Design Patterns

Template

COMP3607
Object Oriented Programming II

Week 7

Outline

- Design Patterns
  - Template

4

Template Design Pattern

- Define the skeleton of an algorithm in an operation, deferring some steps to client subclasses.
- Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.
- Base class declares algorithm 'placeholders', and derived classes implement the placeholders.

Variant vs Invariant Steps

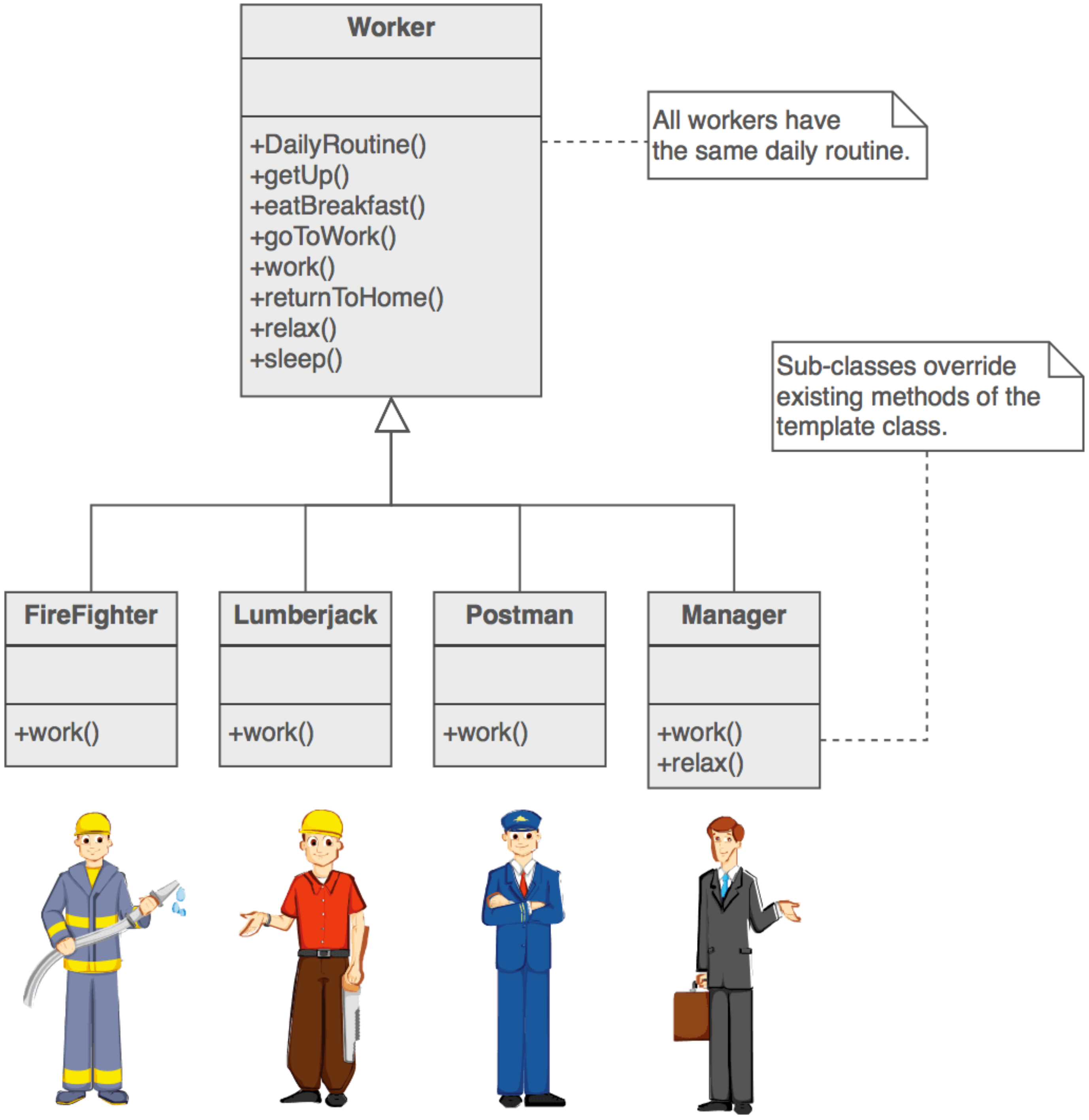
The component designer decides which steps of an algorithm are **invariant** (or standard), and which are **variant** (or customizable).

