/ Constructor with color and filledQ1 A Write a program in java to create a stack class of variable size with push and pop methods. Create two objects of stack with 10 data items in both. Compare the stack elements of both stacks and print the comparison results.

```
import java.util.Arrays;

// Stack class
class Stack {
    private int[] stackArray;
    private int top;

    // Constructor to initialize the stack with a given size
    public Stack(int size) {
        stackArray = new int[size];
        top = -1;
    }

    // Method to push an element onto the stack
    public void push(int value) {
        if (top == stackArray.length - 1) {
            System.out.println("Stack Overflow: Cannot push " + value);
        } else {
            stackArray[++top] = value;
        }
    }

    // Method to pop an element from the stack
    public int pop() {
        if (top == -1) {
            System.out.println("Stack Underflow: Cannot pop from an empty stack");
            return -1; // Indicates stack is empty
        } else {
            return stackArray[top--];
        }
    }

    // Method to get all elements of the stack
    public int[] getElements() {
        return Arrays.copyOfRange(stackArray, 0, top + 1);
    }

    // Method to check if the stack is empty
    public boolean isEmpty() {
        return top == -1;
```

```
    }
}

public class StackComparison {
    public static void main(String[] args) {
        // Create two stacks with size 10
        Stack stack1 = new Stack(10);
        Stack stack2 = new Stack(10);

        // Push 10 elements into stack1
        for (int i = 0; i < 10; i++) {
            stack1.push(i + 1); // Push 1 to 10
        }

        // Push 10 elements into stack2 (modify as needed to create differences)
        for (int i = 0; i < 10; i++) {
            stack2.push(i + 1); // Push 1 to 10 (same as stack1)
        }

        // Compare the elements of both stacks
        compareStacks(stack1, stack2);
    }

    // Method to compare two stacks
    public static void compareStacks(Stack stack1, Stack stack2) {
        int[] elements1 = stack1.getElements();
        int[] elements2 = stack2.getElements();

        if (Arrays.equals(elements1, elements2)) {
            System.out.println("Both stacks are identical.");
        } else {
            System.out.println("Stacks are not identical.");
            System.out.println("Stack 1 elements: " + Arrays.toString(elements1));
            System.out.println("Stack 2 elements: " + Arrays.toString(elements2));
        }
    }
}
```

Q1 B Design an interface named Colorable with a void method named howToColor(). Every class of a colorable object must implement the Colorable interface. Design a class named Square that extends GeometricObject and implements Colorable. Implement howToColor to display the message Color all four sides. Write a program that creates an array of five GeometricObjects. For each object in the array, display its area and invoke its howToColor method if it is colorable

```
// Interface Colorable
interface Colorable {
    void howToColor();
}

// Abstract class GeometricObject
abstract class GeometricObject {
    // Abstract method to calculate area
    public abstract double getArea();
}

// Square class extends GeometricObject and implements Colorable
class Square extends GeometricObject implements Colorable {
    private double side;

    // Constructor
    public Square(double side) {
        this.side = side;
    }

    // Implement getArea method
    @Override
    public double getArea() {
        return side * side;
    }

    // Implement howToColor method
    @Override
    public void howToColor() {
        System.out.println("Color all four sides.");
    }
}

// Circle class extends GeometricObject (not colorable)
class Circle extends GeometricObject {
    private double radius;
```

```
    // Constructor
    public Circle(double radius) {
        this.radius = radius;
    }

    // Implement getArea method
    @Override
    public double getArea() {
        return Math.PI * radius * radius;
    }
}

public class ColorableTest {
    public static void main(String[] args) {
        // Create an array of GeometricObjects
        GeometricObject[] objects = new GeometricObject[5];

        // Initialize objects (mix of Squares and Circles)
        objects[0] = new Square(5);
        objects[1] = new Circle(3);
        objects[2] = new Square(7);
        objects[3] = new Circle(4);
        objects[4] = new Square(6);

        // Iterate over the objects and display their area
        for (GeometricObject obj : objects) {
            System.out.println("Area: " + obj.getArea());
            if (obj instanceof Colorable) {
                ((Colorable) obj).howToColor();
            }
        }
    }
}
```

