CPS122 Lecture: Defining a Class

last revised January 14, 2016

Objectives:

1. To introduce structure of a Java class
2. To introduce the different kinds of Java variables (instance, class, parameter, local)

Materials:

1. Features that can occur in a class handout
2. BlueJ BankAccount example to project
3. BankAccount example as a Netbeans project to demonstrate javadoc generation
4. Naming conventions / types of variables exercise

I. **Objects and Classes in Java**

A. Review from first class

1. Recall that a program in an object-oriented language such as Java collection of interacting **objects**.

   An object is a software entity that typically models something in the real world and has three critical properties.

   a) **State** - the object encapsulates some information about itself, which are sometimes called its **attributes.**

   b) **Behavior** - the object can do some things on behalf of other objects

   c) **Identity** - the object is distinguishable from all other objects of the same general type.

   (The definition of an object in terms of state, behavior, and identity is a "3 AM phone call fact.")

2. Each object in an object-oriented system belongs to some **class**.

   a) A class can be thought of as the set of all objects of the same kind - e.g. in the example we looked at in the first class, all the savings account objects together constitute the SavingsAccount class.

   b) The basic unit of an OO program is the <u>class</u>. In Java, everything must appear inside a class definition - there are no stand-alone statements or functions.

3. When we write OO programs in a language like Java, we define classes, and then use them as templates for constructing objects. Each individual object is called an instance of its class, and each object is an instance of exactly one class.

   Example: Recall the SavingsAccount example we looked at in the first lecture

   a) OPEN IN BLUEJ - Note two classes

   b) We will now take a look at the code for one of these classes: SavingsAccount

      HANDOUT: Features that Can Occur in a Class

B. Each file should begin with a *file/class comment* - a comment (in English) that describes the purpose of the class and specifies its author and date.

   NOTE in handout

   Generally each class is in its own file. In Java, the name of the file will be the same as the name of the class - e.g. the example would appear in a file called `SavingsAccount.java`. (This is a requirement enforced by the Java compiler).

   There is the possibility of defining a class inside another class - but we won't discuss that now..

C. Each class has a name. The name should be clearly related to the purpose of the class.

   Naming convention: mixed case <u>beginning with capital letter</u>

D. The definition of a class is delimited by braces. (These are always required).

E. A class generally has *instance variables*.

   1. *Each object* that is an instance of a given class has *its own copy* of each instance variable. These variables exist as long as the object exists.

   2. In general, each piece of information that an object needs to know will be represented by an an instance variable.

      NOTE instance variables `accountNumber` and `currentBalance` in handout.

      Insofar as possible, the name of an instance variable should make its purpose crystal clear. In some cases, a supplementary comment may be desirable to add information not obvious from the name. (e.g. the comment on `currentBalance` that notes how it is represented.)

   3. Further, each object will also need an instance variable to refer to each kind of other object that it knows. (I.e. the instance variables hold answers to two questions: what does this object know? and who does this object know?)

      There are three possibilities for these instance variables

      a) An instance variable may be of a primitive type - thus representing a simple value:

         Examples in handout: `accountNumber`, `currentBalance`

b) An instance variable may simply be a reference to some other object.

   Example in handout: `owner`

c) Or, an instance variable may be a special kind of object called a *collection* which holds references to several other objects of the same kind.

   EXAMPLE: Since bank accounts can have multiple owners (joint accounts), instead of an instance variable called owner, we might have one called the owner<u>s</u> that is a collection. (Collections are a topic we will discuss later this semester - so far now we don't know how to do this in Java, so we'll stick with a single owner for this example.)

4. Conceptually, we decide on the instance variables for a class early in the process. In this example, they are placed at the beginning of the class. However, a case can also be made for listing the instance variables near the end of the class declaration. Either place is acceptable (but not strung out everywhere!)

F. A class generally has one or more instance methods (mutators and accessors)

1. In general, each thing an object needs to be able to do will be represented by an instance method.

   NOTE in handout

2. Methods are of two general types

a) A mutator changes the state of the object.

