OBJECTIVES:

The objective of this Java online shopping cart implementation is to showcase the use of OOP (Object-Oriented Programming) concepts such as inheritance, encapsulation, polymorphism, abstract classes, and the extends keyword to connect the superclass and subclasses. The goal is to develop a system that allows users to interact with smartphones, manage shopping cart items, apply discounts, and calculate the final price.

Problem:

Develop a Java program that simulates an online shopping cart for smartphones. The program should have classes for smartphones, shopping cart items, discounts, and the shopping cart itself. The main class should demonstrate the functionality of the implemented classes.

Prerequisites:

Basic understanding of Java programming, OOP concepts (inheritance, encapsulation, polymorphism, abstract classes), and the extends keyword.

Learning Outcome:

By completing this exercise, learners will gain practical experience in implementing OOP concepts in Java and will be able to create a functioning online shopping cart system.

Problem Analysis:

1. Define a superclass Smartphone with attributes: price, model, and brand.
2. Implement a subclass ShoppingCartItem that extends Smartphone and adds an additional attribute: quantity.
3. Create a subclass Discount that also extends Smartphone and adds an attribute for the discount value.
4. Develop another subclass ShoppingCart that contains a list of ShoppingCartItem objects and provides methods to manage the shopping cart, such as adding/removing items and calculating the total price after applying discounts.
5. Implement the main class that demonstrates the use of all these classes by creating smartphone objects, shopping cart items, discounts, and the shopping cart itself.

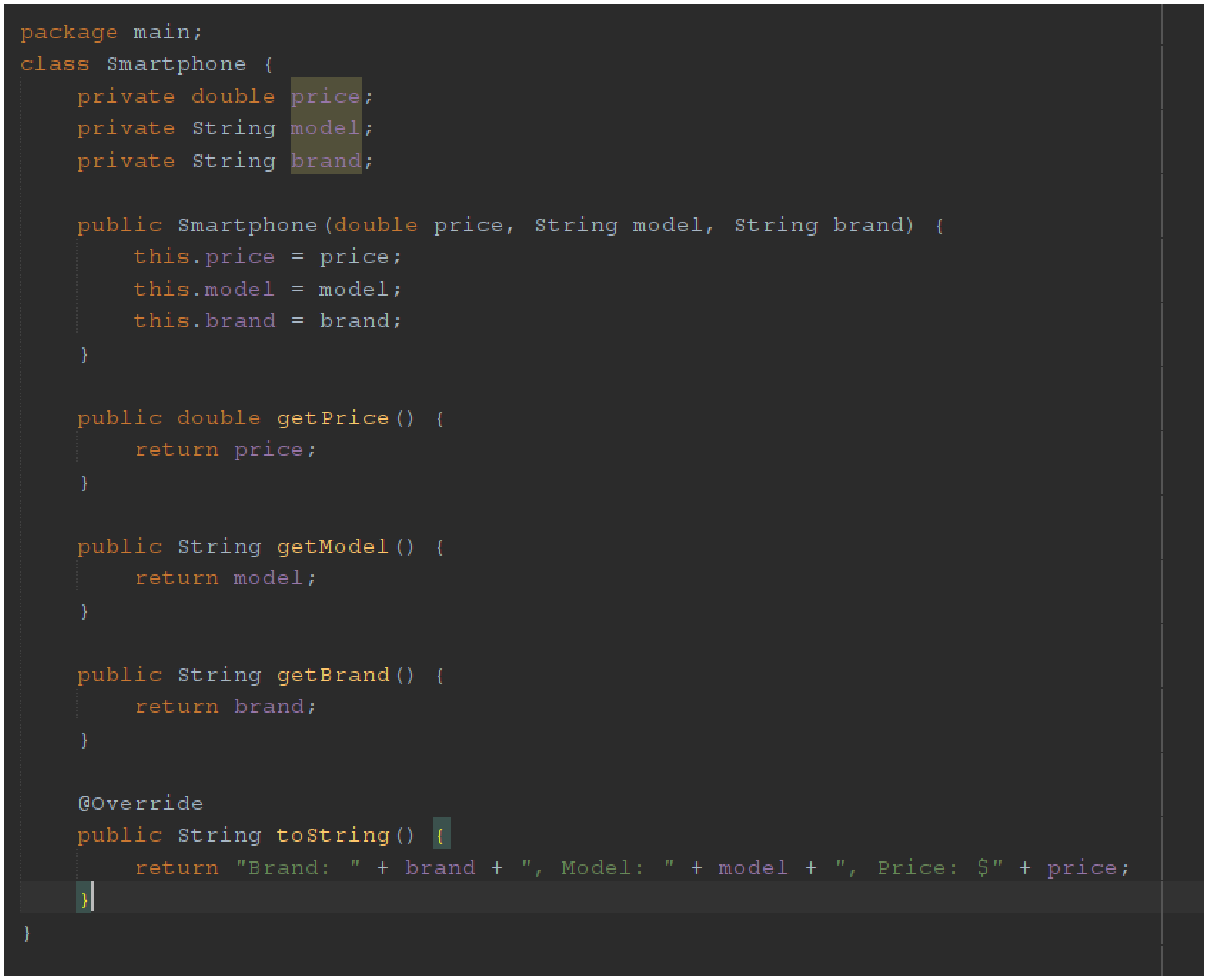
Background Theory:

