



Southeast University

Department of Computer Science and Engineering (CSE)

School of Sciences and Engineering

Semester: (Summer, Year: 2025)

LAB REPORT NO: 07

Course Title: Introduction to Programming Language II (Java) Lab

Course Code: CSE282.2

Batch: 65

**Lab Experiment Name: Object Oriented Programming concepts
(Inheritance and Polymorphism).**

Student Details

	Name	ID
1.	Md. Mahmud Hossain	2023200000799

Submission Date : 11-09-25

Course Teacher's Name : Dr. Mohammed Ashikur Rahman

Lab Report Status

Marks:

Comments:.....

Signature:.....

Date:.....

Lab Task 6: Object Oriented Programming concepts (Inheritance and Polymorphism)

PROBLEM:

1. Create a class named "Vehicle" with properties such as "brand", "model", "price", and "color". Create two subclasses, "Car" and "Motorcycle," with additional properties specific to each class. Create a main class to initialize objects of the subclasses, store them in an ArrayList, and display their information. Use the 'super' keyword to call the constructor of the superclass from the subclasses and utilize method overriding.
2. Create a class named "Shape" with properties such as "name" and "color". Create two subclasses, "Circle" and "Rectangle," with additional properties specific to each class. Implement methods in the subclasses to calculate the area and perimeter of each shape. Create objects of the subclasses, store them in an ArrayList, and display their information.
3. Create a class named "Employee" with properties such as "name," "id," and "salary." Create two subclasses, "Manager" and "Engineer," with additional properties specific to each class. Implement a method in each subclass to calculate the total salary by including additional bonuses. Create objects of the subclasses, store them in an ArrayList, and display their information.
4. Create a class named "Book" with properties such as "title," "author," and "price." Create two subclasses, "FictionBook" and "NonFictionBook," with additional properties specific to each class. Implement methods in the subclasses to display the book details and perform book - specific actions. Create objects of the subclasses, store them in an ArrayList, and display their information.

Solution:

1.

Problem Analysis:

Managing vehicle information efficiently requires a program that can represent common vehicle attributes while also accommodating type-specific features. The Vehicle class defines general properties like brand, model, price, and color. Subclasses such as Car and Motorcycle extend Vehicle to add attributes unique to them, such as the number of doors for cars and engine capacity for motorcycles. By using inheritance and overriding the toString() method, the program ensures that each type of vehicle can display both common and specialized details. The main

class creates an ArrayList of vehicles, stores different vehicle objects in it, and prints their information, showcasing polymorphism in action.

Background Theory:

Encapsulation: Attributes like brand, model, price, and color are private and accessed through getters and setters.

Inheritance: Car and Motorcycle inherit common properties from the Vehicle class while adding their unique features.

Polymorphism: The overridden toString() method in each subclass ensures that objects display their own specific details when called in a loop.

ArrayList: A dynamic data structure is used to store different types of vehicles, allowing iteration without knowing their exact type at compile time.

Overriding: The subclasses override the toString() method to extend its functionality, ensuring specialized output.

main() Method: Demonstrates the functionality by applying all three sorting algorithms on the same input array and printing the results.

Algorithm Design:

1. Define a Vehicle Class:

- Attributes: brand, model, price, color.
- Methods: constructor, getters/setters, and a toString() method.

2. Define a Car Class (extends Vehicle):

- Additional attribute: numDoors.
- Constructor initializes both common and specific fields.
- Override toString() to display car-specific details. Define a mergeSort() method:

3. Define Motorcycle Class (extends Vehicle)

- Create an array of integers.
- Clone it for each sorting algorithm to ensure fairness in testing.
- Apply Bubble Sort, Selection Sort, and Merge Sort separately.

4. Main Class (VehicleMain)

- Create an ArrayList<Vehicle>.
- Add Car and Motorcycle objects to the list.
- Use a for-each loop to iterate through the list and print details of each vehicle using polymorphism.

