**Chapter 1 - Java Building Blocks**

Welcome to the beginning of your journey to become certified on Java. We assume this isn't the first Java programming book you've read. Although we do talk about the basics, we do so only because we want to make sure you have all the terminology and detail you'll need you the OCA exam. If you've never written a Java program before, we recommend you pick up and introductory book on any version of Java-something like Head First Java, 2nd Edition (O'Reilly Media, 2005); Java for Dummies (For Dummies, 2014), or Thinking in Java, 4th Edition (Prentice Hall, 2006). (It's okay if the book covers an older version of Java-even Java 1.3 is fine.) Then come back to this certification study guide.

The chapter covers the fundaments of Java. You'll see how to define and run a Java class, and learn about packages, variables, and the object life cycle.

*Bienvenido al comienzo de su viaje para obtener la certificación en Java. Suponemos que esto no es el primer libro de programación Java que han leído. A pesar de que hablamos de lo básico, lo hacemos sólo porque queremos asegurarse de que tiene toda la terminología y el detalle que usted necesita el examen de OCA. Si usted nunca ha escrito un programa Java antes, se recomienda recoger y libro de introducción en cualquier versión de Java, algo así como La cabeza primero Java, 2nd Edition (O'Reilly Media, 2005); Java para Dummies (For Dummies, 2014), o pensando en Java, 4ª edición (Prentice Hall, 2006). (Está bien si el libro cubre una versión anterior de Java, incluso Java 1.3 está muy bien.) A continuación, volver a esta guía de estudio de certificación.*

*El capítulo cubre los fundamentos de Java. Usted verá cómo definir y ejecutar una clase Java, y aprender acerca de los paquetes, las variables y el ciclo de vida del objeto.*

**Understanding the Java Class Structure**

In Java programs, classes are the basic building blocks. When defining a class, you describe all the  parts and characteristics of one those building blocks. To use most classes, you have to create objects. An object is a runtime instance of a class in memory. All the various objects of all the different classes represent the state of your program.

In the following sections, we'll look at fields, methods and comments. We'll also explore the relationship between classes and files.

*En los programas de Java, las clases son los bloques de construcción básicos. Cuando se define una clase, usted describe todos los elementos y características de uno de esos bloques de construcción. Para utilizar la mayoría de las clases, usted tiene que crear objetos. Un objeto es una instancia de tiempo de ejecución de una clase en la memoria. Todos los diversos objetos de todas las diferentes clases representan el estado de su programa.*

*En las siguientes secciones, vamos a ver los campos, los métodos y los comentarios. También vamos a explorar la relación entre las clases y los archivos.*

**Fields and Methods**

Java classes have two primary elements: methods, often called functions or procedures in other languages, and fields, more generally known as variables. Together these are called the members of the class. Variables hold the state of the program, and methods operate  on that state. If the change is important to remember, a variable stores that change. That's all classes really do. It's the programmer who creates and arranges these elements is such a way that the resulting code is useful and, ideally, easy for other programmers to understand.

Other building blocks  include interfaces, which you'll learn about in Chapter 5, "Class Desing," and enums, which you'll learn about when you start studying for the OCP exam.

*clases de Java tienen dos elementos principales: métodos, a menudo llamadas funciones o procedimientos en otros idiomas, y campos, más generalmente conocidas como variables. En conjunto, estos se llaman los miembros de la clase. Las variables tienen el estado del programa y métodos operan en ese estado. Si el cambio es importante recordar, una variable tiendas que cambian. Eso es lo que hacen realmente todas las clases. Es el programador que crea y organiza estos elementos es de tal manera que el código resultante es útil e, idealmente, fácil para otros programadores de entender.*

*Otros bloques de construcción incluyen interfaces, que usted aprenderá acerca de en el capítulo 5, "Clase Desing", y enumeraciones, que usted aprenderá acerca de cuando se inicia el estudio para el examen de OCP.*

The simplest Java class you can write looks like this:

*La clase Java simple que se puede escribir es el siguiente:*

1: public class Animal {

2: }

Java calls a word with special meaning a keyword. The public keyword on line 1 means the class can be used other classes. The class keyword indicates you're defining a class. Animal gives the name of the class. Granted, this isn't a very interesting class, so add your first field.

*Java llama una palabra con un significado especial una palabra clave. La palabra clave “public” en la línea 1 significa que la clase puede ser usada por otras clases. La palabra clave “class” indica que está definiendo una clase. Animal da el nombre de la clase. Por supuesto, esto no es una clase muy interesante, por lo tanto agregue su primer campo.*

1: public class Animal {

2:    String name;

3: }

On line 2, we define a variable named name. We also define the type of that variable to be a String. A String is a value that we can put text into, such as "this is a string". String is also a class supplied with Java. Next you can add methods:

*En la línea 2, se define una variable llamada “name”. También definimos el tipo de esa variable para que sea una cadena (String). Una cadena (String) es un valor que se puede introducir texto, tal como "esto es una cadena".*

*La cadena (String) es también una clase suministrada con Java. A continuación, puede añadir métodos:*

1: public class Animal {

2:    String name;

3:    public String getName(){

4: return name;

5:    }

6:    public void setName(String newName){

7:     name = newName;

8:    }

9: }

On lines 3-5, you've defined your first method. A method is an operation that can be called. Again, public is used to signify that this method may be called from other classes. Next comes the return type-in this case, the method returns a String. On lines 6-8 is another method. This one has a special return type called void. void means that no value at all is returned. This method requires information be supplied to it from the calling method; this information is called a parameter. setName has one parameter named newName, and it is of type String. This means the caller should pass in one String parameter and expect nothing to be returned.