The invariant steps are implemented in an abstract base class.

The variant steps are either given a default implementation, or no implementation at all.

The variant steps represent "**hooks**", or "placeholders", that can, or must, be supplied by the component's client in a concrete derived class.

Example



Worker

+DailyRoutine()
+getUp()
+eatBreakfast()
+goToWork()
+work()
+returnToHome()
+relax()
+sleep()

All workers have the same daily routine.

Sub-classes override existing methods of the template class.

FireFighter

+work()

Lumberjack

+work()

Postman

+work()

Manager

+work()
+relax()

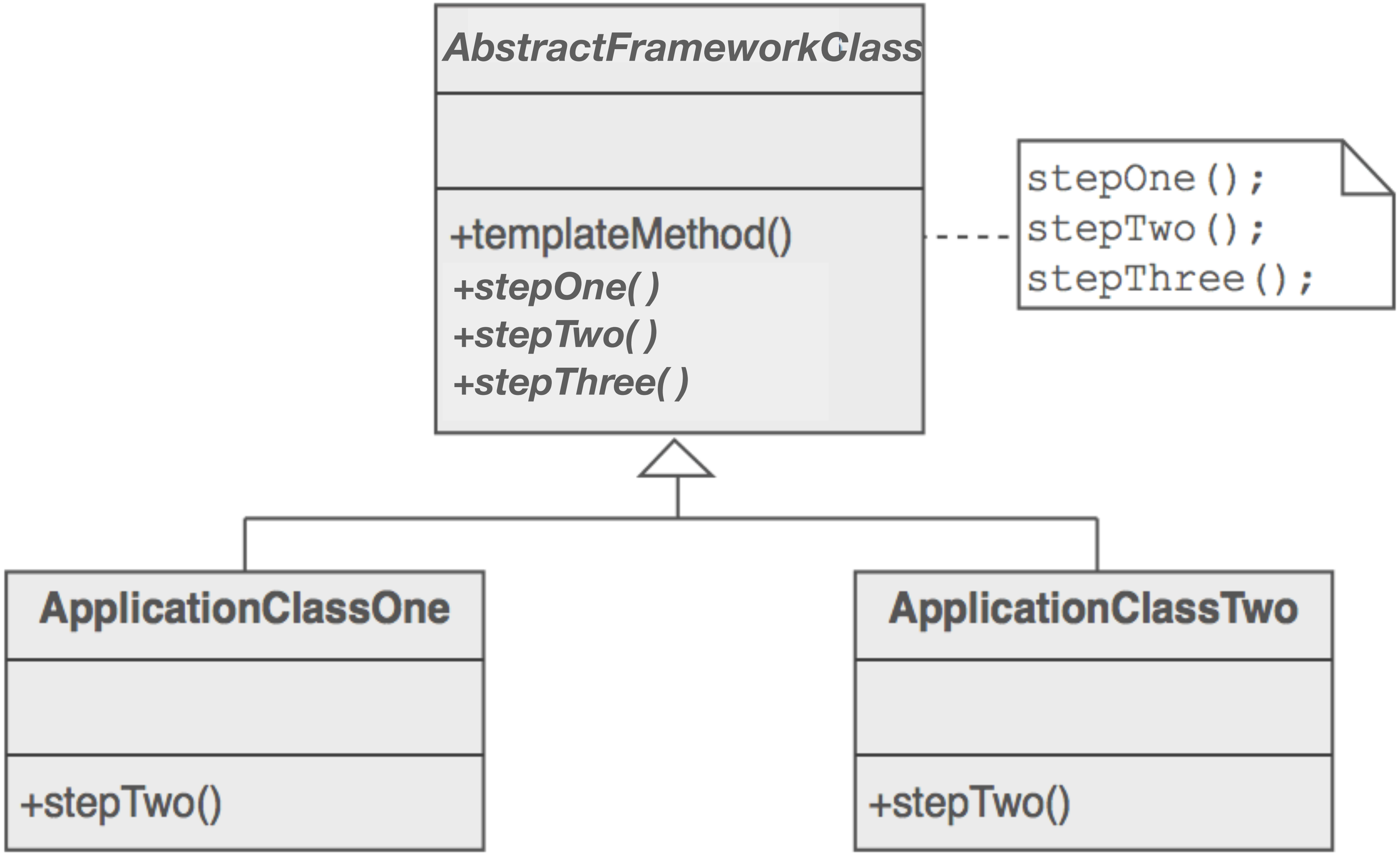
https://sourcemaking.com/design_patterns/template_method

