

# **Why Use Objects and Classes?**

**Simon C. Nash**

**Chief Technical Officer, Java Technology  
IBM Hursley Laboratory, England**

**[nash@hursley.ibm.com](mailto:nash@hursley.ibm.com)**

# Agenda

- What are objects and classes?
- Object-oriented programming
  - ▶ with examples, "do"s and "don't"s
- Why use objects?
- Why use classes?

# What is an Object?

- Executable entity combining code and data
- Code is organised into methods
- Data is accessed via the methods
- Some languages also allow public data

# What is a Class?

- Template or factory for creating objects
  - ▶ called "instances" of the class
- Has definitions of methods and data
- Supports inheritance from other classes
  - ▶ hence "class hierarchy"
- In some languages, classes are objects
  - ▶ with methods and data
  - ▶ hence "metaclass" (the class of a class)

# Languages with Objects or Classes

- Smalltalk
- C++
- Java
- C#
- CLOS, Eiffel, Object Pascal, etc., etc.
- Just about every new language invented these days

# (1) Example Class and Object

```
public class Rectangle {  
    public Rectangle(int width, int height) {  
        this.width = width; this.height = height;  
    }  
    public int width() { return width; }  
    public int height() { return height; }  
    public int area() { return width * height; }  
    private int width;  
    private int height;  
}
```

```
myRectangle = new Rectangle(100, 50);  
System.out.println("Area is "+myRectangle.area());
```

## (2) Without Classes and Objects...

```
public int area(int width, int height) {  
    return width * height;  
}
```

```
myRectWidth = 100;  
myRectHeight = 50;  
System.out.println("Area is "+  
    area(myRectWidth, myRectHeight));
```

## (3) Or Possibly...

```
public class Rectangle {
    public int width;
    public int height;
}
public int area(Rectangle rect) {
    return rect.width * rect.height;
}
myRectangle = new Rectangle();
myRectangle.width = 100;
myRectangle.height = 50;
System.out.println("Area is "+area(myRectangle));
```

# Problems with (2)

- OK (just) for things with two data members
- Unmanageable for more complex things
  - ▶ e.g., rectangle with width, height, text, font, line style, text colour, background colour, ...
- Also error prone
  - ▶ e.g., bank account with name, balance, PIN
- Need to be able to combine lower-level data items into a higher-level unit of data

# What About (3)?

- Is "Rectangle" an object?
- If not, why not?
- Object-based:
  - ▶ Encapsulation
- Object-oriented:
  - ▶ Encapsulation, inheritance, dynamic binding, polymorphism

# Encapsulation

- Separation of interface from implementation
- Public methods, private data
- Can reimplement internals without changing the external interface
  - ▶ so client code is not affected
- Provides modularity: changes in one place don't ripple through the code

# Encapsulation Example

```
public class Colour { // not encapsulated
    public int red;
    public int green;
    public int blue;
}
```

```
public class Colour { // encapsulated
    public Colour(int red, int green, int blue)
        { .... }
    public int red() { .... }
    public int green() { .... }
    public int blue() { .... }
    private int hue;
    private int saturation;
    private int brightness;
}
```

# Inheritance

- Arrange types into a hierarchy
- Properties of base types (superclasses) apply to derived types (subclasses)
  - ▶ an X "is a" kind of Y
- Derived types have additional properties (more specialised)
- Allows factoring of code
- Useful taxonomy for learning purposes
  - ▶ e.g., collections

# Inheritance Example

```
public class Person {  
    public String name() { .... }  
    public Date dateOfBirth() { .... }  
    public int age() { .... }  
}
```

```
public class Employee extends Person {  
    public String title() { .... }  
    public int salary() { .... }  
}
```

```
public class Student extends Person {  
    public String course() { .... }  
    public Person advisor() { .... }  
}
```

# Be Careful with Inheritance

- Does Car inherit from SteeringWheel?
- "is a"?
- ...or "has a"?
- Car contains SteeringWheel
- ...and Brakes, Tyres, Engine, etc.

# Dynamic Binding

- Selection of method to run is made at run time, not compile time
- Method search starts at bottom of class hierarchy and proceeds upwards
  - ▶ this isn't as slow as it sounds!
- Allows selective replacement of behaviour

# Dynamic Binding Example

```
public class Employee {  
    public int maxExpense() { return 50; }  
    // etc.  
}
```

```
public class Manager extends Employee {  
    public int maxExpense() { return 2000; }  
    // etc.  
}
```

```
public class ExpensePolicy {  
    public boolean approve(Claim c, Employee e) {  
        return (c.amount() <= e.maxExpense());  
    }  
}
```

# Polymorphism

- "Many forms"
- Deal with different types of objects in a consistent way
- Let the object determine how it should perform an action
- Good alternative to if/then/else
- Can greatly reduce amount of changed code as system evolves

# Polymorphism Example

```
public abstract class Printer {  
    public String print(Document d);  
}
```

```
public class PostScriptPrinter extends Printer {  
    public String print(Document d) { .... }  
}
```

```
public class PCLPrinter extends Printer {  
    public String print(Document d) { .... }  
}
```

```
Printer p = printers.find("Building 9");  
status = p.print(myDocument);  
System.out.println(status);
```

# Why Use Objects?

- They model the real world
  - ▶ combination of behaviour and state
- Behaviour may:
  - ▶ just wrap access to state (personnel record)
  - ▶ be very complex (printer)
- They make programming easier!
  - ▶ Reduced complexity, easier maintenance

# Why Use Classes?

- They provide useful abstraction and structure
- They help you think about the application you are building
  - ▶ Class names are nouns, not verbs
  - ▶ Good: ExpenseClaim, Employee, Printer
  - ▶ Bad: Ordering, Personnel, System
- Don't overdo inheritance...

# Final Thoughts

- Objects are simple yet powerful
- Classes can be tricky, but are well worth the effort
- Be religious about encapsulation
- Use polymorphism wherever possible

# One Last Thought

- A good object is ....
- .... a Java object!