COMP2511

WEEK 9

Would you rather have 4 or 7 eyes?

ADMIN STUFF

- Assignment-ii due TOMORROW 5pm
- Assignment-iii 8%, no penalty until Week 11 Tuesday
- Week 10 Sample Exam

AGENDA

- Identitfy the Design Pattern
- Template Pattern
- Identitify the Code Smell

Finding Patterns

FINDING PATTERNS

Sorting collections of records in different orders.

FINDING PATTERNS

Strategy Pattern

This what Java does with the Collections.sort() method. A Comparator can be provided to determine the order in which elements are sorted.

FINDING PATTERNS

Modelling a file system.

FINDING PATTERNS

Composite pattern

Both folders and files are filesystem entries. Files form the leaves, folders can contain files or other folders.

FINDING PATTERNS

Updating a UI component when the state of a program changes.

FINDING PATTERNS

Observer pattern.

If the state of the program is the subject and the UI an observer, the UI will be notified of any changes to the state and update accordingly.

FINDING PATTERNS

Parsing and evaluating arithmetic expressions.

FINDING PATTERNS

Composite pattern.

The composite pattern can be used to represent a parse-tree. An example of this is given in the code.

FINDING PATTERNS

Adjusting the brightness of a screen based on a light sensor.

FINDING PATTERNS

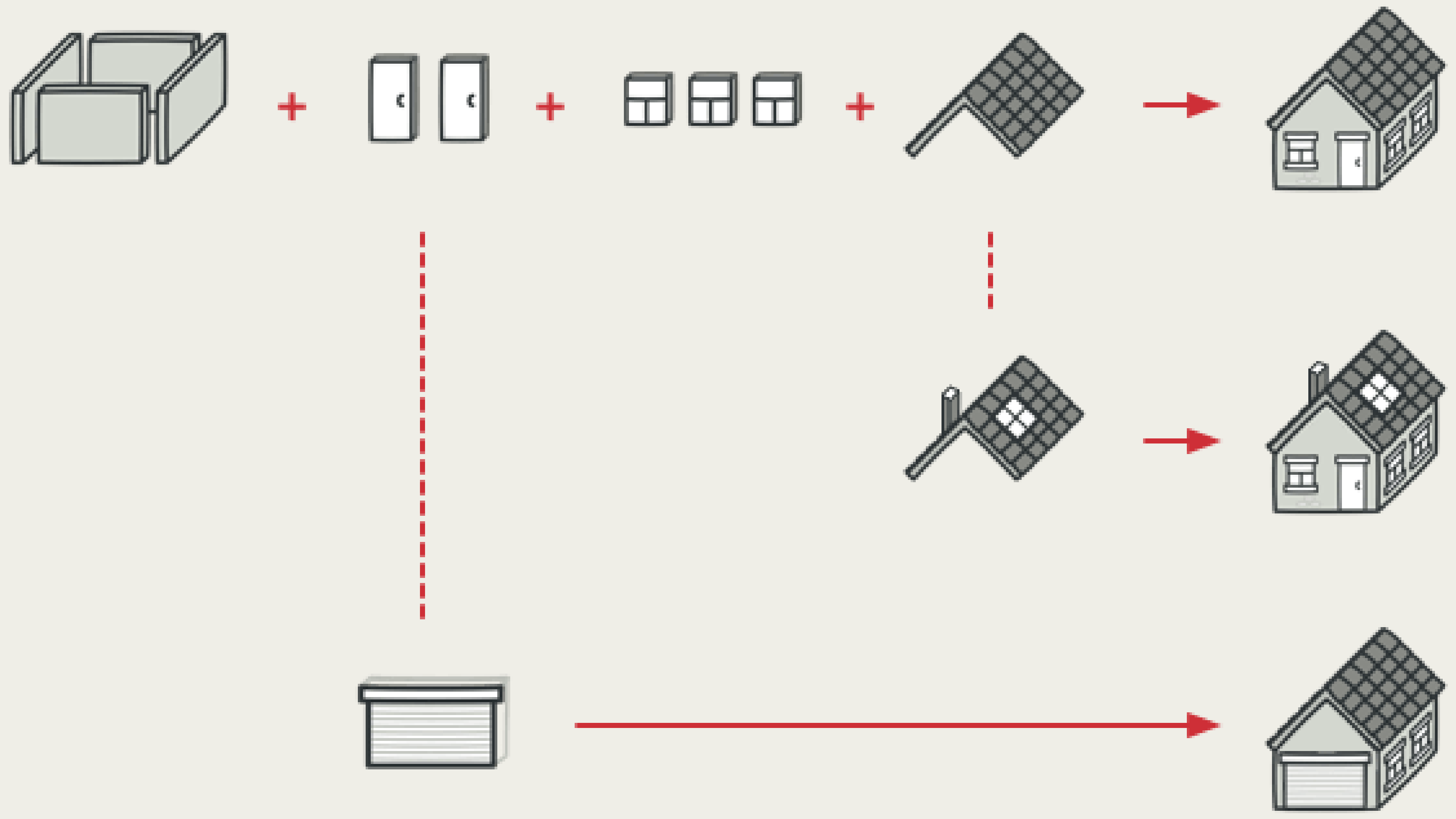
Observer pattern.

If the light sensor is the subject, the observer could be notified on all significant changes to the amount of light hitting the sensor and adjust the brightness of the screen accordingly.

