Exercise 1: Problem Statement on Design patterns

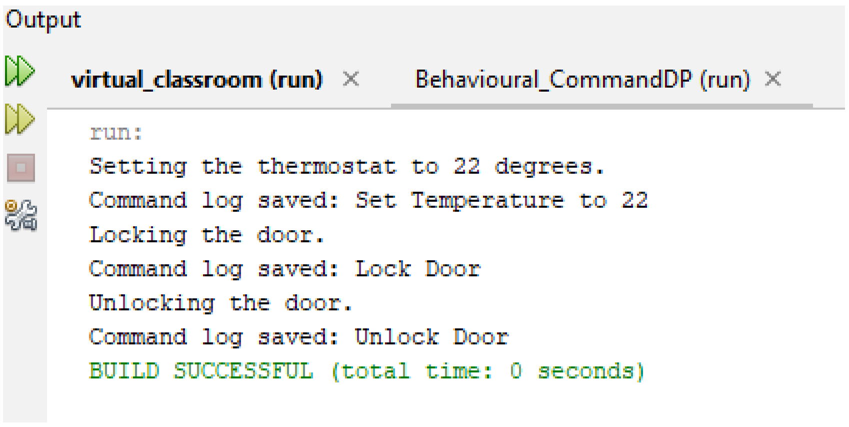
Two use cases to demonstrate two Behavioural Design Pattern.

Command Design Pattern -> Smart Home System

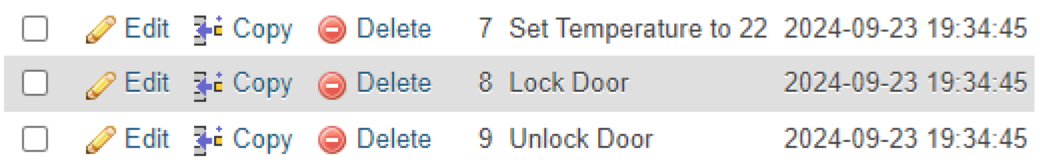
I have simulate a smart home system where the homeowner can issue commands to control devices such as the thermostat or door lock.

Use Case:

- Homeowner can issue commands to set the temperature or lock/unlock the door.
- A **Command Pattern** will be used to encapsulate requests into objects that can be executed or stored.
- The system logs command executions in the database.



```
run:
Setting the thermostat to 22 degrees.
Command log saved: Set Temperature to 22
Locking the door.
Command log saved: Lock Door
Unlocking the door.
Command log saved: Unlock Door
BUILD SUCCESSFUL (total time: 0 seconds)
```



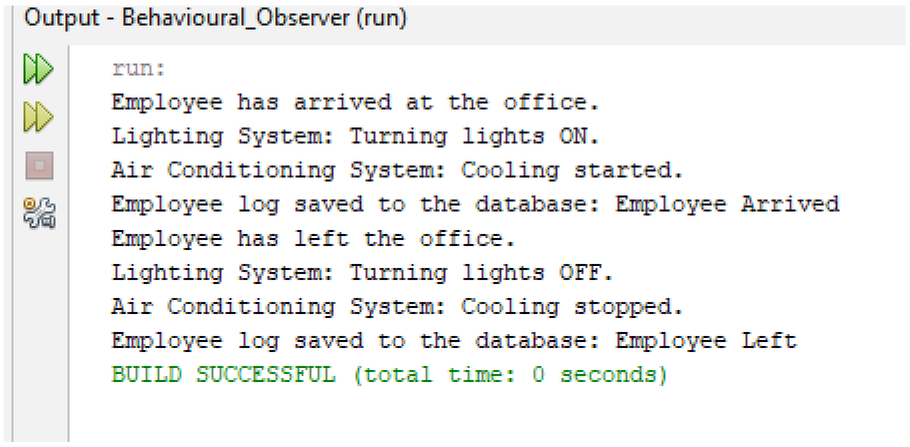
☐	Edit	Copy	Delete	7	Set Temperature to 22	2024-09-23 19:34:45
☐	Edit	Copy	Delete	8	Lock Door	2024-09-23 19:34:45
☐	Edit	Copy	Delete	9	Unlock Door	2024-09-23 19:34:45

Observer Design Pattern -> Smart Office Facility

I have simulate a smart office environment where different systems (e.g., lighting, air conditioning, and security) get notified when certain conditions are met, such as an employee arriving or leaving. The Observer Pattern allows the systems to automatically react to changes in the office.