1. Inheritance: Allows a subclass to inherit attributes and methods from its superclass, facilitating code reuse and creating a hierarchical relationship between classes.
2. Encapsulation: Restricts direct access to class attributes and provides getter and setter methods to control data access.
3. Polymorphism: Enables a subclass to override methods from its superclass, allowing for different implementations of the same method signature.
4. Abstract Classes: A class that cannot be instantiated and may contain abstract methods (without implementation). Subclasses must implement these abstract methods.

Algorithm Design:

1. Define the Smartphone class with attributes, getter, setter methods, and the toString() method.
2. Implement the ShoppingCartItem class that extends Smartphone and includes an additional quantity attribute. Override the toString() method to include quantity and total price details.
3. Create the Discount class, which extends Smartphone and includes a discount attribute, along with getter and setter methods for the discount value.
4. Develop the ShoppingCart class, which contains a list of ShoppingCartItem objects and a totalDiscount attribute. Include methods to add/remove items, set the total discount, display cart details, and calculate the total price.
5. In the main class, create instances of smartphones, shopping cart items, discounts, and the shopping cart itself. Perform operations like adding/removing items, setting discounts, and calculating the final price. Print the output to demonstrate the functionality.

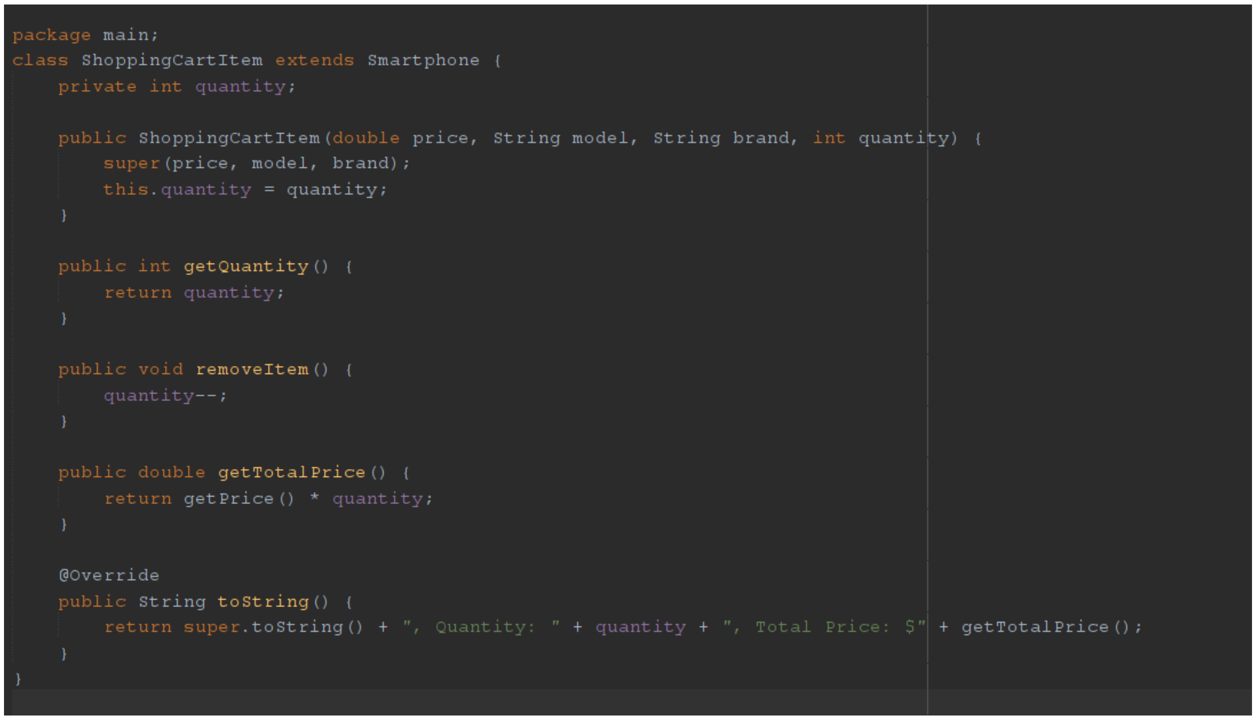
Code:



```java
package main;
class Smartphone {
    private double price;
    private String model;
    private String brand;

    public Smartphone(double price, String model, String brand) {
        this.price = price;
        this.model = model;
        this.brand = brand;
    }

    public double getPrice() {
        return price;
    }

    public String getModel() {
        return model;
    }

    public String getBrand() {
        return brand;
    }

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

Figure 1- Smartphone

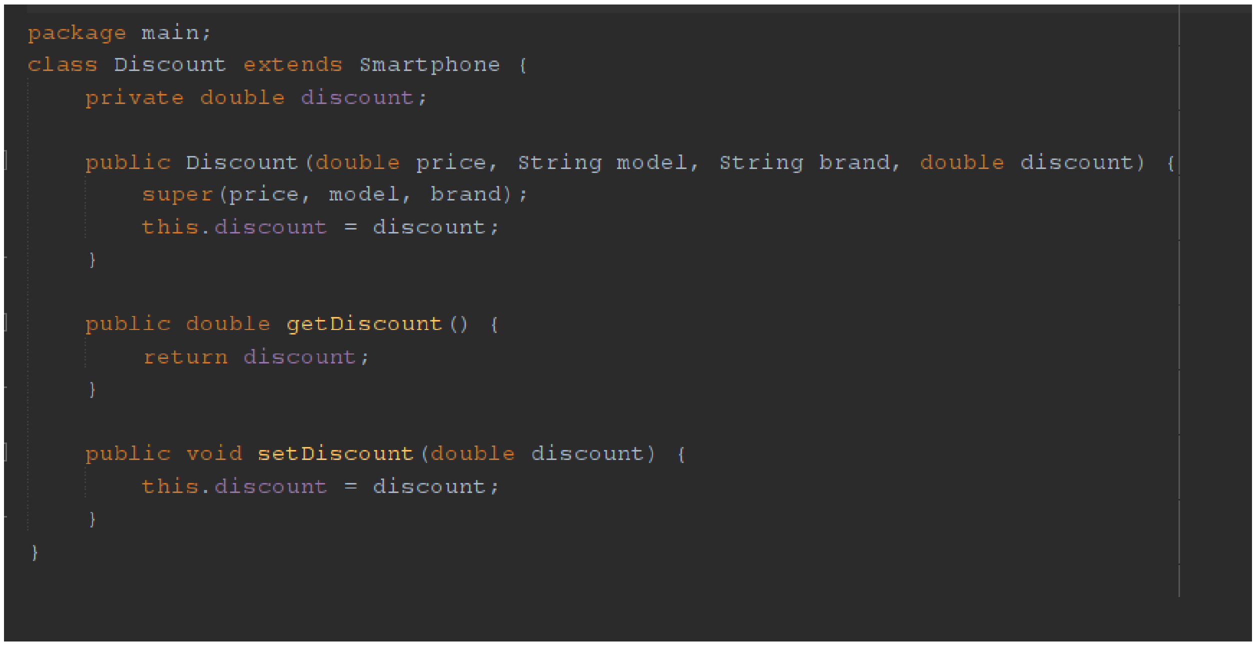
This code defines a class named Smartphone with three private attributes: price, model, and brand. It has a constructor to initialize these attributes, getter methods to access their values, and an overridden toString() method to display the smartphone details in a formatted string. This class represents a basic smartphone model with its price, model name, and brand.