Q. 2 A) Design a class named Triangle that extends GeometricObject. The class contains: ■ Three double data fields named side1, side2, and side3 with default values 1.0 to denote three sides of the triangle. ■ A no-arg constructor that creates a default triangle. ■ A constructor that creates a triangle with the specified side1, side2, and side3. ■ The accessor methods for all three data fields. ■ A method named getArea() that returns the area of this triangle. ■ A method named getPerimeter() that returns the perimeter of this triangle. ■ A method named toString() that returns a string description for the triangle. The toString() method is implemented as follows: return "Triangle: side1 = " + side1 + " side2 = " + side2 + " side3 = " + side3; Implement the classes. Write a program that prompts the user to enter three sides of the triangle, a color, and a Boolean value to indicate whether the triangle is filled. The program should create a Triangle object with these sides and set the color and filled properties using the input. The program should display the area, perimeter, color, and true PCC CS593.3 6 20+ 20 or false to indicate whether it is filled or not.

```
// Abstract class GeometricObject
abstract class GeometricObject {
   private String color;
   private boolean filled;

   // Default constructor
   public GeometricObject() {
      color = "white";
      filled = false;
   }

   // Constructor with color and filled properties
   public GeometricObject(String color, boolean filled) {
      this.color = color;
      this.filled = filled;
   }

   // Accessor and mutator for color
   public String getColor() {
      return color;
   }

   public void setColor(String color) {
      this.color = color;
   }

   // Accessor and mutator for filled
   public boolean isFilled() {
      return filled;
```

```
    }

    public void setFilled(boolean filled) {
        this.filled = filled;
    }

    // Abstract method for area
    public abstract double getArea();

    // Abstract method for perimeter
    public abstract double getPerimeter();
}

// Triangle class
class Triangle extends GeometricObject {
    private double side1;
    private double side2;
    private double side3;

    // No-arg constructor
    public Triangle() {
        this(1.0, 1.0, 1.0);
    }

    // Constructor with specified sides
    public Triangle(double side1, double side2, double side3) {
        if (!isValidTriangle(side1, side2, side3)) {
            throw new IllegalArgumentException("Invalid triangle sides");
        }
        this.side1 = side1;
        this.side2 = side2;
        this.side3 = side3;
    }

    // Accessor methods
    public double getSide1() {
        return side1;
    }

    public double getSide2() {
        return side2;
    }

    public double getSide3() {
```

```
        return side3;
    }

    // Method to calculate the area using Heron's formula
    @Override
    public double getArea() {
        double s = getPerimeter() / 2;
        return Math.sqrt(s * (s - side1) * (s - side2) * (s - side3));
    }

    // Method to calculate the perimeter
    @Override
    public double getPerimeter() {
        return side1 + side2 + side3;
    }

    // toString method
    @Override
    public String toString() {
        return "Triangle: side1 = " + side1 + " side2 = " + side2 + " side3 = " + side3;
    }

    // Helper method to validate triangle sides
    private boolean isValidTriangle(double side1, double side2, double side3) {
        return side1 + side2 > side3 && side1 + side3 > side2 && side2 + side3 > side1;
    }
}

// Main program
import java.util.Scanner;

public class TriangleTest {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Prompt user for triangle sides
        System.out.print("Enter side1: ");
        double side1 = scanner.nextDouble();

        System.out.print("Enter side2: ");
        double side2 = scanner.nextDouble();

        System.out.print("Enter side3: ");
        double side3 = scanner.nextDouble();
```

```
        // Prompt user for color and filled property
        System.out.print("Enter color: ");
        String color = scanner.next();

        System.out.print("Is the triangle filled (true/false): ");
        boolean filled = scanner.nextBoolean();

        try {
            // Create Triangle object
            Triangle triangle = new Triangle(side1, side2, side3);
            triangle.setColor(color);
            triangle.setFilled(filled);

            // Display triangle details
            System.out.println(triangle);
            System.out.println("Area: " + triangle.getArea());
            System.out.println("Perimeter: " + triangle.getPerimeter());
            System.out.println("Color: " + triangle.getColor());
            System.out.println("Filled: " + triangle.isFilled());
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }

        scanner.close();
    }
}
```