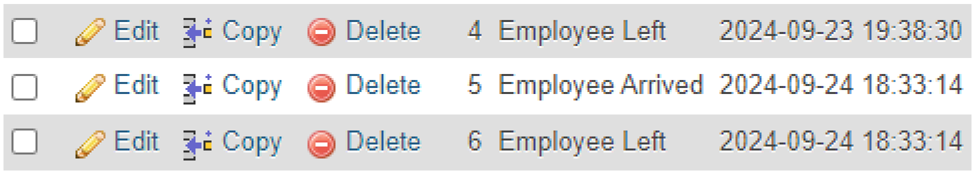
Use Case:

- **LightingSystem** and **AirConditioningSystem** adjust their settings when an employee arrives or leaves the office.
- **Database** stores the logs of employee entry and exit events.



Output - Behavioural_Observer (run)

```
run:
Employee has arrived at the office.
Lighting System: Turning lights ON.
Air Conditioning System: Cooling started.
Employee log saved to the database: Employee Arrived
Employee has left the office.
Lighting System: Turning lights OFF.
Air Conditioning System: Cooling stopped.
Employee log saved to the database: Employee Left
BUILD SUCCESSFUL (total time: 0 seconds)
```



☐	Edit	Copy	Delete	4	Employee Left	2024-09-23 19:38:30
☐	Edit	Copy	Delete	5	Employee Arrived	2024-09-24 18:33:14
☐	Edit	Copy	Delete	6	Employee Left	2024-09-24 18:33:14

Two use cases to demonstrate two Creational Design Pattern.

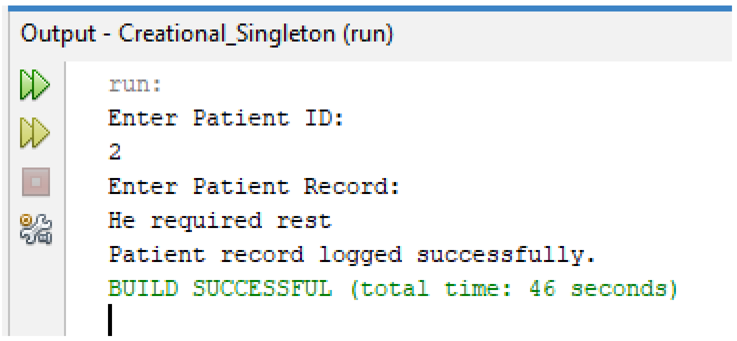
Singleton Design Pattern –> Medical System Database Connection

Use Case: Medical System – Shared Patient Database Connection

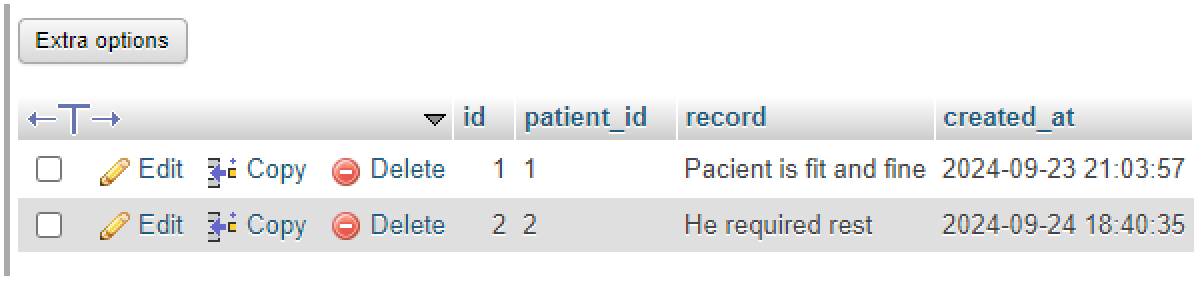
In a hospital or medical system, all patient records need to be accessed and updated frequently. Using a **Singleton pattern**, we can ensure that the system maintains a **single shared MySQL database connection** for accessing patient information, avoiding redundant database connections.

