

Professional Skills
Agile Fundamentals
Jira
Git
Databases Introduction
Java Beginner
Maven
Testing (Foundation)
Java Intermediate
<div><div></div>Optionals</div>
<div><div></div>JDBC CRUD</div>
<div><div></div>Exceptions</div>
<div><div></div>SOLID Principles</div>
<div><div></div>Single Responsibility</div>
<div><div></div>Open/Closed</div>
<div><div></div>Liskov Substituion</div>
<div><div></div>Interface Segregation</div>
<div><div></div>Dependency Inversion</div>
<div><div></div>Best Practice</div>
<div><div></div>Design Patterns</div>
<div><div></div>Creational Design Patterns</div>
<div><div></div>Structural Design Patterns</div>
<div><div></div>Behavioural Design Patterns</div>
<div><div></div>Collection & Map</div>
<div><div></div>HashSets</div>
<div><div></div>HashMaps</div>
<div><div></div>Enums</div>
<div><div></div>Logging</div>
<div><div></div>Generics</div>
<div><div></div>Lambda Expressions</div>
<div><div></div>Streams</div>
<div><div></div>Complexity</div>
<div><div></div>Input and Output</div>
<div><div></div>Local Type Inference</div>
HTML
CSS

Interface Segregation

Contents

- [Overview](#)
- [Interface Segregation in Action](#)
 - [Spreadsheet Converter](#)
 - [Fixing the Spreadsheet Converter](#)
- [Tutorial](#)
- [Exercises](#)
 - [Bird](#)

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.

Javascript
Spring Boot
Selenium
Sonarqube
Advanced Testing (Theory)
Cucumber
MongoDB
Express
NodeJS
React
Express-Testing
Networking
Security
Cloud Fundamentals
AWS Foundations
AWS Intermediate
Linux
DevOps
Jenkins Introduction
Jenkins Pipeline
Markdown
IDE Cheatsheet

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