Q 2 B) Define a class named ComparableCircle that extends Circle and implements Comparable. Implement the compareTo method to compare the circles on the basis of area. Write a class to find the larger of two instances of ComparableCircle objects.

```
// Circle class
class Circle {
    private double radius;

    // Default constructor
    public Circle() {
        this(1.0);
    }

    // Constructor with specified radius
    public Circle(double radius) {
        if (radius < 0) {
            throw new IllegalArgumentException("Radius cannot be negative");
        }
        this.radius = radius;
    }

    // Getter for radius
    public double getRadius() {
        return radius;
    }

    // Setter for radius
    public void setRadius(double radius) {
        if (radius < 0) {
            throw new IllegalArgumentException("Radius cannot be negative");
        }
        this.radius = radius;
    }

    // Method to calculate area
    public double getArea() {
        return Math.PI * radius * radius;
    }

    // Method to calculate perimeter
    public double getPerimeter() {
        return 2 * Math.PI * radius;
    }
```

```
    @Override
    public String toString() {
        return "Circle with radius = " + radius;
    }
}

// ComparableCircle class
class ComparableCircle extends Circle implements Comparable<ComparableCircle> {
    // Constructor with specified radius
    public ComparableCircle(double radius) {
        super(radius);
    }

    // compareTo method to compare circles based on area
    @Override
    public int compareTo(ComparableCircle other) {
        return Double.compare(this.getArea(), other.getArea());
    }

    @Override
    public String toString() {
        return super.toString() + ", Area = " + getArea();
    }
}

// Main class to test ComparableCircle
public class ComparableCircleTest {
    public static void main(String[] args) {
        // Create two ComparableCircle objects
        ComparableCircle circle1 = new ComparableCircle(5.0);
        ComparableCircle circle2 = new ComparableCircle(7.0);

        // Display the circles
        System.out.println("Circle 1: " + circle1);
        System.out.println("Circle 2: " + circle2);

        // Compare the two circles
        int comparison = circle1.compareTo(circle2);
        if (comparison > 0) {
            System.out.println("Circle 1 is larger.");
        } else if (comparison < 0) {
            System.out.println("Circle 2 is larger.");
        } else {
            System.out.println("Both circles are of equal area.");
```

```
        }
    }
}
```

Q3 A) Define two classes Car and Model class. Car has a private variable carname and class Model has a private variable Modelname. Write zero argument and parameterized constructors to initialize the data member and member functions to get, set and display the data member for these classes. In the result class, write a merge() method which takes the 2 arguments as class Car and class Modelobjects and concatenates the data members of these two objects. Write a main function that creates instances of the classes Car and Model, assigns values to each and concatenates the value of data member of these objects using the merge() method. Display the objects before and after concatenation.

```
// Car class
class Car {
    private String carName;

    // Zero-argument constructor
    public Car() {
        this.carName = "Unknown";
    }

    // Parameterized constructor
    public Car(String carName) {
        this.carName = carName;
    }

    // Getter for carName
    public String getCarName() {
        return carName;
    }

    // Setter for carName
    public void setCarName(String carName) {
        this.carName = carName;
    }

    // Method to display carName
    public void display() {
        System.out.println("Car Name: " + carName);
    }
}
```

```
// Model class
class Model {
    private String modelName;

    // Zero-argument constructor
    public Model() {
        this.modelName = "Unknown";
    }

    // Parameterized constructor
    public Model(String modelName) {
        this.modelName = modelName;
    }

    // Getter for modelName
    public String getModelName() {
        return modelName;
    }

    // Setter for modelName
    public void setModelName(String modelName) {
        this.modelName = modelName;
    }

    // Method to display modelName
    public void display() {
        System.out.println("Model Name: " + modelName);
    }
}

// Result class
class Result {
    // Merge method to concatenate carName and modelName
    public static String merge(Car car, Model model) {
        return car.getCarName() + " " + model.getModelName();
    }
}

// Main class
public class CarModelTest {
    public static void main(String[] args) {
        // Create Car object
        Car car = new Car("Toyota");
```

```
        // Create Model object
        Model model = new Model("Corolla");

        // Display objects before concatenation
        System.out.println("Before Concatenation:");
        car.display();
        model.display();

        // Merge car and model names
        String mergedName = Result.merge(car, model);

        // Display concatenated result
        System.out.println("\nAfter Concatenation:");
        System.out.println("Merged Name: " + mergedName);
    }
}
```

