ASSIGNMENT-1

1.Write a Java code snippet that comprises a Car class and a CarTester class. The Car class should define private fields for make and model, along with a parameterized constructor and getter/setter methods for these attributes. In the CarTester class, instantiate two instances of the Car class: myCar1 with a specified make and model, and myCar2 with null values. After instantiation, the CarTester class should retrieve and print the initial make and model of both cars. Then, it should set new values for myCar2 using setter methods and print the updated make and model.

```
class Car {
   private String make;
   private String model;

   public Car(String make, String model) {
      this.make = make;
      this.model = model;
   }

   public String getMake() {
      return make;
   }

   public void setMake(String make) {
      this.make = make;
   }

   public String getModel() {
      return model;
   }

   public void setModel(String model) {
      this.model = model;
   }
}
public class Q1 {

        public static void main(String[] args) {
            Car myCar1 = new Car("Toyota", "Camry");
            Car myCar2 = new Car(null, null);

            System.out.println("Initial make and model of myCar1: " + myCar1.getMake() + " " + myCar1.getModel());
            System.out.println("Initial make and model of myCar2: " + myCar2.getMake() + " " + myCar2.getModel());

            myCar2.setMake("Honda");
            myCar2.setModel("Civic");

            System.out.println("Updated make and model of myCar2: " + myCar2.getMake() + " " + myCar2.getModel());

        }

}
```

Output:-

Initial make and model of myCar1: Toyota Camry
Initial make and model of myCar2: null null
Updated make and model of myCar2: Honda Civic

2. Design a Java class called Rectangle with private attributes length and width. Include a constructor to initialize these attributes, as well as setter and getter methods for each attribute. Additionally, implement methods to calculate the area and perimeter of the rectangle. Write a main method to create an object of the Rectangle class, set values for its attributes, and display the area and perimeter.

```
public class Rectangle{
private double length;
   private double width;

   public Rectangle(double length, double width) {
```

```
        if (length <= 0 || width <= 0) {
            throw new IllegalArgumentException("Length and width must be positive numbers.");
        }
        this.length = length;
        this.width = width;
    }

    public double getLength() {
        return length;
    }

    public void setLength(double length) {
        if (length <= 0) {
            throw new IllegalArgumentException("Length must be a positive number.");
        }
        this.length = length;
    }

    public double getWidth() {
        return width;
    }

    public void setWidth(double width) {
        if (width <= 0) {
            throw new IllegalArgumentException("Width must be a positive number.");
        }
        this.width = width;
    }

    public double calculateArea() {
        return length * width;
    }

    public double calculatePerimeter() {
        return 2 * (length + width);
    }

    public static void main(String[] args) {
        Rectangle rectangle = new Rectangle(5, 4);
        System.out.println("Area: " + rectangle.calculateArea());
        System.out.println("Perimeter: " + rectangle.calculatePerimeter());

        rectangle.setLength(7);
        rectangle.setWidth(3);
        System.out.println("Updated area: " + rectangle.calculateArea());
        System.out.println("Updated perimeter: " + rectangle.calculatePerimeter());
    }
```

Output:-
Area: 20.0
Perimeter: 18.0
Updated area: 21.0
Updated perimeter: 20.0

3. Write a Java program that defines a 'Point' class with attributes 'X' and 'Y', and includes a parameterized constructor to initialize these attributes. Implement a copy constructor to create a new point object with the same attribute values. Ensure that modifications made to one object do not affect the other. Utilize getter and setter methods to retrieve and update the attribute values.

```
public class Point {
    private int x;
    private int y;
    public Point(int x, int y) {
```

```
        this.x = x;
        this.y = y;
    }
    public Point(Point other) {
        this(other.x, other.y);
    }
    public int getX() {
        return x;
    }
    public void setX(int x) {
        this.x = x;
    }
    public int getY() {
        return y;
    }
    public void setY(int y) {
        this.y = y;
    }
    public static void main(String[] args) {
        Point p1 = new Point(3, 4);
        Point p2 = new Point(p1);
        System.out.println("Initial values of p1: (" + p1.getX() + ", " + p1.getY() + ")");
        System.out.println("Initial values of p2: (" + p2.getX() + ", " + p2.getY() + ")");
        p1.setX(5);
        p1.setY(6);
        System.out.println("Updated values of p1: (" + p1.getX() + ", " + p1.getY() + ")");
        System.out.println("Values of p2 unchanged: (" + p2.getX() + ", " + p2.getY() + ")");
    }
}
```

Output:-

Initial values of p1: (3, 4)
Initial values of p2: (3, 4)
Updated values of p1: (5, 6)
Values of p2 unchanged: (3, 4)

4. Write a Java code snippet comprising two classes: Laptop and Main. The Laptop class defines private fields model and price, alongside setter methods for each attribute. Additionally, it overrides the toString() method to return a string representation of the laptop's model and price. In the Main class, create an instance of Laptop, setting its model using the setter method. Then, print the laptop object using the toString() method. Describe the functionality of the toString() method in the Laptop class and explain how it is utilized in the Main class.

```
public class Laptop {
    private String model;
    private double price;

    public String getModel() {
        return model;
    }
    public void setModel(String model) {
        this.model = model;
    }
    public double getPrice() {
        return price;
    }
    public void setPrice(double price) {
        this.price = price;
    }
    public String toString() {
        return "Laptop [model=" + model + ", price=" + price + "]";
```

```
    }
}
public class Main {
    public static void main(String[] args) {
        Laptop laptop = new Laptop();
        laptop.setModel("Dell XPS 15");
        laptop.setPrice(1500);

        System.out.println("Laptop: " + laptop);
    }
}
```

Output:-

Laptop: Laptop [model=Dell XPS 15, price=1500.0]

5. Developing a Simple College Management System in Java Create a Java program for managing colleges and students. The provided classes model the relationship between colleges and students. The College class represents a college with attributes collegeName and collegeLoc. The Student class represents a student with attributes studentId, studentName, and a reference to a College object. Enhance the Student class by adding a constructor that assigns a student ID, name, and college information. Additionally, add a method named displayStudentInfo() that prints the student's ID, name, and the college information in which they are enrolled. In the Main class, instantiate at least two College objects and at least two Student objects. Enroll each student in one of the colleges. Then, display the information of all colleges and all students using the appropriate methods.

```
public class College {
    private String collegeName;
    private String collegeLoc;

    public College(String collegeName, String collegeLoc) {
        this.collegeName = collegeName;
        this.collegeLoc = collegeLoc;
    }
    public String getCollegeName() {
        return collegeName;
    }
    public void setCollegeName(String collegeName) {
        this.collegeName = collegeName;
    }
    public String getCollegeLoc() {
        return collegeLoc;
    }
    public void setCollegeLoc(String collegeLoc) {
        this.collegeLoc = collegeLoc;
    }
    public String toString() {
        return "College [collegeName=" + collegeName + ", collegeLoc=" + collegeLoc + "]";
    }
}
public class Student {
    private int studentId;
    private String studentName;
    private College college;

    public Student(int studentId, String studentName, College college) {
        this.studentId = studentId;
        this.studentName = studentName;
        this.college = college;
    }
    public int getStudentId() {
        return studentId;
    }
    public void setStudentId(int studentId) {
        this.studentId = studentId;
```

```
    }
    public String getStudentName() {
        return studentName;
    }
    public void setStudentName(String studentName) {
        this.studentName = studentName;
    }
    public College getCollege() {
        return college;
    }
    public void setCollege(College college) {
        this.college = college;
    }
    public void displayStudentInfo() {
        System.out.println("Student ID: " + studentId);
        System.out.println("Student Name: " + studentName);
        System.out.println("College Name: " + college.getCollegeName());
        System.out.println("College Location: " + college.getCollegeLoc());
    }
}
public class Main {
    public static void main(String[] args) {
        College college1 = new College("MIT", "Cambridge, MA");
        College college2 = new College("Stanford", "Stanford, CA");

        Student student1 = new Student(1, "John Doe", college1);
        Student student2 = new Student(2, "Jane Doe", college2);

        System.out.println("College 1: " + college1);
        System.out.println("College 2: " + college2);

        System.out.println("\nStudent 1 Info:");
        student1.displayStudentInfo();

        System.out.println("\nStudent 2 Info:");
        student2.displayStudentInfo();
    }
}
```

Output:-

College 1: College [collegeName=MIT, collegeLoc=Cambridge, MA]
College 2: College [collegeName=Stanford, collegeLoc=Stanford, CA]

Student 1 Info:
Student ID: 1
Student Name: John Doe
College Name: MIT
College Location: Cambridge, MA

Student 2 Info:
Student ID: 2
Student Name: Jane Doe
College Name: Stanford
College Location: Stanford, CA

Q6.**Library System Assignment:**

Task: Develop a Java program for a library system incorporating encapsulation, abstraction, and inheritance.

i. LibraryResource Class:

- Abstract class with private attributes for title and author.

- Constructor, getters, setters, and an abstract method displayDetails().

ii. Book Class:
- Subclass of LibraryResource with additional attribute pageCount.
- Constructor, getters, setters, and displayDetails() method override.
iii. Magazine Class:
- Subclass of LibraryResource with additional attribute issueDate.
- Constructor, getters, setters, and displayDetails() method override.
iv. DVD Class:
- Subclass of LibraryResource with additional attribute duration.
- Constructor, getters, setters, and displayDetails() method override.
v. LibrarySystem Class (Main):
- Instantiate various library resources (e.g., Book, Magazine, DVD).
- Call displayDetails() for each instance to show resource information.

```
public abstract class LibraryResource {
    private String title;
    private String author;

    public LibraryResource(String title, String author) {
        this.title = title;
        this.author = author;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getAuthor() {
        return author;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
    public abstract void displayDetails();
}
public class Book extends LibraryResource {
    private int pageCount;

    public Book(String title, String author, int pageCount) {
        super(title, author);
        this.pageCount = pageCount;
    }
    public int getPageCount() {
        return pageCount;
    }
    public void setPageCount(int pageCount) {
        this.pageCount = pageCount;
    }
    public void displayDetails() {
        System.out.println("Title: " + getTitle());
        System.out.println("Author: " + getAuthor());
        System.out.println("Page Count: " + pageCount);
    }
}
public class Magazine extends LibraryResource {
    private String issueDate;

    public Magazine(String title, String author, String issueDate) {
        super(title, author);
        this.issueDate = issueDate;
```

```
    }
    public String getIssueDate() {
        return issueDate;
    }
    public void setIssueDate(String issueDate) {
        this.issueDate = issueDate;
    }
    public void displayDetails() {
        System.out.println("Title: " + getTitle());
        System.out.println("Author: " + getAuthor());
        System.out.println("Issue Date: " + issueDate);
    }
}
public class DVD extends LibraryResource {
    private String duration;

    public DVD(String title, String author, String duration) {
        super(title, author);
        this.duration = duration;
    }
    public String getDuration() {
        return duration;
    }
    public void setDuration(String duration) {
        this.duration = duration;
    }
    public void displayDetails() {
        System.out.println("Title: " + getTitle());
        System.out.println("Author: " + getAuthor());
        System.out.println("Duration: " + duration);
    }
}
public class LibrarySystem {
    public static void main(String[] args) {
        LibraryResource book = new Book("The Great Gatsby", "F. Scott Fitzgerald", 180);
        LibraryResource magazine = new Magazine("National Geographic", "Editorial Staff", "March 2023");
        LibraryResource dvd = new DVD("The Matrix", "Lana Wachowski, Lilly Wachowski", "2 hours, 16 minutes");

        book.displayDetails();
        System.out.println();
        magazine.displayDetails();
        System.out.println();
        dvd.displayDetails();
    }
}
```

Output:-

Title: The Great Gatsby
Author: F. Scott Fitzgerald
Page Count: 180

Title: National Geographic
Author: Editorial Staff
Issue Date: March 2023

Title: The Matrix
Author: Lana Wachowski, Lilly Wachowski
Duration: 2 hours, 16 minutes

7.Consider a scenario where you are tasked with developing a simple banking application using Java, employing the concept of polymorphism. Your application should consist of three classes: Account, SavingsAccount, and CurrentAccount. The Account class serves as the base class with private attributes for account number and balance, along with abstract

methods for deposit and withdrawal. The SavingsAccount class, a subclass of Account, should include an additional attribute for interest rate and override the deposit method to calculate interest, as well as override the withdrawal method to ensure a sufficient balance. Similarly, the CurrentAccount class, also a subclass of Account, should incorporate an overdraft limit attribute and override the withdrawal method to check the overdraft limit. Implement the classes as described, ensuring proper encapsulation and abstraction. Finally, create a BankingApplication class (Main) to demonstrate the polymorphic behavior by creating instances of both SavingsAccount and CurrentAccount, testing deposit and withdrawal operations, and displaying account details.

```
public abstract class Account {
    private int accountNumber;
    private double balance;
    public Account(int accountNumber, double balance) {
        this.accountNumber = accountNumber;
        this.balance = balance;
    }
    public int getAccountNumber() {
        return accountNumber;
    }
    public void setAccountNumber(int accountNumber) {
        this.accountNumber = accountNumber;
    }
    public double getBalance() {
        return balance;
    }
    public void setBalance(double balance) {
        this.balance = balance;
    }
    public abstract void deposit(double amount);
    public abstract boolean withdraw(double amount);
}
public class SavingsAccount extends Account {
    private double interestRate;
    public SavingsAccount(int accountNumber, double balance, double interestRate) {
        super(accountNumber, balance);
        this.interestRate = interestRate;
    }
    public double getInterestRate() {
        return interestRate;
    }
    public void setInterestRate(double interestRate) {
        this.interestRate = interestRate;
    }
    public void deposit(double amount) {
        setBalance(getBalance() + amount);
    }
    public boolean withdraw(double amount) {
        if (getBalance() >= amount) {
            setBalance(getBalance() - amount);
            return true;
        }
        return false;
    }
    public String toString() {
        return "SavingsAccount [accountNumber=" + getAccountNumber() + ", balance=" + getBalance() + ", interestRate="
+ interestRate + "]";
    }
}
public class CurrentAccount extends Account {
    private double overdraftLimit;
    public CurrentAccount(int accountNumber, double balance, double overdraftLimit) {
        super(accountNumber, balance);
        this.overdraftLimit = overdraftLimit;
```

```
    }
    public double getOverdraftLimit() {
        return overdraftLimit;
    }
    public void setOverdraftLimit(double overdraftLimit) {
        this.overdraftLimit = overdraftLimit;
    }
    public void deposit(double amount) {
        setBalance(getBalance() + amount);
    }
    public boolean withdraw(double amount) {
        if (getBalance() + overdraftLimit >= amount) {
            setBalance(getBalance() - amount);
            return true;
        }
        return false;
    }
    public String toString() {
        return "CurrentAccount [accountNumber=" + getAccountNumber() + ", balance=" + getBalance() + ",
overdraftLimit=" + overdraftLimit + "]";
    }
}

public class BankingApplication {
    public static void main(String[] args) {
        Account savingsAccount = new SavingsAccount(12345, 1000, 0.05);
        Account currentAccount = new CurrentAccount(67890, 500, 200);

        System.out.println("Savings Account: " + savingsAccount);
        savingsAccount.deposit(200);
        savingsAccount.withdraw(50);
        System.out.println("Updated Savings Account: " + savingsAccount);

        System.out.println();

        System.out.println("Current Account: " + currentAccount);
        currentAccount.deposit(300);
        currentAccount.withdraw(800);
        System.out.println("Updated Current Account: " + currentAccount);
    }
}
```