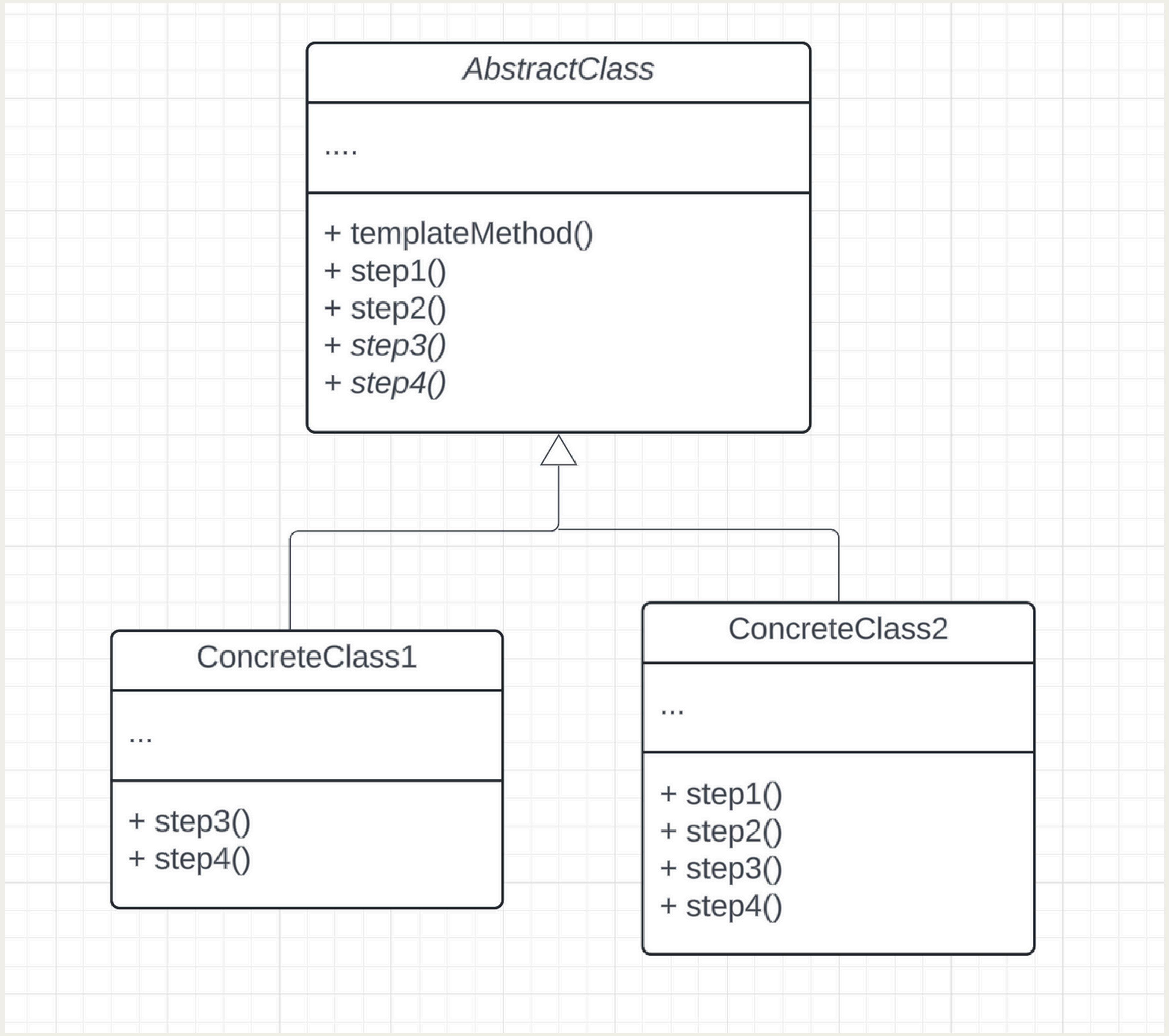
Template Method

TEMPLATE METHOD

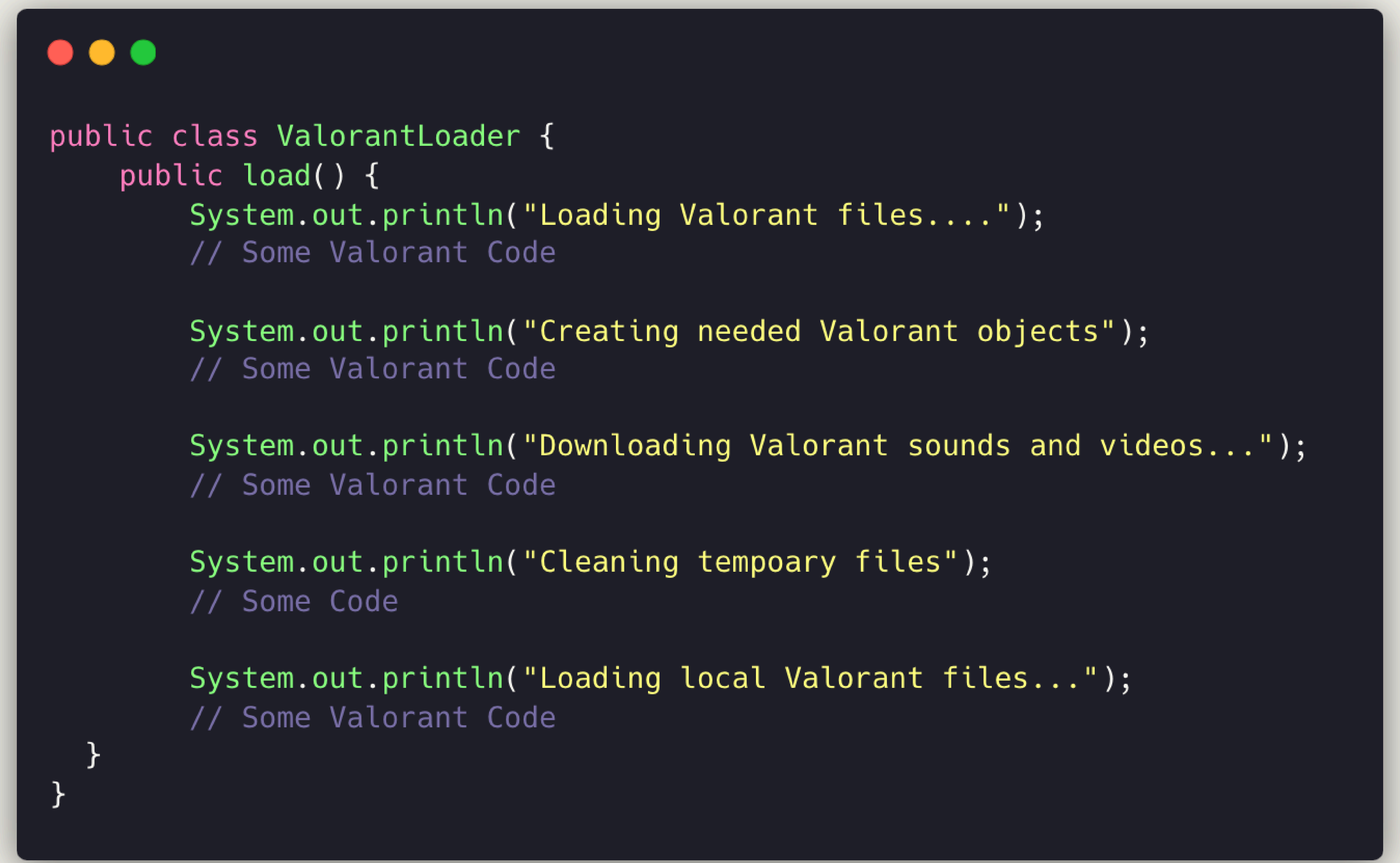
Template Method is a behavioral design pattern that defines the skeleton of an algorithm in the superclass but lets subclasses override specific steps of the algorithm without changing its structure.



