Muhammad Shaheer | FA24-BSE-104(B)

Object Oriented Programming

Sir Ameer Gillani

COMSATS UNIVERSITY ISLAMABAD, SAHIWAL

CLO-01: Demonstrate fundamental principles and concepts of object-oriented programming.

27 September 2025

ASSIGNMENT#01

1. **EXPLAIN** THE DIFFERENCE BETWEEN A CLASS AND AN OBJECT IN OBJECT-ORIENTED PROGRAMMING. PROVIDE AN EXAMPLE OF A CLASS AND HOW AN OBJECT IS CREATED FROM IT IN A PROGRAMMING LANGUAGE OF YOUR CHOICE.

No.	CLASS	OBJECT
1	Class is a blueprint or template.	Object is an actual instance created from the class.
2	Does not consume memory by itself.	Consumes memory when it is created.
3	Exists throughout program execution	Created and destroyed during runtime
4	Logical entity	Physical entity
5	Defined once using “class” keyword	Can create multiple objects from one class

Example:

```
class Student{ //Student class
    private String Name;  //Instance Variable
    private int rollNo;
    private int sem;

    public Student(String name, int rollNo, int sem) {
        this.name = name;
        this.rollNo = rollNo;
        this.sem = sem;
    }
}

public class Main {
    public static void main(String[] args) {

        Student st1 = new Student("Shaheer", 104, 3);
//Object Created and values passed

    }
}
```

2. **EXPLAIN** PASSING OBJECTS AND ARRAY OF OBJECTS FROM A METHOD IN JAVA. PROVIDE AN EXAMPLE OF BOTH IN THE FORM OF A CODE.

Passing Objects to Methods:

When we pass an object to a method we pass the reference copy not the original value. The method can change the object's properties but can't replace the entire object.

```
class Student {
    String name;
    int age;

    Student(String n, int a) {
        name = n;
        age = a;
    }
}

class Main {
    static void printStudent(Student s) {
        System.out.println(s.name + " - " + s.age);
    }

    public static void main(String[] args) {
        Student st = new Student("Shaheer", 20);
        printStudent(st);  // passing object
    }
}
```

Passing Array of Objects:

When we pass an array of objects to a method only the reference copy of the array is passed. The method can change the properties of the objects inside the array, but it cannot replace the whole array itself.

```
class Student {
    String name;
    int age;

    Student(String n, int a) {
        name = n;
        age = a;
    }
}

class Main {
    static void printAll(Student[] arr) {
        for (Student s : arr) {
            System.out.println(s.name + " - " + s.age);
        }
    }

    public static void main(String[] args) {
        Student[] students = {
                new Student("Shaheer", 20),
                new Student("Ali", 19),
                new Student("Ahmed", 18)
        };

        printAll(students);  // passing array of objects
    }
}
```

3. **DEFINE** ENCAPSULATION, AND WHY IS IT IMPORTANT IN OOP? DESCRIBE HOW ENCAPSULATION IS ACHIEVED IN A PROGRAMMING LANGUAGE LIKE JAVA, USING AN EXAMPLE.

Encapsulation:

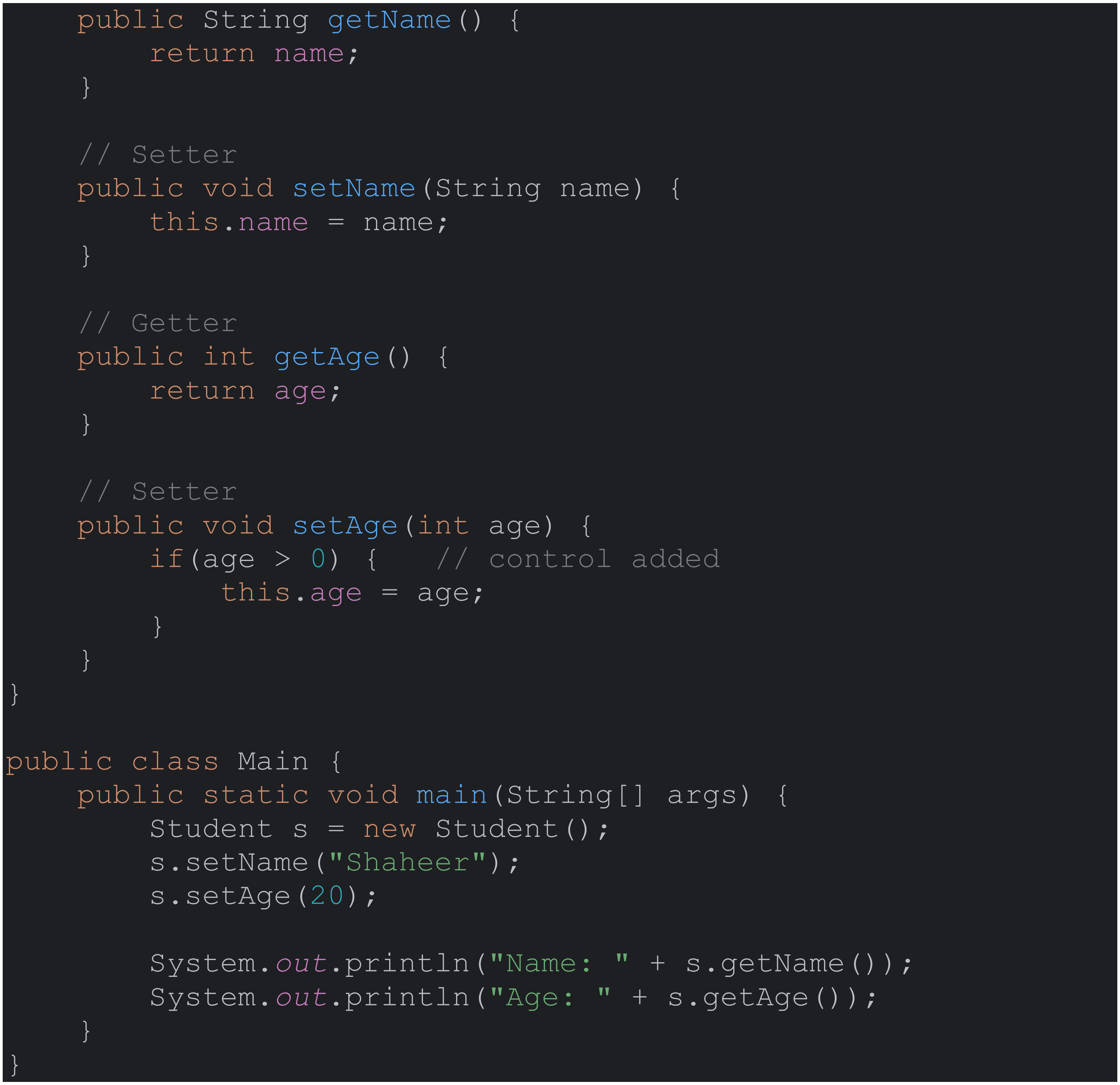
Encapsulation is the process of hiding the internal data of a class and only exposing what is necessary through methods. It means keeping variables private and providing public getters and setters to access or modify them safely.

Importance:

- Protects data from unauthorized access.
- Provides control over how variables are set or changed.
- Improves maintainability and reduces errors.

```
class Student {
    private String name;   // private variable
    private int age;

    // Getter
```



```java
    public String getName() {
        return name;
    }

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

    // Getter
    public int getAge() {
        return age;
    }

    // Setter
    public void setAge(int age) {
        if(age > 0) {    // control added
            this.age = age;
        }
    }
}

public class Main {
    public static void main(String[] args) {
        Student s = new Student();
        s.setName("Shaheer");
        s.setAge(20);

        System.out.println("Name: " + s.getName());
        System.out.println("Age: " + s.getAge());
    }
}
```

4. **EXPLAIN** THE CONCEPT OF POLYMORPHISM IN OOP. DIFFERENTIATE BETWEEN COMPILE-TIME POLYMORPHISM (OVERLOADING) AND RUNTIME POLYMORPHISM (OVERRIDING) WITH EXAMPLES.

