Chapter 1 - Java Building Blocks

Bob Smiths

May 9, 2022

1 Java Class Structure

Java basic structure is composed of classes that are instatiated as objects.

CLASSES := Classes are the basic building blocks.

OBJECT := An object is a runtime instace of a class in memory.

1.1 Field and Methods

Java classes have two primary elements: *methods* and *fields*

PUBLIC := Is used to signify that this method may be called from other classes.

VOID := void means that no value at all is returned.

PARAMETER := The information required as input in the method.

METHOD SIGNATURE := The full declarationn of a method.

1.2 Classes vs. Files

You can even put two classes in the same file. At most one of the classes in the file is allowed to be public. The public class needs to match the filename.

1.3 Writing a *main()* Method

COMPILE JAVA CODE := javac on .java source file.

RUN COMPILED JAVA := java on .class bytecode file (no file extension needed) :w

ACCESS MODIFIER := Declares the method's level of exposure.

***static* KEYWORD** := Binds a method to tis class so it can be called by just the class name.

ARRAY := An array is a fixed-size list of items that are all of the same type. An array is represented by "[]" brackets.

VARARGS := The characters "..." are called varargs (variable argument lists)

1.4 Understanding Package Declarations and Imports

PACKAGES := Packages are logical groupings for classes.

IMPORT STATEMENT := The import statement tells the compiler which package to look in to find a class.

PACKAGE NAMING := The rules for package names are the same as for variable names.

COMPILERS FIGURES OUT WHAT IS ACTUALLY NEEDED := One might think that including so many classes slows down the program, but it doesn't. The compiler figures out what's actually needed.

NAMING CONFLICTS := One of the reasons for using packages is so that class names don't have to be unique across all of java.

DEFAULT PACKAGE := This is a special unnamed package that you should use only for throwaway code. Is the default package because there's no package name.

1.5 Constructors

The name of the constructor matches the name of the class and there's no return type.

The purpose of the constructor is to initialize fields.

For most classes, you don't have to code a constructor - the compiler will supply a "do nothing" default constructor for you.

1.6 Instance Initializer Blocks

CODE BLOCK := The code between the braces is called a *code block*

INSTANCE INITIALIZER := Code blocks that appear outside a method.

1.7 Order of Initialization

The order of initialization is:

- Fields and instance initializer blocks are run in the order in which they appear in the file.
- The constructor runs after all fields and instance initializer blocks have run.

NOTE: Order matters for the fields and blocks of code.

1.8 Distinguishing Between Object References and Primitives

Java applications contains two types of data:

1. Primitive types
2. Reference tyeps

1.9 Primitive Types

Java has eight built-in data types, called *primitive types*. These eight data types represent the building blocks for Java objects, because all Java objects are just a complex collection of these primitive data types:

- boolean
- byte
- short
- int
- long
- float
- double
- char

LITERAL := When a number is present in hte code, it is called literal

1.9.1 Java Bases

Another way to specify numbers is to change the "base". Java allows you to specify digits in several formats:

- octal (digits 0-7), which uses the number 0 as a prefix - e.g.: 017
- hexadecimal (digits 0-9 and letters A-F), which uses the number 0 followed by x of X as a prefix - e.g.: 0xF5
- binary (digits 0-1), which uses the number 0 followed by b or B as a prefix - e.g.: 0b1011

1.9.2 Underscore in Numbers

You can have underscores in numbers to make them easier to read.
RULE := You can add underscores anywhere except:

- At the beginning of a literal
- At the end of a literal
- Right before a decimal point
- Right after a decimal point

1.10 Reference Types

A reference type refers to an object (an instace of a class). Unlike primitive types that hold their values in the memory where the variable is allocated, references do not hold the value of the object they refer to. Instead, a reference "points" to an object by sotring the memory address where the object is located.

POINTER := A reference "points" to an object by storing the memory address where the object is located.

ASSIGN REFERENCE := There are 2 ways to assign a value to a reference:

- A reference can be assigned to another object of the same type.
- A reference can be assigned to a new object using the new keyword.

1.10.1 Key Differences

1. **NULL ASSIGNMENT** : Reference Types can be assigned null, which means they do not currently refer to an object. Primitive types will give you a compiler error if you attempt to assign them null.
2. **CALL METHODS** : Reference types can be used to call methods when they do not point to null. Primitives do not have methods declared on them.
3. All the primitive types have lowercase type names. All classes that come with Java begin with uppercase.

