A screenshot of a cell phone

Description automatically generated

Experiment 7 – Design Patterns

Members:

Hannah Mae Antaran

Kathleen Tupas

Darrel Tristan Virtusio

Group: 1

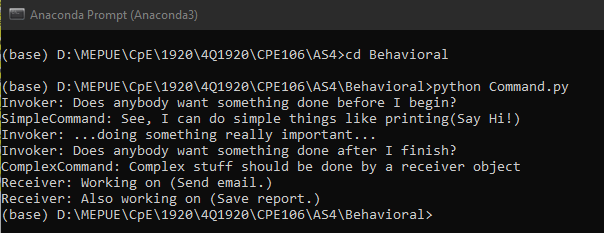
Section: E04

**PreLab**

**InLab**

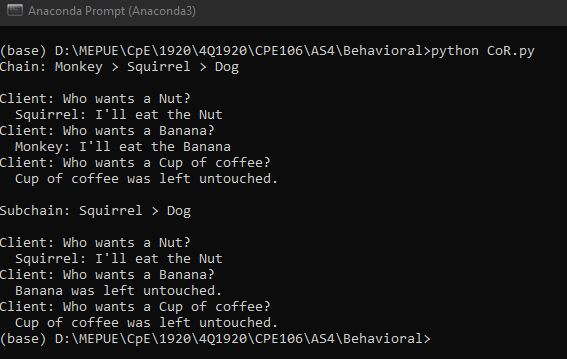
* **Objectives**
  + To understand about design patterns
  + To differentiate the different types of design patterns
  + To demonstrate the types of design patterns using Python and VS Code
* **Screenshots of tools used:**

**Behavioral Patterns**

****

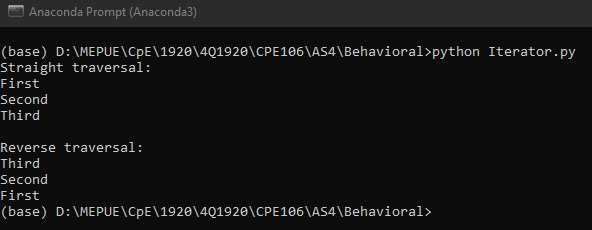
**Figure 1.1.** Command pattern

Command is a behavioral design pattern that turns a request into a stand-alone object that contains all information about the request.



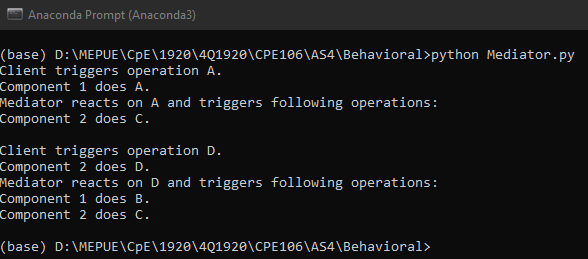
**Figure 1.2.** Chain of Responsibility pattern

Chain of Responsibility is a behavioral design pattern that lets you pass requests along a chain of handlers.



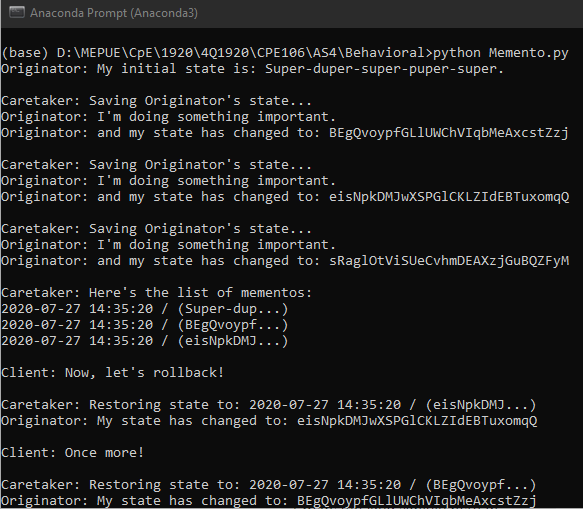
**Figure 1.3.** Iterator pattern

Iterator is a behavioral design pattern that lets you traverse elements of a collection without exposing its underlying representation (list, stack, tree, etc.).



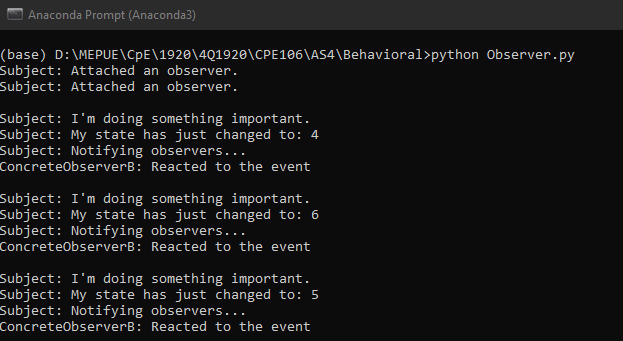
**Figure 1.4.** Mediator pattern

Mediator is a behavioral design pattern that lets you reduce chaotic dependencies between objects. The pattern restricts direct communications between the objects and forces them to collaborate only via a mediator object.



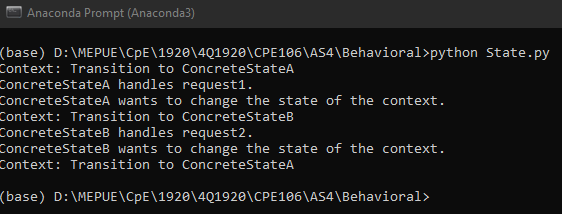
**Figure 1.5.**  Memento pattern

Memento is a behavioral design pattern that lets you save and restore the previous state of an object without revealing the details of its implementation.



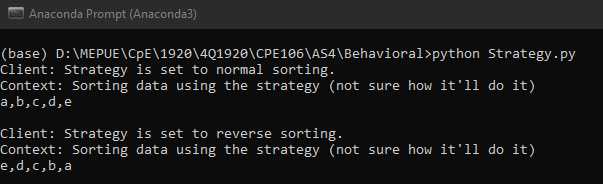
**Figure 1.6** Observer pattern

Observer is a behavioral design pattern that lets you define a subscription mechanism to notify multiple objects about any events that happen to the object they’re observing.



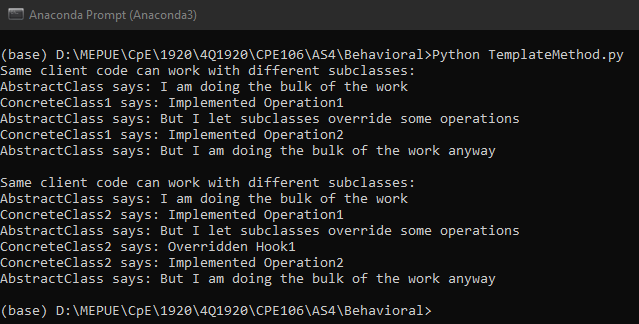
**Figure 1.7.** State pattern

State is a behavioral design pattern that lets an object alter its behavior when its internal state changes. It appears as if the object changed its class.



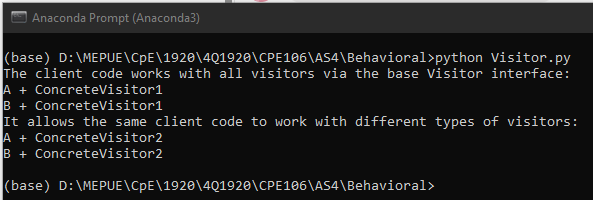
**Figure 1.8.** Strategy pattern

Strategy is a behavioral design pattern that lets you define a family of algorithms, put each of them into a separate class, and make their objects interchangeable.



**Figure 1.9.** Template Method pattern

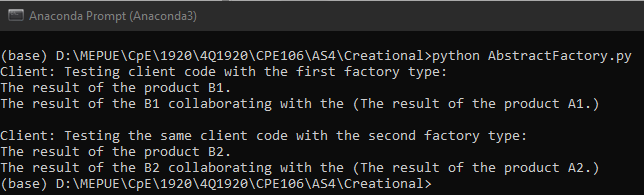
Template Method is a behavioral design pattern that defines the skeleton of an algorithm in the superclass but let’s subclasses override specific steps of the algorithm without changing its structure.



