Learner Assignment Submission Format

Learner Details

- **Name: Adnan Sameer**
- **Enrollment Number:-----**
- **Batch / Class:-----**
- **Assignment: (Bridge Course Day 6)**
- **Date of Submission: 1-07-2025**

Problem Solving Problem 1.1

1. Employee Hierarchy

Create a base class Employee with:
Subclass Manager:
Subclass Developer:

---

2. Algorithm

1. Define a base class Employee with attributes name and id.

2. Create a constructor Employee that initializes name and id.

3. Define a method showDetails() to display employee details.

4. Create subclasses Manager and Developer that extend the Employee class.

5. In the subclasses, define constructors that call the superclass constructor using super.

6. Define a method showRole() in each subclass to display the role of the employee.

7. In the main method:

   - Create instances of Manager and Developer classes.

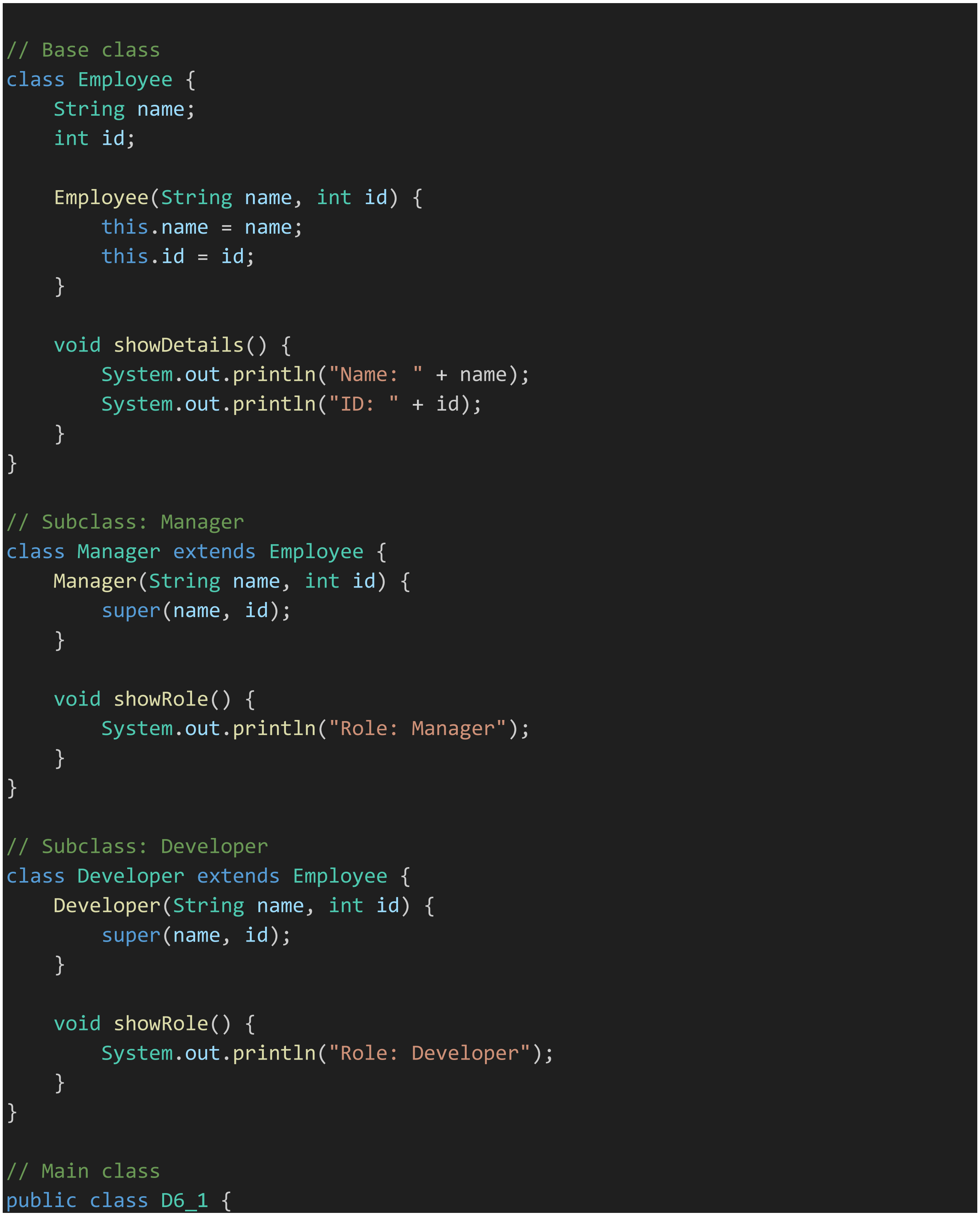
   - Call showDetails() and showRole() methods for each instance.

---

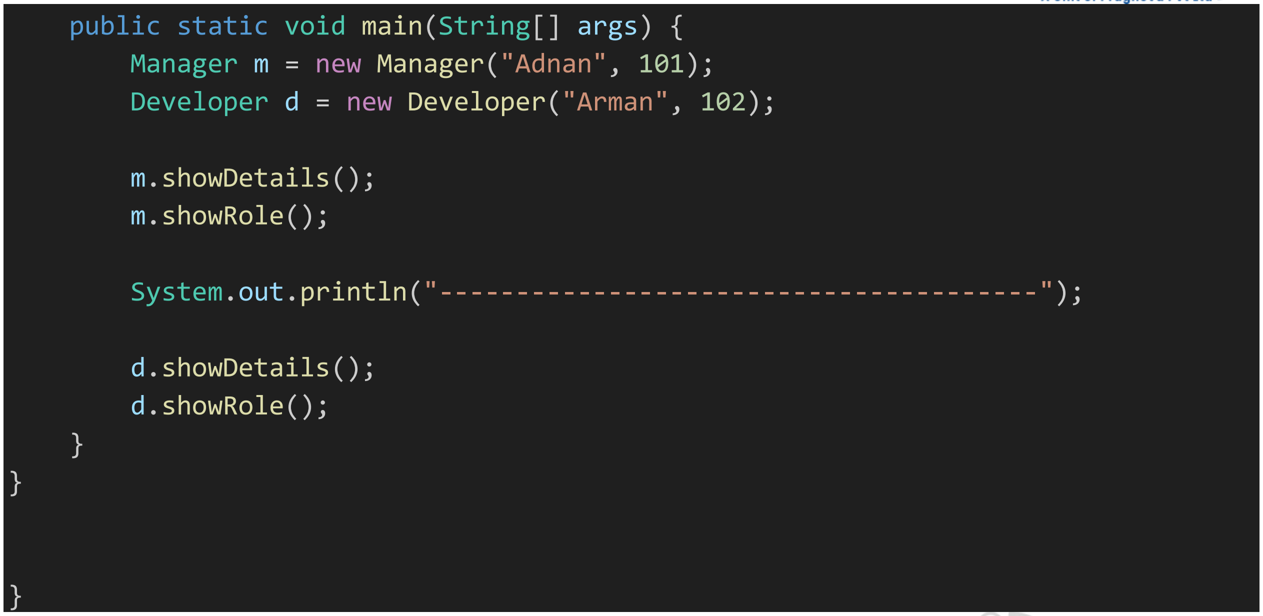
3. Pseudocode

```
CLASS Employee
  ATTRIBUTES
    name
    id
  CONSTRUCTOR Employee(name, id)
    INITIALIZE name, id
  METHOD showDetails()
    PRINT "Name: " + name
    PRINT "ID: " + id
CLASS Manager EXTENDS Employee
  CONSTRUCTOR Manager(name, id)
    CALL super(name, id)
  METHOD showRole()
    PRINT "Role: Manager"
CLASS Developer EXTENDS Employee
  CONSTRUCTOR Developer(name, id)
    CALL super(name, id)
  METHOD showRole()
    PRINT "Role: Developer"
MAIN
  CREATE Manager m WITH name = "Adnan", id = 101
  CREATE Developer d WITH name = "Arman", id = 102
  CALL m.showDetails()
  CALL m.showRole()
  PRINT "----------------------------------------"
  CALL d.showDetails()
  CALL d.showRole()
```

