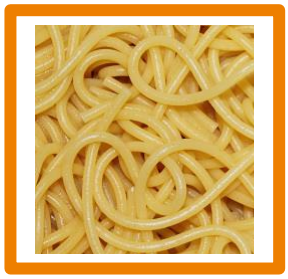
ToiletPaper #90

Pasta Orientata Agli Oggetti

jambit WHERE INNOVATION WORKS

Author: M. W.

✕ Problem



Spaghetti code is often associated with "unstructuredness". Object-oriented (orientata agli oggetti) programming and paradigms help to avoid spaghetti code. Unfortunately, inheritance and the template method pattern can cause new ways to develop spaghetti code. However, the terms *spaghetti code* and *unstructured* are too vague to explain this aspect. Therefore, this first attempt to define spaghetti code more detailed:

The program flow of spaghetti code complicates
that implementation details can be isolated on the one hand
and on the other hand can be hidden.

➔ Example

Personal and generic letters should be created using a `compose()` method. The abstract class delegates to templates. The concrete implementations use `compile()`. An implementation is outlined in `PersonalLetter`.

```
public abstract class BaseLetter {
 public String compose(String msg) {
   return salutation() + body(msg)
     + signature();
 }
 protected abstract String salutation();
 protected abstract String body(String);
 protected abstract String signature();
 protected String compile(String) {...}
}
```

```
public class PersonalLetter
    extends BaseLetter {
  private String name;
  protected String salutation() {
    return compile("Dear _"+name+"_,");
  }
  protected String body(String) {...}
  protected String signature() {...}
}
```

The code is structured, but `compile()` is difficult to test. Details are only hidden to a limited extent. The reason is the program flow, which meanders through the class hierarchy. Similar examples are: `BaseMapper`, `BaseTest`, `Base` ...

✓ Suggestions for improvement

Death to the `protected`! The template method pattern is convenient for `public` methods such as `addAll()` and `add()` for `AbstractList`. An inheritance that is only about reuse can often be replaced by delegation. In this case, the better solution is using the strategy pattern. Classes can easily be isolated and details can be hidden with `private`. The `CompositionStrategy` interface defines the template methods and `MarkdownCompiler` implements `compile()`.

```
public class LetterComposer {
  private CompositionStrategy strategy;
  public String compose(String msg) {
    return strategy.salutation()
      + strategy.body(msg)
      + strategy.signature();
  }
}
```

```
public class PersonalCompositionStrategy
    implements CompositionStrategy {
  private String name;
  private MarkdownCompiler compiler;
  public String salutation() {...}
  public String body(String) {...}
  public String signature() {...}
}
```

✚ Further Aspects

- Template Method Pattern
- Strategy Pattern
- SOLID Design Principles