```java
package main;
class ShoppingCartItem extends Smartphone {
    private int quantity;

    public ShoppingCartItem(double price, String model, String brand, int quantity) {
        super(price, model, brand);
        this.quantity = quantity;
    }

    public int getQuantity() {
        return quantity;
    }

    public void removeItem() {
        quantity--;
    }

    public double getTotalPrice() {
        return getPrice() * quantity;
    }

    @Override
    public String toString() {
        return super.toString() + ", Quantity: " + quantity + ", Total Price: $" + getTotalPrice();
    }
}
```

Figure-2 Shopping Cart Item

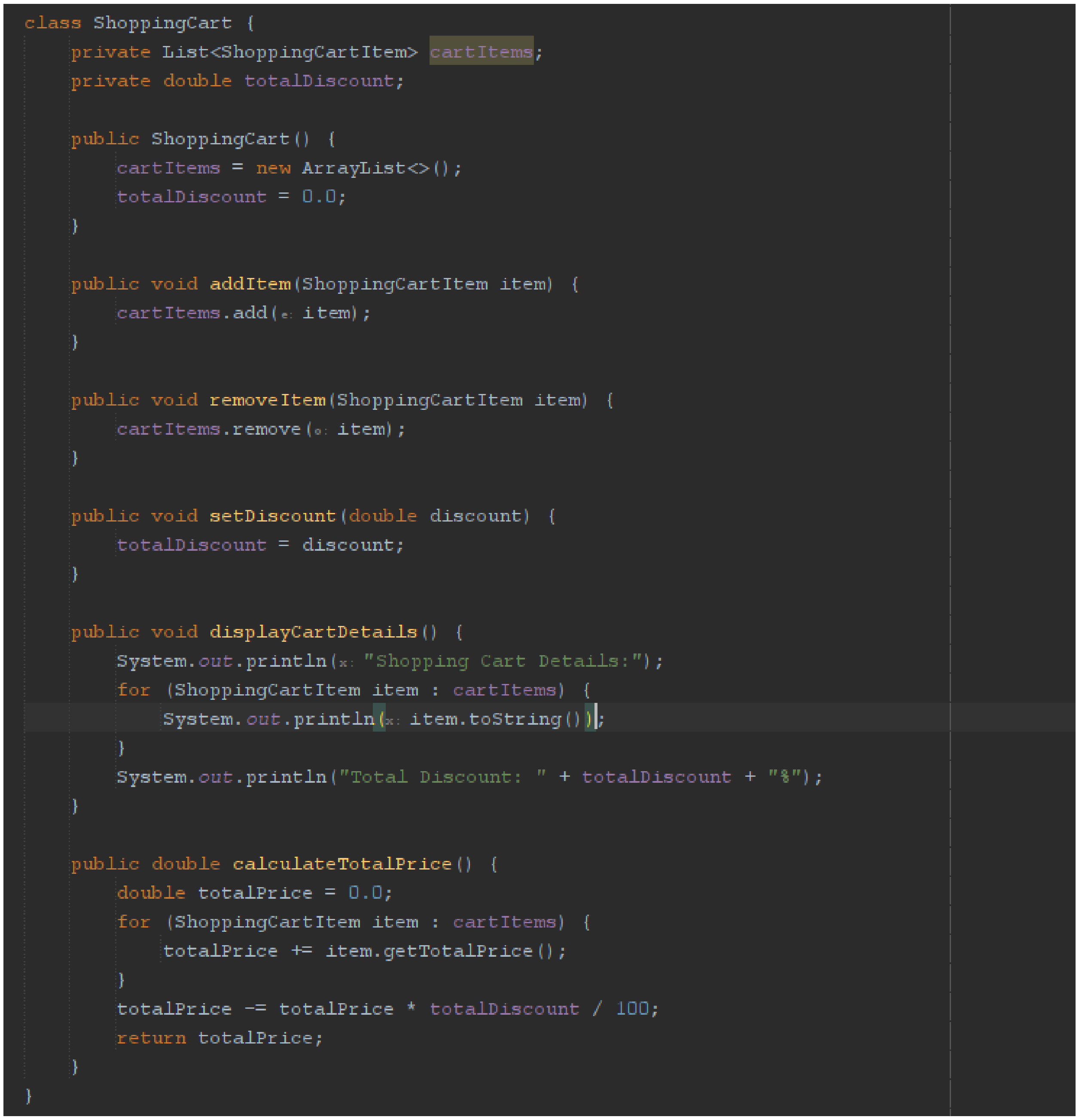
This code defines a subclass ShoppingCartItem that extends the superclass Smartphone. It represents an item in a shopping cart with additional features for managing quantity and calculating the total price based on the quantity and smartphone price. The toString() method is overridden to provide a formatted string representation of the shopping cart item, including brand, model, price, quantity, and total price.



