# 1. Algorithm for Implementing Inheritance with Calculation and My\_Calculation Classes

1. **Start**
2. **Create Class Calculation**:
   * Declare int z.
   * Define methods:
     + addition(x, y): Calculate z = x + y, print z.
     + subtraction(x, y): Calculate z = x - y, print z.
3. **Create Class My\_Calculation**:
   * Extend Calculation.
   * Define multiplication(x, y): Calculate z = x \* y, print z.
4. **Write Main Method**:
   * Create My\_Calculation object.
   * Call addition(), subtraction(), multiplication() with sample inputs.
5. End

# 2. Algorithm for

**1. Start**

* Begin the process.

**2. Define the Point Class**

* **Action 1:** Add x and y as instance variables.
* **Action 2:** Create a constructor to initialize x and y.
* **Action 3:** Implement toString() method to return "(x, y)".

**3. Define the Circle Class**

* **Action 1:** Extend Point class.
* **Action 2:** Add radius as a new instance variable.
* **Action 3:** Create a constructor to initialize x, y, and radius using super.
* **Action 4:** Implement getArea() to calculate πr2\pi r^2πr2.
* **Action 5:** Implement getCircumference() to calculate 2πr2\pi r2πr.
* **Action 6:** Override toString() method to return "Circle[center=(x, y); radius=r]".

**4. Main Method**

* **Action 1:** Create a Circle object.
* **Action 2:** Print the string representation.
* **Action 3:** Print the area.
* **Action 4:** Print the circumference.

**5. End**

# 3. Algorithm

1. Prompt the user to enter a string.
2. Read the string using Scanner.
3. Create a StringBuffer object with the input string.
4. Reverse the string using the reverse() method of StringBuffer.
5. Convert the reversed StringBuffer to a String.
6. Print the reversed string.

# 4. Algorithm

1. Define abstract class A with abstract methods setArray1 and setArray2.
2. Create class C extending A:
   * Implement setArray1 and setArray2 to initialize matrix1 and matrix2.
   * Add multiply method to perform matrix multiplication:
     + Check compatibility of dimensions.
     + Compute the result using nested loops.
   * Add displayMatrix method to print matrices.
3. In Main:
   * Create an object of C.
   * Initialize and set two matrices using setArray1 and setArray2.
   * Display the matrices using displayMatrix.
   * Multiply the matrices using multiply and display the result.

# 5. Algorithm

**Algorithm:**

1. Start
2. Define interface A:
   * takeInputForAutomorphic gets user input.
   * isAutomorphic checks if the number’s square ends with the number.
3. Define interface B:
   * takeInputForDuck gets user input.
   * isDuckNumber checks if the number contains 0 without leading zeros.
4. Implement class C:
   * Implement methods from A and B.
   * Add runChecks to:
     + Check and display if a number is automorphic.
     + Check and display if a number is a duck number.
5. In Main:
   * Create an object of C and call runChecks.
6. End

# 6. Algorithm

1. Start
2. **Take Input:**
   * Prompt the user for total admission test marks.
3. **Check Total Marks:**
   * If marks are less than 40, throw an ArithmeticException.
4. **Take Subject Marks:**
   * Prompt the user for Math and Physics marks.
5. **Validate Subject Marks:**
   * If any subject mark is 0, negative, or less than 20, throw an AdmissionTestException.
6. **Check Admission Eligibility:**
   * If all checks pass, print that the student is eligible for admission in the CSE department.
7. **Handle Exceptions:**
   * Catch and print specific exceptions for invalid inputs or failed conditions.
8. **Close Resources:**
   * Use finally to close the Scanner object.

# 7. Algorithm

1. Start
2. **Define a Safety Feature Interface:**
   * Include an alarm method to warn of danger.
3. **Create a Car Class:**
   * Add properties: isOnWrongSide and isSpeeding.
   * Implement alarm to notify the driver of oncoming cars.
   * Add methods:
     + checkSpeed for speed warnings.
     + detectOncomingCar to detect danger.
4. **Create a Driver Class:**
   * Link a Car object and simulate driving with a drive method.
5. **Main Class:**
   * Instantiate Car and Driver objects.
   * Call drive to simulate driving and trigger alarms.
6. End

# 8. Algorithm

1. **Initialize Balance:** Start with an initial balance of 10,000.
2. **Create withdraw Method:**
   * Check if the amount is divisible by 500 and maintains a minimum balance of 500.
   * Deduct the amount if valid and display the remaining balance.
3. **Create checkBalance Method:** Display the current balance.
4. **Main Logic:**
   * Use a while loop to show a menu:
     + Option 1: Call withdraw.
     + Option 2: Call checkBalance.
     + Option 3: Exit the loop.

# 9. Algorithm

1. Start
2. **Define Abstract Class Employee:**
   * Add common properties: name and baseSalary.
   * Define an abstract method calculateSalary().
3. **Create Manager Class:**
   * Extend Employee.
   * Implement calculateSalary() by adding a bonus to baseSalary.
4. **Create Programmer Class:**
   * Extend Employee.
   * Implement calculateSalary() by adding overtime pay to baseSalary.
5. **Main Class:**
   * Create Manager and Programmer objects with appropriate details.
   * Call calculateSalary() for each and display the results.
6. End

# 10. Algorithm

1. Start
2. **Abstract Class Kitchen:**
   * Method useKitchen(): Print "Using the shared kitchen."
   * Abstract method prepareFood() for each family member's food preparation.
3. **Subclass for Each Member:**
   * You: Implement prepareFood() for "bread and butter."
   * Mother: Implement prepareFood() for "bread, vegetables, and tea."
   * Sister: Implement prepareFood() for "noodles and mango juice."
4. **In FamilyKitchen (Main class):**
   * Create objects for You, Mother, Sister.
   * Call useKitchen() and prepareFood() for each.
5. End