# Lexer, SOLID

#06 More indirections

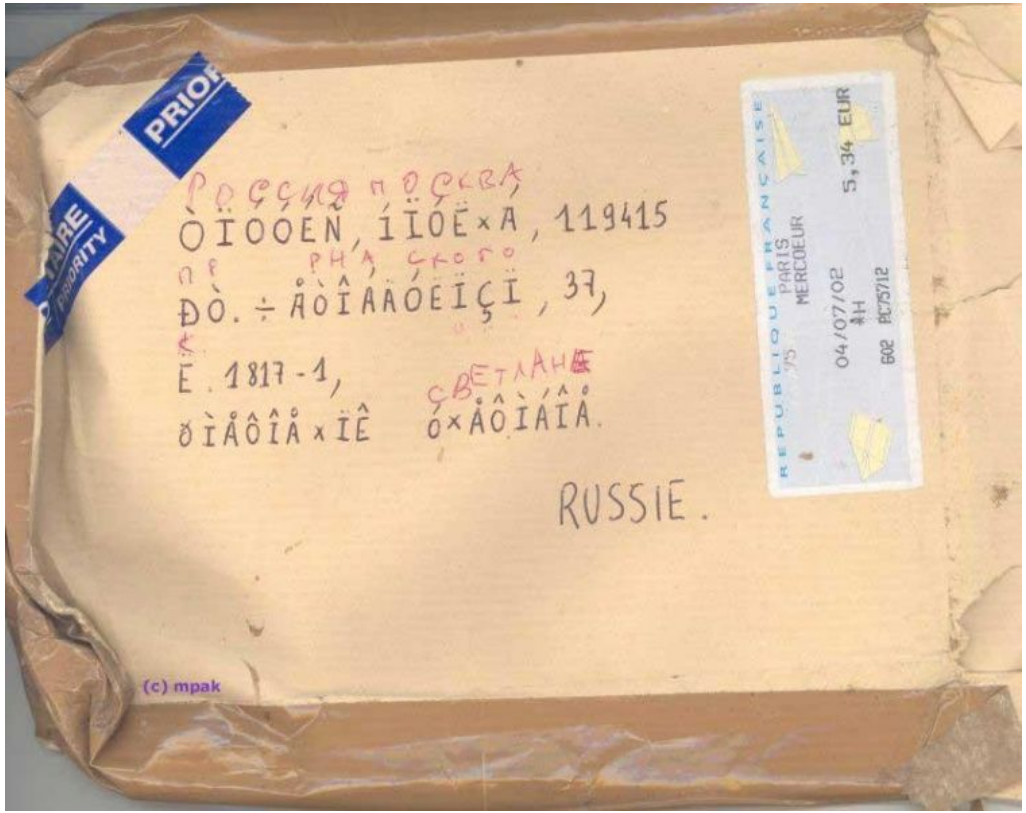
# Checklist

1. Correct project structure, packages, .gitignore
2. IReader, IWriter interfaces
3. String IO implementation
4. Formatter uses IReader, IWriter
5. >10 JUnit tests of Formatter
6. <5 CheckStyle warnings
7. File IO Implementation, IClosable
8. `main()` uses File IO, opens and closes files
9. Maven builds the project
10. Runs successfully with `java -jar`

# Кракозябры

— Don't underestimate the power of charsets!

— Read chars, Luke!



# UTF8 → CP1251/CP866

```
val s = "У Лукоморья дуб зелёный"
val utf8Bytes = s.toByteArray(Charsets.UTF_8)

utf8Bytes.toString(charset("CP1251"))
РЈ Р›СґРєРsРjРsСЂСЊСЏ РґСґР± Р·РµР»С‘РSС‹ Р№

utf8Bytes.toString(charset("CP866"))
╨г ╨Ы╥Г╨│╨л╨Л╨л╥A╥M╥П ╨┤╥Г╨▒ ╨╗╨╢╨╗╥C╨Л╥Л╨╢
```

# CP866 → CP1251/UTF8

```
val s = "У Лукоморья дуб зелёный"
val cp866Bytes =
     s.toByteArray(charset("CP866"))

cp866Bytes.toString(charset("CP1251")
" ‹ г€®¬®амп ¤гЎ §Ґ«сл©
cp866Bytes.toString(Charsets.UTF_8)
□ □□□□□□ □□ □□□□□
```

# Reader like Iterator

```
class ReaderIterator(
        private val fileName: String
) : Iterator<Char> {
```

# Reader like Iterator

```
private val reader: Reader

private var readChar: Int = -1
```

# Reader like Iterator

```
init {

    reader = Files.newBufferedReader(
        Paths.get(fileName),
        StandardCharsets.UTF_8)

    readChar = reader.read()

}
```

# Reader like Iterator

```
override fun hasNext(): Boolean {
    return readChar >= 0
}
```

# Reader like Iterator

```
override fun next(): Char {
    val prevChar = readChar
    readChar = reader.read()
    return prevChar.toChar()
}
```

# try-with-resources (Java 7)

```
try (
    Reader in = new FileReader("in.file");
    Writer out = new FileWriter("out.file")
) {
    format(in, out);
}
```

# Catch

```
} catch (Exception e) {
    e.printStackTrace();
}
```

# Catch

```
} catch (Exception e) {
    logger.error("Format failed", e);
}
```

# Logs

```
System.out.println("Hello");
System.err.println("World");
new Exception("stack").printStackTrace();
```


```
Hello
World
java.lang.Exception: stack
    at example.logs.OutPrint.main(OutPrint.java:8)
```