Code:

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

```
class Vehicle {  
    private String brand;  
    private String model;  
    private double price;  
    private String color;  
  
    public Vehicle(String brand, String model, double price, String color) {  
        this.brand = brand;  
        this.model = model;  
        this.price = price;  
        this.color = color;  
    }  
  
    public String getBrand() {  
        return brand;  
    }  
  
    public void setBrand(String brand) {  
        this.brand = brand;  
    }  
  
    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 getColor() {
        return color;
    }

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

    @Override
    public String toString() {
        return "Brand: " + brand + ", Model: " + model + ", Price: " + price + ", Color: " + color;
    }
}

class Car extends Vehicle {
    private int numDoors;

    public Car(String brand, String model, double price, String color, int numDoors) {
        super(brand, model, price, color);
        this.numDoors = numDoors;
    }

    public int getNumDoors() {
        return numDoors;
    }

    public void setNumDoors(int numDoors) {
        this.numDoors = numDoors;
    }

    @Override
    public String toString() {
        return super.toString() + ", Number of Doors: " + numDoors;
    }
}

class Motorcycle extends Vehicle {
    private double engineCapacity;

```

```

    public Motorcycle(String brand, String model, double price, String color, double
engineCapacity) {
        super(brand, model, price, color);
        this.engineCapacity = engineCapacity;
    }

    public double getEngineCapacity() {
        return engineCapacity;
    }

    public void setEngineCapacity(double engineCapacity) {
        this.engineCapacity = engineCapacity;
    }

    @Override
    public String toString() {
        return super.toString() + ", Engine Capacity: " + engineCapacity + " cc";
    }
}

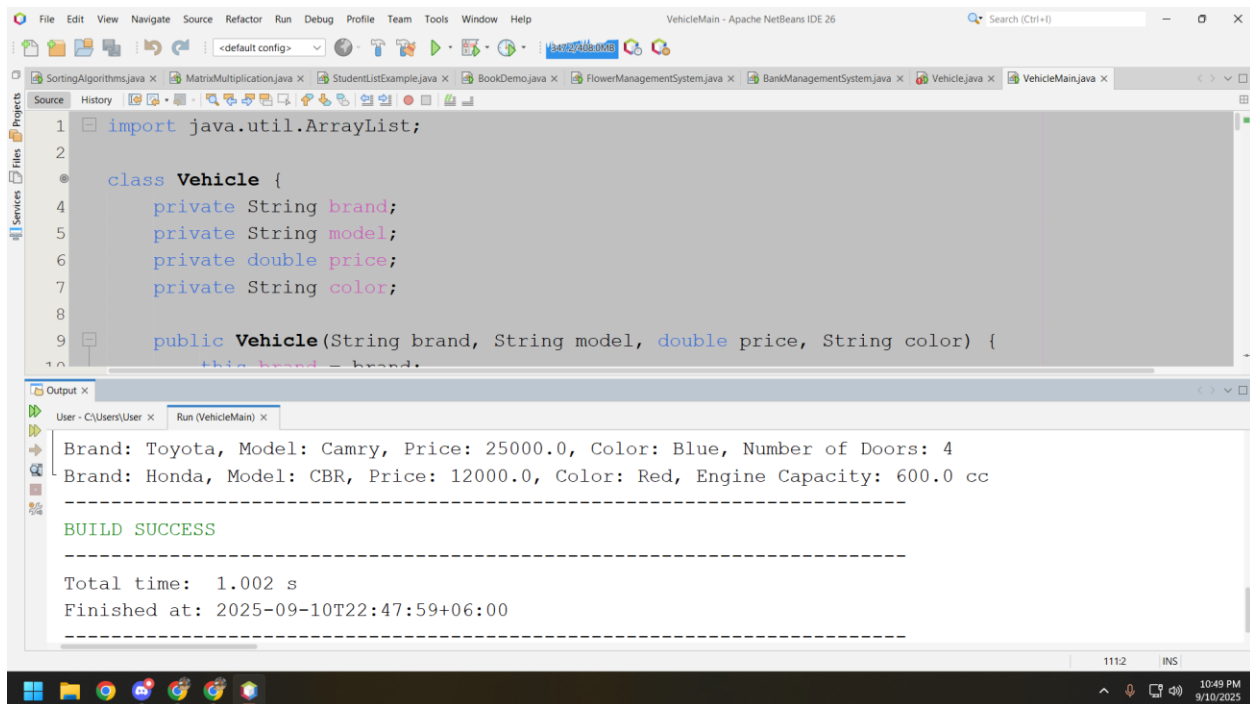
public class Main {
    public static void main(String[] args) {
        ArrayList<Vehicle> vehicles = new ArrayList<>();
        Car car1 = new Car("Toyota", "Camry", 25000, "Blue", 4);
        Motorcycle bike1 = new Motorcycle("Honda", "CBR", 12000, "Red", 600.0);

        vehicles.add(car1);
        vehicles.add(bike1);

        for (Vehicle vehicle : vehicles) {
            System.out.println(vehicle.toString());
        }
    }
}

