CS2510
MODERN PROGRAMMING LANGUAGES

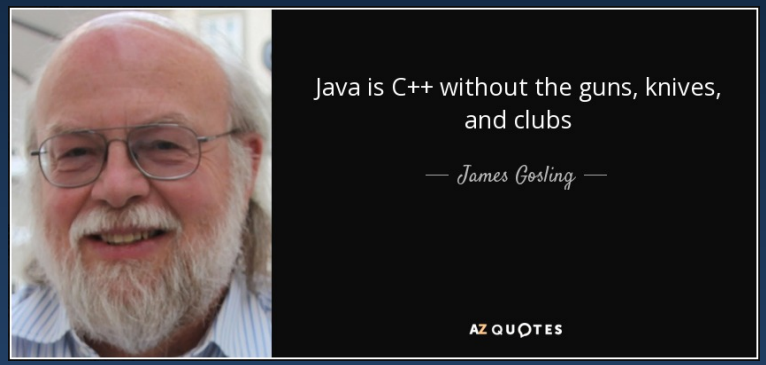
Object-Oriented Programming 4

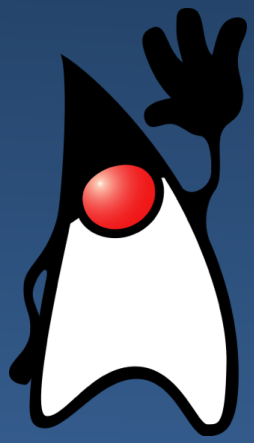
Prof. Peter Edwards
p.edwards@abdn.ac.uk

Java

Introduction to Java

- Originally conceived as a language for intelligent consumer electronics - 1991.
- Sun Microsystems project “Green” developed language Oak -> Java!
  - *Write Once – Run Anywhere*
- Java and C++ syntax very close.





James Gosling, Bill Joy, Guy L Steele Jr, Gilad Bracha, *The Java Language Specification*

WORA …

- Does ***Write Once Run Anywhere*** mean that Java is an interpreted language ?
  - Yes, source is compiled into bytecodes.
- Aren't interpreted languages inherently slower than compiled ones ?
  - Yes.
- Java trades speed for:
  - Platform independence
  - Programmer safety
- Java compilers are pretty good.

Java - Basics

- Class syntax:

```
[access_modifier] [abstract, static, final]
          class class_name {

   // class body

}
```

- Instance variable syntax:

```
[access_modifier] [static, final] type identifier;
```

- Method syntax:

```
[access_modifier] [abstract, static, final]
          return_type method_name(parameters, …) {

// method body

}
```

```
public class Cake {

private int layers;
private float price;
private String bakerName;

// the Cake class constructors

public Cake() {
    layers = 1;
    price = 10.0;
    bakerName = "Unknown";
}

public Cake(String name) {
    layers = 1;
    price = 10.0;
    bakerName = name;
}

// a sample method

public void bake(int temp) {
    System.out.println(bakerName +
       " is baking the cake at " + temp + "C.");
}
}
```

Note the strong similarity to C++

```
Cake cake1 = new Cake();
cake1.bake(250);
```

Java – Basics contd.

- (Almost) everything is an object
  - Only primitive types (boolean, char, int, long, float, double) are not objects
- Method arguments are always passed by value.
- Objects are not copied – only their references are.
- Method overloading – but no operator overloading.
- No structs
- Nice solution to name collisions (***packages***)
- Inherently multi-threaded
  - Threads are supported at the language level and are also objects.
- Powerful and easy-to-use libraries for data structures, multi-threading, networking, I/O, graphics, GUI

Java – Access Controls

- Access modifier on classes:

public	Class declared as `public` is visible to all classes everywhere.
default	Class with no *access_modifier* declaration is visible within its own package (named group of related classes).

- Access modifier on class members:

public	Class members declared as `public` are accessible in any class.
default	Class members with no *access_modifier* declaration are accessible by classes within own package (named group of related classes).
protected	Class members declared as `protected` are accessible by classes within own package (named group of related classes). Additionally, they can be used by classes derived from the class.
private	Class members declared as `private` can only be accessed in own class.

