Inheritance & Polymorphism

Murray Cole

Classifying Things

- Hierarchies help us to classify things and understand their similarities and differences
- Some aspects are common to everything at some level
- Some aspects don't make sense in all categories
- Some aspects are inherited implicitly from higher levels

Manipulating Things

Similarly, build operations on our classifications at different levels

- `getName` on any lobster might return "A Lobster"
- `getName` on a particular pet cat might return "Tiddles the Cat"

`getName` can be applied to any animal (it is generic), but may behave differently on some sub-categories (it can be specialised)

Inheritance

This also applies to the things we want to model (with classes) in our programs

We could (dangerously!)

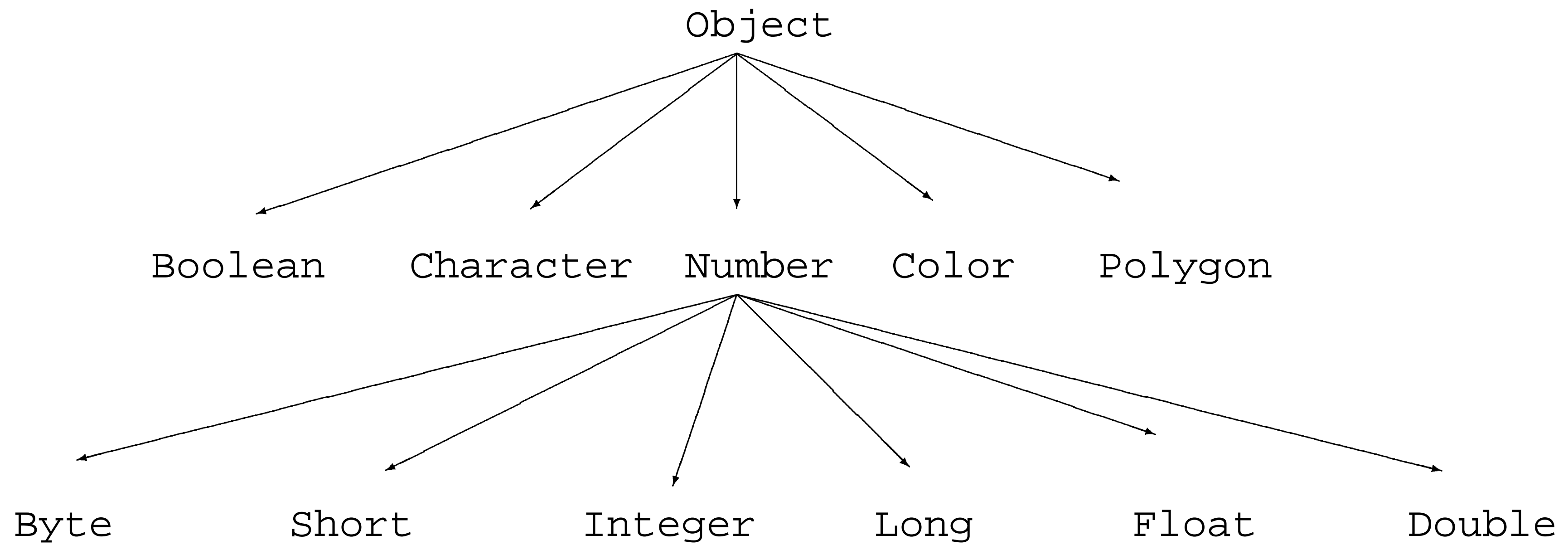
- write the details afresh for each new class
- cut & paste the bits which are shared

Much better, let the language do the work with inheritance and overriding

Objects and the class hierarchy

- Classes in an object-oriented programming language such as Java are organised into a hierarchy.
- In the Java class hierarchy we have `Object` at the root, meaning that every class which we define is a subclass of `Object`.
- Below `Object` we have `Boolean`, `Character`, `Number`, `Color`, `Polygon` and many, many others.
- Beneath `Number` we have `Byte`, `Short`, `Integer`, `Long`, `Float` and `Double`.

