Course Administration

Outline

1. Statements
2. Classes and Objects
3. Subclassing
4. Interfaces
5. Doing OOP in Java
6. Concurrency
7. GUI
8. Fun topics: Applets, Networking, JDBC, or others

Assessment

- Four programming assignments
- One project: Software or Study

Contacts

- Email: `trebla@cs.utoronto.ca`
- Webpage: `www.cs.utoronto.ca/~trebla/javacourse/`
- Office: Somewhere in SF4008

Reference Books

- Ken Arnold and James Gosling, *The Java Programming Language*, second edition. Addison-Wesley, 1997.
- Mary Campione and Kathy Walrath, *The Java Tutorial*, second edition. Addison-Wesley, 1998. `http://java.sun.com/docs/books/tutorial/index.html`
- Cay S. Horstmann and Gary Cornell, *Core Java*, volume 1: Fundamentals, fourth edition. Prentice Hall, 1999.
- David Flanagan, *Java in a Nutshell*, second edition. O'Reilly, 1997.
- Sun Microsystems, *JDK 1.1.x Documentation*. `http://java.sun.com/products/jdk/1.1/docs/index.html`
- Sun Microsystems, *Java 2 SDK Documentation*. `http://java.sun.com/products/jdk/1.2/docs/index.html`

Java Code

It is said that in C everything is a function.

- Whatever program code you write, it must be in some function.

Well, in that sense, in Java everything is a class.

- Whatever program code you write, it must be in some class.
- In fact, the code will go into some function, which in turn will go into the class.

```
public class HelloWorld
{
  public static void main(String[] args)
  {
    System.out.println("Hello world!");
  }
}
```


Executing Java Code

How do we actually enter and run a Java program?

1. Edit the file `HelloWorld.java`:

   ```
   public class HelloWorld
   {
     public static void main(String[] args)
     {
       System.out.println("Hello world!");
     }
   }
   ```

   Note the relation between the filename and the class name.

2. To compile:

   ```
   javac HelloWorld.java
   ```

   The object code will be in `HelloWorld.class`.

3. To run:

   ```
   java HelloWorld
   ```

Simple Output

You can print strings and numbers with `System.out.println`:

```
public class Demo1
{
  public static void main(String[] args)
  {
    System.out.println("A string");
    System.out.println(43);
  }
}
```

The output is

```
A string
43
```


Simple Output (cont.)

`System.out.println` starts a new line after the printing.

If you do not want that, use `System.out.print`:

```
public class Demo2
{
  public static void main(String[] args)
  {
    System.out.print("My number is ");
    System.out.print(43);
    System.out.println(".");
  }
}
```

Output:

```
My number is 43.
```


Data Types

Numbers like 43 and strings like `"Hello world!"` are generally called *literals*.

Each literal has a *data type*. The type of 43 is `int` and the type of `"Hello world!"` is `String`.

Here are some other data types (in fact all of the *primitive* data types):

`boolean`	Either `true` or `false`
`char`	A single character. Example: `'a'`
`byte`	Integer between -2^7 and 2^7-1
`short`	Integer between -2^{15} and $2^{15}-1$
`int`	Integer between -2^{31} and $2^{31}-1$
`long`	Integer between -2^{63} and $2^{63}-1$
`float`	Single-precision floating-point number
`double`	Double-precision floating-point number

FAQ: Why is 43 not a `byte` or a `short`?

Answer: `int` is of the most optimal size (speedwise) for most computers.

Variables and Expressions

Local variables and expressions are just like in C.

```
public class DemoVar
{
  public static void main(String[] args)
  {
    int x;
    String s;
    final char c = 'h';  // named constant
    double r, a = 1.0, b = 2.0, c = 1.0;

    x = 43;
    s = "Hello world!";
    r = (-b - Math.sqrt(b*b - 4*a*c)) / (2*a);
    System.out.println(s + s);
    System.out.println(x);
    System.out.println(c);
    System.out.println(r);
  }
}
```

Operators

Simple Input

Input in Java is a bit hard to use.

You are provided with a SimpleInput library for casual use.

```
public class InputDemo
{
  public static void main(String[] args)
  {
    int i;
    double d;
    String s;

    i = SimpleInput.getInt();
    d = SimpleInput.getDouble();
    s = SimpleInput.getWord();
    System.out.println(i);
    System.out.println(d);
    System.out.println(s);
  }
}
```

Make sure `SimpleInput.java` is in the same directory as your code.

