JAVA SEMPREPARATION

1.a) write a Java program to create a class for the object Employee (id, name, basic pay, allowances and deductions) with necessary methods

```
public class Employee {
    // Instance variables
    private int id;
    private String name;
    private double basicPay;
    private double allowances;
    private double deductions;
 // Constructor to initialize the Employee object
    public Employee(int id, String name, double basicPay, double allowances, double
deductions) {
        this.id = id;
        this.name = name;
        this.basicPay = basicPay;
        this.allowances = allowances;
        this.deductions = deductions;
    }
// Method to calculate the gross salary
    public double calculateGrossSalary() {
        return basicPay + allowances;
    }
// Method to calculate the net salary after deductions   public double
calculateNetSalary() {
        return calculateGrossSalary() - deductions;
    }
 // Getters and Setters for the instance variables
    public int getId() {
        return id;
    }
public String getName() {
        return name;
    }
 public double getBasicPay() {
        return basicPay;
    }
public double getAllowances() {
        return allowances;
    }
public double getDeductions() {
        return deductions;
    }
public void setId(int id) {
        this.id = id;
  }
public void setName(String name) {
        this.name = name;
    }
public void setBasicPay(double basicPay) {
        this.basicPay = basicPay;
    }
public void setAllowances(double allowances) {
        this.allowances = allowances;
    }
public void setDeductions(double deductions) {
        this.deductions = deductions;
    }
// Main method for testing
    public static void main(String[] args) {
        // Create an Employee object
        Employee employee = new Employee(1, "John Doe", 50000, 2000, 5000);//
Display employee information
        System.out.println("Employee ID: " + employee.getId());
        System.out.println("Employee Name: " + employee.getName());
        System.out.println("Basic Pay: " + employee.getBasicPay());
        System.out.println("Allowances: " + employee.getAllowances());
        System.out.println("Deductions: " + employee.getDeductions());

        // Calculate and display gross and net salary
        System.out.println("Gross Salary: " + employee.calculateGrossSalary());
        System.out.println("Net Salary: " + employee.calculateNetSalary());
    }
}
```

Algorithm:
Certainly! I'll explain the Java program line by line:

1. `public class Employee {`
   - This line defines the start of the `Employee` class.
2. `private int id;`
   - Declares a private integer variable `id` to store the employee's ID.
3. `private String name;`
   - Declares a private string variable `name` to store the employee's name.
4. `private double basicPay;`
   - Declares a private double variable `basicPay` to store the basic salary of the employee.
5. `private double allowances;` - Declares a private double variable `allowances` to store any allowances the employee receives.

6. `private double deductions;`
   - Declares a private double variable `deductions` to store any deductions from the employee's salary.
7. `public Employee(int id, String name, double basicPay, double allowances, double deductions) {`
   - This is a constructor for the `Employee` class. It takes parameters for `id`, `name`, `basicPay`, `allowances`, and `deductions` and initializes the corresponding instance variables.
8. `this.id = id;`
   - Assigns the `id` parameter to the `id` instance variable using the `this` keyword to differentiate between the parameter and instance variable with the same

name.
9. `this.name = name;`
   - Assigns the `name` parameter to the `name` instance variable.
10. `this.basicPay = basicPay;`
    - Assigns the `basicPay` parameter to the `basicPay` instance variable.
11. `this.allowances = allowances;`
    - Assigns the `allowances` parameter to the `allowances` instance variable.
12. `this.deductions = deductions;`
    - Assigns the `deductions` parameter to the `deductions` instance variable.
13. `public double calculateGrossSalary() {`
 - Defines a method `calculateGrossSalary` that calculates the gross salary of the employee.
14. `return basicPay + allowances;`
    - Returns the sum of the `basicPay` and `allowances` as the gross salary.
15. `public double calculateNetSalary() {`
    - Defines a method `calculateNetSalary` that calculates the net salary of the employee after deductions.
16. `return calculateGrossSalary() - deductions;`
    - Calls the `calculateGrossSalary` method and subtracts `deductions` to compute the net salary.
17. The following lines define getter and setter methods for the instance variables, allowing you to access and modify the attributes of an `Employee` object.
18. `public static void main(String[] args) {`
    - This is the entry point of the program, the `main` method.
19. `Employee employee = new Employee(1, "John Doe", 50000, 2000, 5000);`
    - Creates a new `Employee` object with the specified attributes (ID, name, basic pay, allowances, and deductions).
20. The following lines display the employee's information and calculate the gross and net salary using the created `Employee` object.
21. The program ends.

This program demonstrates the creation of an `Employee` class with attributes and methods to calculate the gross and net salary of an employee. The `main` method is used for testing the class by creating an `Employee` object and performing some operations with it.

