1. **Java Building Blocks**

* **Writing a *main()* Method**
  + A main() method is the gateway between the startup of a Java process, which is managed by the *Java Virtual Machine* (JVM), and the beginning of the programmer’s code. The JVM calls on the underlying system to allocate memory and CPU time, access fi les, and so on.
* **Redundant Imports**
  + some imports that don’t work:

import java.nio.\*; // NO GOOD – a wildcard only matches

//class names, not "file.\*Files"

import java.nio.\*.\*; // NO GOOD – you can only have one wildcard

//and it must be at the end

import java.nio.files.Paths.\*; // NO GOOD – you cannot import methods

//only class names

* + **Reading and Writing Object Fields**
    - Reading a variable is known as getting it and Writing to a variable is known as setting it.

Ex1: int numberEggs = 1; // set variable

System.out.println(numberEggs); // read variable

Ex2 : String first = "Theodore";

String last = "Moose";

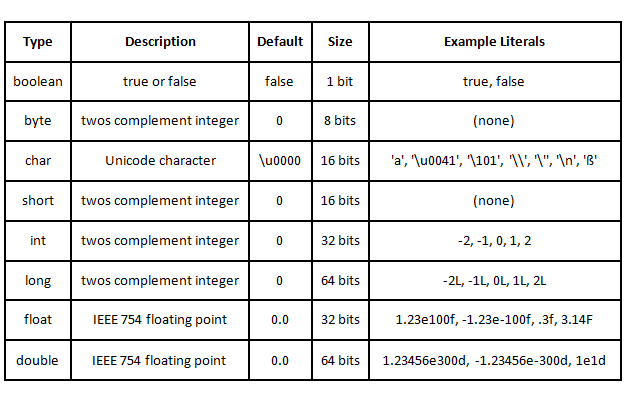
String full = first + last;//Reading and writing

* + **Order of Initialization**
    - 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.
    - Order matters for the fi elds and blocks of code.

***{ System.out.println(name); } // DOES NOT COMPILE***

***private String name = "Fluffy";***

* + **Distinguishing Between Object References and Primitives**
    - Java has eight built-in data types, referred to as the Java *primitive types*.

****

* + - Examples:

long max = 3123456789; // DOES NOT COMPILE

long max = 3123456789L; // now Java knows it is a long

* Java allows you to specify digits in several other formats:

1. octal (digits 0–7), which uses the number 0 as a prefix—for example, 017
2. hexadecimal (digits 0–9 and letters A–F), which uses the number 0 followed by x or X as a prefix—for example, 0xFF
3. binary (digits 0–1), which uses the number 0 followed by b or B as a prefix—for example, 0b10

* Examples:

System.out.println(56); // 56

System.out.println(0b11); // 3

System.out.println(017); // 15

System.out.println(0x1F); // 31

* Underscore in literals. Examples;  
  int million1 = 1000000;

int million2 = 1\_000\_000;

double notAtStart = \_1000.00; // DOES NOT COMPILE

double notAtEnd = 1000.00\_; // DOES NOT COMPILE

double notByDecimal = 1000\_.00; // DOES NOT COMPILE

double annoyingButLegal = 1\_00\_0.0\_0; // this one compiles

* + **Reference Types**
    - A *reference type* refers to an object (an instance of a class).
      * 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.
  + **Key Differences**
    - Reference types can be assigned null, Primitive types will give you a compiler error if you attempt to assign them null.
    - Reference types can be used to call methods when they do not point to null.
* **Identifiers**
  + There are only three rules to remember for legal identifiers:
    - The name must begin with a letter or the symbol $ or \_.
    - Subsequent characters may also be numbers.
    - You cannot use the same name as a Java *reserved word*
* List of Java keywords



* Examples:

***Okidentifier //legal***

***$OK2Identifier //legal***

***\_alsoOK1d3ntifi3r //legal***

***\_ \_SStillOkbutKnotsonice$ //legal***

***3DPointClass // identifiers cannot begin with a number***

***hollywood@vine // @ is not a letter, digit, $ or \_***

***\*$coffee // \* is not a letter, digit, $ or \_***

***public // public is a reserved word***

* Understanding Default Initialization of Variables
  + Local variables must be initialized before use. Compiler will not let you read an uninitialized value.
  + Instance variables—in scope from declaration until object garbage collected.
  + Class variables—in scope from declaration until program ends
  + Tricky ex

***public void findAnswer(boolean check) {***

***int answer;***

***int onlyOneBranch;***

***if (check) {***

***onlyOneBranch = 1;***

***answer = 1;***

***} else {***

***answer = 2;***

***}***

***System.out.println(answer);***

***System.out.println(onlyOneBranch); // DOES NOT COMPILE***

***}***

* Ordering Elements in a Class



* **Destroying Objects**
  + 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.

* Java provides a method called System.gc(), System.gc() is not guaranteed to run.
* An object is no longer reachable when one of two situations occurs:
  1. The object no longer has any references pointing to it.
  2. All references to the object have gone out of scope.
* **Objects vs. References**
* All objects in **Java** are **stored** on the heap. The "**variables**" that hold **references** to them can be on the stack or they can be contained in other objects (then they are not really **variables**, but fields), which puts them on the heap also. The Class objects that define Classes are also heap objects.



* ***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.
  + For the exam, you need to know that this finalize() call could run zero one time.
* **Benefits of Java**
  + Object Oriented
  + Encapsulation
  + Platform Independent
  + Robust
  + Simple
  + Secure

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*