The vast majority of you have taken COMP 202, so I will assume 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 an equivalent intro to Java course should have spent week 1 reading chapter 1 of the course textbook and should have read the slides of COMP 202. I would strongly advise you to go further either buy or borrow an "Intro to Java" book (there are dozens in the Schulich library) and to read as much as could can about Java in the next few weeks to bring yourself up to speed. You can also read Chapters 5-10 in *The Little Book on Java* – see the PDF on the course web page.

Today I begin by 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 our memories, in particular, to highlight those terms and concepts that will be important in the weeks ahead. Not all of what you learned about Java in COMP 202 will be central in COMP 250. I will try today to discuss only the parts that are central.

I begin by reviewing an example of a class which is similar to the one given in the Ch.1 of the textbook [see GT, p. 43, 44]. See my code in the files:

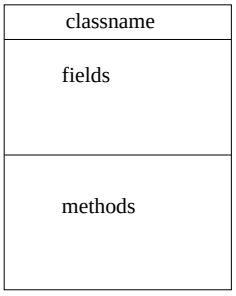
`http://www.cim.mcgill.ca/~langer/250/EXAMPLES/CreditCard.java`
`http://www.cim.mcgill.ca/~langer/250/EXAMPLES/TestCreditCard.java`

I spent 10 minutes or so walking through this example. I will not repeat the entire discussion here. Instead I will try to highlight some of the vocabulary used, which was covered in 202 and which you will need in the next few lectures.

## Vocabulary

### Classes and objects

A *class* consists of a *header* and a *body*. The header consists of various *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.)




- 'class' - a description of a set of objects
- 'method' - a named sequence of instructions, an action performed by an object
- 'object' - a program that contains data and can perform certain actions

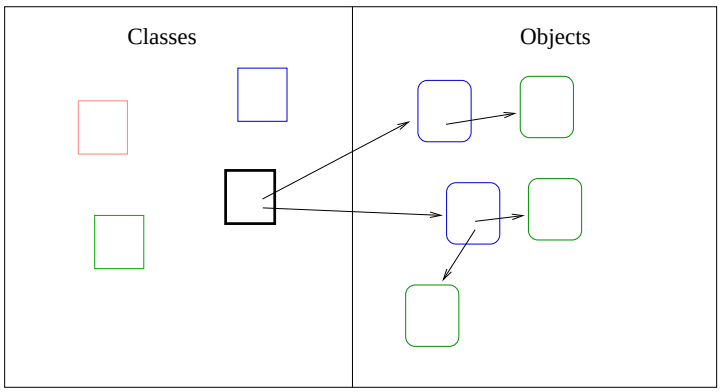
- 'to invoke' - to call e.g. "the object o invoked (its) method m"
- 'instance' - a particular object whose methods and data are specified by its class
- 'to instantiate' = to create a new object (of some class)
- 'to implement a class' - to specify how objects of a class are instantiated, and what the 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. This is *not* correct, however. A class exists without any program running, whereas objects exist only when a program is running.

Perhaps a better way to begin thinking about objects and classes is illustrated in the figure below. On the left we have our classes. To run a program, you specify some class that has a main() method. The statements in the main() method are executed, and this causes objects to be instantiated. When an object is instantiated, it appears on the right side. Each object has its own fields and methods.

The figure has arrows which represent the values of reference variables, namely the addresses of objects. These variables can belong to the class (in which case they are called static) or they can belong to the object (in which case they are not static).

The black class on the left is the class containing the method main() i.e. the program. (The figure should be viewed in color.) The three other classes are red, green, and blue. The blue and green classes are both instantiated more than once i.e. see objects on the right side. The red class is not instantiated.




Let's now look at several vocabulary terms, some of which came up in the Credit Card example, some not. These are terms that you have seen in COMP 202 and which I would like you to understand. Some of these were discussed in class.

**Variables**

- 'primitive type' = 'base type' - there are eight in Java (int, short, long, float, double, byte, char, boolean )
- '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. 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. (Don't need a modifier specifying it is non-static.)
- 'access modifier' = 'visibility modifier' (for a class, method, field), specifies where/by whom it can be used (there are 4 in Java, public, private, protected, package – in this course, you will be mostly using public vs. private)
- 'use modifier' - there are three in Java: static, final, abstract (more on this 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'
- '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

```
public class Student{
      :
     String   studentName;
      :

     public void setStudent(String studentName) {
        this.studentName = studentName;
     }
}
```

This completes our brief review of the COMP 202 material which I assume you are familiar with. Next class, we will begin some new material on object oriented design (in Java).