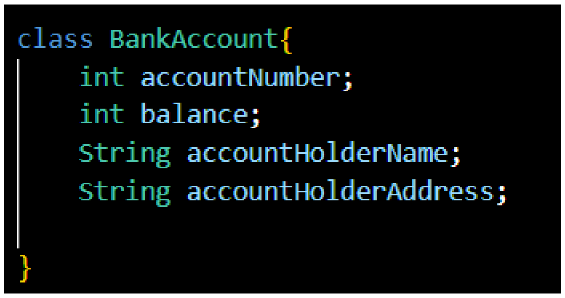
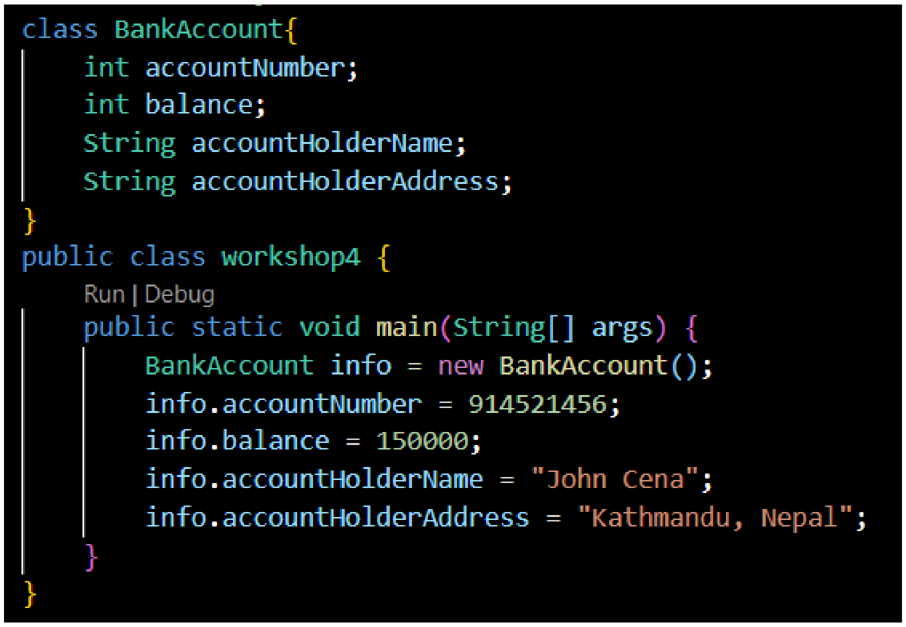
Basics of Classes and Objects:

1. Define a class named BankAccount with attributes accountNumber, balance, and accountHolderName, accountHolderAddress.



```
class BankAccount{
    int accountNumber;
    int balance;
    String accountHolderName;
    String accountHolderAddress;

}
```

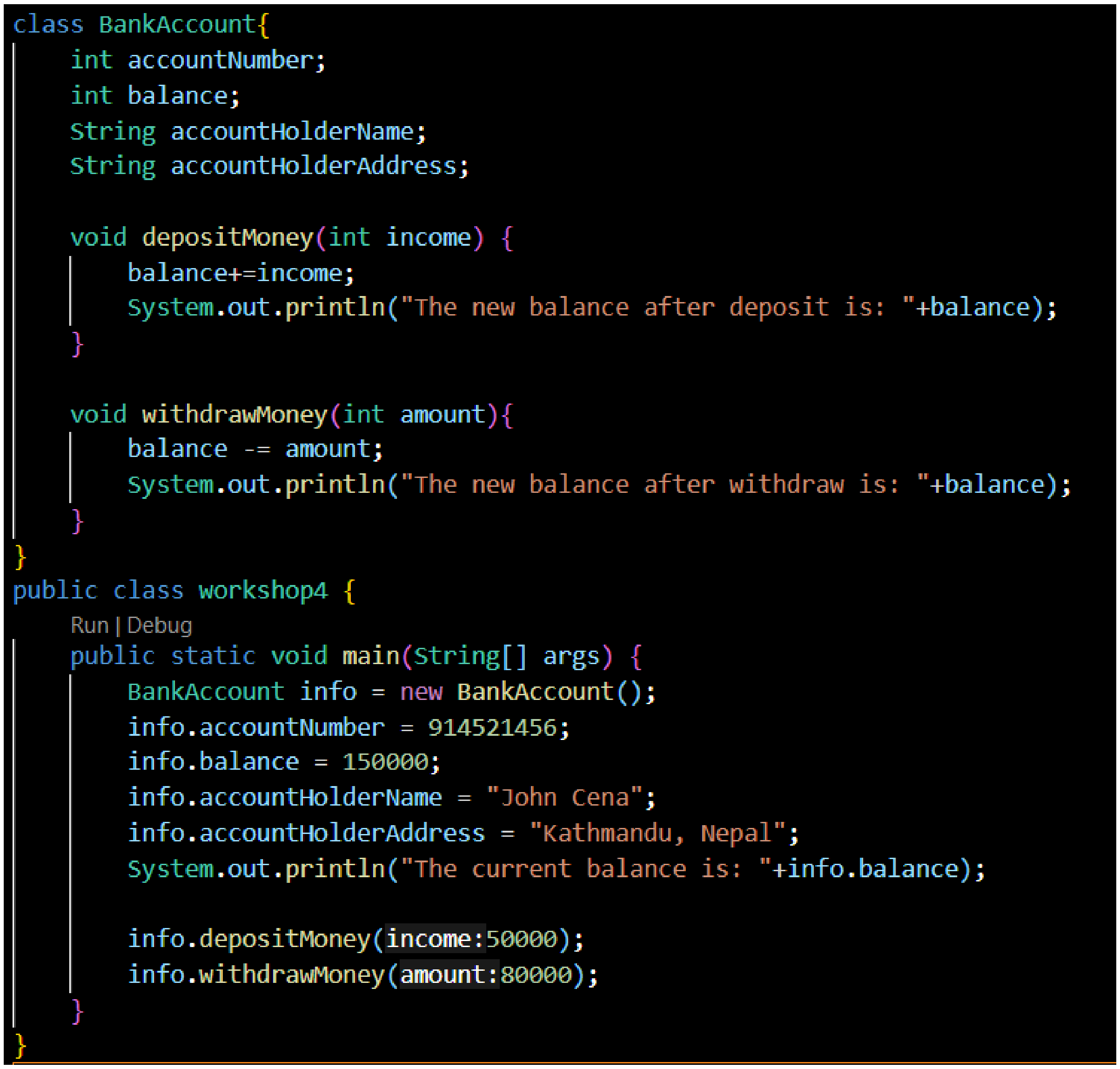
2. Create an object of this class and initialize its attributes.



