# Template Method Pattern

Michal Moravik, SD20w2

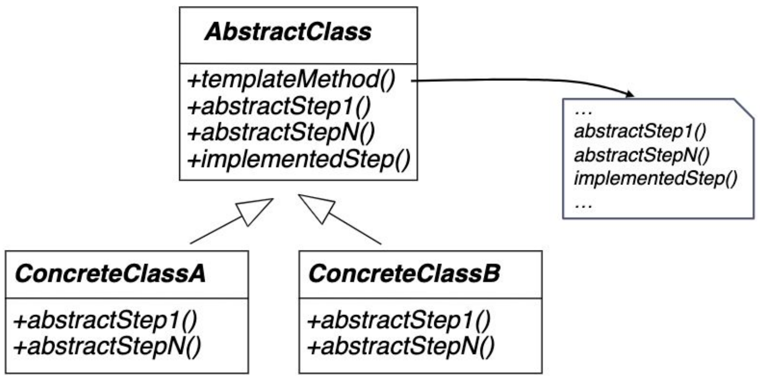
**Name**: Template Method Pattern

**Category**: Behavioral

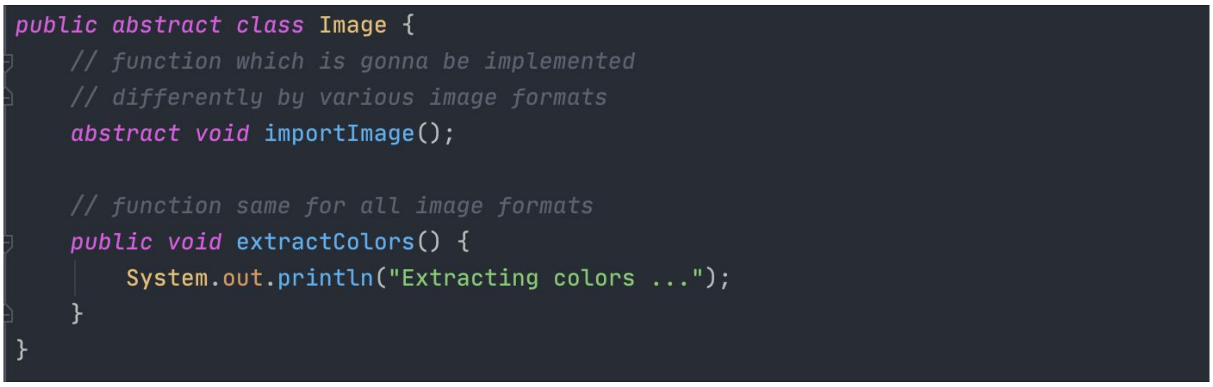
**Intent**: Defines the skeleton of an algorithm in the superclass, leaves “placeholders”, and lets subclasses override the particular steps (fill in the specific functionality).

**Motivation:**
Imagine you are trying to build an app which takes data from multiple types of files, for instance, an app where you can upload JPG, PNG, and/or other. Then the image is processed and the app is able to extract and display colours. The process of importing different types of files is slightly different but still does the same kind of job. Then the processing in order to display the colours can be the same for all of them. **To avoid code duplication**, you can just have a skeleton and use different subclasses’ import functionality based on the type of an image.

