  }
}
```

8.Differentiate between String and String Buffer

StringBuffer objects are like String class objects, but they can be modified to perform manipulating operations on the StringBuffer

String, every time a new memory is allocated for the updated String, no updation takes place in the original String.

9.Why java string are immutable?

A String in Java that is specified as immutable, as the content shared storage in a single pool to minimize creating a copy of the same value. String class and all wrapper classes in Java that include Boolean, Character, Byte, Short, Integer, Long, Float, and Double are immutable. A user is free to create immutable classes of their own.

10.How do you check if a string contains a specific substring in Java?

You can check if a string contains a specific substring in Java using the contains method or by using the indexOf method.

PART B

1.Write a JAVA program to read the contents of a file named “a.txt” and to print it on console.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class reading {
public static void main(String[] args)
{
try {
File Obj = new File("a.txt");
Scanner Reader = new Scanner(Obj);
while (Reader.hasNextLine()) {
String data = Reader.nextLine();
System.out.println(data);
}
Reader.close();
}
catch (FileNotFoundException e) {
System.out.println("An error has occurred.");
}}}
```

2.Write a Java program to check the given string is Palindrome or not.

```
import java.util.Scanner;

class ChkPalindrome

{

  public static void main(String args[])

  {

    String str, rev = "";

    Scanner sc = new Scanner(System.in);


    System.out.println("Enter a string:");

    str = sc.nextLine();

     int length = str.length();

    for ( int i = length - 1; i >= 0; i-- )

      rev = rev + str.charAt(i);

    if (str.equals(rev))

      System.out.println(str+" is a palindrome");

    else
```

```
        System.out.println(str+" is not a palindrome");
    }}
```

3.How exceptions are handled in file in Java? Discuss the types of exceptions that may occur

```
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;

public class FileExceptionHandling {
   public static void main(String[] args) {
      try {
         // Code that may throw exceptions
         FileReader reader = new FileReader("example.txt");

         // Additional file handling code

         // Close the file
         reader.close();
      } catch (FileNotFoundException e) {
         // Handle file not found exception
```

4.Write a java program to compare two strings.

```
public class test {
    public static void main(String args[])
    {
        String string1 = new String("Geeksforgeeks");
        String string2 = new String("Practice");
        String string3 = new String("Geeks");
        String string4 = new String("Geeks");
        String string5 = new String("geeks");
        // Comparing for String 1 != String 2
        System.out.println("Comparing " + string1 + " and " + string2
                                        + " : " + string1.equals(string2));

        // Comparing for String 3 = String 4
        System.out.println("Comparing " + string3 + " and " + string4
                                        + " : " + string3.equals(string4));
```

```
        // Comparing for String 4 != String 5
        System.out.println("Comparing " + string4 + " and " + string5
                                        + " : " + string4.equals(string5));

        // Comparing for String 1 != String 4
        System.out.println("Comparing " + string1 + " and " + string4
                                        + " : " + string1.equals(string4));
    } }
```

5.List the different methods available in String?Illustrate any 2 methods with an example.

char charAt(int index)

int length()

String substring(int beginIndex)

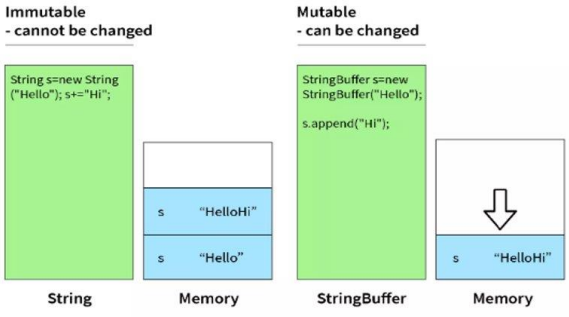
boolean equals(Object another)

String concat(String str)

String replace(char old, char new)

int indexOf(int ch)

6.Explain in detail about String buffers in Java.



- StringBuffer class is used for storing the mutable sequence of different datatypes which means we can update the sequence of the StringBuffer class very easily and efficiently without creating a new sequence in memory.
- StringBuffer is faster than the String class and provides various additional methods for deleting the sequence elements, updating the sequence elements, etc.
- The memory allocation of the StringBuffer object is done in the heap section of memory.
- StringBuffer <object_name> = new StringBuffer();