Main concepts from Chapter 1:

- Objects
  - an Object:
    1. contains information (data)
    2. performs functions (operations)
  - Objects have *types*. The type usually gives us some idea as to what the object can be used for.
  - Objects can be told to perform their operations by sending them *messages*.
  - The textbook shows a way to represent objects graphically (p. 17).

- Classes
  - Classes define a type of Object. Many objects of the same type can be created (instantiated) from a single class. (Think of the class as being the cookie cutter and the objects as the cookies.)
  - A class can also contain data and perform operations. Because a class defines the data and operations to be used by its objects, we will have to have some way to distinguish between data and operations used by the class, and data and operations used by the objects.
  - The textbook shows a way to represent classes graphically (p. 18).

- Messages
  - We can send a message to a specific object. The object must have a *method* that matches the given message. If it does, the method is executed (the operation is performed); if not, an error occurs.
  - Messages can contain extra information the the method may require in order to do its job. The extra pieces of information send along with a message are called *arguments*.
  - Methods can send information back to whoever sent the message (*e.g.* the result of a calculation or something to say if the operation worked or not). Only a single piece of information can be sent back to the sender of the message, and this information is called the *return value*.

Example 1: `NumberGuesser` and `NumberHolder`

- define `NumberGuesser` class by describing what type of data, and what each method (operation) can be performed:
  - contains no data
  - contains a single method. (Note a constructor wasn't needed because there was no data to initialize.):
    1. `guess`. When a `NumberGuesser` object receives the message labeled `guess`, it must think of some random number between 0 and 100, and ask some `NumberHolder` object if that number is the one it is holding. Since `NumberGuesser` and `NumberHolder` objects are not actually connected to each other, the `NumberGuesser` object must be told which `NumberHolder` to ask. After finding out if its number was correct, the `NumberGuesser` object tells whoever sent the `guess` message the result.

- define `NumberHolder` class by describing what type of data, and what each method (operation) can be performed:
  - contains one piece of data:
    1. an integer between 0 and 100 (inclusive)
  - contains two methods:
    1. the constructor. When a `NumberHolder` object is created, it must think of a number between 0 and 100 and save that number as its piece of data.
    2. `isIt`. When the appropriate message is received, it will be accompanied by an argument which will be an integer between 0 and 100. This method simply compares the argument integer to the data contained in the object itself. It then tells the message sender the result of the comparison.

- Still need a `Program`:
  - contains a single method, `main`. This method creates the `NumberHolder` and `NumberGuesser` objects from their respective classes (using the special message `new`), and then makes the `NumberGuesser`s repeatedly guess at the `NumberHolder`s' numbers.

- Inheritance
  - Classes can "inherit" information from other classes. This is used when one class is a special type of another class. For example, a `SmallNumberHolder` might be a class that behaves exactly the same way as the `NumberHolder` class, except that its objects store integers between 0 and 10 instead of 0 and 100. We could then make `SmallNumberHolder` a *subclass* of `NumberHolder`. (Note that `SmallNumberHolder` is a *type of* `NumberHolder`, not the other way around. This means `NumberHolder` is the *superclass* and `SmallNumberHolder` is the *subclass*.)
  - A class can inherit from at most a single class (that is, it can only have one "parent", or superclass.
  - A class can have many different types of "children", or subclasses.
  - An *inheritance hierarchy* can be drawn showing the relationship

between many classes. See page 29 of the textbook for an example.

– We won't be using class inhertance during this course, except possibly near the end if there is time. You should understand the very basic ideas, but don't worry about having to use it in your programs.

- Life Cycle of Software
  - Five basic stages:
    1. *analysis*: deciding on specifications.
    2. *design*: deciding on what classes and objects will be needed. We focus on the data and methods required, not how the methods will be performed.
    3. *coding*: the classes are written up on the computer using our chosen programming language.
    4. *testing*: the code is verified to work by running the programs.
    5. *operation*: the code is put into use.
  - All the steps except 3 don't even require knowing what language you're programming in. Programming is only one part creating software! In this course you should spend as much time on learning how to design good object-oriented programs as you do learning how to write java code.