OUTPUT:

```
Employee ID: 1
Employee Name: John Doe
Basic Pay: 50000.0
Allowances: 2000.0
Deductions: 5000.0
Gross Salary: 52000.0
Net Salary: 47000.0
```

1.b) Write a Java program to check the given number is automorphic number or not. (Note: an automorphic number if and only if the square of the given number ends with the same number itself)

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

```
public class AutomorphicNumber {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();
        scanner.close();

        if (isAutomorphicNumber(number)) {
            System.out.println(number + " is an automorphic number.");
        } else {
            System.out.println(number + " is not an automorphic number.");
        }
    }

    // Function to check if a number is an automorphic number
    public static boolean isAutomorphicNumber(int number) {
        // Calculate the square of the number
        int square = number * number;

        // Convert the number and its square to strings
        String numStr = String.valueOf(number);
        String squareStr = String.valueOf(square);

        // Check if the square string ends with the number string
        return squareStr.endsWith(numStr);
    }
}
```

Algorithm:
Certainly! I'll explain the provided Java program line by line:
1. `import java.util.Scanner;` - This line imports the `Scanner` class from the `java.util` package, which is used for reading input from the user.
2. `public class AutomorphicNumber {`- This line defines the start of the `AutomorphicNumber` class.
3. `public static void main(String[] args) {`- This is the entry point of the program, the `main` method. It takes an array of strings as arguments (though not used in this program).
4. `Scanner scanner = new Scanner(System.in);`   - This line creates a new `Scanner` object named `scanner` to read input from the standard input (keyboard).
5. `System.out.print("Enter a number: ");`   - It prints a prompt to the console, asking the user to enter a number.
6. `int number = scanner.nextInt();`- This line reads an integer from the user using the `nextInt()` method of the `Scanner` class and stores it in the variable `number`.
7. `scanner.close();`- Closes the `Scanner` object to release system resources when we're done with it.

8. `if (isAutomorphicNumber(number)) {`- Calls the `isAutomorphicNumber` method and checks if it returns `true`. If so, it means the number is an automorphic number.

9. `System.out.println(number + " is an automorphic number.");`- Prints a message indicating that the number is an automorphic number.
10. `} else {` - If the condition in line 8 is `false`, it means the number is not an automorphic number. So, the program proceeds to this block.
11. `System.out.println(number + " is not an automorphic number.");` - Prints a message indicating that the number is not an automorphic number.
12. `}`- Closes the `else` block.
13. `public static boolean isAutomorphicNumber(int number) {` - This line defines the `isAutomorphicNumber` method, which takes an integer `number` as an argument and returns a boolean (`true` if automorphic, `false` if not).
14. `int square = number * number;`- Calculates the square of the given `number` and stores it in the variable `square`.
15. `String numStr = String.valueOf(number);` - Converts the `number` to a string and stores it in the variable `numStr`.
16. `String squareStr = String.valueOf(square);` - Converts the `square` to a string and stores it in the variable `squareStr`.
17. `return squareStr.endsWith(numStr);` - Uses the `endsWith` method to check if the string representation of the square (`squareStr`) ends with the string representation of the number (`numStr`). If it does, the method returns `true`, indicating that the number is automorphic. Otherwise, it returns `false`.
18. `}`- Closes the `isAutomorphicNumber` method.

19. `}` - Closes the `main` method.
This program allows the user to input a number, and it checks whether that number is an automorphic number (a number whose square ends with the same number itself). It then prints the result to the console.
Output:
```
Enter a number: 5
5 is an automorphic number.
```

2.a)Write a Java program to create a class for the object Product (id, name, and price) with necessary methods
```java
public class Product {
    // Instance variables
    private int id;
    private String name;
    private double price;

    // Constructor to initialize the Product object
    public Product(int id, String name, double price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }

    // Getter method for retrieving the product ID
    public int getId() {
        return id;
    }
    // Getter method for retrieving the product name
    public String getName() {
        return name;
    }

    // Getter method for retrieving the product price
    public double getPrice() {
        return price;
    }

    // Setter method for updating the product price
    public void setPrice(double newPrice) {
        price = newPrice;
    }

    // Method to display product information
    public void displayProductInfo() {
        System.out.println("Product ID: " + id);
        System.out.println("Product Name: " + name);
        System.out.println("Product Price: " + price);
    }

    // Main method for testing
    public static void main(String[] args) {
        // Create a Product object
        Product product = new Product(1, "Laptop", 999.99);

        // Display product information
        product.displayProductInfo();

        // Update the product price
        product.setPrice(899.99);

        // Display updated product information
        System.out.println("Updated Product Price: " + product.getPrice());
    }
}
```

Algorithm:
Certainly! I'll explain the provided Java program line by line:

1. `public class Product {`
   - This line defines the start of the `Product` class.

2. `private int id;`
   - Declares a private integer variable `id` to store the product's ID.

3. `private String name;`
   - Declares a private string variable `name` to store the product's name.

4. `private double price;`
   - Declares a private double variable `price` to store the product's price.

5. `public Product(int id, String name, double price) {`
   - This is a constructor for the `Product` class. It takes parameters for `id`, `name`, and `price` and initializes the corresponding instance variables.

6. `this.id = id;`
   - Assigns the `id` parameter to the `id` instance variable using the `this` keyword to differentiate between the parameter and instance variable with the same name.

7. `this.name = name;`
   - Assigns the `name` parameter to the `name` instance variable.

8. `this.price = price;`
   - Assigns the `price` parameter to the `price` instance variable.

9. `public int getId() {`
   - Defines a getter method `getId` to retrieve the product's ID.

10. `return id;`
    - Returns the value of the `id` instance variable.

11. `public String getName() {`
    - Defines a getter method `getName` to retrieve the product's name.

12. `return name;`
    - Returns the value of the `name` instance variable.

13. `public double getPrice() {`
    - Defines a getter method `getPrice` to retrieve the product's price.

14. `return price;`
    - Returns the value of the `price` instance variable.

15. `public void setPrice(double newPrice) {`
    - Defines a setter method `setPrice` to update the product's price.

