G51PRG: Introduction to Programming Second semester

Lecture 3

Natasha Alechina
School of Computer Science & IT
nza@cs.nott.ac.uk

Previous lecture

- basic types in Java are passed by value
- objects are passed by reference
- this has consequences for making assignments and comparisons between objects

Plan of the lecture

- ***static*** modifier : revision
- inheritance, extending classes

Static fields

- storage location for a static field is created only once;
- does not depend on any object (it is shared by all objects in the class);
- can be accessed without having an instance of a class.

Examples of uses

- Constants like **`Math.PI`**
- Something which has the same value for all objects in the class; for example a variable to hold retirement age in class Person should be declared static; it is the same for all objects in the class, so why duplicate it and waste memory
- More interesting use: if we want a class to keep track of how many instances of the class are created, or some other class-level information, we need a static variable to hold it.

Example: counting instances

```
class Invitation{
  static int count;
  String name;

  Invitation(String name){
    this.name = name;
    count++;
  }
}
```

Example: other class properties

```
class Invitation{
  static String longestName;
  String name;
  Invitation(String name){
    this.name = name;
    if (name.length() >
            longestName.length()){
        longestName=name;
    }
  }
}
```


Static methods

- do not depend on any object of the class;
- can be called without having to create an instance of a class

Examples of static methods

- **`main`** method is always static
- any method which can work without having access to instance fields, for example class **`Math`** has a static method

```
public static double sqrt(double a,
  double b)
```

which you can call when you need to compute a square root:

```
public double distance(Point p){
  return Math.sqrt(Math.pow((p.x-x),2) +
  Math.pow((p.y-y),2)); }}
```


Examples of static methods contd.

If a method returns the value of a static field, for example in the Person class we could have put a static field

```
private static int retirementAge
```

and a static method

```
public static int retiresAt()
```

which returns the value of this field.

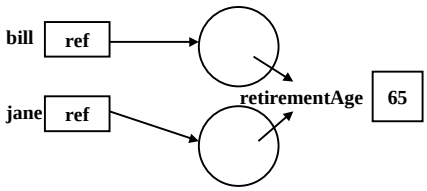
Retirement example

```
class Person {
  private static int retirementAge = 65;
  private int age;
  private String name;
  public static int retiresAt() {
      return retirementAge;
  }
  public Person(String name, int age){
     this.name = new String(name);
     this.age = age; }
```

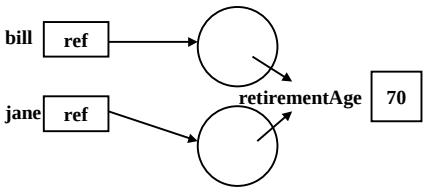

Retirement example contd.

```
public static void main(String[] args){
  Person bill = new Person(“Bill”,51);
  Person jane = new Person(“Jane”,55);
  System.out.println(retiresAt());
  bill.retirementAge = 70;
  System.out.println(jane.retirementAge);
  System.out.println(retiresAt());
}
```

How it works



How it works



Extending classes (Inheritance)

- Extending classes is one of the main techniques of object-oriented programming.
- In a nutshell, you write the code for class A. Then you define a class B which extends A: simplistically, this means that B has all the code A has (you don't have to type it again) and you can add additional fields and methods which only B has. In other words, B inherits fields and methods of A (not the ones which are declared private).
- We'll see how inheritance really works in this and subsequent lectures.

`extends` keyword

The general form for extending a class is

```
class Subclass extends Superclass {
```


`Example`

(Taken from: Ken Arnold and James Gosling, The Java programming language, Second edition)

```
class Point {
   public int x,y;
   public void clear() {
      this.x = 0;
      this.y = 0;
   }
}
class Pixel extends Point {
   Color colour;
}
```


`Example continued`

- Pixel class inherits fields x and y and method clear() from the Point class. We don't have to repeat the code for clear() in the Pixel class.
- If in a program we created a Pixel object then we can access its x and y fields and call its clear() method.

```
Pixel p = new Pixel();
p.x = 2.0;
p.y = 1.0;
p.clear();
```

What is inherited

- Subclasses inherit those superclass members (fields and methods) declared as public or protected.
- Subclasses inherit those superclass members declared with no access specifier as long as the subclass is in the same package as the superclass.
- Constructors are not members and are not inherited.

Why use inheritance

- Why don't we just cut and paste the code into Pixel?
- Why do we need the Point class if it contains strictly less information than Pixel?

General use of extending classes

- Define a class (a parent class, or superclass) and then use it to define a number of specialised classes based on it (extending it). The specialised classes are called subclasses, or children.
- The parent class holds all the methods which all its children share. If you need to change their implementation you only have to do this once.
- Classes are smaller and easier to understand and debug.
- All the code which is written to work with objects of the parent class will work with the objects of children classes.

Writing the code once

- If you have a method **`distance`** which calculates distance between two **`Points`**, you can use the same method for **`Pixels`**.
- So if the **`Point`** class contains

```
public double distance(Point p){
    return Math.sqrt(Math.pow((p.x-x),2)
                + Math.pow((p.y-y),2));
}
```

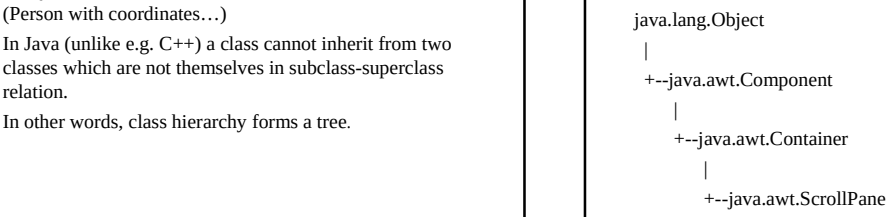
- you can do

```
Pixel p = new Pixel(4,5,Color.green);
p.distance(new Pixel(5,6,Color.red));
```


Multiple inheritance?

- Can you inherit from both a Point class and a Person class? (Person with coordinates…)
- In Java (unlike e.g. C++) a class cannot inherit from two classes which are not themselves in subclass-superclass relation.
- In other words, class hierarchy forms a tree.

Example: AWT (Abstract Windowing Toolkit)



Using the parent's constructor

- Usually, a constructor will need to set more values than in the parent class
- Can call super() to do the work which the constructor in superclass does
- Using a constructor from a superclass is called superconstructing.

Using the parent's constructor

- For example, if Point has the following constructor:

```
Point(int x, int y){
    this.x = x;
    this.y = y;
 }
```

- Then we could add the following constructor to Pixel

```
Pixel(int x, int y, Color color){
    super(x,y);
    this.color = color;
 }
```


Next lecture

- **`super`** keyword
- overriding methods
- polymorphism

- I have not yet covered everything you need for the first exercise but you can do most of it (apart from overriding the **`print()`** method).

Summary and further reading

- Reading: Java Gently Chapter 9 and Sun Java tutorial

http://java.sun.com/docs/books/tutorial/java/javaOO/index.html