Design Patterns

nlhdung@fit.hcmus.edu.vn
What Are Design Patterns?

• Solutions to common problems
• Targets of refactoring, not design
• Powerful, flexible, reusable
Pattern Types
Creational Patterns

Concerned with the creation of objects and instances.
Structural Patterns

Concerned with the overall design of the system and its constituent classes and objects.
Behavioral Patterns

Concerned with the assignment of responsibilities to objects and classes.
Style for Describing Patterns

• We will use this structure:
  • *Pattern name*
  • *Purpose*: what problem the pattern addresses and the general approach of the pattern
  • *UML for the pattern*
    • *Participants*: a description as a class diagram
  • *Use Example(s)*: examples of this pattern, in C# and other
Singleton
Singleton - Purpose

• This pattern ensures that a class has only one instance and provides a global point of access to it.
  • Exactly one instance of a class is required.
  • Controlled access to a single object is necessary
Singleton - UML

Singleton

- instance: Singleton

- Singleton()
+ GetInstance(): Singleton

Singleton Pattern
Singleton – Sample 1

Eager initialization of singleton

```java
public class Singleton {

    private static Singleton instance = new Singleton();
    private Singleton() {}

    public static Singleton GetInstance
    {
        get
        {
            return instance;
        }
    }
}
```
Singleton – Sample 2

lazy initialization of singleton

```java
public class Singleton {
    private static Singleton instance = null;
    private Singleton() {}

    public static Singleton GetInstance {
        get {
            if (instance == null)
                instance = new Singleton();
            return instance;
        }
    }
}
```
Singleton – Sample 3

Thread safe initialization of singleton

```java
public class Singleton {
    private static Singleton instance = null;

    private Singleton() {} // Prevent direct instantiation

    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton(); // Lazy initialization
        }
        return instance;
    }
}
```
Composite
Composite - Purpose

• This pattern is used to separate an abstraction from its implementation so that both can be modified independently.

• It is used when we need to treat a group of objects and a single object in the same.

• It creates a class contains group of its own objects. This class provides ways to modify its group of same objects.
Composite – When to use

• Hierarchical representations of objects are required.
• The Composite pattern is used when data is organized in a tree structure
• A single object and a group of objects should be treated in the same way.
Composite – UML

Component
+ operation()

Leaf
+ operation()

Composite
+ operation()
+ add()
+ remove()
+ getChild()
Composite – UML (2)
Composite – Sample

• The classes, interfaces and objects:
  • IEmployed - Component Interface.
  • Employee- Composite Class.
  • Contractor- Leaf Class
Composite – Sample
Composite – Sample

- Project’s source code...

CuuDuongThanCong.com

https://fb.com/tailieudientucntt
Composite – Sample

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Rahul</td>
</tr>
<tr>
<td>2</td>
<td>Amit</td>
</tr>
<tr>
<td>4</td>
<td>Rita</td>
</tr>
<tr>
<td>5</td>
<td>Hari</td>
</tr>
<tr>
<td>3</td>
<td>Mohan</td>
</tr>
<tr>
<td>6</td>
<td>Kamal</td>
</tr>
<tr>
<td>7</td>
<td>Raj</td>
</tr>
<tr>
<td>8</td>
<td>Sam</td>
</tr>
<tr>
<td>9</td>
<td>Tim</td>
</tr>
</tbody>
</table>
Prototype
Prototype - Purpose

• Create objects by cloning a prototypical instance.
• Consumes less resources than creating new objects.
Prototype – UML

Prototype Pattern

ConcretePrototype1
+Clone()

ConcretePrototype2
+Clone()
Prototype – Sample
Prototype – Sample

• Project’s source code...

CuuDuongThanCong.com

CuuDuongThanCong.com
Template
Template - Purpose

• This pattern is used to define the basic steps of an algorithm/task and allow the implementation of the individual steps to be changed.
Template – UML

```
void templateMethod()
{
    // for example
    operation1();
    operation2();
}
```

AbstractClass

+ void : templateMethod()

void : operation1();
void : operation2();

ConcreteClassA

void : operation1();
void : operation2();

ConcreteClassB

void : operation1();
void : operation2();

CuuDuongThanCong.com

https://fb.com/tailieudientucntt
Template – Sample
Prototype – Sample

• Project’s source code...

CuuDuongThanCong.com

https://fb.com/tailieudientucntt
Questions?

cuu duong than cong . com

CuDuongThanCong.com  https://fb.com/tailieudientucntt
Thanks!

Reference:

• Erich Gamma, John Vlissides, Ralph Johnson, and Richard Helm, “Design Patterns: Elements of Reusable Object-Oriented Software”, 1994

• Bert Bates, Kathy Sierra, Eric Freeman, Elisabeth Robson, “Head First Design Patterns”, 2009