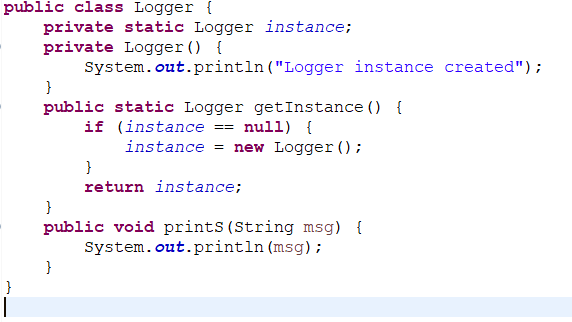
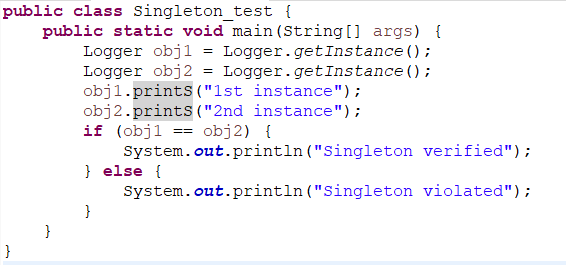
**Exercise 1: Implementing the Singleton Pattern**

**Solution:**

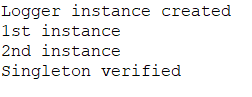
Logger Class:



Test Class:



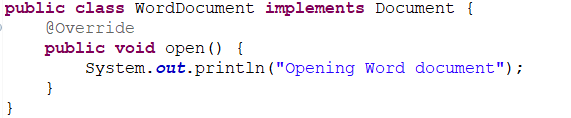
Output:

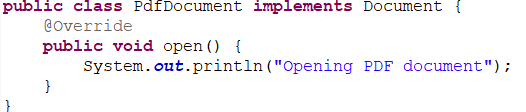


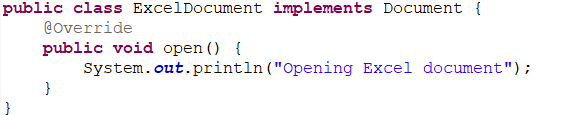
**Exercise 2: Implementing the Factory Method Pattern**

**Solution:**

Document Classes:



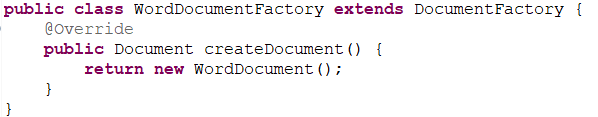


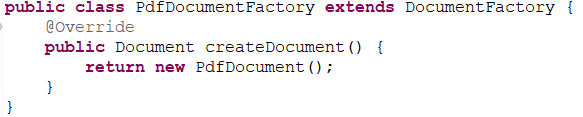


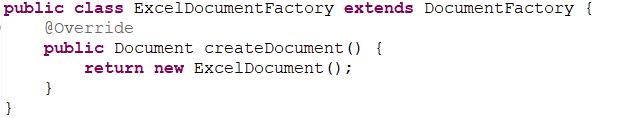


Concrete Factory Classes:

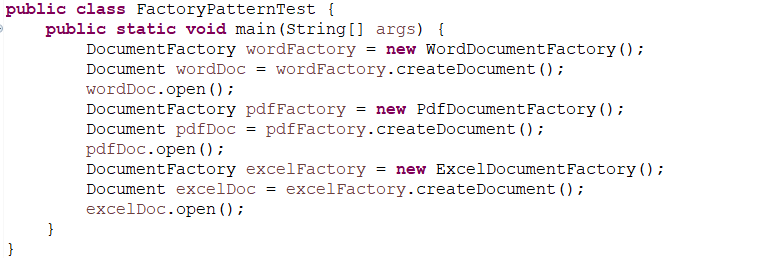




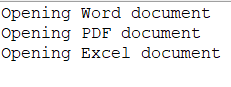




Test Class:



OUTPUT:



**Exercise 3: Implementing the Builder Pattern**

**Solution:**

Computer Class:

**public** **class** Computer {

**private** **final** String cpu;

**private** **final** String ram;

**private** **final** String storage;

**private** **final** String graphicsCard;

**private** **final** **boolean** isWifiEnabled;

**private** **final** **boolean** isBluetoothEnabled;

**private** Computer(Builder builder) {

**this**.cpu = builder.cpu;

**this**.ram = builder.ram;

**this**.storage = builder.storage;

**this**.graphicsCard = builder.graphicsCard;

**this**.isWifiEnabled = builder.isWifiEnabled;

**this**.isBluetoothEnabled = builder.isBluetoothEnabled;

}

**public** **static** **class** Builder {

**private** String cpu;

**private** String ram;

**private** String storage;

**private** String graphicsCard;

**private** **boolean** isWifiEnabled;

**private** **boolean** isBluetoothEnabled;

**public** Builder setCpu(String cpu) {

**this**.cpu = cpu;

**return** **this**;

}

**public** Builder setRam(String ram) {

**this**.ram = ram;

**return** **this**;

}

**public** Builder setStorage(String storage) {

**this**.storage = storage;

**return** **this**;

}

**public** Builder setGraphicsCard(String graphicsCard) {

**this**.graphicsCard = graphicsCard;

**return** **this**;

}

**public** Builder setWifiEnabled(**boolean** wifiEnabled) {

**this**.isWifiEnabled = wifiEnabled;

**return** **this**;

}

**public** Builder setBluetoothEnabled(**boolean** bluetoothEnabled) {

**this**.isBluetoothEnabled = bluetoothEnabled;

**return** **this**;

}

**public** Computer build() {

**return** **new** Computer(**this**);

}

}

@Override

**public** String toString() {

**return** "Computer{" +

"cpu='" + cpu + '\'' +

", ram='" + ram + '\'' +

", storage='" + storage + '\'' +

", graphicsCard='" + graphicsCard + '\'' +

", isWifiEnabled=" + isWifiEnabled +

", isBluetoothEnabled=" + isBluetoothEnabled +

'}';