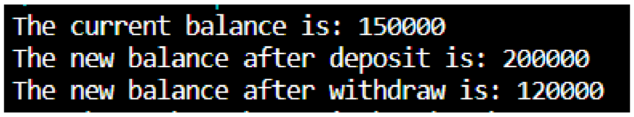
```
class BankAccount{
    int accountNumber;
    int balance;
    String accountHolderName;
    String accountHolderAddress;
}
public class workshop4 {
    Run | Debug
    public static void main(String[] args) {
        BankAccount info = new BankAccount();
        info.accountNumber = 914521456;
        info.balance = 150000;
        info.accountHolderName = "John Cena";
        info.accountHolderAddress = "Kathmandu, Nepal";
    }
}
```

Methods:

3. Create a method, **depositMoney()** in the BankAccount class to deposit money. Implement another method, **withdrawMoney()** to withdraw money. (The current balance should also be printed).

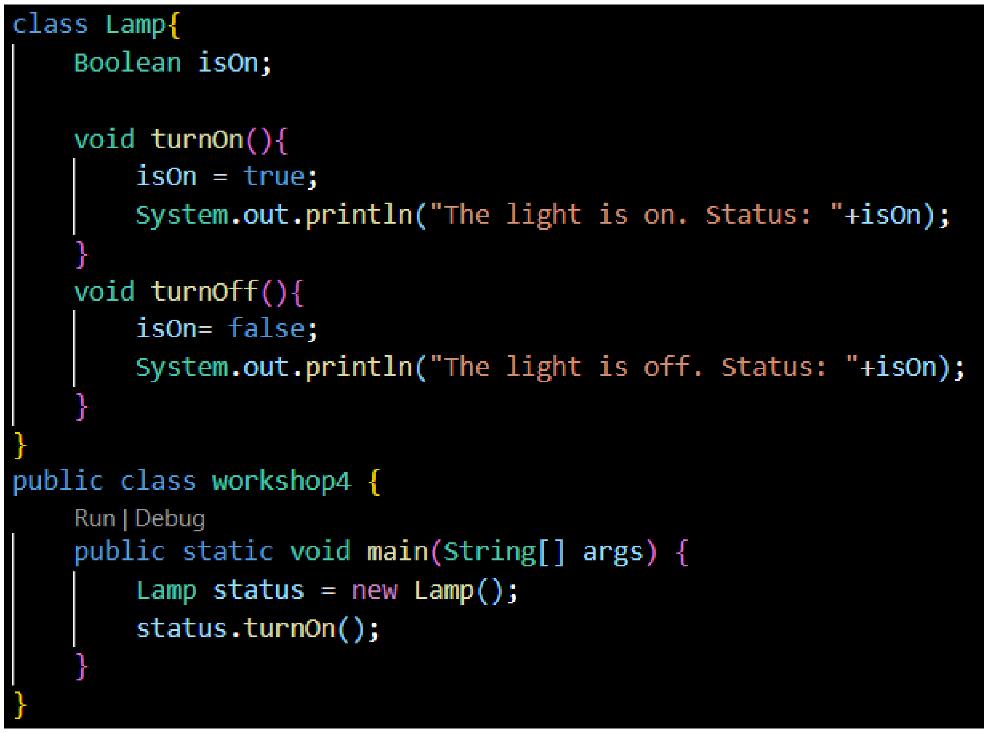


```java
class BankAccount{
    int accountNumber;
    int balance;
    String accountHolderName;
    String accountHolderAddress;

    void depositMoney(int income) {
        balance+=income;
        System.out.println("The new balance after deposit is: "+balance);
    }

    void withdrawMoney(int amount){
        balance -= amount;
        System.out.println("The new balance after withdraw is: "+balance);
    }
}
public class workshop4 {
    Run | Debug
    public static void main(String[] args) {
        BankAccount info = new BankAccount();
        info.accountNumber = 914521456;
        info.balance = 150000;
        info.accountHolderName = "John Cena";
        info.accountHolderAddress = "Kathmandu, Nepal";
        System.out.println("The current balance is: "+info.balance);

        info.depositMoney(income:50000);
        info.withdrawMoney(amount:80000);
    }
}
```

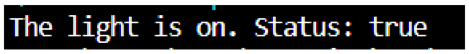


```
The current balance is: 150000
The new balance after deposit is: 200000
The new balance after withdraw is: 120000
```

4. Create a class Lamp with attributes **isOn** to store boolean value. Also create a method **turnOn()** to turn on the light, and **turnOff()** to turn off the light and print the on status of the light.



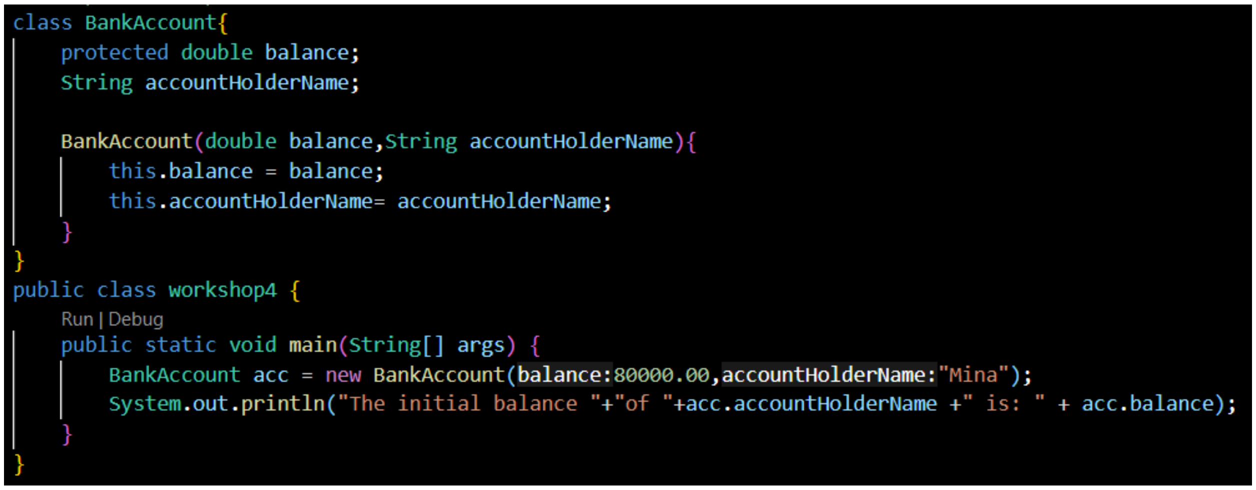
```
class Lamp{
    Boolean isOn;

    void turnOn(){
        isOn = true;
        System.out.println("The light is on. Status: "+isOn);
    }
    void turnOff(){
        isOn= false;
        System.out.println("The light is off. Status: "+isOn);
    }
}
public class workshop4 {
    Run | Debug
    public static void main(String[] args) {
        Lamp status = new Lamp();
        status.turnOn();
    }
}
```



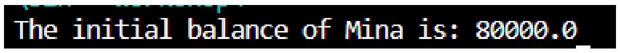
```
The light is on. Status: true
```

Constructors:

5. Implement a parameterized constructor for the BankAccount class that initializes the account attributes. Create an object using this constructor.

