* Classes:
  + They are the basic building blocks. When you define a class you describe all the parts and characteristics of one of those building blocks. You have to create objects to use a class. An object is a runtime instance of a class in memory.
* Fields and methods:
  + Java classes has two primary elements: methods and variables.
  + Variables hold the state of the program and methods operate on that state

* Comments:
  + There are three types of comments:
  + Single line comment: //
  + Multiple line comment: starts with /\* and ends with \*/ and sometimes include \* at the beginning of each comment for readability
  + Javadoc comments: starts with /\*\* and ends with \*/ and it tells Javadoc tool to pay attention to the comment. They have a specific structure that the Javadoc tool knows how to read.
* Classes vs. Files
  + Java does not require that each class is public. This class is just fine class Animal {String name; }
  + When you put 2 classes in the same file, at most one of the classes in the file is allowed to be public. If you have a public class it needs to match the filename. Public class Animal2 would not comple in a file named Animal1.java.
* Writing a main() methods
  + A java program begins executions within its main method.
  + A main() method is the gateway between the startup of java process, which is managed by the JVM. The JVM calls on the underlying system to allocate memory, CPU time, access files, and so. On.
  + The result of a compiled file is bytecode, which consists of instructions that the JVM knows how to execute.
  + When running a class, we must omit the .class extension to run Zoo.java because the period has a reserved meaning in the JVM.
  + Public keyword is called an access modifier, declares the method’s level of exposure.
  + The keyword static binds a method to its class so it can be called by just the class name.
  + Java will throw an error if main method is not present and an exception if the static keyword is not present.
  + [] in the main method is a fixed size list of items that are all of the same type.
  + All command line arguments are treated as string object even if they represent another type
* Understanding package declarations and imports
  + Java organizes classes similar to a file cabinet. You put all your pieces of paper into a folder.
  + Java puts classes in packages. These are logical groupings for classes.
* Wildcards:
  + Classes in the same package are often imported together. You can use a shortcut to import all the classes in a package
  + Import java.util.\*;
  + This only import classes and not sub packages.
  + This import does not slow your program because the compiler figures out what is actually needed
  + The java.lang package is automatically imported by java.
* Naming conflicts:
  + If you use packages then the class names do not have to be unique across all of java
  + A common example of this is the dates class
  + If you explicitly import a clas name, it takes precedence over any wildcards present.
  + You can also use package right in your code java.util.Date date; java.sql.Date sqlDate;
* Creating a new package:
  + If there is no package name then the code is in the default package.
  + You should always use package to avoid naming conflicts
* Constructors:
  + You have to write new to declare a new instance of a class. Random r = new Random()
  + First step is declaring the type of the variable, which gives a place for jave to store a reference to that object.
  + Constructor is a special type of method that created a new object.
  + The name of the constructor should always match the name of the class and it does not have a return type.
  + If a constructor has a return type in it then it is a regular method not a constructor
  + The purpose of a constrictor is to initialize fields, but you can put any code in there.
* Reading and writing object fields:
  + You can read and write fields on the line declaring them
    - String first=”hi”; String last=”none”; String full = first + last;
* Instance initializer blocks:
  + The code between braces is called a code block.
  + Instance initializers: code blocks that appear outside of methods
  + To count code block you just count number of braces
* Order of initialization:
  + Constructors run after all fields and instance initializer blocks have run
* Distinguishing between object references and primvities
  + Java has 8 built-in data types, referred to as the Java primitive types.
  + All of the java objects are complex collection of these primitive data types
  + Here are the 8 primitive types:
    - Boolean (true or false)
    - Byte (8-bit integral value) can hold value from -128 to 127
    - Short (16-bit integral value)
    - Int (32-bit integral value)
    - Long (64-bit integral value)
    - Float (32-bit floating-point value)
    - Double (64-bit floating-point value)
    - Char (16-bit Unicode value)
  + When a number is present in the code, it is called a literal. By default Java assumes you are defining an int in your code, unless you explicitly say f,L
    - Long max = 3122009909 out of range therefore it should be long max = 3122009909L
  + Java allows to specify digits in other formats:
    - Octal (digits 0-7), used the number 0 as a prefix 017
    - Hexadecimal (digits 0-9 and letters A-F), which uses 0 followed by x or X as a prefix for example 0xFF
    - Binary (digits 0-1), uses 0 followed by b or B as a prefix for example 0b10
  + Starting in Java 7 you can add underscore in numbers to make them easier to read int million2 = 1\_000\_000;
    - You can add underscores anywhere expect in the beginning and end of a literal, right before a decimal point, or right after a decimal point.