Example Scenario:

A hospital's medical system needs to log patient data (such as vital signs, diagnosis, and treatment records) from different departments like emergency, radiology, and cardiology. Using the Singleton pattern, the system ensures that all departments use the same shared database connection.



```
run:
Enter Patient ID:
2
Enter Patient Record:
He required rest
Patient record logged successfully.
BUILD SUCCESSFUL (total time: 46 seconds)
```



	id	patient_id	record	created_at
Edit Copy Delete	1	1	Pacient is fit and fine	2024-09-23 21:03:57
Edit Copy Delete	2	2	He required rest	2024-09-24 18:40:35

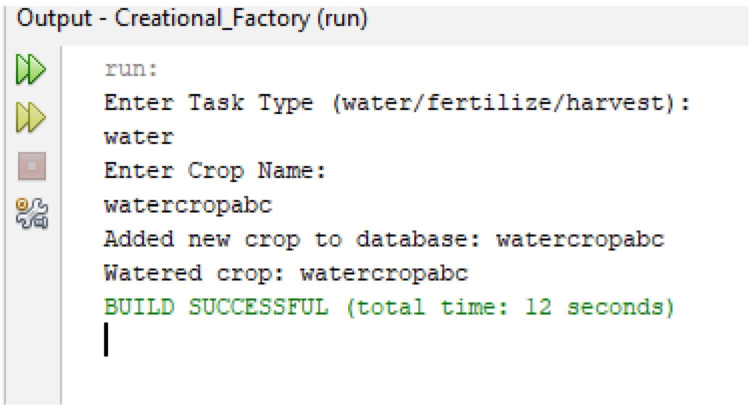
Factory Design Pattern –> Agricultural Crop Management

Use Case: Agricultural System – Task Management for Crop Maintenance

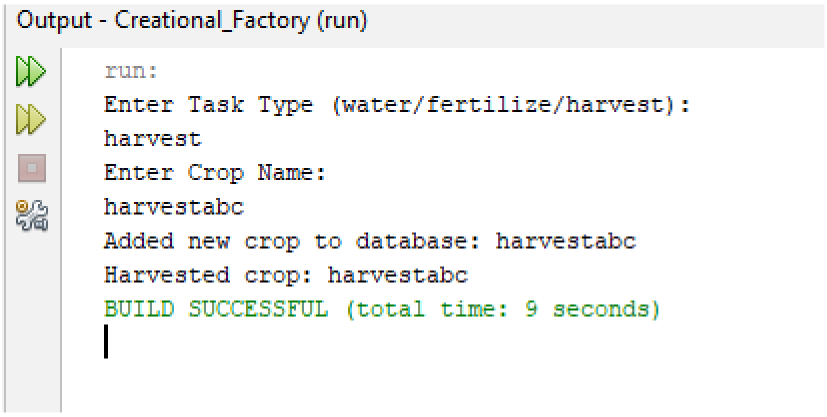
In a large-scale farming operation, different tasks such as watering, fertilizing, and harvesting need to be scheduled and performed at different intervals. Using the **Factory pattern**, you can dynamically generate task executors (watering, fertilizing, etc.) based on the type of task needed for the crop management system.

