CSE 470
Software Engineering

Software Design Patterns

Imran Zahid
Lecturer
Computer Science and Engineering, BRAC University

Recap

Topics covered before Mid-Term

- Software Development Lifecycles
- Agile Development Process and Frameworks
- Requirement Engineering: Functional and Non-Functional Requirements
- Class Diagram
- Software Architectural Patterns: MVC, Layered, Repository, Client-Server, Pipe-Filter

Design Patterns

Why do we need Design Patterns

Designing Reusable software is difficult.

- Finding good objects and abstraction
- Flexibility, modularity, elegance reuse
- Takes time to emerge, trial and error

Successful design does exist

- Exhibit recurring class and object structure

How to describe these recurring structure?

Design Patterns

- Design patterns represent the best practices used by experienced object-oriented software developers.
- Design patterns are solutions to general problems that software developers faced during software development.
- These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time.

What is Gang of Four (GOF)?

- In 1994, four authors Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides published a book titled Design Patterns - Elements of Reusable Object-Oriented Software which initiated the concept of Design Pattern in Software development.
- These authors are collectively known as Gang of Four (GOF). According to these authors design patterns are primarily based on the following principles of object oriented design.
  - Program to an interface not an implementation
  - Favor object composition over inheritance

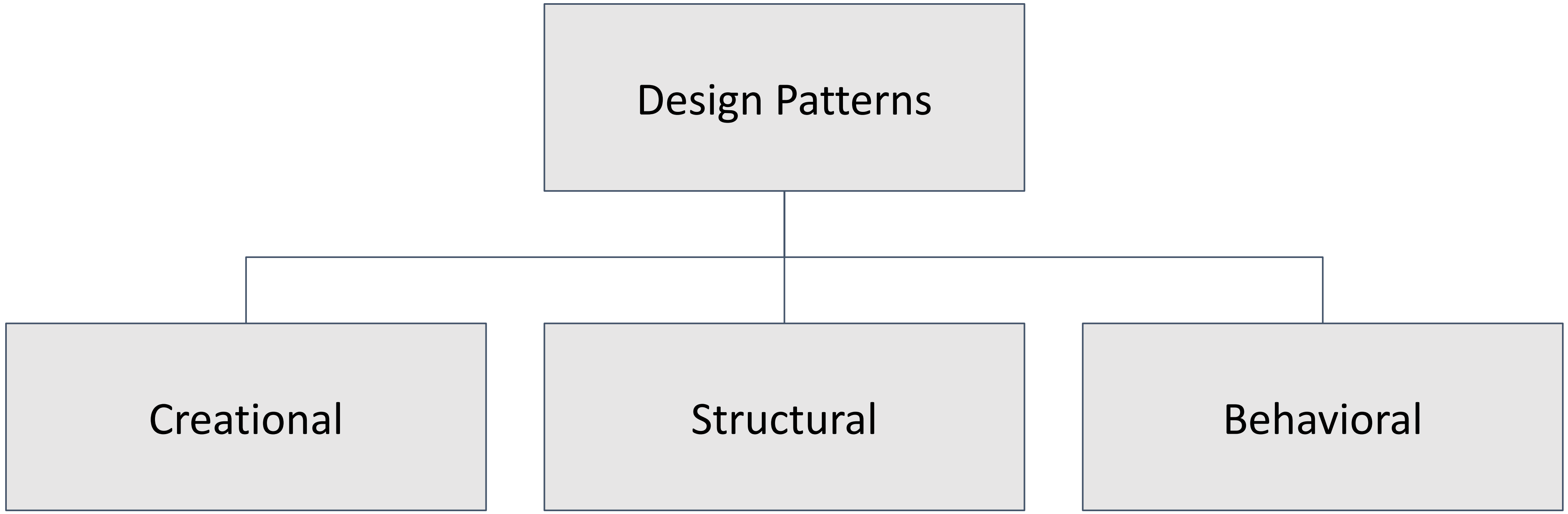
Usage of Design Pattern

- Design Patterns have two main usages in software development.
- Common platform for developers
  - Design patterns provide a standard terminology and are specific to particular scenario. For example, a singleton design pattern signifies use of single object so all developers familiar with single design pattern will make use of single object and they can tell each other that program is following a singleton pattern.
- Best Practices
  - Design patterns have been evolved over a long period of time and they provide best solutions to certain problems faced during software development. Learning these patterns helps inexperienced developers to learn software design in an easy and faster way.

Types of Design Patterns

- As per the design pattern reference book Design Patterns - Elements of Reusable Object-Oriented Software, there are 23 design patterns which can be classified into three categories: Creational, Structural and Behavioral patterns.
- We'll also discuss another category of design pattern: J2EE design patterns.

Types of Design Patterns



Creational

These design patterns provide a way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator. This gives program more flexibility in deciding which objects need to be created for a given use case.

- Talks about object creations.
- To make design more readable and less complexity.
- Talks about efficiently object creation.

Structural

These design patterns concern class and object composition. Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities.

- Talks about how classes and objects can be composed.
- Parents-child relationship like inheritance, extend.
- Deals with simplicity in identifying relationships.

Behavioral

These design patterns are specifically concerned with communication between objects.

- Deals with interactions and responsibility of objects.
- Deals with static, private, protected.

J2EE Patterns

- These design patterns are specifically concerned with the presentation tier. These patterns are identified by Sun Java Center.

