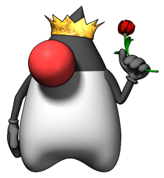
Java Programming 2 – Lecture #8 – Jeremy.Singer@glasgow.ac.uk



Constructor Chaining

In a constructor body, the first action must be to call a superclass constructor. If there is no explicit superclass constructor call, then the compiler inserts a default no-args constructor to the superclass, i.e. `super()`. Every time a constructor is invoked, there is a chain of constructor calls going up the inheritance hierarchy all the way back to `java.lang.Object`. We can see this by inserting `println` statements into a set of constructors:

```
public class A {
  public A() { /*super();*/ System.out.println("A constructor"); }
}
public class B extends A {
  public B() { /*super();*/ System.out.println("B constructor"); }
}
public class C extends B {
  public C() { /*super();*/ System.out.println("C constructor"); }
}
```

If there is not a no-args constructor in the superclass, then the subclass constructor must specify *explicitly* which superclass constructor is to be called.

Calling Superclass Methods

In a similar way to superclass constructor invocation, it is possible to invoke a method from the superclass that is overridden in a subclass, using the `super` pseudo-variable[1]. The `super` variable is a reference to the current instance, with the type of its immediate superclass in the inheritance hierarchy. Invoking a method through the `super` reference is not subject to polymorphic overriding (unlike method invocation via the `this` reference.)

Leaves on the Inheritance Tree

In some cases, a developer may not want a class to be subclassed. If the class is marked as `final`, then it cannot be subclassed. Similarly, if a method is marked as `final`, then it cannot be overridden in a subclass. `final` classes and methods can improve security[2] (or predictability) – a developer can be certain that an instance of a `final` class does what is expected, rather than any overriding behaviour. In the code below, marking the PasswordChecker class as `final` (or the check method) would prevent subclass injection attacks.

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

[2] See http://www.oracle.com/technetwork/java/seccodeguide-139067.html#4 for attacks and corresponding defence techniques.

```
public class PasswordChecker {
  public boolean check(String username, String password) {
    String passwordHash = hash(password);
    String correctHash = lookupHash(username);
    return (passwordHash.equals(correctHash);
  }
}

public class DodgyChecker {
  public boolean check(String username, String password) {
    return true;
  }
}
```

More on Exceptions

Recall that when an `Exception` is thrown in a `try` block, the associated `catch` blocks are examined in sequential order and only the *first* matching `catch` block (if any) is executed. This means that `catch` blocks should be ordered from least general to most general. The Java compiler will complain about unreachable code if more general `catch` blocks (e.g. `catch (Exception e){}`) are positioned above less general catch blocks.

Three useful methods in Exception objects are:

- `e.getMessage()` – returns a String with some information about the exception
- `e.printStackTrace()` – prints out the calling context of the exception at the point it was thrown
- `e.toString()` – generally returns a String indicating the concrete type of the Exception instance

`Exception` messages may be printed to the `System.err PrintStream`, rather than the usual `System.out PrintStream`.

Questions

How would you create a constructor for class Foo that creates an exact copy of another instance of Foo? See the helpful Java Practices website[3] for more details.

[3] http://www.javapractices.com/topic/TopicAction.do?Id=12