16. `price = newPrice;`
    - Sets the `price` instance variable to the new price provided as an argument to the method.

17. `public void displayProductInfo() {`
    - Defines a method `displayProductInfo` to display the product's information.

18. `System.out.println("Product ID: " + id);`
    - Prints the product's ID to the console.

19. `System.out.println("Product Name: " + name);`
    - Prints the product's name to the console.

20. `System.out.println("Product Price: " + price);`
    - Prints the product's price to the console.

21. `public static void main(String[] args) {`
    - This is the entry point of the program, the `main` method.

22. `Product product = new Product(1, "Laptop", 999.99);`
    - Creates a new `Product` object with the specified attributes (ID, name, and price).

23. `product.displayProductInfo();`
    - Calls the `displayProductInfo` method to display the product information.

24. `product.setPrice(899.99);`
    - Calls the `setPrice` method to update the product's price.

25. `System.out.println("Updated Product Price: " + product.getPrice());`
    - Prints the updated product price to the console.

26. `}`
    - Closes the `main` method.

27. `}`
    - Closes the `Product` class.

This program defines a `Product` class with attributes, getter and setter methods, and a method to display the product information. In the `main` method, it creates a `Product` object, displays its information, updates the price, and displays the updated information.

OUTPUT:
```
Product ID: 1
Product Name: Laptop
Product Price: 999.99
Updated Product Price: 899.99
```

2.b)(b) Write a Java program to print the Armstrong numbers between 1000 and 9999
```java
public class ArmstrongNumbers {
    public static void main(String[] args) {
        System.out.println("Armstrong numbers between 1000 and 9999:");
        for (int number = 1000; number <= 9999; number++) {
            if (isArmstrongNumber(number)) {
                System.out.println(number);
            }
        }
    }

    // Function to check if a number is an Armstrong number
    public static boolean isArmstrongNumber(int number) {
        int originalNumber = number;
        int numberOfDigits = (int) Math.log10(number) + 1;
        int sum = 0;

        while (number > 0) {
            int digit = number % 10;
            sum += Math.pow(digit, numberOfDigits);
            number /= 10;
        }

        return originalNumber == sum;
    }
}
```

ALGORITHM:
Certainly, I'll explain the Java program to find Armstrong numbers between 1000 and 9999 line by line:

1. `public class ArmstrongNumbers {`
   - This line defines the start of the `ArmstrongNumbers` class.

2. `public static void main(String[] args) {`
   - This is the entry point of the program, the `main` method. It takes an array of strings as arguments (not used in this program).

3. `System.out.println("Armstrong numbers between 1000 and 9999:");`
   - Prints a header indicating that the program will display Armstrong numbers between the specified range.

4. `for (int number = 1000; number <= 9999; number++) {`
   - Starts a `for` loop that iterates from 1000 to 9999, inclusive.

5. `if (isArmstrongNumber(number)) {`
   - Checks if the current number is an Armstrong number by calling the `isArmstrongNumber` method.

6. `System.out.println(number);`
   - If the number is an Armstrong number, it is printed to the console.

7. `}` (end of the `if` block)

8. `}` (end of the `for` loop)

9. `public static boolean isArmstrongNumber(int number) {`
   - This line defines the `isArmstrongNumber` method, which checks if a given number is an Armstrong number and returns a boolean (`true` if it's an Armstrong

number, `false` if not).

10. `int originalNumber = number;`
    - Stores the original number for later comparison.

11. `int numberOfDigits = (int) Math.log10(number) + 1;`
    - Calculates the number of digits in the given number using the logarithm function.

12. `int sum = 0;`
    - Initializes a `sum` variable to 0, which will store the sum of each digit raised to the power of the number of digits.

13. `while (number > 0) {`
    - Starts a `while` loop to iterate through the digits of the number.

14. `int digit = number % 10;`
    - Retrieves the last digit of the number using the modulo operation and stores it in the `digit` variable.

15. `sum += Math.pow(digit, numberOfDigits);`
    - Raises the current digit to the power of the number of digits and adds the result to the `sum`.

16. `number /= 10;`
    - Removes the last digit from the number to continue processing the remaining digits.

17. `}` (end of the `while` loop)

18. `return originalNumber == sum;`
    - Compares the original number with the `sum` to check if they are equal. If they are equal, the method returns `true`, indicating that the number is an Armstrong number. Otherwise, it returns `false`.

19. `}` (end of the `isArmstrongNumber` method)

20. `}` (end of the `ArmstrongNumbers` class)

This program defines a class, `ArmstrongNumbers`, with a `main` method to find and display Armstrong numbers in the range 1000 to 9999. It uses a helper method, `isArmstrongNumber`, to check whether a given number is an Armstrong number or not.
OUTPUT:

```
Armstrong numbers between 1000 and 9999:
1634
8208
9474
```

