Websites

Java Forum: <https://javaranch.com/>

Head First Java的Java code下載地址：wickedlysmart.com

Learning habit

## Principle

A person’s brain craves novelty

e.g. unusual, interesting, dangerous

## A stimulating learning process

1. Get - and keep – the reader’s attention.
2. Make it (knowledge & skills) visual （視覺刺激）
3. Use a conversational and personalized style （陪伴）
4. Get the learner to think more deeply

* The big picture & layout
* Think（Don’t just read. Stop and think.）
* Practice（Your brain is turned to learn and remember more when you do things than when you read about things.）
* Repetition (刺激multiple senses, the content gets coded into more than one area of your brain) （執行力）

1. Touch learns’ emotions

* curiosity, surprise, fun （過程好玩）
* achievement (when a problem is solved or you have made some progress)（結果有成就感）

1. Rest

* Part of the learning (especially the transfer to long-term memory) happens after you put the book down. Your brain needs time on its own, to do more processing.（消化）
* When your brain is getting overloaded, rest. （放鬆）

Every Java application has to have at least one class, and at least one main method (not one main per class; just one main per application).

About Exercise

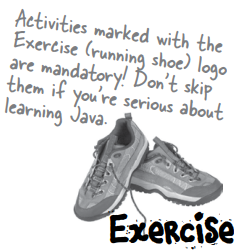
Pxxvii

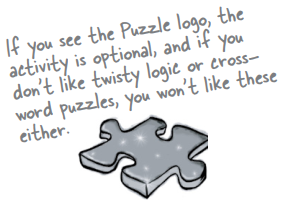
Read the “There are No Dumb Questions”

That means all of them. They are not optional side-bars – they’re part of the core content! Sometimes the questions are more useful than the answers.

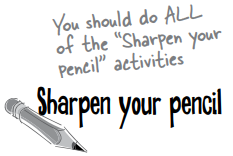
Type and run the code.

Pxxix

The end-of-chapter exercises are mandatory.

Puzzles are optional.

Answers for both are at the end of each chapter.

The ‘Sharpen Your Pencil’ exercises don’t have answers.

Not printed in the book, anyway. For some of them, there is no right answer, and for the others, part of the learning experience for the Sharpen activities is for you to decide if and when your answers are right. (Some of our suggested answers are available on wickedlysmart.com)

Terms

Java Virtual Machine (JVM)

a pair of curly braces { }

a comparison operator: < less than, > greater than, == equality

The equals operator uses two equals signs ==

The assignment operator is one equals sign =

Declare an int variable with a name and a type: int x;

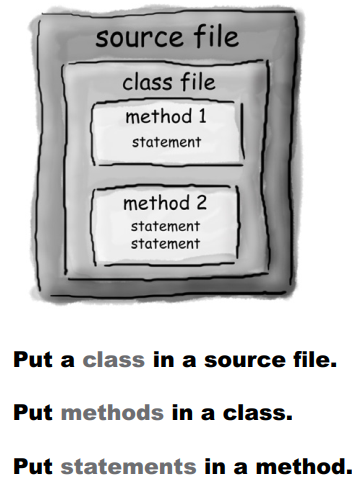
int x = 4; // assign 4 to x

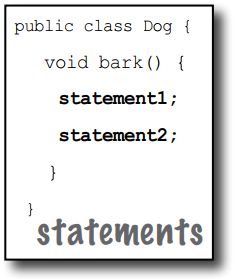
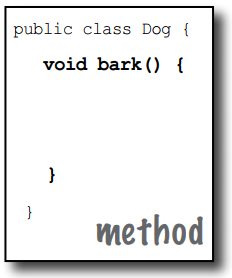
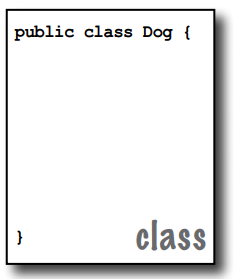
Put a boolean test inside parentheses（小括號）: while (x == 4) { }

Coding

## Chapters 1 & 2

### Java code structure





### While loops

Q: In my other language I can do a boolean test on an integer. In Java, can I say something like: int x = 1; while (x){ } Is this right?