```java
package main;
class Discount extends Smartphone {
    private double discount;

    public Discount(double price, String model, String brand, double discount) {
        super(price, model, brand);
        this.discount = discount;
    }

    public double getDiscount() {
        return discount;
    }

    public void setDiscount(double discount) {
        this.discount = discount;
    }
}
```

Figure-3 Discount

The provided code defines a class Discount that extends the Smartphone class. It adds an extra attribute discount to represent the discount percentage on a smartphone. The class provides methods to get and set the discount value.



```java
class ShoppingCart {
    private List<ShoppingCartItem> cartItems;
    private double totalDiscount;

    public ShoppingCart() {
        cartItems = new ArrayList<>();
        totalDiscount = 0.0;
    }

    public void addItem(ShoppingCartItem item) {
        cartItems.add( e: item);
    }

    public void removeItem(ShoppingCartItem item) {
        cartItems.remove( o: item);
    }

    public void setDiscount(double discount) {
        totalDiscount = discount;
    }

    public void displayCartDetails() {
        System.out.println( x: "Shopping Cart Details:");
        for (ShoppingCartItem item : cartItems) {
            System.out.println( x: item.toString());
        }
        System.out.println("Total Discount: " + totalDiscount + "%");
    }

    public double calculateTotalPrice() {
        double totalPrice = 0.0;
        for (ShoppingCartItem item : cartItems) {
            totalPrice += item.getTotalPrice();
        }
        totalPrice -= totalPrice * totalDiscount / 100;
        return totalPrice;
    }
}
```

Figure-4 Shopping Cart

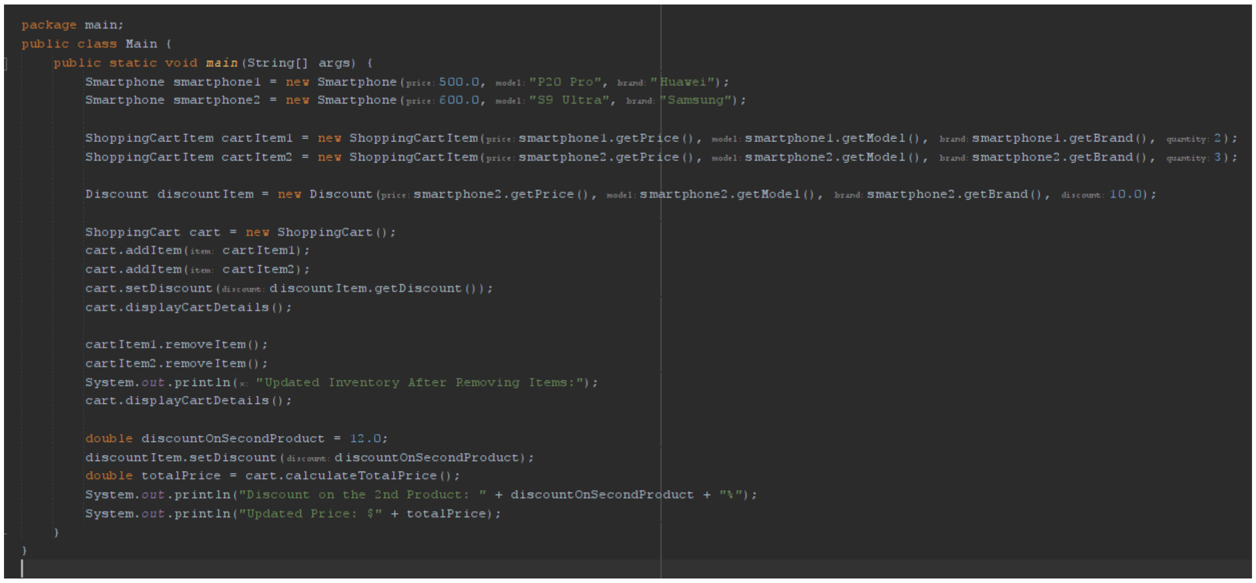
This code defines a Java class called ShoppingCart, which represents an online shopping cart. The ShoppingCart class has attributes for storing a list of ShoppingCartItem objects and a total discount value. The class contains methods to add and remove items from the cart, set the total discount, display cart details, and calculate the final price after applying the discount.

