Carl Kingsford, 02-201, Fall 2015

Lecture 22: Java

Overall Structure

Classes & Objects

Every function in Java must be inside a `class`, which are similar to Go's `struct`s. For example:

```java
class MyProgram {

    int times = 100;

    public void printError() {
        System.out.println("Must provide a command line argument");
    }

    // this is the main function.
    public static void main(String[] argv) {
        if (argv.length >= 2) {
            for (int i = 0; i < times; i++) {
                System.out.println(argv[1]);
            }
        } else {
            printError();
        }
    }
}
```

You can put each of your classes in a separate file (names `ClassName.java`).

Every statement must end with ';'

You have to end every statement with a ';'

Types

Java and Go have similar types, but with different names:

Java	Go
byte	int8
short	int16
int	int32
long	int64
float	float32
double	float64
boolean	bool
String	string

The special type `void` means: "no type" --- it's used to indicate that a function doesn't return anything.

Lists (aka Arrays)

Lists, called *Arrays* in Java have a type that puts the `[]` after the type of variables included in the array:

```
int[]        // a list of integers
float[]      // a list of floats
String[]     // a list of strings
```

Just like in Go, you have to "make" a list, but in Java it's called `new`:

```
A = new int[100];
```

creates an array of 100 `int` s.

You can create 2-dimensional lists directly:

```
B = new int[100][200];
```

This creates a new 2-dimensional array `B`.

Pointers

Java does not explictly have pointers. All "big" variables are implicitly pointers.

Structs

Java does not have `stuct`s. Instead it has `class`es, like Python.

Variables

You must declare all variables before you use them (as in Go). The syntax is to place the type before the new variable name:

```
int a;
float b;
String c;
int[] d = new int[100];
```

Operators

The basic operators in Java are the same as in Go:

Java	Go	Definition
=	=	assignment
==	==	equals
+,*,-,/	+,*,-,/	math operators
+	+	string concatenation

Functions (methods)

Functions all must live in a class someplace. The syntax is similar to Go:

```java
public int Square(int x) {
    return x*x;
}
```

The main differences: the return type goes *before* the function name; and inside the parameter list, types go before the variables.

Functions also have "permissions":

permission	**meaning**
public	anyone can call
private	only functions inside the same class can call this function
protected	only things in the same package (and subclasses) can call the function

If Statements

The syntax for an `if` statement is nearly the same as in Go, except that `()` are required:

```java
if (a > 10) {   // The () are required
    // do this if a is bigger than 10
} else {
    // otherwise, do this
}
```

Java also has a `switch` statement, that works similarily to, but different than Go (Java's is more like C's):

```java
switch(a) {
case 3: case 5: case 7:
    prime = true;
    break;  // THIS IS REQUIRED
case 2: case 4: case 6: case 8: case 9:
    prime = false;
    break;
default:
    System.out.println("Number too big!")
}
```

The main differences are (1) you have to repeat the `case` keyword, and (2) you MUST end each `case` block with `break` --- otherwise the program will continue executing the next cases.

Loops

for loops

The Java `for` loop is nearly the same as one form of the Go `for` loop:

```
for (int i = 10; i < 100; i++) {     // the () are required
    // do something for i = 10...99
}
```

You can delcare a variable in the first part of the `for` loop. Java doesn't have a `:=` -like operator, so you declare the variable using the normal `TYPE NAME = VALUE` syntax.

If you have an array (or any other variable that can be interated over), you can write a `for ... range` -like version of the for loop:

```
int[] A = new int[100];
for (int v : A) {
    // do something for every element in A
}
```

while loops

Java also has a `while` loop, that works just like a `for` loop but without the initialization or increment statement:

```
int a = 23512;
while (a < 10) {
    a = a - a*a;
}
```

do...while loops

Java also has something that C/C++ has but that Go lacks: a loop that always executes at least once:

```
do {
    fmt.Println(a);
    a--;
} while (a < 10);
```

This will print `a` even if it is `< 10` to start.

Printing

The equivalent of `fmt.Println` is similar:

```
System.out.println("Hi there");
```

Importing packages

Like Go, Java functions are organized into packages. For example, Java has a `Random` class that generates random numbers. It lives in the `java.util.random` package. You can use it in your code in two different ways:

```
java.util.random.Random R = new java.util.random.Random();
```

or, because this is a lot of typing,

```
import java.util.random; /* this goes at the top of
                            your .java file before any
                            class definitions */

// ...

Random R = new Random();
```

Exceptions

When something goes wrong, Java or you can "throw an exception". These errors are then passed back up the call stack until someone handles it. If no one handles it, your program will terminate with an error.

They way you handle an exception is to surround the code that might throw it in a `try...catch` block:

```java
public void bad() throws IndexOutOfBoundsException {
    if (10 < 100) {
        throw new IndexOutOfBoundsException();
    }
}

public void trySomethingBad() {
    try {
        bad();
    } catch(IndexOutOfBoundsException e) {
        System.out.println("Hey, something went wrong... I'll ignore it.");
    }
}
```

If anything inside the `try` block throws an `IndexOutOfBoundsException`, then the code in the `catch` part will be run.

Example: An integer stack

```java

public class Stack {

    /**
     * These are member variables inside of the Stack class.
     */
    private int[] items;  // storage for the stack
    private int numItems; // current # of items in the stack

    /** Creates a new stack with a small amount of
     * storage space.
     */
    public Stack() {
        items = new int[16];
        numItems = 0;
    }

    /** Create a new stack with a user-supplied guess
     * of how big it will get.
     */
    public Stack(int sizeGuess) {
        items = new int[sizeGuess];
        numItems = 0;
    }

    /**
     * Pushes an integer on the stack.
     * @param x - the integer to push onto the stack
     */
    public void push(int x) {
        if (numItems >= items.length) {
            // create a new list
            int[] newList = new int[2*items.length];

            // copy current list over to the new list over
            int i = 0;
            for (int v : items) {
                newList[i] = v;
                i++;
            }
            items = newList;
        }
        // add the item to the end of the list
        numItems++;
        items[numItems-1] = x;
    }

    /**
     * Removes the top item from the stack and returns it.
     * @return the former top of the stack
     * @throws IndexOutOfBoundsException
     */
    public int pop() throws IndexOutOfBoundsException {
        // if we try to pop an empty stack, throw an error
        // (similar to Go's panic)
        if (numItems == 0) {
            throw new IndexOutOfBoundsException();
        }
        int x = items[numItems-1];
        numItems--;
        return x;
    }

    /**
     * Prints the stack items to the console, separated
     * by commas
     */
    public void print() {
        for (int i = 0; i < numItems; i++) {
            // note that items[i] is automatically converted
            // to a string when needed.
            System.out.print(items[i] + ",");
        }
        System.out.println();
    }

    /**
     * A demo usage of the stack.
     * @param args
     */
    public static void main(String[] args) {
        Stack S = new Stack(100);
        S.push(10);
        S.push(20);
        S.push(31);
        System.out.println(S.pop());
        S.push(42);
        S.print();
    }
}
```

Summary

- Java syntax is largely similar to Go
- Every function must live in a class
- Types go before function names and variable names
- "new" is used where "make" is used in Go
- There are numerous other small syntax differences
- Java has extensive support for "object-oriented programming" --- much more than we can cover in this class.