A: No. A boolean and an integer are not compatible types in Java. Since the result of a conditional test must be a boolean, the only variable you can directly test (without using a comparison operator) is a boolean. For example, you can say: boolean isHot = true; while(isHot) { }

### System.out.print vs. System.out.println

System.out.println inserts a newline (think of println as printnewline while System.out.print keeps printing to the same line. If you want each thing you print out to be on its own line, use println. If you want everything to stick together on one line, use print.

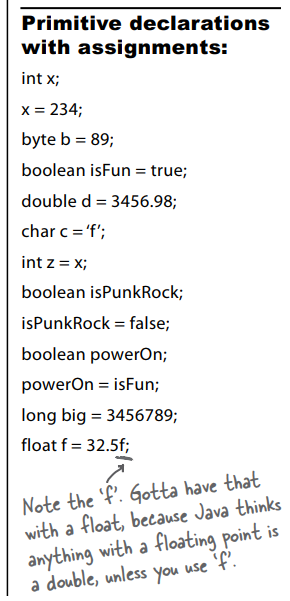
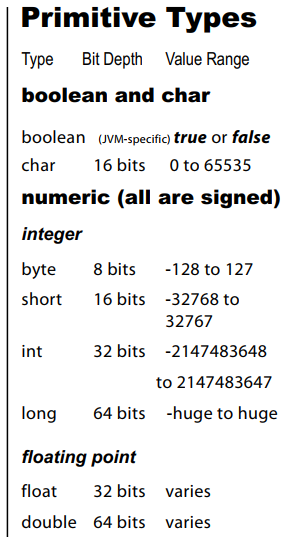
### To make a method or a variable ‘global’

marking a method as public and static makes it behave much like a ‘global’

If you mark a variable as public, static, and final – you have essentially made a globally-available constant.

## Chapter 3 Primitives and References

### Primitives



### assign a value to a variable

You can assign a value to a variable in one of several ways including:

* type a literal value after the equals sign (x=12, isGood = true, etc.)
* assign the value of one variable to another (x = y)
* use an expression combining the two (x = y + 43)

### spillage

You can’t put a large value into a small cup.

For example, you can’t pour an int-full of stuff into a byte-sized container, as follows:

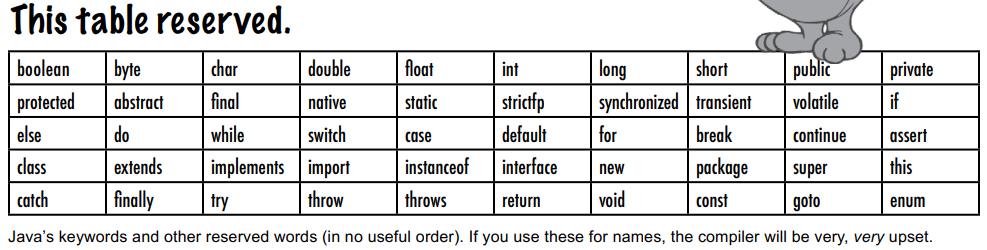
int x = 24;

byte b = x;

//won’t work!!

### names for variables

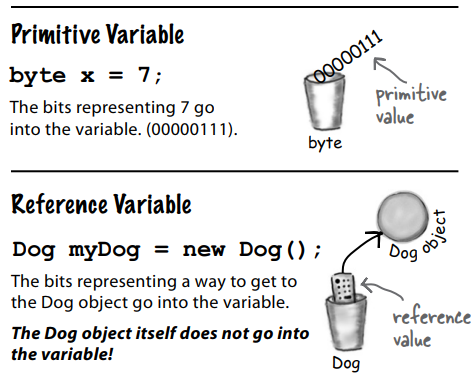
* It must start with a letter, underscore (\_), or dollar sign ($). You can’t start a name with a number.
* After the first character, you can use numbers as well. Just don’t start it with a number.
* It can be anything you like, subject to those two rules, just so long as it isn’t one of Java’s reserved words.