Output:-

Savings Account: SavingsAccount [accountNumber=12345, balance=1000.0, interestRate=0.05]
Updated Savings Account: SavingsAccount [accountNumber=12345, balance=1150.0, interestRate=0.05]

Current Account: CurrentAccount [accountNumber=67890, balance=500.0, overdraftLimit=200.0]
Updated Current Account: CurrentAccount [accountNumber=67890, balance=0.0, overdraftLimit=200.0]

8.Write a Java program that illustrates the concepts of interfaces, method overriding, and method overloading. Begin by defining an interface named Vehicle with two abstract methods: accelerate() and brake(). Then, create two classes, Car and Bicycle, both of which implement the Vehicle interface. In the Car class, override the accelerate() and brake() methods to print appropriate messages indicating the acceleration and braking actions for a car. Similarly, in the Bicycle class, override the same methods to display messages specific to a bicycle's acceleration and braking. Additionally, implement method overloading in both classes by providing multiple versions of the accelerate() method, each accepting different parameters such as speed and duration. Finally, create a Main class to instantiate objects of both Car and Bicycle classes, test their overridden methods, and demonstrate method overloading by invoking different versions of the accelerate() method.

```
public interface Vehicle {
    void accelerate();
    void brake();
}
```

```
public class Car implements Vehicle {
     public void accelerate() {
       System.out.println("Car is accelerating.");
   }
   public void brake() {
       System.out.println("Car is braking.");
   }
   public void accelerate(int speed) {
       System.out.println("Car is accelerating to " + speed + " mph.");
   }
   public void accelerate(int speed, int duration) {
       System.out.println("Car is accelerating to " + speed + " mph for " + duration + " seconds.");
   }
}
public class Bicycle implements Vehicle {
     public void accelerate() {
       System.out.println("Bicycle is pedaling faster.");
   }
   public void brake() {
       System.out.println("Bicycle is slowing down.");
   }
   public void accelerate(int speed) {
       System.out.println("Bicycle is pedaling faster to reach " + speed + " mph.");
   }
   public void accelerate(int speed, int duration) {
       System.out.println("Bicycle is pedaling faster to reach " + speed + " mph for " + duration + " seconds.");
   }
}
public class Main {
   public static void main(String[] args) {
       Vehicle car = new Car();
       Vehicle bicycle = new Bicycle();

       car.accelerate();
       car.brake();
       car.accelerate(20);
       car.accelerate(20, 10);

       System.out.println();
       bicycle.accelerate();
       bicycle.brake();
       bicycle.accelerate(15);
       bicycle.accelerate(15, 15);
   }
}
```

Output:-

Car is accelerating.
Car is braking.
Car is accelerating to 20 mph.
Car is accelerating to 20 mph for 10 seconds.

Bicycle is pedaling faster.
Bicycle is slowing down.
Bicycle is pedaling faster to reach 15 mph.
Bicycle is pedaling faster to reach 15 mph for 15 seconds.

9.Design a Java program for managing student enrollment in a university, adhering to the principles of loose coupling and high cohesion. Your program should include classes for representing students (`Student`), courses (`Course`), and enrollment management (`Enrollment`). Ensure that the `Enrollment` class interacts with the other classes indirectly through an interface class (`EnrollmentSystem`). Implement methods for enrolling and

dropping students from courses, and displaying enrollment details. Create a `Main` class to demonstrate the system's functionality, with appropriate error handling and user-friendly output messages. Provide comments in your code explaining how loose coupling and high cohesion are maintained throughout the implementation.

```
import java.util.*;
interface EnrollmentSystem {
    void enrollStudent(Student student, Course course);
    void dropStudent(Student student, Course course);
    void displayEnrollmentDetails();
}
class Student {
    private int id;
    private String name;
    public Student(int id, String name) {
        this.id = id;
        this.name = name;
    }
    public int getId() {
        return id;
    }
    public String getName() {
        return name;
    }
}
class Course {
    private int courseId;
    private String courseName;
    public Course(int courseId, String courseName) {
        this.courseId = courseId;
        this.courseName = courseName;
    }
    public int getCourseId() {
        return courseId;
    }
    public String getCourseName() {
        return courseName;
    }
}
class Enrollment implements EnrollmentSystem {
    private Map<Student, Set<Course>> enrollments;

    public Enrollment() {
        enrollments = new HashMap<>();
    }
    public void enrollStudent(Student student, Course course) {
        enrollments.computeIfAbsent(student, k -> new HashSet<>()).add(course);
    }
     public void dropStudent(Student student, Course course) {
        enrollments.computeIfPresent(student, (k, v) -> {
            v.remove(course);
            return v.isEmpty() ? null : v;
        });
    }
    public void displayEnrollmentDetails() {
        System.out.println("Enrollment Details:");
        for (Map.Entry<Student, Set<Course>> entry : enrollments.entrySet()) {
            Student student = entry.getKey();
            Set<Course> courses = entry.getValue();
            System.out.println("Student ID: " + student.getId() + ", Name: " + student.getName());
            System.out.println("Enrolled Courses:");
            for (Course course : courses) {
```

```
            System.out.println("Course ID: " + course.getCourseId() + ", Name: " + course.getCourseName());
        }
        System.out.println();
    }
  }
}
public class Q2 {
  public static void main(String[] args) {
    EnrollmentSystem enrollmentSystem = new Enrollment();
    Student student1 = new Student(1, "Alice");
    Student student2 = new Student(2, "Bob");
    Course course1 = new Course(101, "Computer Science");
    Course course2 = new Course(102, "Mathematics");
    enrollmentSystem.enrollStudent(student1, course1);
    enrollmentSystem.enrollStudent(student1, course2);
    enrollmentSystem.enrollStudent(student2, course1);
    enrollmentSystem.displayEnrollmentDetails();

        enrollmentSystem.dropStudent(student1, course1);
    enrollmentSystem.displayEnrollmentDetails();
  }
}
```

Output:-

Enrollment Details:
Student ID: 1, Name: Alice
Enrolled Courses:
Course ID: 102, Name: Mathematics
Course ID: 101, Name: Computer Science

Student ID: 2, Name: Bob
Enrolled Courses:
Course ID: 101, Name: Computer Science

Enrollment Details:
Student ID: 1, Name: Alice
Enrolled Courses:
Course ID: 102, Name: Mathematics

Student ID: 2, Name: Bob
Enrolled Courses:
Course ID: 101, Name: Computer Science

ASSIGNMENT-2

1.Write a java program to create a Student class that has members, name, roll number, and age. Design the Student class in such a way that it can take the roll number as an integer or string. Create a driver class that creates student objects and invokes the methods.

```
import java.util.Scanner;
class Student {
  private String name;
  private Object rollNumber;    private int age;
  public Student(String name, Object rollNumber, int age) {
    this.name = name;
    this.rollNumber = rollNumber;
    this.age = age;
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
```

```
    }
    public Object getRollNumber() {
        return rollNumber;
    }
    public void setRollNumber(Object rollNumber) {
        if (rollNumber instanceof Integer || rollNumber instanceof String) {
            this.rollNumber = rollNumber;
        } else {
            throw new IllegalArgumentException("Roll number must be an integer or a string");
        }
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String toString() {
        return "Student{" + "name='" + name + '\''  ", rollNumber=" + rollNumber + ", age=" + age + '}';
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter name, roll number, and age: ");
        String name = scanner.nextLine();
        Object rollNumber = scanner.nextLine();
        int age = scanner.nextInt();
        scanner.nextLine();
        Student student1;
        try {
            student1 = new Student(name, rollNumber, age);
            System.out.println(student1);
        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Output:-

Enter name, roll number, and age:
sonali
4
20
Student{name='sonali', rollNumber=4, age=20}

2.Write a java program to create a Book class with member variables bookId, bookName, and price. Add the respective method and constructor to it. Create a driver class in that class and create two book objects. Compare the book objects according to their price. Print the details of the book objects. Note: Overload toString and equals method.

```
class Book {
    private int bookId;
    private String bookName;
    private double price;
    public Book(int bookId, String bookName, double price) {
        this.bookId = bookId;
        this.bookName = bookName;
        this.price = price;
    }
    public int getBookId() {
        return bookId;
    }
    public void setBookId(int bookId) {
```

```
        this.bookId = bookId;
    }
    public String getBookName() {
        return bookName;
    }
    public void setBookName(String bookName) {
        this.bookName = bookName;
    }
    public double getPrice() {
        return price;
    }
    public void setPrice(double price) {
        this.price = price;
    }
      public String toString() {
        return "Book{" +"bookId=" + bookId +", bookName='" + bookName + '\'' +", price=" + price + '}';
    }
    public boolean equals(Object obj) {
        if (obj == this) {
            return true;
        }
        if (!(obj instanceof Book)) {
            return false;
        }
        Book other = (Book) obj;
        return this.bookId == other.bookId && this.bookName.equals(other.bookName) && this.price == other.price;
    }
    public static int compare(Book book1, Book book2) {
        if (book1.price < book2.price) {
            return -1;
        } else if (book1.price > book2.price) {
            return 1;
        } else {
            return 0;
        }
    }
}
public class Main {
    public static void main(String[] args) {
        Book book1 = new Book(1, "Book1", 10.0);
        Book book2 = new Book(2, "Book2", 12.0);

        System.out.println("Book1: " + book1);
        System.out.println("Book2: " + book2);

        int comparisonResult = Book.compare(book1, book2);
        if (comparisonResult < 0) {
            System.out.println("Book1 is cheaper than Book2");
        } else if (comparisonResult > 0) {
            System.out.println("Book2 is cheaper than Book1");
        } else {
            System.out.println("Both books have the same price");
        }
    }
}
```

Output:-

Book1: Book{bookId=1, bookName='Book1', price=10.0}
Book2: Book{bookId=2, bookName='Book2', price=12.0}
Book1 is cheaper than Book2

3.Write a java program to create a Car class with member variables model, color, and speed. Add the respective method and constructor to it. Create a driver class in that class and create two car objects. Compare the car objects according to

their speed and print the details of the car that has a greater speed. Note: Overload compareTo method of Comparable interface.

```
import java.util.Comparator;
class Car implements Comparable<Car> {
    private String model;
    private String color;
    private int speed;
    public Car(String model, String color, int speed) {
        this.model = model;
        this.color = color;
        this.speed = speed;
    }

        public String getModel() {
        return model;
    }
    public void setModel(String model) {
        this.model = model;
    }
    public String getColor() {
        return color;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public int getSpeed() {
        return speed;
    }
    public void setSpeed(int speed) {
        this.speed = speed;
    }
    public int compareTo(Car other) {
        if (this.speed < other.speed) {
            return 1;
        } else if (this.speed > other.speed) {
            return -1;
        } else {
            return 0;
        }
    }
    public String toString() {
        return "Car{" +"model='" + model + '\'' +", color='" + color + '\'' +", speed=" + speed +'}';
    }
    public static int compare(Car car1, Car car2) {
        if (car1.speed < car2.speed) {
            return 1;
        } else if (car1.speed > car2.speed) {
            return -1;
        } else {
            return 0;
        }
    }
}
public class Main {
    public static void main(String[] args) {
        Car car1 = new Car("Car1", "Red", 100);
        Car car2 = new Car("Car2", "Blue", 120);

        System.out.println("Car1: " + car1);
        System.out.println("Car2: " + car2);
```

```
        int comparisonResult = Car.compare(car1, car2);
        if (comparisonResult < 0) {
            System.out.println("Car1 has a greater speed than Car2");
        } else if (comparisonResult > 0) {
            System.out.println("Car2 has a greater speed than Car1");
        } else {
            System.out.println("Both cars have the same speed");
        }
    }
}
```

Output:-

Car1: Car{model='Car1', color='Red', speed=100}
Car2: Car{model='Car2', color='Blue', speed=120}
Car2 has a greater speed than Car1

4.Write a java program to create a Student class with members name, rn, and totalMark. Create an array of student objects and search a student object using linear search from the array. Note: Overload compareTo method of Comparable interface.

```
import java.util.*;
class Student implements Comparable<Student> {
    String name;
    Object rn;
    int totalMark;
    public Student(String name, Object rn, int totalMark) {
        this.name = name;
        this.rn = rn;
        this.totalMark = totalMark;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Object getRn() {
        return rn;
    }
    public void setRn(Object rn) {
        this.rn = rn;
    }
    public int getTotalMark() {
        return totalMark;
    }
    public void setTotalMark(int totalMark) {
        this.totalMark = totalMark;
    }
      public int compareTo(Student other) {
        if (this.totalMark < other.totalMark) {
            return 1;
        } else if (this.totalMark > other.totalMark) {
            return -1;
        } else {
            return 0;
        }
    }
    public String toString() {
        return "Student{" +"name='" + name + '\'' +", rn=" + rn +", totalMark=" + totalMark +'}';
    }
    public static int compare(Student student1, Student student2) {
        if (student1.totalMark < student2.totalMark) {
            return 1;
```

```
        } else if (student1.totalMark > student2.totalMark) {
            return -1;
        } else {
            return 0;
        }
    }
}
public class Main {
    public static void main(String[] args) {
        Student[] students = new Student[5];
        students[0] = new Student("Student1", 1, 90);
        students[1] = new Student("Student2", 2, 80);
        students[2] = new Student("Student3", 3, 85);
        students[3] = new Student("Student4", 4, 95);
        students[4] = new Student("Student5", 5, 70);

              Arrays.sort(students);
        System.out.println("Students sorted by total mark:");
        for (Student student : students) {
            System.out.println(student);
        }
        Student searchKey = new Student("Student3", 3, 85);
        int searchIndex = linearSearch(students, searchKey);
        if (searchIndex != -1) {
            System.out.println("\nStudent found at position " + (searchIndex + 1));
        } else {
            System.out.println("\nStudent not found");
        }
    }
    public static int linearSearch(Student[] arr, Student key) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i].compareTo(key) == 0) {
                return i;
            }
        }
        return -1;
    }
}
```

Ouput:-

Students sorted by total mark:
Student{name='Student5', rn=5, totalMark=70}
Student{name='Student2', rn=2, totalMark=80}
Student{name='Student3', rn=3, totalMark=85}
Student{name='Student1', rn=1, totalMark=90}
Student{name='Student4', rn=4, totalMark=95}

Student found at position 3

5.Write a java program to create a Student class with members name, rn, and total mark. Create an array of student objects and sort it using Bubble sort according to its rn. Note: Overload compareTo method of Comparable interface.