Defining a Design Pattern

A design pattern is characterized by the following four components:

- Intent
  - What the design pattern is intended to do
- Motivation
  - Why the design pattern is needed
- Participants
  - Classes that will participate in the design pattern
- Structure

Singleton Pattern

Singleton Pattern

- Ensures a class has only one instance and provide a global point of access to it.
- Define the instance operation that lets clients access its unique instance. Instance is a class operation (static member function).

Example

```
public class HelpDesk {
    public void getService() {
    // Implement the service
    }
}
```

```
public class Doctor {
    HelpDesk hd = new HelpDesk();
    hd.getService();
}
```

```
public class Nurse {
    HelpDesk hd = new HelpDesk();
    hd.getService();
}
```

In reality one Single HelpDesk is serving all. No need for multiple objects

Singleton Pattern

Intent: Ensure a class only has one instance, and provide a global point of access to it.

Motivation:

- More than one instance will result in incorrect program behaviour. (thread specific)
- More than one instance will result the overuse of resources. (ex: database connection string)
- Some classes should have only one instance throughout the system for (ex: printer spooler)

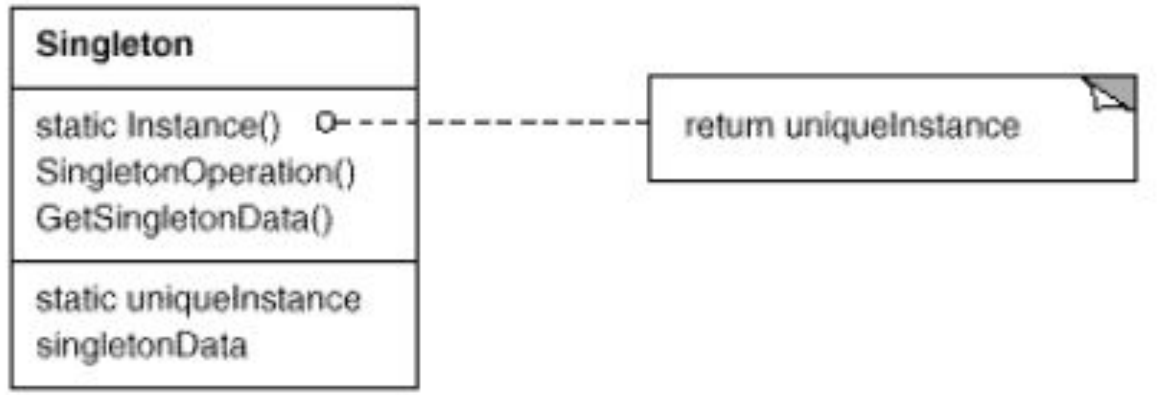
Singleton Pattern

Participants:

Singleton-defines an Instance operation that lets clients access its unique instance. Instance is a class operation .It may be responsible for creating its own unique instance. (ex: Singleton)

Singleton Pattern

Structure:



Classification: Classified as one of the most known Creational Pattern

Structure of Singleton (Python)

```
class Singleton:
    _instance = None
    def __new__(cls):
        if cls._instance is None:
            print(' instance creating’)
            cls._instance = super(Singleton, cls).__new__(cls)
        return cls._instance

def get_service(self):
    print("Service has been provided by the Singleton  instance.")
```

Structure of Singleton Cotd. - Usage

```
if __name__ == "__main__":
    # Let's assume that multiple clients need the service
    client1 = Singleton()
    client2 = Singleton()
    client3 = Singleton()

    client1.get_service()
    client2.get_service()
    client3.get_service()
    print(client1 is client2 is client3)
```

Structure of Singleton Cotd. - Output

OUTPUT=>

```
instance creating
Service has been provided by the Singleton instance.
Service has been provided by the Singleton instance.
Service has been provided by the Singleton instance.
True
```

Example Cotd.

```
class HelpDesk:
    _instance = None
    def __new__(cls):
        if cls._instance is None:
                print('Creating the HelpDesk instance')
                cls._instance = super(HelpDesk,cls).__new__(cls)
        return cls._instance

    def get_service(self):
        print("Service provided by the HelpDesk.")
```

Example Cotd. - Usage

```
if __name__ == "__main__":
    student_help_desk = HelpDesk()
    teacher_help_desk = HelpDesk()

    student_help_desk.get_service()
    teacher_help_desk.get_service()

    print(student_help_desk is teacher_help_desk)
```

Structure of Singleton (Java)

```
public class Singleton{
        private static Singleton singleInstance;
        private Singleton(){
                //nothing to do as object initiation will be done once
        }
        public static Singleton getInstance(){
                        if(singleInstance == null){
                                singleInstance = new Singleton(); // Lazy instance
                        }
                        return singleInstance;
        }
}
```

Structure of Singleton - Usage

From method call –

```
Singleton instance = Singleton.getInstance();
```

Example Cotd.

```
public class HelpDesk{
        private static HelpDesk helpDesk;
        public void getService(){
                // Implement the service
        }
        private HelpDesk(){
                // No other class will be able to create instance
        }
        public static HelpDesk getInstance(){
                if(helpDesk == null){
                                helpDesk = new HelpDesk(); // Lazy instance
                }
                return helpDesk;
        }
}
```

Example Cotd. - Usage

```
public class Student{
        HelpDesk hd = HelpDesk.getInstance();
        hd.getService();
}

public class Teacher{
        HelpDesk hd = HelpDesk.getInstance();
        hd.getService();
}
```

Lazy Instance

Lazy instance: Singleton make use of lazy initiation of the class. It means that its creation is deferred until it is first used. It helps to improve performance, avoid wasteful computation and reduce program memory requirement.

```
if(singleInstance == null){
    singleInstance = new Singleton(); // Lazy initialization
}
```

Thank you