# SLF4J

http://www.slf4j.org/

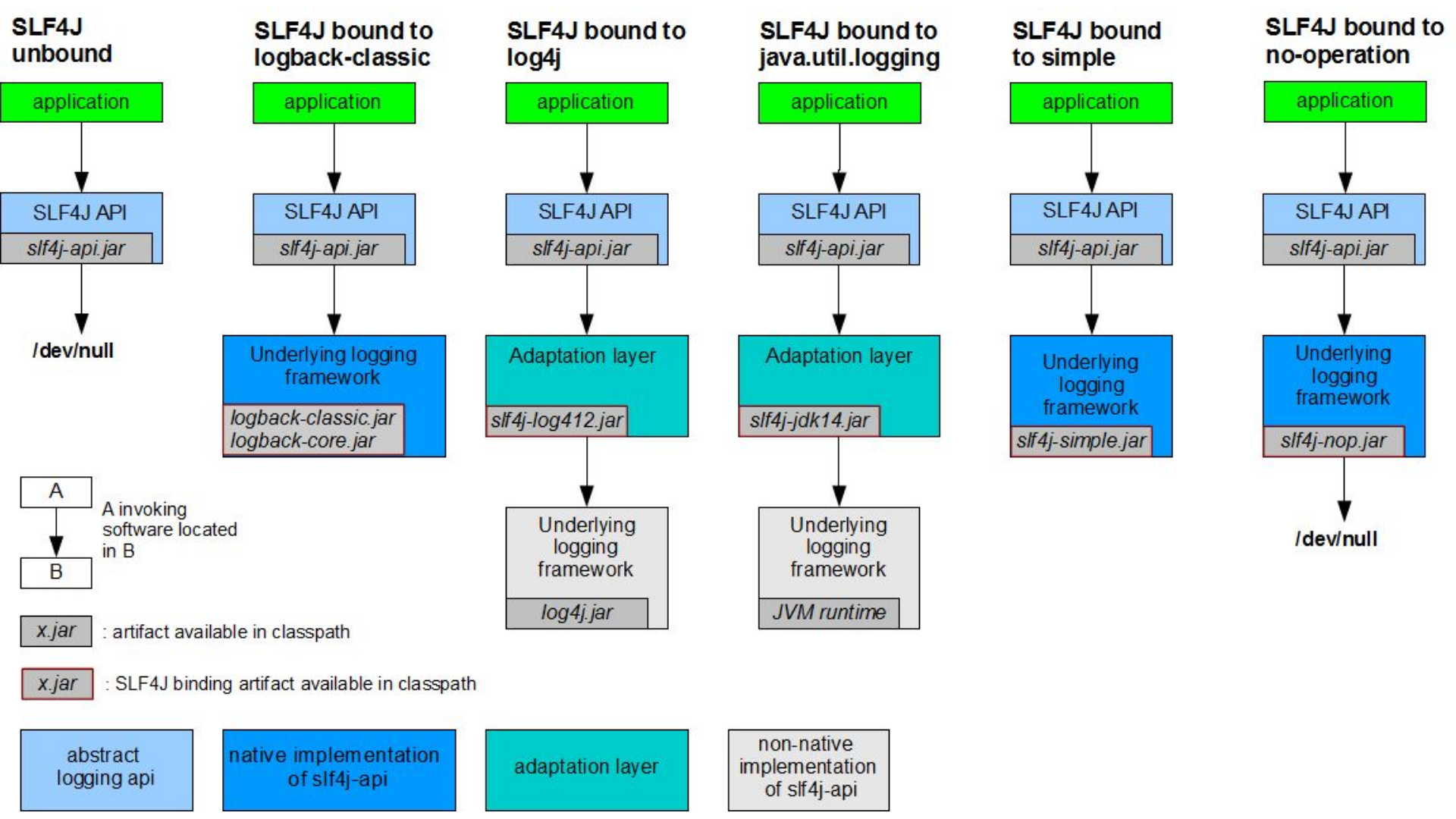
```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.25</version>
</dependency>
```

# Simple Logging Facade for Java

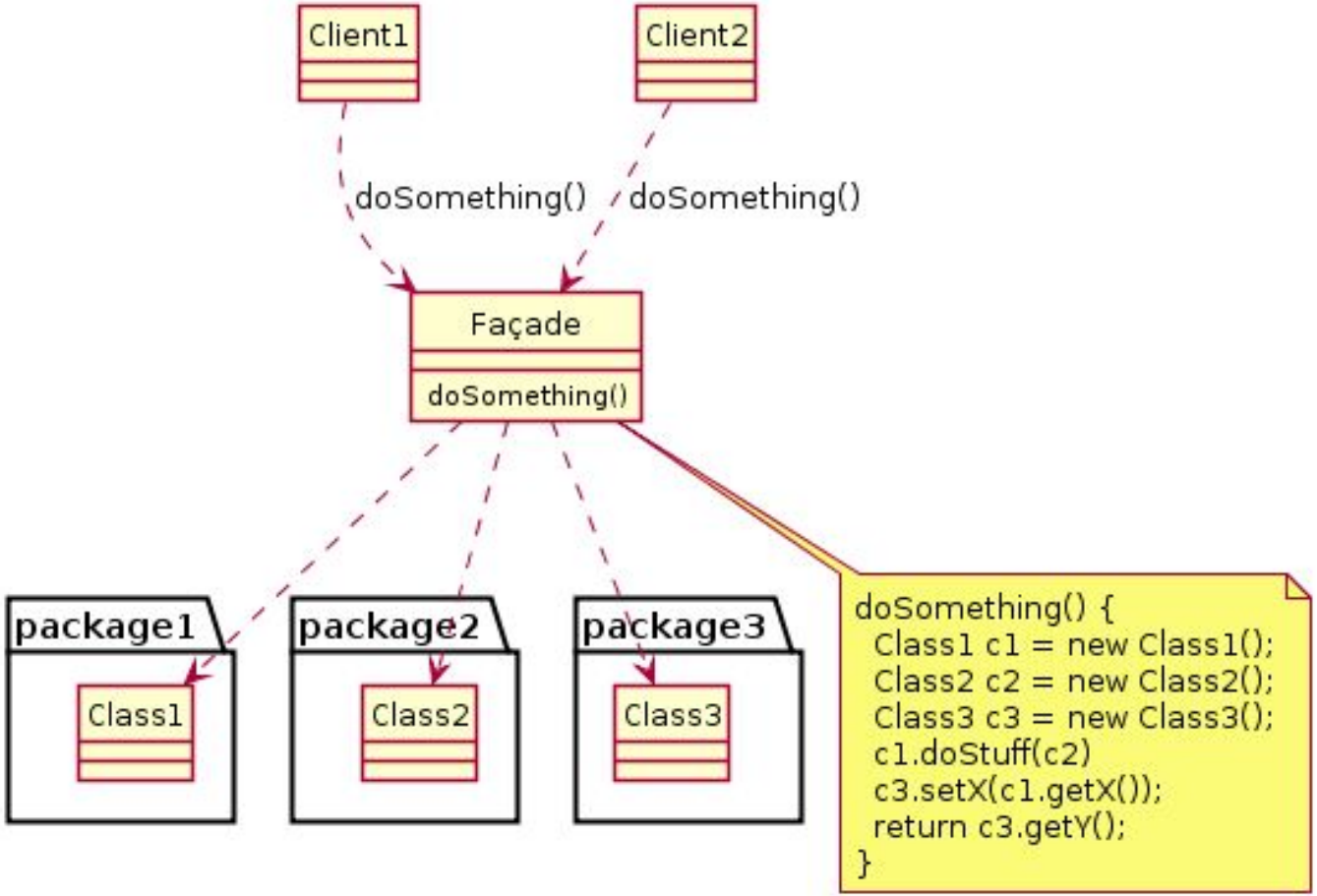
```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class SLF4JLogSample {
    final static Logger logger =
        LoggerFactory.getLogger(SLF4JLogSample.class);
    public static void main(String[] args) {
        logger.info("Hello");
        logger.warn("World");
        logger.error("error", new Exception("exception"));
    }
}
```