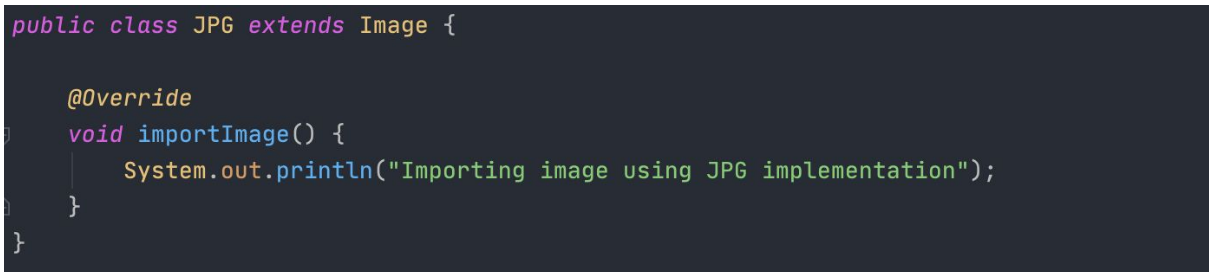
**UML:**



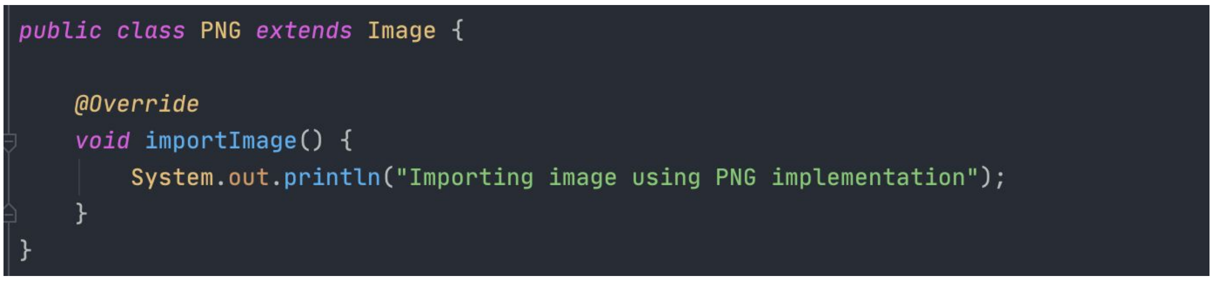
**Implementation:**



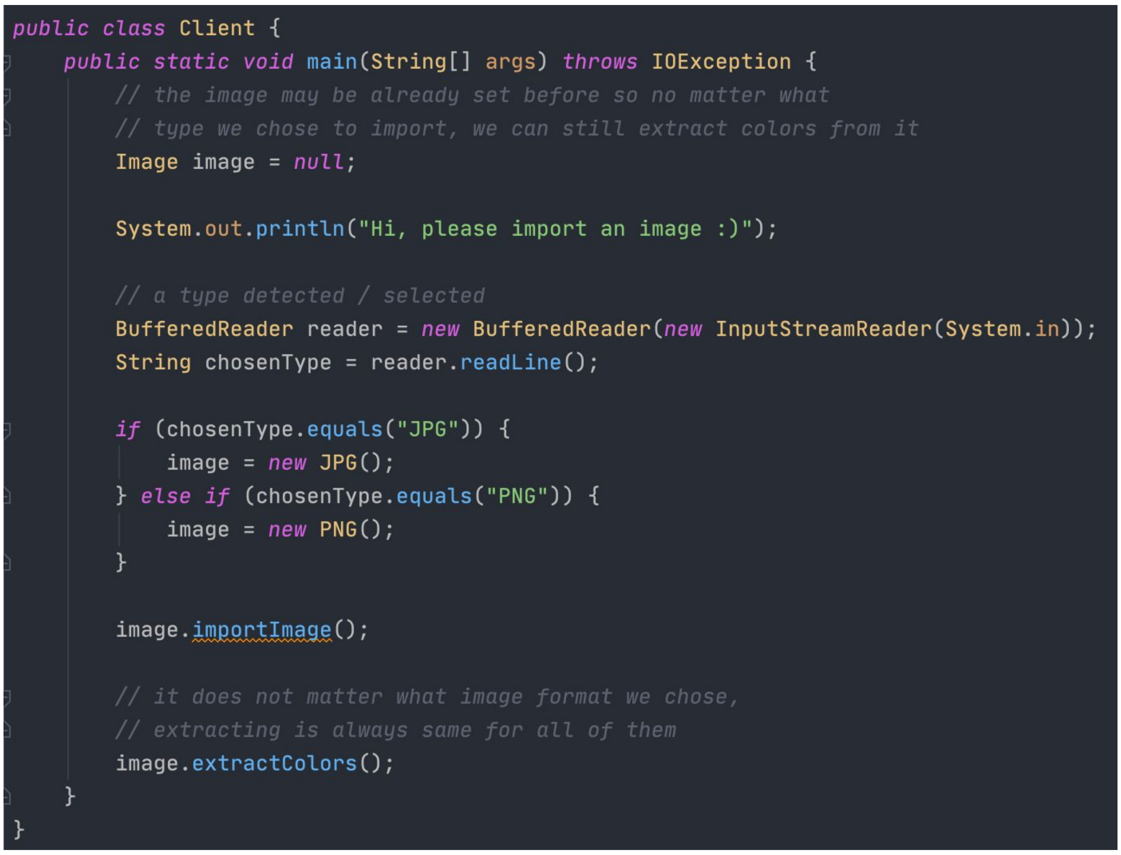
```java
public abstract class Image {
    // function which is gonna be implemented
    // differently by various image formats
    abstract void importImage();

    // function same for all image formats
    public void extractColors() {
        System.out.println("Extracting colors ...");
    }
}
```



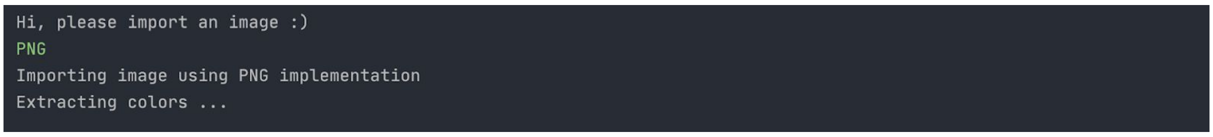
```java
public class JPG extends Image {

    @Override
    void importImage() {
        System.out.println("Importing image using JPG implementation");
    }
}
```



```java
public class PNG extends Image {

    @Override
    void importImage() {
        System.out.println("Importing image using PNG implementation");
    }
}
```



```java
public class Client {
    public static void main(String[] args) throws IOException {
        // the image may be already set before so no matter what
        // type we chose to import, we can still extract colors from it
        Image image = null;

        System.out.println("Hi, please import an image :)");

        // a type detected / selected
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        String chosenType = reader.readLine();

        if (chosenType.equals("JPG")) {
            image = new JPG();
        } else if (chosenType.equals("PNG")) {
            image = new PNG();
        }

        image.importImage();

        // it does not matter what image format we chose,
        // extracting is always same for all of them
        image.extractColors();
    }
}
```



```
Hi, please import an image :)
PNG
Importing image using PNG implementation
Extracting colors ...
```

**Consequences:**
**Pros:**

- Ability to have one skeleton and override only particular parts of the skeleton
- The code which is the same for all classes (duplicated code) can be moved into a superclass instead

**Cons:**

- The provided skeleton can be a limitation if you need a different one

**Known uses:**

- If you want to extend only particular steps of a class but not the whole class
- If you have multiple classes containing the same process but with minor differences in implementation
- Java core libraries implement this approach

**Related patterns:**

- Factory method