```

Output:



The screenshot shows the Apache NetBeans IDE interface. The main editor window displays the following Java code:

```
1 import java.util.ArrayList;
2
3 class Vehicle {
4     private String brand;
5     private String model;
6     private double price;
7     private String color;
8
9     public Vehicle(String brand, String model, double price, String color) {
10         this.brand = brand;
11     }
12 }
```

The Output window at the bottom shows the following text:

```
Brand: Toyota, Model: Camry, Price: 25000.0, Color: Blue, Number of Doors: 4
Brand: Honda, Model: CBR, Price: 12000.0, Color: Red, Engine Capacity: 600.0 cc
-----
BUILD SUCCESS
-----
Total time: 1.002 s
Finished at: 2025-09-10T22:47:59+06:00
-----
```

Figure 1: Output

2.

Problem Analysis:

This program performs Geometric shapes share common properties such as a name and color, but each type of shape has unique attributes and distinct formulas for calculating area and perimeter. To avoid redundancy and allow flexibility, a base class Shape is defined with common attributes and placeholder methods. Subclasses like Circle and Rectangle extend Shape, overriding the methods calculateArea() and calculatePerimeter() with their specific implementations. The ShapeMain class creates a collection of different shapes, stores them in an ArrayList, and displays their details using polymorphism.

matrix multiplication between two matrices A and B. The multiplication is valid only if the number of columns in the first matrix equals the number of rows in the second matrix. The program checks this condition, computes the product using nested loops, and stores the result in a new matrix C. Finally, it prints the resulting matrix in row-column format.

Background Theory:

Encapsulation: Shape attributes (name, color, radius, length, width) are private and accessed through getters/setters.

Inheritance: Circle and Rectangle extend the Shape class, reusing its fields and adding specific features.

Polymorphism: The overridden methods in each subclass allow the program to compute the correct area and perimeter depending on the actual object type.

Method Overriding: The calculateArea() and calculatePerimeter() methods are redefined in subclasses with shape-specific formulas.

ArrayList: Used to store multiple Shape objects of different types, making iteration easier without knowing the exact type beforehand.

main() Method: Declares two matrices A and B, checks if multiplication is possible, computes the result, and prints the product matrix.

Algorithm Design:

1. Define Base Class – Shape:

- Attributes: name, color.
- Methods: calculateArea() and calculatePerimeter() returning default values (0.0).
- Override toString() to show common details. Define a Car Class (extends Vehicle):

2. Define Subclass – Circle

- Additional attribute: radius.
- Override calculateArea() = $\pi \times r^2$.
- Override calculatePerimeter() = $2 \times \pi \times r$.
- Extend toString() to display radius, area, and perimeter.

3. Define Subclass – Rectangle

- Additional attributes: length, width.
- Override calculateArea() = length \times width.
- Override calculatePerimeter() = $2 \times (\text{length} + \text{width})$.
- Extend toString() to display length, width, area, and perimeter.

4. Main Class – ShapeMain

- Create an ArrayList<Shape>.
- Add a Circle and Rectangle object.

- Use a for-each loop to print each shape's details.