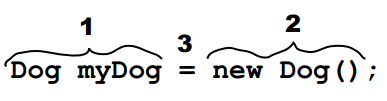
### object reference variables

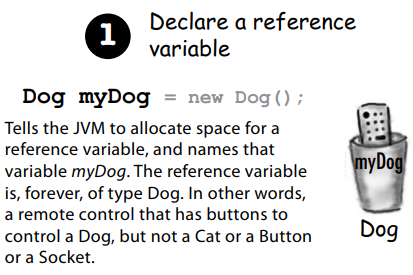
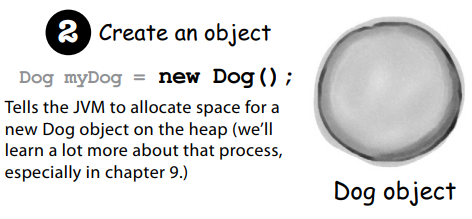
Although a primitive variable is full of bits representing the actual value of the variable, an object reference variable is full of bits representing a way to get to the object. It doesn’t hold the object itself, but it holds something like a pointer. Or an address.

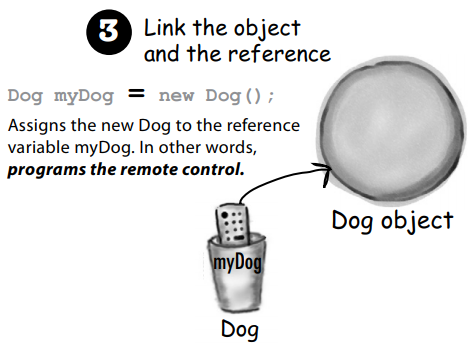
Objects live in one place and one place only—the garbage collectible heap!



### The 3 steps of object declaration, creation and assignment





### ‘re-reference’

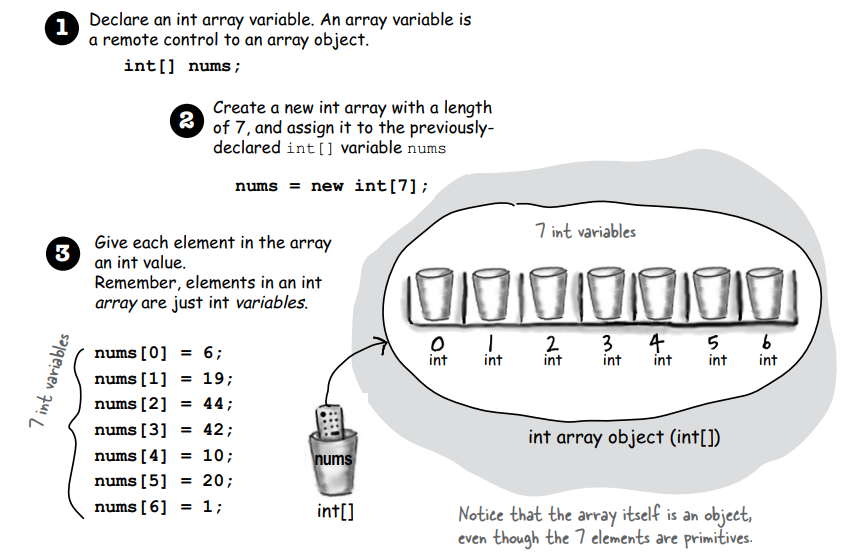
An object reference variable can be referring to one object (e.g. Dog), and then can refer to some other object (e.g. another Dog). But it cannot refer to a Cat.

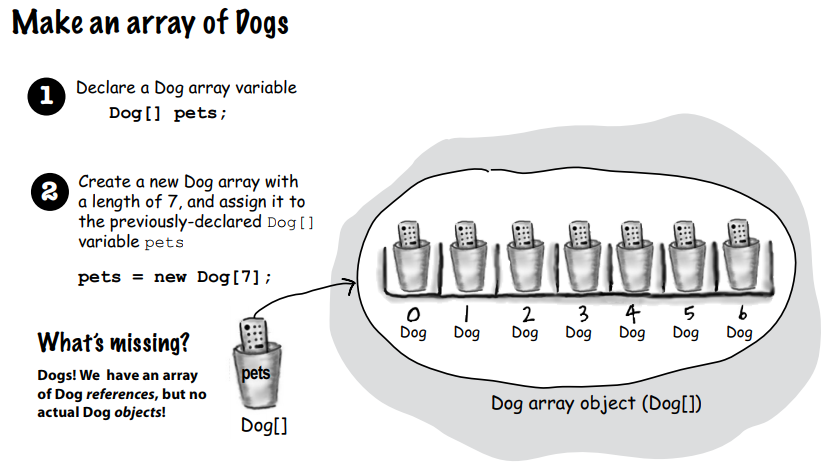
If an object reference variable is marked final, then once it is assigned a Dog, no other object can be assigned to it.