TEMPLATE METHOD

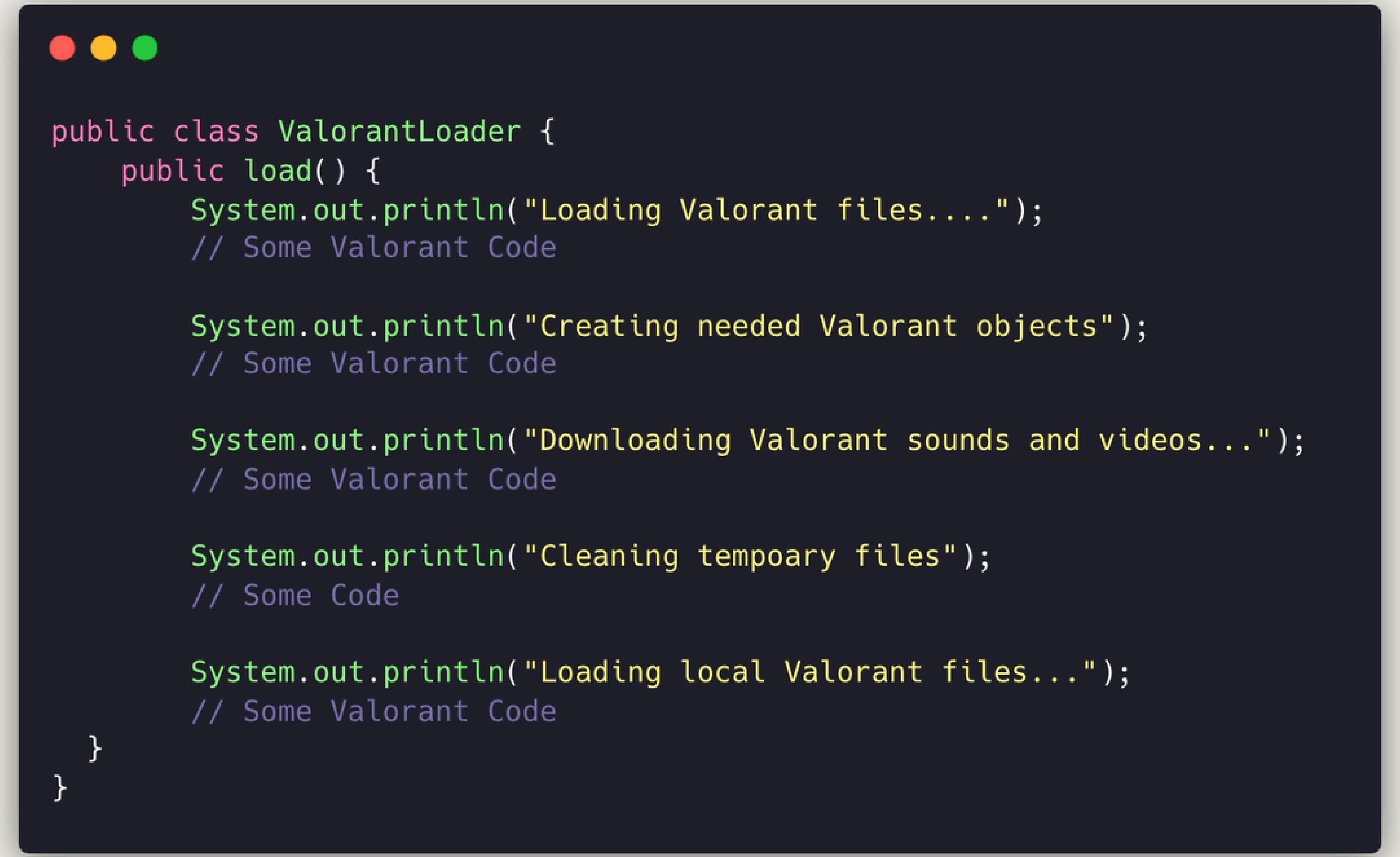


TEMPLATE METHOD

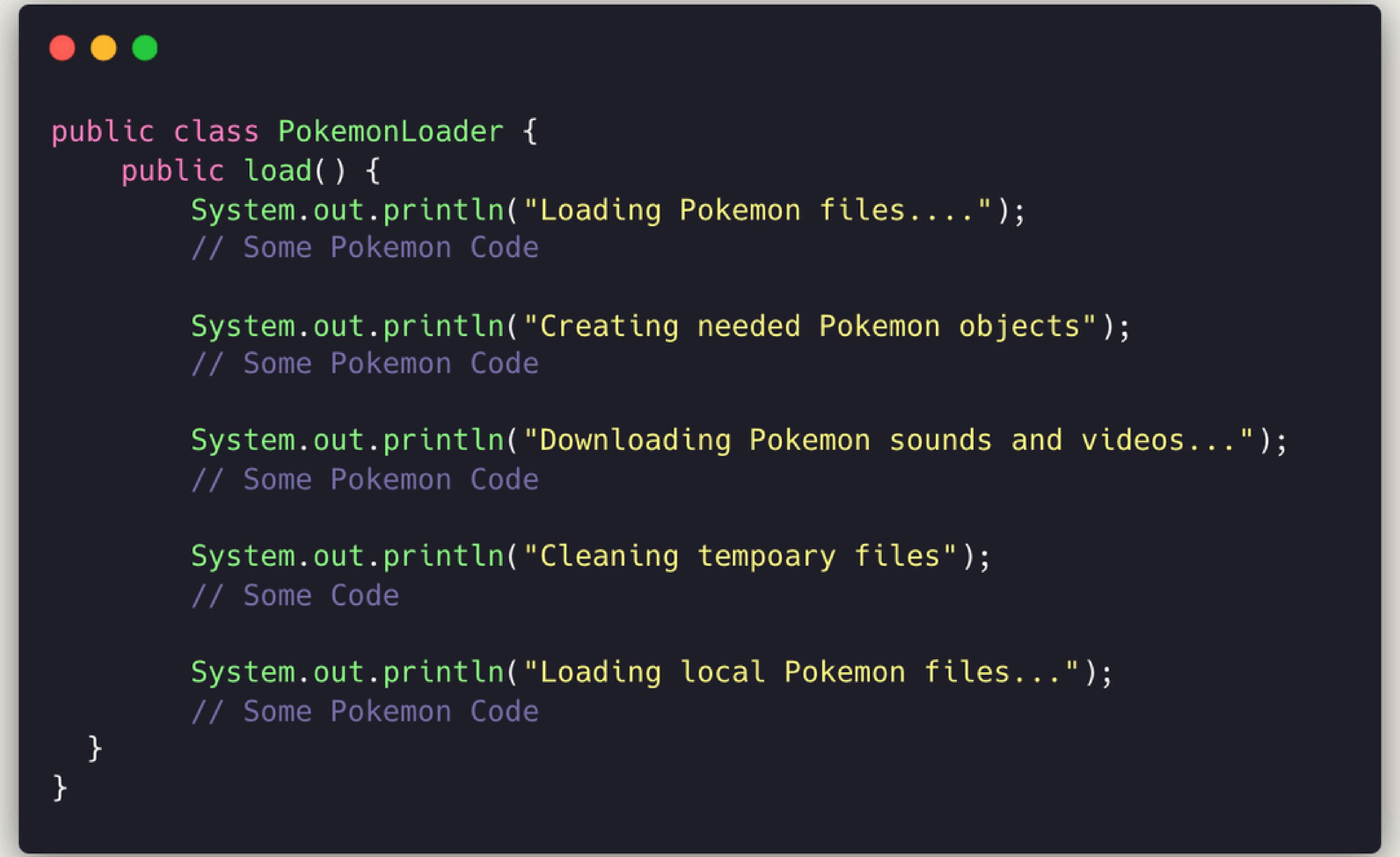


```
public class ValorantLoader {
    public load() {
        System.out.println("Loading Valorant files....");
        // Some Valorant Code

        System.out.println("Creating needed Valorant objects");
        // Some Valorant Code

        System.out.println("Downloading Valorant sounds and videos...");
        // Some Valorant Code

        System.out.println("Cleaning tempoary files");
        // Some Code

        System.out.println("Loading local Valorant files...");
        // Some Valorant Code
  }
}
```