Java – Hello World

- Java source code files all end in .java
- Java compiler expects filename to be name of the public class in the source file (case sensitive).
- Only one public class in a source file; can have other private classes, or nested classes.
- Every Java application must contain a `main` method whose signature is:
  `public static void main(String[] args)`

```
/**
* The HelloWorldApp class implements an application that
* simply prints "Hello World!" to standard output.
*/

public class HelloWorldApp {

public static void main(String[] args) {
    System.out.println("Hello World!");    // Display the string.
}

}
```

comments

public class

main() method

```
% ls

HelloWorldApp.java
% javac HelloWorldApp.java
% ls

HelloWorldApp.class   HelloWorldApp.java
% java HelloWorldApp
Hello World!
%
```

Constructors (& ~~Destructors~~)

- Objects are always allocated in the heap, using `new`, as in:

```
Foo f = new Foo();
```

  - `f` itself is always allocated in the ***stack***
  - the *object* referenced by `f` is allocated in the ***heap*** (heap-dynamic).
- ***Constructors***
  - Method with same name as the class, no return type.
  - Default constructor and constructor chaining.
- No ***Destructors*** in Java!
- ***Garbage Collection***
  - Java keeps track of how many valid references exist for each object – when an object has no more references to it, the memory space it occupies in the heap gets reclaimed.
  - A **`finalize()`** method is implicitly called when the garbage collector is about to reclaim the storage occupied by an object.
    - Programmer can override the default finalize() to implement finalization.

Class Variables & Methods

- Classes can define data members that are associated with the class, instead of each object.
  - `static` keyword used.
- Static methods also supported.
  - Have the `static` modifier in their declarations.
  - Invoked with the class name, without the need for creating an instance of the class:
    - `Cake.getNumberofCakes();`

```
public class Cake {

private int layers;
private float price;
private String bakerName;

// class variable to keep track of number
// of Cake objects
private static int numberOfCakes = 0;

public Cake() {
    layers = 1;
    price = 10.0;
    bakerName = "Unknown";
    numberOfCakes++;
}

public static int getNumberOfCakes() {
    return numberOfCakes;
}

}
```

Inheritance

- Only single inheritance is supported
  (but see *interfaces* later).
- Method overriding is permitted.
- Methods can be declared `final` (cannot be overridden).
- `super` keyword can be used to invoke parent class method/constructor.

```
[access_modifier] class derived_class extends base_class {

   // class body

};
```

```
public class BirthdayCake extends Cake {

// BirthdayCake class constructor calls
// parent class constructor using “super”

public BirthdayCake(String name) {
    super(name);
}

// a sample final method - cannot be overridden

public final void putCandlesOnCake(int num) {
     System.out.println("Putting " + num +
     " candles on the birthday cake.");
}

}
```

Inheritance

- ***Abstract classes*** help reduce code dependencies.
  - To make a class abstract, declare with keyword `abstract`
  - Abstract class cannot be used to create objects.
- Contain one or more *abstract methods*.
  - No implementation, must be overridden.

Multiple Inheritance via *Interfaces*

- Java only has single inheritance.
- A special category of class provides some of the benefits of multiple inheritance (***interface***).
  - An *interface* can include only certain method declarations and named constants.
- Declaration begins with `interface` keyword.
- A class `implements` an interface (and all its methods).

```
[access_modifier] interface interface_name
                            [extends other_interface, …] {

   // constant declarations
   // method signatures, default or static methods

};
```

```
[access_modifier] class class_name implements
                                        interface_name, … {

   // class body
   // implements method signatures from interface(s)

};
```

```
public interface Bakeable extends Edible {

    void bake();
}

public class Cake implements Bakeable {

    // Cake data members, constructors, etc here

    public void bake() { // bake method body };
}
```

Generics

- ***Generics*** enable types (classes and interfaces) to be parameters when defining classes, interfaces and methods.
- Using generics, programmers can implement generic algorithms that work on collections of different types.

```
class class_name<T1, T2, ..., Tn> {

    // generic class body

};
```

```
public class Box<T> {    // T stands for "Type"

    private T t;

    public void set(T t) { this.t = t; }

    public T get() { return t; }
}
```

Note use of `this` keyword
Within an instance method or a constructor, `this` is a reference to the *current object.*

```
Box<Cake> cakeBox = new Box<>();
cakeBox.set(cake1);
```