Q3 B) Write a class named Octagon that extends GeometricObject and implements the Comparable and Cloneable interfaces. Assume that all eight sides of the octagon are of equal length. The area can be computed using the following formula: area = (2 + 4/√2)× side × side. Write a program that creates an Octagon object with side value 5 and displays its area and perimeter. Create a new object using the clone method and compare the two objects using the compareTo method.

```
// Octagon class
class Octagon extends GeometricObject implements Comparable<Octagon>, Cloneable {
    private double side;

    // Constructor
    public Octagon(double side) {
        this.side = side;
    }

    // Getter for side
    public double getSide() {
        return side;
    }

    // Setter for side
    public void setSide(double side) {
        this.side = side;
    }
```

```
    // Method to calculate area
    public double getArea() {
        return (2 + 4 / Math.sqrt(2)) * side * side;
    }

    // Method to calculate perimeter
    public double getPerimeter() {
        return 8 * side;
    }

    // Override compareTo method
    @Override
    public int compareTo(Octagon o) {
        return Double.compare(this.getArea(), o.getArea());
    }

    // Override clone method
    @Override
    public Octagon clone() throws CloneNotSupportedException {
        return (Octagon) super.clone();
    }

    // toString method
    @Override
    public String toString() {
        return "Octagon with side: " + side + ", Area: " + getArea() + ", Perimeter: " +
getPerimeter();
    }
}

// Main class
public class OctagonTest {
    public static void main(String[] args) {
        // Create an Octagon object with side 5
        Octagon octagon1 = new Octagon(5);

        // Display area and perimeter
        System.out.println("Original Octagon:");
        System.out.println(octagon1);

        try {
            // Clone the octagon
            Octagon octagon2 = octagon1.clone();
```

```
            // Display cloned octagon
            System.out.println("\nCloned Octagon:");
            System.out.println(octagon2);

            // Compare the two octagons
            int comparison = octagon1.compareTo(octagon2);
            System.out.print("\nComparison Result: ");
            if (comparison == 0) {
                System.out.println("The two octagons are equal in area.");
            } else if (comparison > 0) {
                System.out.println("The original octagon is larger than the cloned octagon.");
            } else {
                System.out.println("The cloned octagon is larger than the original octagon.");
            }
        } catch (CloneNotSupportedException e) {
            System.out.println("Cloning not supported.");
        }
    }
}
```

Q4 A) Write a program that prompts the user to enter a positive integer and displays all its smallest factors in decreasing order. For example, if the integer is 120, the smallest factors are displayed as 5, 3, 2, 2, 2. Use the StackOfIntegers class to store the factors (e.g., 2, 2, 2, 3, 5) and retrieve and display them in reverse order.

```
import java.util.Stack;

// StackOfIntegers class
class StackOfIntegers {
    private Stack<Integer> stack;

    public StackOfIntegers() {
        stack = new Stack<>();
    }

    public void push(int value) {
        stack.push(value);
    }

    public int pop() {
        return stack.pop();
    }
```

```
    public boolean isEmpty() {
        return stack.isEmpty();
    }
}

// Main class
public class Factorization {
    public static void main(String[] args) {
        java.util.Scanner scanner = new java.util.Scanner(System.in);

        // Prompt user for input
        System.out.print("Enter a positive integer: ");
        int number = scanner.nextInt();

        // Validate input
        if (number <= 0) {
            System.out.println("Please enter a positive integer.");
            return;
        }

        // Find smallest factors
        StackOfIntegers factors = new StackOfIntegers();
        int factor = 2;
        while (number > 1) {
            while (number % factor == 0) {
                factors.push(factor);
                number /= factor;
            }
            factor++;
        }

        // Display factors in decreasing order
        System.out.print("Smallest factors in decreasing order: ");
        while (!factors.isEmpty()) {
            System.out.print(factors.pop());
            if (!factors.isEmpty()) {
                System.out.print(", ");
            }
        }
    }
}
```

Q4 B) Design a class named Fan to represent a fan. The class contains: ■ Three constants named SLOW, MEDIUM, and FAST with the values 1, 2, and 3 to denote the fan speed. ■ A private int data field named speed that specifies the speed of the fan (the default is SLOW). ■ A private boolean data field named on that specifies whether the fan is on (the default is false). ■ A private double data field named radius that specifies the radius of the fan (the default is 5). ■ A string data field named color that specifies the color of the fan (the default is blue). ■ The accessor and mutator methods for all four data fields. ■ A no-arg constructor that creates a default fan. ■ A method named toString() that returns a string description for the PCC CS593.3 6 20+ 20 fan. If the fan is on, the method returns the fan speed, color, and radius in one combined string. If the fan is not on, the method returns the fan color and radius along with the string "fan is off" in one combined string. Implement the class. Write a program that creates two Fan objects. Assign maximum speed, radius 10, color yellow, and turn it on to the first object. Assign medium speed, radius 5, color blue, and turn it off to the second object. Display the objects by invoking their toString method.

```
import java.util.Stack;

// StackOfIntegers class
class StackOfIntegers {
    private Stack<Integer> stack;

    public StackOfIntegers() {
        stack = new Stack<>();
    }

    public void push(int value) {
        stack.push(value);
    }

    public int pop() {
        return stack.pop();
    }

    public boolean isEmpty() {
        return stack.isEmpty();
    }
}

// Main class
public class Factorization {
    public static void main(String[] args) {
        java.util.Scanner scanner = new java.util.Scanner(System.in);
```

```
        // Prompt user for input
        System.out.print("Enter a positive integer: ");
        int number = scanner.nextInt();

        // Validate input
        if (number <= 0) {
            System.out.println("Please enter a positive integer.");
            return;
        }

        // Find smallest factors
        StackOfIntegers factors = new StackOfIntegers();
        int factor = 2;
        while (number > 1) {
            while (number % factor == 0) {
                factors.push(factor);
                number /= factor;
            }
            factor++;
        }

        // Display factors in decreasing order
        System.out.print("Smallest factors in decreasing order: ");
        while (!factors.isEmpty()) {
            System.out.print(factors.pop());
            if (!factors.isEmpty()) {
                System.out.print(", ");
            }
        }
    }
}
```

```
        // Prompt user for input
        System.out.print("Enter a positive integer: ");
        int number = scanner.nextInt();

        // Validate input
        if (number <= 0) {
            System.out.println("Please enter a positive integer.");
            return;
```

```
        }

        // Find smallest factors
        StackOfIntegers factors = new StackOfIntegers();
        int factor = 2;
        while (number > 1) {
            while (number % factor == 0) {
                factors.push(factor);
                number /= factor;
            }
            factor++;
        }

        // Display factors in decreasing order
        System.out.print("Smallest factors in decreasing order: ");
        while (!factors.isEmpty()) {
            System.out.print(factors.pop());
            if (!factors.isEmpty()) {
                System.out.print(", ");
            }
        }
    }
}
```

```
        // Prompt user for input
        System.out.print("Enter a positive integer: ");
        int number = scanner.nextInt();

        // Validate input
        if (number <= 0) {
            System.out.println("Please enter a positive integer.");
            return;
        }

        // Find smallest factors
        StackOfIntegers factors = new StackOfIntegers();
        int factor = 2;
        while (number > 1) {
            while (number % factor == 0) {
                factors.push(factor);
                number /= factor;
```

```
            }
            factor++;
        }

        // Display factors in decreasing order
        System.out.print("Smallest factors in decreasing order: ");
        while (!factors.isEmpty()) {
            System.out.print(factors.pop());
            if (!factors.isEmpty()) {
                System.out.print(", ");
            }
        }
    }
}



    }

    public boolean isOn() {
        return on;
    }

    public void setOn(boolean on) {
        this.on = on;
    }

    public double getRadius() {
        return radius;
    }

    public void setRadius(double radius) {
        this.radius = radius;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    // toString method
```

```
    @Override
    public String toString() {
        if (on) {
            return "Fan speed: " + speed + ", color: " + color + ", radius: " + radius;
        } else {
            return "Fan color: " + color + ", radius: " + radius + ", fan is off";
        }
    }
}

// Main class
public class FanTest {
    public static void main(String[] args) {
        // Create first Fan object
        Fan fan1 = new Fan();
        fan1.setSpeed(Fan.FAST);
        fan1.setRadius(10);
        fan1.setColor("yellow");
        fan1.setOn(true);

        // Create second Fan object
        Fan fan2 = new Fan();
        fan2.setSpeed(Fan.MEDIUM);
        fan2.setRadius(5);
        fan2.setColor("blue");
        fan2.setOn(false);

        // Display the fans
        System.out.println("Fan 1: " + fan1);
        System.out.println("Fan 2: " + fan2);
    }
}
```

Q5 A) Write a program that launches 1,000 threads. Each thread adds 1 to a variable sum that initially is 0. Define an Integer wrapper object to hold sum. Run the program with and without synchronization and print the output in both of the cases.

```
public class ThreadSumTest {

    private static Integer sum = 0;

    public static void main(String[] args) {
        // Without synchronization
        System.out.println("Running without synchronization:");
```

```
        runThreads(false);

        // Reset sum
        sum = 0;

        // With synchronization
        System.out.println("\nRunning with synchronization:");
        runThreads(true);
    }

    private static void runThreads(boolean synchronizedMode) {
        Thread[] threads = new Thread[1000];

        for (int i = 0; i < threads.length; i++) {
            threads[i] = new Thread(() -> {
                if (synchronizedMode) {
                    synchronized (sum) {
                        sum++;
                    }
                } else {
                    sum++;
                }
            });
        }

        for (Thread thread : threads) {
            thread.start();
        }

        for (Thread thread : threads) {
            try {
                thread.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println("Final sum: " + sum);
    }
}
```