# SLF4J

```
SLF4J: Failed to load class
"org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger
implementation
SLF4J: See
http://www.slf4j.org/codes.html#StaticLoggerBinder for
further details.
```


SLF4J unbound
application
SLF4J API
slf4j-api.jar
/dev/null
SLF4J bound to logback-classic
application
SLF4J API
slf4j-api.jar
Underlying logging framework
logback-classic.jar
logback-core.jar
SLF4J bound to log4j
application
SLF4J API
slf4j-api.jar
Adaptation layer
slf4j-log412.jar
Underlying logging framework
log4j.jar
SLF4J bound to java.util.logging
application
SLF4J API
slf4j-api.jar
Adaptation layer
slf4j-jdk14.jar
Underlying logging framework
JVM runtime
SLF4J bound to simple
application
SLF4J API
slf4j-api.jar
Underlying logging framework
slf4j-simple.jar
SLF4J bound to no-operation
application
SLF4J API
slf4j-api.jar
Underlying logging framework
slf4j-nop.jar
/dev/null
A
B
A invoking software located in B
x.jar
: artifact available in classpath
x.jar
: SLF4J binding artifact available in classpath
abstract logging api
native implementation of slf4j-api
adaptation layer
non-native implementation of slf4j-api

# Facade


Client1
Client2
doSomething()
doSomething()
Façade
doSomething()
package1
Class1
package2
Class2
package3
Class3
doSomething() {
Class1 c1 = new Class1();
Class2 c2 = new Class2();
Class3 c3 = new Class3();
c1.doStuff(c2)
c3.setX(c1.getX());
return c3.getY();
}


not
invented
here

# SLF4J SimpleLogger

```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-simple</artifactId>
    <version>1.7.25</version>
</dependency>
```

# SLF4J SimpleLogger

```
[main] INFO example.logs.SLF4JLogSample - Hello
[main] WARN example.logs.SLF4JLogSample - World
[main] ERROR example.logs.SLF4JLogSample - error
java.lang.Exception: exception
    at
example.logs.SLF4JLogSample.main(SLF4JLogSample.java:13)
```

# Log Level

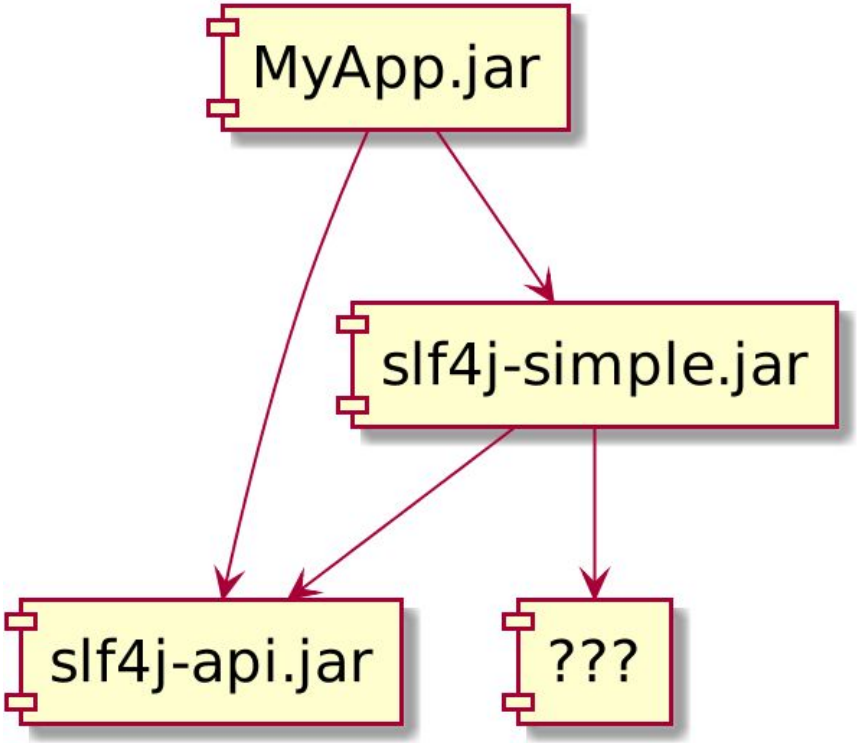
```
logger.trace("tracing (calls)");
logger.debug("debugging (values)");
logger.info("informing (events)");
logger.warn("warning (recoverable errors)");
logger.error("alerting (hard errors)");
```

# Optimizations

```
String name = "World";

if (logger.isDebugEnabled()) {
    logger.debug("Hello, {}!", name);
}
```

# Dependencies



# Jar with Dependencies

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-assembly-plugin</artifactId>
    <version>3.1.0</version>
    <configuration>
        <descriptorRefs>
        <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
        <archive>
            <manifest>
```