1.11 Declaring and Initializing Variables

VARIABLE := A variable is a name for a piece of memory that stores data.

1.12 Identifiers Rules

There are only three rules to remember for legal identifiers:

1. The name must begin with a letter or the symbol $ or _.
2. Subsequent characters may also be numbers.
3. You cannot use the same name as a Java *reserved word*.

1.13 Reserved Words

A reserved word is a keyword that Java has reserved so that you are not allowed to use it. NOTE: Remember that Java is case sensitive, so you can use versions of the keywords that only differ in case. Please don't, though.

Here is the list with all the 53 reserved words:

- abstract

- assert
- boolean
- break
- byte
- case
- catch
- char
- class
- const*
- continue
- default
- do
- double
- else
- enum
- extends
- false
- final
- finally
- float
- for
- goto*
- if
- implements
- import
- instanceof
- int
- interface

- long
- native
- new
- null
- package
- private
- protected
- public
- return
- short
- static
- strictfp
- super
- switch
- synchronized
- this
- throw
- throws
- transient
- true
- try
- void
- volatile
- while

JAVA CONVENTION: CamelCase : Java has conventions so that code is readable and consistent. This consistency includes CamelCase. In CamelCase, each word begins with an uppercase letter. This makes multiple-word variable names easier to read (e.g. ThisIsCamelCaseNaming).

1.14 variables

Types of variables:

- local
- instance
- class

Local Variable : A local variable is a variable defined whithin a method. Local variables must be initialized before use. They do not have a default value and contain garbage data until initialized. The compiler will not let you read an uninitialized value.

Instance Variable (field) : Variables that are not local variables are known as instance variables or class variables. Instance variables are also called fields.

Class Variable : Class variables are shared cross multiple objects. You can tell a variable is a class variable because it has the keyword static before it.

Instance and class variable do not require you to initialize them. As soon as you declare these variables, they are given a default value.

Variable Type	Default Initialization value
boolean	false
byte, short, int, long	0
float, double	0.0
char	'\0000' (NUL)
All object references (everything else)	null

Note: A method parameter is a local variable. Local variables can never have a scope larger than the method they are defined in. However, they can have a smaller scope.

Variable Scopes :

- Local variables - in scope from declarationn to end of block.
- Instance variables - in scope from declaration until object garbage collected.
- Class variables - in scope from declaration until program ends.

2 Ordering Elements in a Class

Element	Example	Required?	Where does it go?
Package Declaration	package abc;	No	First line in the file
Import statements	import java.util.*;	No	Immediately after the package
Class declaration	public class C	Yes	Immediately after the import
Field declarations	int value;	No	Anywhere inside a class
Method declarations	void method()	No	Anywhere inside a class

PIC: Package, Import, Class

Multiple classes can be dfined in the same file, but only one of them is allowed to be public. The public class must match the name of the file.

3 Destroying Objects

Java automatically takes care of that for you. Java provides a garbage collector to automatically look for objeects that arent't needed anymore. All Java objects are stored in your program memory's heap. The ehap, which is also referred to as the free store, represents a large pool of unused memory allocated to your Java application.

3.1 Garbage Collection

Garbage collection refers to the process of automatically freeing memory on the heap by deleting objects that are no longer reachable in your program. You need to know that System.gc() is not guaranteed to run, and you should be able to recognize when objects become eligible for garbage collection. When calling System.gc() we are merely suggesting Java to kick off a garbage collection run. Java is free to ignore the request. An object will remain in the heap until it is no longer reachable. An object is no longer reachable when one of two situations occurs:

- The object no longer has any references pointing to it.
- All references to the object have gone out of scope.

3.2 finalize()

Java allows objects to implement a method called finalize() that might get called. This method gets called if the garbage collector tries to collect the object. If the garbage collector doesn't run, the method doesn't get called. If the garbage collector fails to collect the object and tries to run it again later, the method doesn't get called a second time. Keep in mind that it might not get called and that it definitely won't be called twice. finalize() call could run zero or one time. finalize() is only run when the object is eligible for garbage collection.

4 Benefits of Java

Java has some key benefits:

- **Object Oriented**
- **Encapsulation**
- **Platform Independent**
- **Robust**
- **Simple**
- **Secure**