Computer Science II

Java

OO Programming
Scott C Johnson
Rochester Institute of Technology
Outline

- Object-Oriented (OO) Programming and Design
- Into to Java
- Compiling and Running Java Programs
- The Hello World Program in Java
- Static Typing
- Basic Types
- Class Types
- Control-Flow Statements
- Generic Types

Diagram:

1. HelloWorld.java
2. `jавса HelloWorld.java` → HelloWorld.class
3. `java HelloWorld` → "Hello World !"
OO Programming and Design

- OO Design and Programming
  - “build from pieces”
  - “individual functions solving individual algorithmic problems”
- Making a large software project
  - Build from pieces
  - Need a tool to verify the pieces fit
  - In Python this was done at run time
- In Java and OO
  - A program is made of objects
  - Objects can communicate
  - Each object represents a conceptual “thing”
    - Maintains its own state and behavior
  - OO is meant to model the real world
We will use Java!
  - Widely used
  - Designed to support and encourage OO
  - Contains features to verify the pieces fit
Introduction to Java

- Java is great for OO… but not yet
  - Need to become comfortable with procedural programming in Java
- We will compare Java and Python

<table>
<thead>
<tr>
<th>Java</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>code must exist in a class</td>
<td>code can exist anywhere</td>
</tr>
<tr>
<td>typically stand-alone execution (by java virtual machine)</td>
<td>interactive execution (by python interpreter)</td>
</tr>
<tr>
<td>block syntax</td>
<td>layout syntax</td>
</tr>
<tr>
<td>variables explicitly declared before use</td>
<td>variables implicitly declared at use</td>
</tr>
<tr>
<td>variables statically typed</td>
<td>variables dynamically typed</td>
</tr>
</tbody>
</table>
Compiling and Running Java

- Python was automatically compiled at run time
  - Used the Python interpreter *(python3)*
  - Translated source code into byte-code as needed
    - Byte-code is a more compact version of code
    - Not often human readable
- Java does this in two steps
  - Java compiler *(javac)*
    - .java → .class
    - Usage: *javac* file0.java [file1.java ...filen.java]
  - Java Virtual Machine *(java)*
    - Runs the program
    - Requires a .class file
      - Class file must contain a *main* function
    - Usage: *java* MainClassName [ arg0 arg1 ...argN ]
- *Javadoc*
  - Used to produce an html documentation of your code
  - You will be use to reading these by the end of the semester
Hello World Program

- HelloWorld.java

```java
/* code/HelloWorld/HelloWorld.java */

/**
 * Display "Hello, world!" and end the program.
 * @author Matthew Fluet
 * @author Ben K Steele
 */
public class HelloWorld {

  /**
   * @param args args are unused
   */
  public static void main( String[] args ) {
    System.out.println( "Hello, world!" );
    return;
  }
}
} // end class
```
Hello World Program

• Observations
  • *All code in a Java program in written within classes!!!*
    • The file must be named the same as the class!
      • Class `HelloWorld` in `HelloWorld.java`
    • No functions or variables outside of a class
    • A static method belongs to the class itself
      • Does not belong to an object created from this class
    • Braces (`{` and `}`) enclose and surround blocks of code
      • Indentation is irrelevant, unlike Python
      • But is good style!
    • Statements end with semicolon `;`
      • Can be on many lines
    • Every variables and methods must have a declared type
      • And must be declared before use
Hello World Program

- Command-line Execution

```
[mtf@fenrir HelloWorld]$ ls
HelloWorld.java
[mtf@fenrir HelloWorld]$ javac HelloWorld.java
[mtf@fenrir HelloWorld]$ ls
HelloWorld.class HelloWorld.java
[mtf@fenrir HelloWorld]$ java HelloWorld
Hello, world!
```
Static Typing

- Used to make sure “pieces fit together”
  - Java uses types to help with this
  - Python has types but they are determined at run time (*dynamic typing*)
- Unlike Python, Java requires you to state the type
  - This is done in your code (*static typing*)
  - This allows Java to ensure you cannot assign a string to a number
    - Recall `rit_lib`? Java does this to a much greater extent

```java
/* Declare 'b' to be a variable of type 'boolean'. */
boolean b;

/* Declare 'i' to be a variable of type 'int'. */
int i;

boolean b = true;
int i = 1;
```
Static Typing