Example Scenario:

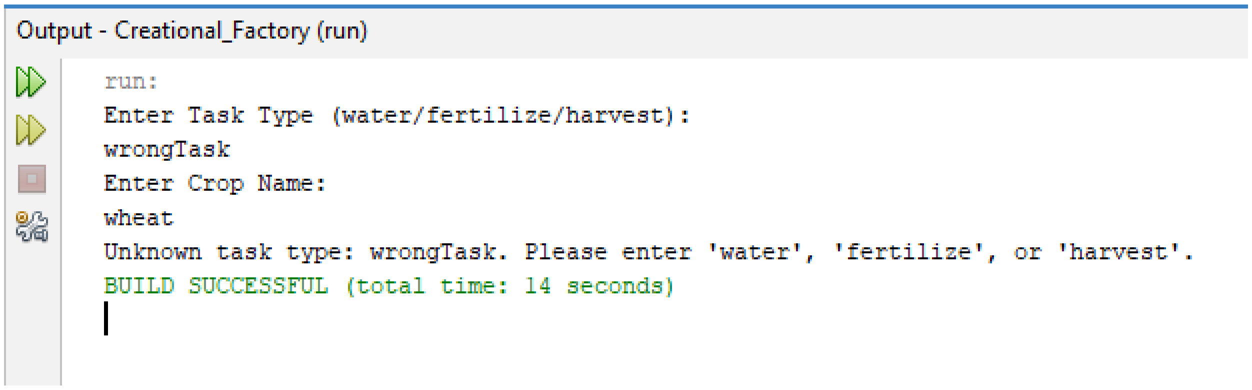
A smart agricultural system monitors crops and automates tasks such as irrigation, fertilization, and pesticide spraying. The system uses the Factory pattern to generate task executors dynamically depending on the type of crop maintenance required.


Output - Creational_Factory (run)

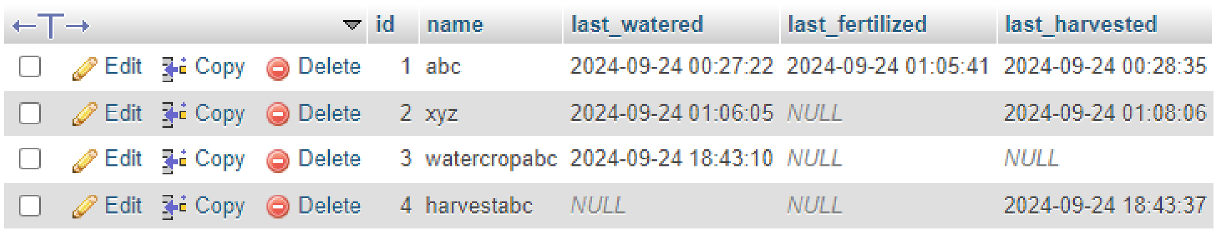
```
run:
Enter Task Type (water/fertilize/harvest):
water
Enter Crop Name:
watercropabc
Added new crop to database: watercropabc
Watered crop: watercropabc
BUILD SUCCESSFUL (total time: 12 seconds)
```


Output - Creational_Factory (run)

```
run:
Enter Task Type (water/fertilize/harvest):
harvest
Enter Crop Name:
harvestabc
Added new crop to database: harvestabc
Harvested crop: harvestabc
BUILD SUCCESSFUL (total time: 9 seconds)
```


Output - Creational_Factory (run)

```
run:
Enter Task Type (water/fertilize/harvest):
wrongTask
Enter Crop Name:
wheat
Unknown task type: wrongTask. Please enter 'water', 'fertilize', or 'harvest'.
BUILD SUCCESSFUL (total time: 14 seconds)
```



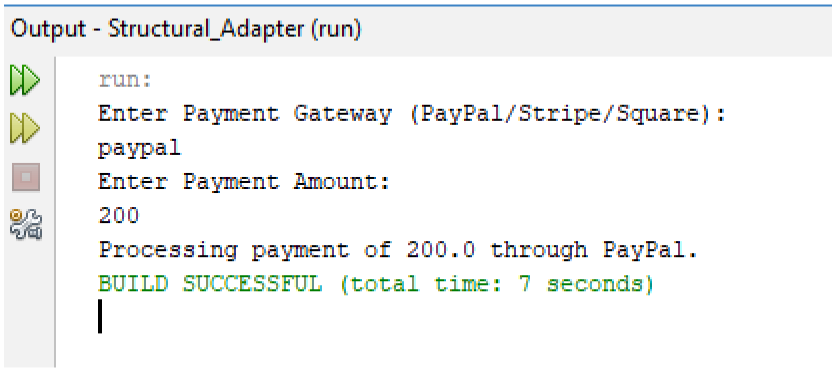
←T→	id	name	last_watered	last_fertilized	last_harvested
☐ Edit Copy Delete	1	abc	2024-09-24 00:27:22	2024-09-24 01:05:41	2024-09-24 00:28:35
☐ Edit Copy Delete	2	xyz	2024-09-24 01:06:05	*NULL*	2024-09-24 01:08:06
☐ Edit Copy Delete	3	watercropabc	2024-09-24 18:43:10	*NULL*	*NULL*
☐ Edit Copy Delete	4	harvestabc	*NULL*	*NULL*	2024-09-24 18:43:37