TEMPLATE METHOD

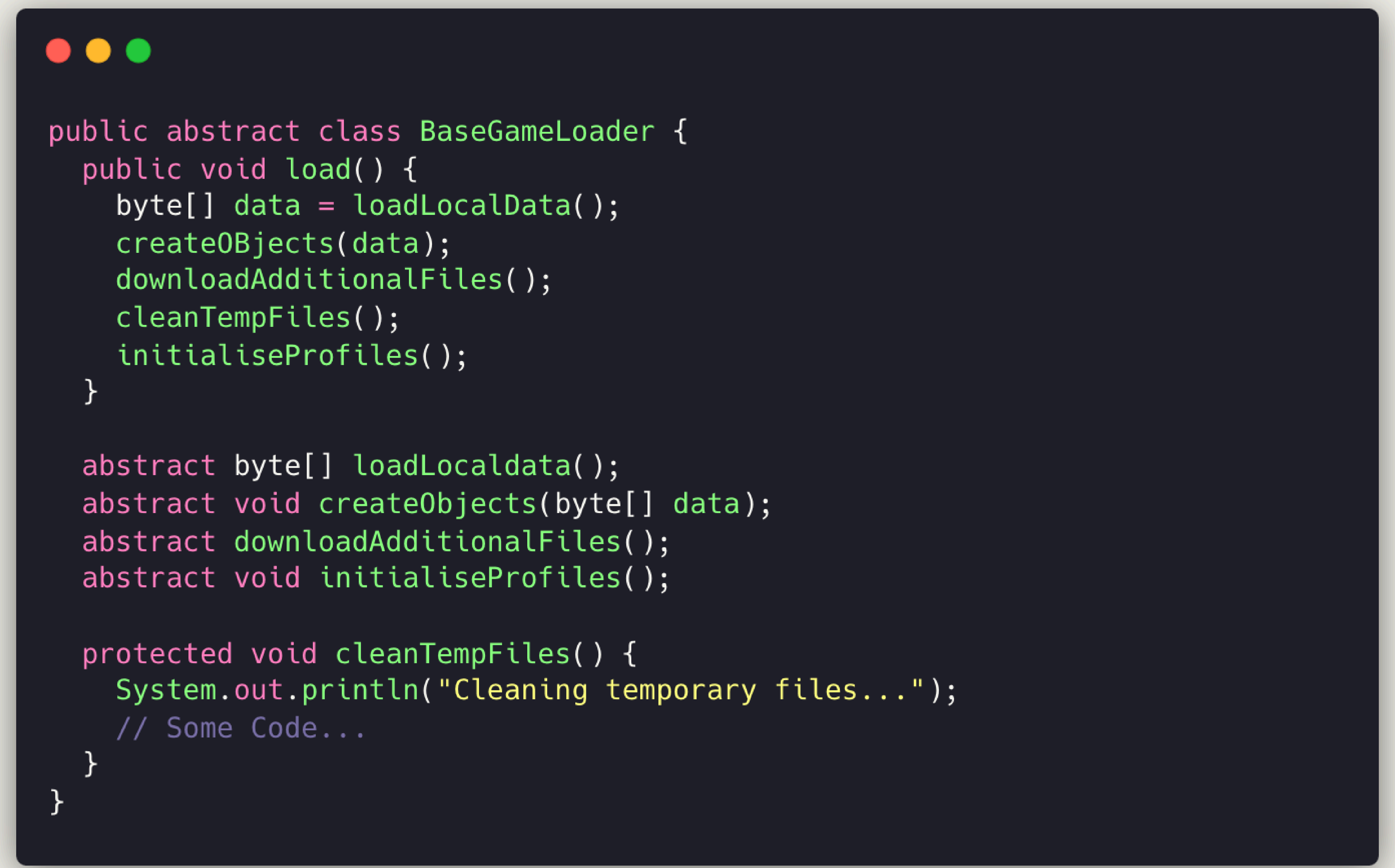


```java
public class ValorantLoader {
    public load() {
        System.out.println("Loading Valorant files....");
        // Some Valorant Code

        System.out.println("Creating needed Valorant objects");
        // Some Valorant Code

        System.out.println("Downloading Valorant sounds and videos...");
        // Some Valorant Code

        System.out.println("Cleaning tempoary files");
        // Some Code

        System.out.println("Loading local Valorant files...");
        // Some Valorant Code