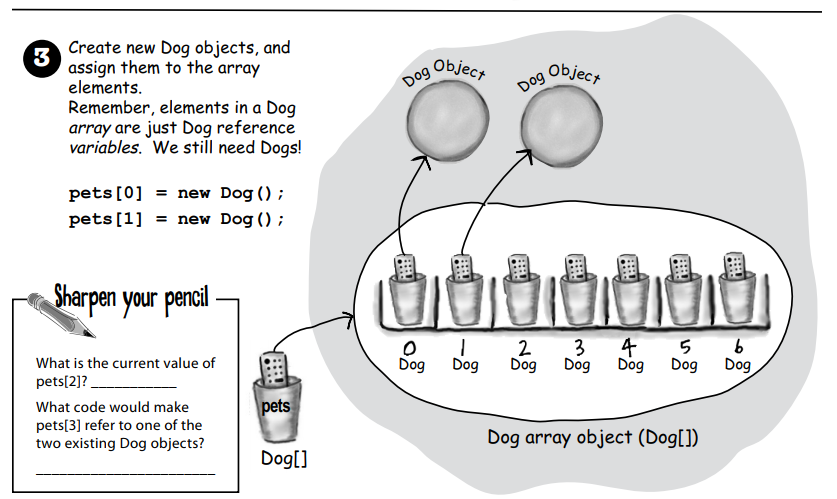
An object reference variable can refer to nothing, meaning the object reference variable is null. Null is a value. The reference is still like a remote control, but it’s like you brought home a new remote control and you don’t have a TV. The remote control is not programmed to control anything.

### arrays

Arrays are always objects, whether they are declared to hold primitives or object references.







## Chapter 4 Methods use instance variables

### parameter VS argument

A method uses parameters.

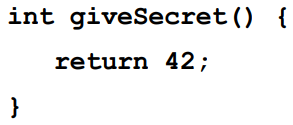
A caller passes arguments.

