Java Programming II

Lab1

514770-1
Fall 2020
9/15/2020

Kyoung Shin Park
Computer Engineering
Dankook University

DRY (Don’t Repeat Yourself) Principle

- In the book “The Pragmatic Programmer”, DRY is defined as “Every piece of **knowledge** must have a single, unambiguous, authoritative representation within a system."
  - Knowledge – a precise functionality or an algorithm
- Violations of DRY
  - WET, "We enjoy typing," or “Waste everyone’s time”.
- How to Achieve DRY
  - To avoid violating the DRY principle, divide your system into pieces. Divide your code and logic into **smaller reusable units** and use that code by calling it where you want.
- DRY Benefits
  - Less code is good: It saves time and effort, is easy to maintain, and also reduces the chances of bugs.

KISS (Keep It Simple Stupid) Principle

- “Keep It Simple Stupid”, “Keep It Short and Simple”
- The KISS principle is descriptive to **keep the code simple and clear**, making it **easy to understand**.
- Violations of KISS
  - "Why they have written these unnecessary lines and conditions when we could do the same thing in just 2-3 lines?"
- How to Achieve KISS
  - To avoid violating the KISS principle, try to **write simple code**. Whenever you find lengthy code, divide that into multiple methods — **refactor**.
- KISS Benefits
  - If the code is written simply, then there will not be any difficulty in understanding that code, and also will be easy to modify.

YAGNI (You Aren't Gonna Need It) Principle

- YAGNI says "don't implement something until it is necessary." YAGNI tells us to cut off any unnecessary part while KISS advises to make the rest as simple as possible.
- Violations of YAGNI
  - "over engineering" - a feature for every possible case, functions with a lot of input parameters, multiple if-else branches, rich and detailed interfaces, all those could be a smell of over engineering.
- How to Achieve YAGNI
  - Always **implement things when you actually need them**, never when you just foresee that you need them.
- YAGNI Benefits
  - Software developers don't have enough information to make the call on extra features, the time spent could be used elsewhere more productively. Extra features mean extra development time, testing time, documentation time, code review time.