Two use cases to demonstrate two Structural Design Pattern.

Adapter Pattern -> Fintech Payment System

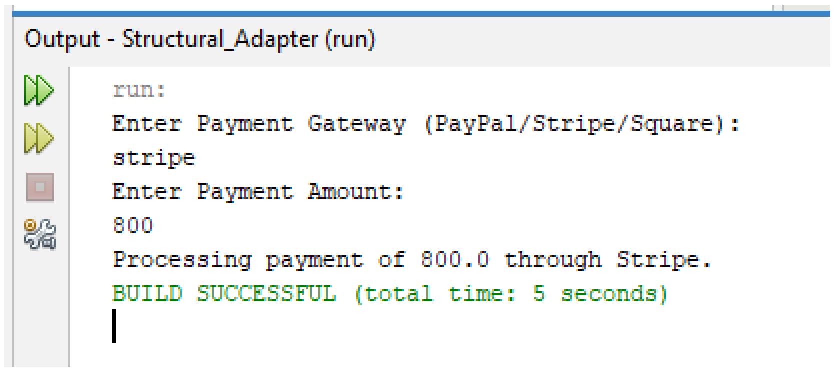
Scenario:

I am building a **Fintech Payment System** that supports multiple payment gateways, each with different APIs. The system adapts these APIs to a standard format for processing payments.



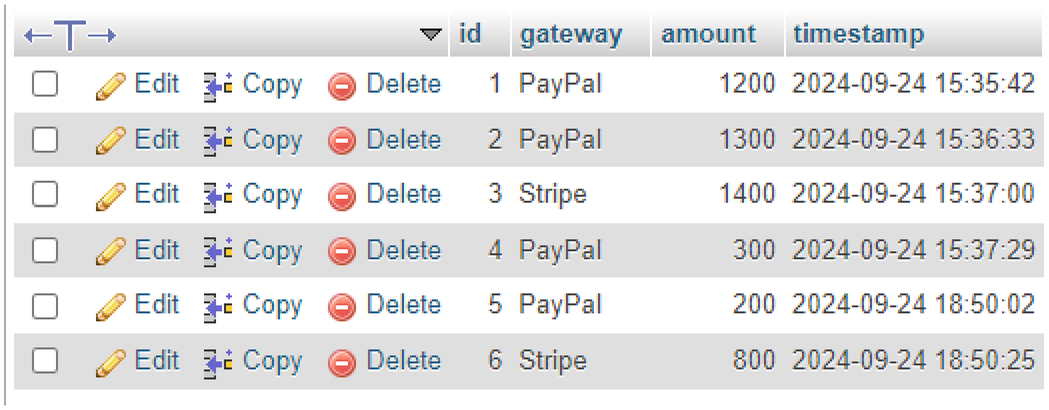
Output - Structural_Adapter (run)

```
run:
Enter Payment Gateway (PayPal/Stripe/Square):
paypal
Enter Payment Amount:
200
Processing payment of 200.0 through PayPal.
BUILD SUCCESSFUL (total time: 7 seconds)
```



Output - Structural_Adapter (run)

```
run:
Enter Payment Gateway (PayPal/Stripe/Square):
stripe
Enter Payment Amount:
800
Processing payment of 800.0 through Stripe.
BUILD SUCCESSFUL (total time: 5 seconds)
```