**Figure 1.10.** Visitor pattern

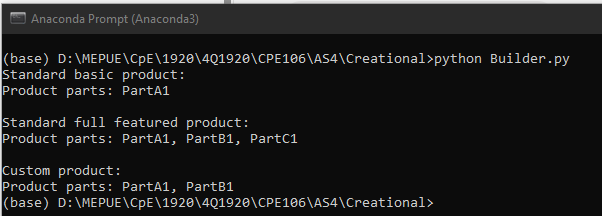
Visitor is a behavioral design pattern that lets you separate algorithms from the objects on which they operate.

**Creational Patterns**

****

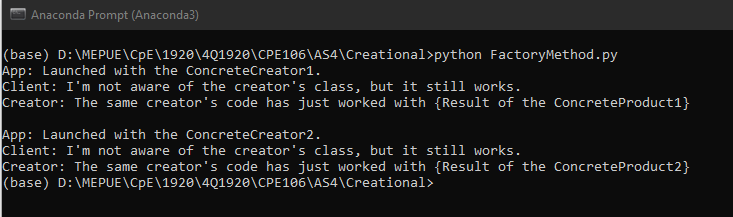
**Figure 2.1.** Abstract Factory pattern

Abstract Factory is a creational design pattern that lets you produce families of related objects without specifying their concrete classes.

**.**

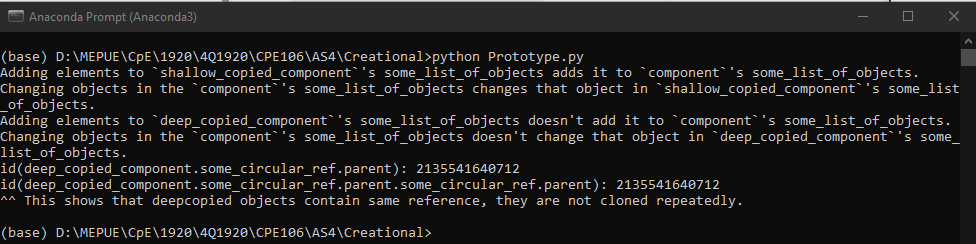
**Figure 2.2.** Builder pattern

Builder is a creational design pattern that lets you construct complex objects step by step. The pattern allows you to produce different types and representations of an object using the same construction code.



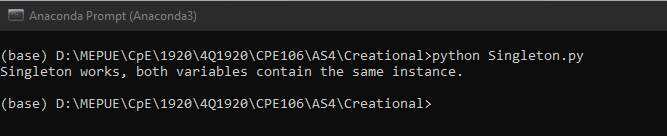
**Figure 2.3.** Factory method pattern

Factory Method is a creational design pattern that provides an interface for creating objects in a superclass but allows subclasses to alter the type of objects that will be created.



**Figure 2.4.** Prototype pattern

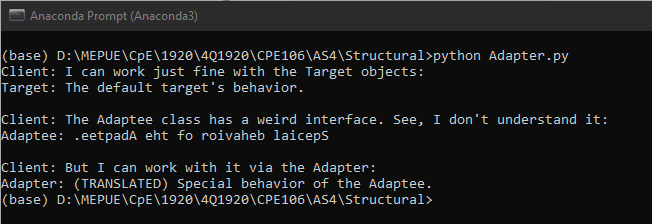
Prototype is a creational design pattern that lets you copy existing objects without making your code dependent on their classes.



**Figure 2.5.** Singleton pattern

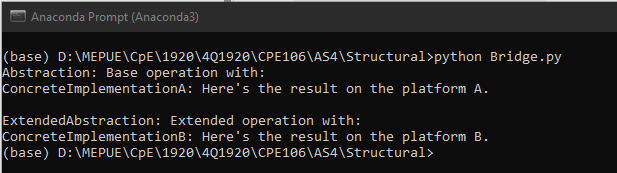
Singleton is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.