}

}

Test Class:

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Computer basicComputer = **new** Computer.Builder()

.setCpu("Ryzen 5")

.setRam("4GB")

.setStorage("512GB")

.build();

System.***out***.println("Basic Configuration:");

System.***out***.println(basicComputer);

Computer gamingComputer = **new** Computer.Builder()

.setCpu("AMD Ryzen 5000")

.setRam("64GB")

.setStorage("512GB SSD")

.setGraphicsCard("NVIDIA GEFORCE GTX")

.setWifiEnabled(**true**)

.setBluetoothEnabled(**true**)

.build();

System.***out***.println();

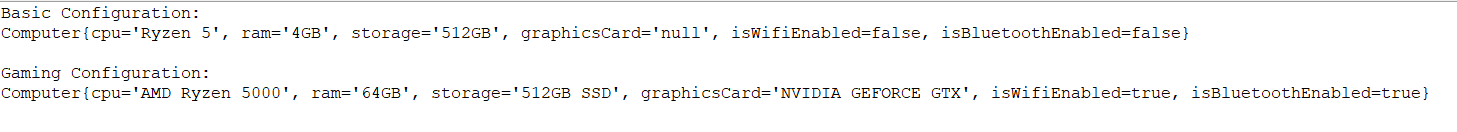
System.***out***.println("Gaming Configuration:");

System.***out***.println(gamingComputer);

}

}

OUTPUT:



**Exercise 4: Implementing the Adapter Pattern**

**Solution:**

Payment Processor interface:

**public** **interface** PaymentProcessor {

**void** processPayment(**double** amount);

}

Payment Gateways Classes:

**public** **class** PhonePe {

**public** **void** makePhonePePayment(**double** amount) {

System.***out***.println("Paid ₹" + amount + " using PhonePe.");

}

}

**public** **class** GPay {

**public** **void** payUsingGPay(**double** amount) {

System.***out***.println("Paid ₹" + amount + " using GPay.");

}

}

Adapter Classes:

**public** **class** GPayAdapter **implements** PaymentProcessor {

**private** GPay gpay;

**public** GPayAdapter(GPay gpay) {

**this**.gpay = gpay;

}

@Override

**public** **void** processPayment(**double** amount) {

gpay.payUsingGPay(amount);

}

}

**public** **class** PhonePeAdapter **implements** PaymentProcessor {

**private** PhonePe phonePe;

**public** PhonePeAdapter(PhonePe phonePe) {

**this**.phonePe = phonePe;

}

@Override

**public** **void** processPayment(**double** amount) {

phonePe.makePhonePePayment(amount);

}

}

Test Class:

**public** **class** test {

**public** **static** **void** main(String[] args) {

GPay gpay = **new** GPay();

PaymentProcessor gpayProcessor = **new** GPayAdapter(gpay);

gpayProcessor.processPayment(500.0);

PhonePe phonePe = **new** PhonePe();

PaymentProcessor phonePeProcessor = **new** PhonePeAdapter(phonePe);

phonePeProcessor.processPayment(1000.0);

}

}

OUTPUT:

]

**Exercise 5: Implementing the Decorator Pattern**

**Solution:**

Notifier Interface:

**public** **interface** Notifier {

**void** send(String message);

}

Concrete Component:

**public** **class** EmailNotifier **implements** Notifier {

@Override

**public** **void** send(String message) {

System.***out***.println("Sending Email: " + message);

}

}

Decorator Class:

**public** **abstract** **class** NotifierDecorator **implements** Notifier {

**protected** Notifier wrappedNotifier;

**public** NotifierDecorator(Notifier notifier) {

**this**.wrappedNotifier = notifier;

}

@Override

**public** **void** send(String message) {

wrappedNotifier.send(message);

}

}

Notifier Classes with reference to notifier object:

**public** **class** SlackNotifier **extends** NotifierDecorator {

**public** SlackNotifier(Notifier notifier) {

**super**(notifier);

}

@Override

**public** **void** send(String message) {

**super**.send(message);

sendSlack(message);

}

**private** **void** sendSlack(String message) {

System.***out***.println("Sending Slack Message: " + message);

}

}

**public** **class** SMSNotifierDecorator **extends** NotifierDecorator {

**public** SMSNotifierDecorator(Notifier notifier) {

**super**(notifier);

}

@Override

**public** **void** send(String message) {

**super**.send(message);

sendSMS(message);

}

**private** **void** sendSMS(String message) {

System.***out***.println("Sending SMS: " + message);

}

}

Test Class:

**public** **class** test {

**public** **static** **void** main(String[] args) {

Notifier emailNotifier = **new** EmailNotifier();

Notifier smsEmailNotifier = **new** SMSNotifierDecorator(emailNotifier);

Notifier fullNotifier = **new** SlackNotifier(smsEmailNotifier);

fullNotifier.send("Server is down!");

}

}

OUTPUT:

