A simple java class

Greeter.java

```java
public class Greeter
{
    public Greeter(String aName)
    {
        name = aName;
    }
    public String sayHello()
    {
        return "Hello, " + name + "!";
    }
    private String name;
}
```

GreeterTester.java

```java
public class GreeterTester
{
    public static void main(String[] args)
    {
        Greeter worldGreeter = new Greeter("World");
        String greeting = worldGreeter.sayHello();
        System.out.println(greeting);
    }
}
```

Compilation

- To compile the program enter at the prompt (Unix or Dos) (Greeter.java and GreeterTest.java must be in the current directory):

  ```bash
  javac GreeterTester.java
  ```

  - javac is the java compiler
  - Greeter.java is automatically compiled since GreeterTester requires it.
  - If successful, this command creates the files Greeter.class and GreeterTester.class in the same directory
  - the *.class files contain platform-independent bytecode
  - bytecode is interpreted (executed) by a Java Virtual Machine (JVM), and will run on a JVM installed on any platform
  - The program does NOT need to be recompiled to run on another platform.
Execution

• To run the program enter at the prompt (Unix or Dos):

```bash
workspace jill$ java GreeterTester
Hello World!
workspace jill$
```

✦ This runs the java bytecode on a Java Virtual Machine.
✦ The java tool launches a Java application. It does this by starting a Java runtime environment, loading a specified class, and invoking that class’s main method.
✦ The main method must be declared public and static, it must not return any value, and it must accept a String array as a parameter.

Java Platform

• a bundle of related programs that allow for developing and running programs written in the Java programming language
• two distributions:
  ✦ Java Runtime Environment (JRE) contains the part of the Java platform required to run Java programs (the JVM)
  ✦ Java Development Kit (JDK) is for developers and includes development tools such as the Java compiler, Javadoc, Jar, and a debugger.

Editions of Java

• Different editions of java target different application environments
  ✦ Java Platform, Micro Edition (Java ME) — targeting environments with limited resources.
  ✦ Java Platform, Standard Edition (Java SE) — targeting workstation environments.
  ✦ Java Platform, Enterprise Edition (Java EE) — targeting large distributed enterprise or Internet environments.
• Each edition offers slightly different libraries (APIs) suited for the given environment.
• API: Application Programming Interface: the specification of the interface.

Releases of Java

• Different releases of Java
  ✦ JDK 1.0 (1996) Codename: Oak
  ✦ JDK 1.1 (1997)
  ✦ J2SE 1.2 (1998)
  ✦ J2SE 1.3 (2000)
  ✦ J2SE 5.0 (2004) (1.5)
  ✦ Java SE 6 (2006) (1.6)
  ✦ Java SE 7 (2011) (1.7)
  ✦ Java SE 8 (2014) (1.8) (I have this one)
Principles

• There were five primary goals in the creation of the Java language:
  ✦ It should be "simple, object-oriented and familiar"
  ✦ It should be "robust and secure"
  ✦ It should be "architecture-neutral and portable"
  ✦ It should execute with "high performance"
  ✦ It should be "interpreted, threaded, and dynamic"

Features

• Interesting features of Java
  ✦ Object-oriented: everything is an object
  ✦ Inheritance
  ✦ Polymorphism: can use a subclass object in place of the superclass
  ✦ Garbage collection (dynamic memory allocation)
  ✦ Exception handling: built-in error handling
  ✦ Concurrency: built-in multi-threading
  ✦ Persistence: support for saving objects’ state between executions
  ✦ Platform independence: supports web programming

Primitive types

• These are NOT objects
• Size is not machine-dependent, always the same

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>−2,147,483,648... 2,147,483,647</td>
</tr>
<tr>
<td>short</td>
<td>2 bytes</td>
<td>−32768... 32767</td>
</tr>
<tr>
<td>byte</td>
<td>1 byte</td>
<td>−128... 127</td>
</tr>
<tr>
<td>char</td>
<td>2 bytes</td>
<td>'\u0000'... '\uFFFF'</td>
</tr>
<tr>
<td>boolean</td>
<td></td>
<td>false, true</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
<td>approximately ±1.79769313486231570E+308</td>
</tr>
<tr>
<td>float</td>
<td>4 bytes</td>
<td>approximately ±3.40282347E+38F</td>
</tr>
</tbody>
</table>

Math functions

• These functions are from the Math library class
• The parameters are numbers

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math.sqrt(x)</td>
<td>Square root of x, ( \sqrt{x} )</td>
</tr>
<tr>
<td>Math.pow(x, y)</td>
<td>( x^y (x &gt; 0, or \ x = 0 \text{ and } y &gt; 0, or \ x &lt; 0 \text{ and } y \text{ is an integer}) )</td>
</tr>
<tr>
<td>Math.toRadians(x)</td>
<td>Converts ( x ) degrees to radians (i.e., returns ( x \cdot \pi/180 ))</td>
</tr>
<tr>
<td>Math.toDegrees(x)</td>
<td>Converts ( x ) radians to degrees (i.e., returns ( x \cdot 180/\pi ))</td>
</tr>
<tr>
<td>Math.round(x)</td>
<td>Closest integer to x (as a long)</td>
</tr>
<tr>
<td>Math.abs(x)</td>
<td>Absolute value (</td>
</tr>
</tbody>
</table>
Control flow in Java (same as C++)

- if-else
  ```java
  if(Boolean-expression)
  statement
  else
  statement
  ```
- while, do-while, and for
  ```java
  while(Boolean-expression)
  statement
  do
  statement
  while(Boolean-expression);
  ```
- break and continue
- switch statement like C++

Classes in Java, fields

- A Class defines a type with fields (data) and methods (operations)
- Fields can be objects or primitives
  ```java
  class ClassA {
  int i;
  Weeble w;
  }
  ```
- Can create an object of this class using new:
  ```java
  ClassA a = new ClassA();
  ```
- Fields are accessible using dot operator
  ```java
  a.i = 11;
  a.w = new Weeble();
  ```

All objects in Java are really references

- Methods in Java determine the messages an object can receive.
- They are functions that the object can execute on itself
- Syntax is very similar to C++
  ```java
  class ClassA {
  int i;
  Weeble w;
  int mult (int j) {
    return i*j;
  }
  }
  ```
- Methods are accessible using dot operator
  ```java
  ClassA a = new ClassA();
  a.i = 10;
  int x = a.mult(4);
  ```
- Methods are accessible using dot operator
  ```java
  ```
- Note: references are on the run-time stack, objects are in heap.
Operators in Java

- Mathematical operators, same as C++
  
  ```
  +  -  *  /  \\
  ++  --  %=  /=  %=  
  ```

  ✦ Integer division truncates, like C++

- Relational operators yield boolean result (not int)
  
  ```
  <  <=  >=  ==  !=  
  ```

  ✦ == over objects tests the value of the reference (the pointers)

- Logical operators
  
  ```
  &&  ||  !  
  ```

- String + is concatenation:
  
  ```
  "abc" + "def"  
  ```

  "abcdef"

  this yields a new String object:

Assignment in Java

- Assignment in Java is like in C++
  
  ✦ For primitive types, values are copied

  ```
  int a;  
  a = 10;  
  ```

  ✦ For objects, the reference is copied so both variables refer to the same object.

  ```
  Weeble b = new Weeble();  
  Weeble a;  
  a = b;  // a and b refer to same Weeble object  
  ```

  ✦ changes to a will also affect b

Parameter Passing in Java

- Java uses call by value:
  
  ✦ For primitive types, values are copied to the function parameter

  ✦ For objects, the address of the object is copied to the function parameter

- Objects can be changed by calling mutators on the parameter

  ```
  public class Greeter {  
    public void setName(Greeter other) {  
      other.name = this.name;  
    }  
    public void copyNameTo(Greeter other) {  
      other.name = this.name;      //changes name of other  
    }  
    . . .  
  }  
  ```

  ```
  public class Greeter {  
    public void copyLengthTo(int n) {  
      n = name.length();  
    }  
    public void copyGreeterTo(Greeter other) {  
      other = new Greeter(name);  
    }  
    . . .  
  }  
  ```

  ```
  int length = 0;  
  Greeter worldGreeter = new Greeter("World");  
  Greeter dave = new Greeter("Dave");  
  worldGreeter.copyLengthTo(length);    //does not change length  
  worldGreeter.copyGreeterTo(dave);     //does not change dave  
  ```
this

• The this keyword—which can be used only inside a method—produces a reference to the object the method has been called on.
  ✦ in Java it’s a reference, not a pointer

```java
class ClassA {
    int i;
    void seti(int i) {
        this.i = i;
    }
}
```

• It can also be used to call a constructor from another constructor (Unlike C++):

```java
class ClassA {
    int i;
    ClassA(int i) {
        this.i = i;
    }
    ClassA() {
        this(0); // calls ClassA(0)
    }
}
```

Packages

• Classes can be grouped into packages.
• Package names are dot-separated identifier sequences

```java
class ClassA {
    int i;
    void seti(int i) {
        this.i = i;
    }
}
```

```java
ClassA x = new ClassA();
x.seti(10);  //inside seti, “this” is equal to x
```

• package statement must come first in the file:

```java
package myPackage;
public class SmallBrain { ...
```

• Other classes (outside of myPackage) wanting access to SmallBrain must import myPackage, or fully specify it: myPackage.SmallBrain.

```java
package anotherPackage;
import myPackage.*;
...
SmallBrain a; // myPackage.SmallBrain
```

Packages and Directories

• Package names must match subdirectory names and structure.
• To put your classes in a package called xx.myPackage:
  ✦ Declare the package on the first line of each java file

```java
package xx.myPackage;
import ....
public class SmallBrain { ....
```

• Put all the files in package xx.myPackage in the following directory:
  ...
  src/xx/myPackage

• Make src the current directory:

```bash
cd ...src
```

• To compile:

```bash
javac xx/myPackage/*.java
```

• To run:

```bash
java xx.myPackage.ClassA
```

Assuming ClassA contains a main method

Accessing classes from libraries

• In Java libraries, elements are grouped into packages
• Packages have dotted path names (like internet domains)
• To use a class from a package, import the qualified class name:

```java
import java.util.ArrayList;
```

• Or import the entire package:

```java
import java.util.*;
```
Java library documentation

- Online documentation for Java 1.8 API
  
  http://docs.oracle.com/javase/8/docs/api/

- java.lang is always implicitly loaded
  ✦ System class, contains out field (a static PrintStream)
  ✦ PrintStream has overloaded println methods
- Look for Date in the online documentation
  ✦ java.util.Date
  ✦ shows constructor and other methods in documentation

String

- The String class represents character strings.
- string literals like "abc" are implemented as instances of this class.
- strings are immutable (no methods to change their contents).
- Methods (many more available):
  ✦ length() Returns the length of this string.
  ✦ charAt(int i) Returns the char value at the specified index (but this cannot appear on the left of an assignment, you cannot change the string).
  ✦ + for string concatenation (returns a new string)

```java
String str = "abc";
for (int i=0; i<str.length(); i++)
    System.out.println(str.charAt(i));
System.out.println(str+"def");
```

ArrayList class

- A Generic class: ArrayList<E> contains objects of type E
- Must specify the element types (base type) when declaring:
  ✦ The base type must be a class (NOT primitive type).
- Basic methods:
  ✦ add(E x) Appends the specified element to the end of this list. Starts at position 0, increases size by 1.
  ✦ get(int i) Returns the element at the specified position in this list.
  ✦ set(int i, E x) changes element in position i to x.
  ✦ size() Returns the number of elements in this list (not the capacity).

```java
ArrayList<String> list = new ArrayList<String>();
```
ArrayList class

- ArrayList increase in size as needed automatically
- These methods insert and remove from the middle:
  - add (int i, E x) inserts x at position i, after shifting all the elements from i to the end up by one location
  - remove(int i) Removes the element at the specified position in this list, and closes the gap.
- ArrayList can be iterated over using a “for-each” loop:

```java
ArrayList<String> list = new ArrayList<String>();
//Some code here to fill the list
for (String s : list)
    System.out.println(s); //does this for each String in list
```

Arrays in Java

- Arrays can store objects of any type, including primitives.
- Array length is fixed, array variable is a reference (an object)

```java
int[] numbers = new int[10];
```

- Arrays have bounds checking
  - unable to access memory outside its block (using the array): runtime error
- Arrays are objects
  - member length returns size of array
  - can access elements using [x]

```java
Weeble[] c = new Weeble[4];
for (int i = 0; i < c.length; i++) //can also use foreach loop
    if (c[i] == null)
        c[i] = new Weeble();
```

static keyword

- When a field or method is declared static, it means that data or method is not tied to any particular object instance of that class
- Instances of the class share the same static fields
- Static methods may not access non-static fields

```java
class StaticFun {
    static i = 11;
    static void incr () { i++; } 
}
```

- Static fields and methods may be accessed without instantiating any objects by using the class name, or from an existing object.

```java
StaticFun.i = 100;
StaticFun sf = new StaticFun();
sf.incr();
```

The final keyword

- Java’s final keyword has slightly different meanings depending on the context, but in general it says “This cannot be changed.”
- Data
  - To create named constants (primitive type):

```java
public static final int VAL_THREE = 39;
```
  - Use static so the class does not recreate it for each instance
  - If you create an object that is final, it only means the reference cannot change, but the contents of the object itself could

```java
private final Value v2 = new Value(22);
```
  - Cannot assign v2 to something else, but you could change its fields

```java
v2.setValue(25);
```
Javadoc

- javadoc: a tool to extract comments embedded in source code and put them in a useful form:
  - HTML files, viewable from a browser.
  - Can regenerate the HTML files whenever the comments/code change.
- Uses a special comment syntax to mark the documentation inside the source code
- javadoc also pulls out the class name or method name that adjoins the comment(s).
- html files are similar to the online Java API documentation.
- Purpose is to document the public interface: the class names and public methods.

Javadoc syntax

- The javadoc commands occur only within /** ... */ comments
  - Note the initial double asterisks, normal comments have only one.
- Each javadoc comment must precede the class definition, instance variable definition or method definition that it is documenting.

```java
/** A class comment */
public class DocTest {
    /** A variable comment */
    public int i;
    /** A method comment */
    public void f() {}
}
```

- The javadoc comments may contain the following:
  - embedded html code, especially for lists and formatting code snippets
  - “doc tags”: special keywords that begin with @ that have special meaning to the javadoc tool.

Javadoc tags

- This table summarizes the more commonly used tags.

<table>
<thead>
<tr>
<th>TAG</th>
<th>USED WHERE</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>@author name</td>
<td>Interface and Classes</td>
<td>Indicates the author of the code.</td>
</tr>
<tr>
<td>@since version</td>
<td>Interfaces and Classes</td>
<td>Indicates the version item was introduced.</td>
</tr>
<tr>
<td>@version description</td>
<td>Interfaces and Classes</td>
<td>Indicates the version of the source code.</td>
</tr>
<tr>
<td>@deprecated</td>
<td>Interfaces, Classes and Methods</td>
<td>Indicates a deprecated API item.</td>
</tr>
<tr>
<td>@param name description</td>
<td>Methods</td>
<td>Indicates the method's parameters.</td>
</tr>
<tr>
<td>@return description</td>
<td>Methods</td>
<td>Indicates the method's return value.</td>
</tr>
<tr>
<td>@throws name description</td>
<td>Methods</td>
<td>Indicates exceptions the method throws.</td>
</tr>
<tr>
<td>@see Classname</td>
<td>All</td>
<td>Indicates additional class to see.</td>
</tr>
<tr>
<td>@see Classname</td>
<td>member</td>
<td>All</td>
</tr>
</tbody>
</table>

* required for this class
Javadoc: generating the html files

- Use the javadoc command (from the JDK) to produce the html files:
  ```
javadoc -d api Container.java
  ```
- The -d option indicates a target directory for the html files
- Generates multiple .html files
- click on api/Container.html to see the result.

- For more details on javadoc, follow the javadoc links on the class website “readings” page:
  ```
  http://cs.txstate.edu/~js236/cs4354/readings.html
  ```