INTERNET OF THINGS CLUB

What is IoT Club?

- Collection of tech enthusiasts
- Learning/sharing hub for all
- Meme and ice cream appreciation center
- Only $2/year membership

First Event: CONNECT

- Yasuko Hiraoka Myer Room
- Lvl1, Sidney Myer Asia Center
- Friday (8 Mar) 4 – 6 p.m.

Link:
bit.ly/2BXNBGQ

/iotclub.unimelb

@iotclub.unimelb

iotclub.unimelb@gmail.com

Assess Yourself

Who is Matt's favourite superhero?

SWEN20003
Object Oriented Software Development

Classes and Objects

Semester 1, 2019

How This Semester Works

- Rather than teaching you Java foundations first, we're diving straight into Object Oriented Programming
- We assume you have experience in **at least one** programming language
- Grok worksheets teach the Java foundations **first**
- Most of the code should be intuitive, or at least make some sense
- Some things won't "click" for a few weeks, when we add the finer details
- It's our first time trying this, so be patient, and give us feedback!

Grok

- Worksheets 1-7 now available
- Worksheets teach content to *complement* and *reinforce* the lectures
- Go at your own pace
- Not assessed, does not contribute to your marks
- Don't just answer the questions; it is assumed you will have **read the slides** as well
- Meme-tastic

Let's take a quick look at the worksheets.


THE ROAD SO FAR

The Road So Far

Lectures

- Welcome to SWEN20003

Tutorials

- TBD

Lecture Objectives

After this lecture you will be able to:

- Explain the difference between a *class* and an *object*
- Create classes, and give them *properties* and *behaviours*
- Implement the core components of a basic class
- Read a *specification*, and turn it into a series of well-defined classes
- Design basic Object Oriented systems

Motivating Example

Throughout this lecture we'll be referring to the following specification:

Develop a system (a set of classes) that can "replicate" the behaviour of IMDB. It should be able to store the details of actors, movies, their associations with each other, and their ratings.

How would you develop this, right now? What additional information do you need?

Hello World

```
public class Main {

    public static void main(String args[]) {

        System.out.println("Hello World");

    }

}
```

- Every Java programmer's first class
- Explored in detail in Grok worksheets
- For now, we only care about the *class* keyword

Classes

- A "generalization" of a real world (or "problem world") entity
  - A physical real world thing, like a student or book
  - An abstract real world thing, like a university subject
  - An even more abstract thing like a list or a string (data)
- Represents a template for things that have common properties
- Contains *attributes* and *methods*
- Defines a new **data type**

Keyword

Class: Fundamental unit of abstraction in *Object Oriented Programming*. Represents an "entity" that is part of a problem.

Objects

- Are an *instance* of a class
- Contain **state**, or dynamic information
- "**X** is of type A", "**X** is an object of the class A", and "**X** is an instance of the class A" are all equivalent

Keyword

Object: A specific, concrete example of a class

Keyword

Instance: An object that exists in your code

IMDB Clone

What classes can we use for our example problem?

Fundamental:

- `Actor`
- `Movie`
- `Database` ("main")

Additional:

- `Rating`
- `Comment`

Defining a Class

```
<privacy> class <ClassName> {
    <variable declarations>

    <method declarations>
}
```

Classes and Objects

Keyword

Class (Static) Variable: A **property** or **attribute** that is common to/shared by all instances of a *class*

Keyword

Instance Variable: A **property** or **attribute** that is unique to each *instance (object)* of a class

Classes and Objects

Keyword

Class (Static) Method: An **action** that can be performed by a *class*, or a **message** that can be sent to it

Keyword

Instance Method: An **action** that can be performed by an *object*, or a **message** that can be sent to it

Class (Static) Variables

```
public class <ClassName> {
    public static <type> varName = <value>;
}
```

- A **property** or **attribute** that is common to all instances of a class
- Can be a constant or variable
- One copy **per class**

```
public class Movie {
    public static final int MAX_RATING = 5;
}
```

Instance Variables

```
public class <ClassName> {
    public <type> varName = <value>;
}
```

- A **property** or **attribute** that is unique to each instance of a class
- Can be a constant or variable
- One copy **per object**

```
public class Movie {
    public String title;
}
```

Class (Static) Methods

```
public class <ClassName> {
    public static <return_type> methodName(<arguments>) {

    }
}
```

- Defines an **action** that can be performed by a **class**, or a **message** that can be sent to it
- Does not refer to any **instance variables**

```
public class Movie {
    public static String getDefaultBlurb() {
        return "Better than Batman vs. Superman
                    but that's not hard.";
    }
}
```

Instance Methods

```
public class <ClassName> {
    public <return_type> methodName(<arguments>) {
    }
}
```

- Defines an **action** that can be performed by an **object**, or a **message** that can be sent to it
- Actions that are only useful with an object's **data**

```
public class Actor {

    public String firstName, lastName;

    public String getFullName() {
        return String.format("%s, %s", lastName, firstName);
    }
}
```

Pitfall: Instance vs. Static Methods

If a method doesn't use any instance variables, it should be static.