Code:

```
import java.util.ArrayList;

class Shape {
    private String name;
    private String color;

    public Shape(String name, String color) {
        this.name = name;
        this.color = color;
    }

    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 double calculateArea() {
        return 0.0;
    }

    public double calculatePerimeter() {
        return 0.0;
    }
}
```

```

@Override
public String toString() {
    return "Name: " + name + ", Color: " + color;
}
}

class Circle extends Shape {
    private double radius;

    public Circle(String name, String color, double radius) {
        super(name, color);
        this.radius = radius;
    }

    public double getRadius() {
        return radius;
    }

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

    @Override
    public double calculateArea() {
        return Math.PI * radius * radius;
    }

    @Override
    public double calculatePerimeter() {
        return 2 * Math.PI * radius;
    }

    @Override
    public String toString() {
        return super.toString() + ", Radius: " + radius + ", Area: " + calculateArea() + ", Perimeter: " + calculatePerimeter();
    }
}

class Rectangle extends Shape {

```

```

private double length;
private double width;

public Rectangle(String name, String color, double length, double width) {
    super(name, color);
    this.length = length;
    this.width = width;
}

public double getLength() {
    return length;
}

public void setLength(double length) {
    this.length = length;
}

public double getWidth() {
    return width;
}

public void setWidth(double width) {
    this.width = width;
}

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

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

@Override
public String toString() {
    return super.toString() + ", Length: " + length + ", Width: " + width + ", Area: " +
calculateArea() + ", Perimeter: " + calculatePerimeter();
}
}

```

```

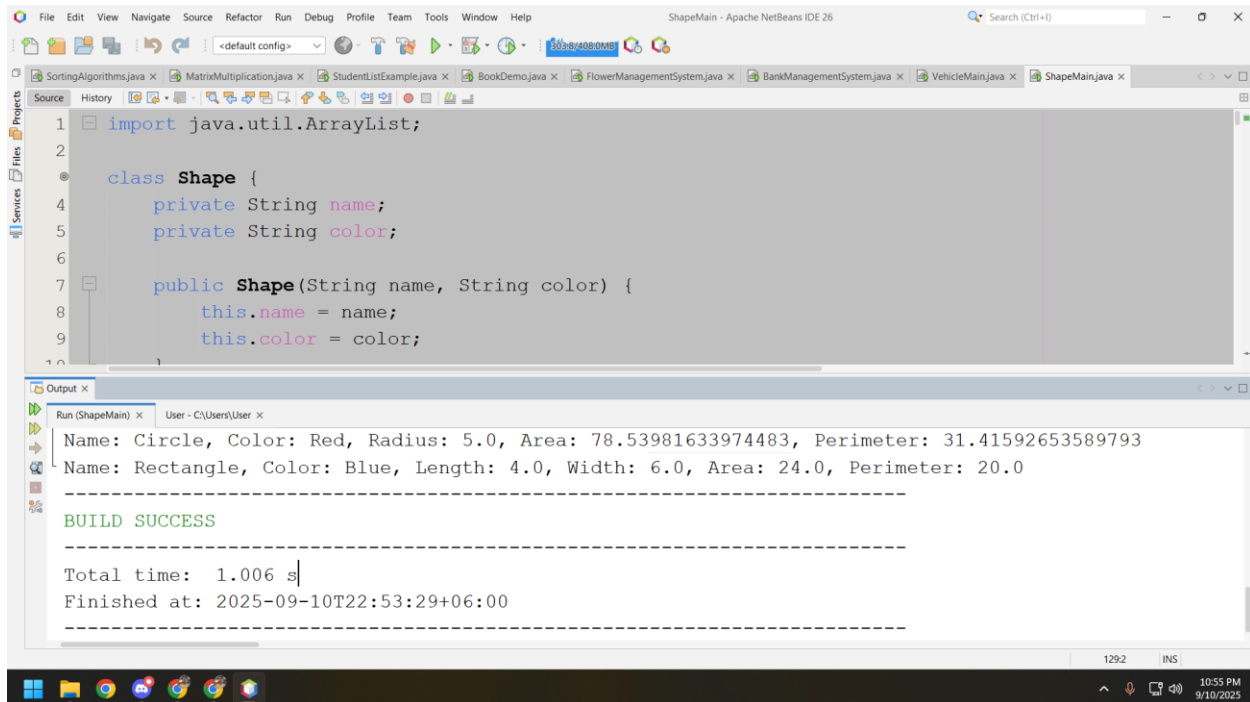
public class ShapeMain {
    public static void main(String[] args) {
        ArrayList<Shape> shapes = new ArrayList<>();
        Circle circle = new Circle("Circle", "Red", 5.0);
        Rectangle rectangle = new Rectangle("Rectangle", "Blue", 4.0, 6.0);

        shapes.add(circle);
        shapes.add(rectangle);

        for (Shape shape : shapes) {
            System.out.println(shape.toString());
        }
    }
}

