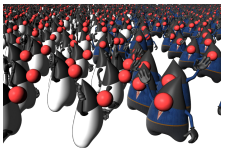
Java Programming 2 – Lecture #6 – Jeremy.Singer@glasgow.ac.uk



Member Visibility Modifiers

In order to limit the visibility of class members, i.e. fields and methods, it is possible to specify an access modifier as part of a member declaration. The table below shows the extent of visibility for members with the various modifiers.

Modifier	Same class	Same package	Any subclass	Any class
`public`	✓	✓	✓	✓
`protected`	✓	✓	✓	
(default)	✓	✓		
`private`	✓			

`private` data fields are used for internal class state that must be accessed in a controlled way, perhaps through getter and setter methods. See example below.

```
public class Person {
  private int age;

  public int getAge() {
    return this.age;
  }

  public void setAge(int age) {
    assert(age>=0);  // age must be non-negative
    this.age = age;
  }
}
```

`private` methods are used for non-API methods that are utility/helper methods internal to a class.

Class Inheritance

Some classes are related to each other via an inheritance hierarchy. More general classes will have characteristics in common with more specialized classes. A class B can be defined as a subclass of class A, in which case B inherits the members of A. Effectively B ***is-a*** specialized version of A, or an extension of A[1]. Subclasses are declared in Java using the `extends` keyword i.e.

```
public class B extends A { … }
```

This notion of class inheritance is one of the most powerful object-oriented features of Java. The Java language supports single inheritance (rather than multiple inheritance like C++). This means that the

[1] See http://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html for more details.

Java inheritance hierarchy is a tree rather than a directed acyclic graph. The root of the Java inheritance hierarchy is the `java.lang.Object` class.

Method Overriding

If a subclass has a method with an identical signature (name, return type and parameter types) as a superclass, then the subclass method is said to ***override*** the superclass method. Effectively, this is the way that the subclass specifies alternative behaviour to the superclass. See the example below.

```
public class Person {
  private Gender g;
  public String getTitle() {
    String title;
    if (g==Gender.MALE) title = "Mr.";
    else title = "Ms.";
    return title;
  }
}

public class TitledPerson extends Person {
  private String title;
  public String getTitle() {
    return this.title;
  }
}
```

Polymorphism

Polymorphism[2] literally means 'many forms'. It means that wherever an instance of class `A` is expected in a program, one may supply an instance of class `B` which is a subclass of `A`. This is an application of the *Liskov substitution principle*[3]. Polymorphism is supported by virtual method invocation in Java – method calls are dynamically dispatched based on the runtime type of the receiver object.

Questions

1) Can `static` methods be overridden in the same way as instance methods? If so, why? If not, why not?
2) What is the point of a class with only `private` constructors?

[2] See http://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html for details.
[3] This is starting to get into hard-core CS theory, see http://c2.com/cgi/wiki?LiskovSubstitutionPrinciple if interested.