The ShoppingCart constructor initializes an empty list of cart items and sets the total discount to 0.0. The addItem method allows adding a ShoppingCartItem object to the cart, and the removeItem method allows removing an item from the cart.

The setDiscount method is used to set the total discount for the entire cart as a percentage. The displayCartDetails method prints the details of all the items in the cart, including their quantity and total price, as well as the overall total discount.

Finally, the calculateTotalPrice method calculates the total price of all items in the cart after applying the discount. It iterates through each ShoppingCartItem in the cart, sums up their individual total prices, and then reduces the total by the discount amount.

Overall, this code provides a basic structure to manage an online shopping cart with items, quantities, and discounts, demonstrating some essential aspects of Object-Oriented Programming in Java.



```java
package main;
public class Main {
    public static void main(String[] args) {
        Smartphone smartphone1 = new Smartphone(price: 500.0, model: "P20 Pro", brand: "Huawei");
        Smartphone smartphone2 = new Smartphone(price: 600.0, model: "S9 Ultra", brand: "Samsung");

        ShoppingCartItem cartItem1 = new ShoppingCartItem(price: smartphone1.getPrice(), model: smartphone1.getModel(), brand: smartphone1.getBrand(), quantity: 2);
        ShoppingCartItem cartItem2 = new ShoppingCartItem(price: smartphone2.getPrice(), model: smartphone2.getModel(), brand: smartphone2.getBrand(), quantity: 3);

        Discount discountItem = new Discount(price: smartphone2.getPrice(), model: smartphone2.getModel(), brand: smartphone2.getBrand(), discount: 10.0);

        ShoppingCart cart = new ShoppingCart();
        cart.addItem(item: cartItem1);
        cart.addItem(item: cartItem2);
        cart.setDiscount(discount: discountItem.getDiscount());
        cart.displayCartDetails();

        cartItem1.removeItem();
        cartItem2.removeItem();
        System.out.println(x: "Updated Inventory After Removing Items:");
        cart.displayCartDetails();