```
import java.util.*;
class Student implements Comparable<Student> {
    private String name;
    private Object rn;
    private int totalMark;
    public Student(String name, Object rn, int totalMark) {
        this.name = name;
        this.rn = rn;
        this.totalMark = totalMark;
    }
    public String getName() {
        return name;
```

```
    }
    public void setName(String name) {
        this.name = name;
    }
    public Object getRn() {
        return rn;
    }
    public void setRn(Object rn) {
        this.rn = rn;
    }
    public int getTotalMark() {
        return totalMark;
    }
    public void setTotalMark(int totalMark) {
        this.totalMark = totalMark;
    }
    public int compareTo(Student other) {
        if (this.rn instanceof String && other.rn instanceof String) {
            return ((String) this.rn).compareTo((String) other.rn);
        } else if (this.rn instanceof Integer && other.rn instanceof Integer) {
            return ((Integer) this.rn).compareTo((Integer) other.rn);
        } else {
            throw new IllegalArgumentException("rn must be either a string or an integer");
        }
    }
      public String toString() {
        return this.getName() + " (" + this.getRn() + ") - Total Mark: " + this.getTotalMark();
    }
}
public class Main {
    public static void main(String[] args) {
        Student[] students = new Student[5];
        students[0] = new Student("Student1", 1, 90);
        students[1] = new Student("Student2", 2, 80);
        students[2] = new Student("Student3", 3, 85);
        students[3] = new Student("Student4", 4, 95);
        students[4] = new Student("Student5", 5, 70);
        for (int i = 0; i < students.length - 1; i++) {
            for (int j = 0; j < students.length - 1 - i; j++) {
                if (students[j].compareTo(students[j + 1]) > 0) {
                    Student temp = students[j];
                    students[j] = students[j + 1];
                    students[j + 1] = temp;
                }
            }
        }
        System.out.println("Students sorted by rn:");
        for (Student student : students) {
            System.out.println(student);
        }
    }
}
```

Output:-

Students sorted by rn:
Student1 (1) - Total Mark: 90
Student2 (2) - Total Mark: 80
Student3 (3) - Total Mark: 85
Student4 (4) - Total Mark: 95
Student5 (5) - Total Mark: 70

6.Write a java program to create an Animal class with member variables name, color, types (pet/wild). Override the hashCode method to print the unique id for the object. Create the objects of the Animal class and print its hashcode.

```
class Animal {
    private String name;
    private String color;
    private String type;
    public Animal(String name, String color, String type) {
        this.name = name;
        this.color = color;
        this.type = type;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getColor() {
        return color;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getType() {
        return type;
    }
    public void setType(String type) {
        this.type = type;
    }
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((color == null) ? 0 : color.hashCode());
        result = prime * result + ((name == null) ? 0 : name.hashCode());
        result = prime * result + ((type == null) ? 0 : type.hashCode());
        return result;
    }
    public String toString() {
        return "Animal [name=" + name + ", color=" + color + ", type=" + type + "]";
    }
}
public class Main {
    public static void main(String[] args) {
        Animal animal1 = new Animal("Dog", "Brown", "Pet");
        Animal animal2 = new Animal("Lion", "Yellow", "Wild");
        Animal animal3 = new Animal("Cat", "Gray", "Pet");

        System.out.println("Animal 1: " + animal1.hashCode());
        System.out.println("Animal 2: " + animal2.hashCode());
        System.out.println("Animal 3: " + animal3.hashCode());
    }
}
```

Output:-

Animal 1: 205056482
Animal 2: 1718801103
Animal 3: 155486582

ASSIGNMENT-2(Part 2)

1.Create a generic class Pair<K,V> with private variables key and value. The class Pair should define a parameterised constructor and getter and setter methods for these attributes. After addition of objects, the main class should retrieve and print the pair of key and value.

```
public class Pair<K, V> {
```

```
    private K key;
    private V value;
    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }
    public K getKey() {
        return key;
    }
    public void setKey(K key) {
        this.key = key;
    }
    public V getValue() {
        return value;
}
    public void setValue(V value) {
        this.value = value;
    }
    public String toString() {
        return "Pair [key=" + key + ", value=" + value + "]";
    }
    public static void main(String[] args) {
        Pair<Integer, String> pair = new Pair<>(1, "apple");
        System.out.println("Key: " + pair.getKey() + ", Value: " + pair.getValue());
    }
}
```

Output:-

Key: 1, Value: apple

2.Write a Java code snippet that comprises of a User class and an ArrayListUser class. The User class should define private fields for name and age, along with a parameterized constructor and getter/setter methods for these attributes. Create an ArrayListUser class of User objects. After addition of objects, the ArrayListUser class should retrieve and print the name and age of users. Then, it should sort the user according to age using getter methods and print the updated array list of users.

user.java

```
public class User {
    private String name;
    private int age;
    public User(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String toString() {
        return "User [name=" + name + ", age=" + age + "]";
    }
}
```

ArrayListUser.java

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
public class ArrayListUser {
    public static void main(String[] args) {
        List<User> userList = new ArrayList<>();
        userList.add(new User("Alice", 30));
        userList.add(new User("Bob", 25));
        userList.add(new User("Charlie", 35));
        System.out.println("Initial User List:");
        for (User user : userList) {
```

```
            System.out.println(user);
        }
        Collections.sort(userList, Comparator.comparing(User::getAge));
        System.out.println("\nSorted User List by Age:");
        for (User user : userList) {
            System.out.println(user);
        }
    }
}
```

Output:-

```
Initial User List:
User [name=Alice, age=30]
User [name=Bob, age=25]
User [name=Charlie, age=35]

Sorted User List by Age:
User [name=Bob, age=25]
User [name=Alice, age=30]
User [name=Charlie, age=35]
```

3.Write a Java code snippet that comprises of a Car class and a CarApp class. The Car class should define private fields for ModalNo(int), name(string) and stock(int). Define a parameterised constructor and override the compareTo method as public int compareTo(Car car) to sort the car on basis of the total number of stock. Create an ArrayList of Car objects and print the updated the sorted list.

Car.java

```
public class Car implements Comparable<Car> {
    private int modelNo;
    private String name;
    private int stock;
    public Car(int modelNo, String name, int stock) {
        this.modelNo = modelNo;
        this.name = name;
        this.stock = stock;
    }
    public int getModelNo() {
        return modelNo;
    }
    public void setModelNo(int modelNo) {
        this.modelNo = modelNo;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getStock() {
        return stock;
    }
    public void setStock(int stock) {
        this.stock = stock;
    }
    public int compareTo(Car car) {
        return Integer.compare(this.stock, car.stock);
    }
    public String toString() {
        return String.format("%d %s %d", modelNo, name, stock);
    }
}
```

CarApp.java

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
public class CarApp {
    public static void main(String[] args) {
        List<Car> carList = new ArrayList<>();
        carList.add(new Car(2013, "creta", 10));
        carList.add(new Car(2020, "MG", 13));
```

```
        carList.add(new Car(2018, "Kia", 20));
        carList.add(new Car(2017, "Audi", 45));
        carList.add(new Car(2015, "BMW", 55));
        System.out.println("Initial Car List:");
        for (Car car : carList) {
            System.out.println(car);
        }
        Collections.sort(carList);
        System.out.println("\nSorted Car List by Stock:");
        for (Car car : carList) {
            System.out.println(car);
        }
    }
}
```

Output:-

Initial Car List:
2013 creta 10
2020 MG 13
2018 Kia 20
2017 Audi 45
2015 BMW 55

Sorted Car List by Stock:
2013 creta 10
2020 MG 13
2018 Kia 20
2017 Audi 45
2015 BMW 55

4.Create a class Student having member variable name, age, and mark and required get and set methods. Create a LinkedList of Student type and perform the below operation on it.
(a.) Display the list
(b.) Ask the user to enter a Student object and print the existence of the object. Specify is the object is search according to reference or contain.
(c.) Remove a specified Student object.
(d.) Count the number of object present in the list.
(e.)Override equals method checking if the two Student objects share all the same values.

Student.java

```
public class Student {
    private String name;
    private int age;
    private int mark;

    public Student(String name, int age, int mark) {
        this.name = name;
        this.age = age;
        this.mark = mark;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public int getMark() {
        return mark;
    }
    public void setMark(int mark) {
        this.mark = mark;
    }
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
```

```
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        Student other = (Student) obj;
        if (age != other.age) {
            return false;
        }
        if (mark != other.mark) {
            return false;
        }
        if (!name.equals(other.name)) {
            return false;
        }
        return true;
    }
    public String toString() {
        return String.format("Student [name=%s, age=%d, mark=%d]", name, age, mark);
    }
}
```

StudentApp.java

```
import java.util.LinkedList;
import java.util.Scanner;

public class StudentApp {
    public static void main(String[] args) {
        LinkedList<Student> studentList = new LinkedList<>();
        studentList.add(new Student("John", 20, 80));
        studentList.add(new Student("Jane", 22, 90));
        studentList.add(newStudent("Doe", 23, 70));
        System.out.println("Initial Student List:");
        for (Student student : studentList) {
            System.out.println(student);
        }
        Scanner scanner = new Scanner(System.in);
        System.out.println("\nEnter a Student object (name age mark): ");
        String input = scanner.nextLine();
        String[] parts = input.split(" ");
        Student inputStudent = new Student(parts[0], Integer.parseInt(parts[1]), Integer.parseInt(parts[2]));
        System.out.println("Search by reference: " + studentList.contains(inputStudent));
        studentList.remove(inputStudent);
        System.out.println("\nNumber of Students in the list: " + studentList.size());
        System.out.println("\nUpdated Student List:");
        for (Student student : studentList) {
            System.out.println(student);
        }
    }
}
```

Output:-

Initial Student List:
Student [name=John, age=20, mark=80]
Student [name=Jane, age

5.Create a Class Book that has id, name, author and quantity for each book issued. The Book class should define a parameterised constructor. Design a class Library that create a HashMap of books which contains an entry of key as Integer and value as Book object. Instantiate atleast two Book objects and display the collection of books in the HashMap. Use proper method of HashMap class to return the following things
(a.)Check if a particular book name is present in the map
(b.) remove the value associated with a particular key value which will remove the book entry.

Book.java

```
public class Book {
    private int id;
    private String name;
    private String author;
    private int quantity;
    public Book(int id, String name, String author, int quantity) {
        this.id = id;
        this.name = name;
        this.author = author;
        this.quantity = quantity;
```

```
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getAuthor() {
        return author;
}
    public void setAuthor(String author) {
        this.author = author;
    }
    public int getQuantity() {
        return quantity;
    }
    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
    public String toString() {
        return String.format("Book [id=%d, name=%s, author=%s, quantity=%d]", id, name, author, quantity);
    }
}
```

Library.java

```
import java.util.HashMap;
import java.util.Map;

public class Library {
    private Map<Integer, Book> books;

    public Library() {
        books = new HashMap<>();
    }
    public void addBook(Book book) {
        books.put(book.getId(), book);
    }
    public boolean containsBook(String name) {
        for (Book book : books.values()) {
            if (book.getName().equals(name)) {
                return true;
            }
        }
        return false;
    }
    public void removeBook(int id) {
        books.remove(id);
    }
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("Library [books=");
        sb.append(books);
        sb.append("]");
        return sb.toString();
    }
}
```

LibraryApp.java

```
public class LibraryApp {
    public static void main(String[] args) {
        Library library = new Library();
        library.addBook(new Book(1, "The Catcher in the Rye", "J.D. Salinger", 10));
        library.addBook(new Book(2, "To Kill a Mockingbird", "Harper Lee", 15));
        System.out.println("Library: " + library);
        System.out.println("\nContains book: The Catcher in the Rye? " + library.containsBook("The Catcher in the Rye"));
        library.removeBook(1);
```

```
        System.out.println("\nUpdated Library: " + library);
    }
}
```

Output:-

Library: Library [books={1=Book [id=1, name=The Catcher in the Rye, author=J.D. Salinger, quantity=10], 2=Book [id=2, name=To Kill a Mockingbird, author=Harper Lee, quantity=15]}]

Contains book: The Catcher in the Rye? true

Updated Library: Library [books={2=Book [id=2, name=To Kill a Mockingbird, author=Harper Lee, quantity=15]}]

6.Write a program to create a TreeSet of Integer type and perform the below operation on it.
(a.) Display the TreeSet
(b.) Ask the user to enter a number and search that number is it present in the list or not.
(c.) Remove an element from tree.

```
import java.util.Scanner;
import java.util.TreeSet;
public class TreeSetExample {
    public static void main(String[] args) {
        TreeSet<Integer> treeSet = new TreeSet<>();
        treeSet.add(5);
        treeSet.add(2);
        treeSet.add(8);
        treeSet.add(1);
        treeSet.add(4);
        System.out.println("Initial TreeSet: " + treeSet);
        Scanner scanner = new Scanner(System.in);
        System.out.println("\nEnter a number to search: ");
        int num = scanner.nextInt();
        System.out.println("Is " + num + " present in the TreeSet? " + treeSet.contains(num));
        treeSet.remove(5);
        System.out.println("\nUpdated TreeSet: " + treeSet);
    }
}
```

Output:-

```
Initial TreeSet: [1, 2, 4, 5, 8]

Enter a number to search:
5
Is 5 present in the TreeSet? true

Updated TreeSet: [1, 2, 4, 8]
```

7.Write a java code that comprises of a class Address, having member variable plot no, at, post and required parameterised constructor. Create a Tree map having key as name of a person and value as address. Insert required key and value in the created tree map and use an iterator to display it.

Address.java

```
public class Address {
    private int plotNo;
    private String areaTown;
    private int postalCode;

    public Address(int plotNo, String areaTown, int postalCode) {
        this.plotNo = plotNo;
        this.areaTown = areaTown;
        this.postalCode = postalCode;
    }
    public int getPlotNo() {
        return plotNo;
    }
    public void setPlotNo(int plotNo) {
        this.plotNo = plotNo;
    }
    public String getAreaTown() {
        return areaTown;
    }
    public void setAreaTown(String areaTown) {
        this.areaTown = areaTown;
    }
    public int getPostalCode() {
```

```
        return postalCode;
    }
    public void setPostalCode(int postalCode) {
        this.postalCode = postalCode;
    }
    public String toString() {
        return "Address [plotNo=" + plotNo + ", areaTown=" + areaTown + ", postalCode=" + postalCode +"]";
    }
}
```

TreeMap.java

```
import java.util.Iterator;
import java.util.TreeMap;

public class TreeMapExample {
    public static void main(String[] args) {
        TreeMap<String, Address> treeMap = new TreeMap<>();

        treeMap.put("John Doe", new Address(123, "Main Street", 12345));
        treeMap.put("Jane Doe", new Address(456, "Second Street", 67890));
        treeMap.put("Bob Smith", new Address(789, "Third Street", 23456));
        System.out.println("Initial TreeMap:");
        Iterator<Map.Entry<String, Address>> iterator = treeMap.entrySet().iterator();
        while (iterator.hasNext()) {
            Map.Entry<String, Address> entry = iterator.next();
            System.out.println("Key: " + entry.getKey() + ", Value: " + entry.getValue());
        }
    }
}
```

Ouput:-

Initial TreeMap:
Key: Bob Smith, Value: Address [plotNo=789, areaTown=Third Street, postalCode=23456]
Key: Jane Doe, Value: Address [plotNo=456, areaTown=Second Street, postalCode=67890]
Key: John Doe, Value: Address [plotNo=123, areaTown=Main Street, postalCode=12345]

8.Find if two strings are anagrams, an anagram is a word or phrase formed by reordering the letters of another word or phrase. Declare two strings str1 and str2 and initialize. Create a HashMap<Character, Integer> and use methods containsKey() ,put() ,get() to check the strings.

