

Professional Skills
Agile Fundamentals
Jira
Git
Databases Introduction
Java Beginner
○ What is Java?
○ Installation
○ Hello World Example
○ Data Types
○ Packages
○ Naming Conventions Cheat Sheet
○ Flow of Control
○ Class Members
○ Operators
○ Conditionals
○ Iteration
○ Arrays
○ ArrayList
○ Enhanced For Loops
○ String Manipulation
○ Class Constructors
○ Access Modifiers
○ Installing Java & Maven To PATH
○ Object-Oriented Programming Principles
○ Encapsulation
○ Inheritance
○ Polymorphism
○ Abstraction
○ Interfaces
○ Type Casting
○ Static
○ Final
○ Garbage Collection
○ Input With Scanner
○ Pass by Value/Reference
○ JUnit

Encapsulation

Encapsulation is an *object-oriented programming principle*.

Contents

- [Overview](#)
- [Encapsulation in action](#)
- [Read and Write-Only modules](#)
- [Tutorial](#)
- [Exercises](#)
 - [Refactor your old stuff!](#)

Overview

Of the four *object-oriented programming principles*, **encapsulation** states that data (*variables*) should be bundled together with the code that operates on that data (*methods*), rather than allowing *direct* access.

Generally, this is done by *restricting access* to the *variables* within a class, and only allowing access through the *methods* of their current class.

To achieve encapsulation, then:

- make the class variables *private*
- provide *public* accessors to these variables: **getters** and **setters**

Encapsulation in action

Let's say we've got a class with directly-accessible variables in it:

```
public class ExtremelyImportantStuff {
    public int passportNumber = 1355417;
    public double bankBalance = -537.86;
    public String memorableWord = "pareidolia";
}
```

The issue with having a class like this is that you can change the value of each variable directly, which could cause problems later:

- we have less control over the variables and the methods which utilise them
- classes which do not need to use these variables will still be able to 'see' them
- every class has full access to these variables, including *read* and *write* functionality

Let's improve this:

<div><div></div><div>Test Driven Development</div></div> <div><div></div><div>UML Basics</div></div> <div><div></div><div>JavaDoc</div></div> <div><div></div><div>Peer Programming</div></div> <div><div></div><div>Code Reviews</div></div>
Maven
Testing (Foundation)
Java Intermediate
HTML
CSS
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

```
public class ExtremelyImportantStuff {
    private int passportNumber = 1355417;
    private double bankBalance = -537.86;
    private String memorableWord = "pareidolia";

    public int getPassportNumber(){
        return passportNumber;
    }

    public void setPassportNumber(int newPassportNumber){
        this.passportNumber = newPassportNumber;
    }

    //etc.
}
```

The `get` method returns the value of the variable name.

The `set` method takes in a parameter and assigns it to the `name` variable - we use `this` to refer to the current object.

Read and Write-Only modules

We can make a module *read-only* by just using `get` methods:

```
public class Delegate {
    private String company = "QA";

    public String getCompany(){
        return company;
    }
}
```

We can make a module *write-only* by just using `set` methods:

```
public class Trainer {
    private String cohort;

    public void setCohort(String newCohort){
        this.cohort = newCohort;
    }
}
```

Tutorial

There is no tutorial for this module.

Exercises

Refactor your old stuff!

Go back through some of your other code and *encapsulate* your variables.

You may find that changing your code's functionality will be the hardest part.