        double discountOnSecondProduct = 12.0;
        discountItem.setDiscount(discount: discountOnSecondProduct);
        double totalPrice = cart.calculateTotalPrice();
        System.out.println("Discount on the 2nd Product: " + discountOnSecondProduct + "%");
        System.out.println("Updated Price: $" + totalPrice);
    }
}
```

Figure-5 Main

This code simulates an online shopping cart for smartphones using Java and OOP concepts. The Main class showcases the functionality of various classes: Smartphone, ShoppingCartItem, Discount, and ShoppingCart.

In the Main class, two smartphone objects (smartphone1 and smartphone2) are created with their prices, models, and brands. Then, two shopping cart items (cartItem1 and cartItem2) are initialized with corresponding smartphone details and quantities. A discount item (discountItem) is created with a discount value for the second smartphone.

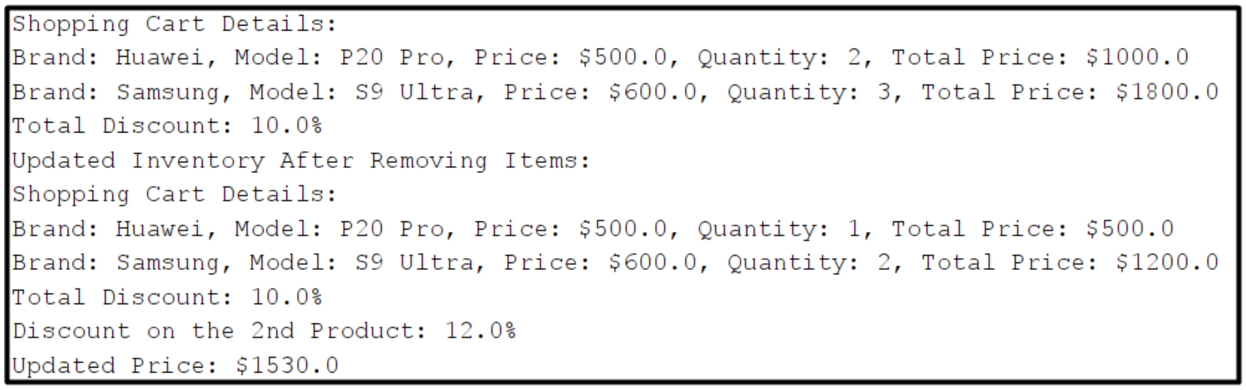
Next, a shopping cart (cart) is instantiated. Both shopping cart items are added to the cart, and the discount is set using the discount item. The cart details, including items and total discount, are displayed.

After that, we simulate removing items from the cart using removeItem() method of ShoppingCartItem. The updated cart details are printed.

Lastly, we apply a new discount to the second smartphone using the setDiscount() method of the Discount class. The final total price of the cart after applying the discounts is calculated and displayed.

In summary, this code demonstrates the functionality of a simple online shopping cart for smartphones, where users can add, remove items, set discounts, and calculate the final price with the help of various classes using OOP concepts in Java.

Output



```
Shopping Cart Details:
Brand: Huawei, Model: P20 Pro, Price: $500.0, Quantity: 2, Total Price: $1000.0
Brand: Samsung, Model: S9 Ultra, Price: $600.0, Quantity: 3, Total Price: $1800.0
Total Discount: 10.0%
Updated Inventory After Removing Items:
Shopping Cart Details:
Brand: Huawei, Model: P20 Pro, Price: $500.0, Quantity: 1, Total Price: $500.0
Brand: Samsung, Model: S9 Ultra, Price: $600.0, Quantity: 2, Total Price: $1200.0
Total Discount: 10.0%
Discount on the 2nd Product: 12.0%
Updated Price: $1530.0
```

Conclusion:

In conclusion, the provided Java code demonstrates the implementation of an online shopping cart for smartphones using object-oriented programming (OOP) concepts. The code consists of a superclass called Smartphone that holds attributes like price, model, and brand, and two subclasses, ShoppingCartItem and Discount, which extend the Smartphone class to add specific functionalities.

The ShoppingCartItem class allows users to manage the shopping cart by setting the quantity of smartphones and calculating the total price for a particular item. The Discount class enables users to set discounts for specific smartphone items.

The main ShoppingCart class manages the overall shopping cart, allowing users to add and remove items, set the total discount, display cart details, and calculate the final price after applying the discounts.

In the Main class, a demonstration of the functionality is provided. It creates two smartphone objects, two shopping cart items, and a discount item. The program adds items to the shopping cart, sets the discount, removes items, and calculates the final price.

In summary, this code showcases the effective use of OOP principles such as inheritance, encapsulation, polymorphism, and the extends keyword. It illustrates the flexibility and extensibility of the OOP paradigm in building a scalable and organized online shopping cart system for smartphones. Developers can easily modify and expand the code to accommodate additional features and product types while maintaining a clear and structured architecture.