Interface Segregation

Contents

Overview

In object-oriented programming, the fourth of the **SOLID Principles** is **I** - which stands for **Interface Segregation**.

The *Interface Segregation Principle* states that you should not be forced to depend on interfaces containing methods that you aren't going to use.

Essentially, you should only ever have methods inside an interface together if they *all* need to be implemented *at once*.

Interface Segregation in Action

Spreadsheet Converter

Let's say we have a `Converter.java` class which implements a `SpreadsheetConverter.java` interface:

▶ Converter
▶ SpreadsheetConverter

This does not adhere to the *Interface Segregation Principle*, because the latter two methods have needlessly been included within the `SpreadsheetConverter.java` interface in order to make `Converter.java` work.

Rather than making the interfaces simpler to implement, we instead end up having to code Exception-handling for our unnecessary methods.

Fixing the Spreadsheet Converter

The solution here is relatively simple:

- remove the unnecessary methods (they can always be added back in at a later date)
- split the interface into separate ones which only contain a single function

▶ ExcelToCsvConverter
▶ CsvToExcelConverter
▶ Converter

There's nothing stopping you from implementing *multiple interfaces*, so it's best to *make your interfaces as small as possible* and then *only implement the ones you need*.

By segregating interfaces this way, we greatly increase the readability and maintainability of our program.

Tutorial

There is no tutorial for this module.

Exercises

Bird

A common pitfall in development is the naming of interfaces or abstract classes after real-world things. The problems that arise from this error are twofold:

- the collection of methods defined in one interface increases as you add more functionality to it (which also violates Single Responsibility)
- implementing a larger interface means implementing several methods, some of which may be unnecessary

Consider the following three modules:

- `Bird.java` (interface)
- `Falcon.java` (class, implements `Bird.java`)
- `Dodo.java` (class, implements `Bird.java`)

▶ Bird
▶ Details
▶ Dodo

It might make sense at first to make a Bird interface, but when encountered with things that don't fit the usual specification for a Bird, things get difficult to deal with quickly.

Refactor this code to adhere to the Interface Segregation Principle, using the following modules, to complete this excercise:

- `EggLayingCreature.java`
- `FlyingCreature.java`
- `ExtinctCreature.java`
- `Dodo.java`
- `Falcon.java`

▶ Show solution