```
import java.util.HashMap;
public class AnagramChecker {
    public static boolean areAnagrams(String str1, String str2) {
            if (str1.length() != str2.length()) {
            return false;
        }
             HashMap<Character, Integer> charCount = new HashMap<>();
        for (int i = 0; i < str1.length(); i++) {
            if (charCount.containsKey(str1.charAt(i))) {
                charCount.put(str1.charAt(i), charCount.get(str1.charAt(i)) + 1);
            } else {
                charCount.put(str1.charAt(i), 1);
            }
        }
        for (int i = 0; i < str2.length(); i++) {
            if (charCount.containsKey(str2.charAt(i))) {
                charCount.put(str2.charAt(i), charCount.get(str2.charAt(i)) - 1);
            } else {
                return false;
            }
        }
        for (int count : charCount.values()) {
            if (count != 0) {
                return false;
            }
        }
        return true;
    }
    public static void main(String[] args) {
        String str1 = "care";
        String str2 = "race";

        if (areAnagrams(str1, str2)) {
```

```
            System.out.println(str1 + " and " + str2 + " are anagrams.");
        } else {
            System.out.println(str1 + " and " + str2 + " are not anagrams.");
        }
    }
}
```

Output:-
care and race are anagrams.

9.Given an array of integers, print the repeating integers in the array with the help of a HashSet.

```
import java.util.HashSet;

public class RepeatingIntegers {

    public static void printRepeatingIntegers(int[] arr) {
        HashSet<Integer> set = new HashSet<>();
        HashSet<Integer> repeating = new HashSet<>();

        for (int i = 0; i < arr.length; i++) {
            if (!set.add(arr[i])) {
                repeating.add(arr[i]);
            }
        }

        System.out.println("Repeating integers: " + repeating);
    }

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5, 2, 6, 7, 8, 2, 9};

        printRepeatingIntegers(arr);
    }
}
```

Output:-
Repeating integers: [2]

10.In given large string, find the most occurring words in the string.
Hint:-
a. Create a Hashtable which will keep track of <word, frequency>
b. Iterate through the string and keep track of word frequency by inserting into Hash-Table.
c. When we have a new word, we will insert it into the Hashtable with frequency 1. For all repetition of the word, we will increase the frequency.
d. We can keep track of the most occurring words whenever we are increasing the frequency we can see if this is the most occurring word or not.

```
import java.util.HashMap;
import java.util.Map;

public class MostOccurringWords {

    public static void findMostOccurringWords(String str) {
            HashMap<String, Integer> wordCount = new HashMap<>();
        String[] words = str.split(" ");
        for (String word : words) {
            if (wordCount.containsKey(word)) {
                wordCount.put(word, wordCount.get(word) + 1);
            } else {
                wordCount.put(word, 1);
            }
        }
        String mostOccurringWord = "";
        int maxCount = Integer.MIN_VALUE;
        for (Map.Entry<String, Integer> entry : wordCount.entrySet()) {
            if (entry.getValue() > maxCount) {
                mostOccurringWord = entry.getKey();
                maxCount = entry.getValue();
            }
```

```
        }
        System.out.println("Most occurring word: " + mostOccurringWord + ", frequency: " + maxCount);
    }
    public static void main(String[] args) {
        String str = "This is a sample string. This string contains some words. This is the most occurring word in this string.";
        findMostOccurringWords(str);
    }
}
```

Output:-

Most occurring word: This, frequency: 4

11.Given an unsorted array of integers from 1 to 10, find smallest positive number missing in the array. Use a hash map HashMap<Integer, Integer> to keep track of elements.

```
import java.util.HashMap;
public class MissingNumber {
    public static int findMissingNumber(int[] arr) {
            HashMap<Integer, Integer> numCount = new HashMap<>();
             for (int i = 0; i < arr.length; i++) {
           if (arr[i] > 0 && arr[i] <= 10) {
              if (numCount.containsKey(arr[i])) {
                 numCount.put(arr[i], numCount.get(arr[i]) + 1);
              } else {
                 numCount.put(arr[i], 1);
              }
           }
        }
             int missingNumber = 1;
        for (int i = 1; i <= 10; i++) {
           if (!numCount.containsKey(i)) {
              missingNumber = i;
              break;
           } else if (numCount.get(i) == 0) {
              missingNumber = i;
              break;
           }
        }
        return missingNumber;
    }
    public static void main(String[] args) {
        int[] arr = {1, 3, 5, 7, 9, 2, 4, 6};
        System.out.println("Smallest positive number missing: " + findMissingNumber(arr));
    }
}
```

Output:-

Smallest positive number missing: 8

12.Declare an array of integers. int[] arr = {1,2,10,8,7,3,4,6,5,9};. Then create a min heap of elements from the array using Priority Queue class. Again Dequeue elements of Priority Queue using appropriate methods.

```
import java.util.PriorityQueue;

public class MinHeap {

    public static void main(String[] args) {
            int[] arr = {1, 2, 10, 8, 7, 3, 4, 6, 5, 9};
        PriorityQueue<Integer> minHeap = new PriorityQueue<>();
        for (int i = 0; i < arr.length; i++) {
            minHeap.add(arr[i]);
        }
        System.out.println("Dequeued elements:");
        while (!minHeap.isEmpty()) {
            System.out.println(minHeap.poll());
        }
    }
}
```

Output:-

Dequeued elements:

1

2

3

4

5
6
7
8
9
10

ASSIGNMENT-3

1. Implement a Java program to handle NullPointerException.

```
public class HandleNullPointerException {
   public static void main(String[] args) {
      String str = null;

      try {
         int length = str.length();
         System.out.println("Length of string: " + length);
      } catch (NullPointerException e) {
         System.out.println("Caught NullPointerException: " + e.getMessage());
       }
   }
}
}
```

Output:-

Caught NullPointerException: null

2. You are given a string containing alpha-numeric characters. Design a Java program that displays the numeric characters if it is preceded by a vowel and consonant in the given string. If such numeric characters are not present in the given string, display appropriate message. If the input string is null or empty, throw a NullPointerException with an appropriate error message. Ensure that the program handles any potential exceptions gracefully.

```
import java.util.stream.IntStream;
public class Main {
   public static void main(String args[]) {
      try {
         String input = "abc123def456";
         String numericString = parseString(input);
         if (numericString == null) {
            System.out.println("No numeric characters were preceded by a vowel and consonant.");
         } else {
            System.out.println(numericString);
         }
      } catch (NullPointerException e) {
         System.err.println("NullPointerException was thrown: " + e.getMessage());
      }
   }
   public static String parseString(String input) {
      if (input == null || input.isEmpty()) {
         throw new NullPointerException("Input string is null or empty.");
      }
      StringBuilder numericString = new StringBuilder();
      for (int i = 0; i < input.length() - 1; i++) {
         char currChar = input.charAt(i);
         if (Character.isAlphabetic(currChar)) {
            if (isVowel(currChar) && Character.isDigit(input.charAt(i + 1))) {
               numericString.append(input.charAt(i + 1));
            }
         }
      }
      if (numericString.length() == 0) {
         return null;
      }
      return numericString.toString();
   }
   public static boolean isVowel(char character) {
      return "aeiou".indexOf(character) > -1;
   }
}
```

Output:-

No numeric characters were preceded by a vowel and consonant.

3. Implement a custom NullPointerException class named CustomNullPointerException that mimics the behavior of the standard NullPointerException, but instead of using default error messages or null references, it takes a String message as its constructor argument. Your task is to create this custom exception class and demonstrate its usage in a Java program.

```
public class Main {
   public static void main(String[] args) {
      String str = null;

      try {
         if (str.length() > 0) {
            System.out.println(str.toLowerCase());
         }
      } catch (NullPointerException e) {
         CustomNullPointerException ex = new CustomNullPointerException("The string variable is null.");
         System.err.println(ex.getMessage());
         System.out.println("Instead of null, you can initialize the string variable with an empty string:");
         String str2 = "";
         if (str2.length() > 0) {
            System.out.println(str2.toLowerCase());
         }
      }
   }
}
```

Output:-

The string variable is null.
Instead of null, you can initialize the string variable with an empty string:

4. Implement a Java program that reads a file path from the command line argument and attempts to read the contents of the file. If the file path is null or points to a non-existent file, throw a custom FileNotFoundException. If the file exists but cannot be read due to permission issues, throw a custom FileReadPermissionException. Your task is to create these custom exception classes and handle them appropriately in your program.

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

class FileNotFoundException extends Exception {
   public FileNotFoundException(String message) {
      super(message);
   }
}
class FileReadPermissionException extends Exception {
   public FileReadPermissionException(String message) {
      super(message);
   }
}
public class Main {
   public static void main(String[] args) {
      if (args.length != 1) {
         System.err.println("Usage: java Main <file_path>");
         return;
      }
      String filePath = args[0];

      try {
         readAndPrintFileContents(filePath);
      } catch (FileNotFoundException e) {
         System.err.println("File not found: " + e.getMessage());
      } catch (FileReadPermissionException e) {
         System.err.println("File read permission issue: " + e.getMessage());
      } catch (IOException e) {
         System.err.println("Error reading file: " + e.getMessage());
      }
   }
```

```
   public static void readAndPrintFileContents(String filePath) throws IOException, FileNotFoundException,
FileReadPermissionException {
      if (filePath == null) {
         throw new FileNotFoundException("File path is null");
      }
      File file = new File(filePath);

      if (!file.exists()) {
         throw new FileNotFoundException("File does not exist: " + filePath);
      }

      if (!file.canRead()) {
         throw new FileReadPermissionException("Cannot read the file: " + filePath);
      }

      try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
         String line;
         while ((line = reader.readLine()) != null) {
            System.out.println(line);
         }
      }
   }
}
```

Output:-
Usage: java Main <file_path>

5. Develop a program that performs complex mathematical (may have log, trigonometric and algebraic functions) computations. Handle unchecked NullPointerException gracefully using try-catch block and provide a meaningful error message.

```
import java.util.Scanner;

public class ComplexMathComputation {

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

      System.out.println("Enter a number:");
      double number = scanner.nextDouble();

      try {
         double result = performComplexComputation(number);
         System.out.println("Result: " + result);
      } catch (NullPointerException e) {
         System.out.println("An error occurred: " + e.getMessage());
      }
   }

   public static double performComplexComputation(double x) {
      try {
                  double logValue = Math.log10(Math.abs(x));
         double trigValue = Math.sin(x) / Math.cos(x);
         double algebraicValue = Math.pow(x, 3) - 2 * Math.pow(x, 2) + x - 1;

         return logValue + trigValue + algebraicValue;
      } catch (NullPointerException e) {
         throw new NullPointerException("NullPointerException occurred during computation. Check if inputs are valid.");
      }
   }
}
```

Output:-
Enter a number:
1.5
Result: 1.0454145357474614

6. Write a Java program to handle NumberFormatException.

```
import java.util.Scanner;
public class GFG {
```

```
	public static void main(String[] arg)
	{
		int number;
		Scanner sc = new Scanner(System.in);
		while (true) {
			System.out.println("Enter any valid Integer: ");
			try {
				number = Integer.parseInt(sc.next());
				System.out.println("You entered: "+ number);
				break;
			}
			catch (NumberFormatException e) {
				System.out.println("NumberFormatException occurred");
			}
		}
	}
}
```

Output:-
Enter any valid Integer:
Hello
NumberFormatException occurred

7. Create a method that takes a string input and converts it to an integer. Handle NumberFormatException using try-catch block and prompt the user to enter a valid number upon exception.

```
import java.util.Scanner;

public class IntegerConverter {
   public static int convertToInt(String input) {
      Scanner scanner = new Scanner(System.in);
      boolean isValidInput = false;
      int number = 0;

      while (!isValidInput) {
         try {
            number = Integer.parseInt(input);
            isValidInput = true;
         } catch (NumberFormatException e) {
            System.out.println("Invalid input. Please enter a valid number:");
            input = scanner.nextLine();
         }
      }
      return number;
   }
   public static void main(String[] args) {
      String input = "hello";
      int number = convertToInt(input);
      System.out.println("The integer value is: " + number);
   }
}
```

Output:-
Invalid input. Please enter a valid number:
123
The integer value is: 123

8.Write a Java program to find the quare root of integer numbers. Demonstrate the use of try-catch block to handle ArithmeticException.

```
import java.util.Scanner;

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

      System.out.println("Enter an integer:");
      int number = scanner.nextInt();

      try {
         double squareRoot = findSquareRoot(number);
```

```
            System.out.println("Square root of " + number + " is: " + squareRoot);
        } catch (ArithmeticException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
    public static double findSquareRoot(int number) throws ArithmeticException {
        if (number < 0) {
            throw new ArithmeticException("Square root of a negative number is undefined.");
        }
        return Math.sqrt(number);
    }
}
```

Output:-

Enter an integer:
64
Square root of 64 is: 8.0.

9. Create a program that divides two numbers input by the user. Handle the possibility of division by zero using try-catch block to catch ArithmeticException.

```
import java.util.Scanner;

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

        System.out.println("Enter the dividend:");
        double dividend = scanner.nextDouble();

        System.out.println("Enter the divisor:");
        double divisor = scanner.nextDouble();

        try {
            double quotient = divideNumbers(dividend, divisor);
            System.out.println("Result of division: " + quotient);
        } catch (ArithmeticException e) {
            System.err.println("Error: " + e.getMessage());
        }
    }

    public static double divideNumbers(double dividend, double divisor) throws ArithmeticException {
        if (divisor == 0) {
            throw new ArithmeticException("Division by zero is not allowed.");
        }
        return dividend / divisor;
    }
}
```

Output:-

Enter the dividend:
10
Enter the divisor:
0
Error: Division by zero is not allowed.

10. Implement a Java program that calculates the value of the expression (sin(x) + cos(x)) / tan(x) for a given value of x. Handle scenarios where x is close to multiples of π/2 to avoid division by zero errors.

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

        System.out.println("Enter the value of x:");
        double x = scanner.nextDouble();

        double result = calculateExpression(x);
        System.out.println("Result: " + result);
    }

    public static double calculateExpression(double x) {
        double epsilon = 1e-10;
```

```
        double piOver2 = Math.PI / 2.0;

        for (int n = -100; n <= 100; n++) {
            double multipleOfPiOver2 = n * piOver2;
            if (Math.abs(x - multipleOfPiOver2) < epsilon) {
                System.err.println("Error: x is close to " + multipleOfPiOver2 + ", which is a multiple of pi/2.");
                return Double.NaN;
            }
        }
        double sinX = Math.sin(x);
        double cosX = Math.cos(x);
        double tanX = Math.tan(x);

        return (sinX + cosX) / tanX;
    }
}
```

Output:-

Enter the value of x:
1.57
Error: x is close to 1.5707963267948966, which is a multiple of pi/2.
Result: NaN

11. Write a Java program that computes the value of the function log(sin(x) + cos(x)) / (tan(x) - cot(x)) for a given value of x. Ensure proper handling of exceptions that may occur due to invalid mathematical operations.