   Examples in handout: `deposit(), withdraw(), calculateInterest()`

b) An accessor furnishes information about the state of the object

   Examples in handout: `reportBalance()`, `getAccountNumber()`

3. A method may or may not have one or more parameters, which represent information flowing into the method from its caller. A parameter is specified in the method header and - like all variables in Java - is declared to be of a particular type.

   Examples in handout: `deposit()` and `withdraw()` each have a single int parameter specifying the amount of money to be deposited or withdrawn.

   `calculateInterest()`, however, does not need a parameter

   Not shown in the handout is the possibility that a method might have more than one parameter. In fact, there is no limit on the number of parameters a method may have.

4. A method may or may not have a single return value, which represents information returned to the caller when the method is used. The type of the return value is specified before the method name; if there is no return value, `void` is used.

   Examples in handout: `deposit(), withdraw(), setAnnualInterestRate(), calculateInterest()` do not have return values, but `reportBalance()` and `getAccountNumber()` do. The return value is a `String` in the case of `reportBalance()`, `int` in the case of `getAccountNumber()`.

5. Can you see any relationship between the type of the return value and whether the method is an accessor or mutator?

   ASK

6. The body of a method is always delimited by braces, even if it is just one line.

   Compare this to Python, where the body of a function was delimited by indentation. In Java, we use the same indentation conventions, but the compiler pays attention to the braces, not our indentation.

7. An instance method is called by giving the object to which it is to be applied and the name of the method (together with its actual parameters), separated by a dot.

   Example: The constructor of `SavingsAccount` invokes the `addAccount()` method of its owner.

8. Naming convention: mixed case <u>beginning with a lowercase letter,</u> always followed by a pair of parentheses

G. A class generally has one or more constructors. A constructor is similar to a method, but is used to initially create an object of the class. It's purpose is to put the instance variables in a correct initial state.

   1. Two things distinguish a constructor from methods:

      a) The name of a constructor is <u>always</u> the same as the name of the class - hence always begins with an uppercase letter.

      b) A constructor <u>never</u> has a return type - and specifying a return type (even `void)` is neither required nor allowed.

         Example: Show in handout

   2. However, a constructor may - and generally does - have parameters which specify the information needed to initially create an object of the class.

Example: the `SavingsAccount()` constructor has the `Customer` who owns the account as a parameter. It does not need a parameter for the account number, which the constructor itself assigns, nor for the initial balance, because this is always zero;

H. A class may also have class variables, constants, or methods .

1. A variable that holds a value that is shared by all the objects in the class can be declared as a class variable.

   Examples in handout: `nextAccountNumber`, `annualInterestRate`

   One is used to ensure that each account gets a unique number. The constructor uses this to assign a number to a newly-created account, and then adds one to it. Obviously, this must be shared by all instances of the class.

   Another is used to hold the interest rate, which is (in this case at least) is the same for all instances of the class.

2. A constant value that is the same for all objects in the class, and that either methods or clients of the class need to make use of can be declared as a class constant.

   Example in handout: many banks have a minimum balance allowed in account for paying interest. (An account with a balance below this amount is not eligible for interest.) Since this rule applies to all accounts, a class constant `MINIMUM_BALANCE_FOR_INTEREST` is used. (Constants use a special naming convention as here)

   (One might argue that this should also be a variable, so that the bank can change its rules. I made it a constant here so that I could illustrate both possibilities!)

3. A method that represents a responsibility of the entire class, not associated with any particular object in the class, can be declared as a class method.

4.
   a) NOTE in handout: `setInterestRate()` method.

   b) A class method is called by giving the name of the class of which it is a part and the name of the method (together with its actual parameters), separated by a dot.

      Example: to change the interest rate for all SavingsAccounts to 3%, one would write

      `SavingsAccount.setAnnualInterestRate(0.03);`

5. Instance methods can freely make use of class constants, variables, and methods; but that class methods *cannot* use instance variables or methods since they are not associated with any particular instance of the class.