3.a)Write a Java program to create a class for the object Book (Accession number, title, author, publication, and edition) necessary methods for library management

```
public class Book {
    // Instance variables
    private int accessionNumber;
    private String title;
    private String author;
    private String publication;
    private int edition;

    // Constructor to initialize the Book object
    public Book(int accessionNumber, String title, String author, String
publication, int edition) {
        this.accessionNumber = accessionNumber;
        this.title = title;
        this.author = author;
        this.publication = publication;
        this.edition = edition;
    }

    // Getter method to retrieve the accession number
    public int getAccessionNumber() {
        return accessionNumber;
    }

    // Getter method to retrieve the book title
    public String getTitle() {
        return title;
    }

    // Getter method to retrieve the author's name
    public String getAuthor() {
        return author;
    }

    // Getter method to retrieve the publication name
    public String getPublication() {
        return publication;
    }

    // Getter method to retrieve the edition number
    public int getEdition() {
        return edition;
    }

    // Setter method to update the edition number
    public void setEdition(int newEdition) {
        edition = newEdition;
    }
    // Method to display book information
    public void displayBookInfo() {
        System.out.println("Accession Number: " + accessionNumber);
        System.out.println("Title: " + title);
        System.out.println("Author: " + author);
        System.out.println("Publication: " + publication);
        System.out.println("Edition: " + edition);
    }

    // Method to check if the book is available
    public boolean isAvailable() {
        // Add logic to check if the book is available in the library
        // For demonstration, we'll return true by default
        return true;
    }

    // Method to borrow the book
    public void borrowBook() {
        // Add logic to update book status (e.g., mark it as borrowed)
        // For demonstration, we'll print a message
        System.out.println("Book has been borrowed.");
    }

    // Method to return the book
    public void returnBook() {
        // Add logic to update book status (e.g., mark it as returned)
        // For demonstration, we'll print a message
        System.out.println("Book has been returned.");
    }

    public static void main(String[] args) {
        // Create a Book object
        Book book = new Book(12345, "Java Programming", "John Doe", "ABC
Publications", 2);

        // Display book information
        book.displayBookInfo();

        // Update the edition
        book.setEdition(3);

        // Display updated book information
        System.out.println("Updated Edition: " + book.getEdition());
    }
}
```

Algorithm:
Certainly! I'll explain the provided Java program for a `Book` class and its methods line by line:

1. `public class Book {`
   - This line defines the start of the `Book` class.

2. `private int accessionNumber;`
   - Declares a private integer variable `accessionNumber` to store the book's accession number.

3. `private String title;`
   - Declares a private string variable `title` to store the book's title.

4. `private String author;`
   - Declares a private string variable `author` to store the author's name.

5. `private String publication;`
   - Declares a private string variable `publication` to store the name of the publication.

6. `private int edition;`
   - Declares a private integer variable `edition` to store the book's edition number.

7. `public Book(int accessionNumber, String title, String author, String publication, int edition) {`
   - This is a constructor for the `Book` class. It takes parameters for `accessionNumber`, `title`, `author`, `publication`, and `edition` and initializes the corresponding instance variables.

8. `this.accessionNumber = accessionNumber;`
   - Assigns the `accessionNumber` parameter to the `accessionNumber` instance variable using the `this` keyword to differentiate between the parameter and instance variable with the same name.

9. `this.title = title;`
   - Assigns the `title` parameter to the `title` instance variable.

10. `this.author = author;`
    - Assigns the `author` parameter to the `author` instance variable.

11. `this.publication = publication;`
    - Assigns the `publication` parameter to the `publication` instance variable.

12. `this.edition = edition;`
    - Assigns the `edition` parameter to the `edition` instance variable.

13. `public int getAccessionNumber() {`
    - Defines a getter method `getAccessionNumber` to retrieve the book's accession number.

14. `return accessionNumber;`
    - Returns the value of the `accessionNumber` instance variable.

15. `public String getTitle() {`
    - Defines a getter method `getTitle` to retrieve the book's title.

16. `return title;`
    - Returns the value of the `title` instance variable.

17. The program also includes getter methods for `author`, `publication`, and `edition`, which are similar to the previous ones.

18. `public void setEdition(int newEdition) {`
    - Defines a setter method `setEdition` to update the book's edition number.

19. `edition = newEdition;`
    - Sets the `edition` instance variable to the new edition provided as an argument to the method.

20. `public void displayBookInfo() {`
    - Defines a method `displayBookInfo` to display the book's information.

21. The program then proceeds to print out the book's accession number, title, author, publication, and edition.

22. The program includes additional methods such as `isAvailable`, `borrowBook`, and `returnBook`, which are stubs for library management operations. These methods don't have specific implementations and return default values or print messages for demonstration purposes.

23. In the `main` method, a `Book` object is created with sample data, and its information is displayed.

24. The program also demonstrates updating the book's edition and displaying the updated information.

25. `}`
    - Closes the `Book` class.

This program defines a `Book` class with attributes and methods for book information and library management operations. The `main` method demonstrates the creation and usage of a `Book` object.

Output:

```
Accession Number: 12345
Title: Java Programming
Author: John Doe
Publication: ABC Publications
Edition: 2
Updated Edition: 3
```

```
-------------------------------------
3.b)(b) Write a Java program to eliminate duplicate elements in an array
import java.util.Arrays;

public class RemoveDuplicatesFromArray {
    public static void main(String[] args) {
        // Sample array with duplicate elements
        int[] arr = {1, 2, 2, 3, 4, 4, 5, 6, 6, 7};

        // Remove duplicates from the array
        int[] uniqueArray = removeDuplicates(arr);

        // Display the unique elements
        System.out.println("Original Array:");
        for (int element : arr) {
            System.out.print(element + " ");
        }

        System.out.println("\nUnique Array (after removing duplicates):");
        for (int element : uniqueArray) {
            System.out.print(element + " ");
        }
    }

    public static int[] removeDuplicates(int[] arr) {
        // Sort the array to group duplicate elements together
        Arrays.sort(arr);

        int n = arr.length;
        int uniqueCount = 0;

        // Count unique elements
        for (int i = 0; i < n; i++) {
            if (i == 0 || arr[i] != arr[i - 1]) {
                uniqueCount++;
            }
        }

        // Create a new array to store unique elements
        int[] uniqueArray = new int[uniqueCount];

        int index = 0;

        // Copy unique elements to the new array
        for (int i = 0; i < n; i++) {
            if (i == 0 || arr[i] != arr[i - 1]) {
                uniqueArray[index] = arr[i];
                index++;
            }
        }
        return uniqueArray;
    }
}
```

Algorithm:

Certainly, I'll explain the Java program for removing duplicate elements from an array line by line:

1. `import java.util.Arrays;`
   - This line imports the `java.util.Arrays` class, which is used to sort the array in the program.

