Free Java textbook available online


- The third edition is a free electronic book:
  
  http://www.mindview.net/Books/TIJ/

- The fourth edition is much more up to date.

A simple java program

Welcome.java

```java
//This program prints Welcome to Java!

public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```

Compilation

- To compile the program enter at the prompt (Unix or Dos):
  
  `javac Welcome.java`

  ✤ `javac` is the java compiler
  ✤ If successful, this command creates the file Welcome.class in the same directory
  ✤ Welcome.class contains 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):

  workspace jill$ java Welcome
  Welcome to Java!
  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 Card for smartcards.
  ✦ 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

Characteristics of Pure object-oriented programming

• Everything is an object.
  ✦ attributes + operations
• A program is a bunch of objects telling each other what to do by sending messages
  ✦ a message as a request to call a method that belongs to a particular object
• Each object has its own memory made up of other objects.
  ✦ this is how to represent complex systems
• Every object has a type.
  ✦ its type is a class, the class specifies the methods of the object
• All objects of a particular type can receive the same messages.
  ✦ Even the instances of the subclasses

All objects in Java are really references

• Everything is treated as an object, using a single consistent syntax.
• However, the identifier you manipulate is actually a “reference” to an object
  ```java
  String s; //this is just a ref, a pointer
  ```
• Safer to initialize a reference when you create it:
  ```java
  String s = “asdf”;
  ```
• Usually you use “new” to create new objects:
  ```java
  String s = new String(“asdf”);
  ```
• Note: references are on the run-time stack, objects are in heap.
Special case: primitive types

- These are NOT references, not objects
- They are stored on the run-time stack
- Size is not machine-dependent, always the same

<table>
<thead>
<tr>
<th>Primitive type</th>
<th>Size</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Wrapper type</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Boolean</td>
</tr>
<tr>
<td>char</td>
<td>16-bit</td>
<td>Unicode 0</td>
<td>Unicode 2^16-1</td>
<td>Character</td>
</tr>
<tr>
<td>byte</td>
<td>8-bit</td>
<td>-128</td>
<td>+127</td>
<td>Byte</td>
</tr>
<tr>
<td>short</td>
<td>16-bit</td>
<td>-2^15</td>
<td>+2^15-1</td>
<td>Short</td>
</tr>
<tr>
<td>int</td>
<td>32-bit</td>
<td>-2^31</td>
<td>+2^31-1</td>
<td>Integer</td>
</tr>
<tr>
<td>long</td>
<td>64-bit</td>
<td>-2^63</td>
<td>+2^63-1</td>
<td>Long</td>
</tr>
<tr>
<td>float</td>
<td>32-bit</td>
<td>IEEE754</td>
<td>IEEE754</td>
<td>Float</td>
</tr>
<tr>
<td>double</td>
<td>64-bit</td>
<td>IEEE754</td>
<td>IEEE754</td>
<td>Double</td>
</tr>
<tr>
<td>void</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Void</td>
</tr>
</tbody>
</table>

Default values for fields

- If you provide no explicit initialization to instance variables, they will be assigned the following default initial values

<table>
<thead>
<tr>
<th>Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>false</td>
</tr>
<tr>
<td>byte</td>
<td>(byte) 0</td>
</tr>
<tr>
<td>short</td>
<td>(short) 0</td>
</tr>
<tr>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>long</td>
<td>0L</td>
</tr>
<tr>
<td>char</td>
<td>\u0000</td>
</tr>
<tr>
<td>float</td>
<td>0.0f</td>
</tr>
<tr>
<td>double</td>
<td>0.0d</td>
</tr>
<tr>
<td>object reference</td>
<td>null</td>
</tr>
</tbody>
</table>

- These apply to fields (and array elements), not to local variables.

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:

```
ClassA a = new ClassA();
```

- Fields are accessible using dot operator

```
a.i = 11;
a.w = new Weeble();
```

Classes in Java, methods

- 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

```
ClassA a = new ClassA();
a.i = 10;
int x = a.mult(4);
```
Arrays in Java

- An array is ALWAYS initialized to default values (see slide 16)
  - cannot access uninitialized elements by mistake
- Arrays have bounds checking
  - unable to access memory outside its block (using the array): runtime error
- This is to enforce safety (though it requires overhead)
- Arrays are objects, contain primitives or references to 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++)
  if(c[i] == null) // test for null reference
    c[i] = new Weeble();
```

ArrayList class

- Must specify the element types (base type) when declaring:
  ```java
  ArrayList<String> list = new ArrayList<String>(20);
  ```
  - 20 is the initial capacity
  - The base type must be a class (NOT primitive type).
- Basic methods:
  - add(BaseType x) Appends the specified element to the end of this list. Starts at position 0, increases capacity as necessary.
  - get(int i) Returns the element at the specified position in this list.
  - size() Returns the number of elements in this list (not the capacity).
  - remove(int i) Removes the element at the specified position in this list, and closes the gap.