Polymorphism:

Polymorphism in OOP means the same method name or operator can perform different tasks depending on the context.

There are two types of polymorphism:

- Overloading
- Overriding

No.	Compile-Time Polymorphism	Runtime Polymorphism
1	In compile time polymorphism the decision of which method to call is made by the compiler during compile time.	In runtime polymorphism the decision of which method to call is made by the JVM during program execution.
2	Same method name, different parameters.	Same method name and same parameters, redefined in subclass
3	Inheritance is not necessary for compile time polymorphism.	Inheritance is mandatory for runtime polymorphism.
4	Method overloading is an example of compile time polymorphism.	Method overriding is an example of runtime polymorphism.

Example Compile-time Polymorphism (Method Overloading) :

```
class Calculator {
    int add(int a, int b) {
        return a + b;
    }
    double add(double a, double b) {    // overloaded
        return a + b;
    }
}

public class Main {
    public static void main(String[] args) {
        Calculator c = new Calculator();
        System.out.println(c.add(2, 3));      // calls int
version
        System.out.println(c.add(2.5, 3.5));  // calls double
version
    }
}
```

Example Runtime Polymorphism (Method Overriding) :

```
class Animal {
    void sound() {
        System.out.println("Animal makes sound");
    }
}

class Dog extends Animal {
    @Override
    void sound() {   // overridden method
        System.out.println("Dog barks");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal a = new Dog();   // parent ref, child object
        a.sound();              // prints "Dog barks"
    }
}
```

5. A SHOPPING CENTER WANTS TO DEVELOP A SYSTEM TO MANAGE PRODUCTS, CUSTOMER PURCHASES, AND INVOICES WHILE REWARDING CUSTOMERS WITH LOYALTY POINTS.

- THE SHOPPING CENTER HAS MULTIPLE PRODUCTS AVAILABLE FOR PURCHASE.
- EACH PRODUCT HAS A NAME AND A PRICE.
- A CUSTOMER CAN PURCHASE ONE OR MORE PRODUCTS.
- WHEN A CUSTOMER MAKES A PURCHASE, AN INVOICE IS GENERATED WITH PRODUCTS PURCHASED, TOTAL

- BILL, APPLIED DISCOUNTS (IF ANY)
- CUSTOMERS EARN LOYALTY POINTS BASED ON THEIR TOTAL PURCHASE AMOUNT. (FOR EXAMPLE, 1 POINT
- FOR EVERY $10 SPENT).

QUESTIONS:

1. IDENTIFY THE CLASSES NEEDED FOR THIS SYSTEM.

2. LIST THE ATTRIBUTES FOR EACH CLASS.

3. DEFINE THE METHODS REQUIRED FOR EACH CLASS.

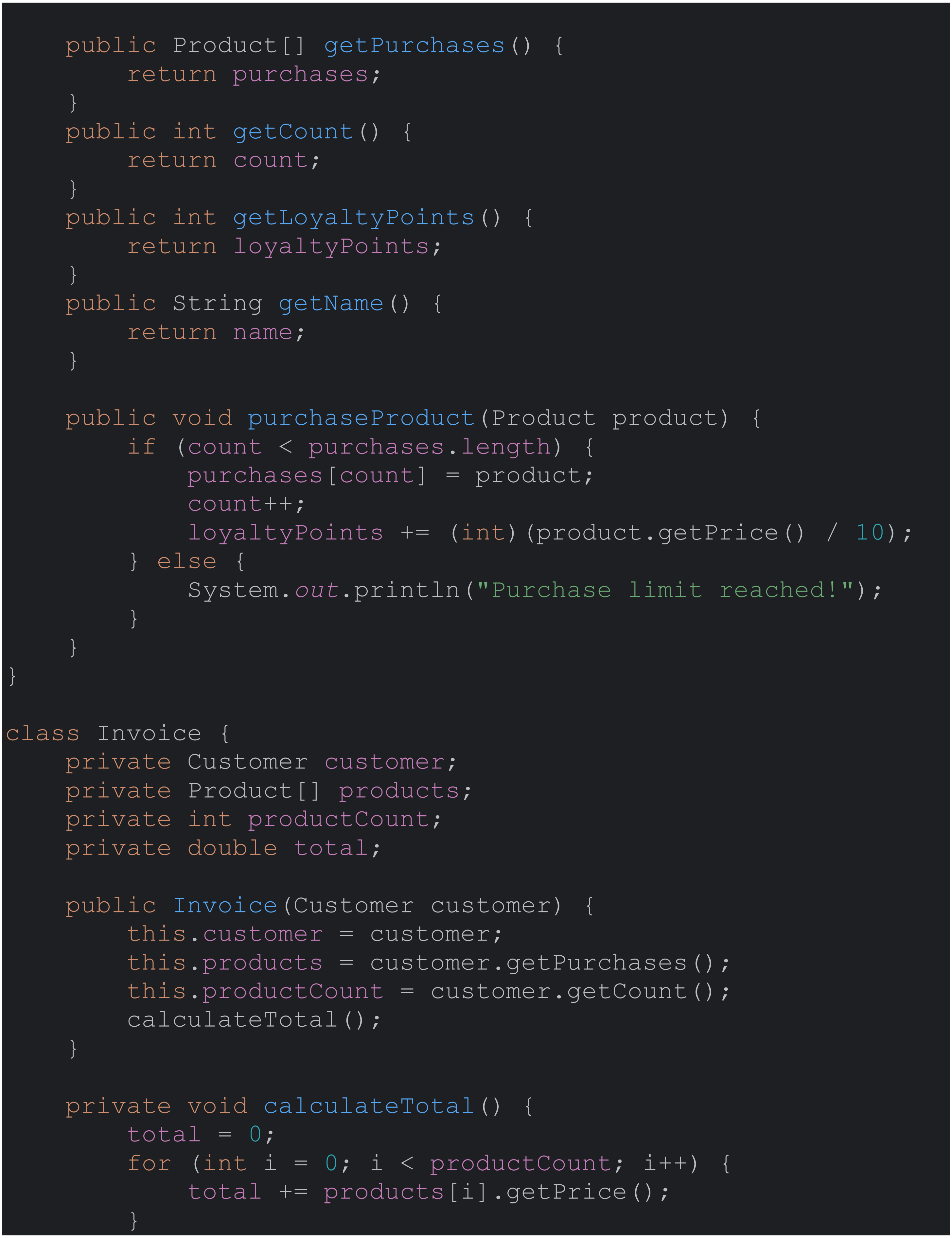
```
class Product {
    private String name;
    private double price;

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

    public String getName() {
        return name;
    }
    public double getPrice() {
        return price;
    }
}

class Customer {
    private String name;
    private int loyaltyPoints;
    private Product[] purchases;
    private int count;

    public Customer(String name, int Products) {
        this.name = name;
        this.loyaltyPoints = 0;
        this.purchases = new Product[Products];
        this.count = 0;
    }
```



