**OCA Oracle Certified Associate – Java SE 8 – Recap**

**Chapter 1** : Java Building Blocks

*Java Class Structure*

An object is a runtime instance of a class in memory. All the objects of different classes represent the state of the program.

Members of the class → Fields(Variables) + Methods

Variable → hold the state of the program.

Methods → Operate on variables state.

Keyword: A word with special meaning.

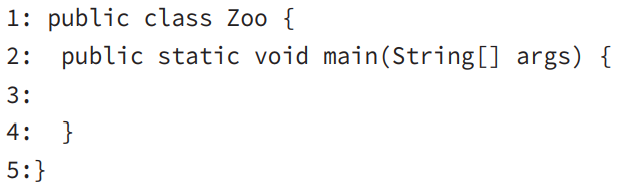
Method: Operation that can be called from other classes.

Method signature: full declaration of a method

*Main method*

A Java program begins execution with its main() method. Main method is the gateway between the startup of a Java process (managed by JVM) and the beginning of the programmer’s code. JVM calls on the underlying system to allocate memory and CPU time, access files and so on.

Example:



To compile and run the code we type it in a file with name **Zoo.java** and we execute:

$ **javac** Zoo.java

$ **java** Zoo

To compile Java Code the file must have the extension: **.java** , while the **name of the file must be the same as the name of the class**. The result of the compilation is a file of ***bytecode*** by the same name, but with the **.class** extension instead of **.java**. **Bytecode** consists of **instructions** that the **JVM** knows how to **execute**.

Main() →

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| public | static | void | main | (String[] args) |  |
| Access modifier, declares the method’s level of exposure to potential callers in the program | Binds the method to the class so it can be called by just the class name ( no objects is needed to be created by Java to call main method)  Presence of a non-static main method will throw an Exception | Return type.  Void return types are preferred when changing an object state | Name | Parameter list. It consists of an array of String objects.  args is only the name of the parameter list. It can be any other name, it just indicates that these are command line arguments.  When giving arguments they are separated by spaces. If you need space inside a word use double quotes: “San Diego” |  |

*Import statements*

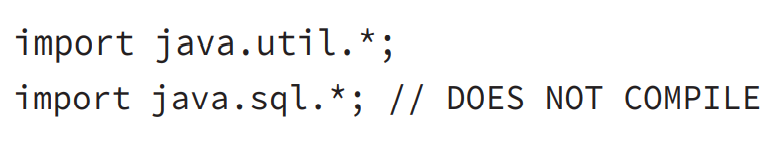
Java classes are grouped into packages. The import statements tell the compiler which package to look in to find a class. This is simlilar to how mailing a letter works. Importing all the classes of a package is done through **wildcard** → **\*. *This kind of approach DOES NOT slow down program. (Wildcard only importes all classes-files, not packages and classes inside nested packages)***

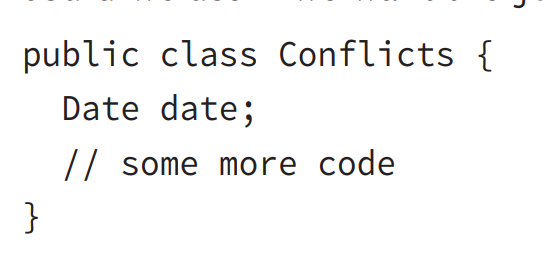
Package: *java.lang* → Is automatically imported in each Java Program.

Classes in the same package are automatically imported.

*Naming conflicts*

When a class (name) used in a program, is present in multiple packages imported , Java will not compile the code. (ambiguous types) for ex. There are 2 Date classes in Java: java.util.Date and java.sql.Date.