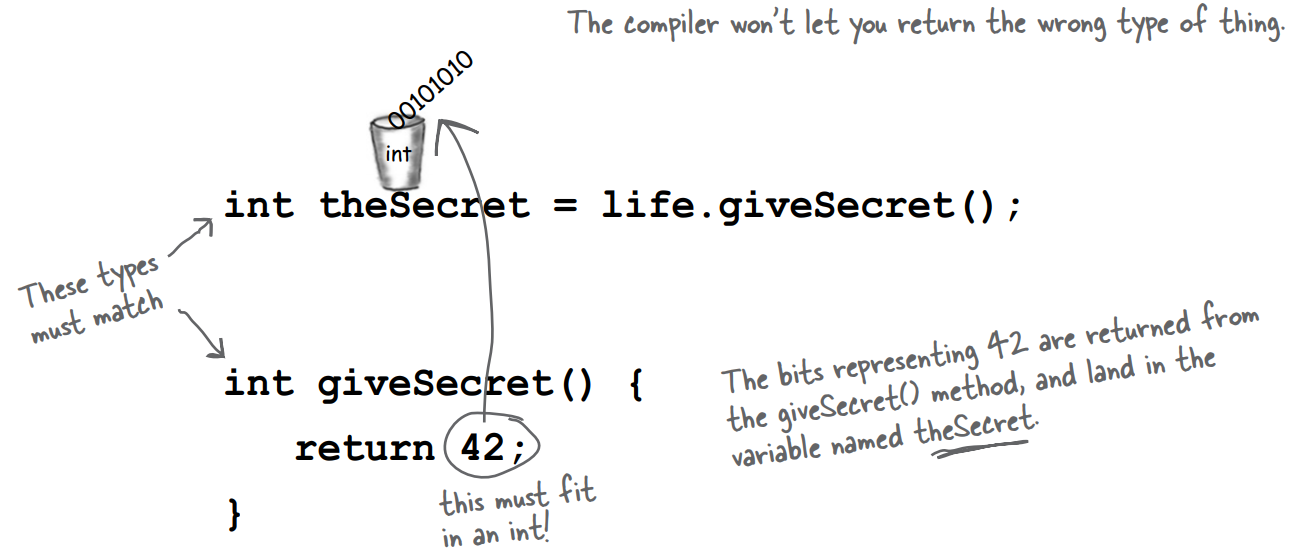
### void method

If a method is made with a void return type, it does not give anything back.

### method returning a specific type

If you declare a method to return a value, you must return a value of the declared type! (Or a value that is compatible with the declared type.)

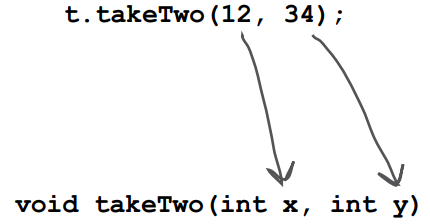




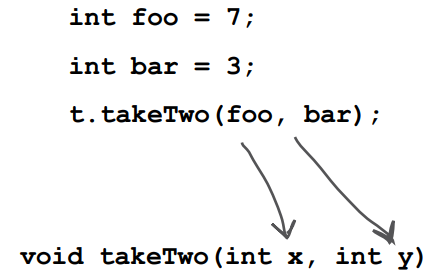
### arguments

If a method has parameters, you must pass arguments of the right type and order.