  }
}
```



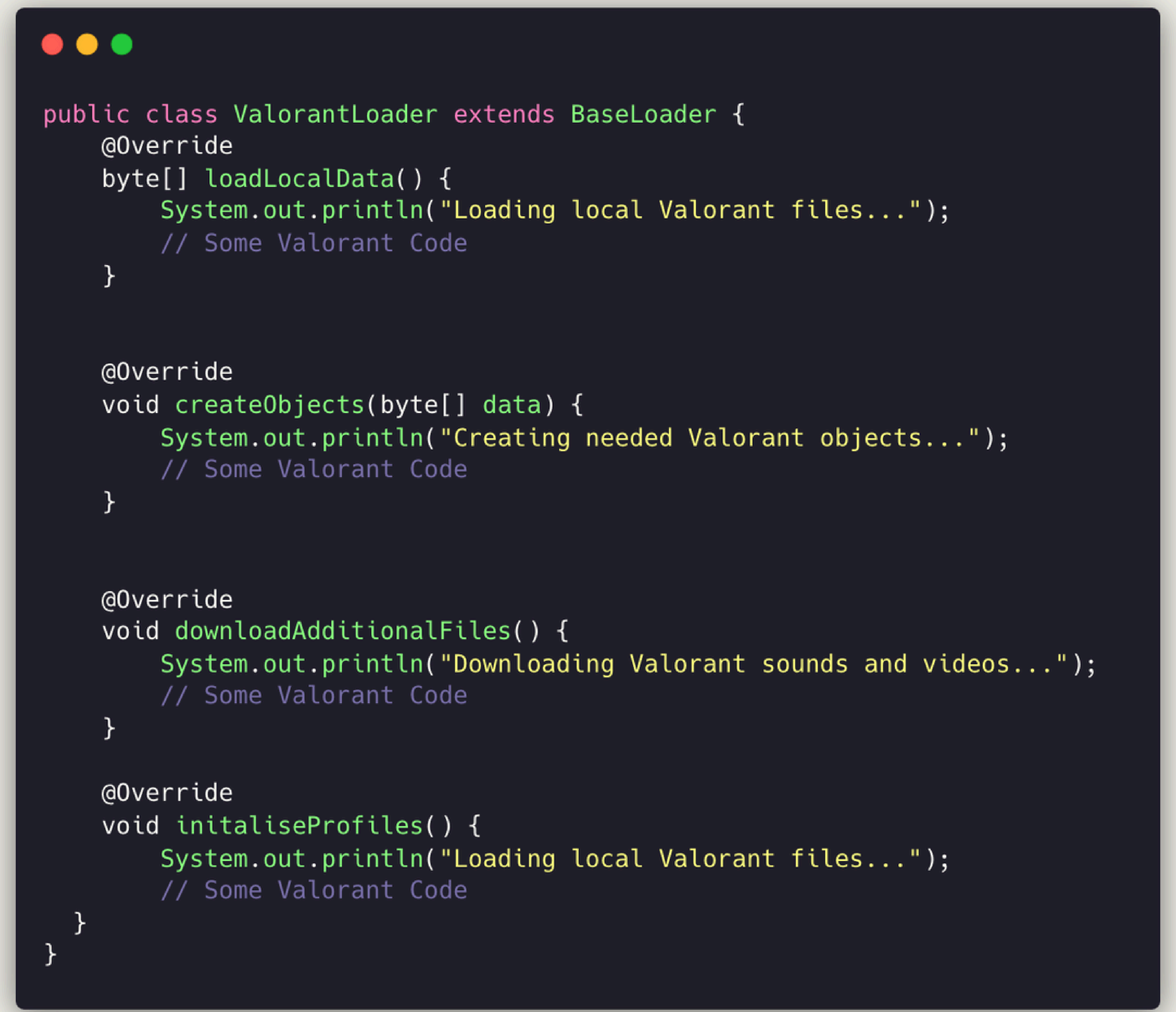
```java
public class PokemonLoader {
    public load() {
        System.out.println("Loading Pokemon files....");
        // Some Pokemon Code

        System.out.println("Creating needed Pokemon objects");
        // Some Pokemon Code

        System.out.println("Downloading Pokemon sounds and videos...");
        // Some Pokemon Code

        System.out.println("Cleaning tempoary files");
        // Some Code

        System.out.println("Loading local Pokemon files...");
        // Some Pokemon Code
  }
}
```