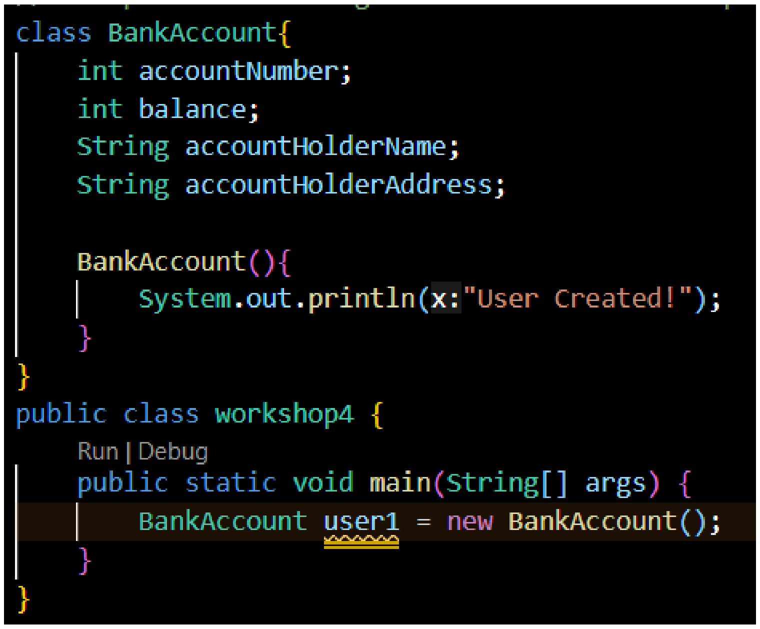


```
class BankAccount{
    protected double balance;
    String accountHolderName;

    BankAccount(double balance,String accountHolderName){
        this.balance = balance;
        this.accountHolderName= accountHolderName;
    }
}
public class workshop4 {
    Run | Debug
    public static void main(String[] args) {
        BankAccount acc = new BankAccount(balance:80000.00,accountHolderName:"Mina");
        System.out.println("The initial balance "+"of "+acc.accountHolderName +" is: " + acc.balance);
    }
}
```

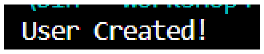


```
The initial balance of Mina is: 80000.0
```

6. Implement a no-argument constructor that prints out “**User created!**” as soon as the instance of the user is created.



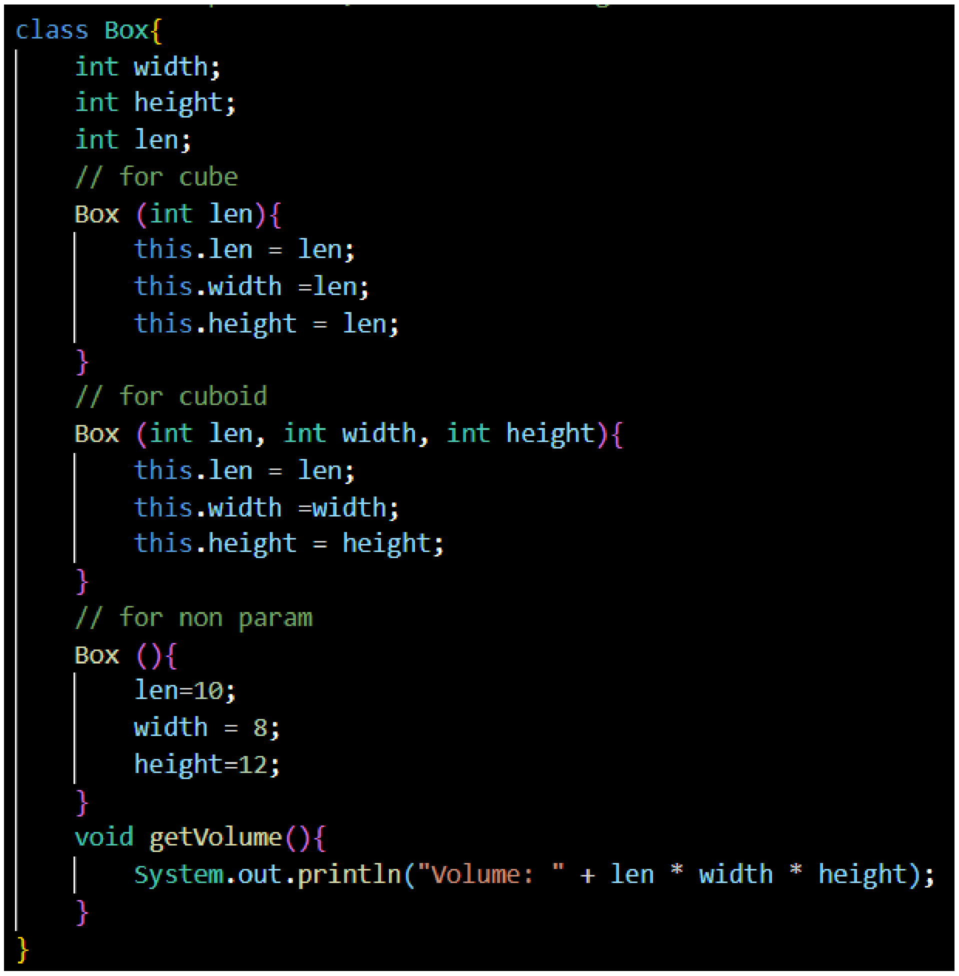
```java
class BankAccount{
    int accountNumber;
    int balance;
    String accountHolderName;
    String accountHolderAddress;

    BankAccount(){
        System.out.println(x:"User Created!");
    }
}
public class workshop4 {
    Run | Debug
    public static void main(String[] args) {
        BankAccount user1 = new BankAccount();
    }
}
```