```

Output:



```

import java.util.ArrayList;

class Shape {
    private String name;
    private String color;

    public Shape(String name, String color) {
        this.name = name;
        this.color = color;
    }
}

public class ShapeMain {
    public static void main(String[] args) {
        ArrayList<Shape> shapes = new ArrayList<>();
        Circle circle = new Circle("Circle", "Red", 5.0);
        Rectangle rectangle = new Rectangle("Rectangle", "Blue", 4.0, 6.0);

        shapes.add(circle);
        shapes.add(rectangle);

        for (Shape shape : shapes) {
            System.out.println(shape.toString());
        }
    }
}

```

Output:

```

Name: Circle, Color: Red, Radius: 5.0, Area: 78.53981633974483, Perimeter: 31.41592653589793
Name: Rectangle, Color: Blue, Length: 4.0, Width: 6.0, Area: 24.0, Perimeter: 20.0
BUILD SUCCESS
Total time: 1.006 s
Finished at: 2025-09-10T22:53:29+06:00

```

Figure 2: Output

3.

Problem Analysis:

This program demonstrates organizations often need to manage different categories of employees with varying salary structures. A base class Employee holds common attributes such as name, ID, and salary, and provides a method to calculate total salary. The subclasses Manager and Engineer extend Employee and override the salary calculation method. A manager's salary includes a bonus percentage, while an engineer's salary includes overtime pay. Using an ArrayList, the system stores both types of employees and uses polymorphism to process and display their details dynamically.

Background Theory:

Encapsulation: Employee attributes are private and accessed through getters and setters.

Inheritance: Manager and Engineer inherit from the Employee class to reuse common properties and methods.

Polymorphism: Different salary calculations are executed based on the object type (Manager or Engineer) at runtime.

Method Overriding: Both subclasses override calculateTotalSalary() and toString() to provide specialized implementations.

ArrayList: Used to manage a collection of employees, allowing easy storage and iteration regardless of employee type.

Algorithm Design:

1. Define Base Class – Employee
 - Attributes: name, id, salary.
 - Method: calculateTotalSalary() returns base salary.
 - Override toString() to print basic details.
2. Define Subclass – Manager
 - Additional attribute: bonusPercentage.
 - Override calculateTotalSalary() to include salary + bonus.
 - Override toString() to display bonus and total salary.
3. Define Subclass – Engineer
 - Additional attribute: overtimePay.

- Override calculateTotalSalary() to include salary + overtime.
 - Override toString() to display overtime and total salary.
4. Main Class – EmployeeMain
- Create an ArrayList<Employee>.
 - Add a Manager and an Engineer.
 - Iterate through the list and print details of each employee using polymorphism.

Code:

```
import java.util.ArrayList;

class Employee {
    private String name;
    private int id;
    private double salary;

    public Employee(String name, int id, double salary) {
        this.name = name;
        this.id = id;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getId() {
        return id;
    }

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

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }
}
```

```

    }

    public double calculateTotalSalary() {
        return salary;
    }

    @Override
    public String toString() {
        return "Name: " + name + ", ID: " + id + ", Base Salary: " + salary;
    }
}

class Manager extends Employee {
    private double bonusPercentage;

    public Manager(String name, int id, double salary, double bonusPercentage) {
        super(name, id, salary);
        this.bonusPercentage = bonusPercentage;
    }

    public double getBonusPercentage() {
        return bonusPercentage;
    }

    public void setBonusPercentage(double bonusPercentage) {
        this.bonusPercentage = bonusPercentage;
    }

    @Override
    public double calculateTotalSalary() {
        double bonus = getSalary() * bonusPercentage / 100;
        return getSalary() + bonus;
    }

    @Override
    public String toString() {
        return super.toString() + ", Bonus Percentage: " + bonusPercentage + "%, Total Salary: " +
        calculateTotalSalary();
    }
}

class Engineer extends Employee {
    private double overtimePay;