```
<configuration>
    <descriptorRefs>
    <descriptorRef>jar-with-dependencies</descriptorRef>
    </descriptorRefs>
    <archive><manifest>
            <mainClass>example.Main</mainClass>
    </manifest></archive>
</configuration>
<executions>
    <execution>
        <phase>package</phase>
        <goals><goal>single</goal></goals>
    </execution>
</executions>
```

# Maven Assembly Plugin

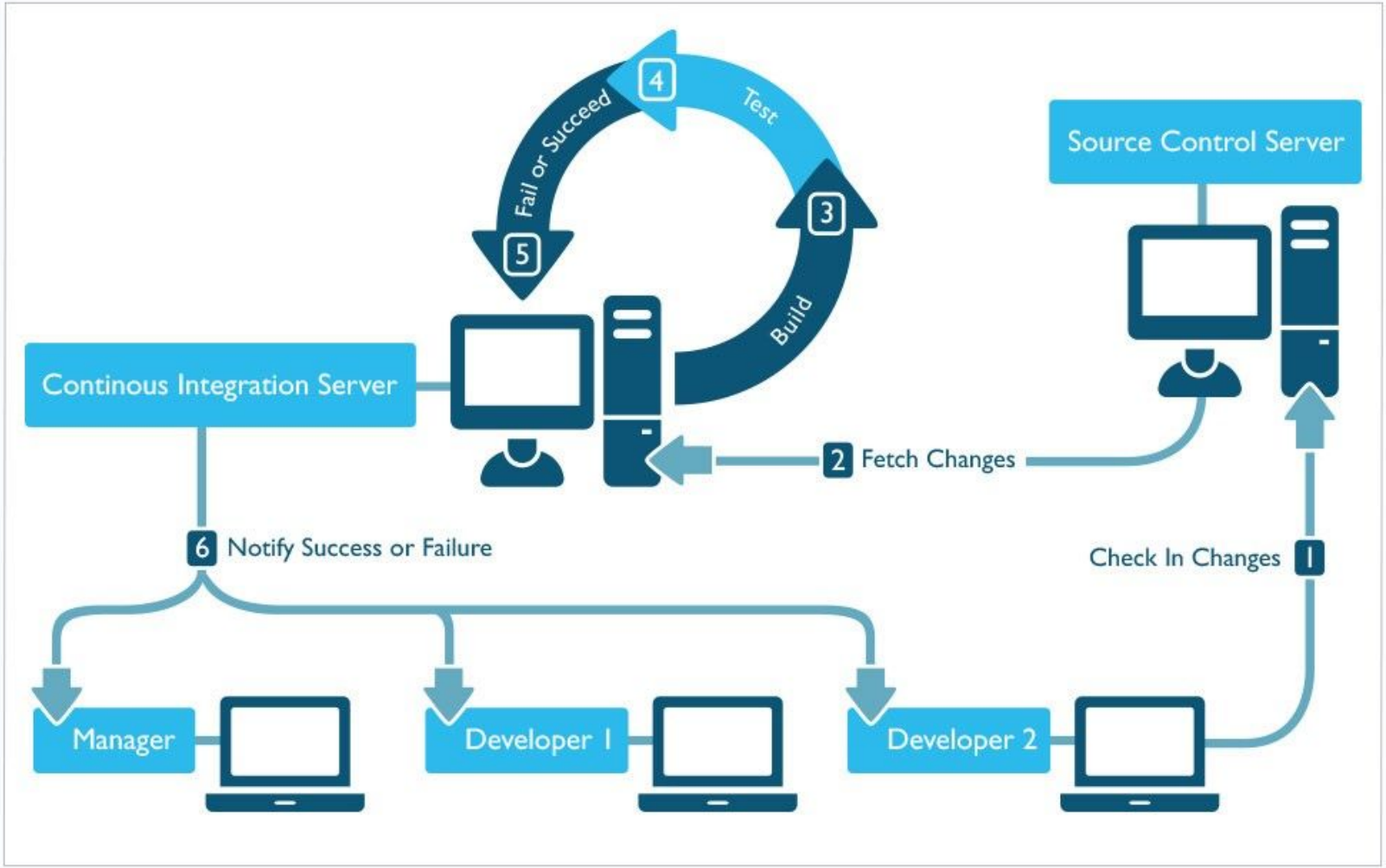
Build:

```
mvn package
```

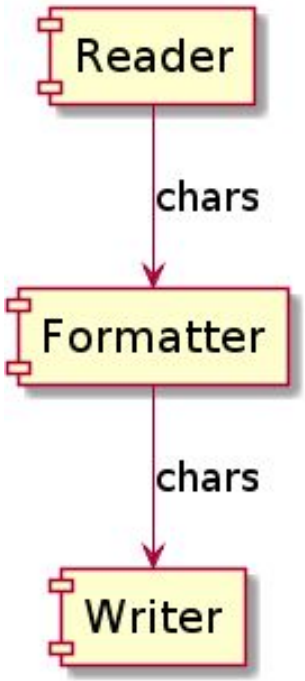
Run:

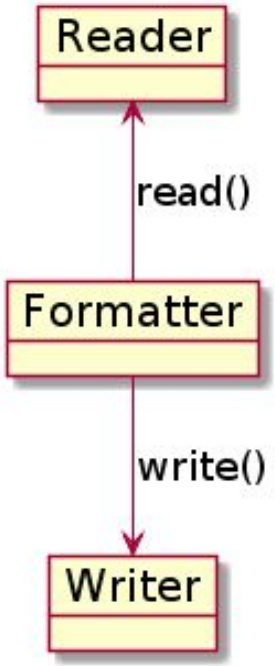
```
java -jar
target/MyApp-1.0-SNAPSHOT-jar-with-dependencies.jar
```

# CI

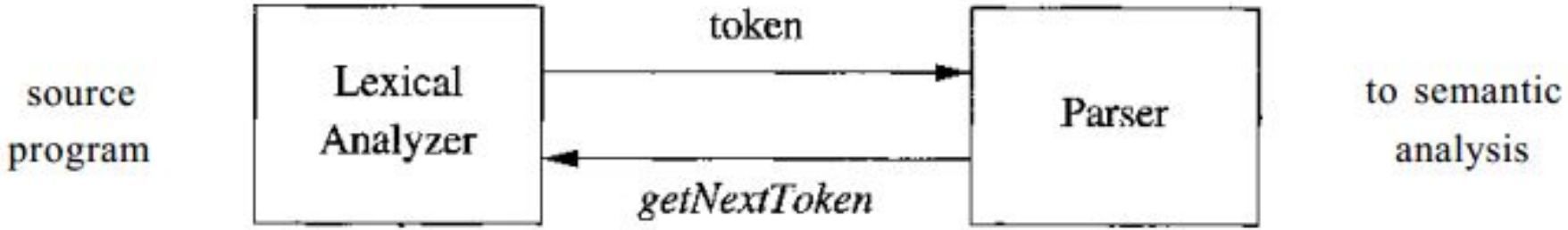

Source Control Server
Continous Integration Server
Test
Build
Fail or Succeed
4
3
5
2 Fetch Changes
6 Notify Success or Failure
Check In Changes 1
Manager
Developer 1
Developer 2

# Formatter


Reader
chars
Formatter
chars
Writer


Reader
read()
Formatter
write()
Writer

# Lexical Analysis



# Terms

- Lexer, Tokenizer or Scanner — performs lexical analysis, lexing or tokenization
  - converts a sequence of characters into a sequence of tokens (strings with an assigned and thus identified meaning)
- Lexeme — a sequence of characters
- Token — a pair consisting of:
  - token name
    - separator, whitespace, comment,...
  - token value
    - lexeme

# Tokens

***C Code***

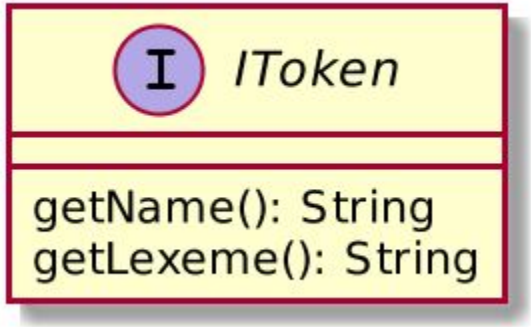
```
while (count <= 100) { /** some loop */
    count++;
    // Body of while continues
    ...
```


***Tokens***