4. Program Code



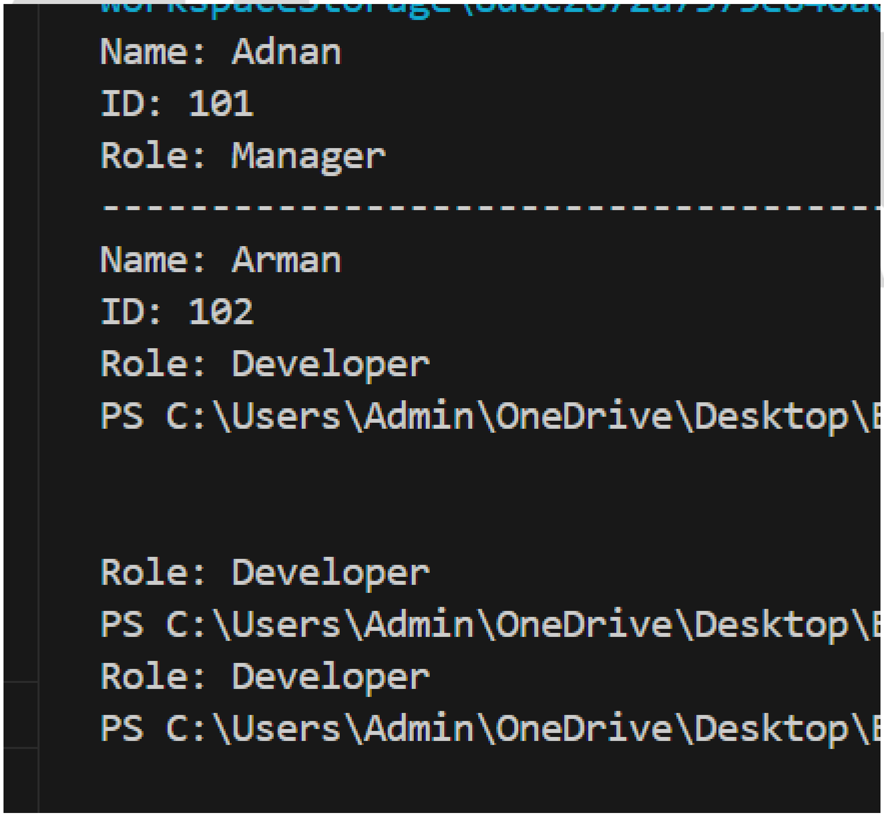
```
// Base class
class Employee {
    String name;
    int id;

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

    void showDetails() {
        System.out.println("Name: " + name);
        System.out.println("ID: " + id);
    }
}

// Subclass: Manager
class Manager extends Employee {
    Manager(String name, int id) {
        super(name, id);
    }

    void showRole() {
        System.out.println("Role: Manager");
    }
}

// Subclass: Developer
class Developer extends Employee {
    Developer(String name, int id) {
        super(name, id);
    }

    void showRole() {
        System.out.println("Role: Developer");
    }
}

// Main class
public class D6_1 {
```



```
    public static void main(String[] args) {
        Manager m = new Manager("Adnan", 101);
        Developer d = new Developer("Arman", 102);

        m.showDetails();
        m.showRole();

        System.out.println("----------------------------------------");

        d.showDetails();
        d.showRole();
    }
}


}
```

---

5. Screenshots of Output



```
workspaceStorage\6a0e2872a7597ce40a
Name: Adnan
ID: 101
Role: Manager
----------------------------------------
Name: Arman
ID: 102
Role: Developer
PS C:\Users\Admin\OneDrive\Desktop\E


Role: Developer
PS C:\Users\Admin\OneDrive\Desktop\E
Role: Developer
PS C:\Users\Admin\OneDrive\Desktop\E
```

Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	Adnan 101 Arman 102	Name: Adnan ID: 101 Role: Manager -------------------------- Name: Arman ID: 102 Role: Developer	Name: Adnan ID: 101 Role: Manager ----------------- Name: Arman ID: 102 Role: Developer	pass

6. Observation / Reflection

Inheritance: The subclasses Manager and Developer inherit the attributes and methods of the Employee class. Code Reusability: The code promotes code reusability by defining common attributes and methods in the Employee class.

Problem Solving Problem 1.2

1. Animal Kingdom

Base class(Super class): Animal with method makeSound()

Subclasses: Dog and Cat, override the method Create and test objects

2. Algorithm

1. Define a base class Animal with a method makeSound().

2. Create subclasses Dog and Cat that extend the Animal class.

3. Override the makeSound() method in the Dog and Cat classes with their specific implementations.

4. Create objects of the Animal, Dog, and Cat classes, using polymorphism to treat Dog and Cat objects as Animal objects.

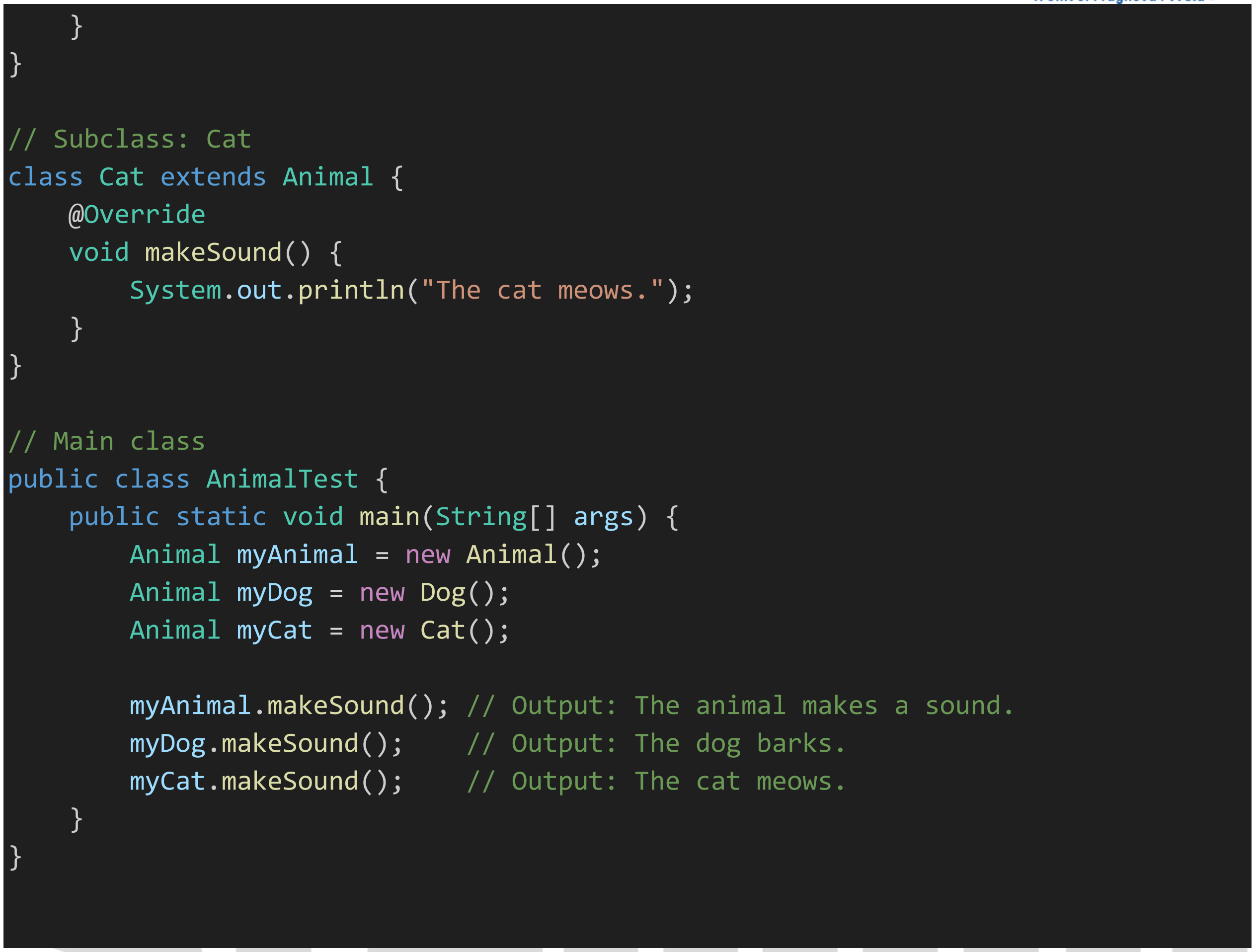
5. Call the makeSound() method on each object to demonstrate polymorphism.

