Java Programming 2
Objects

Mary Ellen Foster

MaryEllen.Foster@glasgow.ac.uk

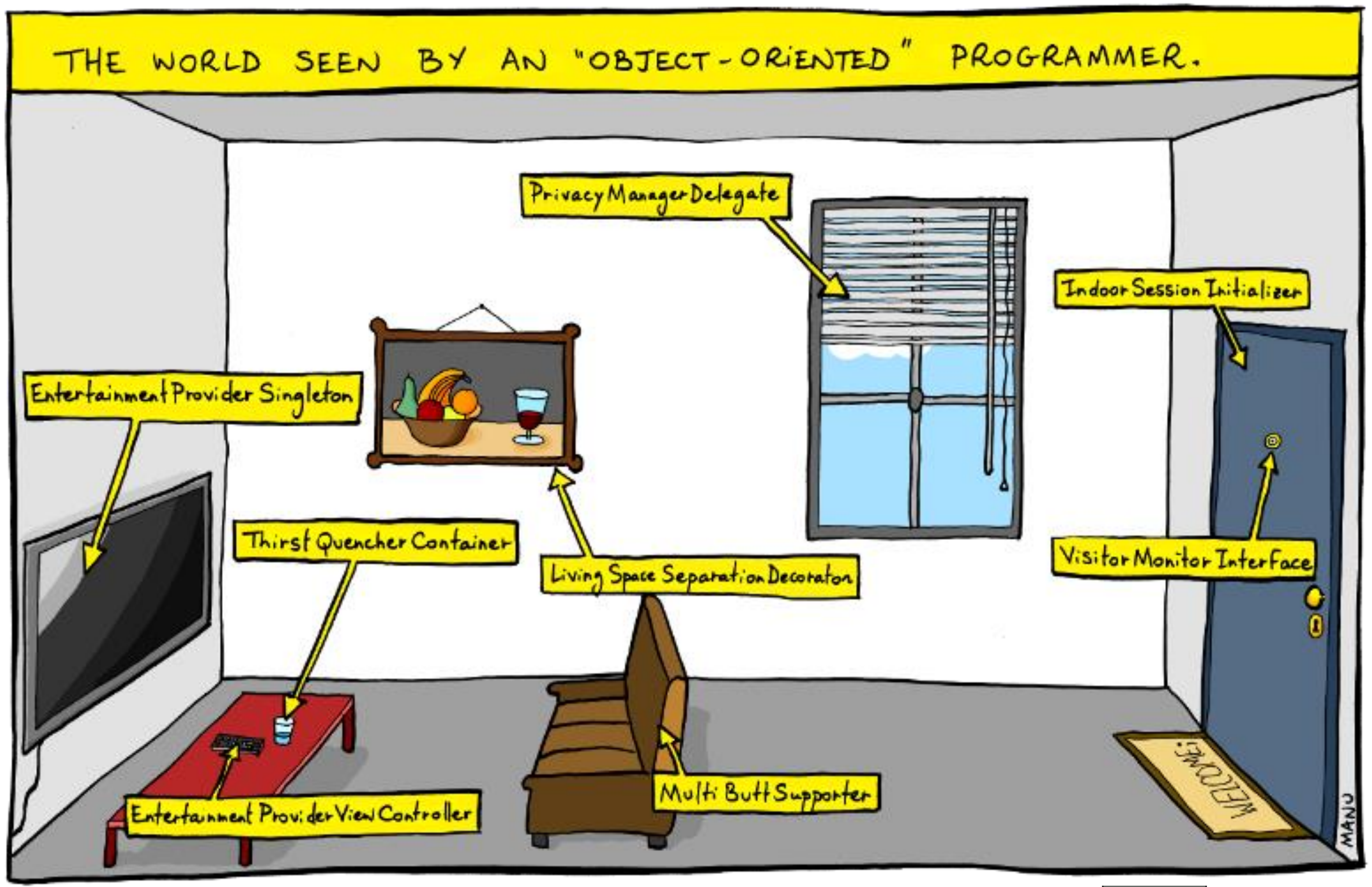
Semester 1 2020/2021

Outline

Objects and classes

Class members

Object instantiation


THE WORLD SEEN BY AN "OBJECT-ORIENTED" PROGRAMMER.
Privacy Manager Delegate
Indoor Session Initializer
Entertainment Provider Singleton
Thirst Quencher Container
Living Space Separation Decorator
Visitor Monitor Interface
Multi Butt Supporter
Entertainment Provider View Controller
WELCOME!
MANU

