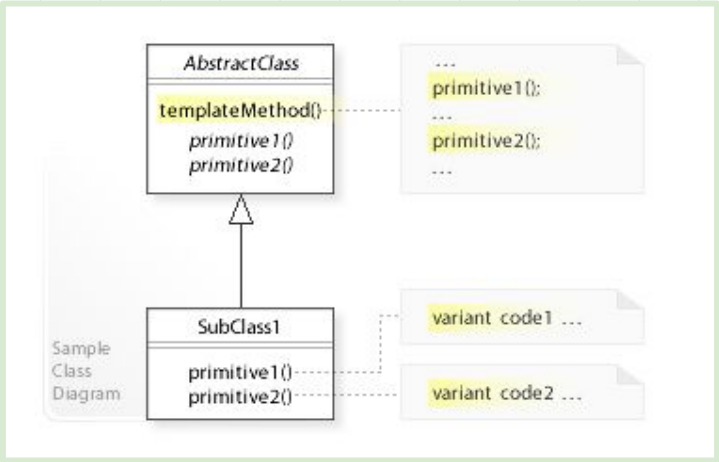
Template Method

By: Laura Salamanca and Daniel Mead

What is the Template Method?

- Is a Behavioral Design pattern
- The template method is typically a method defined in a abstract super class
- It ensures that all aspects that a program needs are defined for a base class



Code Examples

```
public abstract class HouseTemplate {

        //template method, final so subclasses can't override
        public final void buildHouse(){
                buildFoundation();
                buildPillars();
                buildWalls();
                buildWindows();
                System.out.println("House is built.");
        }
```

```
        //methods to be implemented by subclasses
        public abstract void buildWalls();
        public abstract void buildPillars();
```

```
        //default implementation
        private void buildWindows() {
                System.out.println("Building Glass Windows");
        }
```

- **HouseTemplate**
  - Superclass
  - Outlines the structure of the house object
- **buildHouse**
  - The template method
- **buildWalls and buildPillars**
  - Abstract methods to be defined in subclasses
- **buildWindows**
  - Default implementation
  - Can be overridden if needed

Code Examples Cont'd

- **WoodenHouse and GlassHouse**
  - Subclasses of HouseTemplate
  - Provide definitions of abstract methods buildWalls and buildPillars

```java
public class WoodenHouse extends HouseTemplate {

        @Override
        public void buildWalls() {
                System.out.println("Building Wooden Walls");
        }

        @Override
        public void buildPillars() {
                System.out.println("Building Pillars with Wood coating");
        }

}
```

```java
public class GlassHouse extends HouseTemplate {

        @Override
        public void buildWalls() {
                System.out.println("Building Glass Walls");
        }

        @Override
        public void buildPillars() {
                System.out.println("Building Pillars with glass coating");
        }

}
```

- Overriding methods from superclass allows for slight variations in the algorithm's steps

Output Examples

```java
public class HousingClient {

        public static void main(String[] args) {

                HouseTemplate houseType = new WoodenHouse();

                //using template method
                houseType.buildHouse();
                System.out.println("************");

                houseType = new GlassHouse();

                houseType.buildHouse();
        }

}
```

```
Building foundation with cement,iron rods and sand
Building Pillars with Wood coating
Building Wooden Walls
Building Glass Windows
House is built.
************
Building foundation with cement,iron rods and sand
Building Pillars with glass coating
Building Glass Walls
Building Glass Windows
House is built.
```

Pros

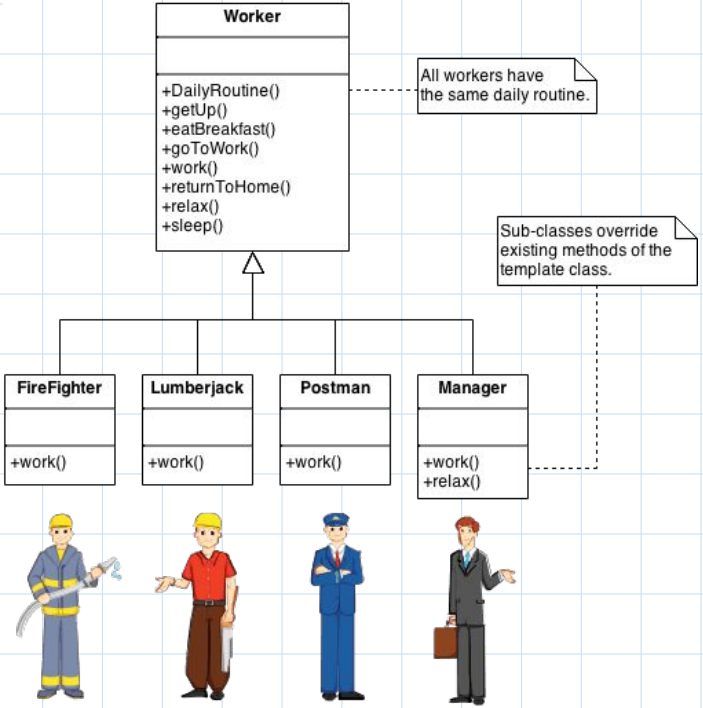
- Reduces duplicate code
- Flexibility
  - Let subclasses decide how to implement certain steps
- Only need to implement the workflow of the program once
- The overall workflow of the program will never change

Cons

- Subclasses are unable to change the overall workflow of the program
- Tight coupling between base class and subclasses

Real World Use

- This would be used in the real world to list all different types of workers at a company
  - They all share the same way of being paid
  - They all share all the same base information
  - However they all differ in what job they perform so that method will be defined in each class



Limitations

- If an algorithm is highly variable with little similarity between steps, the template method might make the program excessively complex
- Relies on predefined structure, so the method may not be viable if changes are anticipated at runtime