3. Pseudocode

```
CLASS Animal
    METHOD makeSound()
        PRINT "The animal makes a sound."
CLASS Dog EXTENDS Animal
    METHOD makeSound()
        PRINT "The dog barks."
CLASS Cat EXTENDS Animal
    METHOD makeSound()
        PRINT "The cat meows."
MAIN
    CREATE Animal myAnimal = new Animal()
    CREATE Animal myDog = new Dog()
    CREATE Animal myCat = new Cat()
    CALL myAnimal.makeSound()
    CALL myDog.makeSound()
    CALL myCat.makeSound()
```

4. Program Code

```java
// Base class: Animal
class Animal {
    void makeSound() {
        System.out.println("The animal makes a sound.");
    }
}

// Subclass: Dog
class Dog extends Animal {
    @Override
    void makeSound() {
        System.out.println("The dog barks.");
```

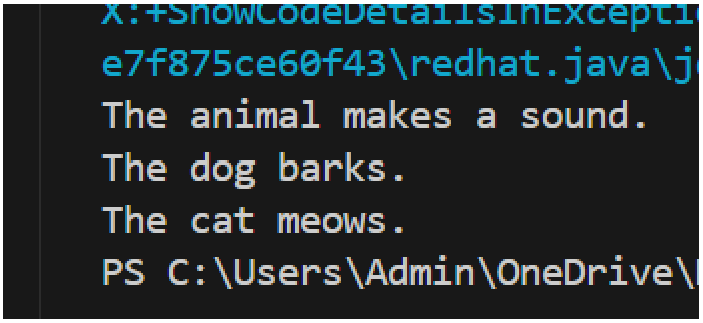


```
    }
}

// Subclass: Cat
class Cat extends Animal {
    @Override
    void makeSound() {
        System.out.println("The cat meows.");
    }
}

// Main class
public class AnimalTest {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();
        Animal myDog = new Dog();
        Animal myCat = new Cat();

        myAnimal.makeSound(); // Output: The animal makes a sound.
        myDog.makeSound();    // Output: The dog barks.
        myCat.makeSound();    // Output: The cat meows.
    }
}
```

5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	The animal makes sound	The animal makes sound	The animal makes sound	pass
2	The Dog Barks	The Dog Barks	The Dog Barks	pass
3	The cat meows	The cat meows	The cat meows	pass

6. Screenshots of Output



```
X:+ShowCodeDetailsInExceptio
e7f875ce60f43\redhat.java\j
The animal makes a sound.
The dog barks.
The cat meows.
PS C:\Users\Admin\OneDrive\
```

7. Observation / Reflection

Learnt Method Overriding and implementation of the Method Overriding

---

Problem Solving Problem 1.3

1. Design an Inheritance Tree

Base: ElectronicDevice Subclasses: Television, Laptop, Smartphone
List attributes and methods per subclass

---

2. Algorithm

1. Define a base class ElectronicDevice with methods turnOn() and turnOff().

2. Create subclasses Television, Laptop, and Smartphone that extend the ElectronicDevice class.

3. Each subclass has its specific method: changeChannel() for Television, runProgram() for Laptop, and makeCall() for Smartphone.

4. In the main method:

   - Create objects of each subclass (Television, Laptop, Smartphone).

   - Call the turnOn() method and the specific method for each object.

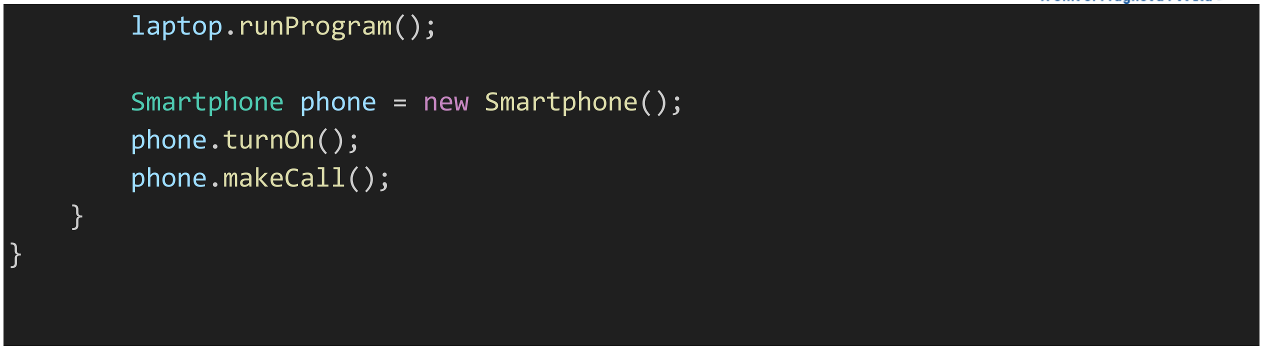
---

3. Pseudocode

```
CLASS ElectronicDevice
    METHOD turnOn()
        PRINT "Device is ON"
    METHOD turnOff()
        PRINT "Device is OFF"
CLASS Television EXTENDS ElectronicDevice
    METHOD changeChannel()
        PRINT "Changing TV channel"
CLASS Laptop EXTENDS ElectronicDevice
    METHOD runProgram()
        PRINT "Running a program on Laptop"
CLASS Smartphone EXTENDS ElectronicDevice
    METHOD makeCall()
        PRINT "Making a call from Smartphone"
MAIN
    CREATE Television tv
    CALL tv.turnOn()
    CALL tv.changeChannel()
    CREATE Laptop laptop
    CALL laptop.turnOn()
    CALL laptop.runProgram()
    CREATE Smartphone phone
    CALL phone.turnOn()
    CALL phone.makeCall()
```

