

6b

More on Classes

1 toString

Java allows us to supply an object wherever a string is expected. The run-time system will automatically apply a conversion function which creates a string representing the state of the object. For example, the statement `System.out.print(new Point())` is legal (assuming class `Point` has been defined).

However, the string supplied by the automatic conversion function is neither elegant nor informative. Java allows us to provide a tailored conversion routine for each class, if we wish, as a public dynamic string-valued function called `toString()` defined in the class. This is illustrated below:

```
class Point {
    private int x; private int y;

    Point(int x0, int y0) {
        x = x0; y = y0;
    }

    public String toString() {
        return "(" + x + "," + y + ")";
    }
}

class ToStringTest {
    public static void main(String[] args) {
        Point p = new Point(3,4);
        System.out.println("My point is " + p);
    }
}
```

```
    }  
}
```

When the `println()` statement is executed, a string representing the object referenced by `p` is computed by invoking `p.toString()` implicitly. Hence the following message appears:

```
My point is (3,4)
```

The only requirement on the definition of `toString()` is that it return a string and be explicitly marked `public`. Note that `toString()` must include `public` explicitly in its declaration. Remember also that `toString` is a function, not a procedure (so the string representation of the object is not printed inside `toString`, but computed and returned).

2 equals for objects

Testing for equality of classes using `==` is just testing for equality of their references, which is almost certainly *not* what you want. Consider the following program:

```
class Point {  
    double x, y;  
  
    Point(double xval, double yval) {  
        x = xval; y = yval;  
    }  
}  
  
class Equality {  
  
    public static void main(String args[] ) {  
        Point p = new Point(3,4);  
        Point q = new Point(3,4);  
        if (p==q) System.out.println("They're equal!");  
        else System.out.println("They're not equal!");  
    }  
}
```

This produces the somewhat surprising output `They're not equal`. Java evaluates `p==q` in the usual way: by comparing the contents of `p` with the contents of `q`, and hence it compares *references*, not objects! Even if you write the equality test as

```
if (p.equals(q)) System.out.println("They're equal!");
```

the program will still not behave as expected because the default equality testing for objects is also based on references rather than contents. If you need to test for equality among objects (of a given class), include a boolean-valued method in the class definition. It is usual to call the method "equals".

```
class Point {
    double x, y;

    Point(double xval, double yval) {
        x = xval; y = yval;
    }

    boolean equals(Point p) {
        // Points are equal if their respective components are equal
        return (p!=null && x == p.x && y == p.y);
    }
}

class Equality2 {

    public static void main(String args[] ) {
        Point p = new Point(3,4);
        Point q = new Point(3,4);
        if (p.equals(q)) System.out.println("They're equal!");
        else System.out.println("They're not equal!");
    }
}
```

When the above program is run, `They're equal!` will be output, as we would wish. We will see a more sophisticated treatment of `equals` later, which has the added property of being compatible with `equals` as it is used in the Java library..

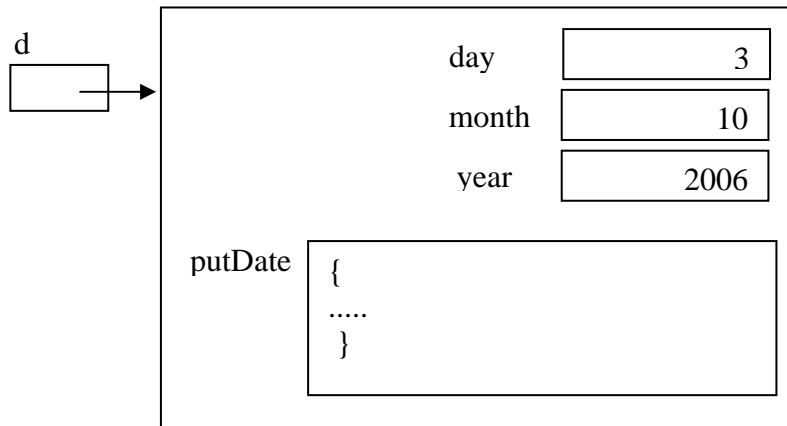
3 Mixing static and instance methods

It is common for classes to include both instance and static methods. A method should be static if it has no good reason to access instance variables or instance methods in the class. This is illustrated in the following class for dates. The class includes an instance `putDate` method which prints the date in a form typified by `03-10-06` (day followed by month followed by year, each occupying two digits).

```
class Date {  
  
    private int day; private int month; private int year;  
  
    Date(int d, int m, int y) {  
        day = d; month = m; year = y;  
    }  
  
    private static String dig2(int n) { // n as 2-digit string, 0<=n<100  
        return("" + n/10 + n%10);  
    }  
  
    void putDate() {  
        System.out.println(dig2(day) + "-" + dig2(month) + "-" + dig2(year%100));  
    }  
  
}
```

`putDate` uses *static* method `dig2` to compute a 2-character string representation of the rightmost digits of its argument. `dig2` makes no reference to any instance variables in the class and so it would be both misleading and computationally expensive to write it as an instance method, although it would be legal Java.

Remember that static methods exist as soon as the program is loaded – they are not included in objects. For example, an execution of `Date d = new Date(3,10,2006)` creates:



– note that the object does *not* contain `dig2()`.