6. Look at the handout: how are class variables, constants, and methods distinguished from instance variables and methods?

   ASK

   They are declared `static`.

I. Note the use of comments throughout the class - a file/class prologue, method prologues, and prologues for constants.

   1. In Java, the preferred way to do this is with a comment that goes immediately before the entity, begins with `/**`, and includes <u>tags</u> that begin with "`@`". (The permitted tags depend on what type of comment you are writing - e.g. for a class comment tags like "`@author`" occur; for a method comment tags like "`@param`" occur.)

2. This style of comment is called a “javadoc” comment - and gets its name from a program (called javadoc) which automatically creates a set of documentation from code containing this style of comment.

   DEMO - “Generate Javadoc” in “Run” menu of Netbeans for BankAccount project. Show result for `SavingsAccount`, walking through the javadoc relative to the handout for the code.

II. **Variables and Constants**

A. We have now seen three different kinds of variables/constants.

1. What are they?

   ASK

   a) Instance variables

   b) Class variables

      Note: instance and class variables are sometimes also called <u>fields</u>

   c) Parameters

2. There is a fourth type of variable, known as a local variable, that pertains only to a single method.

   Examples in handout: `result` and `cents` in `reportBalance()`

   a) Such variables are usable only in the method in which they are declared, and only after the point at which they are declared (so a local variable declared in the middle of a method could not be used at the beginning.)

   b) They do not retain their values - and, in fact, cease to exist at all -between calls to the method

   c) Parameters are like local variables in that they pertain only to the single method in which they are declared.

3. How do these different types of variable differ?

   ASK

   a) Each object belonging to a class has its own copy of each instance variable, that exists as long as the object exists.

      Example: any one of you can answer a question like "what is your name?" (Each of you has an "instance variable" called name)

   b) All objects belonging to a class share a single copy of each class variable.

      Example: all of you would give the same answer to the question "how many credits does it take to graduate?"

   c) Parameters and local variables exist only while a particular method is being executed.

      Example: If I were to ask you "what are you eating?" while you are sitting in Lane, you could give me an answer. But it would be silly for me to ask you that question while you are asleep, and probably offensive for me to ask you that question now!

4. Where are they declared?

   ASK

   a) Instance and class variables are declared outside any method - often at the start or end of a class.

   b) Parameters are declared in a method heading.

   c) Local variables are declared inside the body of a method.

5. The names of variables must be unique within their scope.

a) A class may not have two instance or class variables with the same name. But two different classes can have instance/class variables with the same name.

b) A method may not have two parameters or local variables with the same name. But two different methods can have parameters/local variables with the same name.

c) It <u>is</u> possible for a parameter or local variable of a method to have the same name as an instance or class variable of the class in which the method occurs.

   Example: the `owner` parameter of the `SavingsAccount` constructor has the same name as an instance variable of the `SavingsAccount` class.

d) When this is done, the local declaration creates a <u>hole in the scope</u> of the instance/class declaration.

   (1) When the name occurs by itself, it is understood as referring to the parameter/local variable.

   Example: the second and third occurrences of `owner`

   (2) It is possible to refer to an instance variable by preceding it with `this`.

   Example: the first occurrence of owner. Thus, this line says "set the `owner` instance variable of the `SavingsAccount` being constructed to be equal to the value specified by the caller through the `owner` parameter".

   (3) A construct of this form is something of an idiom in constructors - the parameter specifying the initial value for the instance variable of the same name.

   Example: Note in BlueJ demo: `Customer` constructor

6. Case convention for all types of variables: mixed case, <u>beginning with a lowercase letter.</u>

B. Any variable can be given a value by an initializer when it is declared.

1. Example: `nextAccountNumber` in the `SavingsAccount` example

2. If an instance variable does not have an initializer, then it is initialized to a default value (e.g. 0 for numbers).

3. If a local variable does not have an initializer, it cannot be used until it is given a value by an assignment statement.