4. Program Code

```
// Base class
class ElectronicDevice {
    void turnOn() {
        System.out.println("Device is ON");
    }
    void turnOff() {
        System.out.println("Device is OFF");
    }
}

// Subclass: Television
class Television extends ElectronicDevice {
    void changeChannel() {
        System.out.println("Changing the TV channel");
    }
}

// Subclass: Laptop
class Laptop extends ElectronicDevice {
    void runProgram() {
        System.out.println("Running  program on Laptop");
    }
}

// Subclass: Smartphone
class Smartphone extends ElectronicDevice {
    void makeCall() {
        System.out.println("Making a call from phone");
    }
}

// Main class
public class InheritanceTree {
    public static void main(String[] args) {
        Television tv = new Television();
        tv.turnOn();
        tv.changeChannel();

        Laptop laptop = new Laptop();
        laptop.turnOn();
```

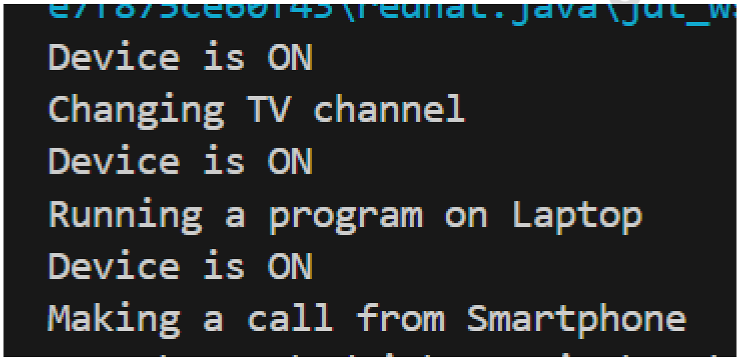


```
        laptop.runProgram();

        Smartphone phone = new Smartphone();
        phone.turnOn();
        phone.makeCall();
    }
}
```

5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	Device is on , off	Device is on , off	Device is on , off	pass
2	Changing Tv channel	Changing Tv channel	Changing Tv channel	pass
3	Making call from phone	Making call from phone	Making call from phone	pass

6. Screenshots of Output



```
e71873ce80f43\redhat.java\jdt_w
Device is ON
Changing TV channel
Device is ON
Running a program on Laptop
Device is ON
Making a call from Smartphone
```

7. Observation / Reflection

The subclasses inherit the turnOn() and turnOff() methods from the ElectronicDevice class.

Although not fully utilized in this example, the code sets the stage for polymorphic behavior by defining a common base class for different electronic devices

---

Problem Solving Problem 2.1:

1. Payment Gateway

Abstract class: PaymentGateway with abstract processPayment(double amount) Subclasses: CreditCardGateway, PayPalGateway Attempt to instantiate abstract class (should fail)

---

2. Algorithm

1. Define an abstract class Payment with an abstract method processPayment(double amount).

2. Create concrete subclasses CreditCard and Phonepe that extend the Payment class and implement the processPayment(double amount) method.

3. In the main method:

- Create objects of type CreditCard and Phonepe and assign them to variables of type Payment.

- Call the processPayment(double amount) method on these objects, demonstrating polymorphism.

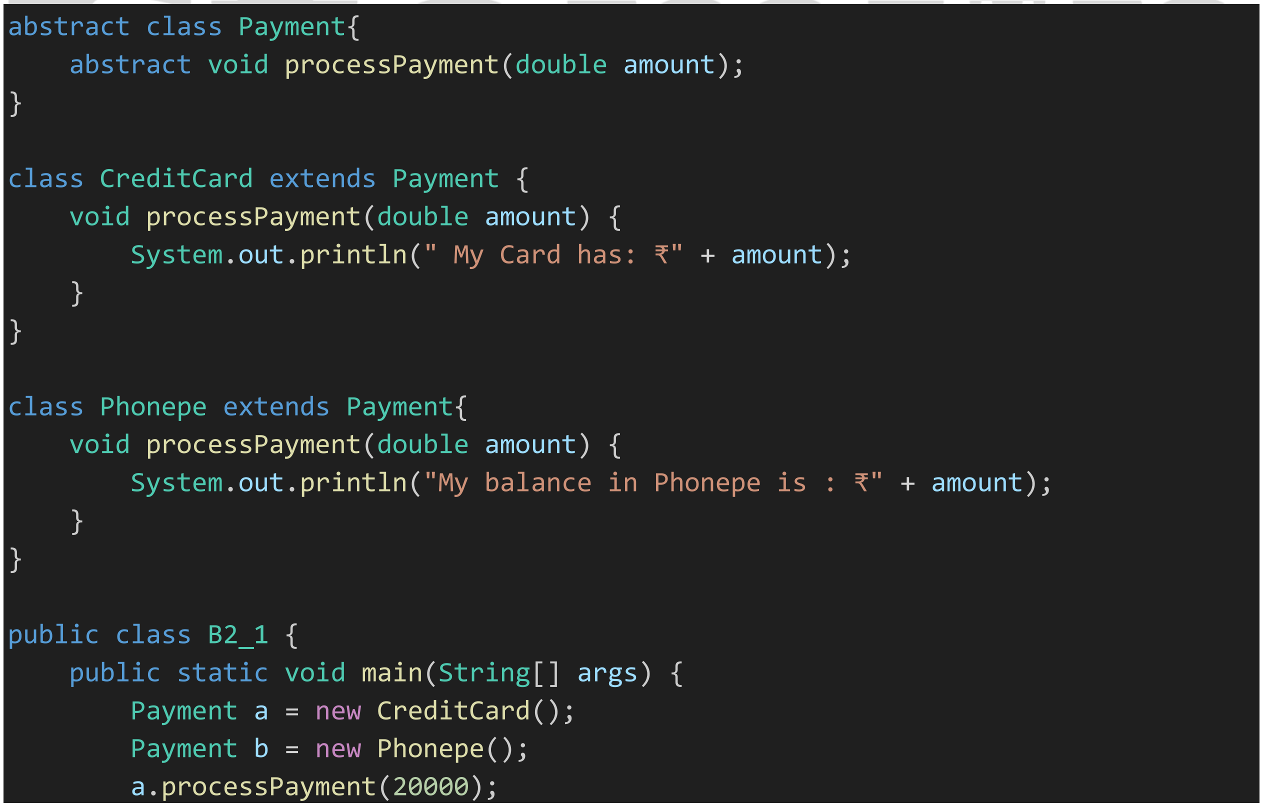
---

3. Pseudocode

```
ABSTRACT CLASS Payment
   ABSTRACT METHOD processPayment(amount)
```

```
CLASS CreditCard EXTENDS Payment
    METHOD processPayment(amount)
        PRINT "My Card has: " + amount


CLASS Phonepe EXTENDS Payment
    METHOD processPayment(amount)
        PRINT "My balance in Phonepe is : " + amount
MAIN
    CREATE Payment a = new CreditCard()
    CREATE Payment b = new Phonepe()
    CALL a.processPayment(20000)
    CALL b.processPayment(12000)
```

