Object Oriented Programming

Programación II
Facultad de Ingeniería
Universidad Austral

Loose Terminology (again!)

State

Fields

Instance Variables

Properties

Variables

Members

Behaviour

Functions

Methods

Procedures

Defining a Class

- It is a user-defined blueprint or prototype from which objects are created.
  - For example, Student is a class while a particular student named Ravi is an object.
- A class in Java could have a set of objects that share common characteristics / behavior and properties / attributes.

Defining a Class

- Properties of Java Classes
  - Class is not a real-world entity. It is just a template or blueprint or prototype from which objects are created.
  - Class does not occupy memory.
  - Class is a group of variables of different data types and a group of methods.
  - A Class in Java can contain:
    - Data members: called fields, attributes, member variables.
    - Methods.
    - Constructors.
    - Nested Classes.
    - Interfaces.

Anatomy of an OOP Program (Java)

Class name

Access modifier

```
public class MyFancyClass {

        public int someNumber;
        public String someText;

        public void someMethod() {

        }

        public static void main(String[] args) {
                MyFancyClass c = new
                        MyFancyClass();
        }

}
```

Class state (properties that an object has such as colour or size)

Class behaviour (actions an object can do)

'Magic' start point for the program (named main by convention)

Create a reference to a MyFancyClass object and call it c

Create an object of type MyFancyClass in memory

Constructors

```
MyObject m = new MyObject();
```

- **You will have noticed that the RHS (right hand side) looks rather like a function call, and that's exactly what it is.**
- **It's a method that gets called when the object is constructed, and it goes by the name of a constructor (it's not rocket science). It maps to the datatype constructors you saw in ML.**
- **We use constructors to initialise the state of the class in a convenient way:**
  - A constructor has the same name as the class
  - A constructor has no return type

Defining a Class without constructors

```
public class Vector3D {
    float x;
    float y;
    float z;
    void add(float vx, float vy, float vz) {
        x=x+vx;
        y=y+vy;
        z=z+vz;
    }
}
```

Constructors

```
public class Vector3D {
        float x;
        float y;
        float z;
        Vector3D(float xi, float yi, float zi) {
                x=xi;
                y=yi;
                z=zi;
        }
    // ...
}
Vector3D v = new Vector3D(1.f,0.f,2.f);
```

Overloaded Constructors

```
public class Vector3D {
       float x;
       float y;
       float z;
       Vector3D(float xi, float yi, float zi) {
            x=xi;
            y=yi;
            z=zi;
       }
       Vector3D() {
            x=0.f;
            y=0.f;
            z=0.f;
}
Vector3D v = new Vector3D(1.f,0.f,2.f);
Vector3D v2 = new Vector3D();
```

Default Constructor

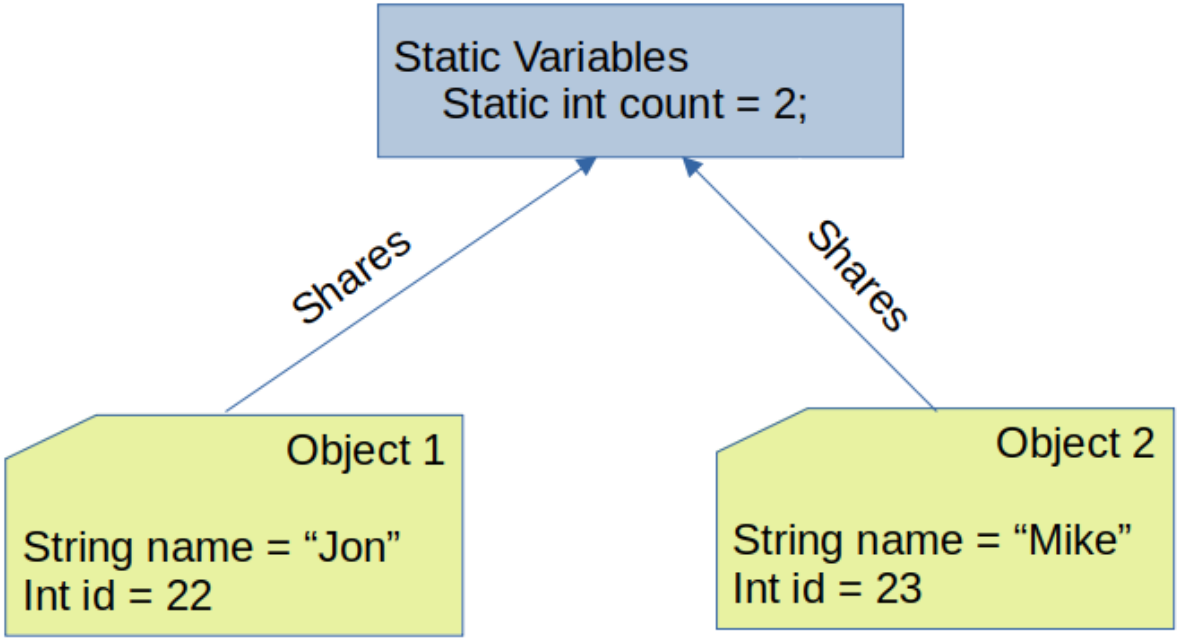
```
public class Vector3D {
            float x;
            float y;
            float z;
}
Vector3D v = new Vector3D();
```

- If you don’t initialise a field it gets set to the ‘zero’ value for that type (don’t do this!)
- If you provide any constructor then the default will not be generated

- No constructor provided
- So blank one generated with no arguments

The Anatomy of the *static* Keyword

- In the Java programming language, the keyword *static* means that the particular member belongs to a type itself, rather than to an instance of that type.
- This means we'll create only one instance of that static member that's shared across all instances of the class.



Class-Level Data and Functionality I

- A **static** field is created only once in the program's execution, despite being declared as part of a class

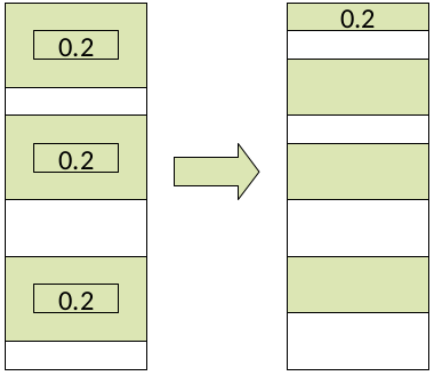
```
public class ShopItem {
  float mVATRate;
  static float sVATRate;
  ....
}
```

One of these created <u>every</u> time a new ShopItem is instantiated. Nothing keeps them all in sync.

<u>Only</u> one of these created <u>ever</u>. Every ShopItem object references it.

static => associated with the class
instance => associated with the object

Class-Level Data and Functionality II



instance field
(one per object)

static field
(one per class)

- Shared between instances
- Space efficient

static fields are good for constants. otherwise use with care.

- Also static methods:

```
public class Whatever {
  public static void main(String[] args) {
    ...
  }
}
```

Why use Static Methods?

- **Easier to debug (only depends on static state)**
- **Self documenting**
- **Groups related methods in a Class without requiring an object**

```
public class Math {
  public float sqrt(float x) {...}
  public double sin(float x) {...}
  public double cos(float x) {...}
}

...
Math mathobject = new Math();
mathobject.sqrt(9.0);
...
```

vs

```
public class Math {
  public static float sqrt(float x) {...}
  public static float sin(float x) {...}
  public static float cos(float x) {...}
}

...
Math.sqrt(9.0);
...
```

Questions?