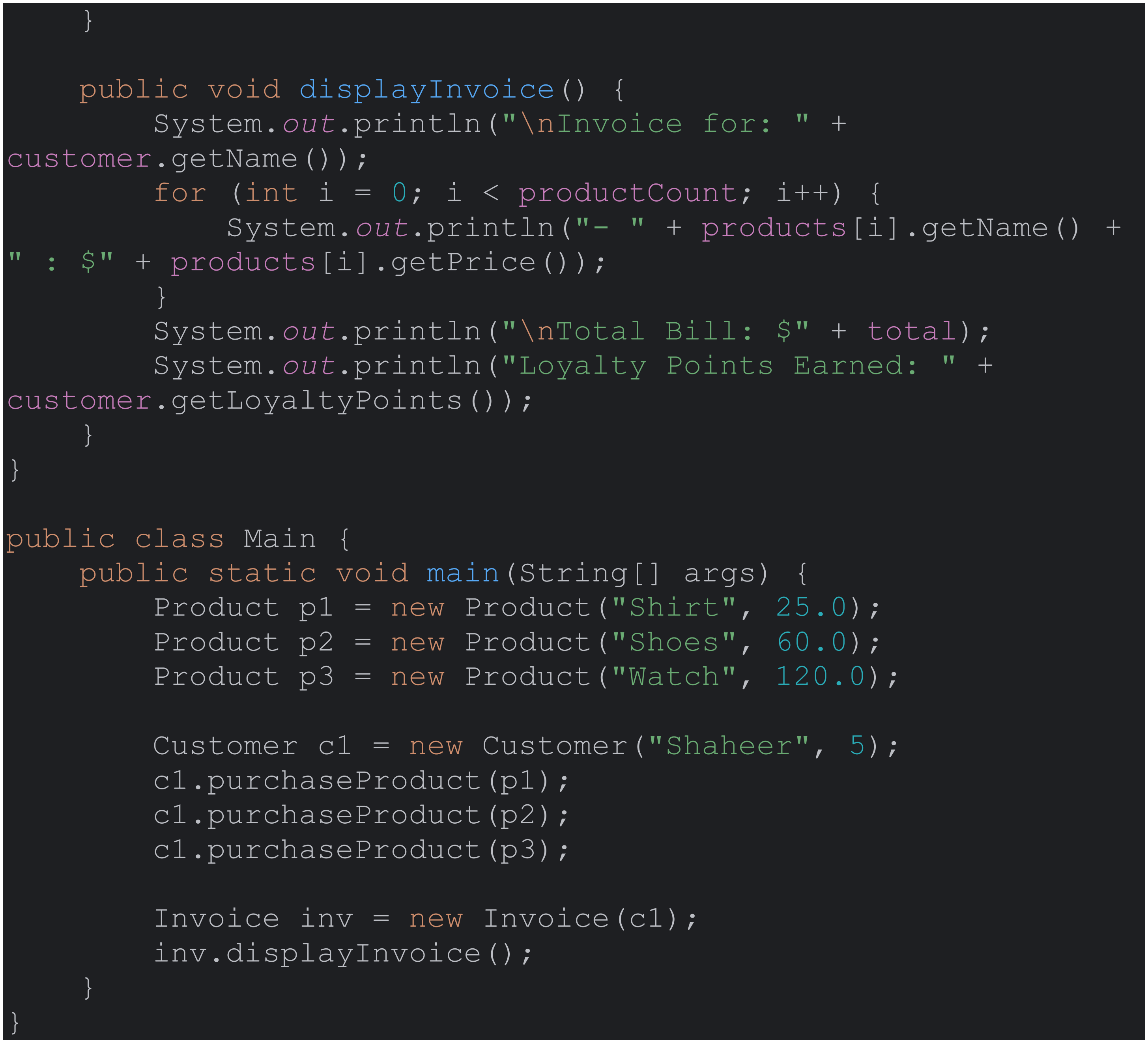
```
    public Product[] getPurchases() {
        return purchases;
    }
    public int getCount() {
        return count;
    }
    public int getLoyaltyPoints() {
        return loyaltyPoints;
    }
    public String getName() {
        return name;
    }

    public void purchaseProduct(Product product) {
        if (count < purchases.length) {
            purchases[count] = product;
            count++;
            loyaltyPoints += (int)(product.getPrice() / 10);
        } else {
            System.out.println("Purchase limit reached!");
        }
    }
}

class Invoice {
    private Customer customer;
    private Product[] products;
    private int productCount;
    private double total;

    public Invoice(Customer customer) {
        this.customer = customer;
        this.products = customer.getPurchases();
        this.productCount = customer.getCount();
        calculateTotal();
    }

    private void calculateTotal() {
        total = 0;
        for (int i = 0; i < productCount; i++) {
            total += products[i].getPrice();
        }
```



