Design Pattern

Level 3 Term 1

October 6, 2024

Contents

1 Strategy Pattern **2**
1.1 Refactoring Guru 2
1.2 TutorialsPoint 2

2 Template Method Pattern **3**
2.1 Refactoring Guru 3
2.2 TutorialsPoint 4

1 Strategy Pattern

1.1 Refactoring Guru

Strategy Pattern

1.2 TutorialsPoint

In Strategy pattern, a class behavior or its algorithm can be changed at run time. This type of design pattern comes under behavior pattern.

In Strategy pattern, we create objects which represent various strategies and a context object whose behavior varies as per its strategy object. The strategy object changes the executing algorithm of the context object.

```
    public interface Strategy {
   public int doOperation(int num1, int num2);
}
public class OperationAdd implements Strategy{
   @Override
   public int doOperation(int num1, int num2) {
      return num1 + num2;
   }
}
public class OperationSubstract implements Strategy{
   @Override
   public int doOperation(int num1, int num2) {
      return num1 - num2;
   }
}
public class OperationMultiply implements Strategy{
   @Override
   public int doOperation(int num1, int num2) {
      return num1 * num2;
   }
}
public class Context {
   private Strategy strategy;

   public Context(Strategy strategy){
```

```
      this.strategy = strategy;
   }

   public int executeStrategy(int num1, int num2){
      return strategy.doOperation(num1, num2);
   }
}
public class StrategyPatternDemo {
   public static void main(String[] args) {
      Context context = new Context(new OperationAdd());
      System.out.println("10 + 5 = " + context.executeStrategy(10, 5));

      context = new Context(new OperationSubstract());
      System.out.println("10 - 5 = " + context.executeStrategy(10, 5));

      context = new Context(new OperationMultiply());
      System.out.println("10 * 5 = " + context.executeStrategy(10, 5));
   }
}
```

Verify the output:

```
10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
```

2 Template Method Pattern

In Template pattern, an abstract class exposes defined way(s)/template(s) to execute its methods. Its subclasses can override the method implementation as per need but the invocation is to be in the same way as defined by an abstract class. This pattern comes under behavior pattern category.

2.1 Refactoring Guru

Template method

2.2 TutorialsPoint

```
public abstract class Game {
   abstract void initialize();
   abstract void startPlay();
   abstract void endPlay();

   //template method
   public final void play(){

      //initialize the game
      initialize();

      //start game
      startPlay();

      //end game
      endPlay();
   }
}

    public class Cricket extends Game {

   @Override
   void endPlay() {
      System.out.println("Cricket Game Finished!");
   }

   @Override
   void initialize() {
      System.out.println("Cricket Game Initialized! Start playing.");
   }

   @Override
   void startPlay() {
      System.out.println("Cricket Game Started. Enjoy the game!");
   }
}
```

```
    public class Football extends Game {

   @Override
   void endPlay() {
      System.out.println("Football Game Finished!");
   }

   @Override
   void initialize() {
      System.out.println("Football Game Initialized! Start playing.");
   }

   @Override
   void startPlay() {
      System.out.println("Football Game Started. Enjoy the game!");
   }
}

    public class TemplatePatternDemo {
   public static void main(String[] args) {

      Game game = new Cricket();
      game.play();
      System.out.println();
      game = new Football();
      game.play();
   }
}
```

Verify the output:

```
Cricket Game Initialized! Start playing.
Cricket Game Started. Enjoy the game!
Cricket Game Finished!

Football Game Initialized! Start playing.
Football Game Started. Enjoy the game!
Football Game Finished!
```