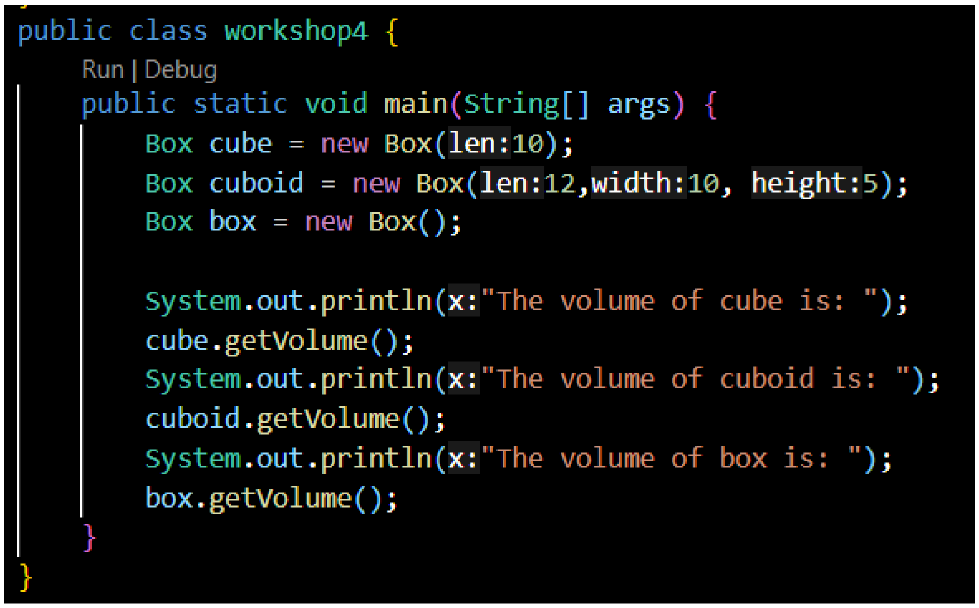
```
User Created!
```

Constructor Overloading:

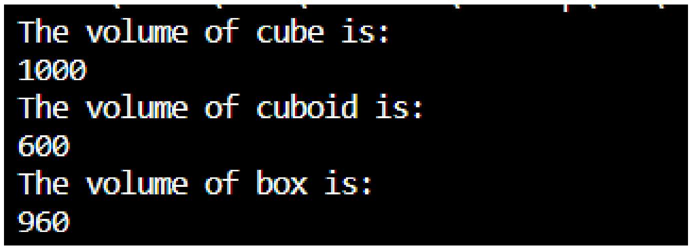
7. Create a class named ,”**Box**” with attributes **width**, **height**, and **depth.** Create multiple constructors for handling following object declarations. Also declare a method **getVolume()** that prints the volume of the declared:
   a. For a cube, declare a constructor to take length only.
   b. For a cuboid, declare a constructor to take length, breadth, and height.
   c. For no parameter, declare a no-argument constructor that sets **length = 10**, **breadth = 8**, and **height = 12**.



```
class Box{
    int width;
    int height;
    int len;
    // for cube
    Box (int len){
        this.len = len;
        this.width =len;
        this.height = len;
    }
    // for cuboid
    Box (int len, int width, int height){
        this.len = len;
        this.width =width;
        this.height = height;
    }
    // for non param
    Box (){
        len=10;
        width = 8;
        height=12;
    }
    void getVolume(){
        System.out.println("Volume: " + len * width * height);
    }
}
```



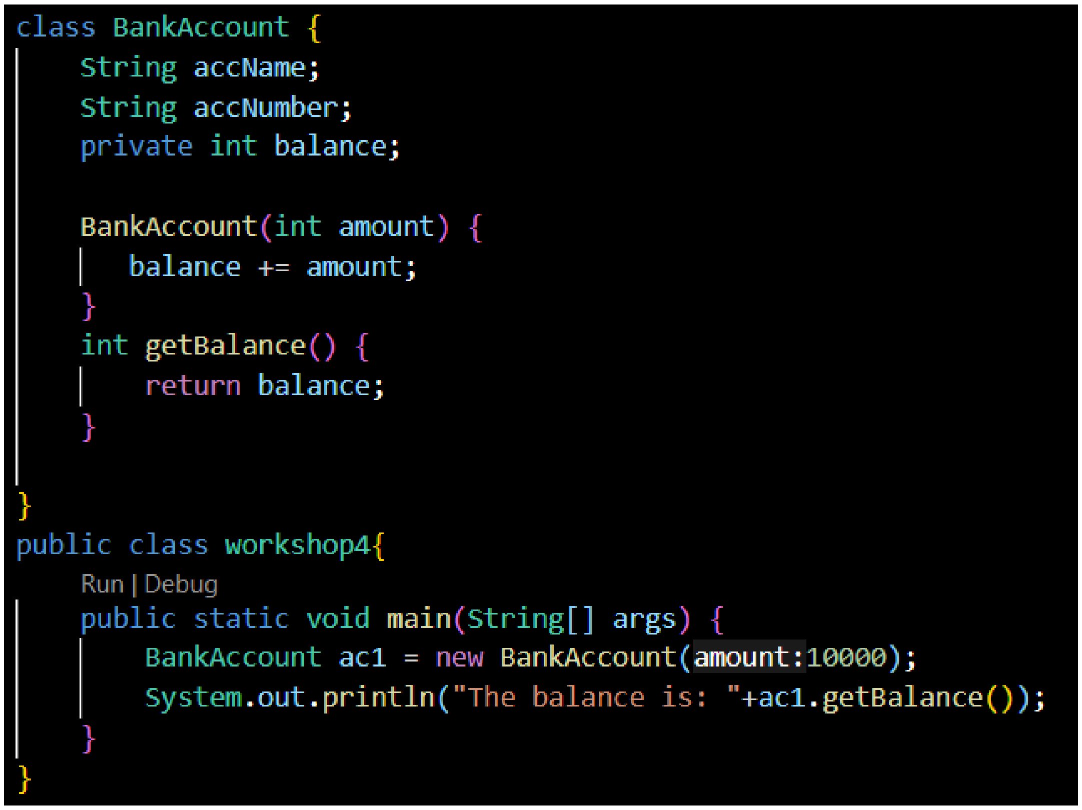
```
public class workshop4 {
    Run | Debug
    public static void main(String[] args) {
        Box cube = new Box(len:10);
        Box cuboid = new Box(len:12,width:10, height:5);
        Box box = new Box();

        System.out.println(x:"The volume of cube is: ");
        cube.getVolume();
        System.out.println(x:"The volume of cuboid is: ");
        cuboid.getVolume();
        System.out.println(x:"The volume of box is: ");
        box.getVolume();
    }
}
```



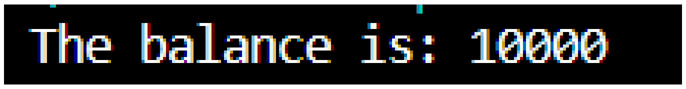
```
The volume of cube is:
1000
The volume of cuboid is:
600
The volume of box is:
960
```

Access Modifiers:

8. Set the balance attribute in the BankAccount class as private. Provide public getter methods for the balance.



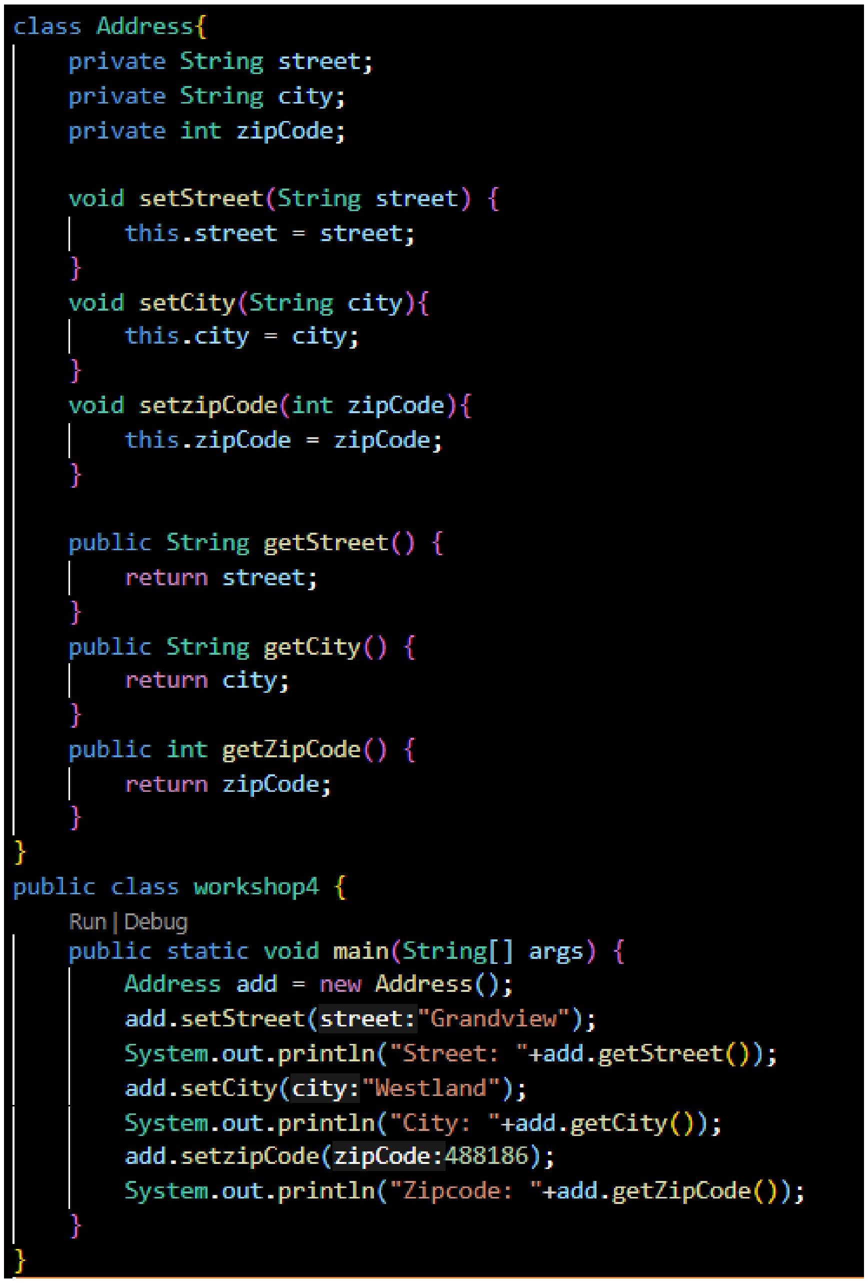
```
class BankAccount {
    String accName;
    String accNumber;
    private int balance;

    BankAccount(int amount) {
       balance += amount;
    }
    int getBalance() {
        return balance;
    }

}
public class workshop4{
    Run | Debug
    public static void main(String[] args) {
        BankAccount ac1 = new BankAccount(amount:10000);
        System.out.println("The balance is: "+ac1.getBalance());
    }
}
```



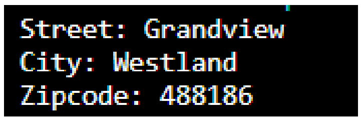
```
The balance is: 10000
```

Encapsulation:

9. Create a class Address with private attributes street, city, and zipCode. Use encapsulation and provide getter method.



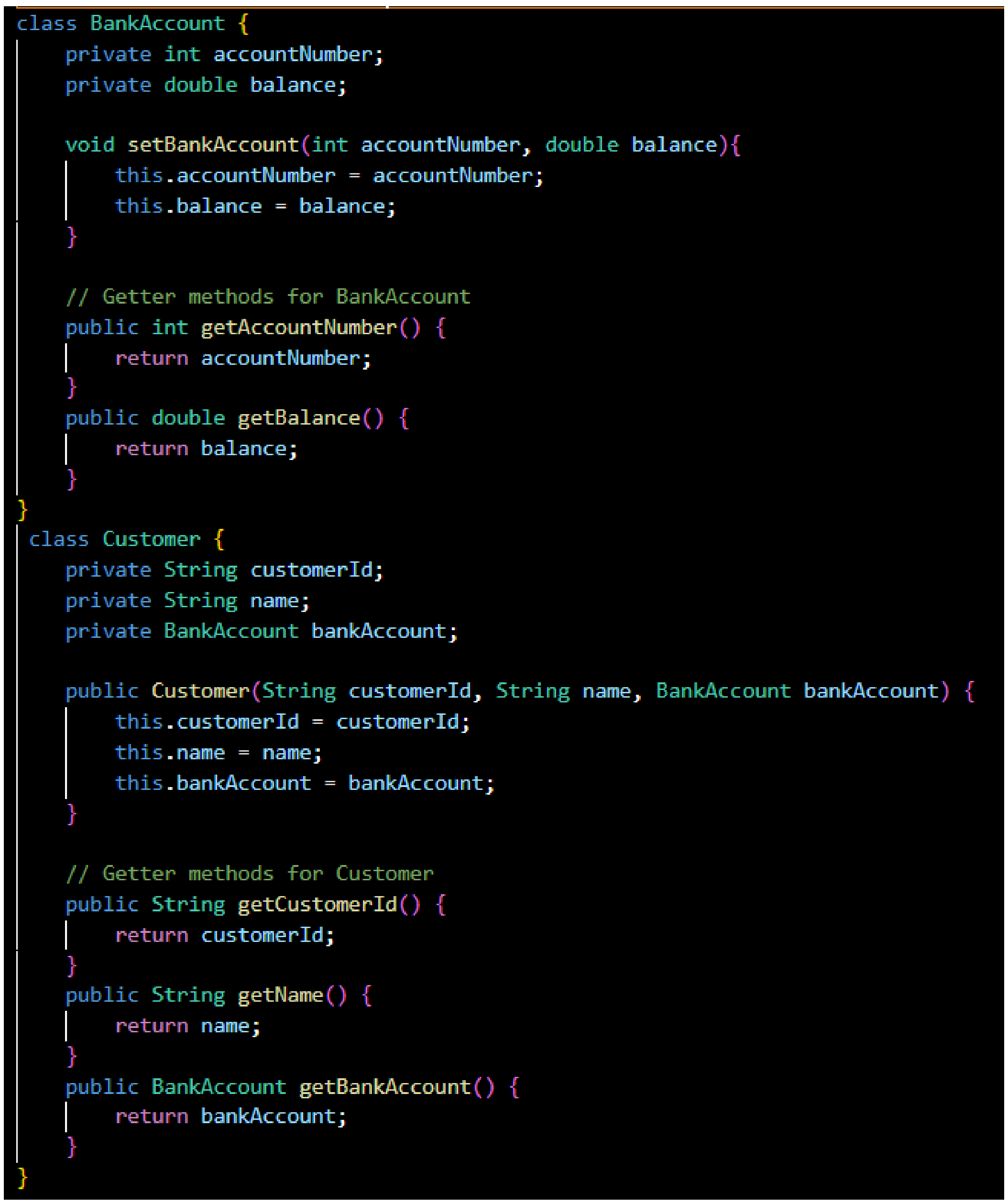
```java
class Address{
    private String street;
    private String city;
    private int zipCode;

    void setStreet(String street) {
        this.street = street;
    }
    void setCity(String city){
        this.city = city;
    }
    void setzipCode(int zipCode){
        this.zipCode = zipCode;
    }

    public String getStreet() {
        return street;
    }
    public String getCity() {
        return city;
    }
    public int getZipCode() {
        return zipCode;
    }
}
public class workshop4 {
    Run | Debug
    public static void main(String[] args) {
        Address add = new Address();
        add.setStreet(street:"Grandview");
        System.out.println("Street: "+add.getStreet());
        add.setCity(city:"Westland");
        System.out.println("City: "+add.getCity());
        add.setzipCode(zipCode:488186);
        System.out.println("Zipcode: "+add.getZipCode());
    }
}
```