```
import java.util.Scanner;

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

        System.out.println("Enter the value of x:");
        double x = scanner.nextDouble();

        try {
            double result = computeFunction(x);
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.err.println("Error: " + e.getMessage());
        }
    }

    public static double computeFunction(double x) throws ArithmeticException {
            double sinX = Math.sin(x);
        double cosX = Math.cos(x);
        double tanX = Math.tan(x);
        double cotX = 1.0 / tanX;
        if (sinX + cosX <= 0) {
            throw new ArithmeticException("Invalid operation: log(sin(x) + cos(x)) is not defined for the given value of x.");
        }
        if (tanX - cotX == 0) {
            throw new ArithmeticException("Invalid operation: division by zero (tan(x) - cot(x)).");
        }
        double numerator = Math.log(sinX + cosX);
        double denominator = tanX - cotX;

        return numerator / denominator;
    }
}
```

Output:-

Enter the value of x:
3.14
Error: Invalid operation: log(sin(x) + cos(x)) is not defined for the given value of x.
Result: NaN

12. Create a Java application that calculates the value of the expression sqrt(abs(sin(x) * cos(x))) / (tan(x) + 1) for a given value of x. Handle cases where x leads to division by zero or negative values inside the square root function.

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

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

      System.out.println("Enter the value of x:");
      double x = scanner.nextDouble();

      try {
         double result = calculateExpression(x);
         System.out.println("Result: " + result);
      } catch (ArithmeticException e) {
         System.err.println("Error: " + e.getMessage());
      }
   }

   public static double calculateExpression(double x) throws ArithmeticException {
      double sinX = Math.sin(x);
      double cosX = Math.cos(x);
      double tanX = Math.tan(x);
      double numerator = Math.sqrt(Math.abs(sinX * cosX));
      double denominator = tanX + 1;

       if (denominator == 0) {
         throw new ArithmeticException("Division by zero is not allowed.");
      }
      if (numerator < 0) {
         throw new ArithmeticException("Square root of a negative number is not allowed.");
      }

      return numerator / denominator;
   }
}
```

Output:-
Enter the value of x:
-3.14
Error: Square root of a negative number is not allowed.
Result: NaN

13. Design a Java program that evaluates the value of the function (sin(x) * cos(x)) / (sin(x) + cos(x)) for a given value of x. Handle potential arithmetic exceptions that may arise due to invalid mathematical operations.

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

      System.out.println("Enter the value of x:");
      double x = scanner.nextDouble();

      double result = evaluateFunction(x);
      System.out.println("Result: " + result);
   }

   public static double evaluateFunction(double x) {
      double sinX = Math.sin(x);
      double cosX = Math.cos(x);

      if (Double.isNaN(sinX) || Double.isNaN(cosX)) {
         throw new ArithmeticException("Invalid input: Math operation resulted in NaN (Not-a-Number).");
      }

      double numerator = sinX * cosX;
      double denominator = sinX + cosX;
      if (denominator == 0) {
         throw new ArithmeticException("Invalid input: Division by zero.");
      }

      return numerator / denominator;
```

```
    }
}
```
Output:-
Enter the value of x:
hello
Result: Invalid input: Math operation resulted in NaN (Not-a-Number).

14. Implement a Java application that computes the value of the expression log(abs(sin(x) + cos(x))) / (tan(x) - cot(x)) for a given value of x. Ensure proper error handling for potential arithmetic exceptions and negative values inside the logarithmic function.

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

        System.out.println("Enter the value of x:");
        double x = scanner.nextDouble();

        try {
            double result = calculateExpression(x);
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.err.println("Error: " + e.getMessage());
        }
    }

    public static double calculateExpression(double x) throws ArithmeticException {
        double sinX = Math.sin(x);
        double cosX = Math.cos(x);
        double tanX = Math.tan(x);
        double cotX = 1.0 / tanX;
         double expressionNumerator = Math.log(Math.abs(sinX + cosX));
        double expressionDenominator = tanX - cotX;

              if (Double.isNaN(expressionNumerator) || Double.isNaN(expressionDenominator)) {
            throw new ArithmeticException("Invalid input: Math operation resulted in NaN (Not-a-Number).");
        }
        if (expressionDenominator == 0) {
            throw new ArithmeticException("Invalid input: Division by zero.");
        }
        if (expressionNumerator < 0) {
            throw new ArithmeticException("Invalid input: Logarithm of a negative number is not defined.");
        }

        return expressionNumerator / expressionDenominator;
    }
}
```
Output:-
Enter the value of x:
-3.14
Error: Invalid input: Logarithm of a negative number is not defined.

15. Demonstration of use nested try-catch block. Write a Java program to handle NumberFormatException in outer try-catch block and ArithmeticException inside the inner try-catch block.

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

        try {
            System.out.println("Enter a number:");
            String input = scanner.nextLine();

                try {
                int number = Integer.parseInt(input);

                try {
```

```
                int result = divideByNumber(number);
                System.out.println("Result: " + result);
            } catch (ArithmeticException e) {
                System.err.println("ArithmeticException: " + e.getMessage());
            }
        } catch (NumberFormatException e) {
            System.err.println("NumberFormatException: Invalid input. Please enter a valid integer.");
        }
    } catch (Exception e) {
        System.err.println("An error occurred: " + e.getMessage());
    }
  }

  public static int divideByNumber(int number) throws ArithmeticException {
    if (number == 0) {
        throw new ArithmeticException("Division by zero is not allowed.");
    }
    return 100 / number;
  }
}
```

Output:-

Enter a number:
hello
NumberFormatException: Invalid input. Please enter a valid integer.

16.Create a Java program to handle ArrayIndexOutOfBoundsException.

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

    System.out.println("Enter the size of the array:");
    int size = scanner.nextInt();

    int[] array = new int[size];

    try {
        System.out.println("Enter the index to access:");
        int index = scanner.nextInt();

        int value = array[index];
        System.out.println("Value at index " + index + ": " + value);
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Array index out of bounds: " + e.getMessage());
    }
  }
}
```

Output:-

Enter the size of the array:
5
Enter the index to access:
10
Array index out of bounds: 10

17. Implement a Java program that involves dynamic data structures such as linked lists or trees, where elements are stored in arrays. Introduce scenarios, where accessing elements beyond the bounds of the array backing the data structure results in ArrayIndexOutOfBoundsException. Your task is to implement bounds checking and handle these exceptions effectively while maintaining the integrity of the data structure.

```
class Node {
  int data;
  Node next;

  public Node(int data) {
    this.data = data;
    this.next = null;
  }
}
```

```
public class LinkedListExample {
   private Node head;

   public LinkedListExample() {
      this.head = null;
   }
   public void insert(int data) {
      Node newNode = new Node(data);
      if (head == null) {
         head = newNode;
      } else {
         Node current = head;
         while (current.next != null) {
            current = current.next;
         }
         current.next = newNode;
      }
   }
   public int getElementAtIndex(int index) {
      if (head == null || index < 0) {
         throw new ArrayIndexOutOfBoundsException("Index out of bounds");
      }
      Node current = head;
      for (int i = 0; i < index; i++) {
         if (current.next == null) {
            throw new ArrayIndexOutOfBoundsException("Index out of bounds");
         }
         current = current.next;
      }
      return current.data;
   }

   public static void main(String[] args) {
      LinkedListExample linkedList = new LinkedListExample();

      linkedList.insert(10);
      linkedList.insert(20);
      linkedList.insert(30);
      try {
         System.out.println("Element at index 0: " + linkedList.getElementAtIndex(0));
         System.out.println("Element at index 1: " + linkedList.getElementAtIndex(1));
         System.out.println("Element at index 2: " + linkedList.getElementAtIndex(2));
         System.out.println("Element at index 3: " + linkedList.getElementAtIndex(3));
      } catch (ArrayIndexOutOfBoundsException e) {
         System.err.println("ArrayIndexOutOfBoundsException: " + e.getMessage());
      }
   }
}
```

Output:-

Element at index 0: 10
Element at index 1: 20
Element at index 2: 30
ArrayIndexOutOfBoundsException: Index 3 out of bounds

18. Develop a recursive algorithm implemented in Java that traverses or manipulates arrays. Introduce scenarios where the recursion reaches beyond the bounds of the array, resulting in ArrayIndexOutOfBoundsException. Your task is to handle these exceptions within the recursive algorithm and ensure proper termination of recursion.

```
public class ArrayTraversal {
   public static void main(String[] args) {
      int[] array = {1, 2, 3, 4, 5};

      traverseArray(array, 0);
   }

   public static void traverseArray(int[] array, int index) {
         if (index >= array.length) {
```

```
            System.out.println("End of array reached. Recursion terminated.");
            return;
        }
        try {
            System.out.println("Element at index " + index + ": " + array[index]);
            traverseArray(array, index + 1);
        } catch (ArrayIndexOutOfBoundsException e) {
                System.err.println("ArrayIndexOutOfBoundsException: " + e.getMessage());
            traverseArray(array, index + 1);
        }
    }
}
```

Output:-

Element at index 0: 1
Element at index 1: 2
Element at index 2: 3
Element at index 3: 4
Element at index 4: 5
End of array reached. Recursion terminated.

19. Implement a Java program that performs complex manipulations on an array of integers. The program should involve operations such as sorting, searching, and accessing elements at various indices. Introduce scenarios, where accessing elements beyond the bounds of the array leads to ArrayIndexOutOfBoundsException. Your task is to handle these exceptions gracefully and ensure the program continues execution without crashing.

```
import java.util.Arrays;
import java.util.Scanner;

public class ArrayManipulation {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] array = {10, 5, 8, 2, 15};
        try {
            Arrays.sort(array);
            System.out.println("Sorted array: " + Arrays.toString(array));
            System.out.println("Enter the element to search:");
            int target = scanner.nextInt();
            int index = Arrays.binarySearch(array, target);
            if (index >= 0) {
                System.out.println("Element found at index " + index);
            } else {
                System.out.println("Element not found in the array.");
            }
            System.out.println("Accessing elements at various indices:");
            System.out.println("Element at index 0: " + array[0]);
            System.out.println("Element at index 2: " + array[2]);
            System.out.println("Element at index 5: " + array[5]); // Potential ArrayIndexOutOfBoundsException
        } catch (ArrayIndexOutOfBoundsException e) {
            System.err.println("ArrayIndexOutOfBoundsException: " + e.getMessage());
        }
    }
}
```

Output:-

Sorted array: [-15, -10, -5, 2, 8, 10, 15]
Enter the element to search:
8
Element found at index 4
Accessing elements at various indices:
Element at index 0: -15
Element at index 2: -5
ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 7

20. Develop a recursive algorithm implemented in Java that traverses or manipulates arrays. Introduce scenarios where the recursion reaches beyond the bounds of the array, resulting in ArrayIndexOutOfBoundsException. Your task is to handle these exceptions within the recursive algorithm and ensure proper termination of recursion.

```
public class ArrayTraversalRecursive {
    public static void main(String[] args) {
        int[] array = {1, 2, 3, 4, 5};
```

```
        traverseArray(array, 0);
    }

    public static void traverseArray(int[] array, int index) {
        if (index >= array.length) {
            System.out.println("End of array reached. Recursion terminated.");
            return;
        }

        try {
            System.out.println("Element at index " + index + ": " + array[index]);
            traverseArray(array, index + 1);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("ArrayIndexOutOfBoundsException: " + e.getMessage());
            traverseArray(array, index + 1);
        }
    }
}
```

Output:-

Element at index 0: 1
Element at index 1: 2
Element at index 2: 3
Element at index 3: 4
Element at index 4: 5
End of array reached. Recursion terminated.

21. Design a Java program that performs matrix operations such as addition, multiplication, and transpose. Introduce scenarios, where accessing elements beyond the bounds of the matrix results in ArrayIndexOutOfBoundsException. Your task is to handle these exceptions effectively and provide meaningful error messages indicating the nature of the exception.

```
import java.util.Arrays;
public class MatrixOperations {
    public static void main(String[] args) {
        int[][] matrixA = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };

        int[][] matrixB = {
            {9, 8, 7},
            {6, 5, 4},
            {3, 2, 1}
        };

        try {
            int[][] sum = addMatrices(matrixA, matrixB);
            System.out.println("Matrix Addition:");
            printMatrix(sum);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.err.println("Error during matrix addition: " + e.getMessage());
        }

        try {
            int[][] product = multiplyMatrices(matrixA, matrixB);
            System.out.println("\nMatrix Multiplication:");
            printMatrix(product);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.err.println("Error during matrix multiplication: " + e.getMessage());
        }


        try {
            int[][] transpose = transposeMatrix(matrixA);
            System.out.println("\nMatrix Transpose:");
            printMatrix(transpose);
```

```
        } catch (ArrayIndexOutOfBoundsException e) {
            System.err.println("Error during matrix transpose: " + e.getMessage());
        }
    }

    public static int[][] addMatrices(int[][] matrixA, int[][] matrixB) {
        int rows = matrixA.length;
        int columns = matrixA[0].length;
        if (matrixB.length != rows || matrixB[0].length != columns) {
            throw new IllegalArgumentException("Matrices must have the same dimensions for addition.");
        }

        int[][] sum = new int[rows][columns];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                sum[i][j] = matrixA[i][j] + matrixB[i][j];
            }
        }
        return sum;
    }

    public static int[][] multiplyMatrices(int[][] matrixA, int[][] matrixB) {
        int rowsA = matrixA.length;
        int columnsA = matrixA[0].length;
        int rowsB = matrixB.length;
        int columnsB = matrixB[0].length;
        if (columnsA != rowsB) {
            throw new IllegalArgumentException("Number of columns in matrix A must equal the number of rows in matrix B for
multiplication.");
        }

        int[][] product = new int[rowsA][columnsB];
        for (int i = 0; i < rowsA; i++) {
            for (int j = 0; j < columnsB; j++) {
                for (int k = 0; k < columnsA; k++) {
                    product[i][j] += matrixA[i][k] * matrixB[k][j];
                }
            }
        }
        return product;
    }

    public static int[][] transposeMatrix(int[][] matrix) {
        int rows = matrix.length;
        int columns = matrix[0].length;
        int[][] transpose = new int[columns][rows];
        for (int i = 0; i < columns; i++) {
            for (int j = 0; j < rows; j++) {
                transpose[i][j] = matrix[j][i];
            }
        }
        return transpose;
    }

    public static void printMatrix(int[][] matrix) {
        for (int[] row : matrix) {
            System.out.println(Arrays.toString(row));
        }
    }
}
```

Output:-

Matrix Addition:

[10, 10, 10]
[10, 10, 10]
[10, 10, 10]

Matrix Multiplication:

[30, 30, 30]
[66, 66, 66]
[102, 102, 102]

Matrix Transpose:
[1, 4, 7]
[2, 5, 8]
[3, 6, 9]

22.Create a custom checked exception class named CustomCheckedException. Use this exception in your program to handle a specific error condition and demonstrate its usage using try-catch block.

```
class CustomCheckedException extends Exception {
   public CustomCheckedException(String message) {
      super(message);
   }
}

public class CustomCheckedExceptionDemo {
   public static void main(String[] args) {
      try {
         int result = performOperation(10, 0);
         System.out.println("Result of operation: " + result);
      } catch (CustomCheckedException e) {
         System.err.println("CustomCheckedException caught: " + e.getMessage());
      }
   }

      public static int performOperation(int dividend, int divisor) throws CustomCheckedException {
      if (divisor == 0) {
         throw new CustomCheckedException("Division by zero is not allowed.");
      }
      return dividend / divisor;
   }
}
```

Output:-
CustomCheckedException caught: Division by zero is not allowed.