TEMPLATE METHOD

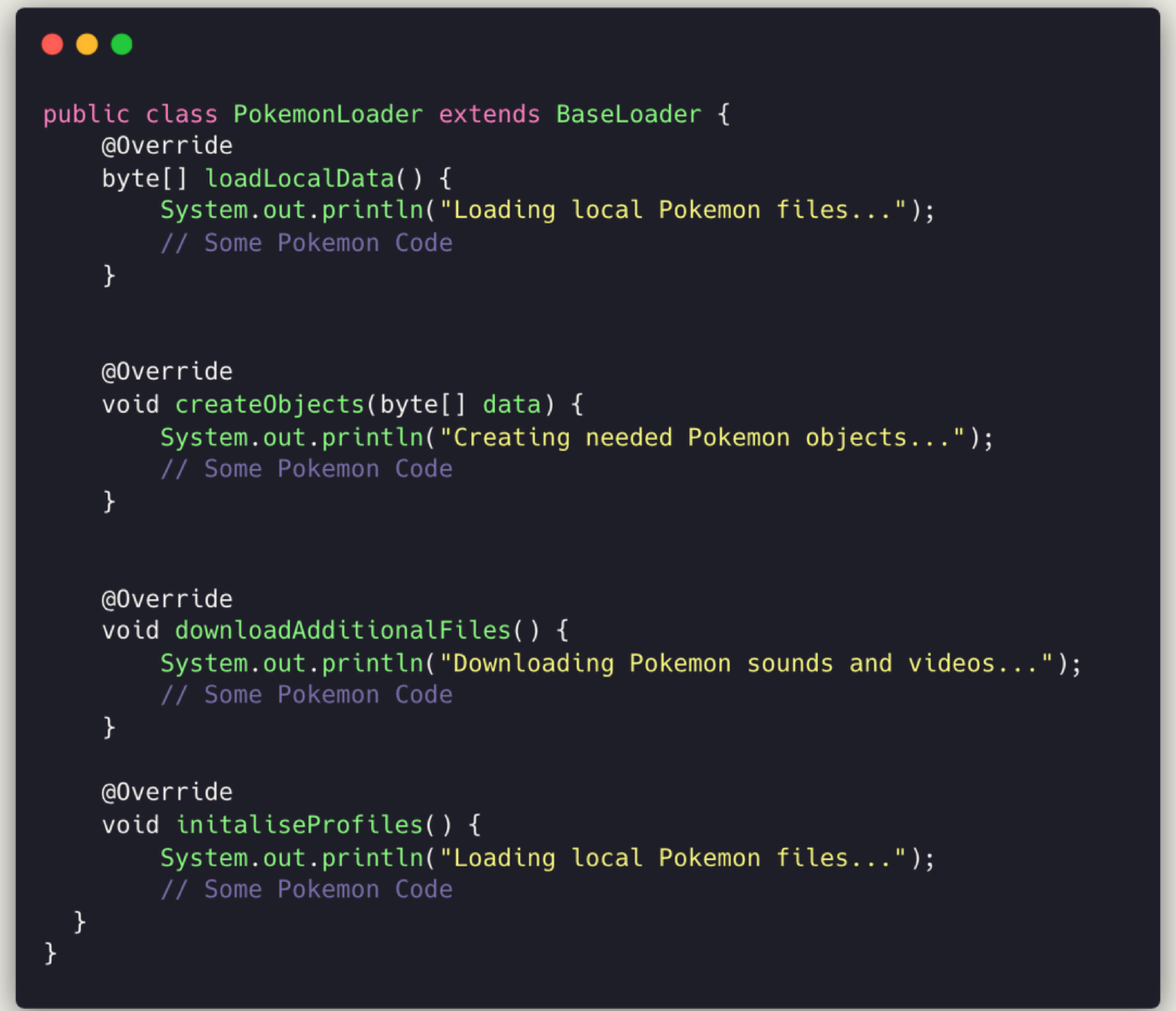


```
public abstract class BaseGameLoader {
  public void load() {
    byte[] data = loadLocalData();
    createOBjects(data);
    downloadAdditionalFiles();
    cleanTempFiles();
    initialiseProfiles();
  }

  abstract byte[] loadLocaldata();
  abstract void createObjects(byte[] data);
  abstract downloadAdditionalFiles();
  abstract void initialiseProfiles();

  protected void cleanTempFiles() {
    System.out.println("Cleaning temporary files...");
    // Some Code...
  }
}
```