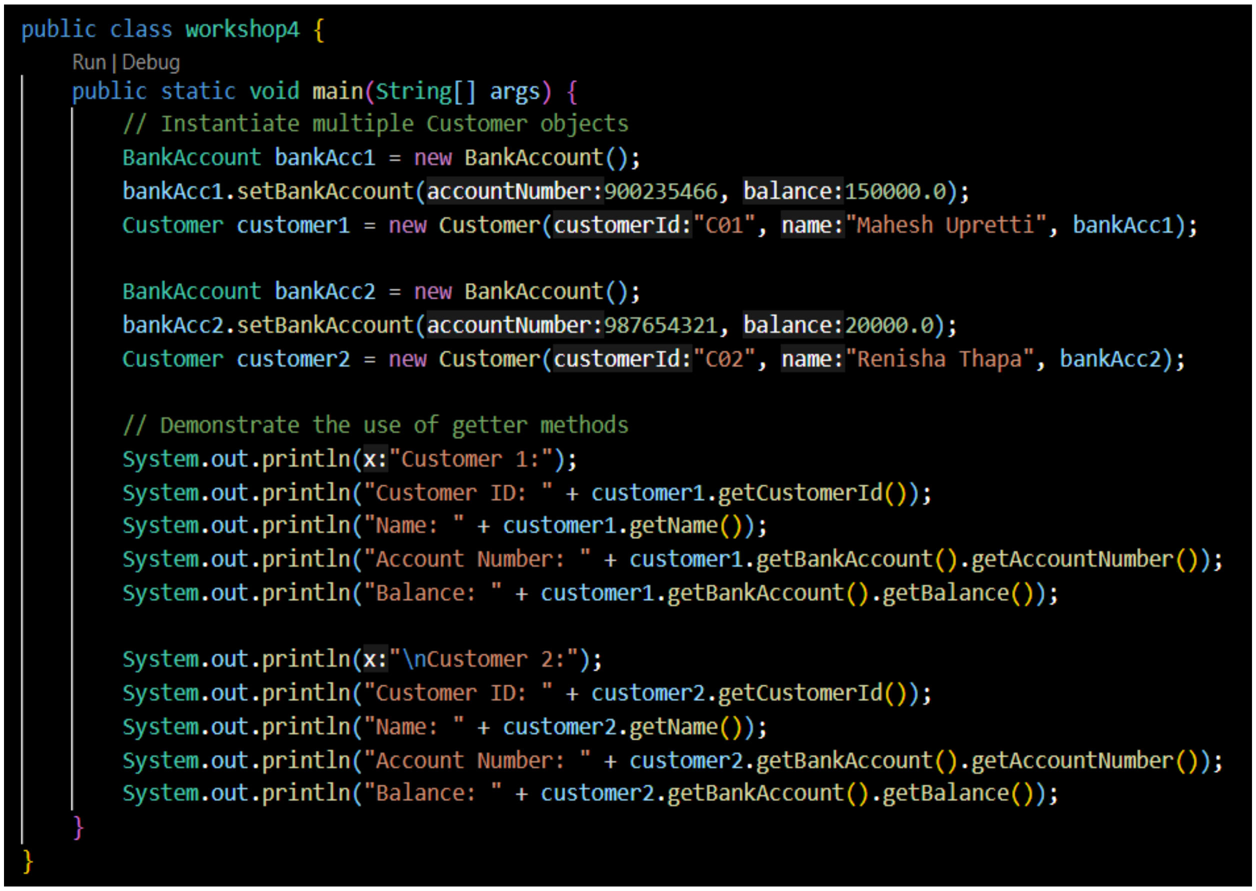
```
Street: Grandview
City: Westland
Zipcode: 488186
```

Combining Concepts:

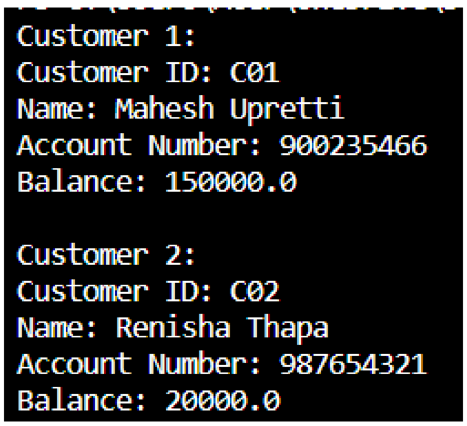
10. Create a class Customer with private attributes customerId, name, and a BankAccount attributes. Implement a parameterized constructor and encapsulate the attributes. Provide getter method. Instantiate multiple Customer objects with different values and demonstrate the use of getters and setters.



```
class BankAccount {
    private int accountNumber;
    private double balance;

    void setBankAccount(int accountNumber, double balance){
        this.accountNumber = accountNumber;
        this.balance = balance;
    }

    // Getter methods for BankAccount
    public int getAccountNumber() {
        return accountNumber;
    }
    public double getBalance() {
        return balance;
    }
}
 class Customer {
    private String customerId;
    private String name;
    private BankAccount bankAccount;

    public Customer(String customerId, String name, BankAccount bankAccount) {
        this.customerId = customerId;
        this.name = name;
        this.bankAccount = bankAccount;
    }

    // Getter methods for Customer
    public String getCustomerId() {
        return customerId;
    }
    public String getName() {
        return name;
    }
    public BankAccount getBankAccount() {
        return bankAccount;
    }
}
```



