Almost all of you have taken COMP 202 or equivalent, so I am assuming that you are familiar with the basic techniques and definitions of Java covered in that course. Those of you who have not taken a COMP 202 or equivalent should spend some time in the first week or two familiarizing yourself with the differences between Java and the programming language that you did learn (probably C and/or Fortran). I would strongly advise you either to buy/borrow an "Intro to Java" book and read as much of it as you can. There are dozens in the Schulich library, in particular, the two mentioned in the course outline are very good. There are also free Java books on the web. The course web page has a link to one (which I have not read, but which some students tell me is good).

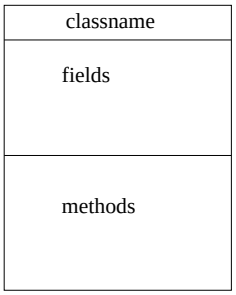
Here I am briefly reviewing some of the basic vocabulary that we will use when talking about Java programs and object oriented programming in general. The purpose of this review is mainly to refresh your memory, in particular, to highlight those Java-specific terms and concepts that will be important in the weeks ahead. **I will concentrate my review on object oriented terms, which those of you who know only C or Fortran will *not* be familiar with and will find rather strange at first. You need to become familiar with many of these terms in the next two weeks, in particular, when you start the programming assignments. So get cracking!**

There are various versions of COMP 202 so there may be a few terms below that weren't covered in the course you took. No worries. You can catch up on these things when the time comes.

Vocabulary

Classes and objects

A *class* consists of a *header* and a *body*. The header consists of *modifiers* and an *identifier* i.e. name. The body consists of *fields* and *methods*. Some classes contain *constructor* methods. Other classes contain a *main()* method. (It is possible for a class to have both constructors and a main method, though this is atypical.)



Here are some terms we use and rough definitions

- 'object' - loosely, a set of data and methods that operate on these data
- 'method' - a named sequence of instructions, an action performed by an object (my apologies for the circularity of the above two definitions)
- 'class' - a description of a set of objects

- 'to invoke' - to call e.g. "the object o invoked (its) method m"
- 'instance' - a particular object; its methods and data are specified by its class
- 'to instantiate' = to create/construct a new object (of some class)
- 'to implement a class' - to specify how objects of a class are instantiated, and what the objects do. Note: a person (you) implements a class. The computer does not do this.

One often says that an object 'belongs' to a class, and so you might be tempted to think of a class as a set of objects (in the mathematical sense). This is *not* quite correct, however. A class exists without any program running, whereas objects exist only when a program is running.

Variables

- 'primitive type' = 'base type' - there are eight in Java (int, short, long, float, double, byte, char, boolean). A variable of this type holds a data item, either a number or character
- 'reference type' - not a primitive type, rather the reference is to an object.
- 'reference variable' - a variable that contains the address of an object
- 'array' - a special kind of class, it isn't named by an identifier. Arrays have methods such as clone(), length().
- 'alias' - two reference variables that contain the same address i.e. they reference (point to) the same object
- 'static field' or 'static variable' - defined once for each class, rather than for each object of that class e.g. useful for keeping track of the number of objects of that class
- 'instance variable' - defined for each object. Most variables are instance variables. (They don't need a modifier specifying they are not static.)
- 'access modifier' = 'visibility modifier' (for a class, method, field), specifies where/by whom it can be used. There are four in Java, public, private, protected, package – in this course, you will be mostly using public vs. private. I will say more about them below.
- 'use modifier' - there are three in Java: static, final, abstract. (We will discuss the 'abstract' modifier later in the course.)
- 'final' modifier for a variable - a variable that cannot be changed. (Whether it is final is independent of whether or not it is static, e.g. you could have different final values for variables of different objects. If the variable is a primitive type, then we call it a 'constant'. If its a reference type (including an array), then it always references (points to) the same object.