- What if you try to store a number in a boolean?
  - Or a boolean in a number?
- **Javac** handles this...
  - Reports an error!

```java
/* Declare 'b' to be a variable of type 'boolean'. */
boolean b;
/* Declare 'i' to be a variable of type 'int'. */
int i;

/* Try to store a 'boolean' in an 'int' variable. */
i = true;
/* Try to store an 'int' in a 'boolean' variable. */
b = 1;

[mtf@fenrir TypeErrors]> javac TypeErrorA.java
TypeErrorA.java:9: incompatible types
  found    : boolean
  required: int
        i = true;
    ^
TypeErrorA.java:11: incompatible types
  found    : int
  required: boolean
        b = 1;
    ^
2 errors
```
Static Typing

- Like Python, Java overloads some operators:
  - `+` can be:
    - Numeric addition
    - String concatenation
  - Java will make some choices based on the types...
  - Mixing types can have some interesting outcomes!
- Every variable and method must have a type
  - The type of a method is the type it returns
  - If the method returns nothing, its type is `void`
Static Typing

- **Access Levels**
  - Determine what other classes can access its parts
  - Classes can be in the same *package*
    - A collection of one or more classes
    - Correspond to file system directories
  - Four access levels:
    - **public**: means that any other classes may access the component.
    - **private**: means that only (instances of) this class may access the component.
    - **protected**: means that only this class, subclasses of this class, and other classes in the same package as this class may access the component.
    - **no keyword**: means that only this class and other classes in the same package as this class may access the component.

- **Compiler will yell if you violate the access rules.**
Basic Types

- Two classifications:
  - Base types (aka primitives)
    - Not objects… have no variables or methods
    - Base types in Java:
      - `boolean`: true/false
      - `char`: characters
      - `byte`: byte in memory
      - `short`: small integer
      - `int`: integers
      - `long`: larger integer
      - `float`: floating point number
      - `double`: double precision number
  - Class Types
    - User defined or library provided
Basic Types

- Standard set of arithmetic and boolean operators:
  - Many are *overloaded*; can do multiple things or multiple types

<table>
<thead>
<tr>
<th>operator</th>
<th>operation</th>
<th>argument base types</th>
<th>result base type</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>numeric</td>
<td>same as arguments</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>numeric</td>
<td>same as arguments</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>numeric</td>
<td>same as arguments</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>numeric</td>
<td>same as arguments</td>
</tr>
<tr>
<td>%</td>
<td>modulo</td>
<td>numeric</td>
<td>same as arguments</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>numeric</td>
<td>boolean</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>numeric or boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>==</td>
<td>equal to</td>
<td>numeric or boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
<td>numeric or boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>numeric</td>
<td>boolean</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>numeric</td>
<td>boolean</td>
</tr>
<tr>
<td>!</td>
<td>not (prefix)</td>
<td>boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>conditional (short-circuit) and conditional (short-circuit) or</td>
<td>boolean</td>
<td>boolean</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>bitwise complement (prefix)</td>
</tr>
<tr>
<td>&amp;</td>
<td>bitwise and</td>
<td>numeric and boolean</td>
<td>same as arguments</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bitwise or</td>
<td>numeric and boolean</td>
</tr>
<tr>
<td>^</td>
<td>bitwise exclusive-or</td>
<td>numeric and boolean</td>
<td>same as arguments</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>shift bits left (filling with zeros)</td>
<td>numeric</td>
<td>same as arguments</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>shift bits right (filling with sign)</td>
<td>numeric</td>
<td>same as arguments</td>
</tr>
<tr>
<td>&gt;&gt;&gt;</td>
<td>shift bits right (filling with zeros)</td>
<td>numeric</td>
<td>same as arguments</td>
</tr>
</tbody>
</table>
Class Types

- Most everything in Java is an object
  - Objects are instances of a class
  - Classes give objects types
- The `new` keyword is used to make a new object
  - Example: `Foo x = new Foo( y );`
- The `null` keyword
  - Not an object
  - No variables or methods
  - Similar to `None` in Python
- When a variable said to have type $T$ when you access it you get:
  - An object value; guaranteed to be of type $T$
  - Or `null`
- `==` or `!=` can be applied to objects, or `null`
  - Examine identity (location or address)
  - Not values contained
Class Types

- The **Object** class
  - *The top level parent of all classes*
  - All methods for **Object** are part of every other class

- The **Array** class
  - Arrays are special objects
  - It has a class type since it is an object
    - But have no class definition source file
    - Builtin to Java language
  - Written using [ ]
  - Example `String[ ] myStringArray;`
  - Arrays have not methods
    - Have a public `length` variable
  - No constructors
    - Example `myStringArray = new String[10];`
    - Size 1s fixed at 10!
  - Accessing elements
    - Example: `myStringArray[3];`
Class Types

