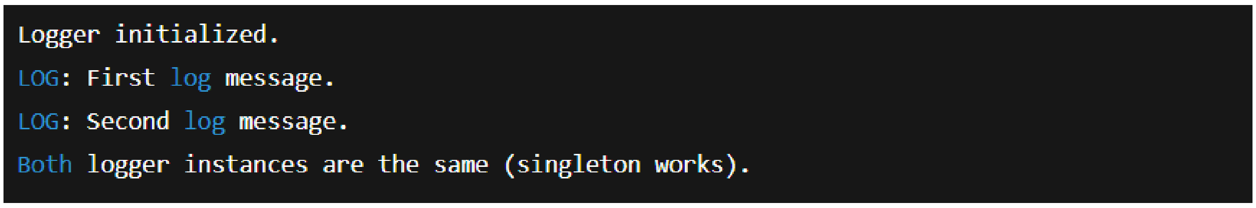
OUTPUTS – SOLID Principles

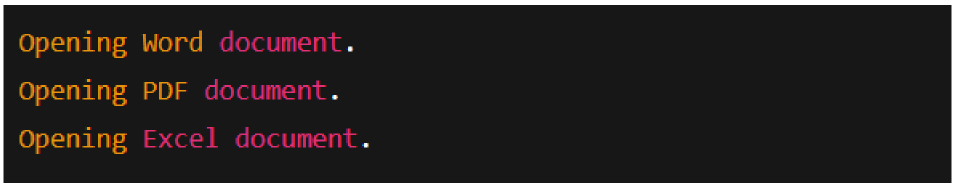
Exercise 1 :



```
Logger initialized.
LOG: First log message.
LOG: Second log message.
Both logger instances are the same (singleton works).
```

Implemented the Singleton pattern by creating a *Logger* class that restricts instantiation to a single object. The constructor is made private, and a public static *getInstance()* method ensures the same instance is returned every time. A simple *log()* method demonstrates usage, and the *Main* class confirms that multiple calls return the same Logger instance.

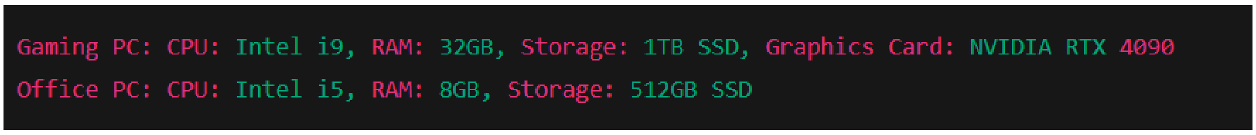
Exercise 2 :



```
Opening Word document.
Opening PDF document.
Opening Excel document.
```

Implemented the Factory Method pattern to create different types of documents (Word, PDF, Excel) without exposing the instantiation logic. A common *Document* interface defines a method *open()*, and each document type has its own class. Separate factory classes implement *createDocument()* to return the correct document type. The *Main* class demonstrates how the pattern enables easy creation of various document types through their respective factories.

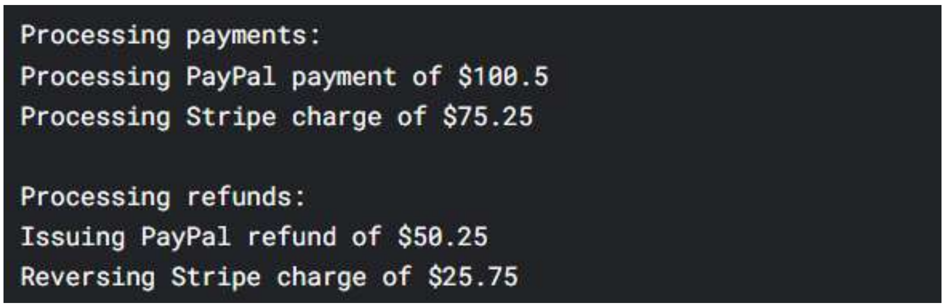
Exercise 3 :



```
Gaming PC: CPU: Intel i9, RAM: 32GB, Storage: 1TB SSD, Graphics Card: NVIDIA RTX 4090
Office PC: CPU: Intel i5, RAM: 8GB, Storage: 512GB SSD
```

Implemented the Builder pattern by creating a *Computer* class with a nested *Builder* class to construct different configurations step-by-step while keeping the construction flexible and readable.

Exercise 4:



```
Processing payments:
Processing PayPal payment of $100.5
Processing Stripe charge of $75.25

Processing refunds:
Issuing PayPal refund of $50.25
Reversing Stripe charge of $25.75
```

Used the Adapter Pattern to allow a unified interface (*PaymentProcessor*) to communicate with different third-party payment gateways (*PayPalGateway*, *StripeGateway*) by wrapping them with corresponding adapters.

Exercise 5 :

```
Sending Email Notification
----
Sending Email Notification
Sending SMS Notification
----
Sending Email Notification
Sending SMS Notification
Sending Slack Notification
```

Implemented the Decorator Pattern by defining a base *Notifier* interface and decorating it dynamically with *SMSNotifierDecorator* and *SlackNotifierDecorator*, allowing flexible combination of notification channels at runtime.

Exercise 6 :

```
Loading image from remote server: nature.jpg
Displaying image: nature.jpg
----
Displaying image: nature.jpg
```

Implemented the Proxy Pattern by creating a *ProxyImage* class that controls access to a *RealImage*. The proxy delays the creation of *RealImage* until the *display()* method is called, enabling lazy loading and caching. This ensures the image is loaded from the remote server only once, even if displayed multiple times.

Exercise 7:

```
Mobile App - New Stock Price: 150.75
Web App - New Stock Price: 150.75
----
Mobile App - New Stock Price: 160.25
Web App - New Stock Price: 160.25
```

Implemented the Observer pattern by creating a *StockMarket* class that notifies registered observers (*MobileApp*, *WebApp*) whenever stock prices change, allowing real-time updates to multiple clients.

Exercise 8 :

```
Paying $100.0 using Credit Card.
----
Paying $250.5 using PayPal.
```

Implemented the Strategy pattern by defining a *PaymentStrategy* interface and separate strategy classes for *CreditCardPayment* and *PayPalPayment*. The *PaymentContext* class uses these strategies at runtime to process payments flexibly.

Exercise 9:

```
Light is ON
----
Light is OFF
```

Implemented the Command Pattern by defining a Command interface and concrete commands (*LightOnCommand*, *LightOffCommand*) that operate on a Light receiver. The *RemoteControl* class acts as the invoker, allowing commands to be set and executed dynamically.

Exercise 10:

```
Student Details:
Name: John Doe
ID: 101
Grade: A
----
Student Details:
Name: Jane Smith
ID: 101
Grade: B+
```

Implemented the MVC pattern by separating the student data (*Model*), its presentation (*View*), and the logic to update and retrieve data (*Controller*). The controller updates the model and refreshes the view accordingly.

Exercise 11:

```
Customer Found: Customer [ID: C101, Name: Alice]
```

Implemented Dependency Injection using constructor injection by passing a *CustomerRepositoryImpl* to *CustomerService*. This decouples the service from the repository implementation, making the system more modular and testable.