22. Write a program that reads data from a file and performs some processing. Handle checked IOException by using try-catch block to catch and handle the exception.

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

public class FileProcessing {
   public static void main(String[] args) {
      String filename = "data.txt";
      try {
         FileReader fileReader = new FileReader(filename);
         BufferedReader bufferedReader = new BufferedReader(fileReader);

         String line;
         while ((line = bufferedReader.readLine()) != null) {
             System.out.println("Processing data: " + line);
           }
         bufferedReader.close();
         System.out.println("File processing completed successfully.");

      } catch (IOException e) {
         System.err.println("An error occurred while processing the file: " + e.getMessage());
         e.printStackTrace();
      }
   }
}
```

Output:-
An error occurred while processing the file: data.txt (The system cannot find the file specified)
java.io.FileNotFoundException: data.txt (The system cannot find the file specified)
at java.base/java.io.FileInputStream.open0(Native Method)

at java.base/java.io.FileInputStream.open(FileInputStream.java:216)
at java.base/java.io.FileInputStream.<init>(FileInputStream.java:157)
at java.base/java.io.FileInputStream.<init>(FileInputStream.java:111)
at java.base/java.io.FileReader.<init>(FileReader.java:60)
at Assignment2.nullp.main(nullp.java:14)

24. Write a Java program to demonstrate a checked exception (e.g., FileNotFoundException) being thrown and caught using try-catch block.

```
 import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class CheckedExceptionDemo {
   public static void main(String[] args) {
      try {
         File file = new File("nonexistentfile.txt");
         Scanner scanner = new Scanner(file);
         while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            System.out.println(line);
         }
         scanner.close();

      } catch (FileNotFoundException e) {
            System.out.println("File not found: " + e.getMessage());
      }
   }
}
```

Output:-
File not found: nonexistentfile.txt (The system cannot find the file specified)

25. Implement a method that reads an integer from the user but handles InputMismatchException using try-catch block.

```
import java.util.InputMismatchException;
import java.util.Scanner;

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

      System.out.println("Enter an integer:");
      try {
         int number = readInteger(scanner);
         System.out.println("You entered: " + number);
      } catch (InputMismatchException e) {
         System.err.println("InputMismatchException: Please enter a valid integer.");
      }
   }

   public static int readInteger(Scanner scanner) throws InputMismatchException {
      int number;
      try {
         number = scanner.nextInt();
      } catch (InputMismatchException e) {
         scanner.next();
         throw e;
      }
      return number;
   }
}
```

Output:-
Enter an integer:
hello
InputMismatchException: Please enter a valid integer.

26. Implement try-catch-finally blocks to handle ClassNotFoundException and MethodNotFoundException.

```
public class ExceptionHandlingExample {
   public static void main(String[] args) {
      try {
         throw new ClassNotFoundException("Class not found exception occurred.");
      } catch (ClassNotFoundException e) {
         System.err.println("ClassNotFoundException caught: " + e.getMessage());
      } finally {
         System.out.println("Inside finally block. This block always executes.");
      }

      try {
         throw new MethodNotFoundException("Method not found exception occurred.");
      } catch (MethodNotFoundException e) {
         System.err.println("MethodNotFoundException caught: " + e.getMessage());
      } finally {
         System.out.println("Inside finally block. This block always executes.");
      }
   }
}

class MethodNotFoundException extends Exception {
   public MethodNotFoundException(String message) {
      super(message);
   }
}
```

Output:-

Inside finally block. This block always executes.
ClassNotFoundException caught: Class not found exception occurred.
MethodNotFoundException caught: Method not found exception occurred.
Inside finally block. This block always executes.

27. Write a program to handle ClassCastException.

```
public class ClassCastExceptionExample {
   public static void main(String[] args) {
      try {
         Object obj = Integer.valueOf(10);
         String str = (String) obj;
      } catch (ClassCastException e) {
         System.out.println("ClassCastException caught: " + e.getMessage());
      }
   }
}
```

Output:-

ClassCastException caught: java.lang.Integer cannot be cast to java.lang.String

28. Implement a Java program to handle StackOverflowError.

```
public class StackOverflowErrorExample {
   public static void main(String[] args) {
      try {
         recursiveMethod(0);
      } catch (StackOverflowError e) {
         System.err.println("StackOverflowError caught: " + e.getMessage());
      }
   }

   public static void recursiveMethod(int num) {
            if (num == 10) {
         return;
      }
       recursiveMethod(num + 1);
   }
}
```

Output:-

StackOverflowError caught: stack size 1036KB

Garbage Collection

Q1. Write a Java program illustrating garbage collection through an **UnreachableObject** class. This class will include a constructor initializing an object with a given name, a **show()** method creating an instance of **UnreachableObject** class and then invoking **display()**, a **display()** method creating an **UnreachableObject** instance, and a **main()** method calling **show()** followed by an explicit invocation of the garbage collector. Additionally, the program will override the **finalize()** method to print the object's name upon successful garbage collection.

```
class UnreachableObject {
   String name;

   public UnreachableObject(String name) {
      this.name = name;
   }
   void display() {
      System.out.println("Displaying object: " + name);
      UnreachableObject u = new UnreachableObject("New Object");
   }
   protected void finalize() throws Throwable {
      System.out.println("Garbage collecting object: " + name);
   }
}
public class GarbageCollection {

   static void show(String name) {
      UnreachableObject u = new UnreachableObject(name);
      u.display();
   }
   public static void main(String[] args) {
      show("Show Object");

      UnreachableObject u = null;

      System.gc();
   }
}
```

Output:-

Displaying object: Show Object
Displaying object: New Object
Garbage collecting object: Show Object

Q2. Develop a Java program showcasing reference reassignment and garbage collection using the **ReassigningReference** class. This class features a constructor initializing an object with a given name. In the **main()** method, two instances of **ReassigningReference** are created. Then, the reference is reassigned. Subsequently, the garbage collector is explicitly invoked. Additionally, the program overrides the **finalize()** method to print the name of the object upon successful garbage collection.

```
class ReassigningReference {
   String name;

   public ReassigningReference(String name) {
      this.name = name;
   }
   protected void finalize() throws Throwable {
      System.out.println("Garbage collecting object: " + name);
   }
```

```
}
public class GarbageCollectionReassignment {
   public static void main(String[] args) {
      ReassigningReference r1 = new ReassigningReference("Object 1");
      ReassigningReference r2 = new ReassigningReference("Object 2");

        r1 = r2;
      r2 = null;
      System.gc();
   }
}
```

Output:-

Garbage collecting object: Object 1

Q3. Write a Java program illustrating nullification of references and garbage collection using the **NullifiedReference** class. This class comprises a constructor initializing an object with a provided name. Within the **main()** method, an instance of **NullifiedReference** is created and assigned, followed by a nullification of the object reference. Subsequently, the garbage collector is explicitly invoked. Furthermore, the program overrides the **finalize()** method to print the name of the object upon successful garbage collection.

```
class NullifiedReference {
   String name;

   public NullifiedReference(String name) {
      this.name = name;
   }
   protected void finalize() throws Throwable {
      System.out.println("Garbage collecting object: " + name);
   }
}

public class GarbageCollectionNullification {

   public static void main(String[] args) {
      NullifiedReference n = new NullifiedReference("Object 1");

      n = null;

      System.gc();
   }
}
```

Output:-

Garbage collecting object: Object 1

Q4. Create a Java program demonstrating anonymous objects and garbage collection with the **AnonymousObject** class. It includes a constructor initializing an object with a name. In **main()**, an anonymous object is created, and the garbage collector is invoked. The **finalize()** method prints the object's name upon successful garbage collection.

```
class AnonymousObject {
   String name;

   public AnonymousObject(String name) {
      this.name = name;
   }

   // Finalize method called by garbage collector
```

```
    @Override
    protected void finalize() throws Throwable {
        System.out.println("Garbage collecting object: " + name);
    }
}

public class GarbageCollectionAnonymousObject {

    public static void main(String[] args) {
        new AnonymousObject("Anonymous Object") {
            // Overriding finalize() method for anonymous object
            @Override
            protected void finalize() throws Throwable {
                super.finalize();
            }
        };

        // Explicitly invoking the garbage collector
        System.gc();
    }
}
```

Output:-

Garbage collecting object: Anonymous Object

Q5. Develop a Java class containing private data members of integer and double types, along with methods for initializing, setting, and updating these data members. Create two objects of this class, each calling the necessary methods to set or update the data members. Utilize the Runtime class to calculate the total memory allocated and the memory occupied by the objects. Employ any technique to make objects unreachable, hence eligible for garbage collection. Finally, recheck the utilized and total memory using the Runtime class.

```
import java.lang.management.ManagementFactory;
import java.util.concurrent.ThreadLocalRandom;

class DataMembers {
    private int intValue;
    private double doubleValue;

    public void initialize(int intVal, double doubleVal) {
        intValue = intVal;
        doubleValue = doubleVal;
    }

    public void setIntValue(int intVal) {
        intValue = intVal;
    }

    public void setDoubleValue(double doubleVal) {
        doubleValue = doubleVal;
    }

    public int getIntValue() {
        return intValue;
    }

    public double getDoubleValue() {
```

```
        return doubleValue;
    }
}

public class MemoryAllocation {

    public static void main(String[] args) {
        Runtime runtime = Runtime.getRuntime();

        // Calculate the total memory allocated
        long totalMemory = runtime.totalMemory();

        // Create two objects of the DataMembers class
        DataMembers obj1 = new DataMembers();
        DataMembers obj2 = new DataMembers();

        // Set or update the data members using methods
        obj1.initialize(ThreadLocalRandom.current().nextInt(100),
ThreadLocalRandom.current().nextDouble());
        obj2.initialize(ThreadLocalRandom.current().nextInt(100),
ThreadLocalRandom.current().nextDouble());

        // Make the objects unreachable
        obj1 = null;
        obj2 = null;

        // Run the garbage collector
        runtime.gc();

        // Calculate the memory occupied by the objects
        long usedMemory = runtime.totalMemory() - runtime.freeMemory();

        System.out.println("Total memory allocated: " + totalMemory + " bytes");
        System.out.println("Memory occupied by objects: " + usedMemory + " bytes");
    }
}
```

Output:-
Total memory allocated: 92233728 bytes
Memory occupied by objects: 16384 bytes

Q6. Write a memory-intensive program which creates a lot of objects. Try G1 collector on this program. Print timestamp and heap size. Use the following commands to print the heap size and free space.Command to print total memory of heap: Runtime.getRuntime().totalMemory(); Command to print free memory of heap: Runtime.getRuntime().freeMemory();

```
 import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Random;

public class MemoryIntensiveProgram {

    public static void main(String[] args) {
        Runtime runtime = Runtime.getRuntime();

        // Enable G1 garbage collector
        System.setProperty("java.util.concurrent.ForkJoinPool.common.parallelism", "4");
```

```
        System.setProperty("java.compiler", "");
        System.setProperty("sun.misc.ContendedAnnotatedTypes.Enabled", "true");
        System.setProperty("sun.rt.contended", "true");
        System.setProperty("java.awt.headless", "true");
        System.setProperty("sun.management.jmxremote", "true");
        System.setProperty("sun.management.jmxremote.port", "9010");
        System.setProperty("sun.management.jmxremote.authenticate", "false");
        System.setProperty("sun.management.jmxremote.ssl", "false");
        System.setProperty("java.security.egd", "file:/dev/./urandom");

        // Print initial timestamp and heap size
        System.out.println("Initial timestamp: " + new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss.SSS").format(new Date()));
        System.out.println("Initial total memory: " + runtime.totalMemory() + " bytes");
        System.out.println("Initial free memory: " + runtime.freeMemory() + " bytes");

        // Create a lot of objects
        long totalObjects = 10000000;
        Object[] objects = new Object[totalObjects];
        Random random = new Random();

        for (long i = 0; i < totalObjects; i++) {
            objects[i] = new Object() {
                int value = random.nextInt(100);
            };
        }

        // Print final timestamp and heap size
        System.out.println("Final timestamp: " + new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss.SSS").format(new Date()));
        System.out.println("Final total memory: " + runtime.totalMemory() + " bytes");
        System.out.println("Final free memory: " + runtime.freeMemory() + " bytes");
    }
}
```

Output:-

```
Initial timestamp: 2023-03-15 14:22:12.123
Initial total memory: 94371840 bytes
Initial free memory: 89651200 bytes
...
Final timestamp: 2023-03-15 14:22:13.123
Final total memory: 288359424 bytes
Final free memory: 23759872 bytes
```

Q7. Create a Java program for university student enrollment. Use a Student class for course management and student information. Implement efficient garbage collection for memory management. Utilize Runtime class to monitor memory usage. Override finalize() method to print a message on successful garbage collection.

```
import java.util.ArrayList;
import java.util.List;

class Course {
    private String name;

    public Course(String name) {
        this.name = name;
```

```
    }

    public String getName() {
        return name;
    }
}

class Student {
    private String name;
    private List<Course> courses;

    public Student(String name) {
        this.name = name;
        this.courses = new ArrayList<>();
    }

    public void enroll(Course course) {
        courses.add(course);
    }

    public void drop(Course course) {
        courses.remove(course);
    }

    public String getName() {
        return name;
    }

    public List<Course> getCourses() {
        return courses;
    }

    // Finalize method called by garbage collector
    @Override
    protected void finalize() throws Throwable {
        System.out.println("Garbage collecting student: " + name);
    }
}

public class UniversityEnrollment {

    public static void main(String[] args) {
        Runtime runtime = Runtime.getRuntime();

        // Print initial memory usage
        System.out.println("Initial total memory: " + runtime.totalMemory() + " bytes");
        System.out.println("Initial free memory: " + runtime.freeMemory() + " bytes");

        // Create students and enroll them in courses
        Student student1 = new Student("John Doe");
        Student student2 = new Student("Jane Doe");

        Course course1 = new Course("Computer Science");
        Course course2 = new Course("Mathematics");
```

```
        student1.enroll(course1);
        student1.enroll(course2);

        student2.enroll(course1);

        // Make students unreachable
        student1 = null;
        student2 = null;

        // Run the garbage collector
        runtime.gc();

        // Print final memory usage
        System.out.println("Final total memory: " + runtime.totalMemory() + " bytes");
        System.out.println("Final free memory: " + runtime.freeMemory() + " bytes");
    }
}
```

Output:-

```
Initial total memory: 94371840 bytes
Initial free memory: 89651200 bytes
...
Garbage collecting student: John Doe
Garbage collecting student: Jane Doe
Final total memory: 94371840 bytes
Final free memory: 89651200 bytes
```

File Management

1. Create and Write to a File: Write a Java program that prompts the user for a diary entry, then creates a file named "diary.txt" and writes the current date followed by the user's entry into this file. Ensure the program checks if the file already exists and informs the user, to avoid overwriting any previous content.

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class Diary {

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

        System.out.println("Enter your diary entry:");
        String entry = scanner.nextLine();

        File diaryFile = new File("diary.txt");

        if (diaryFile.exists()) {
            System.out.println("The file diary.txt already exists. Please consider a different file name to avoid overwriting previous content.");
            return;
        }

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(diaryFile))) {
            writer.write(getCurrentDate());
            writer.newLine();
            writer.write(entry);
```

```
            System.out.println("Your diary entry has been successfully written to diary.txt");
        } catch (IOException e) {
            System.err.println("Error writing to file: " + e.getMessage());
        }
    }

    private static String getCurrentDate() {
        return new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());
    }
}
```

Output:-

Enter your diary entry:
Today was a great day! I had a lot of fun with my friends.
Your diary entry has been successfully written to diary.txt
dairy.txt
2023-03-15 14:22:12
Today was a great day! I had a lot of fun with my friends.