Q5 B) Write a program that creates an ArrayList and adds a Loan object, a Date object, a string, and a Circle object to the list, and use a loop to display all the elements in the list by invoking the object's toString() method..

```
import java.util.ArrayList;
import java.util.Date;

class Loan {
   private double amount;
   private double interestRate;

   public Loan(double amount, double interestRate) {
      this.amount = amount;
      this.interestRate = interestRate;
   }

   @Override
   public String toString() {
      return "Loan[amount=" + amount + ", interestRate=" + interestRate + "]";
   }
}

class Circle {
   private double radius;

   public Circle(double radius) {
      this.radius = radius;
   }

   @Override
   public String toString() {
      return "Circle[radius=" + radius + "]";
   }
}

public class ObjectListTest {

   public static void main(String[] args) {
      ArrayList<Object> list = new ArrayList<>();

      // Add objects to the list
      list.add(new Loan(10000, 5.5));
      list.add(new Date());
      list.add("This is a string");
      list.add(new Circle(7.5));

      // Display all elements using their toString() method
```

```
        for (Object obj : list) {
            System.out.println(obj.toString());
        }
    }
}




class Student extends Person {
    public static final String FRESHMAN = "Freshman";
    public static final String SOPHOMORE = "Sophomore";
    public static final String JUNIOR = "Junior";
    public static final String SENIOR = "Senior";

    private String classStatus;

    public Student(String name, String address, String phone, String email, String
classStatus) {
        super(name, address, phone, email);
        this.classStatus = classStatus;
    }

    @Override
    public String toString() {
        return "Student: " + super.toString() + ", Class Status: " + classStatus;
    }
}

class Employee extends Person {
    private String office;
    private double salary;

    public Employee(String name, String address, String phone, String email, String office,
double salary) {
        super(name, address, phone, email);
        this.office = office;
        this.salary = salary;
    }

    @Override
    public String toString() {
        return "Employee: " + super.toString() + ", Office: " + office + ", Salary: " + salary;
    }
```

```
}

class Faculty extends Employee {
   private String officeHours;
   private String rank;

   public Faculty(String name, String address, String phone, String email, String office,
                  double salary, String officeHours, String rank) {
      super(name, address, phone, email, office, salary);
      this.officeHours = officeHours;
      this.rank = rank;
   }

   @Override
   public String toString() {
      return "Faculty: " + super.toString() + ", Rank: " + rank + ", Office Hours: " +
officeHours;
   }
}

class Staff extends Employee {
   private String title;

   public Staff(String name, String address, String phone, String email, String office,
                double salary, String title) {
      super(name, address, phone, email, office, salary);
      this.title = title;
   }

   @Override
   public String toString() {
      return "Staff: " + super.toString() + ", Title: " + title;
   }
}

public class Main {
   public static void main(String[] args) {
      Person person = new Person("John Doe", "123 Main St", "555-1234",
"john@example.com");
      Student student = new Student("Alice Smith", "456 Elm St", "555-5678",
"alice@example.com", Student.FRESHMAN);
      Employee employee = new Employee("Bob Johnson", "789 Oak St", "555-8765",
"bob@example.com", "Room 101", 50000);
```

```
        Faculty faculty = new Faculty("Dr. Carol White", "321 Pine St", "555-4321",
"carol@example.com",
                                "Room 202", 70000, "MWF 10-11 AM", "Professor");
        Staff staff = new Staff("Eve Davis", "654 Maple St", "555-9876", "eve@example.com",
                                "Room 303", 40000, "Administrative Assistant");

        System.out.println(person);
        System.out.println(student);
        System.out.println(employee);
        System.out.println(faculty);
        System.out.println(staff);
    }
}
```

Q7 A) Create the GeometricObject class to implement the Comparable interface, and define a static max method in the GeometricObject class for finding the larger of two GeometricObject objects. Implement the new GeometricObject class. Write a program that uses the max method to find the larger of two circles and the larger of two rectangles.

```
import java.lang.Math;

abstract class GeometricObject implements Comparable<GeometricObject> {
    public abstract double getArea();

    @Override
    public int compareTo(GeometricObject other) {
        return Double.compare(this.getArea(), other.getArea());
    }

    public static GeometricObject max(GeometricObject g1, GeometricObject g2) {
        return (g1.compareTo(g2) >= 0) ? g1 : g2;
    }
}

class Circle extends GeometricObject {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    @Override
    public double getArea() {
```

```
        return Math.PI * radius * radius;
    }

    @Override
    public String toString() {
        return "Circle with radius: " + radius + ", Area: " + getArea();
    }
}

class Rectangle extends GeometricObject {
    private double width;
    private double height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    @Override
    public double getArea() {
        return width * height;
    }

    @Override
    public String toString() {
        return "Rectangle with width: " + width + ", height: " + height + ", Area: " + getArea();
    }
}

public class Main {
    public static void main(String[] args) {
        Circle circle1 = new Circle(5);
        Circle circle2 = new Circle(3);

        Rectangle rectangle1 = new Rectangle(4, 6);
        Rectangle rectangle2 = new Rectangle(5, 5);

        GeometricObject largerCircle = GeometricObject.max(circle1, circle2);
        GeometricObject largerRectangle = GeometricObject.max(rectangle1, rectangle2);

        System.out.println("Larger Circle: " + largerCircle);
        System.out.println("Larger Rectangle: " + largerRectangle);
    }
}
```

