Crash Course in Java

Why Java?

- Network Programming in Java is very different than in C/C++
  - much more language support
  - error handling
  - no pointers! (garbage collection)
  - Threads are part of the language.
  - some support for common application level protocols (HTTP).

Java notes for C++ programmers

- Everything is an object.
- No code outside of class definition!
- Single inheritance
  - an additional kind of inheritance: interfaces
- All classes are defined in `.java` files
  - one top level public class per file

More for C++ folks

- Syntax is similar (control structures are very similar).
- Primitive data types similar
  - bool is not an int.
- To print to stdout:
  - System.out.println();

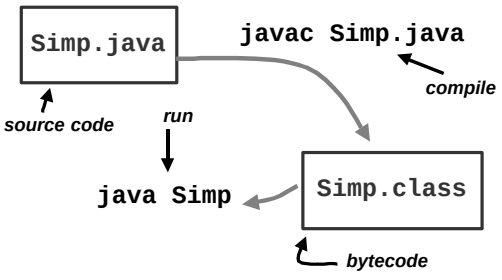
First Program: **`Simp.java`**

```
public class Simp {

  public static void main(String args[]) {
   System.out.println("Hello, Netprog");
  }

}
```


Compiling and Running



Java bytecode and interpreter

- bytecode is an intermediate representation of the program (class).
- The Java interpreter starts up a new "Virtual Machine".
- The VM starts executing the users class by running it's **`main()`** method.

The Language

- Data types
- Operators
- Control Structures
- Classes and Objects
- Packages

Java Data Types

- Primitive Data Types:
  - boolean **`true`** or **`false`** *not an int!*
  - **`char`** unicode! (16 bits)
  - **`byte`** signed 8 bit integer
  - **`short`** signed 16 bit integer
  - **`int`** signed 32 bit integer
  - **`long`** signed 64 bit integer
  - **`float`**,**`double`** IEEE 754 floating point

Other Data Types

- *Reference types (composite)*
  - classes
  - arrays
- strings are supported by a built-in class named **`String`**
- string literals are supported by the language (as a special case).

Operators

- Assignment: =, +=, -=, *=, …
- Numeric: +, -, *, /, %, ++, --, …
- Relational: ==. !=, <, >, <=, >=, …
- Boolean: &&, ||, !
- Bitwise: &, |, ^, ~, <<, >>, …

Just like C/C++!

Control Structures

- More of what you expect:

conditional: if, if else, switch

loop: while, for, do

break and continue (but a little different than with C/C++).

Exceptions

- Terminology:
  - *throw an exception*: signal that some condition (possibly an error) has occurred.
  - *catch an exception*: deal with the error (or whatever).
- In Java, exception handling is necessary (forced by the compiler)!

Try/Catch/Finally

```
try {
  // some code that can throw
  // an exception
} catch (ExceptionType1 e1) {
  // code to handle the exception
} catch (ExceptionType2 e2) {
 // code to handle the exception
} finally {
 // code to run after the stuff in try
 // can handle other exception types
}
```


Exceptions and Network Programming

- Exceptions take care of handling errors
  - instead of returning an error, some method calls will throw an exception.
- A little hard to get used to, but forces the programmer to be aware of what errors can occur and to deal with them.

The synchronized Statement

- Java is multithreaded!
  - threads are easy to use.
- Instead of mutex, use synchronized:

```
synchronized ( object ) {
   // critical code here
}
```


synchronized as a modifier

- You can also declare a method as synchronized:

```
synchronized int blah(String x) {
 // blah blah blah
}
```


Classes and Objects

- “All Java statements appear within methods, and all methods are defined within classes”.
- Java classes are very similar to C++ classes (same concepts).
- Instead of a “standard library”, Java provides a lot of Class implementations.

Defining a Class

- One top level public class per .java file.
  - typically end up with many .java files for a single program.
  - One (at least) has a static public main() method.
- Class name must match the file name!
  - compiler/interpreter use class names to figure out what file name is.

Sample Class

(from Java in a Nutshell)

```
public class Point {
  public double x,y;
  public Point(double x, double y) {
     this.x = x; this.y=y;
  }
  public double distanceFromOrigin(){
     return Math.sqrt(x*x+y*y);
  }
}
```


Objects and **new**

You can declare a variable that can hold an object:

```
Point p;
```

but this doesn't create the object! You have to use new:

```
Point p = new Point(3.1,2.4);
```

there are other ways to create objects…

Using objects

- Just like C++:
  - **`object.method()`**
  - **`object.field`**
- BUT, never like this (no pointers!)
  - **`object->method()`**
  - **`object->field`**

Strings are special

- You can initialize Strings like this:

```
String blah = "I am a literal ";
```

- Or this (+ String operator):

```
String foo = "I love " + "RPI";
```


Arrays

- Arrays are supported as a second kind of reference type (objects are the other reference type).
- Although the way the language supports arrays is different than with C++, much of the syntax is compatible.
  - however, creating an array requires **new**

Notes on Arrays

- index starts at 0.
- arrays can't shrink or grow.
- each element is initialized.
- array bounds checking (no overflow!)
  - ArrayIndexOutOfBoundsException
- Arrays have a *.length*

Reference Types

- Objects and Arrays are *reference types*
- Primitive types are stored as values.
- Reference type variables are stored as references (pointers that we can't mess with).
- There are significant differences!

Primitive vs. Reference Types

```
int x=3;
int y=x;
```

There are two copies of the value 3 in memory

```
Point p = new Point(2.3,4.2);
Point t = p;
```

There is only one Point object in memory!

Passing arguments to methods

- Primitive types: the method gets a copy of the value. Changes won't show up in the caller.
- Reference types: the method gets a copy of the reference, the method accesses the same object!

Packages

- You can organize a bunch of classes into a *package*.
  - defines a namespace that contains all the classes.
- You need to use some java packages in your programs
  - java.lang java.io, java.util

Importing classes and packages

- Instead of **`#include`**, you use **`import`**
- You don't have to import anything, but then you need to know the complete name (not just the class, the package).
  - if you **`import java.io.File`** you can use **`File`** objects.
  - If not – you need to use **`java.io.File`** objects.