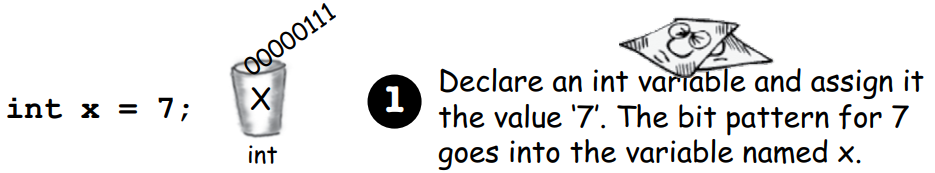




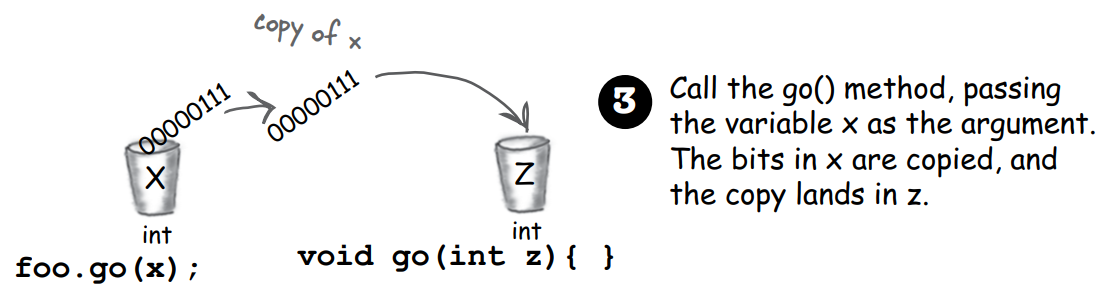


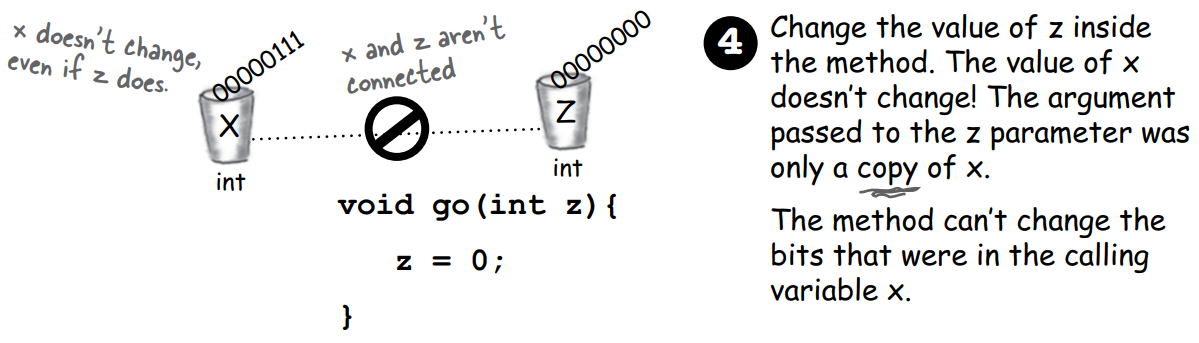
### Java is pass-by-value

Java is pass-by-value. This means pass-by-copy.









It is the same if the argument you want to pass is an object (actually, it should be a reference to an object) instead of a primitive. Java passes everything by value. If you pass a reference to an object into a method, you are passing a copy of the remote control.

### a method declaring multiple return values

A method can declare only one return value. If you want to return, say, three int values, then the declared return type can be an int array. Stuff those ints into the array, and pass it on back.

### Do I have to return the exact type I declared?

You can return anything that can be implicitly promoted to that type. So, you can pass a byte where an int is expected.

When the declared type is smaller than what you’re trying to return you must use an explicit cast.

Same thing with parameters.

### Do I have to do something with the return value of a method?

You can ignore the return value of a method.