2. `public class RemoveDuplicatesFromArray {`
   - This line defines the start of the `RemoveDuplicatesFromArray` class.

3. `public static void main(String[] args) {`
   - This is the entry point of the program, the `main` method. It takes an array of strings as arguments (not used in this program).

4. `int[] arr = {1, 2, 2, 3, 4, 4, 5, 6, 6, 7};`
   - This line defines a sample array named `arr` with duplicate elements.

5. `int[] uniqueArray = removeDuplicates(arr);`
   - Calls the `removeDuplicates` method to eliminate duplicates and stores the result in the `uniqueArray` variable.

6. The program then displays both the original array and the array after removing duplicates.

7. `public static int[] removeDuplicates(int[] arr) {`
   - This line defines the `removeDuplicates` method, which takes an integer array as input and returns an integer array.

8. `Arrays.sort(arr);`
   - Sorts the input array `arr` in ascending order. This is done to group duplicate elements together.

9. `int n = arr.length;`
   - Calculates the length of the sorted array, which is the same as the original array.

10. `int uniqueCount = 0;`
    - Initializes a variable `uniqueCount` to keep track of the count of unique elements.

11. The program then enters a loop to count the number of unique elements.

12. `for (int i = 0; i < n; i++) {`

    - Starts a `for` loop to iterate through the sorted array.

13. `if (i == 0 || arr[i] != arr[i - 1]) {`
    - Checks if the current element is the first element (`i == 0`) or if it's different from the previous element. If either condition is met, it indicates a unique element.

14. `uniqueCount++;`
    - If the condition in the previous line is met, increment `uniqueCount` to count the unique element.

15. The loop continues to count unique elements by iterating through the sorted array.

16. `int[] uniqueArray = new int[uniqueCount];`
    - Creates a new integer array `uniqueArray` of size `uniqueCount` to store the unique elements.

17. `int index = 0;`
    - Initializes an `index` variable to keep track of the position in the `uniqueArray`.

18. The program then enters another loop to copy the unique elements to the `uniqueArray`.

19. `for (int i = 0; i < n; i++) {`
    - Starts a new `for` loop to iterate through the sorted array again.

20. `if (i == 0 || arr[i] != arr[i - 1]) {`
    - Checks if the current element is the first element or if it's different from the previous element, indicating a unique element.

21. `uniqueArray[index] = arr[i];`
    - Copies the unique element to the `uniqueArray` at the current `index`.

22. `index++;`
    - Increments the `index` to prepare for the next unique element.

23. The loop continues to copy unique elements to the `uniqueArray`.

24. Finally, the `uniqueArray` is returned.

25. `}`
    - Closes the `removeDuplicates` method.

26. `}`
    - Closes the `RemoveDuplicatesFromArray` class.

This program defines a class, `RemoveDuplicatesFromArray`, and a method, `removeDuplicates`, to remove duplicate elements from an array. It does so by sorting the array, counting unique elements, and then constructing a new array to store the unique elements. The main method demonstrates this by using a sample array and displaying the original and unique arrays.

Output:

```
Original Array:
1 2 2 3 4 4 5 6 6 7
Unique Array (after removing duplicates):
1 2 3 4 5 6 7
```

4.a)Write a Java program to implement Multi level inheritance with suitable objects

```java
class Product {
    private int id;
    private String name;
    private double price;

    public Product(int id, String name, double price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }

    public void displayInfo() {
        System.out.println("Product ID: " + id);
        System.out.println("Product Name: " + name);
        System.out.println("Product Price: $" + price);
    }
}

class ElectronicProduct extends Product {
    private String brand;

    public ElectronicProduct(int id, String name, double price, String brand) {
        super(id, name, price);
        this.brand = brand;
    }

    public void displayElectronicInfo() {
        displayInfo();
        System.out.println("Brand: " + brand);
    }
}

class Laptop extends ElectronicProduct {
    private String processor;

    public Laptop(int id, String name, double price, String brand, String processor)
{
        super(id, name, price, brand);
        this.processor = processor;
    }

    public void displayLaptopInfo() {
        displayElectronicInfo();
        System.out.println("Processor: " + processor);
    }
}

public class MultiLevelInheritanceDemo {
    public static void main(String[] args) {
        // Create objects for each class
        Product product = new Product(1, "Phone", 499.99);
        ElectronicProduct electronicProduct = new ElectronicProduct(2, "TV", 799.99,
"Sony");
        Laptop laptop = new Laptop(3, "Laptop", 999.99, "Dell", "Intel i7");

        // Display information for each product
        System.out.println("Product Information:");
        product.displayInfo();

        System.out.println("\nElectronic Product Information:");
        electronicProduct.displayElectronicInfo();

        System.out.println("\nLaptop Information:");
        laptop.displayLaptopInfo();
    }
}
```

Algorithm:
I'll explain the provided Java program for multi-level inheritance with product classes line by line:

1. `class Product {`
   - This line defines the `Product` class, which serves as the base class for product objects.