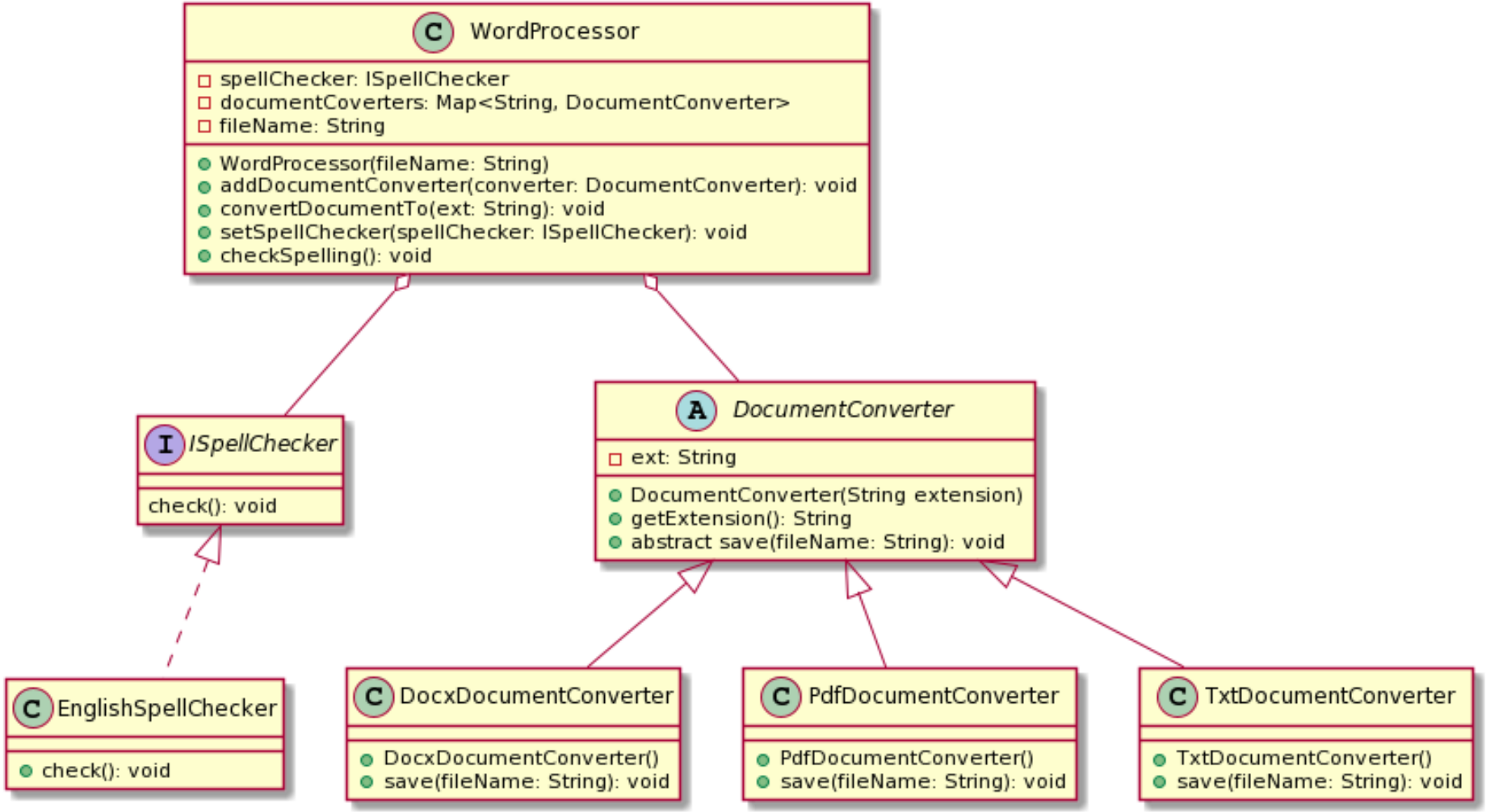
SOLID Principle

- **Single Responsibility Principle**
  - "A class should have one, and only one, reason to change."
- **Open/Closed Principle**
  - "Software entities (e.g. classes, modules, functions, etc) should be open for extension, but closed for modification."
- **Liskov Substitution Principle**
  - "Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program."
- **Interface Segregation Principle**
  - "Clients should not be forced to depend upon interfaces that they do not use." Reduce fat interfaces into multiple smaller and more specific client specific interfaces.
- **Dependency Inversion Principle**
  - One should depend on abstractions (interfaces and abstract classes) instead of concrete implementations (classes).

Lab1

- Practice to write a **word processor** program that has the ability to **spell checking** and **save in different file formats.**
- Given ISpellChecker interface, DocumentConverter abstract class, and Main class
- Write the following classes
  - EnglishSpellChecker
  - DocxDocumentConverter
  - PdfDocumentConverter
  - TxtDocumentConverter
  - WordProcessor

Lab1


WordProcessor
spellChecker: ISpellChecker
documentCoverters: Map<String, DocumentConverter>
fileName: String
WordProcessor(fileName: String)
addDocumentConverter(converter: DocumentConverter): void
convertDocumentTo(ext: String): void
setSpellChecker(spellChecker: ISpellChecker): void
checkSpelling(): void
ISpellChecker
check(): void
DocumentConverter
ext: String
DocumentConverter(String extension)
getExtension(): String
abstract save(fileName: String): void
EnglishSpellChecker
check(): void
DocxDocumentConverter
DocxDocumentConverter()
save(fileName: String): void
PdfDocumentConverter
PdfDocumentConverter()
save(fileName: String): void
TxtDocumentConverter
TxtDocumentConverter()
save(fileName: String): void

Lab1

- EnglishSpellChecker
  - Class implements ISpellChecker
  - check() – print "English Spell Checking..."
- DocxDocumentConverter, PdfDocumentConverter, TxtDocumentConverter
  - Constructor call super and assign the extension of the file format that it converts ("docx", "pdf", "txt")
  - save() – print "Convert the file to filename.ext"
    - filename – the filename passed as a parameter to save()
    - ext – the extension assigned in DocumentConverter

Lab1

- WordProcessor
  - WordProcessor()
    - Get the filename from the user and store it
  - addDocumentConverter()
    - Add document converter object to Map (key - extension, value - DocumentConverter)
  - convertDocumentTo()
    - Convert the document to the specific extension (file format)
  - setSpellChecker()
    - Set SpellChecker
  - checkSpelling()
    - Call the assigned SpellChecker's check()

Lab1

```
public interface ISpellChecker {
    void check();
}
```

Lab1

```
public abstract class DocumentConverter {
    private String ext;

    public DocumentConverter(String extension) {
        ext = extension;
    }

    public String getExtension() {
        return ext;
    }

    public abstract void save(String filename);
}
```

```
public class Main {
  public static void main(String[] args) {
    WordProcessor wp =
                        new WordProcessor("doc1.docx");
    wp.setSpellChecker(new EnglishSpellChecker());
    wp.addDocumentConverter(
                    new DocxDocumentConverter());
    wp.addDocumentConverter(
                    new PdfDocumentConverter());
    wp.addDocumentConverter(
                    new TxtDocumentConverter());
    wp.checkSpelling();
    wp.convertDocumentTo("txt");
    wp.convertDocumentTo("pdf");
    wp.convertDocumentTo("docx");
    wp.convertDocumentTo("wps");
  }
}
```

Lab1

- Execution

```
English Spell Checking
TxtDocumentConverter Save document converted to doc1.txt
PdfDocumentConverter Save document converted to doc1.pdf
DocxDocumentConverter Save document converted to doc1.docx
cannot convertDocumentTo wps file format
```

Submit to e-learning

- Add your code (e.g., additional method, class, routine, etc) in the Lab1 assignment.
- Submit the Lab1 assignment (including the report) to e-learning.