A static method may be invoked by an instance method. When both are defined in the same class, they are invoked by using just the simple name of the static method (as in the example above). If the static method is defined in a different class, then the name of the static method must be prefixed with the name of the class in which it is defined, as in `Character.isDigit(c)`.

Static methods may call instance methods. They must *always* identify the object in which the instance method resides using the usual prefix notation.

4 Static variables

Variables declared directly in a class can be marked as `static`. Static variables have the property that *they are created just once* when the program is loaded and live as long as the program. They do not live in objects, but in the program as a whole. For example, consider the following simple class for bank accounts;

```
class BankAcct {

    private String name;    // name of account holder
    private int balance;    // amount in account (cents)
    private int acctNum;    // unique account number

    private static nextFreeNum = 1;

    BankAcct (String name0, int balance0) {
```

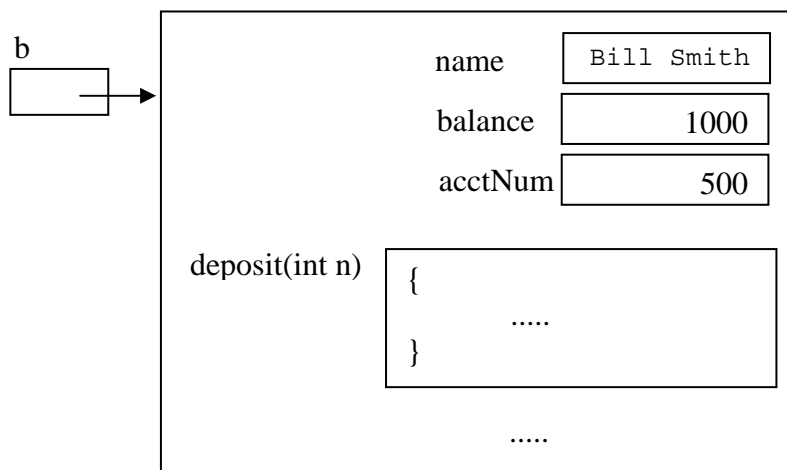
```

        name = name0; balance = balance0;
        acctNum = nextFreeNum; nextFreeNum++;
    }

    void deposit (int n){
        ...
    }
    ...
}

```

Note that variable `nextFreeNum` is static, and that it must be so (what would happen if it wasn't?). `nextFreeNum` is a variable of the program; there isn't a version of it in every object. For example, an execution of `BankAcct b = new BankAcct("Bill Smith",1000)` creates (assuming this is the 500th time that an account has been created):



– note there is no occurrence of `nextFreeNum` in the created object.

(Technical aside: Variables of `String` type contain references to strings rather than a string itself. However, it is a consequence of the fact that strings are immutable (i.e. they cannot be changed once created) that we may picture string variables as containing the actual string. Hence, the representation of “Bill Smith” above.)

All the methods in a class (whether instance or static) can refer freely to the static variables of the class. A static variable (that is not marked as `private`) can be accessed by methods in other classes, but only by using its full name. The full name of a static variable `x` defined in class `MyClass` is `MyClass.x` (note the period).

Note that variables declared inside a method cannot be marked `static` – variables can only be marked `static` when declared directly in a class definition.

5 Testing classes

When you have designed a solution as a collection of classes, it is wise to test each class individually. A good order in which to test the classes is “bottom-up”. This means that you first test those classes at the bottom of the hierarchy, i.e. those which rely on no other classes (in simple programs, all the classes will be of this kind). Then test the classes at the next level up, i.e. those classes which rely only on those you have already tested. And so on. For example, for the dating program above, we would test class `Date` before class `Person`, because `Person` makes use of `Date`. In this way, if you discover an error you can be sure it is the class you are currently testing. Fix any errors as you meet them.

To test a class, run it against some well-chosen test data to confirm the correct behaviour of all the constituent methods. The best way to organise this is to include a `main()` method in the class which exercises all the methods. Consider as an example, class `Name` above. A professional programmer would actually include a `main()` in the class to test the class methods, as follows:

```
class Name { // The name of a person

    private String forename, surname; // first and second names

    void get() { .... }

    void put() { .... }

    boolean lt(Name s) { .... }

    public static void main(String args[]) { // test routine
        Name p, q;
        p = new Name();
        System.out.print("Type a name e.g. Bill Smith ");
        p.get(); // tests get(). Key in "Bill Smith"
        p.put(); // tests put()
        q = new Name();
```

```
    System.out.print("Type a name e.g. Bill Smith ");
    q.get(); // for this, key in "Claude van Damme"
    q.put();
    System.out.println(p.lt(q)); // expect "true" to be displayed
}
}
```

The class can then be tested by compiling it and running it from the command line (just type `java Name`). The system will run `main()` from `Name`, which executes the test routine. When testing is successfully done, there is no need to remove `main()` -- let it sit there in case you change the class and want to test it again.

In general, a Java file will consist of lots of classes, many of which will contain a `main()` method that we wrote for testing purposes. When we run the program for real we type `java MyProg` at the keyboard where `MyProg` is the name of the class containing the real main program -- all the `main()`'s in the other classes sit there unused. However, if we later revise one of the constituent classes (`MyClass`, say) we will test it by invoking `java MyClass`. Once we're satisfied with the changes, we will again run the main program by typing `java MyProg`.