16. Prototype Pattern

16. Prototype Pattern

Prototype Pattern

Scenario:

Imagine you're working on a document editor where you often need to create new documents based on existing ones. For instance, a user might want to create multiple versions of a document with slight modifications or use an existing document as a template for a new one. You want to allow users to clone existing documents quickly and efficiently.

Purpose:

The Prototype Pattern is used to create new objects by copying an existing object, known as the prototype. This pattern is particularly useful when the process of creating an object is more complex or costly than simply copying an existing object.

When to Use:

- When the creation of an object is expensive or complex.
- When you want to create multiple objects with initial state derived from existing objects.
- When you want to avoid subclassing to create new instances.

Key Concepts:

- Prototype: An interface that defines the clone() method, which will be used to copy existing objects.
- Concrete Prototype: Implements the clone() method to create a copy of the object.
- Client: The part of the code that uses the clone() method to create new objects.

How It Works:

- Prototype Interface: Defines a method for cloning objects (e.g., clone()).

- Concrete Prototype: Implements the Prototype interface and provides the actual copying logic.
- Client: Uses the prototype to create new objects by calling the clone() method.

Implementation Example:

Here's how the implementation might look in Java:

```java
// Prototype Interface
interface Document extends Cloneable {
    Document clone();
    void print();
}

// Concrete Prototype
class WordDocument implements Document {
    private String content;

    public WordDocument(String content) {
        this.content = content;
    }

    @Override
    public Document clone() {
        return new WordDocument(this.content);
    }
    @Override
    public void print() {
        System.out.println("Word Document Content: " + content);
    }
}

class PDFDocument implements Document {
    private String content;

    public PDFDocument(String content) {
        this.content = content;
    }

    @Override
    public Document clone() {
        return new PDFDocument(this.content);
    }

    @Override
    public void print() {
        System.out.println("PDF Document Content: " + content);
    }
}
```

```
// Client Code
public class DocumentEditor {
    public static void main(String[] args) {
        // Create a Word document
        Document originalWordDoc = new WordDocument("Design Patterns in Java");
        originalWordDoc.print();

        // Clone the Word document
        Document clonedWordDoc = originalWordDoc.clone();
        clonedWordDoc.print();

        // Create a PDF document
        Document originalPDFDoc = new PDFDocument("Design Patterns in Java - PDF");
        originalPDFDoc.print();

        // Clone the PDF document
        Document clonedPDFDoc = originalPDFDoc.clone();
        clonedPDFDoc.print();
    }
}
```

Key Points to Remember:

- Prototype Pattern allows you to create a new object by copying an existing object (the prototype). This pattern is particularly useful when the cost of creating an object is high, or when creating objects involves many steps that can be skipped by cloning.

- Shallow vs. Deep Cloning: Depending on the nature of the object, you may need to implement shallow or deep cloning. Shallow cloning copies the object's top-level structure, while deep cloning copies all nested objects as well.

Advantages:

- Performance: Cloning is typically faster than creating a new object from scratch.
- Flexibility: The pattern provides a way to create new instances without being tightly coupled to the specific classes.

Disadvantages:

- Cloning Complexity: If the object has complex internal structures (like nested objects), implementing deep cloning can be challenging.
- Potential for Inconsistencies: Cloning might lead to subtle bugs if the clone is not implemented correctly, especially when deep cloning is required.