Comic by Manu Comet -- http://www.bonkersworld.net/object-world/

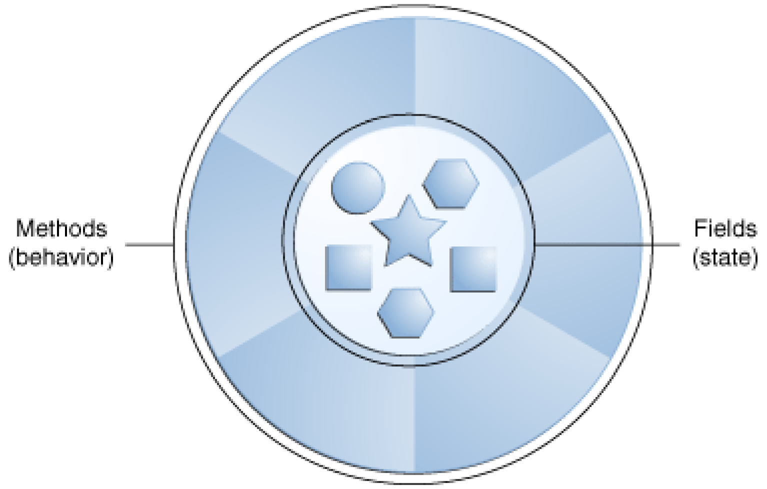
Objects

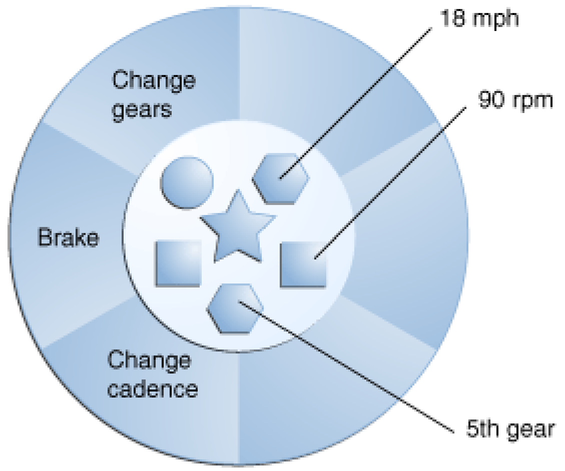
Characteristics of **objects** (real-world or software)

- State
- Behaviour

Why program with objects?

- Modularity
- Information-hiding
- Code re-use
- Pluggability and debugging ease





Java Tutorial "What is an Object?"

Classes vs. objects

Classes are **types**

Objects are **instances of types**

An object is an **instance** of a general **class** of objects

- In other words, a class is a **blueprint** from which individual objects are created

Example 1: `boolean` primitive type has instance values **true** and **false**

Example 2: imagine a `Dog` class

- Instances: toto, lassie, brianGriffin, scoobyDoo, …

Abstraction and encapsulation

Abstraction:

A class is an abstract description of a set of real-world entities

Encapsulation:

Each *instance* of a class has data and behaviour associated with it

Class declaration

Use the `class` keyword

Give the class a name (use *CamelCase*)

Specify class body inside curly brackets

- Fields (properties)
- Methods (behaviours)

(Optional other things: access modifier(s), superclass, interface(s) – we will address these later in the course)

Example class: bank account

```
class BankAccount {

    int balance;
   String name;

    void deposit(int value) { this.balance += value; }

    void withdraw(int value) { this.balance -= value; }

}
```

Fields

Methods

Class members: fields and methods

Data fields:

- Store state that represent some attributes of the object
- *For `Dog` class: name, breed, size, age, …*

Methods:

- Represent behaviour that processes and transforms the object state
- *For `Dog` class: eat(), sleep(), goForWalk(), …*

Special method: `public static void main (String[] args)`

- If a class has a `main` method, then you can run it directly (Eclipse: “Run as Java application”)

Details of Java methods (revisited)

A method declaration has six components (in order):

1. Access modifier(s) (zero or more – details later)
2. Return type (`void` if it does not return a value)
3. Method name (conventionally beginning with a verb)
4. Parameter list in parentheses – comma delimited list of input parameters, preceded by data type, enclosed in parens. No parameters – empty parens.
5. An exception list (more on this in a few lectures)
6. The method body, enclosed in braces { }

$\left.\begin{matrix} 3 \\ 4 \end{matrix}\right\}$ *Method signature*

```
public double calculateAnswer(double wingSpan, int numberOfEngines,
                              double length, double grossTons) {
    //do the calculation here
}
```

Fields in Java

Three types of variables:

- Local variables (declared in a method)
- Method parameters (in a method header)
- Member variables in a class – also known as **fields**

Field declarations look the same as local variable declarations, but occur **outside any methods**

An instance variable is accessible in **all methods of a class**

Bank account class revisited

```
class BankAccount {

   int balance;
   String name;

   void deposit(int value) { this.balance += value; }

   void withdraw(int value) { this.balance -= value; }

   BankAccount(String name, int initialAmount) {

      this.name = name;
      this.balance = initialAmount;

   }
}
```

Fields

Methods

Constructor

```
BankAccount(String name, int
initialAmount)
{
    this.name = name;
    this.balance = initialAmount;
}
```

Constructors

Looks like a method with the **same name as the class**

No return type specified (not even `void`)

Sets up initial values for the data fields to initialise object state
- Using `this` keyword to refer to current object being created

Use `new` keyword to create a new object:
```
BankAccount b = new BankAccount (“Mary”, 0);
```

If no constructor is specified, a default *no-args* constructor is automatically created
- No arguments
- Sets all fields to their default values (usually 0 or `null`)

Calling a method

First, create an instance of the class in question (with a constructor)

```
BankAccount myAccount = new BankAccount (“Mary”, 0);
```

Then, call the method on that instance like this: *variable.method(params)*

```
myAccount.deposit(500);
myAccount.withdraw(250);
```