C. A constant is a special kind of variable that is given a value that cannot be changed.

1. Constants are declared using the word `final`.
2. Constants, of necessity, must be given a value by an initializer.
3. Constants are declared as class constants (`static`), since it makes little sense to think of different instances of a class as each having their own value of some constant!.
4. Constants typically are given names that consist of all uppercase letters, with underscores used to separate words (since mixed case is not used.)

   Example: `MINIMUM_AMOUNT_FOR_INTEREST`.

D. It is also possible to have instance variables that are given an initial value, and then cannot be changed after that.

1. They are also declared using the word `final`.

2. Though they can be given a value by an initializer, they are typically given a value by a constructor.

3. They have a name that looks like any other variable name.

   Example: `accountNumber` - once given a value by the constructor, it cannot be changed.

III. Visibility Specifiers

A. The constructor(s), instance and class variables, instance and class methods, and constants of a class are called the members of the class. Look at the members in the handout. What common characteristic do they all exhibit?

   ASK

   Each is preceded by one of the words `private` or `public`.

   (Note: parameters and local variables are not members of a class - they are local to a particular method in a class.)

B. The words `private` and `public` are called visibility specifiers.

   1. A `private` member of a class is only visible to objects belonging to that class.

   2. A `public` member of a class is visible to objects belonging to any class.

   3. There are two other visibilities which we will meet later (they pertain to things we haven't talked about yet.)

      a) protected visibility- specified by the modifier `protected`.

      b) package private visibility - specified by the absence of any modifier. (Thus, if you forget to use one of `private` and `public` - which works correctly in many cases, but is not what you want to do!)

4. private members often do not have javadoc comments, since (by default) the javadoc program does not include private members from the documentation it generates. However, it is sometimes useful to use comments of this form anywhere for the benefit of someone reading the class.

C. Visibilities pertain to one of the key features of OO: encapsulation. A class encapsulates the details of its implementation - which means that a the details of how a class is implemented can be changed without affecting other classes that make use of it. Moreover, encapsulated variables are not subject to surprise changes.

(This is one aspect of OO that is not supported by Python)

D. For now, as a general rule:

1. Instance and class variables should be private. There will be cases where you want to use protected or package visibility, but it is hard (if not impossible) to imagine a situation where public would be appropriate.

2. Constructors, instance and class methods are often public. However, if a method is used only to help implement some other method, and is not needed outside objects of the class, it may be appropriate to make it private or one of the other visibilities.

3. Constants are often public. Again, if a method is used only to help implement some other method, and is not needed outside objects of the class, it may be appropriate to make it private or one of the other visibilities.

IV. Naming Conventions and Declaration Place

A. There are some well-established naming conventions used in Java (and other OO languages) that make it possible to tell what kind of thing a name refers to from the form of the name.

B. Most names used a mixed-case convention - all letters (save possibly the first) are lower case; but the first letter of each word is capitalized.

   Example: `thisIsTheNameOfSomething`

   Numeric digits can occur anywhere inside a name, though they cannot be the first character

   Example: `r2d2`

C. A mixed case name beginning with an upper-case letter always refers to a class or a constructor. The name of a constructor is always followed by parentheses, while the name of a class is not. (But note that the name of a constructor is always the same as the name of its class)

   Examples: `SavingsAccount` - a class
   `SavingsAccount()` - constructor for this class

D. A mixed case name beginning with a lower-case letter refers either to a variable or a method.

   1. The name of a method is always followed by parentheses, while the name of a variable is not.

      Examples: `currentBalance` - a variable
      `withdraw()` - a method

   2. In the case of variables, what kind of variable it is depends on whether it is declared `static`, and where it is declared.

a) Declared outside of any method: instance or class variable - class variable if `static`, instance variable if not.

b) Declared in the heading of a method: parameter.

c) Declared in the body of a method: local variable.

E. Constants use a name composed of all uppercase letters, with words separated by underscores

Example: `MINIMUM_AMOUNT_FOR_INTEREST`

F. Do exercise on naming conventions / declaration place