Useful Rule of Thumb^{TM}: make all methods static, and then remove static only if the method uses an instance variable.

Do not do the reverse: make methods static, and then make *variables* static when you get an error.

IMDB Clone

What attributes and methods can we add to our classes?

`Actor`:

- Attributes
  - name
  - age
  - country
  - appearances
  - rating
- Methods
  - print
  - appearsIn

IMDB Clone

What attributes and methods can we add to our classes?

`Movie`:

- Attributes
  - title
  - earnings
  - actors
  - rating
- Methods
  - print
  - hasActor

IMDB Clone

What attributes and methods can we add to our classes?

`Database`:

- Attributes
  - actors
  - movies
- Methods
  - main
  - createActors
  - createMovies
  - search

Null

Keyword

null: The Java keyword for "no object here". Null objects can't be "accessed" to get variables or methods, or used in any way.

Instantiating a Class

Objects are **null** until they are *instantiated*.

Keyword

Instantiate: To create an object of a class

```
// Instantiate an Actor object
Actor robertDowneyJr = new Actor();
```

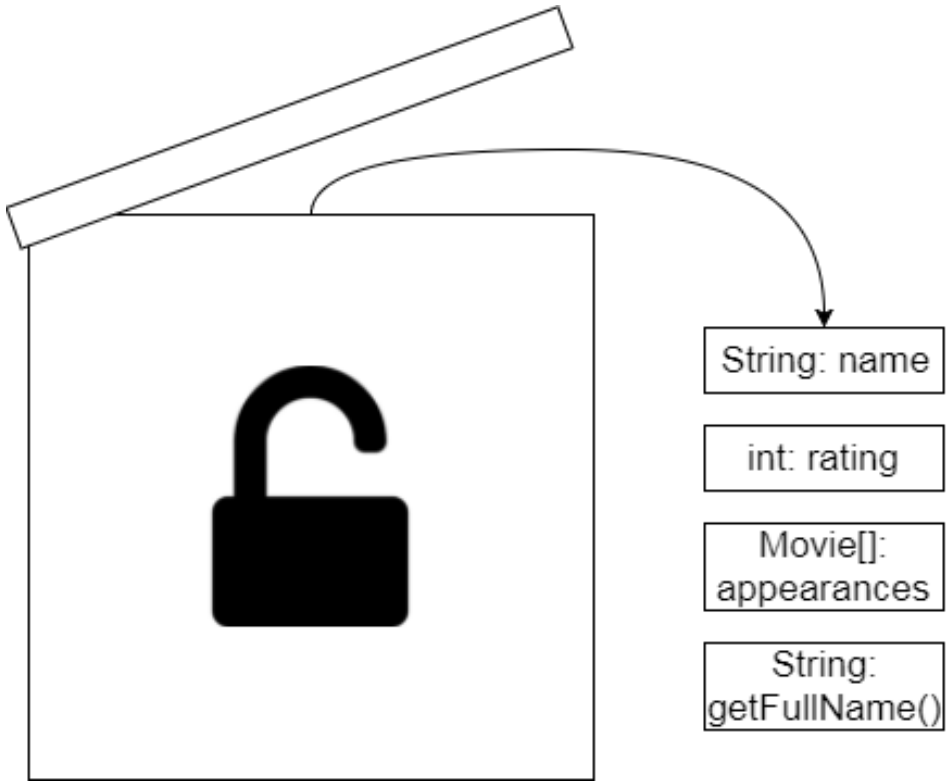
Keyword

new: Directs the JVM to allocate memory for an object, or *instantiate* it

Accessing/Modifying Members

- Accessing the *members* of a class (variables/methods) uses "dot notation"
- The dot operator exposes all **public** members*

```
robertDowneyJr.name = "Robert";
```



Initialising Objects

```
Actor robertDowneyJr = new Actor();

robertDowneyJr.firstName = "Robert";
robertDowneyJr.lastName = "Downey";
robertDowneyJr.rating = 5;
```

- What if we have 100 attributes to initialise?
- What if we have 100 objects to initialise?
- We need a better... **method**
  (this will be funny soon)

Constructors

How does this actually work?

```
Actor robertDowneyJr = new Actor();
```

- The right hand side *invokes* (or calls) a class' *constructor*
- Constructors are **methods**
- Constructors are used to initialize objects
- Constructors have the same name as the class
- Constructors cannot return values
- A class can have one **or more** constructors (called overloading; we'll cover that in later lectures)

Constructors

Keyword

Constructor: A method used to **create** and **initialise** an object.

Constructors

```
public <ClassName>(<arguments>) {
    <block of code to execute>
}
```

Default `Actor` constructor:

```
public Actor() {
    firstName = "";
    lastName = "";
    rating = 0;
}
```

More useful `Actor` constructor:

```
public Actor(String newFirstName, String newLastName, int newRating) {
    firstName = newFirstName;
    lastName = newLastName;
    rating = newRating;
}
```

Pitfall: Constructors

```
public Actor(String firstName, String lastName, int rating) {
    firstName = firstName;
    lastName = lastName;
    rating = rating;
}
```

How does the code differentiate the two variables?