**Structural Patterns**



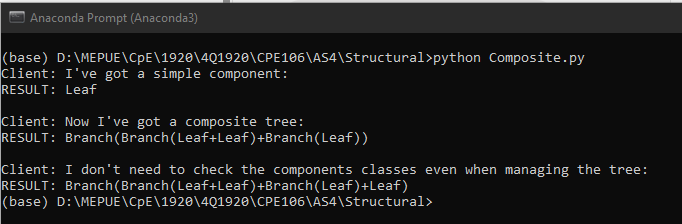
**Figure 3.1.** Adapter pattern

Adapter is a structural design pattern that allows objects with incompatible interfaces to collaborate.



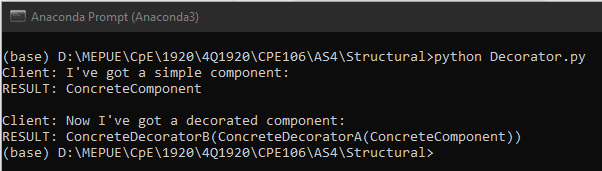
**Figure 3.2.** Bridge pattern

Bridge is a structural design pattern that lets you split a large class or a set of closely related classes into two separate hierarchies—abstraction and implementation—which can be developed independently of each other.



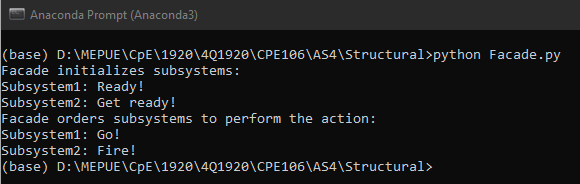
**Figure 3.3.** Composite pattern

Composite is a structural design pattern that lets you compose objects into tree structures and then work with these structures as if they were individual objects



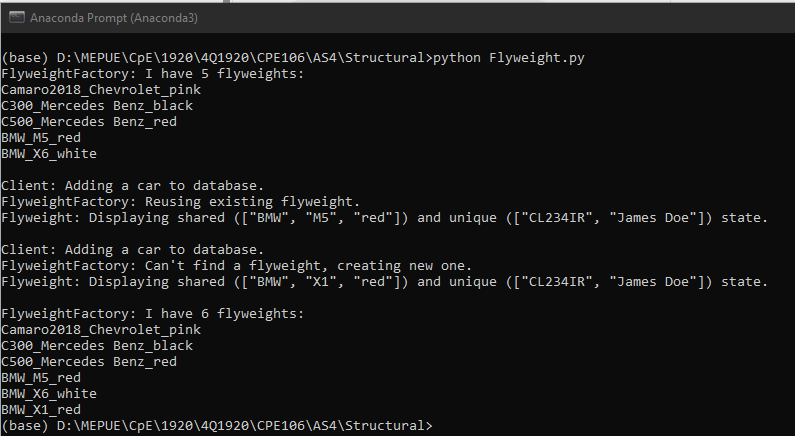
**Figure 3.4.** Decorator pattern

Decorator is a structural design pattern that lets you attach new behaviors to objects by placing these objects inside special wrapper objects that contain the behaviors.



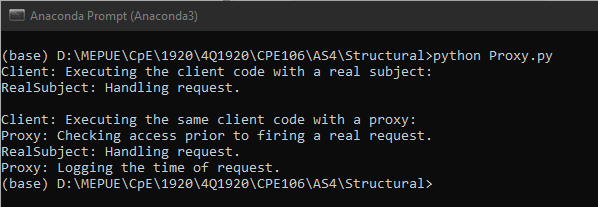
**Figure 3.5.** Facade pattern

Facade is a structural design pattern that provides a simplified interface to a library, a framework, or any other complex set of classes.



**Figure 3.6.** Flyweight pattern

Flyweight is a structural design pattern that lets you fit more objects into the available amount of RAM by sharing common parts of state between multiple objects instead of keeping all of the data in each object.



**Figure 3.7.** Proxy pattern

Proxy is a structural design pattern that lets you provide a substitute or placeholder for another object. A proxy controls access to the original object, allowing you to perform something either before or after the request gets through to the original object.