Crash Course Session 04—Remaining Details

CS1102S: Data Structures and Algorithms

Martin Henz

January 22, 2010

Generated on Friday 22nd January, 2010, 17:34

Crash Course Session 04—Remaining Details

CS1102S: Data Structures and Algorithms

Martin Henz

January 22, 2010

Generated on Friday 22nd January, 2010, 17:34

Three Ways of Using Classes in Other Packages

- Wildcard import
- Single-class import
- Direct addressing

Wildcard Import

- Import all classes from a package using ∗
- Example:

```
import javax.swing.*;   // all classes visible

class ImportTest {
  public static void main(String[] args) {
    JOptionPane.showMessageDialog(null, "Hi");
    System.exit(0);
  }
}
```

Single-class Import

- Import specific class by naming it
- Example:

```
import javax.swing.JOptionPane;

class ImportTest {
  public static void main(String[] args) {
    JOptionPane.showMessageDialog(null, "Hi");
    System.exit(0);
  }
}
```

Direct Addressing

- Name the full path of the class, including package name and sub-package names
- Example:

```
class ImportTest {
    public static void main(String[] args) {
        javax.swing.JOptionPane
            .showMessageDialog(null, "Hi");
        System.exit(0);
    }
}
```

Constructors

- Constructor gets called when you create a new instance using new
- Used to initialize object fields
- Constructor name must be the name of the class

Constructors (cont)

- If you do not specify a constructor, a default constructor is created. Thus if you define a class:

```
class MyClass {
    // no constructor here
    ...
}
```

  it becomes

```
class MyClass {
    MyClass() {
        super();
    }
}
```

Differences between Methods and Constructors

- Constructors do not have a return type
- Constructors do not have a return statement
- The first line of a constructor must be a call to another constructor in the same class, or a call of a constructor of the superclass.
- If there is no such a first line, the compiler inserts a call:

```
super();
```

Constructor Call in Same Class

- One constructor can call another constructor in the same class, using this
- Example:

```
public class Point {
    int m_x; int m_y;
    public Point(int x, int y) {
        m_x = x; m_y = y;
    }
    public Point() {
        this(0, 0);
    }
}
```

Constructor Call of Superclass

- One constructor can call a constructor of the superclass, using super
- Example:

```
public class ColorPoint extends Point{
    Color color;
    public ColorPoint(int x, int y, Color c) {
        super(x,y);
        color = c;
    }
}
```

Interfaces

- Interface contains methods that have to be defined by any class that implements it.
- Similar to completely abstract class (but cannot have static methods)
- Interface methods do not have bodies

Declaring an Interface

- Interfaces are declared using interface
- Example:

```
public interface ActionListener {
  public void actionPerformed(ActionEvent e);
}
```

Implementing an Interface

- Interfaces are used ("implemented") via implements
- Example:

```
public class MyPanel extends JPanel
          implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    /* Method body */
  }
  // Everything else in this class.
}
```

this in Methods

- Every non-static method can refer to the current object using `this`
- `this` can be used wherever identifiers are used.
- Examples:

```
p(this.myfield);
this.myMethod(1,2);
someFunction(1,2,this);
```

super in Methods

- We have seen super in action for constructors
- super can be used to call a superclass method
- Example:

```
class Himalayan extends Cat {
    public Himalayan() {}
    public Himalayan(String nameIn) {
        name = nameIn;
    }
    public String getName() {
        return (name + " the Himalayan");
    }
    public String getNameAsCat() {
        return super.getName();
}   }
```