```
while
(
count
<=
100
)
{
count
++
;
...
```

- Lexeme
- Meaning

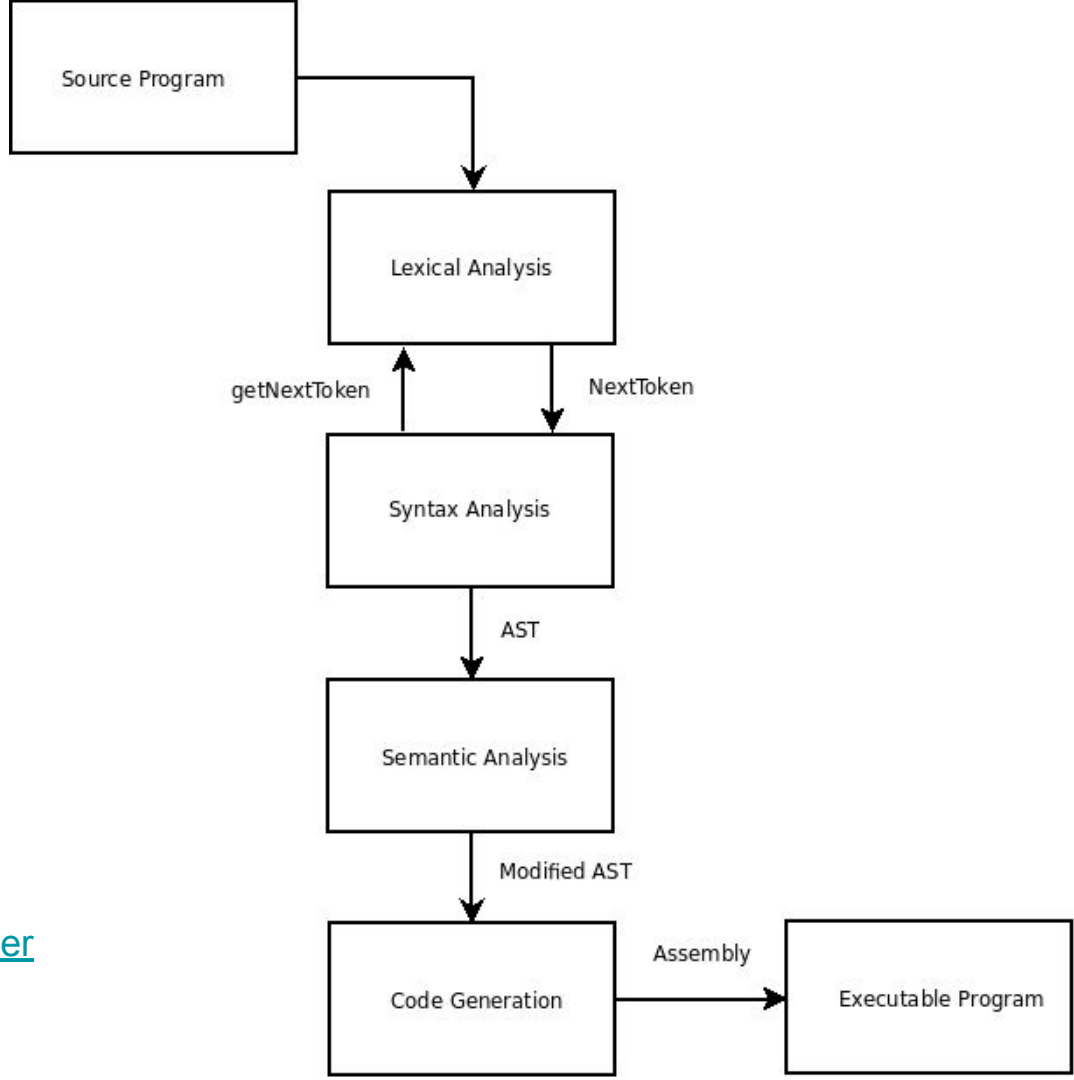
# Token



# Token Name

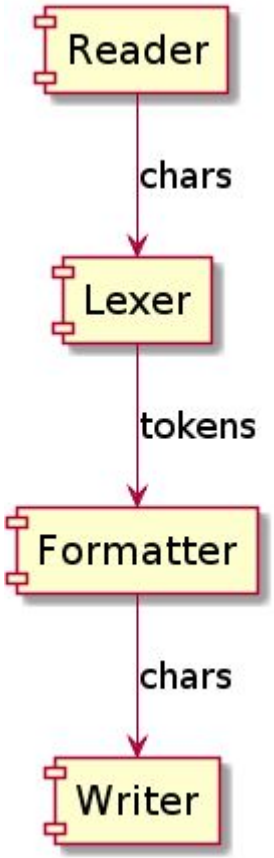
- ~~Integer~~
  - Explains nothing
- ~~Enum~~
  - Not extendable, not SOLID
- String
  - Self-explained, extendable, can be a key in HashMap
  - Spelling problem
    - Loose coupling

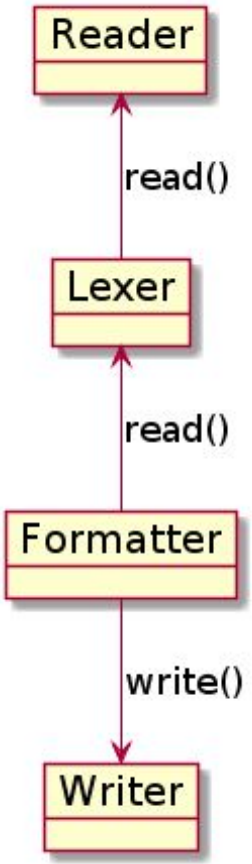
# Compiler




https://en.wikipedia.org/wiki/Multi-pass_compiler

# Lexer
# +
# Formatter



# Formatter

```
public void format(
        final ILexer lexer,
        final IWriter writer) {
    while (lexer.hasMoreTokens()) {
        IToken token = lexer.readToken();
        String lexeme = token.getLexeme();
        // do something
        write(writer, lexeme);
    }
}
```

# main()