This

Keyword

this: A **reference** to the **calling object**, the object that owns/is executing the method.

```
public Actor(String firstName, String lastName, int rating) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.rating = rating;
}
```

Standard Methods - *equals*

```
public boolean equals(<type> var) {
    return <boolean expression>;
}
```

- It is useful to be able to compare if two objects are **equal**
- How they are equal is up to you; use **one or more** properties of the objects
- This is version one; we'll "improve" it as we go

```
public boolean equals(Fruit otherFruit) {
    return this.colour.equals(otherFruit.colour);
}
```

IMDB Clone

What determines if two `Actor` objects are equal? Two `Movie` objects?

```
public boolean equals(Actor otherActor) {
    return this.name.equals(otherActor.name) &&
        this.lastName.equals(otherActor.lastName);
}
```

```
public boolean equals(Movie otherMovie) {
    return this.title.equals(otherMovie.title) &&
        this.rating == otherMovie.rating;
}
```

Standard Methods - *toString*

```
public String toString() {
    return <String>;
}
```

- Returns the `String` **representation** of an object
- Automatically called when the object acts like a `String`
- E.g. **printing** an object

```
public String toString() {
    return String.format("%ss are %s!", this.name, this.colour);
}
```

```
System.out.println(new Fruit("apple", "red"));
```

```
apples are red!
```

IMDB Clone

What is the `String` representation of an `Actor`? A `Movie`?

`Actor`

```
public String toString() {
    return String.format("%s, %s", this.lastName,
                                    this.firstName);
}
```

`Movie`

```
public String toString() {
    return String.format("%s is rated %d/%d", this.title,
        this.rating, MAX_RATING);
}
```

Garbage Collection

Objects that have no references pointing to them are marked for cleanup automatically.

Throughout your program, "dead" objects are periodically deleted by the *garbage collector*.

Finalize

```
public void finalize() {
    <block of code to execute>
}
```

- Method called when objects are deleted
- Useful for cleaning up, record keeping, etc.

```
public void finalize() {
    numActors--;
}
```

Assess Yourself

Design classes, including attributes and methods, for the following scenario:

An up and coming entrepreneur wants your advice on their latest venture: a system that allows "decision makers" like local governments to import, view, and visualise data.

The system should be able to read CSV and XLS documents, should be able to present the data in a table with appropriate filters, and also visualise the data with graphs, charts, etc.

Assess Yourself

Class: `Data`

- Attributes
  - values
  - nRows
  - nCols
- Methods
  - readData
  - add
  - remove

Assess Yourself

Class: `Display`

- Attributes
  - values
  - nRows
  - nCols
- Methods
  - filter
  - delete

Assess Yourself

Class: `Chart`

- Attributes
  - values
  - width
  - height
  - colours
- Methods
  - build
  - update
  - changeColours

Assess Yourself

Design classes, including attributes and methods, for the following scenario:

As a software engineer at RobotOverlords Inc., you've been tasked with designing the software systems for Murder Bot V3, your flagship fly-swatting robot.

Murder Bot has a number of sensors (battery, GPS, motors), actuators (arms, legs), and controls (electric swatter, flamethrower).

Assess Yourself

Class: `Sensor`

- Attributes
  - value
  - name
- Methods
  - measure
  - calibrate

Assess Yourself

Class: `Actuator`

- Attributes
  - position
- Methods
  - calibrate
  - measurePosition

Assess Yourself

Class: `Control`

- Attributes
  - isOn
  - batteryLevel
- Methods
  - activate
  - measureBattery

Assess Yourself

Design classes, including attributes and methods, for the following scenario:

You have been asked to develop the user interface (and associated backend) for a shopping centre's in-store map system.

The system should allows users to search for stores, find directions to stores, and list stores and their details.

Assess Yourself

Class: `Interface/UserInterface`

- Attributes
  - shops
  - map
- Methods
  - search
  - getDirections
  - listStores

Assess Yourself

Class: `Shop`

- Attributes
  - type
  - contactName
  - contactNumber
  - location
- Methods
  - openingHours

Assess Yourself

Class: `Map`

- Attributes
  - level
  - shops
  - source
  - destination
- Methods
  - changeLevel
  - displayRoute
  - overlayText

Assess Yourself

Class: `CustomerInterface`

- Attributes
  - shops
  - map
  - directory
- Methods
  - search
  - getDirections
  - listStores

Metrics

A world class chef is working with you to develop a robotic assistant, and they've suggested you build a simulated kitchen to test it out.

The robot needs to be able to:

- Use normal human tools and utensils (oven, fork)
- Open things, like cupboards and containers
- Prepare ingredients in various ways
- Be operated by a human (for safety reasons)

How would you develop this system? What classes would you use? What methods and attributes would they have? What interface does your program have between the user and the robot?