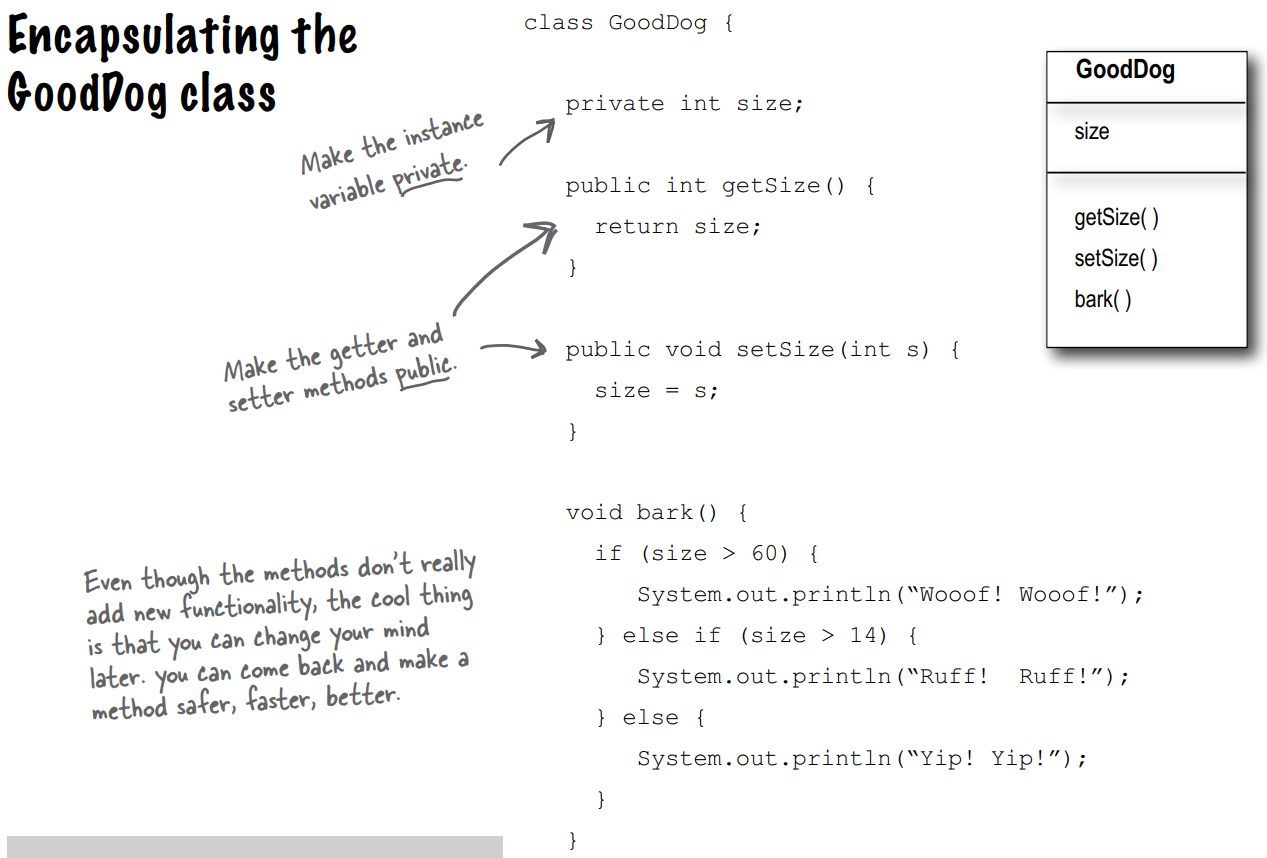
### getters and setters

Getters and Setters (Accessors and Mutators) let you, well, get and set things. Instance variable values, usually.

### hide the data

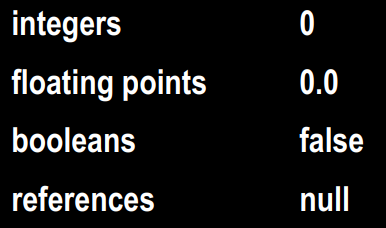
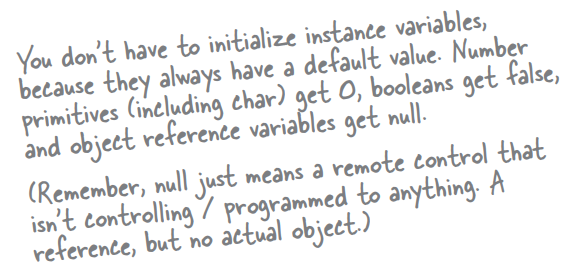
Mark instance variables private.

Mark getters and setters public.

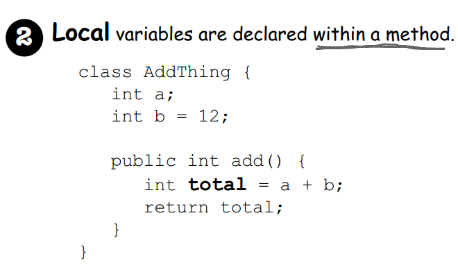
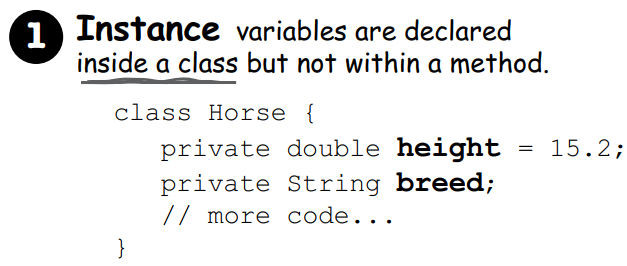


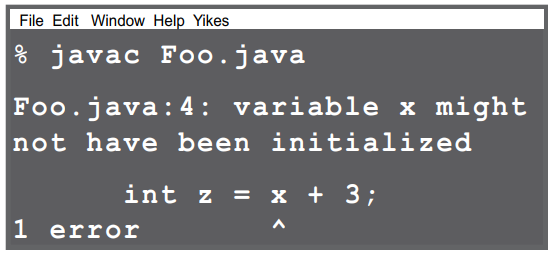
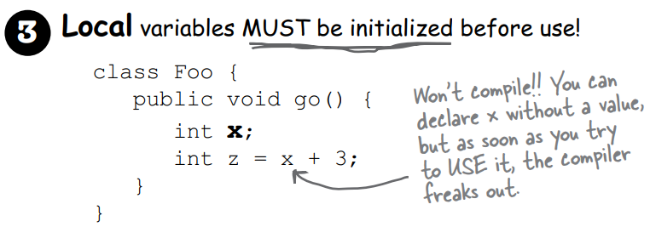
### Instance variables get a default value.