If you must use 2 classes with the same name inside one program you can use the qualified names of each or at leas one.

public class Conflicts {

java.util.Date date;

java.sql.Date sqlDate;

}

*Instance initilalizer blocks*

Code Block outside a method → **instance initializer.**

**public class Chicken {**

private String field = “String”;

**{** System.out.println(“Set field to String”); **}**

**void someMethod(){ }**

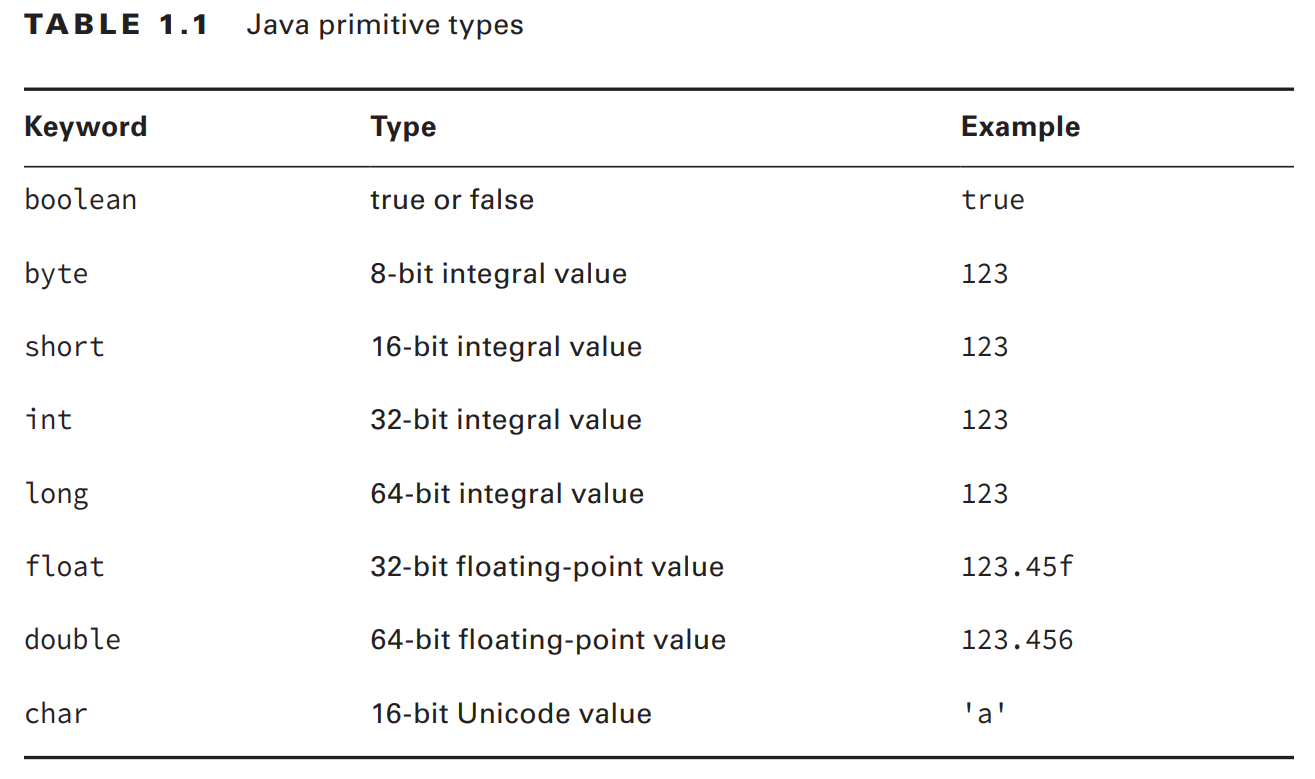
**}**

*Fields and instance initializers are run*

**1)** *In the order they appear in*

**2)** *Before the constructor*

***Object references vs Primitives***

****

When a number is present in the code, it is called a literal.

Java allows you to specify digits in several other formats:

* octal
* hexadecimal (0xFF)
* binary

Number literals can have multiple underscores in order to make them easier to read:

for example:

int million = 1000000;

int million = 1\_000\_000;

Underscores can **NOT** be added in this places :

* in the beginning of the literals
* in the end of the literals
* right before a decimal point
* right after a decimal point

**Reference types:**

A reference type refers to an object. Unlike primitive types that hold their values in memory where the variable is allocated,references do not hold the value of the object they refer to. Instead they hold a reference which points to an object by storing the memory address where the object is located. Unlike other languages, Java does not allow to learn what the physical memory address is. You can only use the reference to refer to the object.

A value is assigned to a reference in one of 2 ways:

* 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

**An object in memory can be accessed only via reference.**

***Key differences***

Reference types can be assigned to null (they don’t currently point to an object) , while primitive values can’t. Trying to set a primitive type to null will give a compiler error and not compile.