2. `private int id;`
   - Declares a private integer variable `id` to store the product's ID.

3. `private String name;`
   - Declares a private string variable `name` to store the product's name.

4. `private double price;`
   - Declares a private double variable `price` to store the product's price.

5. `public Product(int id, String name, double price) {`
   - This is the constructor for the `Product` class. It initializes the `id`,

`name`, and `price` instance variables when a `Product` object is created.

6. `this.id = id;`
   - Assigns the value of the `id` parameter to the `id` instance variable using the `this` keyword.

7. `this.name = name;`
   - Assigns the value of the `name` parameter to the `name` instance variable.

8. `this.price = price;`
   - Assigns the value of the `price` parameter to the `price` instance variable.

9. `public void displayInfo() {`
   - Defines a method `displayInfo` to display information about the product.

10. `System.out.println("Product ID: " + id);`
    - Displays the product's ID.

11. `System.out.println("Product Name: " + name);`
    - Displays the product's name.

12. `System.out.println("Product Price: $" + price);`
    - Displays the product's price.

13. `}` (end of the `Product` class)

14. `class ElectronicProduct extends Product {`
    - This line defines the `ElectronicProduct` class, which inherits from the `Product` class and extends it.

15. `private String brand;`
    - Declares a private string variable `brand` to store the brand of the electronic product.

16. `public ElectronicProduct(int id, String name, double price, String brand) {`
    - This is the constructor for the `ElectronicProduct` class. It initializes the inherited attributes from the `Product` class and the `brand` attribute.

17. `super(id, name, price);`
    - Calls the constructor of the parent class (`Product`) to initialize the `id`, `name`, and `price` attributes.

18. `this.brand = brand;`
    - Assigns the value of the `brand` parameter to the `brand` attribute.

19. `public void displayElectronicInfo() {`
    - Defines a method `displayElectronicInfo` to display information about the electronic product.

20. `displayInfo();`

    - Calls the `displayInfo` method from the parent class (`Product`) to display the basic product information.

21. `System.out.println("Brand: " + brand);`
    - Displays the brand of the electronic product.

22. `}` (end of the `ElectronicProduct` class)

23. `class Laptop extends ElectronicProduct {`
    - This line defines the `Laptop` class, which inherits from the `ElectronicProduct` class and further extends the hierarchy.

24. `private String processor;`
    - Declares a private string variable `processor` to store the processor information for the laptop.

25. `public Laptop(int id, String name, double price, String brand, String processor) {`
    - This is the constructor for the `Laptop` class. It initializes attributes inherited from the `ElectronicProduct` class and the `processor` attribute.

26. `super(id, name, price, brand);`
    - Calls the constructor of the parent class (`ElectronicProduct`) to initialize the inherited attributes.

27. `this.processor = processor;`
    - Assigns the value of the `processor` parameter to the `processor` attribute.

28. `public void displayLaptopInfo() {`
    - Defines a method `displayLaptopInfo` to display information about the laptop.

29. `displayElectronicInfo();`
    - Calls the `displayElectronicInfo` method from the parent class (`ElectronicProduct`) to display electronic product information.

30. `System.out.println("Processor: " + processor);`
    - Displays information about the laptop's processor.

31. `}` (end of the `Laptop` class)

32. `public class MultiLevelInheritanceDemo {`
    - This line defines the `MultiLevelInheritanceDemo` class, which contains the `main` method and serves as the entry point for the program.

33. `public static void main(String[] args) {`
    - This is the `main` method where the program execution begins.

34. Objects for each class are created: `Product`, `ElectronicProduct`, and `Laptop`.

35. The program demonstrates multi-level inheritance by calling methods on these objects to display their information.

36. `}` (end of the `main` method)

37. `}` (end of the `MultiLevelInheritanceDemo` class)

The program demonstrates multi-level inheritance by creating objects of product classes in a hierarchy. It allows the display of information about products at different levels of the hierarchy, starting from the base class (`Product`) to the derived classes (`ElectronicProduct` and `Laptop`).

```
OUTPUT:
Product Information:
Product ID: 1
Product Name: Phone
Product Price: $499.99

Electronic Product Information:
Product ID: 2
Product Name: TV
Product Price: $799.99
Brand: Sony

Laptop Information:
Product ID: 3
Product Name: Laptop
Product Price: $999.99
Brand: Dell
Processor: Intel i7
```

4.b)(b) Write a Java program to sort the array elements in descending order

```
import java.util.Arrays;
import java.util.Comparator;

public class SortArrayDescending {
    public static void main(String[] args) {
        // Sample array
        int[] arr = {5, 2, 9, 1, 5, 6};

        // Sort the array in descending order
        Arrays.sort(arr);
        // Reverse the sorted array to get descending order
        reverseArray(arr);

        // Display the sorted array in descending order
        System.out.println("Array in descending order: " + Arrays.toString(arr));
    }
    // Helper method to reverse the array
    public static void reverseArray(int[] arr) {
        int start = 0;
        int end = arr.length - 1;

        while (start < end) {
            // Swap elements at start and end positions
            int temp = arr[start];
            arr[start] = arr[end];
            arr[end] = temp;

            start++;
            end--;
        }
    }
}
```

