- **2 Behavioral Patterns**
- **2 Creational Patterns**
- **2 Structural Patterns**

Exercise 1: Design Pattern Demonstrations

◇ Behavioral Design Patterns

1. Observer Pattern – Task Notification

Use case: Whenever a new task is added to the schedule, all observers (e.g., astronauts, loggers) get notified.

```
// Observer interface
interface TaskObserver {
    void onTaskAdded(String task);
}

// Concrete observer
class ConsoleObserver implements TaskObserver {
    public void onTaskAdded(String task) {
        System.out.println("Observer: New task added -> " + task);
    }
}

// Subject
class TaskManager {
    private List<TaskObserver> observers = new ArrayList<>();
    void addObserver(TaskObserver obs) { observers.add(obs); }
    void addTask(String task) {
        System.out.println("Task added: " + task);
        for (TaskObserver o : observers) o.onTaskAdded(task);
    }
}
```

Demo:

```
TaskManager manager = new TaskManager();
manager.addObserver(new ConsoleObserver());
manager.addTask("Morning Exercise");
```

2. Strategy Pattern – Sorting Tasks

Use case: Sort tasks either by **start time** or **priority** dynamically.

```
interface SortStrategy {
    List<String> sort(List<String> tasks);
}

class SortAlphabetically implements SortStrategy {
    public List<String> sort(List<String> tasks) {
        tasks.sort(String::compareTo);
        return tasks;
    }
}

class SortByLength implements SortStrategy {
    public List<String> sort(List<String> tasks) {
        tasks.sort(Comparator.comparingInt(String::length));
        return tasks;
    }
}
```

Demo:

```
List<String> tasks = new ArrayList<>(List.of("Meeting", "Exercise",
"Lunch"));
SortStrategy strategy = new SortAlphabetically();
System.out.println("Alphabetical: " + strategy.sort(tasks));
strategy = new SortByLength();
System.out.println("By Length: " + strategy.sort(tasks));
```

◇ Creational Design Patterns

3. Singleton Pattern – Config Manager

Use case: Ensure only one configuration object exists (e.g., app logger).

```
class ConfigManager {
    private static ConfigManager instance;
    private ConfigManager() {}
    public static synchronized ConfigManager getInstance() {
        if (instance == null) instance = new ConfigManager();
        return instance;
    }
    public void log(String msg) { System.out.println("[LOG] " +
msg); }
}
```

Demo:

```
ConfigManager.getInstance().log("Scheduler started...");
```

4. Factory Pattern – Task Creator

Use case: Centralized creation of Task objects with validation.

```
class Task {
    String desc, priority;
    public Task(String d, String p){ desc=d; priority=p; }
    public String toString(){ return desc + " [" + priority + "]"; }
}

class TaskFactory {
    public static Task create(String desc, String priority) {
        return new Task(desc, priority);
    }
```

```
}
```

Demo:

```
Task t1 = TaskFactory.create("Team Meeting", "High");
System.out.println(t1);
```

◇ Structural Design Patterns

5. Adapter Pattern – Time Adapter

Use case: Convert string times into `LocalTime`.

```
import java.time.LocalTime;

interface TimeParser {
    LocalTime parse(String t);
}

class StringToLocalTimeAdapter implements TimeParser {
    public LocalTime parse(String t) { return LocalTime.parse(t); }
}
```

Demo:

```
TimeParser parser = new StringToLocalTimeAdapter();
System.out.println("Parsed: " + parser.parse("07:30"));
```

6. Facade Pattern – Scheduler Facade

Use case: Provide a simple interface to manage tasks (hides complexity).

```
class SchedulerFacade {
    private List<Task> tasks = new ArrayList<>();
    public void addTask(Task t) { tasks.add(t); }
    public void showTasks() { tasks.forEach(System.out::println); }
}
```

Demo:

```
SchedulerFacade facade = new SchedulerFacade();
facade.addTask(TaskFactory.create("Breakfast", "Low"));
facade.addTask(TaskFactory.create("Launch Prep", "High"));
facade.showTasks();
```