- The **String** class
  - Fixed sequence of characters
  - Written surrounded by double quotes (""")
    - Single quotes (‘’) is a character
  - Cannot use `myString[3]` to access a character
    - Use `myString.charAt(3)`
  - The + can concatenate Strings with some other type
    - The other type is converted to a String type
    - Uses the object’s `toString()` method if not a primitive
  - String class has a `length()` method
Class Types

- **Wrapper Classes**
  - Wrap the base types
  - One for each base type
  - Useful when an object is needed
  - Example `Integer i = new Integer(42);`
  - Java will convert automatically between base types and wrapper
    - Known as *autoboxing* and *autounboxing*

<table>
<thead>
<tr>
<th>base type name</th>
<th>wrapper class name</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>char</td>
<td>Character</td>
</tr>
<tr>
<td>byte</td>
<td>Byte</td>
</tr>
<tr>
<td>short</td>
<td>Short</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>long</td>
<td>Long</td>
</tr>
<tr>
<td>float</td>
<td>Float</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
</tbody>
</table>
Control-flow Statement

- Many you have seen in Python
  - Some you have not
- The `if` statement
  - Takes a boolean expression, in parentheses
  - Executes when expression is true
  - Optionally followed by an `else if` or `else` block
  - Requires braces to enclose the blocks

```java
if ( i < 0 ) {
    System.out.println( i + " is negative." );
} else if ( i > 0 ) {
    System.out.println( i + " is positive." );
} else {
    System.out.println( i + " is zero." );
}
```
Control-flow Statement

- Looping Statements
  - Three different
    - ForEach
    - Similar to Python `for` loop
    - Can be used with arrays

```java
class ArgsForEach {
    public static void main(String[] args) {
        for (String s : args) {
            System.out.println(s);
        }
    }
}
```
Control-flow Statement

- Looping Statements (con’t)
  - Three different
    - While loop
    - Similar to Python `while` loop

```java
class ArgsWhile {
    public static void main(String[] args) {
        int i;
        i = 0;
        while (i < args.length) {
            System.out.println(args[i]);
            i = i + 1;
        }
    }
}
```
Control-flow Statement

- Looping Statements (con’t)
  - Three different
    - For while loop

```java
class ArgsForCount {
    public static void main(String[] args) {
        int i;
        for (i = 0; i < args.length; i = i + 1) {
            System.out.println(args[i]);
        }
    }
}
```
Generic Types

- Using collections of objects
  - Types cause issues here
  - Java needs to know the types of the internal objects
  - We do this with generics
    - Allow us to state the internal type when we make it
    - Means Java collections only can hold one type
Generic Types

- The `ArrayList` class
  - Good example of generic type usage
  - Very similar to Python lists
    - But all of its objects must be of the same type
    - We must tell it this type
    - Size in not fixed
  - Example `ArrayList<String> al = new ArrayList<String>();`
    - Made an array list containing strings
  - Adding to an ArrayList:
    ```java
    al.add("One");
    al.add("Two");
    ```
  - Looping an ArrayList and getting value:
    ```java
    for ( int i = 0; i < al.size(); i += 1 ) { 
        System.out.println("al.get(" + i + ") = " + 
                           al.get(i));
    }
    ```
Generic Types

- The **ArrayList** class (con’t)
  - Removing an item
  - Example `al.remove("Two");`
    - Returns true if removed, false otherwise

```java
if ( ! al.remove("Two") ) {
    System.out.println( "\"Two\" not removed" );
}
```

- For Each loop

```java
for ( String s : al ) {
    System.out.println(s);
}
```
Generic Types

- Using ArrayLists and generic types wrong

```java
// Trying to add an 'int' to an 'ArrayList<String>'.
al.add(5);

// add is not a method of the String class.
System.out.println("al.get(" + i + ").add(\"Nine\") = " +
al.get(i).add("Nine");
```

- Compiler will scream!

```
[mtf@fenrir Generics]$ javac GenericsTypeError.java
GenericsTypeError.java:13: cannot find symbol
symbol : method add(int)
location: class java.util.ArrayList<java.lang.String>
al.add(4);
  ^
GenericsTypeError.java:39: cannot find symbol
symbol : method add(java.lang.String)
location: class java.lang.String
  al.get(i).add("Nine");
    ^
2 errors
```
Questions?