Template-Method-Pattern

Carsten Gips (FH Bielefeld)

Motivation: Syntax-Highlighting im Tokenizer

```java
public class Lexer {
    private final List<Token> allToken;  // alle verfügbaren Token-Klassen

    public List<Token> tokenize(String string) {
        List<Token> result = new ArrayList<>();

        while (string.length() > 0) {
            for (Token t : allToken) {
                Token token = t.match(string);
                if (token != null) {
                    result.add(token);
                    string = string.substring(token.getContent().length(), string.length());
                }
            }
        }

        return result;
    }
}
```

Token-Klassen mit formatiertem Inhalt

```
public abstract class Token {
    protected String content;

    abstract protected String getHtml();
}
public class KeyWord extends Token {
    @Override
    protected String getHtml() {
        return "<font color=\"red\"><b>" +  this.content + "</b></font>";
    }
}
public class StringContent extends Token {
    @Override
    protected String getHtml() {
        return "<font color=\"green\">" +  this.content + "</font>";
    }
}


Token t = new KeyWord();
LOG.info(t.getHtml());
```

Don’t call us, we’ll call you

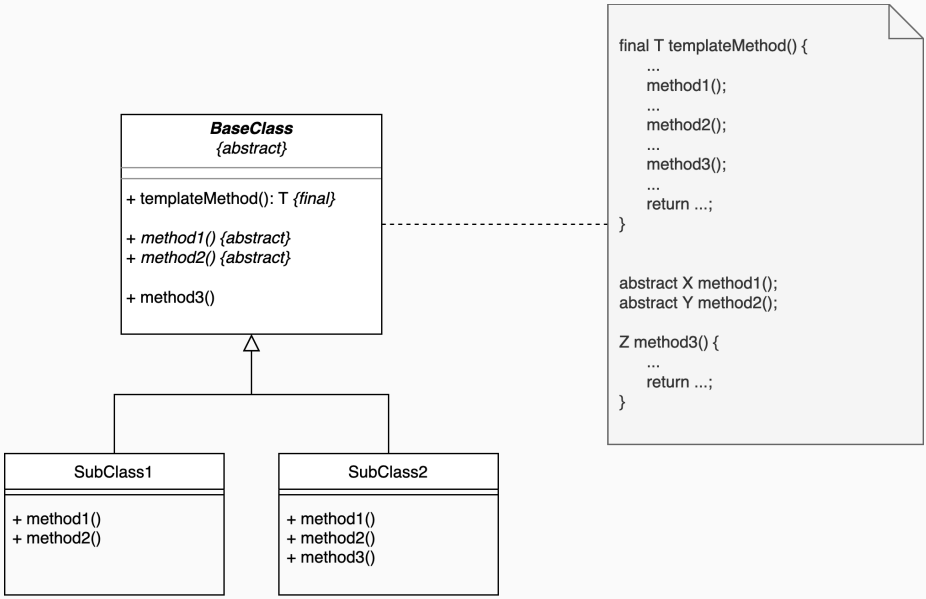
```java
public abstract class Token {
    protected String content;

    public final String getHtml() {
        return htmlStart() + this.content + htmlEnd();
    }

    abstract protected String htmlStart();
    abstract protected String htmlEnd();
}
public class KeyWord extends Token {
    @Override protected String htmlStart() { return "<font color=\"red\"><b>"; }
    @Override protected String htmlEnd() { return "</b></font>"; }
}
public class StringContent extends Token {
    @Override protected String htmlStart() { return "<font color=\"green\">"; }
    @Override protected String htmlEnd() { return "</font>"; }
}


Token t = new KeyWord();
LOG.info(t.getHtml());
```

Template-Method-Pattern



Wrap-Up

Template-Method-Pattern: Verhaltensänderung durch Vererbungsbeziehungen

- Basis-Klasse:
  - Template-Methode, die Verhalten definiert und Hilfsmethoden aufruft
  - Hilfsmethoden: Abstrakte Methoden (oder “Hook”: Basis-Implementierung)
- Ableitende Klassen: Verfeinern Verhalten durch Implementieren der Hilfsmethoden
- Zur Laufzeit: Dynamische Polymorphie: Aufruf der Template-Methode nutzt die im tatsächlichen Typ des Objekts implementierten Hilfsmethoden

LICENSE