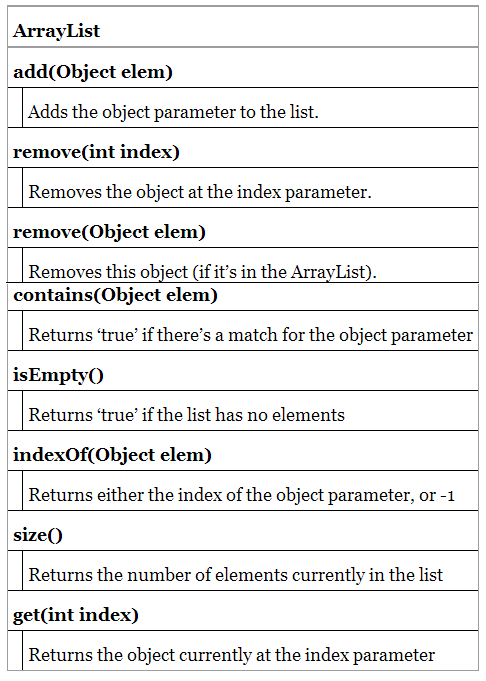
**B”H**

**Chapter 6**

* Java library is the Java API.
* ArrayList (a class in the core Java library (the API)).

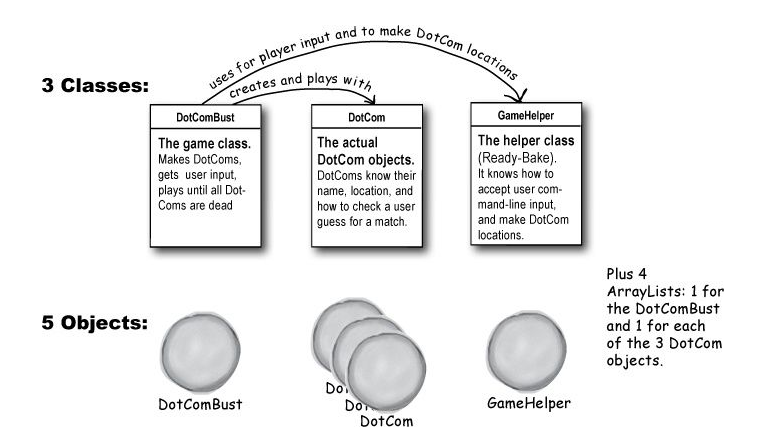
**private ArrayList<DotCom> dotComsList = new ArrayList<DotCom>();**

This is just a sample of SOME of the methods in ArrayList. Note: the add( Object elem) method actually looks a little stranger than the one we’ve shown here... we’ll get to the real one later in the book.



* An ArrayList can change size dynamically
* You can put a primtive in an ArrayList, as long as it’s wrapped in a primitive wrapper class (you’ll see a lot more on that in Chapter 10). And as of Java 5.0, that wrapping (and unwrapping when you take the primitive out again) happens automatically.
* If you’re using an ArrayList of primitives, it is probably is faster with an array, because of all the wrapping and unwrapping,

Overview of the game structure:



* See pages 142 and 143 for informative images that don’t show well I kindle version
* ‘And’ and ‘Or’ Operators ( **&&, ||** )
* Instead of becoming an expert in the arcane world of precedence, just use parentheses to make your code clear.
* Not equals ( **!= and !** )

**if (model != 2000) {**

**// do non-model 2000 stuff**

**}**

**if (!brand.equals("X")) {**

**// do non-brand X stuff**

**}**

* The operators we’ve looked at so far, **&&** and **||**, are known as short circuit operators. If the JVM sees that the left side of a && expression is false, it stops right there! Doesn’t even bother to look at the right side. Similarly, with | |, if the JVM sees that the left side is true, it declares the entire statement to be true and doesn’t bother to check the right side.
* Say you’re not sure whether a reference variable has been assigned to an object. If you try to call a method using this null reference variable you’ll get a NullPointerException. So, try this:

**if (refVar != null && refVar.isValidType() ) {**

**// do 'got a valid type' stuff**

**}**

* **&** and **|** operators act like their **&&** and **||** counterparts, except that they force the JVM to always check both sides of the expression. Typically, **&** and **|** are used in another context, for manipulating bits.

**Using the Library (the Java API)**

* In the Java API, classes are grouped into packages.
* To use a class in the API, you have to know which package the class is in.
* Every class in the Java library belongs to a package. The package has a name, like **javax.swing**. **ArrayList** is in the package called **java.util**, which holds a pile of utility classes.
* You’ve already been using classes from a package. **System** (**System.out.println**), **String**, and **Math** (**Math.random()**), all belong to the **java.lang** package.
* You have to know the full name of the class you want to use in your code. Unless the class is in the java.lang package.
* Packages are important for three main reasons.
  1. First, they help the overall organization of a project or library.
  2. Second, packages give you a name-scoping, to help prevent collisions if you and 12 other programmers in your company all decide to make a class with the same name. If you have a class named Set and someone else (including the Java API) has a class named Set, you need some way to tell the JVM which Set class you’re trying to use.
  3. Third, packages provide a level of security, because you can restrict the code you write so that only other classes in the same package can access it. (You’ll learn all about that in Chapter 17)
* What’s to prevent two people from giving a class the same package name? Java has a naming convention that usually prevents this from happening, as long as developers adhere to it. We’ll get into that in more detail in Chapter 17.
* Two import options:
  1. Put an import statement at the top of your source code file: **import java.util.ArrayList;**
  2. Type the full name everywhere in your code. Each time you use it. For example:

**java.util.ArrayList<Dog> list = new java.util.ArrayList<Dog>();**

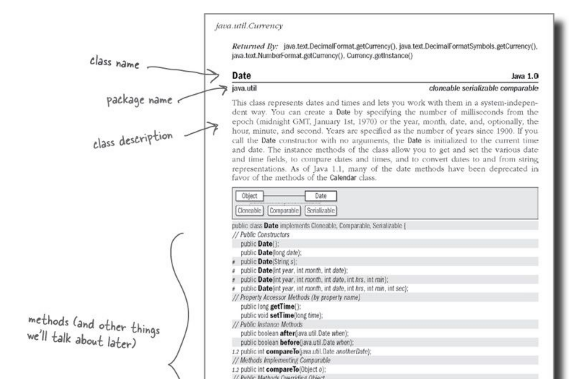
**public void go(java.util.ArrayList<Dog> list) { }**

**public java.util.ArrayList<Dog> foo() {...}**

* See page 156 on why there its **javax.swing** and not just **java.swing**
* An import statement saves you from typing. That’s really it. You don’t have to worry about your code becoming bloated, or slower, from too many imports. An import is simply the way you give Java the full name of a class.

**How to play with the API**

1. Browse a Book; such as [Java In A Nutshell, 5th Edition by David Flanagan](http://www.amazon.com/Java-Nutshell-5th-David-Flanagan/dp/0596007736/ref=la_B000APEZR4_1_10?s=books&ie=UTF8&qid=1401483672&sr=1-10). Flipping through a reference book is the best way to find out what’s in the Java library. You can easily stumble on a class that looks useful, just by browsing pages.



1. Use the HTML API docs (<http://docs.oracle.com/javase/7/docs/api/>):
   * The online docs called the Java API are part of a larger set called the Java 5 Standard Edition Documentation. You can also browse them at java.sun.com though might be slow. Trust us, you probably want these on your hard drive.
   * The API docs are the best reference for getting more details about a class and its methods. The book tells you a little about it, enough to know that this is indeed what you want to use, but you still need to know more about the methods.