```
IReader reader = new FileReader(args[0]);
IWriter writer = new FileWriter(args[1]);

ILexer lexer = new Lexer(reader);
IFormatter formatter = new Formatter();
// formatter.format(reader, writer);
formatter.format(lexer, writer);
```

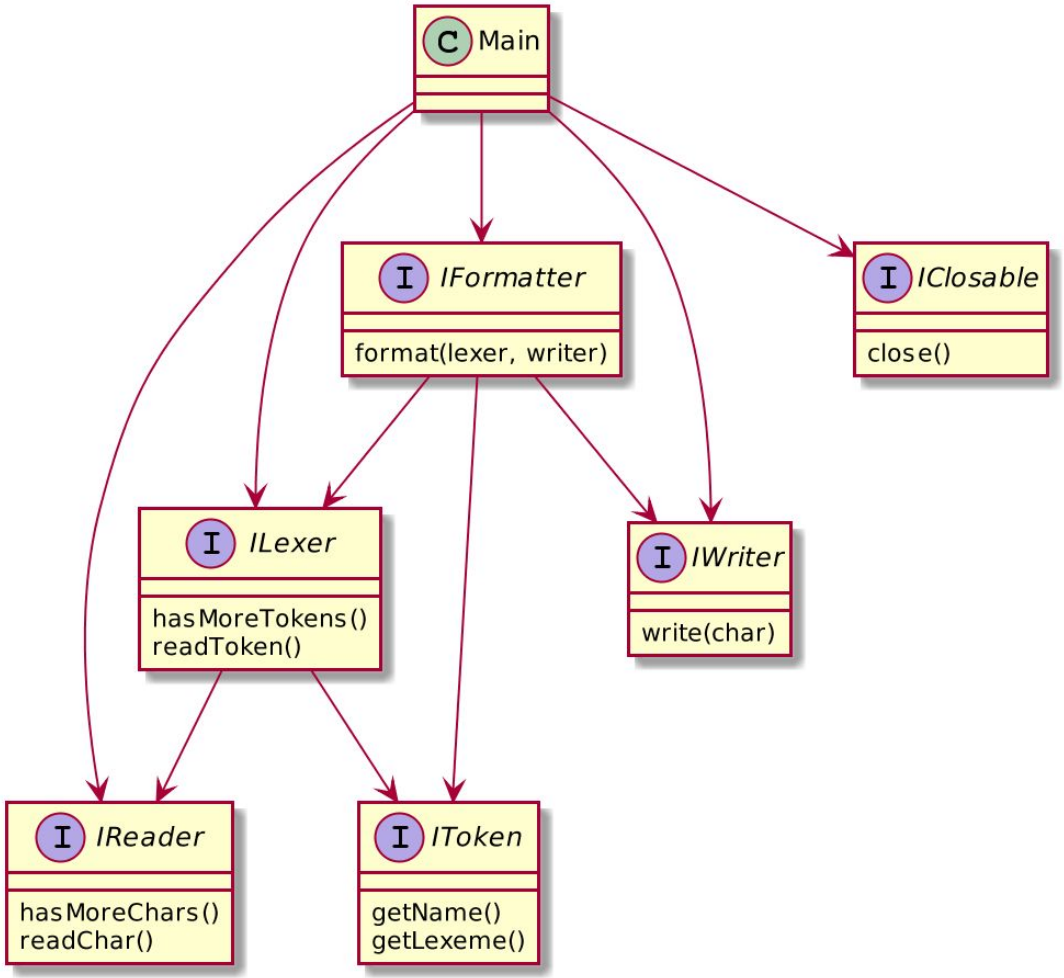
# Lexer

