Introduction to Java

Handout-2b

OOP key principles

- Abstraction √
- Encapsulation √
- Inheritance √
- Polymorphism

Polymorphism

- It's Greek for "many shapes". "Name re-use" would be a better name
- Two types of polymorphism
  - Overloading
  - Overriding

Overloading (i)

- Same name can be used for several different methods
  - Methods with the same name should have different signatures
  - The return type and the exceptions the method may raise are not looked at when resolving same name functions
- Resolved by the compiler at compile time

Overloading (ii)

Ex:

```
public static int parseInt(String s) throws
   NumberFormatException

public static int parseInt(String s, int radix) throws
   NumberFormatException
```

Overriding

- It occurs when a class extends another and the subclass has a method with the exact same signature as a method in the superclass. Which one is invoked?
  - If it's an object of the subclass, the subclass one
  - If it's an object of the superclass, then superclass
- Resolved at run-time

Forcing Overriding off: `final`

- Assume the java.lang.Math class where all trig operations are done in radians not degrees. Can we extend the class and override just the trig methods such that they work in degrees instead of radians?

Forcing Overriding off: `final`

- The code might look like this (the code below won't work!)

```
public class DegreeMath extends java.lang.Math {
  public double sin(double d) {
    double radians = super.toRadians(d);
    double result = super.sin(radians);
    return super.toDegrees(result);
  }
}
```

Forcing Overriding off: `final`

```
public final class Math {
  public static native double sin(double a);
…
}
```

- A class labeled as `final` cannot be extended
- The sin() method is `static`. Static methods do not participate in overriding. Why?

Forcing Overriding off: `final`

- `final class`: the class may not be further extended by anyone
- `final` *`someMethod`*: the method may not be overridden when its class is inherited
- Usually we prevent further inheritance for reasons of performance and security

Forcing Overriding: `abstract`

- `abstract`: a keyword that forces overriding
- The keyword tells the compiler "this thing is incomplete and must be extended to be used"
- `abstract` can be applied to individual methods or the whole class
  - `abstract` *`someMethod`*: the method has no body; its purpose is to force some subclass to override it and provide a concrete implementation
  - `abstract class`: zero or more of its methods are abstract

Forcing Overriding: `abstract`

- You make a method abstract when three conditions are fulfilled
  - There will be several subclasses
  - You want to handle all the different subclasses as an instance of the superclass
  - The superclass alone does not make sens as an object