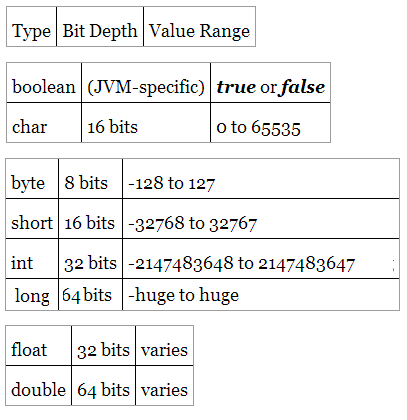
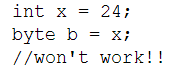
**B”H**

**Chapter 3**

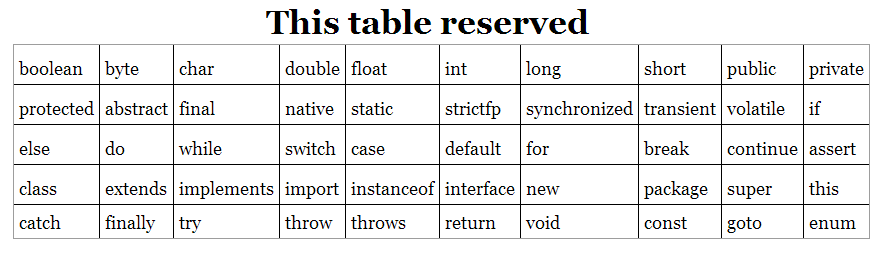
* Variables come in two flavors:
  + Primitive: hold fundamental values (think: simple bit patterns) including integers, booleans, and floating point numbers.
  + Object Reference. Hold references to objects
* Note: When you see a statement like: “an object of type X”, think of type and class as synonyms. (We’ll refine that a little more in later chapters.)



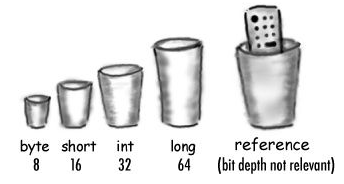




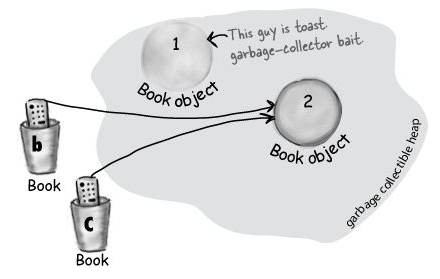
* All the compiler cares about is that you’re trying to put a big thing into a small thing, and there’s the possibility of spilling.
* Pouring a small cup into a big one? No problem. The compiler always errs on the side of safety.
* You can name a class, method, or variable according to the following rules (the real rules are slightly more flexible, but these will keep you safe):
  + It must start with a letter, underscore (\_), or dollar sign ($). You can’t start a name with a number.
  + After the first character, you can use numbers as well. Just don’t start it with a number.
  + It can be anything you like, subject to those two rules, just so long as it isn’t one of Java’s reserved words.

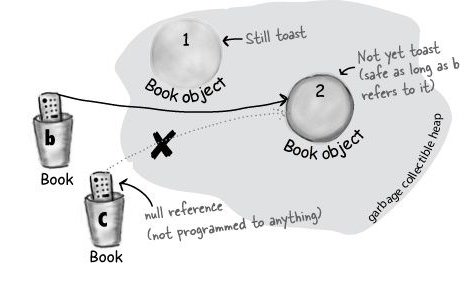


* What about objects?
  + There is actually no such thing as an object variable.
  + There’s only an object reference variable.
  + An object reference variable holds bits that represent a way to access an object.
  + It doesn’t hold the object itself, but it holds something like a pointer. Or an address. Except, in Java we don’t really know what is inside a reference variable. We do know that whatever it is, it represents one and only one object. And the JVM knows how to use the reference to get to the object.
* We often think we can stuff an object into a variable ... we say things like, “The method returns a Dog”, or, “I put a new Foo object into the variable named myFoo.” But that’s not what happens. There aren’t giant expandable cups that can grow to the size of any object. Objects live in one place and one place only — the garbage collectible heap! (You’ll learn more about that later in this chapter.)
* Think of a Dog reference variable as a Dog remote control. You use it to get the object to do something (invoke methods).

* With reference variables, the value of the variable is... bits representing a way to get to a specific object. You don’t know (or care) how any particular JVM implements object references.
* Unless marked final, then a reference variable can refer to one Dog and then refer to a different Dog later.
* A reference variable can be set to null. It’s a new universal remote control and you don’t have a TV. When value is set to null then prior linked object is eligible for garbage collection if no other object variable points to it.
* Many reference variables can refer to one object



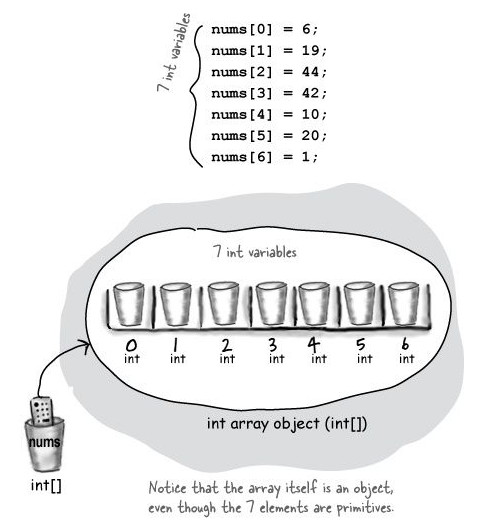


* An array is like a tray of cups. Note arrays are objects too.

**int[] nums;**

**nums = new int[7];**

* Remember, elements in an int array are just int variables.



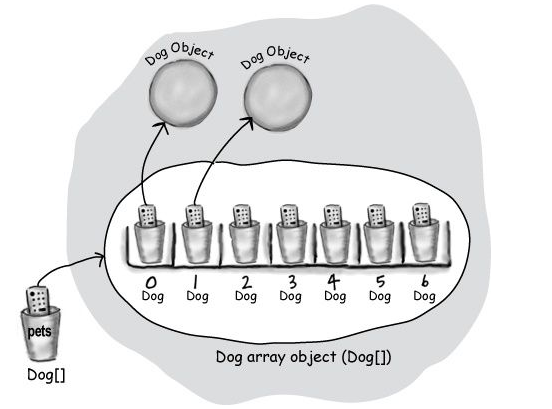
* An array is an object, even though it’s an array of primitives. Arrays are always objects, whether they’re declared to hold primitives or object references.

**Dog[] pets;**

**pets = new Dog[7];**

**pets[0] = new Dog();**

**pets[1] = new Dog();**



* When the Dog is in an array, we don’t have an actual variable name (like fido). Instead we use array notation and push the remote control button (dot operator) on an object at a particular index (position) in the array:

pets[0].name = "Fido";

pets[0].bark();

* The Java standard library includes lots of sophisticated data structures including maps, trees, and sets (see Appendix B ), but arrays are great when you just want a quick, ordered, efficient list of things. Arrays give you fast random access by letting you use an index position to get to any element in the array.
* Once you’ve declared an array, you can’t put anything in it except things that are of the declared array type. (You can, however, put a byte into an int array, because a byte will always fit into an int-sized cup.)