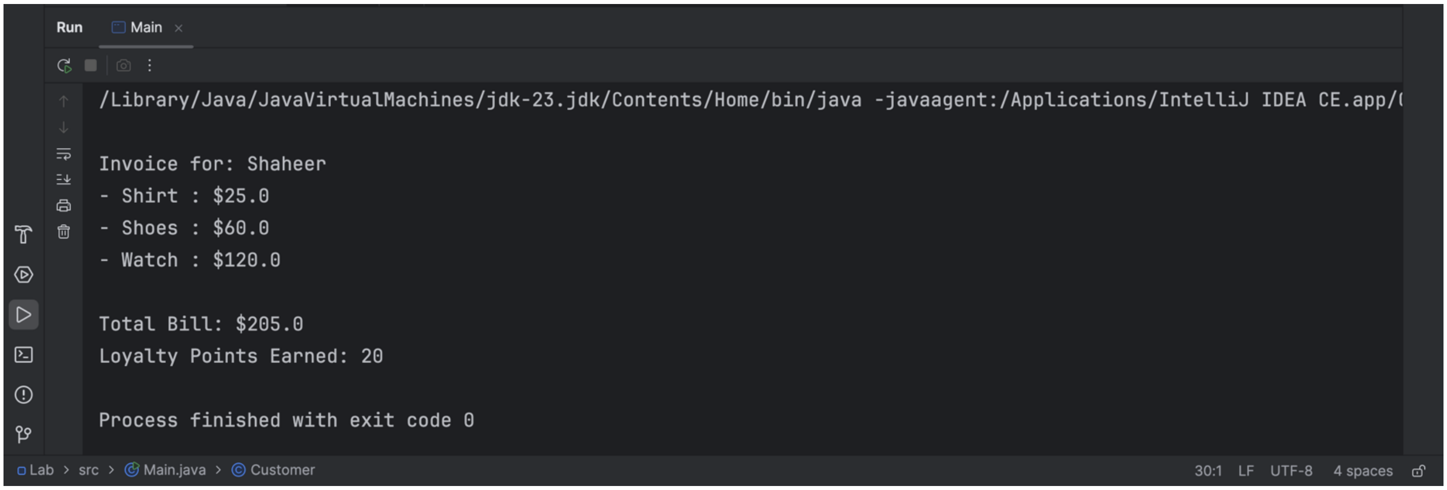
```java
    }

    public void displayInvoice() {
        System.out.println("\nInvoice for: " +
customer.getName());
        for (int i = 0; i < productCount; i++) {
            System.out.println("- " + products[i].getName() +
" : $" + products[i].getPrice());
        }
        System.out.println("\nTotal Bill: $" + total);
        System.out.println("Loyalty Points Earned: " +
customer.getLoyaltyPoints());
    }
}

public class Main {
    public static void main(String[] args) {
        Product p1 = new Product("Shirt", 25.0);
        Product p2 = new Product("Shoes", 60.0);
        Product p3 = new Product("Watch", 120.0);

        Customer c1 = new Customer("Shaheer", 5);
        c1.purchaseProduct(p1);
        c1.purchaseProduct(p2);
        c1.purchaseProduct(p3);

        Invoice inv = new Invoice(c1);
        inv.displayInvoice();
    }
}
```



```
/Library/Java/JavaVirtualMachines/jdk-23.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/

Invoice for: Shaheer
- Shirt : $25.0
- Shoes : $60.0
- Watch : $120.0

Total Bill: $205.0
Loyalty Points Earned: 20

Process finished with exit code 0
```