```

```

public Engineer(String name, int id, double salary, double overtimePay) {
    super(name, id, salary);
    this.overtimePay = overtimePay;
}

public double getOvertimePay() {
    return overtimePay;
}

public void setOvertimePay(double overtimePay) {
    this.overtimePay = overtimePay;
}

@Override
public double calculateTotalSalary() {
    return getSalary() + overtimePay;
}

@Override
public String toString() {
    return super.toString() + ", Overtime Pay: " + overtimePay + ", Total Salary: " +
calculateTotalSalary();
}
}

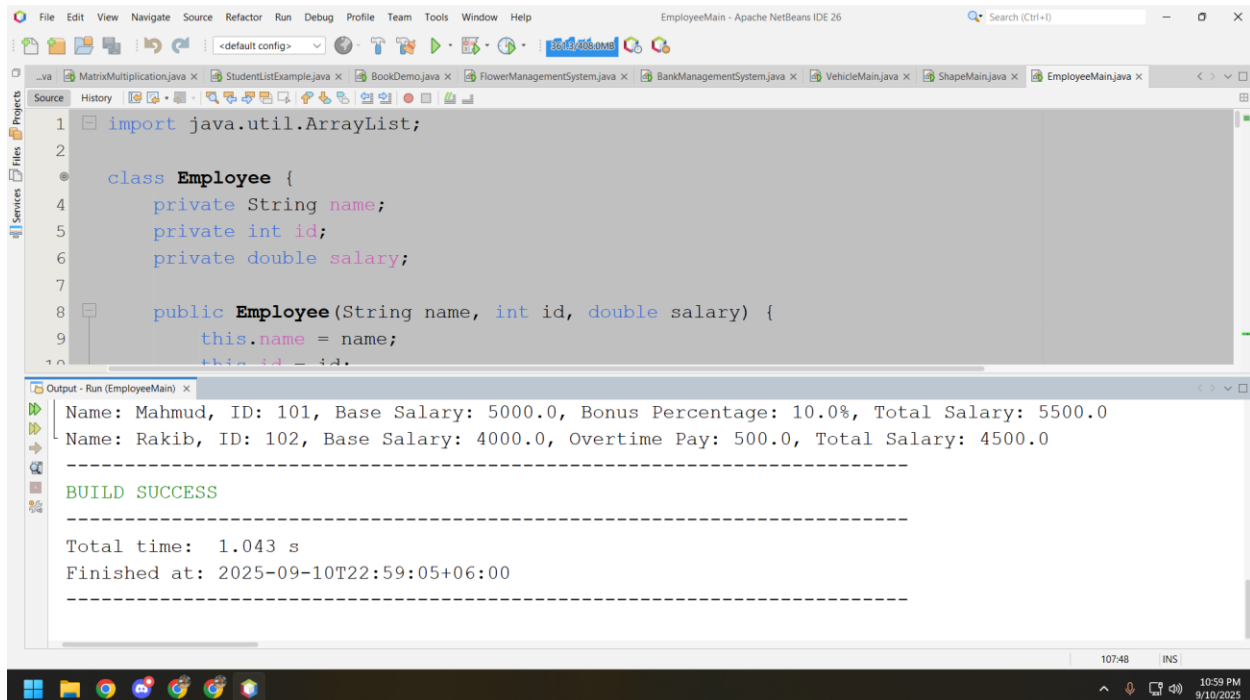
public class EmployeeMain {
    public static void main(String[] args) {
        ArrayList<Employee> employees = new ArrayList<>();
        Manager manager = new Manager("John", 101, 5000, 10);
        Engineer engineer = new Engineer("Alice", 102, 4000, 500);

        employees.add(manager);
        employees.add(engineer);

        for (Employee employee : employees) {
            System.out.println(employee.toString());
        }
    }
}

```


Output:



The screenshot shows the Apache NetBeans IDE interface. The main editor window displays a Java class named `Employee` with the following code:

```
1 import java.util.ArrayList;
2
3 class Employee {
4     private String name;
5     private int id;
6     private double salary;
7
8     public Employee(String name, int id, double salary) {
9         this.name = name;
10        this.id = id;
11    }
12 }
```

The output window at the bottom shows the results of running the program:

```
Name: Mahmud, ID: 101, Base Salary: 5000.0, Bonus Percentage: 10.0%, Total Salary: 5500.0
Name: Rakib, ID: 102, Base Salary: 4000.0, Overtime Pay: 500.0, Total Salary: 4500.0
-----
BUILD SUCCESS
-----
Total time: 1.043 s
Finished at: 2025-09-10T22:59:05+06:00
-----
```

Figure 3: Output

4.

Problem Analysis:

This program demonstrates the use of Bookstores and libraries often manage books of different categories, where each book shares common properties like title, author, and price but also has category-specific details. To solve this, a base class `Book` defines the general attributes and methods. Subclasses `FictionBook` and `NonFictionBook` extend `Book`, adding attributes such as genre and subject. Both subclasses override `displayDetails()` and `toString()` to include their specific information. An `ArrayList` is used to store different book types, and polymorphism ensures that each object's details are displayed correctly at runtime.

Background Theory:

Encapsulation: Attributes of Book (title, author, price) are private and accessed via getters/setters.

Inheritance: FictionBook and NonFictionBook inherit common attributes and methods from Book.

Polymorphism: Subclasses override methods, allowing the same method call to produce different outputs depending on the object type.

Method Overriding: displayDetails() and toString() are redefined in subclasses to extend functionality.

ArrayList: Used to dynamically store a collection of different types of books, supporting runtime flexibility.

Algorithm Design:

1. Define Base Class – Book
 - Attributes: title, author, price.
 - Methods: constructor, getters/setters, displayDetails(), toString().
2. Define Subclass – FictionBook
 - Additional attribute: genre.
 - Override displayDetails() to include genre.
 - Override toString() to append genre information.
3. Define Subclass – NonFictionBook
 - Additional attribute: subject.
 - Override displayDetails() to include subject.
 - Override toString() to append subject information.
4. Main Class – BookMain
 - Create an ArrayList<Book>.
 - Add FictionBook and NonFictionBook objects.
 - Use a for-each loop to iterate and print each book's details using polymorphism.

Code:

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

```
class Book {  
    private String title;  
    private String author;  
    private double price;
```

```

public Book(String title, String author, double price) {
    this.title = title;
    this.author = author;
    this.price = price;
}

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 double getPrice() {
    return price;
}

public void setPrice(double price) {
    this.price = price;
}

public void displayDetails() {
    System.out.println("Title: " + title + ", Author: " + author + ", Price: " + price);
}

@Override
public String toString() {
    return "Title: " + title + ", Author: " + author + ", Price: " + price;
}
}

class FictionBook extends Book {
    private String genre;

    public FictionBook(String title, String author, double price, String genre) {

```

```

        super(title, author, price);
        this.genre = genre;
    }

    public String getGenre() {
        return genre;
    }

    public void setGenre(String genre) {
        this.genre = genre;
    }

    @Override
    public void displayDetails() {
        super.displayDetails();
        System.out.println("Genre: " + genre);
    }

    @Override
    public String toString() {
        return super.toString() + ", Genre: " + genre;
    }
}

class NonFictionBook extends Book {
    private String subject;

    public NonFictionBook(String title, String author, double price, String subject) {
        super(title, author, price);
        this.subject = subject;
    }

    public String getSubject() {
        return subject;
    }

    public void setSubject(String subject) {
        this.subject = subject;
    }

    @Override
    public void displayDetails() {
        super.displayDetails();
        System.out.println("Subject: " + subject);
    }
}