4. Program Code

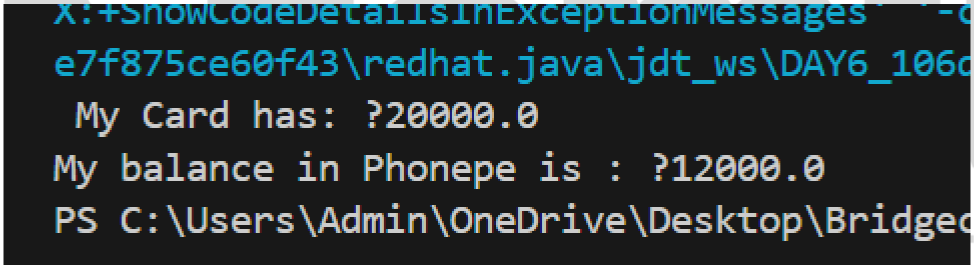


```java
abstract class Payment{
    abstract void processPayment(double amount);
}

class CreditCard extends Payment {
    void processPayment(double amount) {
        System.out.println(" My Card has: ₹" + amount);
    }
}

class Phonepe extends Payment{
    void processPayment(double amount) {
        System.out.println("My balance in Phonepe is : ₹" + amount);
    }
}

public class B2_1 {
    public static void main(String[] args) {
        Payment a = new CreditCard();
        Payment b = new Phonepe();
        a.processPayment(20000);
        b.processPayment(12000);

        // Payment p = new Payment(); // ❌ Error: abstract class cannot be
instantiated
    }
}
```

5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	20000	20000	20000	pass
2	12000	12000	12000	pass

6. Screenshots of Output



```
X:+ShowCodeDetailsInExceptionMessages' '-c
e7f875ce60f43\redhat.java\jdt_ws\DAY6_106c
 My Card has: ?20000.0
My balance in Phonepe is : ?12000.0
PS C:\Users\Admin\OneDrive\Desktop\Bridgec
```

7. Observation / Reflection

Inheritance , Inheritance , Code Reusability: The code promotes code reusability by defining a common interface (processPayment(double amount)) in the Payment class that can be shared between subclasses.

Problem Solving Problem 2.2

1. Instrument Sounds

Abstract class: Instrument with abstract play() Subclasses: Guitar, Piano Implement and test

---

2. Algorithm

1. Define an abstract class with an abstract method.

2. Create concrete subclasses that extend the abstract class and implement the abstract method.

3. In the main method:

   - Create objects of the subclasses and assign them to variables of the abstract class type.

   - Call the abstract method on these objects, demonstrating polymorphism.

---

3. Pseudocode

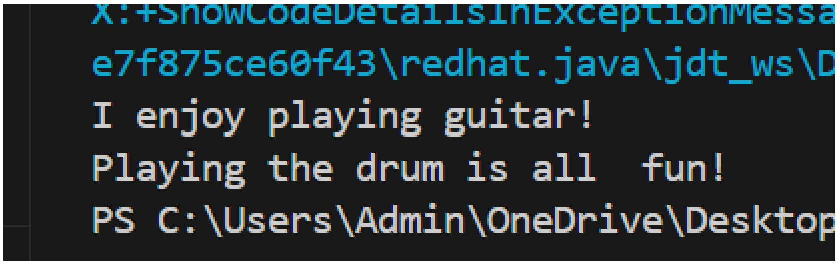
```
ABSTRACT CLASS Payment
    ABSTRACT METHOD processPayment(amount)


CLASS CreditCard EXTENDS Payment
    METHOD processPayment(amount)
        // Implementation specific to CreditCard


CLASS Phonepe EXTENDS Payment
    METHOD processPayment(amount)
        // Implementation specific to Phonepe
```

```
MAIN
    CREATE Payment a = new CreditCard()
    CREATE Payment b = new Phonepe()
    CALL a.processPayment(amount)
    CALL b.processPayment(amount)
```

4. Program Code

```java
abstract class Instrument {
    abstract void play();
}

class Guitar extends Instrument {
    void play() {
        System.out.println("I enjoy playing guitar!");
    }
}

class Drum extends Instrument {
    void play() {
        System.out.println("Playing the drum is all  fun!");
    }
}

public class  B2_2{
    public static void main(String[] args) {
        Instrument g = new Guitar();
        Instrument p = new Drum();
        g.play();
        p.play();
    }
}
```

5. Screenshots of Output



```
X:+ShowCodeDetailsInExceptionMessa
e7f875ce60f43\redhat.java\jdt_ws\D
I enjoy playing guitar!
Playing the drum is all  fun!
PS C:\Users\Admin\OneDrive\Desktop
```

6. Observation / Reflection

The code demonstrates object-oriented programming principles, specifically abstraction and polymorphism. By using an abstract class and subclasses, the code promotes code reusability and extensibility. The polymorphic behavior allows for flexible coding and easy addition of new payment types.

Problem Solving Problem 2.3

1. Abstracting a Task

Base: AutomatedTask, method execute() Subclasses: EmailSender, FileArchiver, DatabaseBackup Use abstraction to simplify the execution of tasks

2. Algorithm

1. Define an abstract class with an abstract method.

2. Create concrete subclasses that extend the abstract class and implement the abstract method.

3. In the main method:

   - Create objects of the subclasses and assign them to variables of the abstract class type.

   - Call the abstract method on these objects, demonstrating polymorphism. 4. In the main method:

   - Create a new D5Book object with specified title, author, and numPages.

   - Call displayInfo() to show the book's initial state.

- Call openBook() and closeBook() to demonstrate book state changes.
- Call displayInfo() again to show the final state.

---

3. Pseudocode

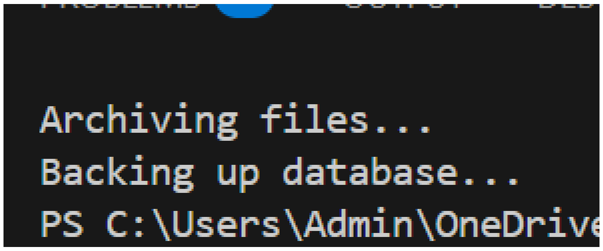
```
CLASS D5Book
  ATTRIBUTES
    title
    author
    numPages
    isOpen
  CONSTRUCTOR D5Book(title, author, numPages)
    INITIALIZE title, author, numPages
    SET isOpen TO true
  METHOD openBook()
    SET isOpen TO true
    PRINT "The book is now open."
  METHOD closeBook()
    SET isOpen TO false
    PRINT "The book is now closed."
  METHOD displayInfo()
    PRINT "Title: " + title
    PRINT "Author: " + author
    PRINT "Number of Pages: " + numPages
      PRINT "Is the book open? " + isOpen
MAIN
  CREATE D5Book myBook WITH title = "MY BOOK", author = "Sameer", numPages = 100
  CALL myBook.displayInfo()
```

```
    CALL myBook.openBook()
    CALL myBook.closeBook()
    CALL myBook.displayInfo()

ABSTRACT CLASS Automation
    ABSTRACT METHOD execute()

CLASS EmailSender EXTENDS Automation
    METHOD execute()
        PRINT "Sending emails..."

CLASS FileArchiver EXTENDS Automation
    METHOD execute()
        PRINT "Archiving files..."

CLASS DatabaseBackup EXTENDS Automation
    METHOD execute()
        PRINT "Backing up database..."

MAIN
    CREATE Automation email = new EmailSender()
    CREATE Automation archive = new FileArchiver()
    CREATE Automation backup = new DatabaseBackup()
    CALL email.execute()
    CALL archive.execute()
    CALL backup.execute()
```

4. Program Code

```java
abstract class Automation {
    abstract void execute();
}

class EmailSender extends Automation {
    void execute() {
        System.out.println("Sending emails...");
    }
}

class FileArchiver extends Automation  {
    void execute() {
        System.out.println("Archiving files...");
    }
}

class DatabaseBackup extends Automation {
    void execute() {
        System.out.println("Backing up database...");
    }
}

public class B2_3 {
    public static void main(String[] args) {
        Automation  email = new EmailSender();
        Automation archive = new FileArchiver();
        Automation backup = new DatabaseBackup();

        email.execute();
        archive.execute();
        backup.execute();
    }
}
```

6. Screenshots of Output



```
Archiving files...
Backing up database...
PS C:\Users\Admin\OneDrive
```

7. Observation / Reflection

The abstract class Automation provides a common interface for different automation tasks.The code showcases polymorphic behavior by treating objects of different classes as objects of a common superclass. The design allows for easy addition of new automation tasks by creating additional subclasses.