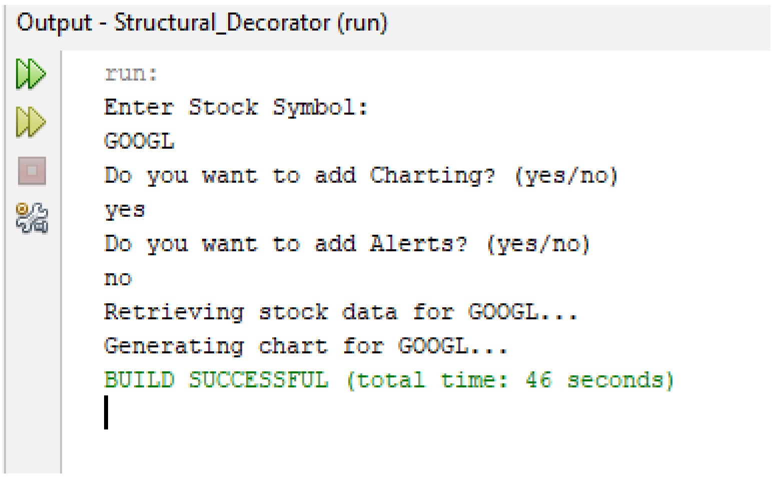


	id	gateway	amount	timestamp
Edit Copy Delete	1	PayPal	1200	2024-09-24 15:35:42
Edit Copy Delete	2	PayPal	1300	2024-09-24 15:36:33
Edit Copy Delete	3	Stripe	1400	2024-09-24 15:37:00
Edit Copy Delete	4	PayPal	300	2024-09-24 15:37:29
Edit Copy Delete	5	PayPal	200	2024-09-24 18:50:02
Edit Copy Delete	6	Stripe	800	2024-09-24 18:50:25

Decorator Pattern -> Stock Market Analysis System

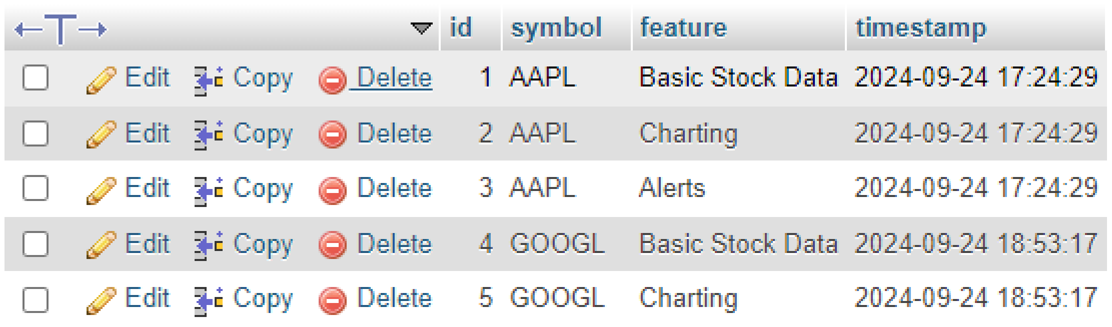
Scenario:

I am building a **Stock Market Analysis System** where the base functionality retrieves stock prices. Additional features, like charting and alerts, are dynamically added using the **Decorator Pattern**.



Output - Structural_Decorator (run)

```
run:
Enter Stock Symbol:
GOOGL
Do you want to add Charting? (yes/no)
yes
Do you want to add Alerts? (yes/no)
no
Retrieving stock data for GOOGL...
Generating chart for GOOGL...
BUILD SUCCESSFUL (total time: 46 seconds)
```



	id	symbol	feature	timestamp
Edit Copy Delete	1	AAPL	Basic Stock Data	2024-09-24 17:24:29
Edit Copy Delete	2	AAPL	Charting	2024-09-24 17:24:29
Edit Copy Delete	3	AAPL	Alerts	2024-09-24 17:24:29
Edit Copy Delete	4	GOOGL	Basic Stock Data	2024-09-24 18:53:17
Edit Copy Delete	5	GOOGL	Charting	2024-09-24 18:53:17