```
public IToken readToken() {
    while (reader.hasMoreChars()) {
        char c = reader.readChar();
        // append character to current lexeme
        // OR
        return newToken();
    }
    return newToken();
}
```

| | |
|---|---|
| S | Single responsibility principle |
| O | Open/closed principle |
| L | Liskov substitution principle |
| I | Interface segregation principle |
| D | Dependency inversion principle |


Robert C. Martin
"Uncle Bob"

# S — Single Responsibility

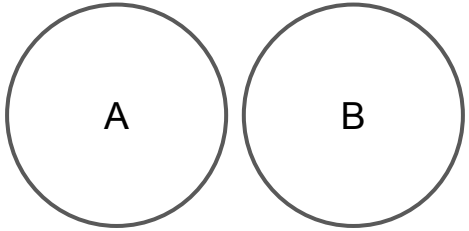


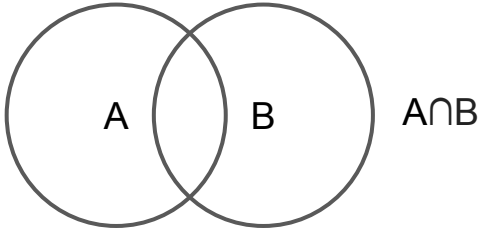
# O — Open/Closed

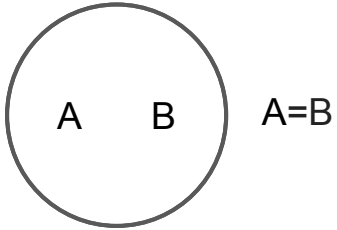
Software entities (classes, modules, functions, etc.)
should be open for extension,
but closed for modification.

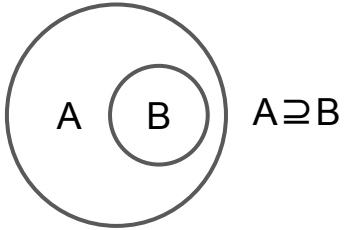
# Extends?

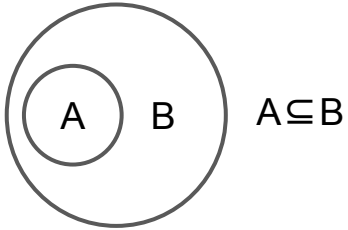
```
class B extends A
