

A Java Interlude

Illustrating inheritance and javadoc (from Parsons).

```
/**
    IDCard.java <br>
    This class is a simple ID card class that
    contains only a title, initial and
    surname.
    @author David Parsons
    @version 1.0

*/
public class IDCard {

    /** the private attributes, note the
    initial is represented by a single
    'char', not a string of characters.
    */
    private String title;
    private char initial;
    private String surname;

    /** the constructor sets all three
    attributes from the parameter list
    @param titleIn is the title of the person
    @param initialIn is their first initial
    @param surnameIn is their surname
    */
    public IDCard(String titleIn, char
    initialIn, String surnameIn) {
        title = titleIn;
        initial = initialIn;
        surname = surnameIn;
    }
}
```

```

/** the three 'get' methods return the
values of the three attributes
*/
    public String getTitle()    {

        return title;
    }

    public char getInitial()    {

        return initial;
    }

    public String getSurname()  {

        return surname;
    }
/** to enable easy display of the object
we provide our own version of the
'toString' method. This means that using
'println' with IDCard objects displays
useful information.
*/
    public String toString()    {

        String tempString = new String(title +
        initial+ ". " + surname);
        return tempString;
    }

}

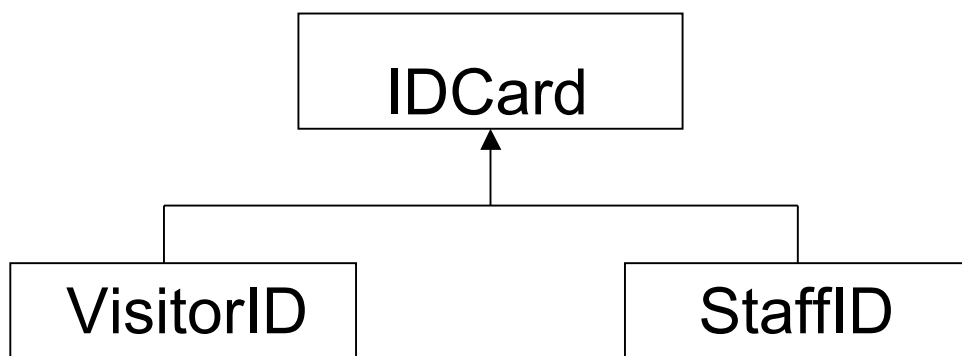
```

To obtain fullest information from `javadoc` add flags as follows

```
javadoc -private -author -version  
IDCard.java
```

This produces `.html` files with information on attributes, classes and methods.

For full details on use of `javadoc` itself consult the relevant online documentation (start from the API pages linked from the main DIS pages, or explore directly).



`IDCard` is the superclass of `VisitorID` and `StaffID` which both extend it (and so inherit from it). These classes illustrate the use of 'extends' and show how to call superclass methods using 'super'.

```
/** StaffID.java - this file contains a  
    derived class of IDCard called  
    StaffID that adds methods to get and  
    set the name of a staff member's  
    department and extends the superclass  
    version of 'toString'.  
*/
```

```

public class StaffID extends IDCard    {

    //an extra attribute is added
    private String department;

    //constructor passes its parameters to
    //the superclass

    public StaffID(String title, char
    initial, String surname)  {

        super(title, initial, surname);
    }

    //additional set and get methods

    public void setDepartment(String
    deptIn)  {

        department = deptIn;
    }

    public String getDepartment()  {
        return department;
    }

    //a version of 'toString' which
    //extends the superclass version

    public String toString()  {
        return super.toString() + " of the
        " + department + " department";
    }
}

```

The code for the VisitorID and a test program can be found in the public directories for the course.

In an interface declaration all the methods are abstract, i.e. they have no implementations. A class is abstract if at least one method is abstract. The role of such classes is to be a superclass for other classes. It is not possible to create objects from an abstract class.

Here is an abstract class (from Goodrich & Tomassia) for stepping through and printing out a numerical progression. It defines an integer `cur` field which stores the current value, and the following four methods:

- `first()` : reset the progression to the first value
- `nextValue()`: step to the next value and return it
- `valueAt(i)`: set progression to ith value, return it
- `printProgression(n)`: print first n values

```
public abstract class Progression {
//a simple class for numeric progressions
    protected int cur; //current value

    Progression() {
        cur = 0;
    }

    protected int currentValue() {
//return the current value
        return cur;
    }
}
```

```

protected int first()    {
//reset to the first value
    cur = 0 ;
    return cur;
}

protected int nextValue() {
//should return the "next" value
//but this superclass just returns cur
//value
    return cur;
}

protected int valueAt(int i) {
//return the ith value
    first();
    for (int j = 1; j < i; j++)
        nextValue();
    return currentValue;
}

public void printProgression(int n) {
// print the first n values

System.out.println(first() + " " );
for (int i = 1; i < n; i++)
    System.out.println(nextvalue() + "
");
System.out.println();
}
}

```

The class `ArithProgression` inherits `cur` and methods `currentValue()`, `valueAt()`, and `printProgression(n)`, it adds a new field `inc`, and overrides other methods.

```
class ArithProgression extends
Progression {
//arithmetic progression

    protected int inc; //increment value

    ArithProgression() {
        inc = 1; //default increment
    }

    ArithProgression(int increment) {
        inc = increment;
    }

    protected int first() {

//reset (overriding)
        cur = 0;
        return cur;
    }

    protected int nextvalue() {

//next (overriding)
        cur = cur + inc;
        return cur;
    }
}
```

```

class GeomProgression extends Progression
{
//geometric progression

    int b; //the base of the progression

    GeomProgression() { b = 2 /*default*/;
}
    GeomProgression(int base) {
        b = base;
    }
    protected int first() { //overriding
        cur = b;
        return cur;
    }

    protected int nextValue() {
//next (overriding)
        cur = cur * b;
        return cur;
    }

}

```

Another class `FibonacciProgression` **extends** `Progression` **where** `first()` **and** `nextValue()` **have their own definitions.** These classes and a test class are in a directory 'progressions' in the public directory for the course.