    - Wrong values+ \_1000.00, 1000.00\_, 1000\_.00
    - Correct value= 1\_00\_0.0\_0;
* Reference types:
  + A reference type refers to an object (an instance of a class)
  + A reference points to an object by storing the memory address where the object is located.
  + Java.util.Date today, String greeting;
    - The today variable is a reference of type Date and can only point to a Date object.
    - The greeting variable is a reference that can only point to a string object. A value is assigned to a reference in one of two ways.
      * A reference can be assigned to another object of the same type
      * A reference cam be assigned to a mew object using the new keyword
* Difference between primitives and references
  + Reference types can be assigned null, which means they do not currently refer to an object.
  + Primitive types will give you a compile error if you assign them to null.
  + Reference types can be used to call methods when they do not point to null.
  + Primitives do not have methods declared on them.
  + Primitive types have lowercase type names. All classes that come with Java begin with upper case.
* Declaring and initializing variables:
  + A variable is a name for a piece of memory that stores data. When you declare a variable you always need to declare variable type and give it a name.
* Declaring multiple variables:
  + String s1,s2;
  + String s3=”yes”,s4=”no”;
  + Int num, String value; //does not compile because it tried to declare multiple variables of different types in the same statement. The shortcut to declare multiple variables in the same statement only works when they share the same type.
  + Double d1, double d2 is illegal because they must share the same type, which they do and the type must not repeat.
  + Int i1; int i2; it is legal because each of them is a separate statement
* Identifiers:
  + Name must beging with letter or sysmol $ or \_
  + Subsequent characters may also be numbers
  + You can not use the same name as a Java reserved word.
* Local variables:
  + Defined within a method
  + Must be initialized before use
  + They do not have default value and contain garbage data until initialized
  + You cannot declare a variable outside of a class
* Instance and class variables
  + Variables that are not local variables are known as instance variables or class variables. Instance variables are also called fields. A class variable has the keyword static before it.
  + Instance and class variable do not require you to initialize them. They are given default values as soon as you declare them.
    - Boolean has a default value of false
    - Byte,short,int,long has default of 0
    - Float, double has a default of 0.0
    - Char has a default of ‘\u0000’ (NUl)
    - All objects reference everything else null
* Understanding variable scope:
  + Local variables – in scope from declaration to end of block and they have to be initialized
  + Instance variables – in scope from declaration until object is garbage collected
  + Class variables (static variables) – in scope from declaration until program ends
* Ordering elements in a class:
  + Package is usually first line in file
  + Import goes after package
  + Class declaration
  + Field declarations
  + Method declarations
* Destroying objects:
  + Java objects are stored in program memory’s heap. The heap, represents a large pool of unused memory allocated to your Java application. There is a limit to the size of the heap. Heap will run out of memory if you keep initializing objects and leaving them on the heap
* Garbage collection:
  + Process of automatically freeing memory on the heap by deleting objects that are no longer reachable in your program.
  + Java providers a method called System.gc(), it just SUGGESTS that now might be a good time to run garbage collection
  + An object is no longer reachable when
    - Object no longer has any references pointing to it
    - All reference to the object have gone out of scope
* Finalize()
  + Java allows objects to implement a method alled finalize()
  + This method gets called if the garbage collector tries to collect the object.
  + If the garbage collector does not run, the method does not get called. If the garbage collector fails to collect the object and tries to run it again later, the method does not get called a second time.
* Benefits of Java
  + **Object oriented:** all code defined in classes can be instantiated into objects
    - Many languages before Java were procedural, which meant there were routines or methods but no classes.
  + **Encapsulation (access modifiers):**
    - Protects data from unintended access and moficiation.
  + **Platform independent:** 
    - Java is an interpreted language because it gets compiled to bytecode. Write once run everywhere
  + **Robust:**
    - Prevents memory leaks
  + **Simple**
  + **Secure:**
    - Java code runs inside the JVM, thus creating a sandbox that makes it hard for Java code to do evil things to the computer it is running on.