*En las líneas 3-5, ha definido su primer método. Un método es una operación que puede ser llamado. Una vez más, “public” es usada para indicar que este método puede ser llamado desde otras clases. Luego viene el tipo de retorno, en este caso, el método devuelve una cadena (un String). En las líneas 6-8 es otro método. Éste tiene un tipo de retorno especial llamado “void”. “void” significa que no devuelve valor en absoluto. Este método requiere suministrar información a la misma desde el método de llamada; Esta información se denomina un parámetro. setName tiene un parámetro llamado “newName”, y es de tipo cadena (String). Esto significa que al llamado debe pasar un parámetro cadena “String” y no esperar retorno alguno.*

The full declaration of a method is called a method signature. In this example, can you identify the return type and parameters?

*La declaración completa de un método se denomina firma de método. En este ejemplo, usted puede identificar el tipo de retorno y los parámetros?*

public int numberVisitors(int month)

The return type is int, which is a numeric type. There's one parameter named month, which is of type int as well.

*El tipo de retorno es int, que es un tipo numérico. Hay un parámetro llamado mes (month), que es de tipo int también.*

**Comments**

Another common part of the code is called a comment. Because comments aren't executable code, you can place them anywhere. Comments make your code easier to read. You won't see many comments on the exam-the exam creators are trying to make the code difficult to read-but you'll see them in this book as we explain the code. And we hope you use them in your own code. There are three types of comments in Java. The first is called a single-line comment:

*Otra parte común del código se llama un comentario. Dado que los comentarios no son código ejecutable, se puede colocar en cualquier lugar. Comentarios hacen su código más fácil de leer. No verá muchos comentarios en el examen. Los creadores del examen están tratando de hacer que el código sea difícil de leer, pero se podrán ver en este libro como explicamos el código. Y esperamos que los utilice en su propio código. Hay tres tipos de comentarios en Java. El primero es llamado comentario de una sola línea:*

*//comment until end of line*

A single-line comment begins with two slashes. Anything you type, after that on the same line is ignored by the compiler. Next comes the multiple-line comment:

*Un comentario de línea simple inicia con dos barras oblicuas. Todo lo que escriba, después de que en la misma línea es ignorado por el compilador. Seguido viene el comentario de varias líneas:*

*/\* Multiple*

*\* line comment*

*\*/*

A multiple-line comment (also known as a multiline comment) includes anything starting from the symbol /\* until the symbol \*/. People often type an asterisk (\*) at the beginning of each line of a multiline comment to make it easier to read, but you don’t have to. Finally, we have a Javadoc comment:

*Un comentario de múltiple línea (también conocido como un comentario multilínea) incluye todo lo que inicia desde el símbolo /\* hasta el símbolo \*/. La gente a menudo escribe un asterisco (\*) ans inicio de cada línea en un comentario multilínea para que sea más fácil de leer, pero no tiene que hacerlo. Finalmente, tenemos el comentario Javadoc:*

*/\*\**

*\* Javadoc multiple-line comment*

*\* @autor Jeanne and Scott*

*\*/*

This Comment is similar to a multiline comment except it starts with /\*\*. This especial syntax tells the Javadoc tool to pay attention to the comment. Javadoc comments have a specific structure that the Javadoc tool knows how to read. You won’t see a Javadoc comment on the exam-just remember it exists so you can read up on it online when you start writing programs for others to use.

As a bit of practice, can you identify which type of comment each of these six words is in? Is it a single-line or a multiline comment?

*Este comentario es similar a un comentario multilínea excepto que empieza con /\*\*. Esta sintaxis especial indica a la herramienta Javadoc que preste atención al comentario. Los comentarios Javadoc tienen una estructura específica que la herramienta Javadoc sabe como leer. No verá un comentario Javadoc en el examen, solo recuerda que existe y también podrá leer en línea cuando empiece a escribir programas que otros usaran.*

*Con un poco de práctica, podrá identificar cualquier tipo de comentario de cada una de estas seis palabras debajo? Esto es un comentario de línea simple o multilínea?*

*/\**

*\* // anteater*

*\*/*

*// bear*

*// // cat*

*// /\* dog \*/*

*/\* elephant \*/*

*/\**

*\* /\* ferret \*/*

*\*/*

Did you look closely? Some of these are tricky. Even though comments technically aren’t on the exam, it is good to practice to look at code carefully.

Okay, on to the answers, anteater is in a multiline comment. Everything between /\* and \*/ is part of a multiline comment-even if it includes a single-line comment within it! bear is your basic single-line comment. Cat and Dog are also single-line comments.

Everything from // to the end of the line is part of the comment, even if it is another type of comment, elephant is your basic multiline comment.

The line with ferret is interesting in that it doesn’t compile. Everything from the first /\* to the first \*/ is part of the comment, which means the compiler sees something like this:

*/\* \*/ \*/*

We have a problem. There is an extra \*/. That’s not valid syntax-a fact the compiler is happy to inform you about.

**Classes vs. Files**

Most of the time, each Java class is defined in its own \*.java.file. It is usually public, which means any code can call it. Interestingly, Java does not require that the class be public. For example, this class is just fine:

1: class Animal {

2:   String name;

3: }

You can even put two classes in the same file. When you do so, at most one of the classes in the file is allowed to be public. That means a file containing the following is also fine:

1: class Animal {

2:   private String name;

3: }

4: class Animal2 {

5: }

If you do have a public class, it needs to match the filename. public class Animal2 would not compile in a file named Animal.java. In Chapter 5, we will discuss what nom-public access means.

**Writing a main() Method**

A Java program begins execution with its main() method. A main() method is the gateway between the startup of a process, which is managed by the Java Virtual Machine (JVM), and the beginning