Template Method Operations

- **Concrete methods:** Standard complete methods that are useful to the subclasses. These methods are usually utility methods.
- **Abstract methods**: Methods containing no implementation that must be implemented in subclasses.
- **Hook methods:** Methods containing a default implementation that may be overidden in some classes. Hook methods are intended to be overridden, concrete methods are not.
- **Template methods:** A method that calls any of the methods listed above in order to describe the algorithm without needing to implement the details.

4

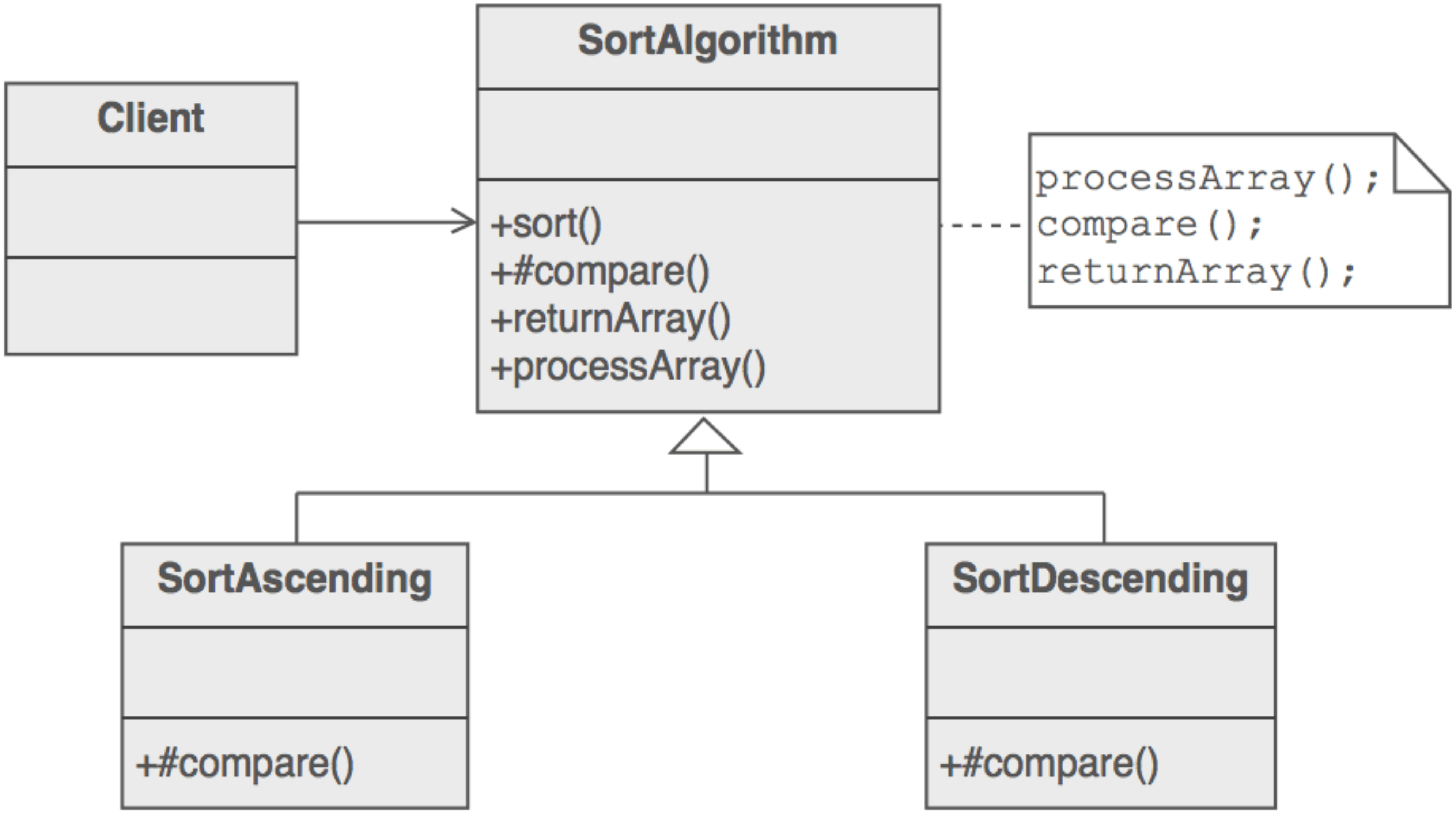
Template (UML)



```
stepOne();
stepTwo();
stepThree();
```