Alogorithm:

Sure, let's go through the provided Java program that sorts an array in descending order using the `Arrays.sort()` method and then reverses the sorted array. I'll explain the code line by line:

```
import java.util.Arrays;
import java.util.Comparator;
```

1. Import the necessary classes: `java.util.Arrays` for array manipulation and `java.util.Comparator` (although it's not used in this program).

```
public class SortArrayDescending {
```

2. Define a class named `SortArrayDescending`. This is the main class for our program.

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

3. Start the `main` method, which is the entry point of the program.

```
        // Sample array
        int[] arr = {5, 2, 9, 1, 5, 6};
```

4. Create an integer array named `arr` with sample elements.

```
        // Sort the array in descending order
     Arrays.sort(arr);
```

5. Use the `Arrays.sort()` method to sort the `arr` array in ascending order. The array will be modified in-place.

```
        // Reverse the sorted array to get descending order
        reverseArray(arr);
```

6. Call the `reverseArray` method to reverse the sorted array, effectively putting it in descending order.

```
        // Display the sorted array in descending order
        System.out.println("Array in descending order: " + Arrays.toString(arr));
```

7. Display the sorted and reversed array in descending order using `System.out.println`. The `Arrays.toString(arr)` method is used to convert the array into a string for display.

```
    }

    // Helper method to reverse the array
    public static void reverseArray(int[] arr) {
```

8. Define a helper method named `reverseArray`, which takes an integer array `arr` as a parameter. This method will reverse the order of elements in the array.

```
        int start = 0;
        int end = arr.length - 1;
```

9. Initialize two variables, `start` and `end`, to the beginning and end of the array, respectively.

```
      while (start < end) {
```

10. Enter a `while` loop that continues as long as the `start` index is less than the `end` index, meaning there are still elements to swap.

```
            // Swap elements at start and end positions
            int temp = arr[start];
            arr[start] = arr[end];
            arr[end] = temp;
```

11. Within the loop, swap the elements at the `start` and `end` positions. This is done by using a temporary variable `temp` to hold the value of `arr[start]`, then assigning `arr[end]` to `arr[start]` and finally assigning `temp` to `arr[end]`.

```
            start++;
            end--;
```

12. Update the `start` and `end` indices to move closer to each other, effectively reversing the order of elements in the array.

```
        }
    }
```

```
}
```

13. Close the `while` loop and the `reverseArray` method definition.

This program first sorts the array in ascending order using `Arrays.sort()` and then reverses it in place to obtain the elements in descending order. Finally, it displays the sorted array in descending order.

Output:

```
Array in descending order: [9, 6, 5, 5, 2, 1]
```

5.(a) Write a Java program to implement Hierarchical inheritance with suitable objects

```
class Vehicle {
    protected String brand;
    protected String model;

    public Vehicle(String brand, String model) {
        this.brand = brand;
        this.model = model;
    }

    public void displayInfo() {
        System.out.println("Brand: " + brand);
        System.out.println("Model: " + model);
    }
}

// Child class 1 (Subclass) inheriting from the Vehicle class
class Car extends Vehicle {
    private int numDoors;

    public Car(String brand, String model, int numDoors) {
        super(brand, model); // Call the constructor of the parent class (Vehicle)
        this.numDoors = numDoors;
    }

    public void displayCarInfo() {
        displayInfo(); // Call the method from the parent class (Vehicle)
        System.out.println("Number of doors: " + numDoors);
    }
}

// Child class 2 (Subclass) inheriting from the Vehicle class
class Bike extends Vehicle {
    private boolean hasBasket;
```

```java
    public Bike(String brand, String model, boolean hasBasket) {
        super(brand, model); // Call the constructor of the parent class (Vehicle)
        this.hasBasket = hasBasket;
    }

    public void displayBikeInfo() {
        displayInfo(); // Call the method from the parent class (Vehicle)
        System.out.println("Has basket: " + hasBasket);
    }
}

public class Main {
    public static void main(String[] args) {
        // Create a Car object
        Car car = new Car("Toyota", "Camry", 4);
        car.displayCarInfo();

        // Create a Bike object
        Bike bike = new Bike("Schwinn", "Fastback", true);
        bike.displayBikeInfo();
    }
}
```

Algorithm:

Certainly, let's go through the provided Java program line by line, explaining its structure and functionality:

java
Copy code

```java
class Vehicle {
    protected String brand;
    protected String model;

    public Vehicle(String brand, String model) {
        this.brand = brand;
        this.model = model;
    }

    public void displayInfo() {
        System.out.println("Brand: " + brand);
        System.out.println("Model: " + model);
    }
}
```

A class named Vehicle is defined. This class serves as the base class for all types of vehicles.

The class has two protected attributes, brand and model, which are used to store the brand and model of the vehicle.

A constructor Vehicle(String brand, String model) is defined to initialize the brand and model attributes when a Vehicle object is created.

The class also has a method displayInfo() that displays the brand and model of the vehicle.
Another child class named Bike is defined, also inheriting from the Vehicle class. This class represents a different type of vehicle, a bike.

The Bike class has an additional private attribute hasBasket, which indicates whether the bike has a basket.

A constructor Bike(String brand, String model, boolean hasBasket) is defined to initialize the attributes of the Bike class. It calls the constructor of the parent class (Vehicle) using super(brand, model).

