Introduction to Java V: Abstract Classes and Interfaces

CS 1025 Computer Science Fundamentals I

Stephen M. Watt

University of Western Ontario
Behaviour Hierarchy Problems

• Sometimes there will be a set of subclasses that share a base class, but where it only makes sense to have objects that belong to the subclasses.

  There will never be an object belonging just to the base class.

• “Abstract” base classes are used for this situation.

• Sometimes we want a set of classes to share some behaviour but they do not share a base class.

• “Interfaces” are used for this situation.
• For Sudoku we can have

```java
public abstract class Slice {
    abstract public int size();
    abstract public int getValue(int i);
}

public class RowSlice extends Slice {
    RowSlice(Tableau t, int rowno) {
    }
    int size() {
    }
    int getValue(int i) {
    }
}

public class ColumnSlice extends Slice {
    ColumnSlice(Tableau t, int colno) {
    }
    int size() {
    }
    int getValue(int i) {
    }
}
```
Abstract Classes II

• Methods in an abstract class may either be “abstract,” in which case no body is given, just a semi-colon.

• Normal (non-abstract) methods may be given that use other normal methods and abstract methods to give a common implementation.
Non-Abstract Methods in Abstract Classes

• An example:

```java
abstract class DoIt {
    protected String _afix;

    protected DoIt(String afix) {_afix = afix; }

    abstract public void once(String s);

    public void twice() {
        once("First time");
        once("Second time");
    }
}
```
class Say extends DoIt {
    public Say(String afix) { super(afix); }

    public void once(String s) {
        System.out.println(_afix+s);
    }
}

class Sing extends DoIt {
    public Sing(String afix) { super(afix); }

    public void once(String s) {
        System.out.println(_afix+s+_afix);
    }
}
public class Prolix {
    public static void main(String[] args) {
        DoIt say = new Say("Ahem! ");
        DoIt sing = new Sing("...Tra-la-la...");

        say.twice();
        sing.twice();

    }
}

Ahem! First time
Ahem! Second time
...Tra-la-la...First time...Tra-la-la...
...Tra-la-la...Second time...Tra-la-la...
Abstract Classes Conclusion

• An abstract class is used to collect behaviour, but when there will not be any objects belonging *just* to that class.

• New objects can be allocated that belong to non-abstract subclasses, but not to the abstract base class itself. (E.g. can do “new Sing” or “new Say” but *not* “new DoIt”)

• Can declare variables, parameters, etc to have an abstract class as their type.

• This means that the actual values must be objects of *some* subclass of the abstract class, but we don’t know (or don’t care) which.
Interfaces

• Interfaces are like abstract classes except:
  – All methods are implicitly public and abstract.
  – All fields are implicitly public static final.

• These restrictions allow:
  – A common situation to be handled elegantly.
  – An efficient language implementation that does not impose any conditions on the layout of the objects.
  – Classes to implement multiple interfaces.
interface Vocal {
    void sing(String s);
    void chant(String s);
}

interface Animal {
    boolean canFly();
    int nLegs();
}

class Dog implements Vocal, Animal {
    private String _name;
    Dog(String name) { _name = name; }

    public void sing(String s) {
        System.out.println(_name + " sings: Aroooo " + s);
    }
    public void chant(String s) { sing(s); sing(s); sing(s); }
    public boolean canFly() { return false; }
    public int nLegs() { return 4; }
}
public class Independent {
    public static void main(String[] args) {
        Dog d = new Dog("Rover");

        vocality(d);
        animality(d);
    }

    // This method requires the Vocal interface.
    public static void vocality(Vocal v) {
        v.sing("O Canada");
    }

    // This method relies only on the Animal interface.
    public static void animality(Animal a) {
        System.out.println("Number of legs = " + a.nLegs());
    }
}
• Interfaces are similar to abstract classes in that:
  – Variables and parameters may be declared to have an interface.
  – Classes can implement interfaces and thereby support polymorphic programming.
  – Interfaces can extend other interfaces (leading to the idea of sub-interfaces and super-interfaces.)

• Interfaces are different from abstract classes in that:
  – Classes may directly implement several interfaces, but can directly extend only one base class.
  – All methods in interfaces are abstract.
  – Classes “implement” interfaces and “extend” classes.