https://sourcemaking.com/design_patterns/template_method

Example

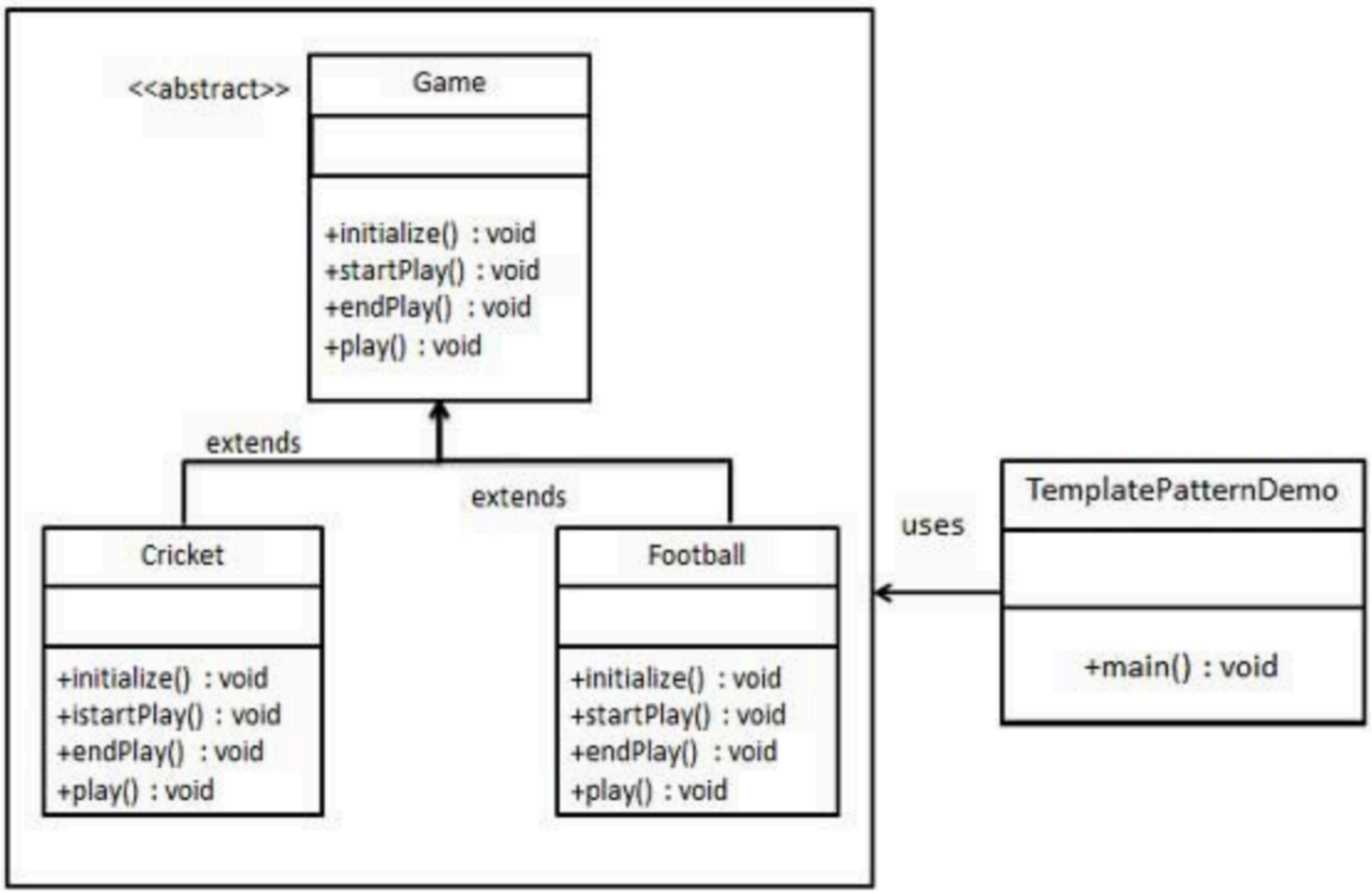


https://sourcemaking.com/design_patterns/template_method

Exercise

10 mins

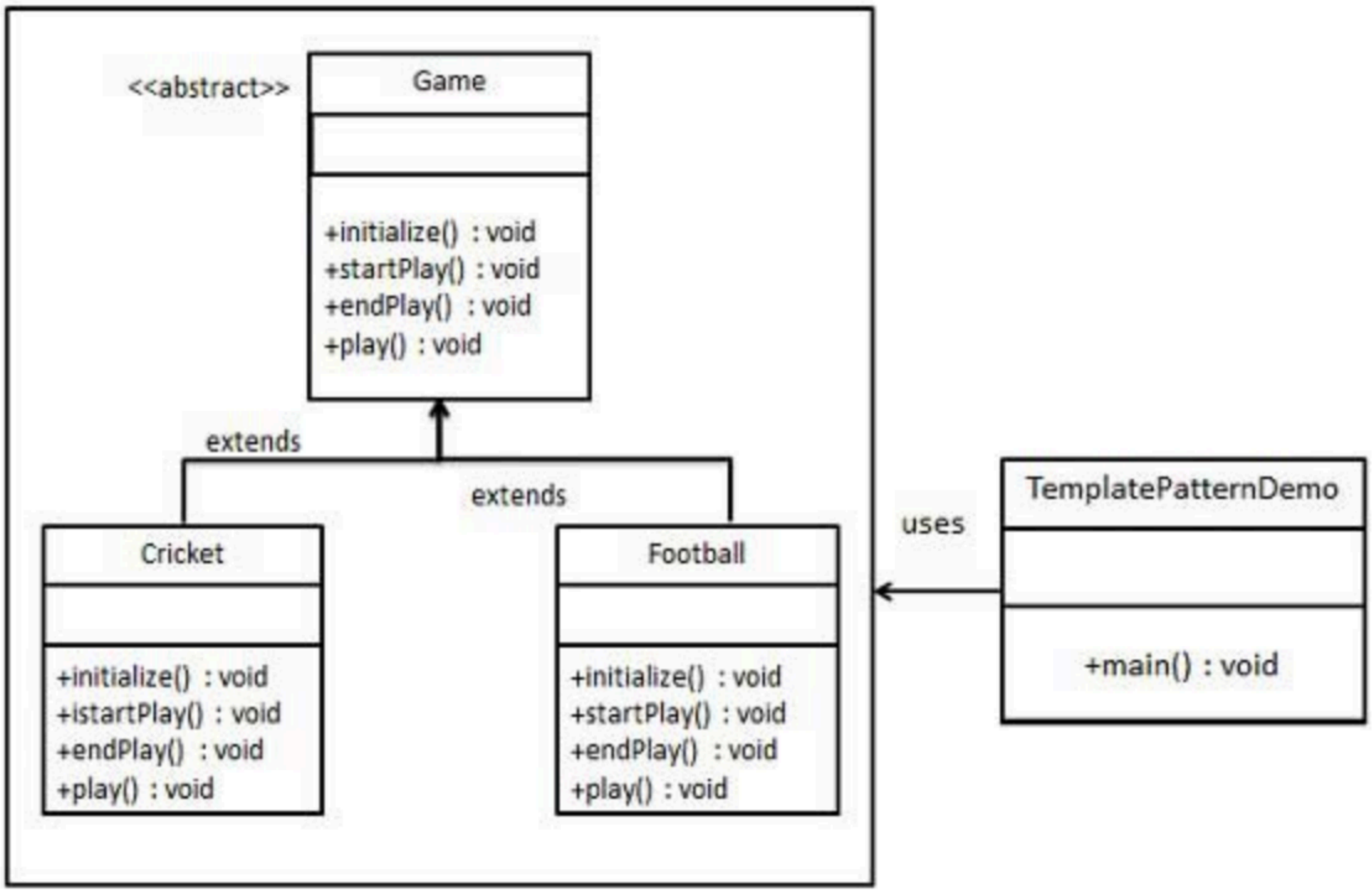
Exercise: Use the Template Design Pattern to write code for the classes specified in the diagram. Create a *Game* abstract class defining operations with a template method set to be final so that it cannot be overridden. *Cricket* and *Football* are concrete classes that extend *Game* and override its methods