TEMPLATE METHOD



```java
public class ValorantLoader extends BaseLoader {
    @Override
    byte[] loadLocalData() {
        System.out.println("Loading local Valorant files...");
        // Some Valorant Code
    }


    @Override
    void createObjects(byte[] data) {
        System.out.println("Creating needed Valorant objects...");
        // Some Valorant Code
    }


    @Override
    void downloadAdditionalFiles() {
        System.out.println("Downloading Valorant sounds and videos...");
        // Some Valorant Code
    }

    @Override
    void initaliseProfiles() {
        System.out.println("Loading local Valorant files...");
        // Some Valorant Code
  }
}
```



```java
public class PokemonLoader extends BaseLoader {
    @Override
    byte[] loadLocalData() {
        System.out.println("Loading local Pokemon files...");
        // Some Pokemon Code
    }


    @Override
    void createObjects(byte[] data) {
        System.out.println("Creating needed Pokemon objects...");
        // Some Pokemon Code
    }


    @Override
    void downloadAdditionalFiles() {
        System.out.println("Downloading Pokemon sounds and videos...");
        // Some Pokemon Code
    }

    @Override
    void initaliseProfiles() {
        System.out.println("Loading local Pokemon files...");
        // Some Pokemon Code
  }
}
```

Code and Design Smells

CODE AND DESIGN SMELLS

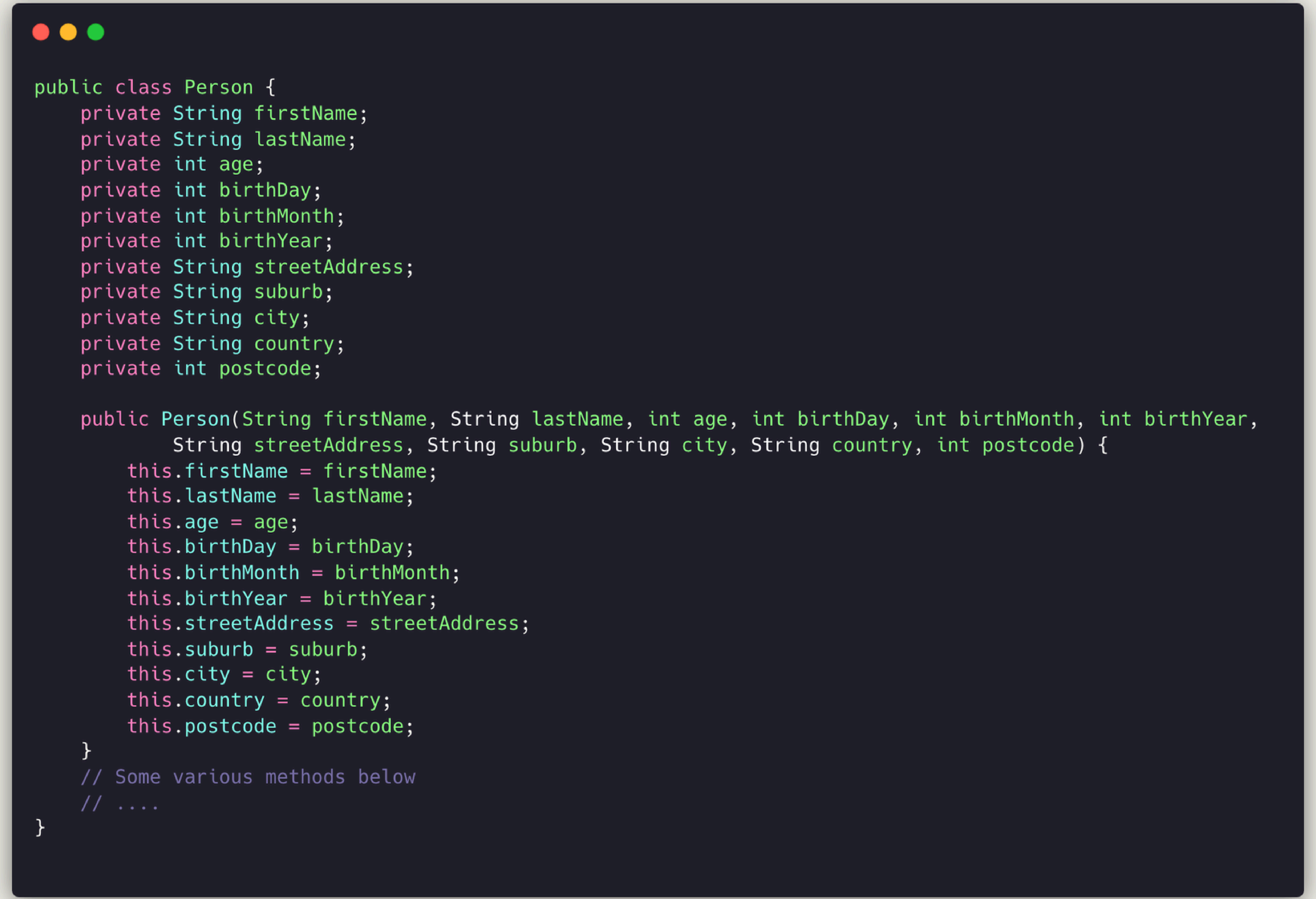
Mark, Bill and Jeff are working on a PetShop application. The PetShop has functionality to feed, clean and exercise different types of animals. Mark notices that each time he adds a new species of animal to his system, he also has to rewrite all the methods in the PetShop so it can take care of the new animal.

CODE AND DESIGN SMELLS

Mark, Bill and Jeff are working on a PetShop application. The PetShop has functionality to feed, clean and exercise different types of animals. Mark notices that each time he adds a new species of animal to his system, he also has to rewrite all the methods in the PetShop so it can take care of the new animal.