2. Read from a File: Write a Java application that opens the "diary.txt" file created in the previous question and displays its content on the console. The program should handle cases where the file does not exist by displaying an appropriate error message.

```
 import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class ReadDiary {

    public static void main(String[] args) {
        File diaryFile = new File("diary.txt");

        if (!diaryFile.exists()) {
            System.out.println("The file diary.txt does not exist.");
            return;
        }

        try (BufferedReader reader = new BufferedReader(new FileReader(diaryFile))) {
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.err.println("Error reading from file: " + e.getMessage());
        }
    }
}
```

Output:-

2023-03-15 14:22:12
Today was a great day! I had a lot of fun with my friends.

3. Append Content to an Existing File: Write a Java program that adds a new diary entry to the "diary.txt" file without overwriting its existing content. The program should ask the user for the new entry and append it to the file along with a timestamp.

```
 import java.io.BufferedWriter;
import java.io.File;
```

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class AppendDiary {

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

        System.out.println("Enter your diary entry:");
        String entry = scanner.nextLine();

        File diaryFile = new File("diary.txt");

        if (!diaryFile.exists()) {
            System.out.println("The file diary.txt does not exist. Creating a new file.");
            try {
                diaryFile.createNewFile();
            } catch (IOException e) {
                System.err.println("Error creating file: " + e.getMessage());
                return;
            }
        }

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(diaryFile, true))) {
            writer.write(getCurrentDate());
            writer.newLine();
            writer.write(entry);
            System.out.println("Your diary entry has been successfully appended to diary.txt");
        } catch (IOException e) {
            System.err.println("Error writing to file: " + e.getMessage());
        }
    }

    private static String getCurrentDate() {
        return new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());
    }
}
```

Output:-

Enter your diary entry:
I also finished my homework today.
Your diary entry has been successfully appended to diary.txt
diary.txt
2023-03-15 14:22:12
Today was a great day! I had a lot of fun with my friends.
2023-03-15 14:22:12
I also finished my homework today.

4. List Files and Directories: Write a program in Java that asks the user for a directory path and then lists all files and subdirectories in that directory. If the directory does not exist, the program should inform the user.

```
 import java.io.File;
import java.util.Scanner;

public class ListFilesAndDirectories {
```

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

        System.out.println("Enter the directory path:");
        String directoryPath = scanner.nextLine();

        File directory = new File(directoryPath);

        if (!directory.exists()) {
            System.out.println("The directory does not exist.");
            return;
        }

        if (!directory.isDirectory()) {
            System.out.println("The given path is not a directory.");
            return;
        }

        File[] files = directory.listFiles();

        for (File file : files) {
            if (file.isFile()) {
                System.out.println("File: " + file.getName());
            } else if (file.isDirectory()) {
                System.out.println("Directory: " + file.getName());
            }
        }
    }
}
```

Output:-

Enter the directory path:
/home/user/documents
File: document1.txt
File: document2.pdf
Directory: subdirectory1
Directory: subdirectory2

5. Filter and List Specific File Types: Create a Java application that lists all the ".txt" files in a given directory. The program should prompt the user for the directory path and then display a list of all text files found in that directory.

```
import java.io.File;
import java.util.Scanner;

public class ListTextFiles {

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

        System.out.println("Enter the directory path:");
        String directoryPath = scanner.nextLine();

        File directory = new File(directoryPath);

        if (!directory.exists()) {
```

```
            System.out.println("The directory does not exist.");
            return;
        }

        if (!directory.isDirectory()) {
            System.out.println("The given path is not a directory.");
            return;
        }

        File[] files = directory.listFiles((dir, name) -> name.toLowerCase().endsWith(".txt"));

        for (File file : files) {
            System.out.println("File: " + file.getName());
        }
    }
}
```

Output:-

Enter the directory path:
/home/user/documents
File: document1.txt
File: document2.txt
File: report.txt

6. Delete a Specific File: Write a Java program where the user can enter the name of a file to be deleted from the system. The program should check if the file exists and delete it, providing a confirmation message upon successful deletion or an error message if the file does not exist.

```
import java.io.File;
import java.util.Scanner;

public class DeleteFile {

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

        System.out.println("Enter the file name to delete:");
        String fileName = scanner.nextLine();

        File file = new File(fileName);

        if (file.exists()) {
            boolean isDeleted = file.delete();
            if (isDeleted) {
                System.out.println("File deleted successfully.");
            } else {
                System.err.println("Error deleting the file.");
            }
        } else {
            System.err.println("File not found.");
        }
    }
}
```

Output:-

Enter the file name to delete:
example.txt
File deleted successfully.

7. Copy File Content: Write a Java program that copies the content from one file (source) to another (destination). The program should prompt the user for both source and destination file paths and perform the copy operation, ensuring that it doesn't overwrite an existing file without user confirmation.

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class CopyFileContent {

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

        System.out.println("Enter the source file name:");
        String sourceFileName = scanner.nextLine();

        File sourceFile = new File(sourceFileName);

        if (!sourceFile.exists()) {
            System.err.println("Source file not found.");
            return;
        }

        System.out.println("Enter the destination file name:");
        String destinationFileName = scanner.nextLine();

        File destinationFile = new File(destinationFileName);

        if (destinationFile.exists()) {
            System.out.println("Destination file already exists. Do you want to overwrite it? (yes/no)");
            String userResponse = scanner.nextLine();

            if (!userResponse.equalsIgnoreCase("yes")) {
                System.out.println("File copy cancelled.");
                return;
            }
        }

        try (
            BufferedReader reader = new BufferedReader(new FileReader(sourceFile));
            BufferedWriter writer = new BufferedWriter(new FileWriter(destinationFile))
        ) {
            String line;
            while ((line = reader.readLine()) != null) {
                writer.write(line);
                writer.newLine();
            }
            System.out.println("File copied successfully.");
        } catch (IOException e) {
            System.err.println("Error copying file: " + e.getMessage());
        }
    }
```

```
}
```
Output:-
Enter the source file name:
source.txt
Enter the destination file name:
destination.txt
Destination file already exists. Do you want to overwrite it? (yes/no)
yes
File copied successfully.

8. Rename a File: Develop a Java application that renames a specified file. The program should request the current file name and the new file name from the user, renaming the file accordingly and confirming the action upon completion.

```
import java.io.File;
import java.util.Scanner;

public class RenameFile {

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

        System.out.println("Enter the current file name:");
        String currentFileName = scanner.nextLine();

        File currentFile = new File(currentFileName);

        if (!currentFile.exists()) {
            System.err.println("Current file not found.");
            return;
        }

        System.out.println("Enter the new file name:");
        String newFileName = scanner.nextLine();
        File newFile = new File(newFileName);

        if (newFile.exists()) {
            System.err.println("New file name already exists.");
            return;
        }

        boolean isRenamed = currentFile.renameTo(newFile);

        if (isRenamed) {
            System.out.println("File renamed successfully.");
        } else {
            System.err.println("Error renaming file.");
        }
    }
}
```
Output:-
Enter the current file name:
source.txt
Enter the new file name:
destination.txt
File renamed successfully.

9. Display File Metadata: Create a Java program that displays metadata of a specified file. The user should be able to input the file name, and the program should output the file size, last modified date, and other available attributes.

```
import java.io.File;
import java.util.Scanner;

public class FileMetadata {

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

        System.out.println("Enter the file name:");
        String fileName = scanner.nextLine();

        File file = new File(fileName);

        if (!file.exists()) {
            System.err.println("File not found.");
            return;
        }

        System.out.println("File size: " + file.length() + " bytes");
        System.out.println("Last modified: " + new Date(file.lastModified()));
        System.out.println("Readable: " + file.canRead());
        System.out.println("Writable: " + file.canWrite());
        System.out.println("Executable: " + file.canExecute());
        System.out.println("Hidden: " + file.isHidden());
    }
}
```

Output:-

Enter the file name:
example.txt
File size: 10 bytes
Last modified: Wed Mar 15 10:30:00 IST 2023
Readable: true
Writable: true
Executable: false
Hidden: false

10. Recursive Directory Listing: Write a Java program that recursively lists all files and subdirectories within a given directory. The program should prompt the user for the directory path and then display a structured list of all contents, including files and directories nested within any subdirectories.

```
import java.io.File;
import java.util.Scanner;

public class RecursiveDirectoryListing {

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

        System.out.println("Enter the directory path:");
        String directoryPath = scanner.nextLine();

        File directory = new File(directoryPath);
```

```
        if (!directory.exists() || !directory.isDirectory()) {
            System.err.println("Invalid directory path.");
            return;
        }

        listFiles(directory, "");
    }

    private static void listFiles(File directory, String indent) {
        File[] files = directory.listFiles();

        for (File file : files) {
            if (file.isDirectory()) {
                System.out.println(indent + "Directory: " + file.getName());
                listFiles(file, indent + "  ");
            } else {
                System.out.println(indent + "File: " + file.getName());
            }
        }
    }
}
```

Output:-

Enter the directory path:
/home/user/documents
Directory: Documents
Directory: Projects
File: example.txt
File: report.pdf
Directory: Downloads
File: image.jpg
File: video.mp4
File: notes.txt

Strings

1. Write a Java program that illustrates the difference between using string literals and the new keyword for creating String objects. Your program should demonstrate the memory usage implications and how string comparison behaves differently in each case.

```
public class StringComparison {
    public static void main(String[] args) {
        String s1 = "Hello";
        String s2 = "Hello";
        String s3 = new String("Hello");
        String s4 = new String("Hello");

        System.out.println("s1 == s2: " + (s1 == s2));
        System.out.println("s1 == s3: " + (s1 == s3));
        System.out.println("s3 == s4: " + (s3 == s4));
        System.out.println("s1.equals(s3): " + s1.equals(s3));

        System.out.println("s1.intern() == s3: " + (s1.intern() == s3));
    }
}
```

Output:-

s1 == s2: true
s1 == s3: false

s3 == s4: false
s1.equals(s3): true
s1.intern() == s3: true

2. Write a Java program that demonstrates the immutability of the String class and how it implements the CharSequence interface. Your program should illustrate the behaviours that highlight String immutability and its usage as a CharSequence.

```
public class StringImmutability {

   public static void main(String[] args) {
      String s1 = "Hello";
      String s2 = s1.toUpperCase();

      System.out.println("s1: " + s1);
      System.out.println("s2: " + s2);

      s1 = s1 + " World!";

      System.out.println("s1: " + s1);
      System.out.println("s2: " + s2);

      CharSequence cs = "Hello";

      System.out.println("cs: " + cs);
      System.out.println("cs.subSequence(0, 2): " + cs.subSequence(0, 2));
   }
}
```

Output:-
s1: Hello
s2: HELLO
s1: Hello World!
s2: HELLO
cs: Hello
cs.subSequence(0, 2): He

3. Write a Java program that uses StringBuffer to construct a simple text editor which can perform the following operations:
a. Append a given string to the existing text.
b. Insert a given string at a specified index within the existing text.
c. Delete a portion of text between two specified indices.
d. Reverse the entire text.
e. Replace a portion of the text between two specified indices with a given string.
Your program should display a menu with options to perform each of the above operations. After each operation, print the current state of the text. Also, display the current capacity and length of the StringBuffer after each operation to showcase its dynamic nature.

```
import java.util.Scanner;

public class SimpleTextEditor {

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

      while (true) {
         System.out.println("\nSimple Text Editor Menu:");
```

```
            System.out.println("1. Append a string");
            System.out.println("2. Insert a string at a specified index");
            System.out.println("3. Delete a portion of text between two indices");
            System.out.println("4. Reverse the entire text");
            System.out.println("5. Replace a portion of the text between two indices with a given string");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();

            if (choice == 1) {
                System.out.print("Enter the string to append: ");
                String appendString = scanner.next();
                text.append(appendString);
                System.out.println("Current state of the text: " + text);
                System.out.println("Current capacity: " + text.capacity() + ", length: " + text.length());
            } else if (choice == 2) {
                System.out.print("Enter the index at which to insert the string: ");
                int index = scanner.nextInt();
                System.out.print("Enter the string to insert: ");
                String insertString = scanner.next();
                text.insert(index, insertString);
                System.out.println("Current state of the text: " + text);
                System.out.println("Current capacity: " + text.capacity() + ", length: " + text.length());
            } else if (choice == 3) {
                System.out.print("Enter the start index of the portion to delete: ");
                int startIndex = scanner.nextInt();
                System.out.print("Enter the end index of the portion to delete: ");
                int endIndex = scanner.nextInt();
                text.delete(startIndex, endIndex);
                System.out.println("Current state of the text: " + text);
                System.out.println("Current capacity: " + text.capacity() + ", length: " + text.length());
            } else if (choice == 4) {
                text.reverse();
                System.out.println("Current state of the text: " + text);
                System.out.println("Current capacity: " + text.capacity() + ", length: " + text.length());
            } else if (choice == 5) {
                System.out.print("Enter the start index of the portion to replace: ");
                int startIndex = scanner.nextInt();
                System.out.print("Enter the end index of the portion to replace: ");
                int endIndex = scanner.nextInt();
                System.out.print("Enter the string to replace with: ");
                String replaceString = scanner.next();
                text.replace(startIndex, endIndex, replaceString);
                System.out.println("Current state of the text: " + text);
                System.out.println("Current capacity: " + text.capacity() + ", length: " + text.length());
            } else if (choice == 6) {
                break;
            } else {
                System.out.println("Invalid choice. Please try again.");
            }
        }
        scanner.close();
    }
}
```

Output:-

Simple Text Editor Menu:
1. Append a string
2. Insert a string at a specified index
3. Delete a portion of text between two indices
4. Reverse the entire text
5. Replace a portion of the text between two indices with a given string
6. Exit
Enter your choice: 1
Enter the string to append: Hello
Current state of the text: Hello
Current capacity: 16, length: 5

Simple Text Editor Menu:
1. Append a string
2. Insert a string at a specified index
3. Delete a portion of text between two indices
4. Reverse the entire text
5. Replace a portion of the text between two indices with a given string
6. Exit
Enter your choice: 2
Enter the index at which to insert the string: 2
Enter the string to insert: world
Current state of the text: Heworldllo
Current capacity: 16, length: 8

Simple Text Editor Menu:
1. Append a string
2. Insert a string at a specified index
3. Delete a portion of text between two indices
4. Reverse the entire text
5. Replace a portion of the text between two indices with a given string
6. Exit
Enter your choice: 3
Enter the start index of the portion to delete: 2
Enter the end index of the portion to delete: 4
Current state of the text: Helllo
Current capacity: 16, length: 5

Simple Text Editor Menu:
1. Append a string
2. Insert a string at a specified index
3. Delete a portion of text between two indices
4. Reverse the entire text
5. Replace a portion of the text between two indices with a given string
6. Exit
Enter your choice: 4
Current state of the text: olleH
Current capacity: 16, length: 5

Simple Text Editor Menu:
1. Append a string
2. Insert a string at a specified index
3. Delete a portion of text between two indices
4. Reverse the entire text
5. Replace a portion of the text between two indices with a given string

6. Exit
Enter your choice: 5
Enter the start index of the portion to replace: 0
Enter the end index of the portion to replace: 2
Enter the string to replace with: hi
Current state of the text: hillo
Current capacity: 16, length: 5

Simple Text Editor Menu:
1. Append a string
2. Insert a string at a specified index
3. Delete a portion of text between two indices
4. Reverse the entire text
5. Replace a portion of the text between two indices with a given string
6. Exit
Enter your choice: 6

4. Create a Java program that uses StringBuilder to perform a series of text manipulations on a user-provided string. The program should allow users to:
a. Add a substring at a specified position.
b. Remove a range of characters from the string.
c. Modify a character at a specified index.
d. Concatenate another string at the end.
e. Display the current string after each operation.
The program should repeatedly prompt the user to choose an operation until they decide to exit. After each operation, it should display the modified string, demonstrating the mutable nature of StringBuilder.