https://www.tutorialspoint.com/design_pattern/template_pattern.htm

4

Code Example



4

Code Example

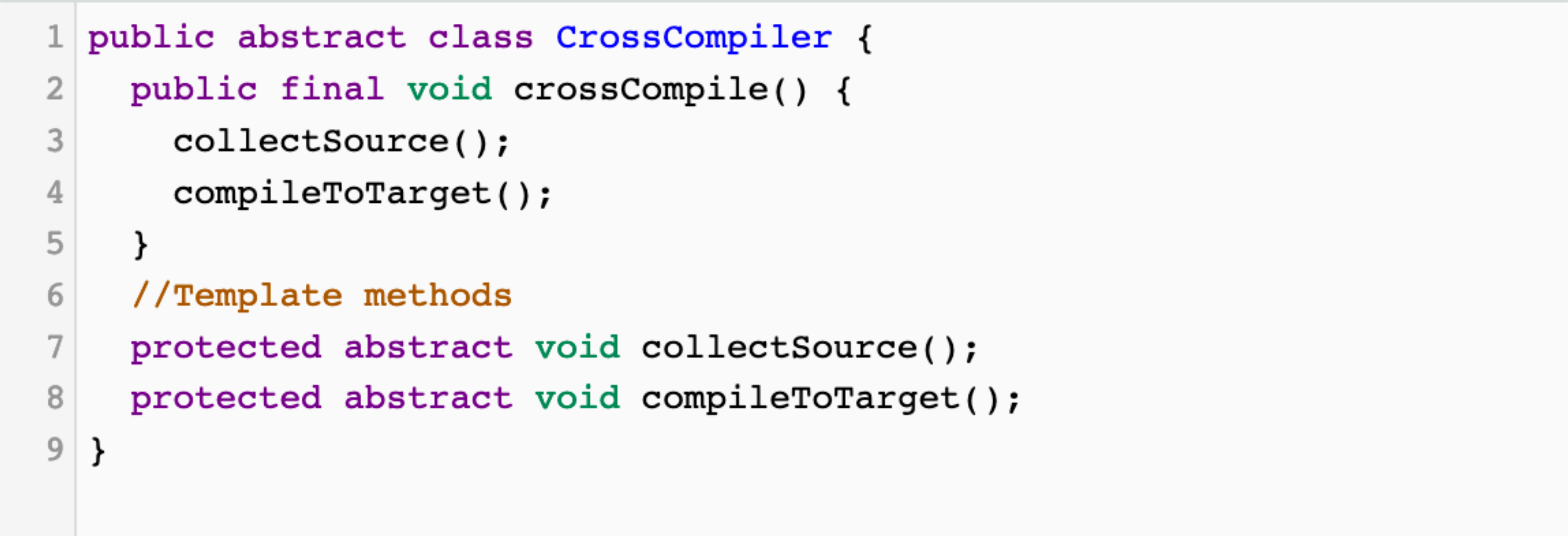
A cross-compiler is a compiler that runs on platform A (the **host**), but generates executables for platform B (the **target**). These two platforms may (but do not need to) differ in CPU, operating system, and/or executable format.

For example, a compiler that runs on a Windows 7 PC but generates code that runs on Android smartphone is a cross compiler