Code smell - Divergent change
Design problem - Open Closed Principle, high coupling

CODE AND DESIGN SMELLS

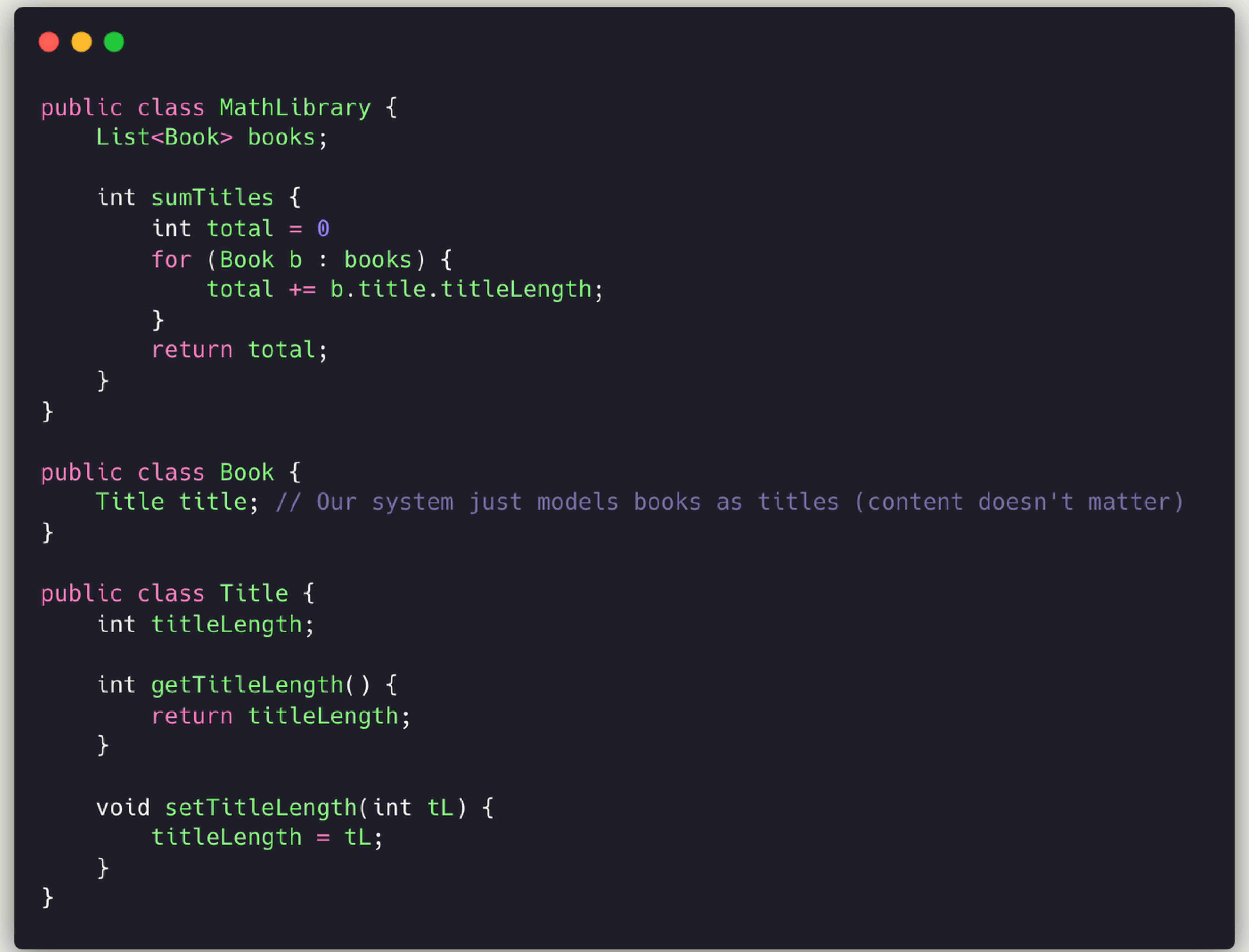


```
public class Person {
    private String firstName;
    private String lastName;
    private int age;
    private int birthDay;
    private int birthMonth;
    private int birthYear;
    private String streetAddress;
    private String suburb;
    private String city;
    private String country;
    private int postcode;

    public Person(String firstName, String lastName, int age, int birthDay, int birthMonth, int birthYear,
            String streetAddress, String suburb, String city, String country, int postcode) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
        this.birthDay = birthDay;
        this.birthMonth = birthMonth;
        this.birthYear = birthYear;
        this.streetAddress = streetAddress;
        this.suburb = suburb;
        this.city = city;
        this.country = country;
        this.postcode = postcode;
    }
    // Some various methods below
    // ....
}
```

CODE AND DESIGN SMELLS

Data clumps, long parameter list
Refactor by making more classes for birthday and address ("Extract Class"/ "Introduce Parameter Object")
Design problem - DRY and KISS

CODE AND DESIGN SMELLS



```java
public class MathLibrary {
    List<Book> books;

    int sumTitles {
        int total = 0
        for (Book b : books) {
            total += b.title.titleLength;
        }
        return total;
    }
}

public class Book {
    Title title; // Our system just models books as titles (content doesn't matter)
}

public class Title {
    int titleLength;

    int getTitleLength() {
        return titleLength;
    }

    void setTitleLength(int tL) {
        titleLength = tL;
    }
}
```

CODE AND DESIGN SMELLS

Inappropriate intimacy (accessing public fields)
Message chains - students might bring up Law of Demeter here
Data classes/Lazy classes Design smell - High coupling, from encapsulation being broken
Fixes - make things private, just delete the classes and represent titles as strings

CODE AND DESIGN SMELLS

1. How do these code smells cause problems when developing code?
   - Reusability, Maintainability, Extensibility
2. Is a code smell always emblematic of a design problem?
   - No - e.g "switch statements" and "comments" are often listed as code smells but are not always actually smells

ITS LAB TIME

YASSS