array vs. ArrayList

- array elements can be any type, ArrayList must contain objects.
- ArrayList can increase in size as needed (array size cannot be changed).
- ArrayList implements a “partially filled array” automatically. For an array, you must manage the size and implement “add” and “remove” operations yourself.
- ArrayList can be iterated over using a “for-each” loop:

```java
ArrayList<String> list = new ArrayList<String>(20);
//Some code here to fill the list
for (String s : list)
  System.out.println(s); //does this for each String in list
```

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.*;
  ```

- General syntax is: for (BaseType var : arrayList) stmt
Java library documentation

• Online documentation for Java 1.8 API
  
  [http://docs.oracle.com/javase/8/docs/api/](http://docs.oracle.com/javase/8/docs/api/)
  
• java.lang is always implicitly loaded
  
  ✦ System class, contains out field (a static PrintWriter)
  
• Look for Date in the online documentation
  
  ✦ java.util.Date
  
  ✦ shows constructor and other methods in documentation

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();
```

A Java program

```java
// HelloDate.java
import java.util.*;

public class HelloDate {
    public static void main(String[] args) {
        System.out.println("Hello, it's: ");
        System.out.println(new Date());
    }
}
```

• Standalone program: one class must have same name as file. that class must have a main method with signature as above.

• args are for command line arguments.

• public means method is available outside the file

• comments: /* ... */ or //...to end of line

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.
- 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</td>
<td>Interface and Classes</td>
<td>Indicates the author of the code.</td>
</tr>
<tr>
<td>@since version</td>
<td>Interface and Classes</td>
<td>Indicates the version item was introduced.</td>
</tr>
<tr>
<td>@version</td>
<td>Interfaces and Classes</td>
<td>Indicates the version.</td>
</tr>
<tr>
<td>@deprecated</td>
<td>Interfaces, Classes and Methods</td>
<td>Indicates a deprecated API item.</td>
</tr>
<tr>
<td>@param name</td>
<td>Methods</td>
<td>Indicates the method's parameters.</td>
</tr>
<tr>
<td>@return name</td>
<td>Methods</td>
<td>Indicates the method's return value.</td>
</tr>
<tr>
<td>@throws name</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>All</td>
<td>Indicates additional member to see.</td>
</tr>
</tbody>
</table>

**Javadoc: generating the html files**

- Use the javadoc command (from the JDK) to produce the html files:
  ```bash
  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:
  ```html
  http://cs.txstate.edu/~js236/cs4354/readings.html
  ```
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:
  this yields a new String object:
  "abc" + "def" = "abcdef"

Assignment in Java

• Assignment in Java is like in C++
  ✦ For primitive types, values are copied
  ```java
  int a;
  a = 10;
  ```
  ✦ For objects, the reference is copied so both variables refer to the same object.
  ```java
  Weeble b = new Weeble();
  Weeble a;
  a = b;  // a and b refer to same Weeble object
  ```
  ✦ changes to a will also affect b

• Objects are passed by reference by default

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);
  for(initialization; Boolean-expression; step)
  statement
  ```

• break and continue (also with labels)
• switch statement like C++

String

• The String class represents character strings.
• All string literals in Java programs, such as "abc", are implemented as instances of this class.
• 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

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

- `toString` is a method that is defined by default for every class
  ```java
  public String toString();
  ```
- The String value returned should represent the data in the object.
- This makes it easy to output an object to the screen. The following are generally equivalent:
  ```java
  System.out.println(w);
  System.out.println(w.toString());
  ```
- You can override the default definition by redefining `toString` for your class:
  ```java
  class ClassA {
      private int i;
      private double x;
      public String toString() {
          return "i: \"+i\" x: \"+x\";
      }
  }
  ```

**Constructors**

- Like C++:
  - classes can have constructor functions to initialize their fields.
  - these are named the same as the class, they have no return type, and can be overloaded.
  - they are called automatically (primarily when “new” is used to create an instance of a class).
  - if you don’t create one for your class, a default (no-arg) constructor is created for you (initializes fields to default values).
- Unlike C++:
  - you can call a constructor from within another constructor (see next slide)

**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:
  ```java
  package myPackage;
  import ....
  public class SmallBrain { ....
  ```
  - Declares these classes to belong to a package called “myPackage”
  - package statement must come first in the file.
  - Other classes (outside of myPackage) wanting access to SmallBrain must import myPackage, or fully specify it: myPackage.SmallBrain.
  - This is a mechanism to manage name spaces: this code will work with another package that has its own SmallBrain class.
  - Anytime you create a package, you implicitly specify a directory structure: this file should be in a directory named “myPackage”
Packages: example

- 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: cd ...src
  ✦ To compile: javac xx/myPackage/*.java
  ✦ To run: java xx.myPackage.ClassA

Assuming ClassA contains a main method

Access specifiers

- keywords that control access to the definitions they modify
  ✦ `public`: accessible to all other classes
  ✦ `protected`: accessible to classes derived from (subclasses of) the class containing this definition as well as other classes in the same package.
  ✦ `package` (unspecified, default): accessible only to other classes in the same package
  ✦ `private`: accessible only from within the class in which it is defined

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);
    ```