https://wiki.osdev.org/GCC_Cross-Compiler

4

Code Example

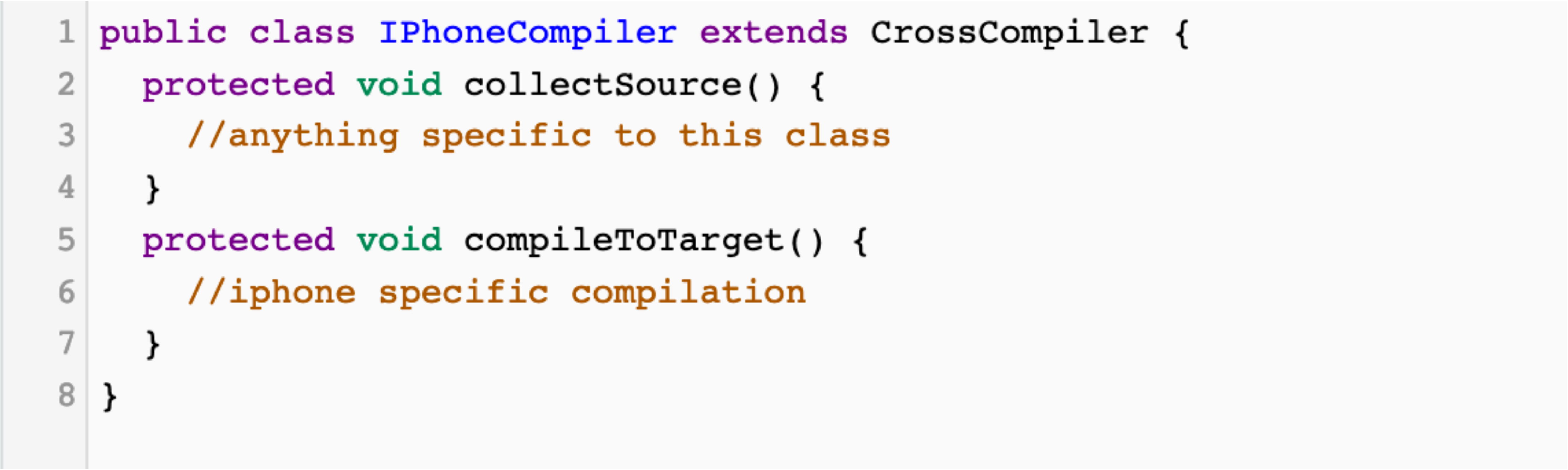


```java
public abstract class CrossCompiler {
  public final void crossCompile() {
    collectSource();
    compileToTarget();
  }
  //Template methods
  protected abstract void collectSource();
  protected abstract void compileToTarget();
}
```

https://dzone.com/articles/design-patterns-template-method

4

Code Example

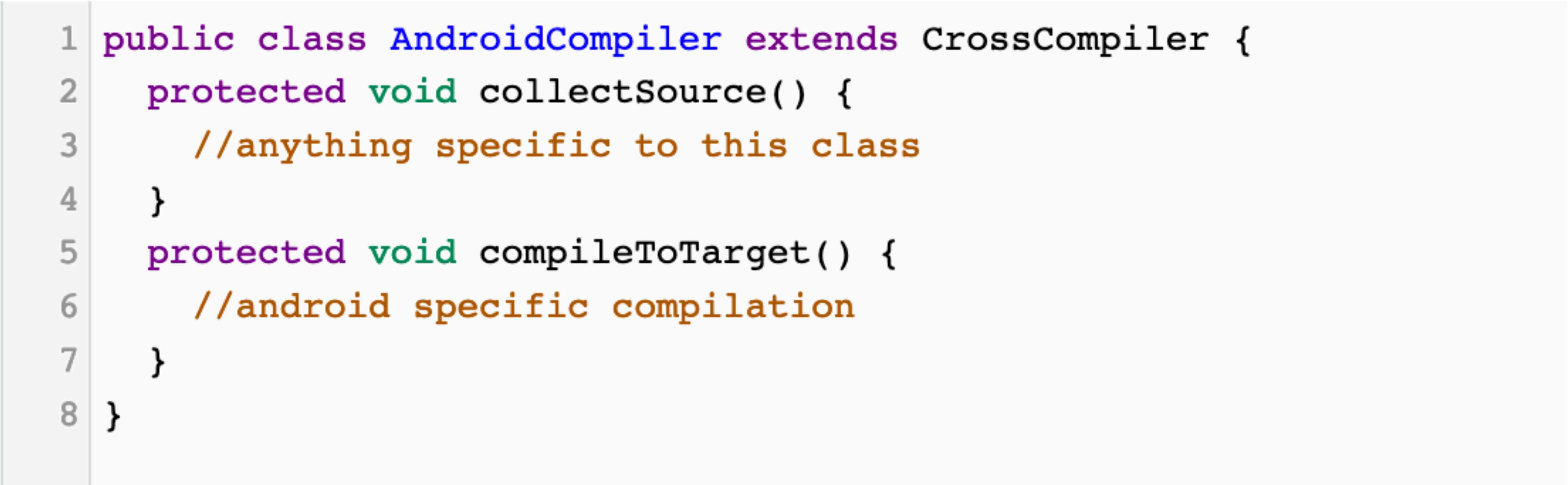


```
public class IPhoneCompiler extends CrossCompiler {
  protected void collectSource() {
    //anything specific to this class
  }
  protected void compileToTarget() {
    //iphone specific compilation
  }
}
```