```

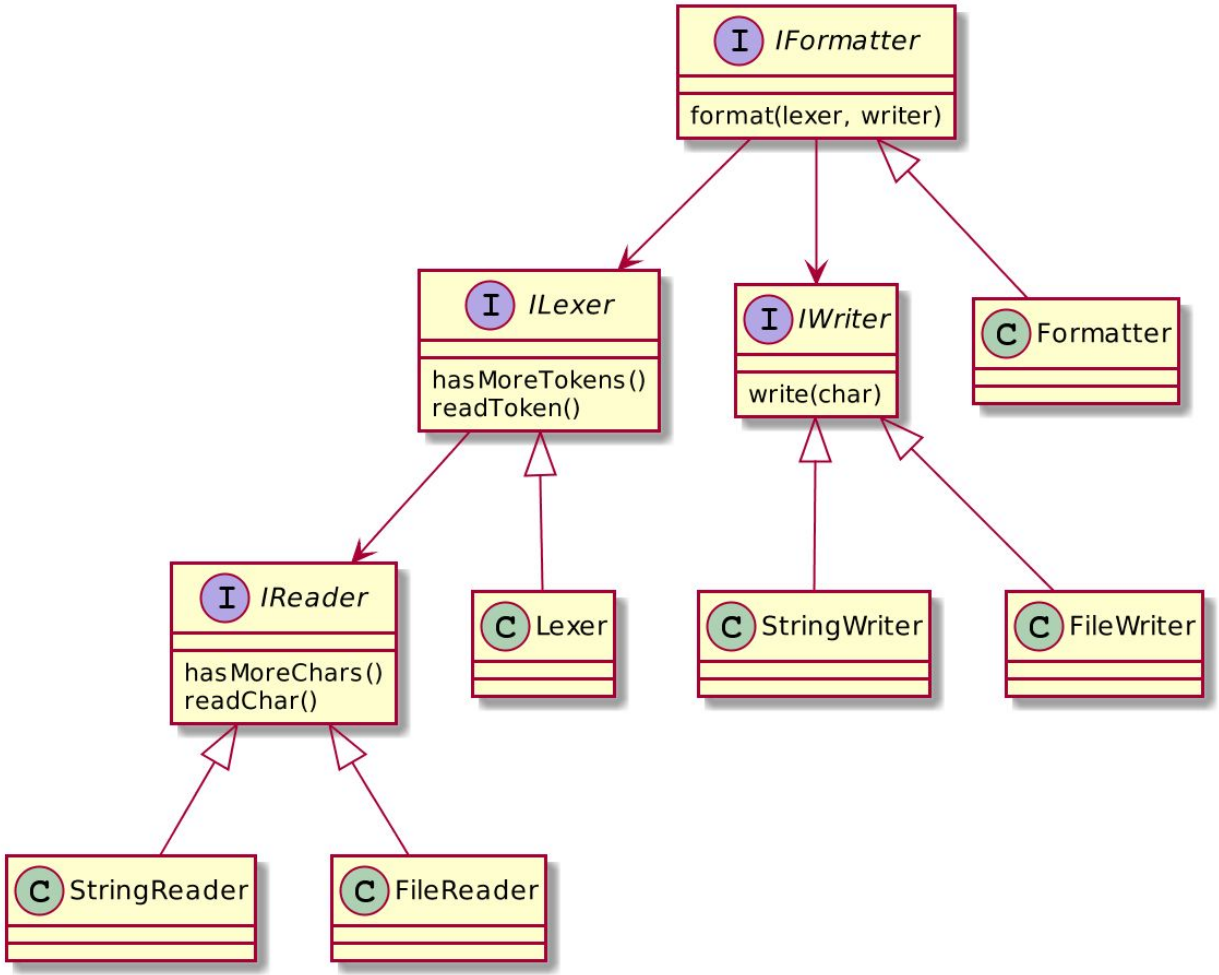



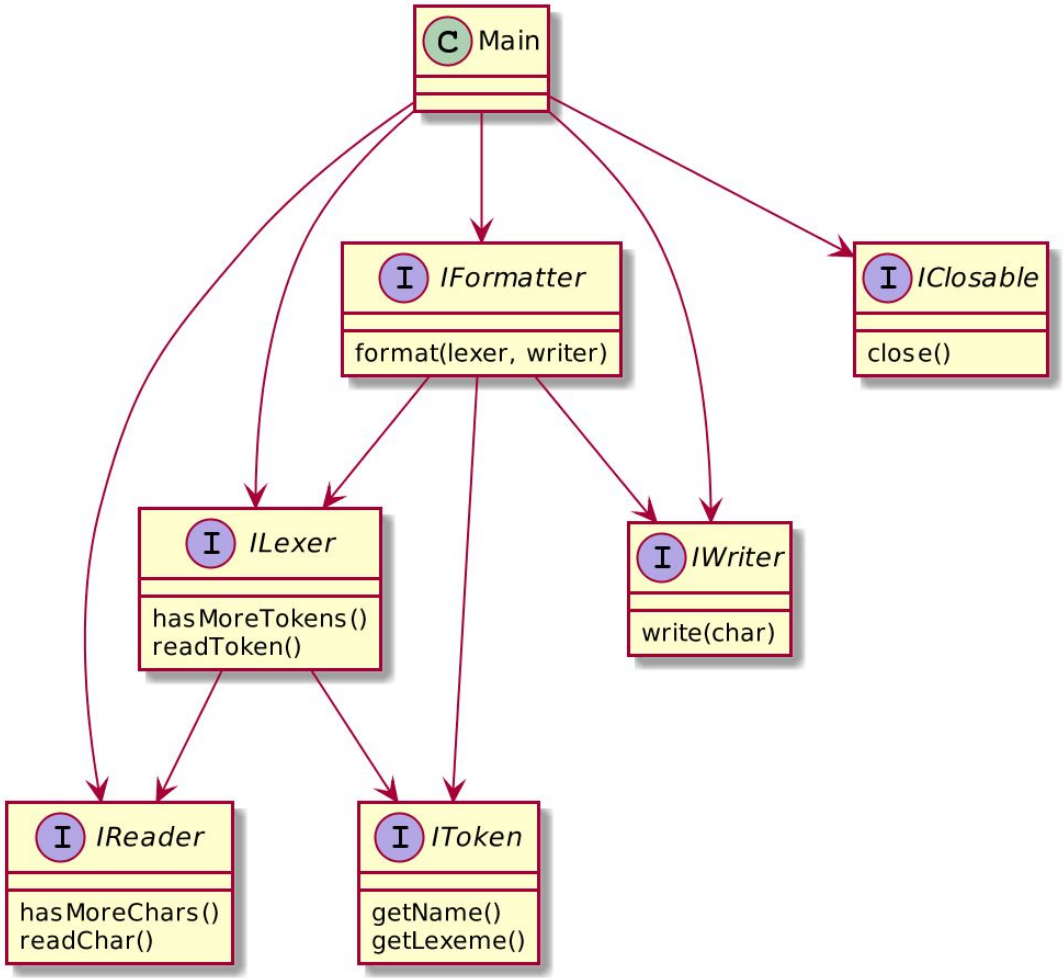



# L — Liskov Substitution

Barbara Liskov



# I — Interface Segregation

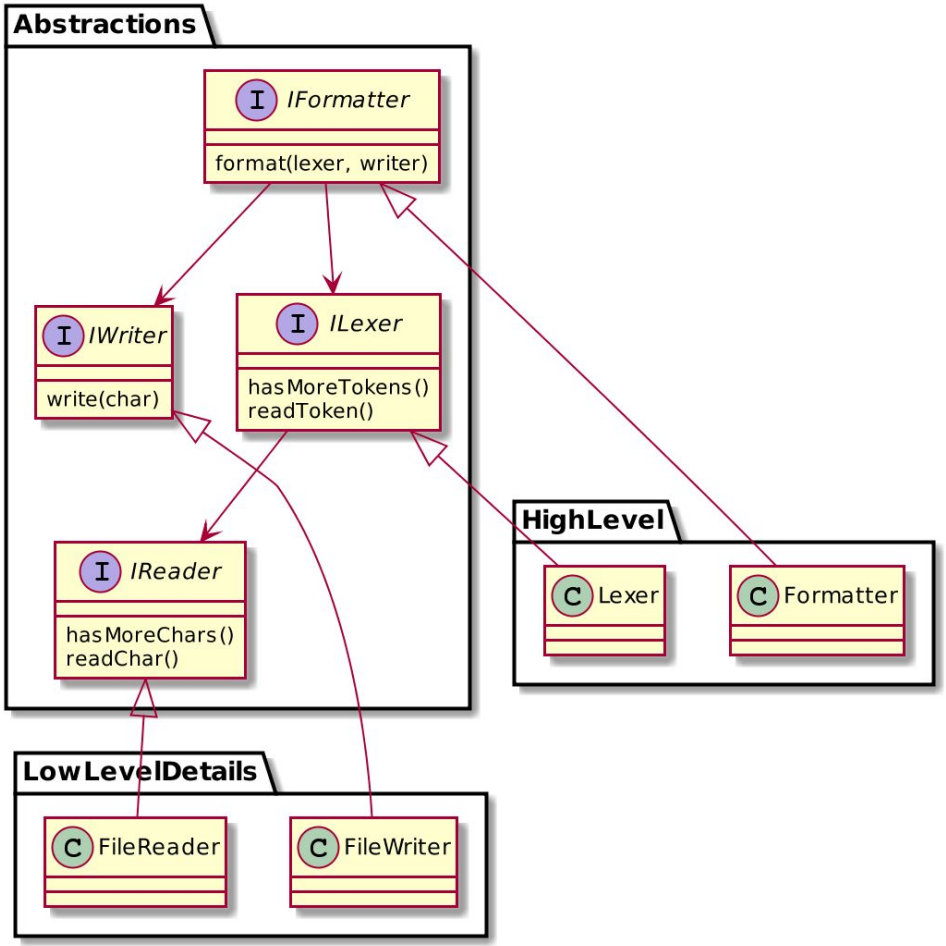



# D — Dependency Inversion

A. High-level modules should not
depend on low-level modules.
Both should depend on abstractions.

B. Abstractions should not depend on details.
Details should depend on abstractions.

# D — Dependency Inversion



# Dependency Inversion

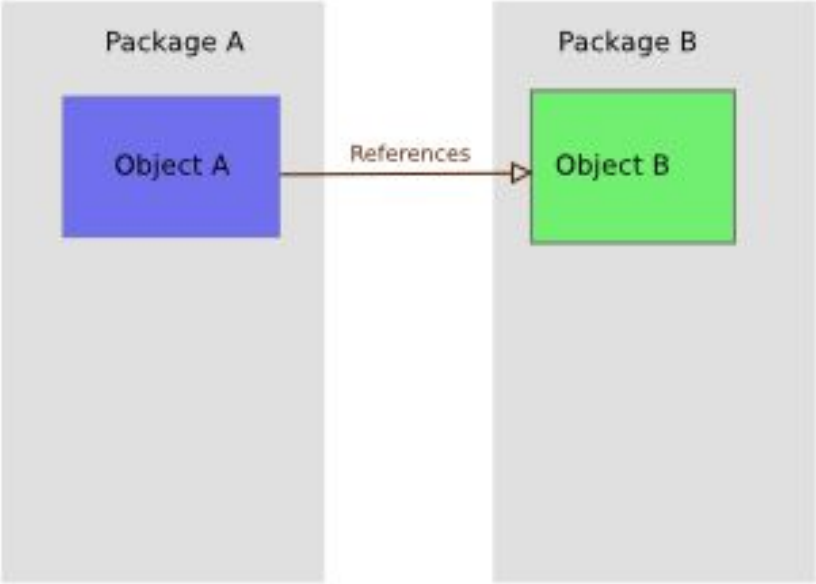



Figure 1

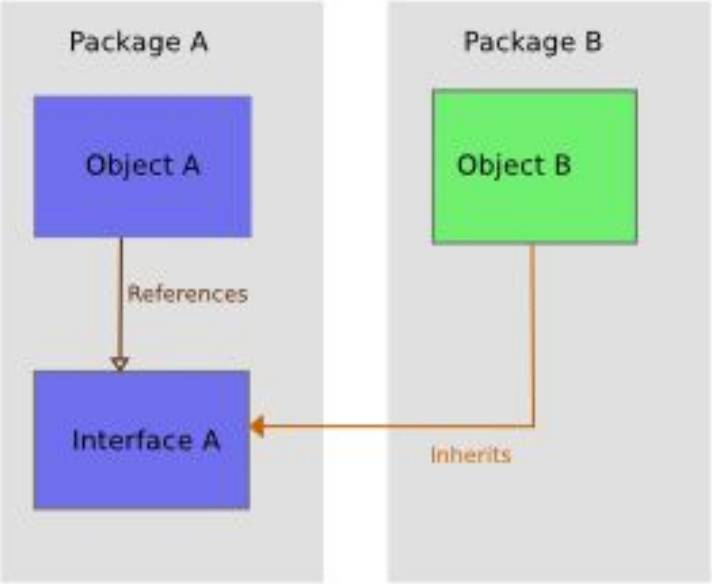



Figure 2

# Homework

- Complete the Checklist
  - https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)
- Add logging *where necessary*
- Make a Lexer
  - Reads the text
  - Extracts *significant* lexemes
  - Outputs lexemes
  - https://en.wikipedia.org/wiki/Lexical_analysis