Q7 B) Write a program that meets the following requirements: ■ Creates an array with 100 randomly chosen integers. ■ Prompts the user to enter the index of the array, then displays the corresponding element value. If the specified index is out of bounds, display the message Out of Bounds.

```
import java.util.Random;
import java.util.Scanner;

public class RandomArrayAccess {
    public static void main(String[] args) {
        Random random = new Random();
        int[] numbers = new int[100];

        // Fill the array with random integers between 0 and 99
        for (int i = 0; i < numbers.length; i++) {
            numbers[i] = random.nextInt(100);
        }

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter an index (0-99): ");

        if (scanner.hasNextInt()) {
            int index = scanner.nextInt();

            if (index >= 0 && index < numbers.length) {
                System.out.println("Value at index " + index + ": " + numbers[index]);
            } else {
                System.out.println("Out of Bounds");
            }
        } else {
            System.out.println("Invalid input. Please enter an integer.");
        }

        scanner.close();
    }
}
```

Q8 A) Design a class named Rectangle to represent a rectangle. The class contains: ■ Two double data fields named width and height that specify the PCC CS 593.3 6 20+ 20 width and height of the rectangle. The default values are 1 for both width and height. ■ A no-arg constructor that creates a default rectangle. ■ A constructor that creates a

rectangle with the specified width and height. ■ A method named getArea() that returns the area of this rectangle. ■ A method named getPerimeter() that returns the perimeter. Write a program that creates two Rectangle objects—one with width 4 and height 40 and the other with width 3.5 and height 35.9. Display the width, height, area, and perimeter of each rectangle in this order.

```
class Rectangle {
    private double width;
    private double height;

    public Rectangle() {
        this.width = 1;
        this.height = 1;
    }

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    public double getArea() {
        return width * height;
    }

    public double getPerimeter() {
        return 2 * (width + height);
    }

    public double getWidth() {
        return width;
    }

    public double getHeight() {
        return height;
    }
}

public class Main {
    public static void main(String[] args) {
        Rectangle rectangle1 = new Rectangle(4, 40);
        Rectangle rectangle2 = new Rectangle(3.5, 35.9);

        System.out.println("Rectangle 1:");
        System.out.println("Width: " + rectangle1.getWidth());
```

```
        System.out.println("Height: " + rectangle1.getHeight());
        System.out.println("Area: " + rectangle1.getArea());
        System.out.println("Perimeter: " + rectangle1.getPerimeter());

        System.out.println("\nRectangle 2:");
        System.out.println("Width: " + rectangle2.getWidth());
        System.out.println("Height: " + rectangle2.getHeight());
        System.out.println("Area: " + rectangle2.getArea());
        System.out.println("Perimeter: " + rectangle2.getPerimeter());
    }
}
```

Q 8B) Design a class named Location for locating a maximal value and its location in a two-dimensional array. The class contains public data fields row, column, and maxValue that store the maximal value and its indices in a two-dimensional array with row and column as int types and maxValue as a double type.

```
import java.util.Random;

class Location {
    public int row;
    public int column;
    public double maxValue;

    public Location(int row, int column, double maxValue) {
        this.row = row;
        this.column = column;
        this.maxValue = maxValue;
    }

    public static Location locateLargest(double[][] a) {
        Location location = new Location(0, 0, a[0][0]);

        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[i].length; j++) {
                if (a[i][j] > location.maxValue) {
                    location.maxValue = a[i][j];
                    location.row = i;
                    location.column = j;
                }
            }
        }

        return location;
```

```
    }
}

public class Main {
    public static void main(String[] args) {
        Random random = new Random();
        double[][] array = new double[5][5];

        for (int i = 0; i < array.length; i++) {
            for (int j = 0; j < array[i].length; j++) {
                array[i][j] = random.nextDouble() * 100; // Random values between 0 and 100
            }
        }

        Location largestLocation = Location.locateLargest(array);

        System.out.println("2D Array:");
        for (double[] row : array) {
            for (double value : row) {
                System.out.printf("%.2f ", value);
            }
            System.out.println();
        }

        System.out.println("\nMax Value: " + largestLocation.maxValue);
        System.out.println("Location: Row " + largestLocation.row + ", Column " +
largestLocation.column);
    }
}
```