Problem Solving Problem 3_1

1. Employee Payroll

Base: Employee, abstract method calculatePayroll() Subclasses: SalariedEmployee, HourlyEmployee Implement payroll logic and process list of employees

2. Algorithm

1. Define an abstract class Employee with an abstract method calculatePayroll().

2. Create concrete subclasses SalariedEmployee and HourlyEmployee that extend the Employee class and implement the calculatePayroll() method.

3. In the main method:

   - Create an array of Employee objects containing instances of SalariedEmployee and HourlyEmployee.

   - Iterate through the array and call the calculatePayroll() method on each object, demonstrating polymorphism.

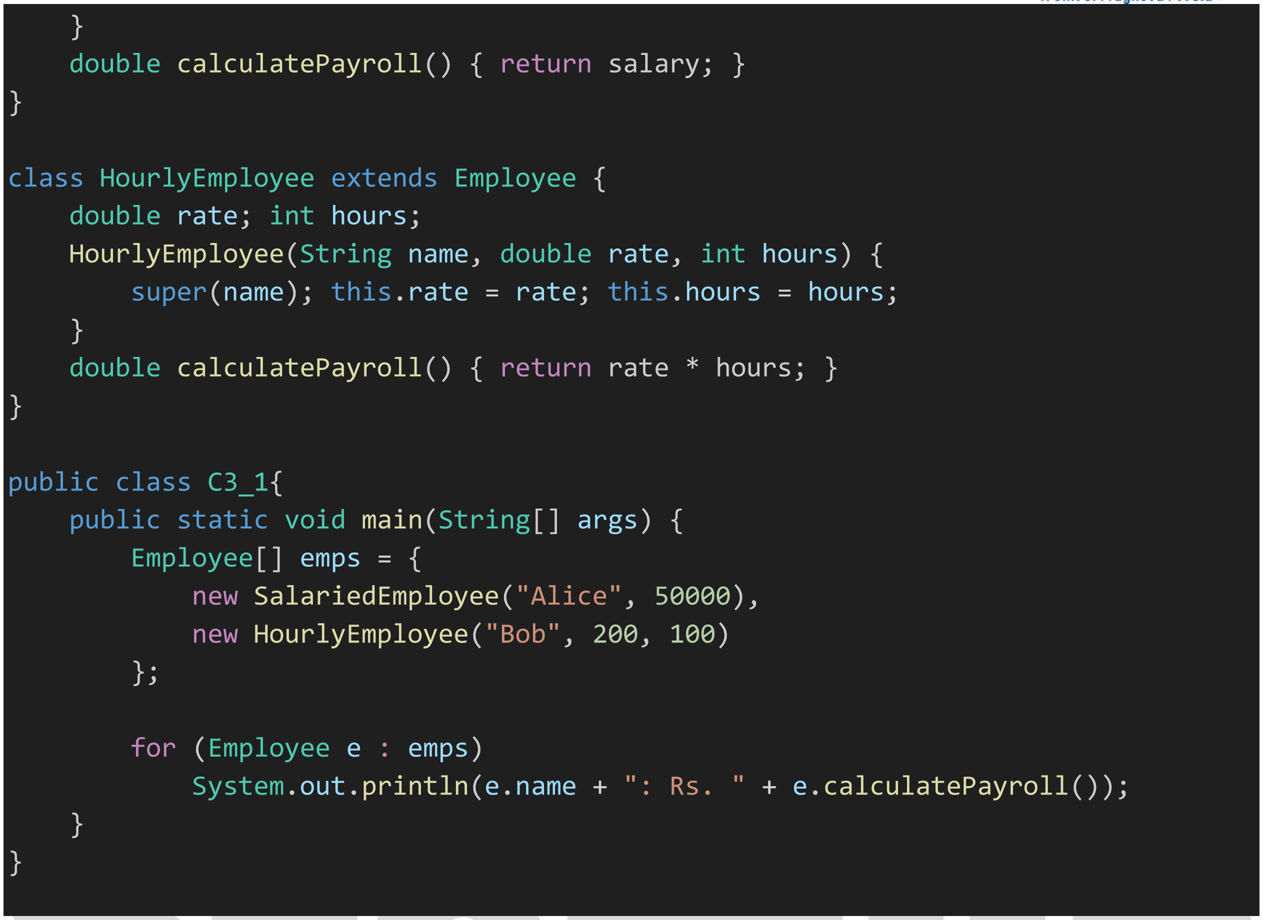
3. Pseudocode

```
ABSTRACT CLASS Employee
    METHOD calculatePayroll()
        // Abstract method

CLASS SalariedEmployee EXTENDS Employee
    METHOD calculatePayroll()
        RETURN salary

CLASS HourlyEmployee EXTENDS Employee
    METHOD calculatePayroll()
        RETURN rate * hours

MAIN
    CREATE Employee[] employees
    FOR EACH employee IN employees
        PRINT employee.name + ": " + employee.calculatePayroll()
```

---

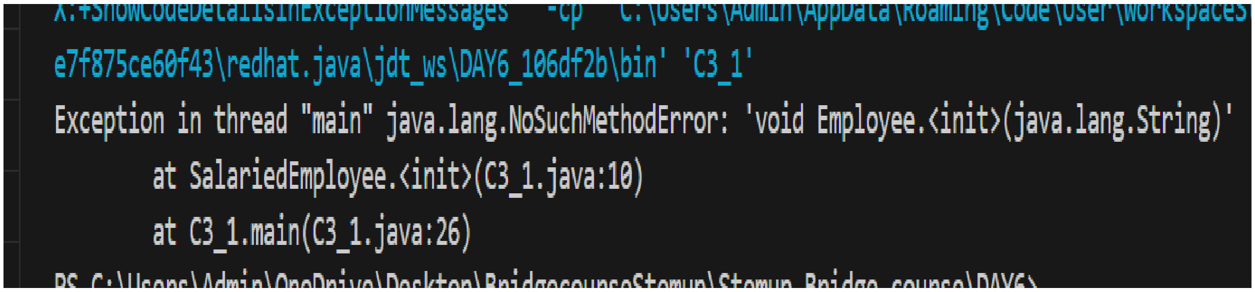
4. Program Code

```java
abstract class Employee {
    String name;
    Employee(String name) { this.name = name; }
    abstract double calculatePayroll();
}

class SalariedEmployee extends Employee {
    double salary;
    SalariedEmployee(String name, double salary) {
        super(name); this.salary = salary;
```



```
    }
    double calculatePayroll() { return salary; }
}

class HourlyEmployee extends Employee {
    double rate; int hours;
    HourlyEmployee(String name, double rate, int hours) {
        super(name); this.rate = rate; this.hours = hours;
    }
    double calculatePayroll() { return rate * hours; }
}

public class C3_1{
    public static void main(String[] args) {
        Employee[] emps = {
            new SalariedEmployee("Alice", 50000),
            new HourlyEmployee("Bob", 200, 100)
        };

        for (Employee e : emps)
            System.out.println(e.name + ": Rs. " + e.calculatePayroll());
    }
}
```

5. Screenshots of Output



```
x:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Admin\AppData\Roaming\Code\User\workspaces\
e7f875ce60f43\redhat.java\jdt_ws\DAY6_106df2b\bin' 'C3_1'
Exception in thread "main" java.lang.NoSuchMethodError: 'void Employee.<init>(java.lang.String)'
        at SalariedEmployee.<init>(C3_1.java:10)
        at C3_1.main(C3_1.java:26)
PS C:\Users\Admin\OneDrive\Desktop\BridgecourseStemup\Stemup_Bridge_course\DAY6>
```

6. Observation / Reflection

The code demonstrates object-oriented programming principles, specifically abstraction and polymorphism. By using an abstract class Employee and its subclasses, the code promotes code reusability and extensibility.