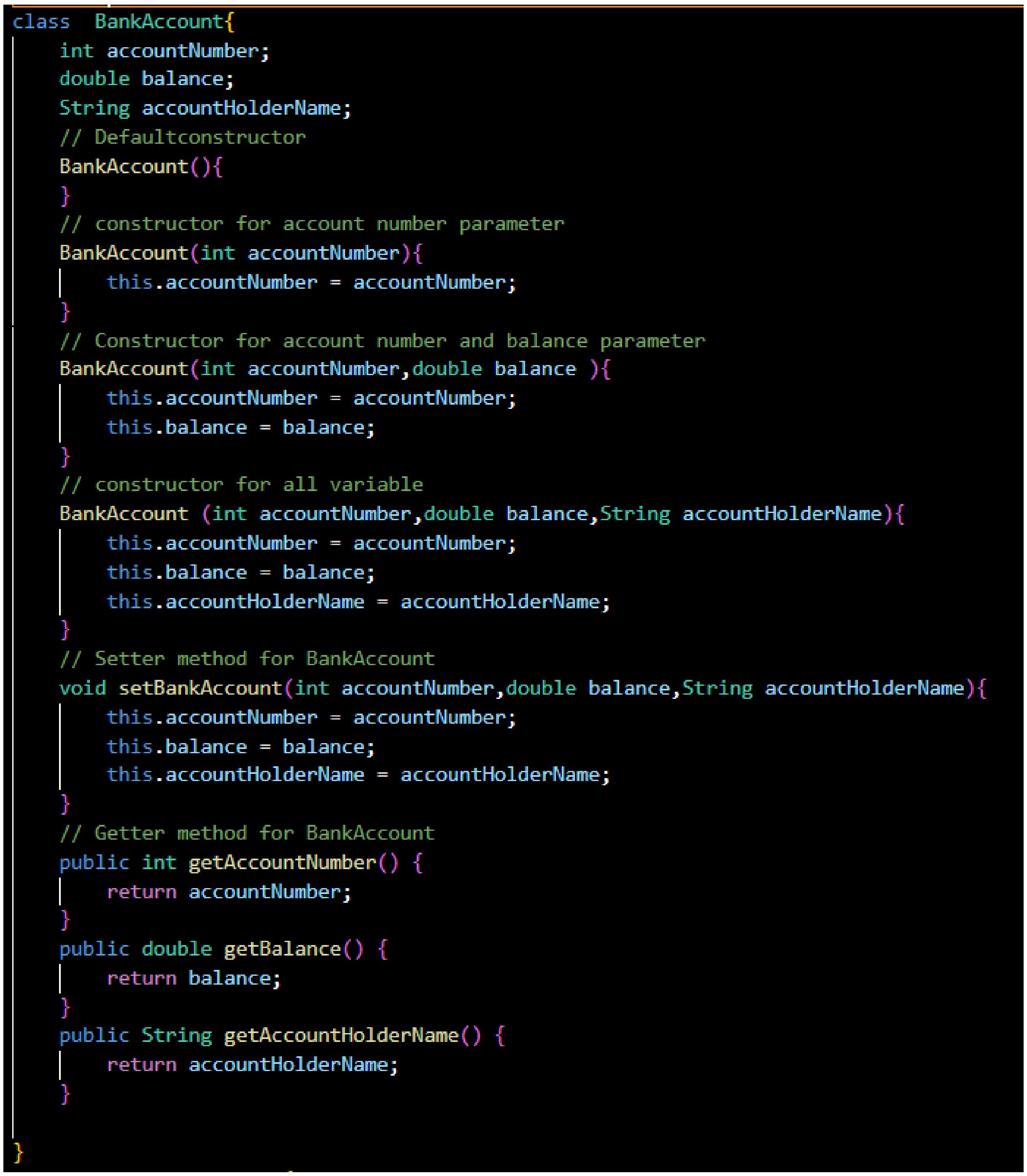
```java
public class workshop4 {
    Run | Debug
    public static void main(String[] args) {
        // Instantiate multiple Customer objects
        BankAccount bankAcc1 = new BankAccount();
        bankAcc1.setBankAccount(accountNumber:900235466, balance:150000.0);
        Customer customer1 = new Customer(customerId:"C01", name:"Mahesh Upretti", bankAcc1);

        BankAccount bankAcc2 = new BankAccount();
        bankAcc2.setBankAccount(accountNumber:987654321, balance:20000.0);
        Customer customer2 = new Customer(customerId:"C02", name:"Renisha Thapa", bankAcc2);

        // Demonstrate the use of getter methods
        System.out.println(x:"Customer 1:");
        System.out.println("Customer ID: " + customer1.getCustomerId());
        System.out.println("Name: " + customer1.getName());
        System.out.println("Account Number: " + customer1.getBankAccount().getAccountNumber());
        System.out.println("Balance: " + customer1.getBankAccount().getBalance());

        System.out.println(x:"\nCustomer 2:");
        System.out.println("Customer ID: " + customer2.getCustomerId());
        System.out.println("Name: " + customer2.getName());
        System.out.println("Account Number: " + customer2.getBankAccount().getAccountNumber());
        System.out.println("Balance: " + customer2.getBankAccount().getBalance());
    }
}
```