```

```

    }

    @Override
    public String toString() {
        return super.toString() + ", Subject: " + subject;
    }
}

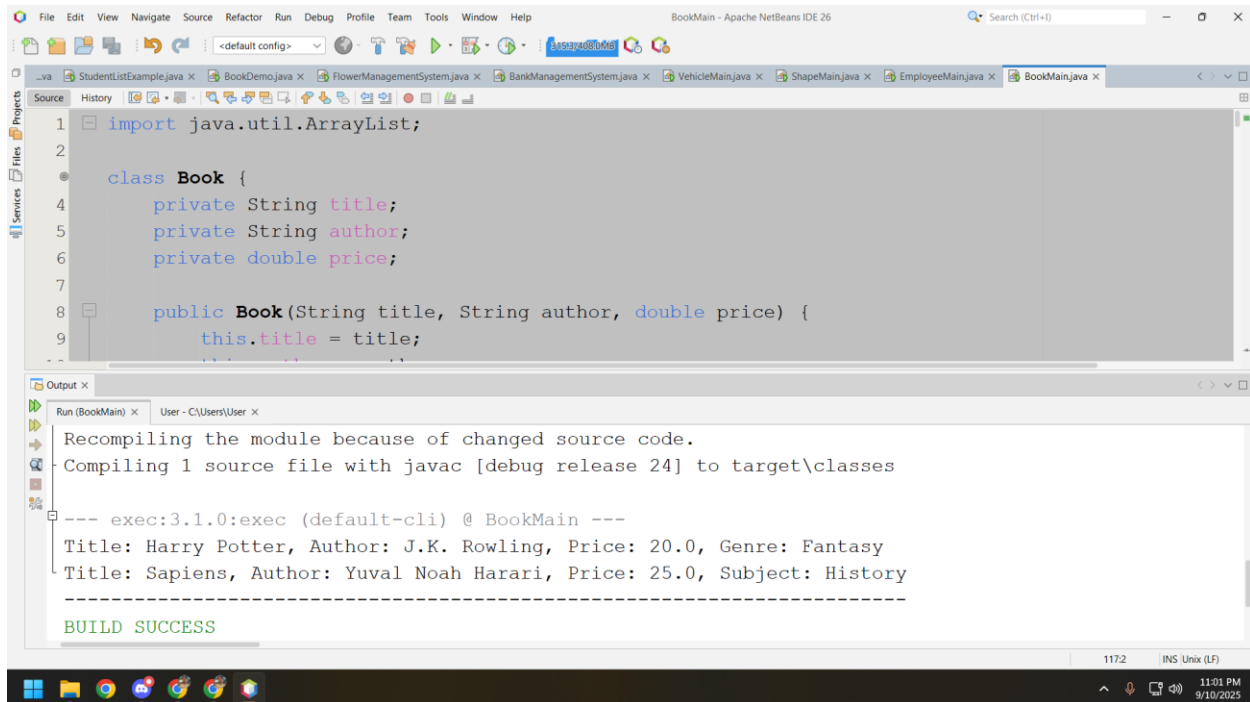
public class BookMain {
    public static void main(String[] args) {
        ArrayList<Book> books = new ArrayList<>();
        FictionBook fictionBook = new FictionBook("Harry Potter", "J.K. Rowling", 20.0,
"Fantasy");
        NonFictionBook nonFictionBook = new NonFictionBook("Sapiens", "Yuval Noah Harari",
25.0, "History");

        books.add(fictionBook);
        books.add(nonFictionBook);

        for (Book book : books) {
            System.out.println(book.toString());
        }
    }
}

```

Output:



The screenshot displays the Apache NetBeans IDE interface. The main editor window shows a Java file named `BookMain.java` with the following code:

```
1 import java.util.ArrayList;
2
3
4 class Book {
5     private String title;
6     private String author;
7     private double price;
8
9     public Book(String title, String author, double price) {
10         this.title = title;
11     }
12 }
```

The Output window at the bottom shows the following text:

```
Run (BookMain) x User - C:\Users\User x
Recompiling the module because of changed source code.
Compiling 1 source file with javac [debug release 24] to target\classes
--- exec:3.1.0:exec (default-cli) @ BookMain ---
Title: Harry Potter, Author: J.K. Rowling, Price: 20.0, Genre: Fantasy
Title: Sapiens, Author: Yuval Noah Harari, Price: 25.0, Subject: History
-----
BUILD SUCCESS
```

The status bar at the bottom right indicates the file encoding is UTF-8 and the current line is 1172.

Figure 4: Output