Problem Solving problem 3.2

1. Geometric Shapes

Abstract base: Shape with getArea() Subclasses: Circle, Square Create polymorphic list and calculate areas

2. Algorithm

1. Define an abstract base class Shape with an abstract method getArea().

2. Create concrete subclasses Circle and Square that extend the Shape class and implement the getArea() method.

3. Create a polymorphic list of Shape objects containing instances of Circle and Square.

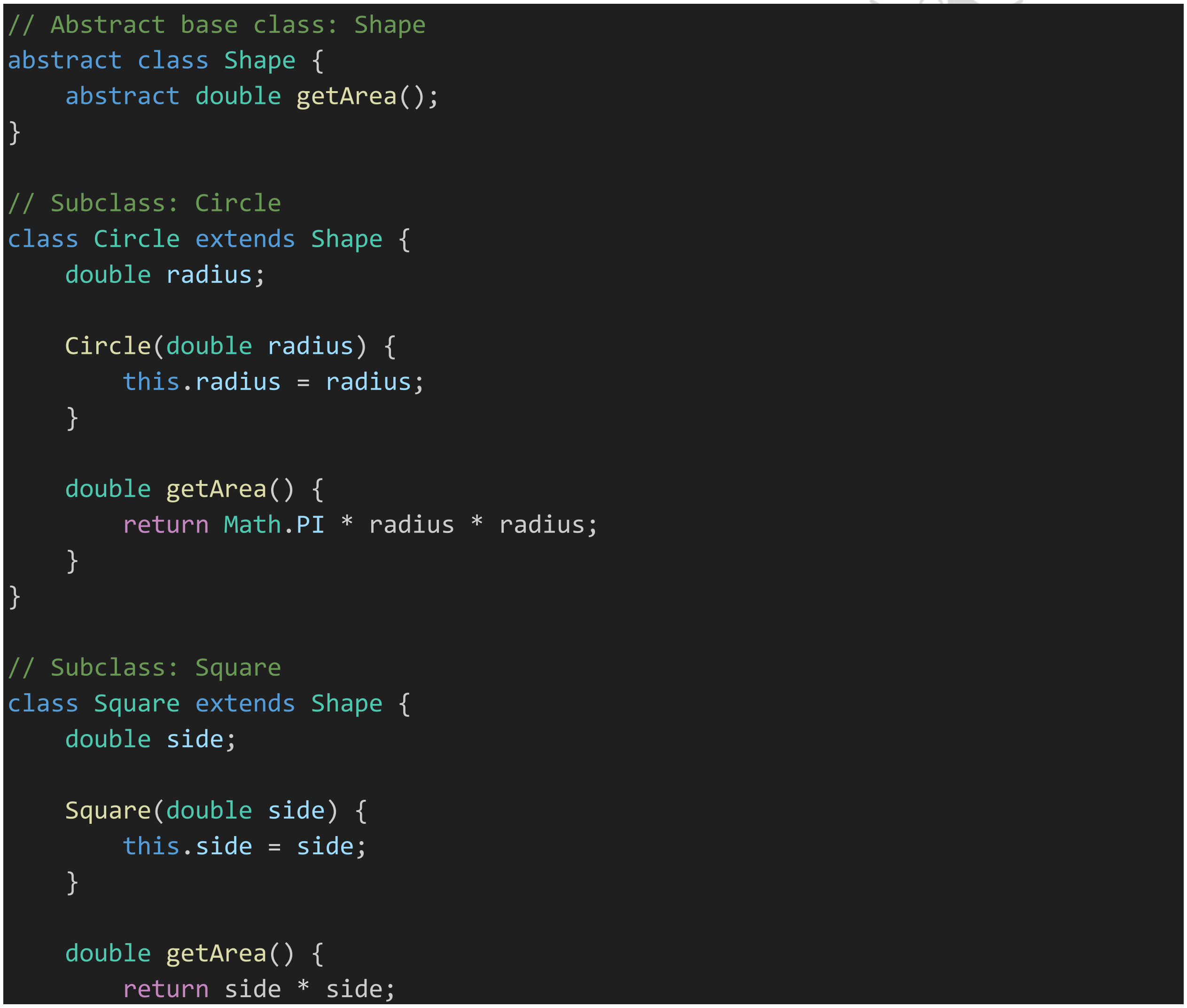
4. Iterate through the list and call the getArea() method on each object, demonstrating polymorphism.

3. Pseudocode

```
ABSTRACT CLASS Shape
    ABSTRACT METHOD getArea()
CLASS Circle EXTENDS Shape
    METHOD getArea()
```

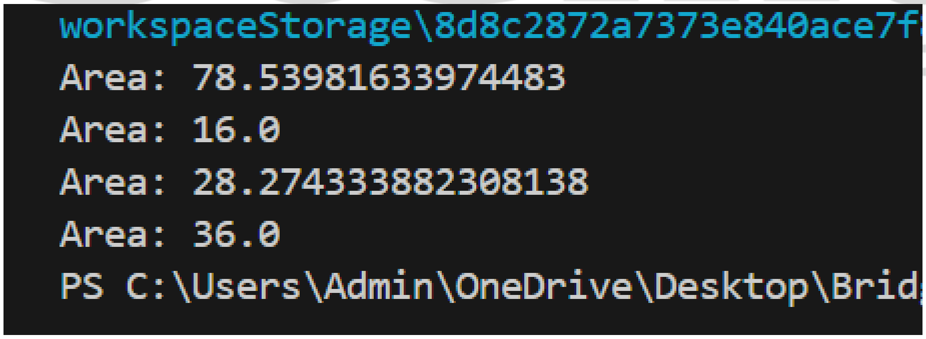
```
        RETURN π * radius * radius
CLASS Square EXTENDS Shape
    METHOD getArea()
        RETURN side * side
MAIN
    CREATE Shape[] shapes
    FOR EACH shape IN shapes
        PRINT "Area: " + shape.getArea()
```

4. Program Code



```java
// Abstract base class: Shape
abstract class Shape {
    abstract double getArea();
}

// Subclass: Circle
class Circle extends Shape {
    double radius;

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

    double getArea() {
        return Math.PI * radius * radius;
    }
}

// Subclass: Square
class Square extends Shape {
    double side;

    Square(double side) {
        this.side = side;
    }

    double getArea() {
        return side * side;
    }
}

public class C3_2 {
    public static void main(String[] args) {
        // Create a polymorphic list of shapes
        Shape[] shapes = new Shape[] {
            new Circle(5.0),
            new Square(4.0),
            new Circle(3.0),
            new Square(6.0)
        };

        // Calculate and print areas
        for (Shape shape : shapes) {
            System.out.println("Area: " + shape.getArea());
        }
    }
}
```

6. Screenshots of Output



```
workspaceStorage\8d8c2872a7373e840ace7f
Area: 78.53981633974483
Area: 16.0
Area: 28.274333882308138
Area: 36.0
PS C:\Users\Admin\OneDrive\Desktop\Brid
```

7. Observation / Reflection

The code demonstrates object-oriented programming principles, specifically abstraction and polymorphism. By using an abstract class Shape and its subclasses, the code promotes code reusability and extensibility.