The class defines a method displayBikeInfo() to display information about the bike, including the brand, model, and whether it has a basket.

Inside the main method:

An object of the Car class named car is created, initialized with the brand "Toyota," model "Camry," and the number of doors as 4.
The displayCarInfo() method is called to display information about the car.
An object of the Bike class named bike is created, initialized with the brand "Schwinn," model "Fastback," and the information that it has a basket.
The displayBikeInfo() method is called to display information about the bike.
This program demonstrates the concept of inheritance in object-oriented programming. The base class Vehicle defines common attributes and methods for all vehicles, while the child classes Car and Bike inherit from the base class and add their specific attributes and methods. When you run the program, it creates objects of both child classes and displays information about a car and a bike.

The Main class is defined as the entry point of the program.

Inside the main method:

An object of the Car class named car is created, initialized with the brand "Toyota," model "Camry," and the number of doors as 4.
The displayCarInfo() method is called to display information about the car.
An object of the Bike class named bike is created, initialized with the brand "Schwinn," model "Fastback," and the information that it has a basket.
The displayBikeInfo() method is called to display information about the bike.
This program demonstrates the concept of inheritance in object-oriented programming. The base class Vehicle defines common attributes and methods for all vehicles, while the child classes Car and Bike inherit from the base class and add their specific attributes and methods. When you run the program, it creates objects of both child classes and displays information about a car and a bike.

OUTPUT:

```
Brand: Toyota
Model: Camry
Number of doors: 4
Brand: Schwinn
Model: Fastback
Has basket: true
```

=---

5(b)   Write a Java program to check whether given number is Prime number or not

```java
import java.util.Scanner;

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

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

        if (isPrime(number)) {
            System.out.println(number + " is a prime number.");
        } else {
            System.out.println(number + " is not a prime number.");
        }

        scanner.close();
    }

    // Function to check if a number is prime
    public static boolean isPrime(int num) {
        if (num <= 1) {
            return false;
        }
        if (num == 2) {
            return true;
        }
        if (num % 2 == 0) {
            return false;
        }

        for (int i = 3; i <= Math.sqrt(num); i += 2) {
            if (num % i == 0) {
                return false;
            }
        }

        return true;
    }
}
```

~~~
Algorithm:

Certainly, let's go through the provided Java program that checks whether a given
number is a prime number or not, line by line:

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

1. Import the `java.util.Scanner` class to enable user input.
public class PrimeNumberChecker {
```

2. Define a class named `PrimeNumberChecker`. This is the main class for our
program.
    public static void main(String[] args) {
```

3. Start the `main` method, which is the entry point of the program.
        Scanner scanner = new Scanner(System.in);
```

4. Create a `Scanner` object named `scanner` to read input from the user.
        System.out.print("Enter a number: ");
```

5. Display the prompt to the user, asking them to enter a number.

        int number = scanner.nextInt();
```

6. Read an integer input from the user and store it in the variable `number`.

        if (isPrime(number)) {
```

7. Use an `if` statement to check if the result of the `isPrime` method (explained
below) is `true`. If it's `true`, execute the code inside the `if` block.

            System.out.println(number + " is a prime number.");
```

8. If the condition in the `if` statement is `true`, display a message indicating
that the entered number is a prime number.
        } else {
```

9. If the condition in the `if` statement is `false`, execute the code inside the
`else` block.
~~~

            System.out.println(number + " is not a prime number.");
```

10. Display a message indicating that the entered number is not a prime number.
        scanner.close();
```

11. Close the `Scanner` to release resources after using it.

    }

    // Function to check if a number is prime
    public static boolean isPrime(int num) {
```

12. Define a function named `isPrime` that takes an integer `num` as a parameter. This function will check if `num` is a prime number and return a `boolean` value.

        if (num <= 1) {
```

13. Check if the number is less than or equal to 1. If so, return `false` because prime numbers must be greater than 1.

            return false;
        }
```

14. If the number is less than or equal to 1, it's not prime, so we return `false`.

        if (num == 2) {
```

15. Check if the number is equal to 2. If so, return `true` because 2 is a prime number.

            return true;
        }

16. If the number is 2, it's prime, so we return `true`.

        if (num % 2 == 0) {

17. Check if the number is even (except for 2) by checking if it's divisible by 2. If it is, return `false` because even numbers (except 2) are not prime.

            return false;
        }

18. If the number is even (except for 2), it's not prime, so we return `false`.

```
        for (int i = 3; i <= Math.sqrt(num); i += 2)
```

19. Use a `for` loop to iterate from 3 up to the square root of the number (`Math.sqrt(num)`), checking only odd numbers (hence `i += 2`).

```
            if (num % i == 0) {
```

20. Check if the number is divisible by `i`. If it is, return `false` because prime numbers should not have divisors other than 1 and themselves.

```
                return false;
            }
        }
```

21. If the number is divisible by any number `i` in the loop, it's not prime, so we return `false`.

```
        return true;
    }
}
```

22. Close the `isPrime` function definition and the `PrimeNumberChecker` class.

This program checks whether a given number is prime and displays the result. It also handles some special cases like numbers less than or equal to 1, even numbers (except 2), and checks divisibility up to the square root of the number for efficiency.

Output (Example):

```
Enter a number: 17
17 is a prime number.
Enter a number: 4
4 is not a prime number.
Enter a number: 2
2 is a prime number.
```