https://dzone.com/articles/design-patterns-template-method

4

Code Example

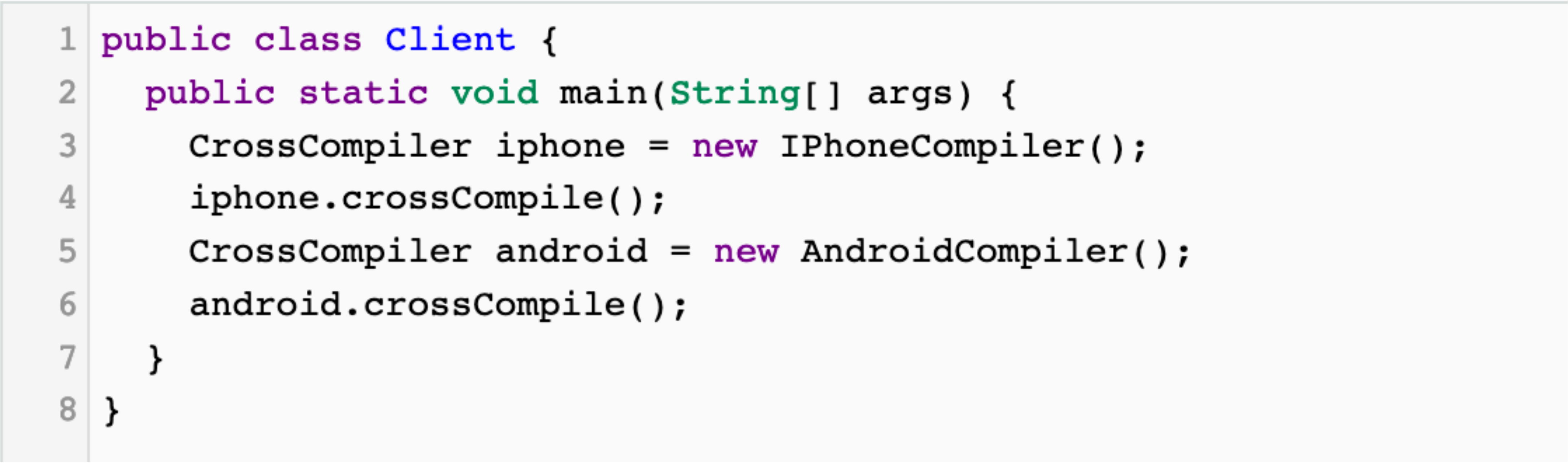


```
public class AndroidCompiler extends CrossCompiler {
  protected void collectSource() {
    //anything specific to this class
  }
  protected void compileToTarget() {
    //android specific compilation
  }
}
```

https://dzone.com/articles/design-patterns-template-method

4

Code Example



```java
public class Client {
  public static void main(String[] args) {
    CrossCompiler iphone = new IPhoneCompiler();
    iphone.crossCompile();
    CrossCompiler android = new AndroidCompiler();
    android.crossCompile();
  }
}
```

https://dzone.com/articles/design-patterns-template-method

4

Applicability: Template

Template design pattern is used:

- to implement the invariant parts of an algorithm once and leave the varying behaviour implementation details up to the subclasses
- when common behaviour among subclasses should be factored and localised in a common class to avoid code duplication.
- to control subclasses extensions.

4

Consequences: Template

- Template methods lead to an inverted control structure: “the Hollywood principle” - “Don’t call us, we’ll call you”
  - A parent class calls the operations of a subclass and not the other way around.
- Fundamental technique for code reuse especially for class libraries (factoring out common behaviour).
- Hook operations: provide default behaviour that subclasses extend by default.
  - Template methods must specify which operations are hooks (may be overridden) and which are abstract (must be overridden).

References

- Design Patterns: online reading resources
  - https://sourcemaking.com/design_patterns/template_method
  - https://dzone.com/articles/design-patterns-template-method