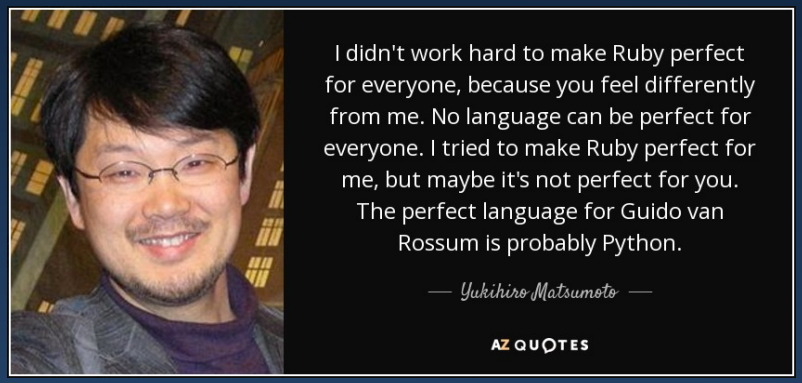
Java - Summary

- Design decisions to support OOP are similar to C++
- Java doesn't support procedural programming – everything has to be in a class (see Hello World).
- No parentless classes
- Interfaces provide a simple form of support for multiple inheritance.
- Polymorphism - ad hoc, subtyping, parametric.
- Nested classes.
- Dynamic binding is used as "normal" way to bind method calls to method definitions.

Ruby

Introduction to Ruby

- ***Ruby*** - a dynamic, reflective, object-oriented, general-purpose programming language.
- Ambition was to realise a truly object-oriented scripting language.
- Designed and developed in the mid-1990s by Yukihiro "Matz" Matsumoto.



David Flanagan, Yukihiro Matsumoto, The Ruby Programming Language

Ruby - Basics

- Everything is an object!
- All computation is through message passing.
- All variables are type-less references to objects.
- Class definitions are executable, allowing secondary definitions to add members to existing definitions.
- Method definitions are also executable.
- Access control is different for data and methods:
  - It is private for all data and cannot be changed
  - Methods can be either public, private, or protected (default: public)
- Single inheritance.
- Operator overloading, but no method overloading!

Ruby - Basics

- Class syntax:

```
class class_name

   # class body

end
```

- Instance variable syntax:

```
@identifier
```

- Method syntax:

```
def method_name(parameters, …)

    # method body

end
```

```
class Cake

  # no need to declare instance variables

  # Cake class has one constructor (initialize)

  def initialize
    @layers = 1
    @price = 10.0
    @bakerName = "Unknown”
  end

  # a sample method

  def bake(temp)
    puts "#{@bakerName} baked the cake at #{temp} C."
  end
end
```

```
c1 = Cake.new
c1.bake(212)
```

Inheritance

- Single inheritance between classes.
  - Default superclass is **`Object`** (which inherits from **`BasicObject`**)
- Method overriding is permitted.
- **`super`** keyword invokes parent class version of method.
- Modules (*mixins*) are used to provide behaviour similar to multiple inheritance.

```
class derived_class < base_class

   # class body

end
```

```
class BirthdayCake < Cake

  # BirthdayCake constructor (initialize) calls Cake constructor 1st

  def initialize
    super
    @bakerName = ”Owen”
  end

  # bake overrides inherited version and changes number of parameters

  def bake(temp, duration)
    puts "#{@bakerName} baked the cake at #{temp} C for #{duration} mins."
  end

  def putCandles(num)
    puts “Putting #{num} candles on the birthday cake.”
  end

end
```

```
c1 = Cake.new
c1.bake(212)
c2 = BirthdayCake.new
c2.bake(250, 30)
```

Ruby - Summary

- All variables are typeless and polymorphic.
- Subclasses are not necessarily subtypes.
- Access controls are weaker than those of other languages that support OOP.
- Does not support abstract classes.
- Does not fully support multiple inheritance (modules as *mixins*).

Closing Discussion

UNIVERSITY OF ABERDEEN

Implementing OO Constructs

- Two interesting and challenging parts:
  - Storage structures for instance variables
  - Dynamic binding of messages to methods

Instance Data Storage

- ***Class instance records*** (CIRs) store the state of an object
  - Static (built at compile time)
- If a class has a parent, the subclass instance variables are added to the parent CIR.
- Because CIR is static, access to all instance variables is handled efficiently.

Dynamic Binding of Methods Calls

- Methods in a class that are statically bound need not be involved in the CIR; methods that will be dynamically bound must have entries in the CIR
  - Calls to dynamically bound methods can be connected to the corresponding code through a pointer in the CIR.
  - The storage structure is sometimes called virtual method tables (***vtable***)
  - Method calls can be represented as offsets from the beginning of the *vtable*.

Summary

- OO programming involves four fundamental concepts: *abstraction*, *encapsulation*, *inheritance*, *polymorphism*.
- Major design issues:
  - Exclusivity of objects, subclasses and subtypes, type checking and polymorphism, single and multiple inheritance, dynamic binding, explicit and implicit de-allocation of objects, nested classes.
- Smalltalk is a pure OO language.
- C++ has two distinct type systems (hybrid).
- Java is not a hybrid language like C++; it supports only OOP.
- Ruby is a relatively recent pure OOP language.
- Implementing OOP involves some new data structures.