Problem Solving Problem 3.3

1. Polymorphism in UI

Base: Tool, method draw() Subclasses: PenTool, EraserTool, LineTool Demonstrate polymorphism using a collection

2. Algorithm

1. Define an abstract class Shape with an abstract method getArea().

2. Create concrete subclasses Circle and Square that extend the Shape class and implement the getArea() method.

3. Create an array of Shape objects containing instances of Circle and Square.

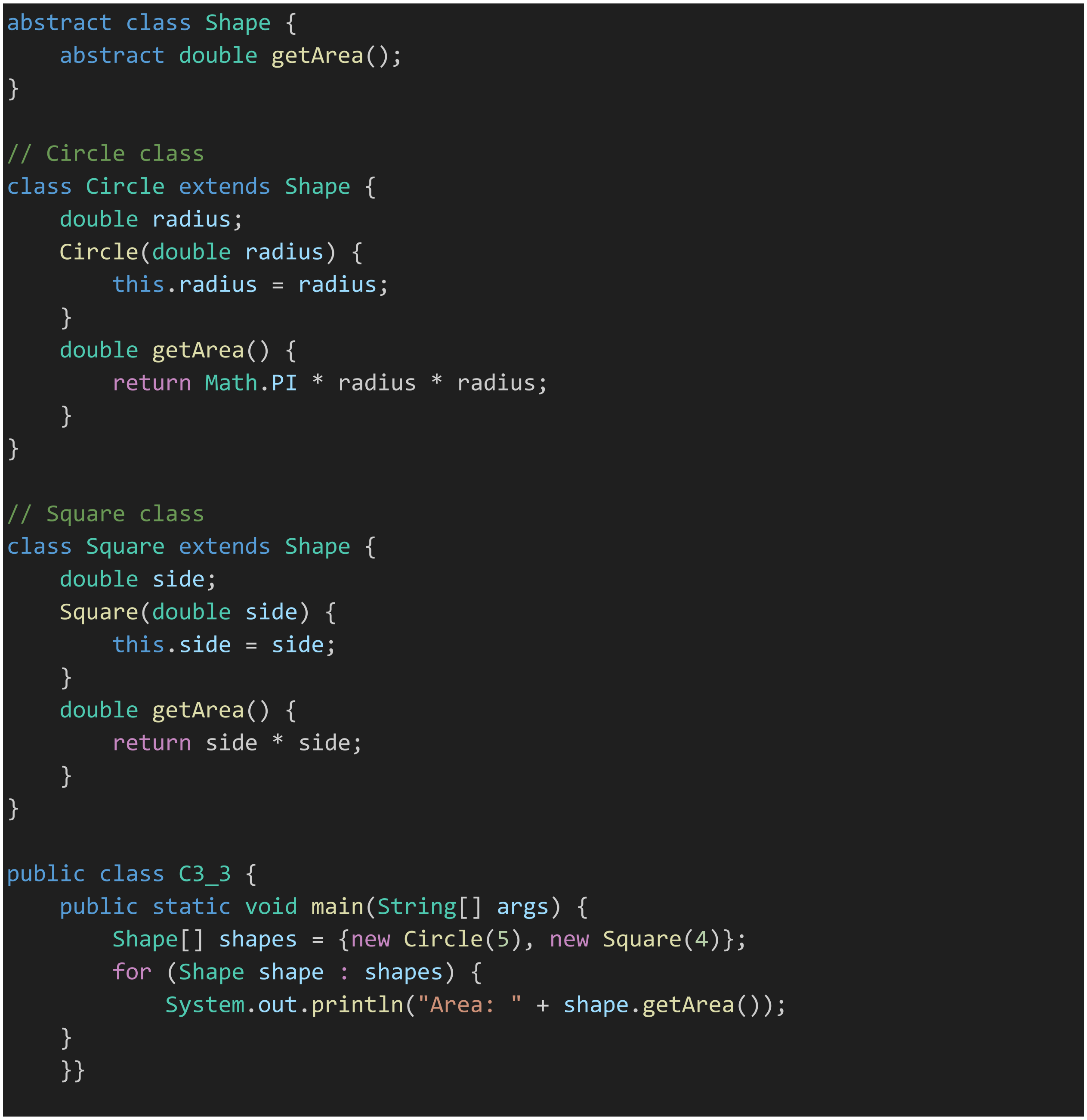
4. Iterate through the array and call the getArea() method on each object, demonstrating polymorphism.

3. Pseudocode

```
ABSTRACT CLASS Shape
    ABSTRACT METHOD getArea()
CLASS Circle EXTENDS Shape
    METHOD getArea()
        RETURN $\pi$ * radius * radius
CLASS Square EXTENDS Shape
    METHOD getArea()
        RETURN side * side
MAIN
    CREATE Shape[] shapes
```

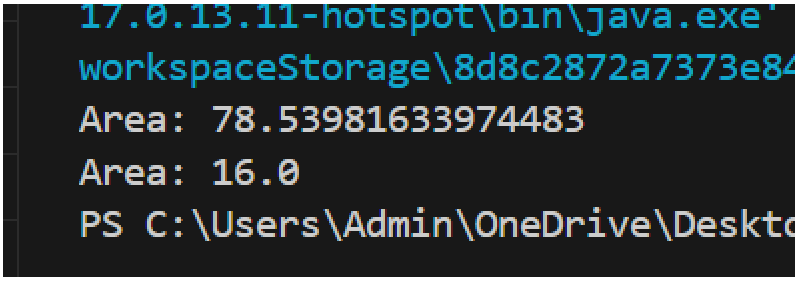
```
FOR EACH shape IN shapes
    PRINT "Area: " + shape.getArea()
```

4. Program Code



```java
abstract class Shape {
    abstract double getArea();
}

// Circle class
class Circle extends Shape {
    double radius;
    Circle(double radius) {
        this.radius = radius;
    }
    double getArea() {
        return Math.PI * radius * radius;
    }
}

// Square class
class Square extends Shape {
    double side;
    Square(double side) {
        this.side = side;
    }
    double getArea() {
        return side * side;
    }
}

public class C3_3 {
    public static void main(String[] args) {
        Shape[] shapes = {new Circle(5), new Square(4)};
        for (Shape shape : shapes) {
            System.out.println("Area: " + shape.getArea());
    }
    }}
```

6. Screenshots of Output



```
17.0.13.11-hotspot\bin\java.exe'
workspaceStorage\8d8c2872a7373e84
Area: 78.53981633974483
Area: 16.0
PS C:\Users\Admin\OneDrive\Deskto
```

7. Observation / Reflection

The code demonstrates object-oriented programming principles, specifically abstraction and polymorphism. By using an abstract class Shape and its subclasses, the code promotes code reusability and extensibility.