Methods

- 'void method' - a method that does not return a value e.g. setter. Also called a 'procedure'.
- 'value method' - a method that returns a value e.g. getter. Also called a 'function'.
- 'accessor method' = 'query method' (sometimes called a "getter")
- 'mutator method' - a method that changes the value of some field
- 'method definition' - header + body
- 'method header' : access-modifier use-modifier return-type method-name(parameter list)
- 'signature' - of a method, name + parameters (number and type)
- 'method body' - sequence of Java statements enclosed in parentheses
- 'formal parameter' - (type, name) pair
- 'argument (of a method)' - also known as the 'actual parameter' - an object whose address is passed to the method, or a variable whose value is passed to a method
- 'local variable' - of a method, different from a parameter but behaves similarly
- 'static method' - used if a method is naturally associated with a class but not naturally associated with an object. You don't need to specify the object. Commonly used static methods are those of the Math class (in package java.lang – see the Java API link from the course web page).
- 'main()' is a static method. It is always defined:
  `public static void main(String[] args) ....`
- 'constructor' - a method that instantiates an object (of some class). Definition is different from other methods. Header has a different format: no return-type is used. The method name is the same as the class name. Constructors cannot be final or static (since they are associated with an object, i.e. 'new' creates the object and the constructor initializes certain fields). And there is no return statement.
- 'new' - an operator that invokes a constructor
- 'this' - the name of the invoking object, (typically used in a method, i.e. a method is invoked by its object). Often used in mutator methods to distinguish parameter of a setter from the field to be set

Next I briefly review several Java modifiers that you either have seen or will see soon. These are the use modifiers `final` and `static`, and the visibility modifiers `public`, `private`, `protected` and `package`. If you have taken COMP 202 then you should have been exposed to some of these already. You defined classes which had *private* fields, so that the only way for another class to access these fields (either by reading the value or writing a new value) was to have getter and setter methods

which were *public*. This distinction of private and public allows you to restrict access to within a class or between classes. There are other levels of access, though, and so I would like to mention these, just to familiarize you. (You will see these when you use Eclipse and you may wonder what they are.)

Package

In order to define these other levels of access, we need to know what a *package* is. A package is a set of classes. You should go to the Java API

`http://java.sun.com/j2se/1.5.0/docs/api/`.

On the upper left corner of that web page is a listing of dozens of packages. A few of them that you are familiar with are:

- `java.lang - Object, String, Math, ...`
- `java.util` - has many of the data structure classes that we will use in this course.

The full name of a class is the name of the package following by the name of the class, for example `java.lang.Object`. You can have packages within packages e.g. `java.lang` is a package within the `java` package. The package and class structures correspond exactly to how the class files are organized in the file system. e.g. you have directors (or folders) and subdirectories (or subfolders) and finally the class files themselves.

To identify a class as being part of a package, you start the class with the package name. The first line of your class file will be
`package PackageName`
If you want to use a package written by someone else – e.g. if you are writing a class that extends a class from that package – you start your class with a statement
`import packagename`
This doesn't actually copy the code into your file. Rather, it just says that you will be using files from that package so when a class is referenced (e.g. instantiated), the Java compiler knows where to look.

Visibility modifiers

Let's next consider modifiers for the variables and methods in a class. In order of decreasing visibility, they are:

- `public` - can be seen from every package
- `package` - can be seen from any class within the same package. This is the default modifier – if you don't write a modifier than the Java compiler assumes you mean `package`
- `private` - can be seen only from within that class

There is also a modifier called `protected` but you need to know about class inheritance to understand what it is. (We will cover inheritance later in the course.)

Use modifiers

`final`

The `final` modifier can mean several different things. For now, you will only use it to define constants i.e. variables whose values are defined once and cannot be changed. e.g.

```
final double PI = 3.1415
```

Note that you need to assign a value for this to make sense. To understand other uses, you need to know about inheritance.

`static`

The modifier `static` specifies that a variable or method is associated with the class, not with particular instances (objects). It is used, for example, to keep track of the number of instances of a class, or the number of times that a method is invoked.

```
static  int  numberOfObjects;              //  A constructor would increment
static  int  getNumberOfObjects(){         //  this variable.
    return numberOfObjects;
}
```

Another example of a `static` method is `main()`.

You should think of a `static` method or field as being in the class definition, not in particular objects. Thus, a static method cannot contain statements that refer to instance fields or that invoke instance methods. The reason is that these instance fields and methods belong to particular instances of the class.

Note that you can have an instance method that invokes a static method or references a static field in a class. You can also have a static method that references an instance variable.

You do this all the time when you use methods such as `sin()` or `exp` in the `java.lang.Math` class. Such methods are `static` (and `final`). Note that to use such methods, in your class definition you would specify that you are using the package `java.lang` and then you would invoke the method just by stating the classes name and the method e.g. `Math.cos( 0.5 )`. Note that the class name is used instead of a reference variable (to an object) which is why we say that `static` methods are "class methods" rather than "instance methods".