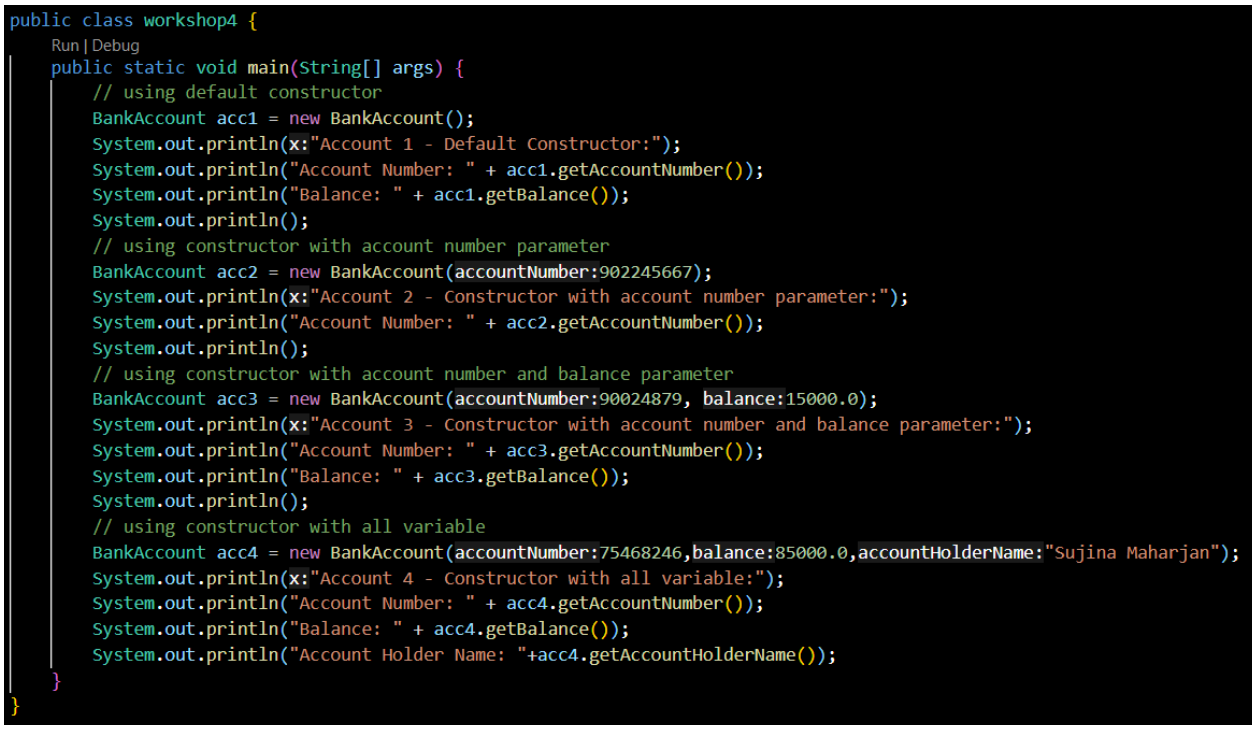
```
Customer 1:
Customer ID: C01
Name: Mahesh Upretti
Account Number: 900235466
Balance: 150000.0

Customer 2:
Customer ID: C02
Name: Renisha Thapa
Account Number: 987654321
Balance: 20000.0
```

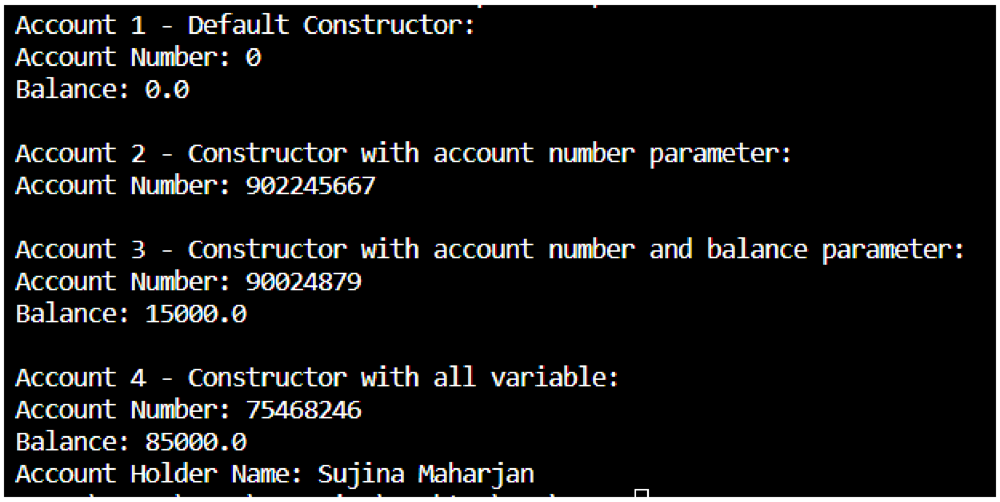
Constructors and Overloading:

11. Implement multiple constructors for the BankAccount class with different parameter sets. Use constructor overloading to create objects with different initialization scenarios.



```
class  BankAccount{
    int accountNumber;
    double balance;
    String accountHolderName;
    // Defaultconstructor
    BankAccount(){
    }
    // constructor for account number parameter
    BankAccount(int accountNumber){
        this.accountNumber = accountNumber;
    }
    // Constructor for account number and balance parameter
    BankAccount(int accountNumber,double balance ){
        this.accountNumber = accountNumber;
        this.balance = balance;
    }
    // constructor for all variable
    BankAccount (int accountNumber,double balance,String accountHolderName){
        this.accountNumber = accountNumber;
        this.balance = balance;
        this.accountHolderName = accountHolderName;
    }
    // Setter method for BankAccount
    void setBankAccount(int accountNumber,double balance,String accountHolderName){
        this.accountNumber = accountNumber;
        this.balance = balance;
        this.accountHolderName = accountHolderName;
    }
    // Getter method for BankAccount
    public int getAccountNumber() {
        return accountNumber;
    }
    public double getBalance() {
        return balance;
    }
    public String getAccountHolderName() {
        return accountHolderName;
    }

}
```


```
public class workshop4 {
    Run | Debug
    public static void main(String[] args) {
        // using default constructor
        BankAccount acc1 = new BankAccount();
        System.out.println(x:"Account 1 - Default Constructor:");
        System.out.println("Account Number: " + acc1.getAccountNumber());
        System.out.println("Balance: " + acc1.getBalance());
        System.out.println();
        // using constructor with account number parameter
        BankAccount acc2 = new BankAccount(accountNumber:902245667);
        System.out.println(x:"Account 2 - Constructor with account number parameter:");
        System.out.println("Account Number: " + acc2.getAccountNumber());
        System.out.println();
        // using constructor with account number and balance parameter
        BankAccount acc3 = new BankAccount(accountNumber:90024879, balance:15000.0);
        System.out.println(x:"Account 3 - Constructor with account number and balance parameter:");
        System.out.println("Account Number: " + acc3.getAccountNumber());
        System.out.println("Balance: " + acc3.getBalance());
        System.out.println();
        // using constructor with all variable
        BankAccount acc4 = new BankAccount(accountNumber:75468246,balance:85000.0,accountHolderName:"Sujina Maharjan");
        System.out.println(x:"Account 4 - Constructor with all variable:");
        System.out.println("Account Number: " + acc4.getAccountNumber());
        System.out.println("Balance: " + acc4.getBalance());
        System.out.println("Account Holder Name: "+acc4.getAccountHolderName());
    }
}
```


```
Account 1 - Default Constructor:
Account Number: 0
Balance: 0.0

Account 2 - Constructor with account number parameter:
Account Number: 902245667

Account 3 - Constructor with account number and balance parameter:
Account Number: 90024879
Balance: 15000.0

Account 4 - Constructor with all variable:
Account Number: 75468246
Balance: 85000.0
Account Holder Name: Sujina Maharjan
```