Instance variables always get a default value. If you don’t explicitly assign a value to an instance variable, or you don’t call a setter method, the instance variable still has a value!



### Local variables do not get a default value.





### method parameters – initialized?

Method parameters are virtually the same as local variables—they’re declared inside the method (well, technically they’re declared in the argument list of the method rather than within the body of the method, but they’re still local variables as opposed to instance variables).

But method parameters will never be uninitialized, so you’ll never get a compiler error telling you that a parameter variable might not have been initialized. But that’s because the compiler will give you an error if you try to invoke a method without sending arguments that the method needs.

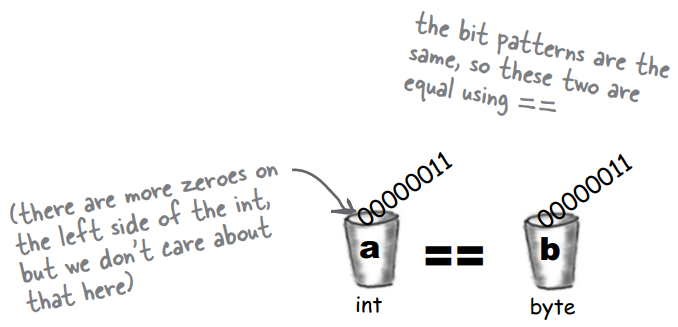
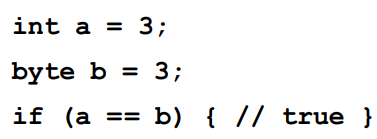
So parameters are ALWAYS initialized, because the compiler guarantees that methods are always called with arguments that match the parameters declared for the method, and the arguments are assigned (automatically) to the parameters.

### comparing variables (primitives and references)

Use == to compare two primitives, or to see if two references refer to the same object.

Use the equals() method to see if two different objects are equal.

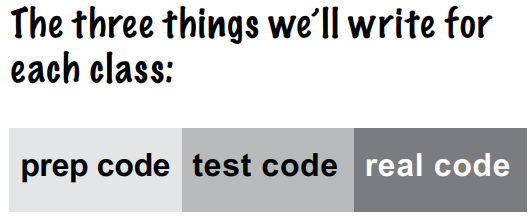
The == operator is used only to compare the bits in two variables. What those bits represent doesn’t matter. The bits are either the same, or they’re not.

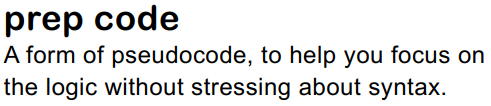


## Chapter 5 Writing a Programme

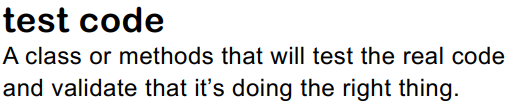
### developing a class

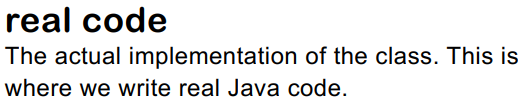
1. Figure out what the class is supposed to do.
2. List the instance variables and methods.
3. Write prepcode for the methods.
4. Write test code for the methods.
5. Implement the class.
6. Test the methods.
7. Debug and reimplement as needed.
8. Express gratitude that we don’t have to test our so-called learning experience app on actual live users.





Most prepcode includes three parts: instance variable declarations, method declarations, method logic.





### Extreme Programming (XP)

Considered by many to be “the way programmers really want to work”, XP emerged in the late 90’s and has been adopted by many companies. The thrust of XP is that the customer gets what he wants, when he wants it, even when the spec changes late in the game.

* Keep it simple.
* Don’t put in anything that’s not in the spec (no matter how tempted you are to put in functionality “for the future”).
* Develop in iteration cycles.
* Make small, but frequent, releases.
* Set realistic schedules, based around small releases.
* No killer schedules; work regular hours.
* Write the test code first.
* Don’t release anything until it passes all the tests.
* Refactor (improve the code) whenever and wherever you notice the opportunity.
* Program in pairs, and move people around so that everybody knows pretty much everything about the code.