Q 9 Design a class named Account that contains: ■ A private int data field named id for the account (default 0). ■ A private double data field named balance for the account (default 0). ■ A private double data field named annualInterestRate that stores the current interest rate (default 0). Assume all accounts have the same interest rate. ■ A private Date data field named dateCreated that stores the date when the account was created. ■ A no-arg constructor that creates a default account. ■ A constructor that creates an account with the specified id and initial balance. ■ The accessor and mutator methods for id, balance, and annualInterestRate. ■

The accessor method for dateCreated. ■ A method named getMonthlyInterestRate() that returns the monthly interest rate. ■ A method named getMonthlyInterest() that returns the monthly interest. ■ A method named withdraw that withdraws a specified amount from the account. ■ A method named deposit that deposits a specified amount to the account. Implement the class. (Hint: The method getMonthlyInterest() is to return monthly interest, not the interest rate. Monthly interest is balance × monthlyInterestRate. monthlyInterestRate is annualInterestRate / 12. PCC CS593.5 6 40 Note that annualInterestRate is a percentage, e.g., like 4.5%. You need to divide it by 100.) Write a program that creates an Account object with an account ID of 1122, a balance of $20,000, and an annual interest rate of 4.5%. Use the withdraw method to withdraw $2,500, use the deposit method to deposit $3,000, and print the balance, the monthly interest, and the date when this account was created

```
import java.util.Date;

class Account {
   private int id = 0;
   private double balance = 0;
   private double annualInterestRate = 0;
   private Date dateCreated;

   public Account() {
      this.dateCreated = new Date();
   }

   public Account(int id, double initialBalance) {
      this.id = id;
      this.balance = initialBalance;
      this.dateCreated = new Date();
   }

   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public double getBalance() {
      return balance;
   }

   public void setBalance(double balance) {
```

```
        this.balance = balance;
    }

    public double getAnnualInterestRate() {
        return annualInterestRate;
    }

    public void setAnnualInterestRate(double annualInterestRate) {
        this.annualInterestRate = annualInterestRate;
    }

    public Date getDateCreated() {
        return dateCreated;
    }

    public double getMonthlyInterestRate() {
        return annualInterestRate / 100 / 12; // Convert percentage to decimal
    }

    public double getMonthlyInterest() {
        return balance * getMonthlyInterestRate();
    }

    public void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
        } else {
            System.out.println("Insufficient funds for withdrawal.");
        }
    }

    public void deposit(double amount) {
        balance += amount;
    }
}

public class Main {
    public static void main(String[] args) {
        Account account = new Account(1122, 20000);
        account.setAnnualInterestRate(4.5);

        account.withdraw(2500);
        account.deposit(3000);
```

```
        System.out.printf("Account ID: %d%n", account.getId());
        System.out.printf("Balance: $%.2f%n", account.getBalance());
        System.out.printf("Monthly Interest: $%.2f%n", account.getMonthlyInterest());
        System.out.printf("Date Created: %s%n", account.getDateCreated());
    }
}
```

Q 10 Create a data file with 1,000 lines. Each line in the file consists of a faculty member's first name, last name, rank, and salary. The faculty member's first name and last name for the ith line are FirstNamei and LastNamei. The rank is randomly generated as assistant, associate, and full. The salary is randomly generated as a number with two digits after the decimal point. The salary for an assistant professor should be in the range from 50,000 to 80,000, for associate professor from 60,000 to 110,000, and for full professor from 75,000 to 130,000. Save the file in Salary.txt. Here are some sample data: FirstName1 LastName1 assistant 60055.95 FirstName2 LastName2 associate 81112.45 . . . FirstName1000 LastName1000 full 92255.21

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;

public class FacultyDataGenerator {
    public static void main(String[] args) {
        String[] ranks = {"assistant", "associate", "full"};
        Random random = new Random();

        try (BufferedWriter writer = new BufferedWriter(new FileWriter("Salary.txt"))) {
            for (int i = 1; i <= 1000; i++) {
                String firstName = "FirstName" + i;
                String lastName = "LastName" + i;
                String rank = ranks[random.nextInt(ranks.length)];
                double salary = generateSalary(rank, random);

                writer.write(String.format("%s %s %s %.2f%n", firstName, lastName, rank, salary));
            }
            System.out.println("Data successfully written to Salary.txt");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static double generateSalary(String rank, Random random) {
```

```
        double salary;
        switch (rank) {
            case "assistant":
                salary = 50000 + (random.nextDouble() * (30000)); // 50,000 to 80,000
                break;
            case "associate":
                salary = 60000 + (random.nextDouble() * (50000)); // 60,000 to 110,000
                break;
            case "full":
                salary = 75000 + (random.nextDouble() * (55000)); // 75,000 to 130,000
                break;
            default:
                salary = 0; // This should not happen
        }
        return salary;
    }
}
```