The Class Hierarchy



Superclasses and inheritance

- We refer to the "parent" of a class as its superclass.
- Classes inherit from their superclass (and from its superclass . . .)
- We can think of inheritance as an **is-a** relationship. An `Integer` **is a** `Number`. A `Horse` **is a** `Mammal`.
- The class `Object` defines methods such as `equals()` and `toString()` so we know that objects of any subclass of `Object` (and so of any class) can be tested for equality or converted to a `String` representation.

Extending the Hierarchy by Subclassing

- A class declaration can include the phrase **extends** c

```
class CompSciStudent extends Student {
    public String username;
    public int diskquota;
}
```

- As well as new fields, a subclass may add new methods
- Or a subclass may replace methods of its superclass with its own versions, which is called method overriding.

When overriding is useful

- Imagine we extend the `Student` class to include a method for sending a message to the student's DOS.

```
class Student {
    public void sendDosNote(String msg) {
      // send note to DOS using internal mail
      InternalMail.sendLetter(this.dos.address,msg);
    }
}
```

When overriding is useful, II

- For a CS student (who has a DOS in CS), there is a better way of sending a message, using email. So we override the `sendDOSNote` method.

```
class CompSciStudent extends Student {
    ...
    public void sendDosNote(String msg) {
      // send email to DOS, should be reliable
      UnivEmail.sendEmail(this.dos.email,msg);
    }
```

Collection classes

- The collection classes in Java are defined to be able to store instances of `Object`.
- This means that we can store a mixed bag of objects as a collection (a *heterogeneous* collection).
- In other programming languages collections are usually constrained to store only a single kind of object (a *homogeneous* collection).

Methods of the `Vector` class

`Vector()` This is the constructor of the class, as usual. We create a new `Vector` with `Vector v = new Vector();`

`addElement()` This method adds an object to the vector. We would invoke it to add the `String` value `"abc"` as `v.addElement("abc");` We could then follow this with a `Date` object with `v.addElement(new Date());`

`removeElement()` We can remove a given element with this method. For example, `v.removeElement("abc");`

`elementAt()` We can inspect the object stored at a particular location by supplying an integer index into the vector.

Methods of the `Vector` class

`insertElementAt()` We can insert an element into the vector at any position. For example, the method invocation `v.insertElementAt("def", 0);` will insert the `String` `"def"` at position zero in the vector.

`setElementAt()` Vectors are updateable so we could overwrite the string stored at location zero with another like this: `v.setElementAt("ghi", 0);`

`iterator()` We can examine all of the elements of the vector by asking for them to be supplied as an `Iterator`.

Polymorphism

- Vectors are a polymorphic data structure (“polymorphic” = “having many forms”). This means that the same operations can be used for vectors of integer objects as for strings or whatever.

- In contrast, many programming languages only support only the definition of monomorphic data structures (“monomorphic” = “having only one form”). In a monomorphic programming language separate implementations of the vector operations would be needed for each kind of vector.

- The use of a single definition which is re-usable at different types (i.e. polymorphic) saves effort and reduces the chance of error.

Processing objects by type

- When we retrieve objects from a collection we need to restore the class which they had before they were coerced to `Object`.
- Casting up the class hierarchy, an operation which is sometimes called widening, is always safe (every `Integer` is a `Number` ...).
- However, casting down the class hierarchy (narrowing), is not always safe (not every `Number` is an `Integer` ...).
- If we get this down-casting wrong the operation will fail at run-time with a `ClassCastException`. We can investigate the class of an object at run-time by the use of the **`instanceof`** relation.

```
1  static void countAndPrint (Vector v) {
2      Iterator i = v.iterator();
3      int stringCount = 0;
4      int intTotal = 0;
5      while (i.hasNext()) {
6          Object o = i.next();
7          if (o instanceof String)
8              stringCount++;
9          else if (o instanceof Integer) {
10             Integer n = (Integer) o;
11             intTotal += n.intValue();
12         }
13     }
14     System.out.println("Strings: " + stringCount);
15     System.out.println("Integer total: " + intTotal);
16 }
```