Flow of Control

Branching

```
if (condition) statement
if (condition) statement else statement
switch (int expr) {
  case const:
    statements
    break;
  etc.
  default:
    statements
}
```

Loops

```
while (condition) statement
do statement while (condition)
for (init; condition; next) statement
```


Flow of Control Examples

```
if (D >= 0) {
  x1 = (-b - Math.sqrt(D))/(2*a);
  x2 = (-b + Math.sqrt(D))/(2*a);
}
else
  System.out.println("complex solution");

while (n % m != 0) n /= m;

for (int i = 0; i < 100; i++)
  System.out.println("Do not talk in class");

switch (m) {
case 0:
  System.out.println("none");
  break;
case 1:
  System.out.println("unique");
  break;
default:
  System.out.println("many");
}
```

Advanced Flow of Control

You can exit a loop with the `break` statement.

```
while (n % m != 0) {
  if (n == 1) break;
  n /= m;
}
```

You can skip the rest of a loop and proceed to the next iteration with the `continue` statement.

```
for (i = 0; i < 10; i++) {
  System.out.print("x");
  if (i == 5) continue;
  System.out.print("y");
}
```


Advanced `break` and `continue`

You can `break` or `continue` several levels of nesting in nested loops with *labels*.

```
i_loop:
for (i = 0; i < 10; i++) {
  for (j = 0; j < 10; j++) {
    if (j == 2) continue;
    // stuff
    if (i == 3) continue i_loop;
    // other stuff
  }
}
```

In fact, you can `break` from any compound statement.

```
for (int i = 0; i < 10; i++) {
  good_stuff:
  {
    if (i == 1) break good_stuff;
    System.out.println("good stuff");
  }
}
```


Arrays

```
int[] A;
int B[];
```

These only declare *references* to arrays, not the array themselves. You must allocate the actual arrays.

```
A = new int[3];
B = new int[] { 1, 2, 3 };
int[] C = { 1, 2, 3 };
int[] D;
D = A;
```

Now you can do stuff.

```
for (int i = 0; i < 3; i++) {
  A[i] = B[i];
  System.out.println(D[i]);
}
```

The length of the array is remembered in `A.length`.

```
for (int i = 0; i < A.length; i++)
  A[i] = B[i];
```


Two Dimensional Arrays

```
int[][] C;
```

Then `C` is a reference to an array of `int[]`'s

Then `C` is a reference to an array of references to arrays of `int`'s.

Matrix example:

```
int[][] M = new int[3][];
for (int i = 0; i < M.length; i++)
  M[i] = new int[3];
int[][] N = new int[3][3];
```

Staggered array example:

```
int[][] T = { {1}, {1,1}, {1,2,1} };
```

Uninitialized:

```
int[][] U = new int[3][];
```

Then `U[0]`, `U[1]`, and `U[2]` are all `null`.

Functions

Functions are like in C, with a few extra magic words.

```
public static void main(String[] args)
{
  int x = foo(1, "hello");
}

public static void bar(String s)
{
  System.out.println(s);
  return;
}

public static int foo(int x, String s)
{
  bar(s);
  if (x >= 0) return x+1;
  return x-1;
}
```

Primitive types are passed by value.

All other types (arrays, strings, objects) are passed by reference.

Returning non-Primitive Types

Do not hesitate to return arrays, strings, and objects, since they are also by reference.

Vector addition example:

```
public static double[] add(double[] v, double[] w)
{
  if (v.length != w.length) return null;
  // there is a better way

  double[] s = new double[v.length];
  for (int i = 0; i < v.length; i++)
    s[i] = v[i] + w[i];
  return s;
}
```


The `main` Function

```
public static void main(String[] args)
```

The array contains the command line parameters.

Example of dumping the command line parameters:

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

If you run the program with

```
java DemoCmdLine hehe haha 123
```

the output will be

```
hehe
haha
123
```


Functions from Other Classes

If you use a function from another class, you must mention the class name.

Example: the SimpleInput class provides a function:

```
public class SimpleInput
{
  // stuff
  public static int getInt()
  // stuff
}
```

Then you use it as in:

```
public class MyClass
{
  public static void main(String[] args)
  {
    // stuff
    x = SimpleInput.getInt();
    // stuff
  }
}
```