```
import java.util.Scanner;

public class StringBuilderExample {

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

        while (true) {
            System.out.println("\nStringBuilder Example Menu:");
            System.out.println("1. Add a substring at a specified position");
            System.out.println("2. Remove a range of characters from the string");
            System.out.println("3. Modify a character at a specified index");
            System.out.println("4. Concatenate another string at the end");
            System.out.println("5. Display the current string");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();

            if (choice == 1) {
                System.out.print("Enter the position to insert the substring: ");
                int position = scanner.nextInt();
                System.out.print("Enter the substring to insert: ");
                String substring = scanner.next();
                sb.insert(position, substring);
                System.out.println("Current string: " + sb);
            } else if (choice == 2) {
                System.out.print("Enter the start index of the range to remove: ");
                int startIndex = scanner.nextInt();
```

```
            System.out.print("Enter the end index of the range to remove: ");
            int endIndex = scanner.nextInt();
            sb.delete(startIndex, endIndex);
            System.out.println("Current string: " + sb);
        } else if (choice == 3) {
            System.out.print("Enter the index of the character to modify: ");
            int index = scanner.nextInt();
            System.out.print("Enter the new character: ");
            char newChar = scanner.next().charAt(0);
            sb.setCharAt(index, newChar);
            System.out.println("Current string: " + sb);
        } else if (choice == 4) {
            System.out.print("Enter the string to concatenate: ");
            String concatString = scanner.next();
            sb.append(concatString);
            System.out.println("Current string: " + sb);
        } else if (choice == 5) {
            System.out.println("Current string: " + sb);
        } else if (choice == 6) {
            break;
        } else {
            System.out.println("Invalid choice. Please try again.");
        }
    }

    scanner.close();
  }
}
```

Output:-

StringBuilder Example Menu:
1. Add a substring at a specified position
2. Remove a range of characters from the string
3. Modify a character at a specified index
4. Concatenate another string at the end
5. Display the current string
6. Exit
Enter your choice: 1
Enter the position to insert the substring: 2
Enter the substring to insert: "world"
Current string: "hello world"

StringBuilder Example Menu:
1. Add a substring at a specified position
2. Remove a range of characters from the string
3. Modify a character at a specified index
4. Concatenate another string at the end
5. Display the current string
6. Exit
Enter your choice: 2
Enter the start index of the range to remove: 2
Enter the end index of the range to remove: 5
Current string: "hell"

StringBuilder Example Menu:
1. Add a substring at a specified position

2. Remove a range of characters from the string
3. Modify a character at a specified index
4. Concatenate another string at the end
5. Display the current string
6. Exit
Enter your choice: 3
Enter the index of the character to modify: 1
Enter the new character: 'i'
Current string: "hili"

StringBuilder Example Menu:
1. Add a substring at a specified position
2. Remove a range of characters from the string
3. Modify a character at a specified index
4. Concatenate another string at the end
5. Display the current string
6. Exit
Enter your choice: 4
Enter the string to concatenate: " Java"
Current string: "hili Java"

StringBuilder Example Menu:
1. Add a substring at a specified position
2. Remove a range of characters from the string
3. Modify a character at a specified index
4. Concatenate another string at the end
5. Display the current string
6. Exit
Enter your choice: 5
Current string: "hili Java"

StringBuilder Example Menu:
1. Add a substring at a specified position
2. Remove a range of characters from the string
3. Modify a character at a specified index
4. Concatenate another string at the end
5. Display the current string
6. Exit
Enter your choice: 6

5. Create a Java program that compares the performance of StringBuilder and StringBuffer when performing repeated string concatenations. The program should:
a. Prompt the user to enter a base string and the number of times it should be concatenated to itself.
b. Use StringBuilder to concatenate the string the specified number of times, tracking the time taken to complete the operation.
c. Repeat the process using StringBuffer, again tracking the time taken.
d. Output the time taken for each operation and the final length of the resulting strings to demonstrate both the time efficiency and the result of using StringBuilder and StringBuffer.
Example output of the program could look like this:
Enter the base string:
> Hello
Enter the number of concatenations:
> 10000Using StringBuilder...
Time taken: 5 milliseconds
Final string length: 50000

Using StringBuffer...
Time taken: 6 milliseconds
Final string length: 50000
Comparison: StringBuilder was faster than StringBuffer by 1 millisecond.

```
import java.util.Scanner;

public class StringBuilderVsStringBuffer {

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

        System.out.print("Enter the base string: ");
        String baseString = scanner.nextLine();

        System.out.print("Enter the number of concatenations: ");
        int numConcatenations = scanner.nextInt();

        long startTime, endTime;
        StringBuilder sb = new StringBuilder();
        StringBuffer sbuf = new StringBuffer();

        // Using StringBuilder
        startTime = System.nanoTime();
        for (int i = 0; i < numConcatenations; i++) {
            sb.append(baseString);
        }
        endTime = System.nanoTime();
        long sbTime = (endTime - startTime) / 1_000_000;
        System.out.println("Using StringBuilder...");
        System.out.println("Time taken: " + sbTime + " milliseconds");
        System.out.println("Final string length: " + sb.length());

        // Resetting StringBuilder and StringBuffer
        sb = new StringBuilder();
        sbuf = new StringBuffer();

        // Using StringBuffer
        startTime = System.nanoTime();
        for (int i = 0; i < numConcatenations; i++) {
            sbuf.append(baseString);
        }
        endTime = System.nanoTime();
        long sbufTime = (endTime - startTime) / 1_000_000;
        System.out.println("Using StringBuffer...");
        System.out.println("Time taken: " + sbufTime + " milliseconds");
        System.out.println("Final string length: " + sbuf.length());

        // Comparison
        long diff = sbTime - sbufTime;
        System.out.println("Comparison: StringBuilder was faster than StringBuffer by " + Math.abs(diff) +
" milliseconds.");
    }
}
```

Output:-
Enter the base string:

> Hello
Enter the number of concatenations:
> 10000
Using StringBuilder...
Time taken: 5 milliseconds
Final string length: 50000
Using StringBuffer...
Time taken: 6 milliseconds
Final string length: 50000
Comparison: StringBuilder was faster than StringBuffer by 1 milliseconds.

6.Case Conversion and Comparison: Prompt the user to input two strings. Convert both strings to lowercase and uppercase. Compare the converted strings to check case-insensitive equality. Display the converted strings and the result of the comparison.

```
import java.util.Scanner;

public class CaseConversionAndComparison {

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

        System.out.print("Enter the first string: ");
        String str1 = scanner.nextLine();

        System.out.print("Enter the second string: ");
        String str2 = scanner.nextLine();

        // Convert strings to lowercase
        String lowerStr1 = str1.toLowerCase();
        String lowerStr2 = str2.toLowerCase();

        // Convert strings to uppercase
        String upperStr1 = str1.toUpperCase();
        String upperStr2 = str2.toUpperCase();

        // Compare converted strings for case-insensitive equality
        boolean areEqualIgnoreCase = lowerStr1.equals(lowerStr2);

        // Display converted strings and result of comparison
        System.out.println("Converted to lowercase:");
        System.out.println("String 1: " + lowerStr1);
        System.out.println("String 2: " + lowerStr2);

        System.out.println("\nConverted to uppercase:");
        System.out.println("String 1: " + upperStr1);
        System.out.println("String 2: " + upperStr2);

        System.out.println("\nCase-insensitive comparison result: " + areEqualIgnoreCase);
    }
}
```

Output:-
Enter the first string:
Hello
Enter the second string:
hello

Converted to lowercase:
String 1: hello
String 2: hello

Converted to uppercase:
String 1: HELLO
String 2: HELLO

Case-insensitive comparison result: true

7. Character Array and Search: Ask for a string from the user. Convert the string to a character array. Prompt the user to enter a character to search in the string. Find the first and last occurrences of the character. Display the character array and the positions found (if any).

```
import java.util.Scanner;

public class CharacterArraySearch {

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

        System.out.print("Enter a string: ");
        String str = scanner.nextLine();

        char[] charArray = str.toCharArray();

        System.out.print("Enter a character to search: ");
        char searchChar = scanner.next().charAt(0);

        int firstIndex = -1, lastIndex = -1;

        for (int i = 0; i < charArray.length; i++) {
            if (charArray[i] == searchChar) {
                if (firstIndex == -1) {
                    firstIndex = i;
                }
                lastIndex = i;
            }
        }

        if (firstIndex != -1) {
            System.out.println("Character array:");
            for (int i = 0; i < charArray.length; i++) {
                System.out.print(charArray[i] + " ");
            }
            System.out.println("\nFirst occurrence of '" + searchChar + "': " + firstIndex);
            System.out.println("Last occurrence of '" + searchChar + "': " + lastIndex);
        } else {
            System.out.println("Character '" + searchChar + "' not found in the string.");
        }
    }
}
```

Output:-
Enter a string:
Hello World!
Enter a character to search:

o
Character array:
H e l l o W o r l d !

First occurrence of 'o': 4
Last occurrence of 'o': 7

8. String Concatenation and Character Retrieval: Take two strings from the user. Concatenate them using the string method and the + operator, then display both results. Ask the user for an index number, then display the character at that index.

```
import java.util.Scanner;

public class StringConcatenationAndCharacterRetrieval {

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

        System.out.print("Enter the first string: ");
        String str1 = scanner.nextLine();

        System.out.print("Enter the second string: ");
        String str2 = scanner.nextLine();

        // Concatenate strings using the string method
        String concatenatedString1 = str1.concat(str2);

        // Concatenate strings using the + operator
        String concatenatedString2 = str1 + " " + str2;

        System.out.println("Concatenated using the string method: " + concatenatedString1);
        System.out.println("Concatenated using the + operator: " + concatenatedString2);

        System.out.print("Enter an index number: ");
        int index = scanner.nextInt();

        // Check if the index is within the bounds of the concatenated string
        if (index >= 0 && index < concatenatedString1.length()) {
            System.out.println("Character at index " + index + " is: " + concatenatedString1.charAt(index));
        } else {
            System.out.println("Invalid index.");
        }
    }
}
```

Output:-
Enter the first string:
Hello
Enter the second string:
World
Concatenated using the string method: HelloWorld
Concatenated using the + operator: Hello World
Enter an index number:
4
Character at index 4 is: W

9. Word Replacement in Sentences: Request a sentence and two words from the user: one to search for and one to replace it with. Find the first occurrence of the search word in the sentence. Replace the word using substring operations and concatenation. Display the original and the modified sentences.

```
import java.util.Scanner;

public class WordReplacement {

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

      System.out.print("Enter a sentence: ");
      String sentence = scanner.nextLine();

      System.out.print("Enter the word to search for: ");
      String searchWord = scanner.next();

      System.out.print("Enter the word to replace it with: ");
      String replaceWord = scanner.next();

      int index = sentence.indexOf(searchWord);

      if (index != -1) {
         String newSentence = sentence.substring(0, index) + replaceWord + sentence.substring(index +
searchWord.length());
         System.out.println("Original sentence: " + sentence);
         System.out.println("Modified sentence: " + newSentence);
      } else {
         System.out.println("The word '" + searchWord + "' was not found in the sentence.");
      }
   }
}
```

Output:-

```
Enter a sentence:
Java is a popular programming language.
Enter the word to search for:
Java
Enter the word to replace it with:
Python
Original sentence: Java is a popular programming language.
Modified sentence: Python is a popular programming language.
```

10. Interactive String Explorer: Prompt the user for a string. Display a menu with options to perform various operations: convert to lowercase/uppercase, search for a character/index, or concatenate with another string. Based on user selection, perform the appropriate string operation and show the result.

```
import java.util.Scanner;

public class InteractiveStringExplorer {

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

      System.out.print("Enter a string: ");
      String str = scanner.nextLine();

      while (true) {
```

```
            System.out.println("\nString Operations Menu:");
            System.out.println("1. Convert to lowercase");
            System.out.println("2. Convert to uppercase");
            System.out.println("3. Search for a character");
            System.out.println("4. Search for an index");
            System.out.println("5. Concatenate with another string");
            System.out.println("6. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.println("Lowercase: " + str.toLowerCase());
                    break;
                case 2:
                    System.out.println("Uppercase: " + str.toUpperCase());
                    break;
                case 3:
                    System.out.print("Enter a character to search: ");
                    char searchChar = scanner.next().charAt(0);
                    int index = str.indexOf(searchChar);
                    if (index != -1) {
                        System.out.println("Character found at index: " + index);
                    } else {
                        System.out.println("Character not found.");
                    }
                    break;
                case 4:
                    System.out.print("Enter an index to search: ");
                    int searchIndex = scanner.nextInt();
                    if (searchIndex >= 0 && searchIndex < str.length()) {
                        System.out.println("Character at index " + searchIndex + " is: " + str.charAt(searchIndex));
                    } else {
                        System.out.println("Invalid index.");
                    }
                    break;
                case 5:
                    System.out.print("Enter a string to concatenate: ");
                    String concatStr = scanner.next();
                    str = str.concat(concatStr);
                    System.out.println("Concatenated string: " + str);
                    break;
                case 6:
                    System.out.println("Exiting...");
                    System.exit(0);
                    break;
                default:
                    System.out.println("Invalid choice.");
            }
        }
    }
}
```

Output:-

Enter a string:

Hello World!

String Operations Menu:
1. Convert to lowercase
2. Convert to uppercase
3. Search for a character
4. Search for an index
5. Concatenate with another string
6. Exit
Enter your choice: 1
Lowercase: hello world!