CS 11 java track: lecture 1

- Administrivia
  - need a CS cluster account
    - http://www.cs.caltech.edu/
      cgi-bin/sysadmin/account_request.cgi
  - need to know UNIX
    - www.its.caltech.edu/its/facilities/labsclusters/
      unix/unixtutorial.shtml
  - track home page:
    - www.cs.caltech.edu/courses/cs11/material/java/mike

prerequisites

- some programming experience
  - CS 1 ideal, not required
- familiarity with C syntax

assignments

- 1st assignment is posted now
- due one week after class, midnight
- late penalty: 1 mark/day
- redos

textbook, online tutorials

- Arnold, Gosling, Holmes:

<u>The Java Programming Language, 3rd. ed.</u>

    - earlier editions NOT acceptable

- java on-line tutorials:
    - http://java.sun.com/docs/books/tutorial/reallybigindex.html
    - very good material!

what is java?

- java is
  - an object-oriented *programming language*
  - a programming *environment*
  - a large set of *libraries* (java API)
  - a philosophy

java philosophy

- programs should be *portable*
  - "write once, run anywhere"
- programs should be *safe*
  - no core dumps, no memory corruption
- programs should be *easy to write and understand*
- programs should be as *efficient* as possible
  - subject to the above constraints

programming in java (1)

- version: java 1.4.2 (on CS cluster)
- programmer writes source code
  - files end in ".java" extension
- java compiler (javac) converts (compiles) source code into "bytecode" (files ending in ".class")
  - bytecode is "machine code" for Java Virtual Machine (JVM)

programming in java (2)

- example:

```
% javac Foo.java
```

→ Foo.class

→ (may compile other files too if "Foo.java" depends on them)

programming in java (3)

- JVM (program name: java) executes bytecode to run the program
- JVM implementations exist for most platforms (Windows, Linux, Mac...)

```
% java Foo
```

- executes bytecode in Foo.class
- can be compiled to machine code on-the-fly

libraries

- java API (application programming interface)
- HUGE set of libraries, including
  - graphics
  - networking
  - database
  - input/output
- http://java.sun.com/j2se/1.4.2/docs/api/index.html

the java language (1)

- "object oriented"
- object: data + functions acting on that data
- class: template for building objects; includes
  - data (fields) that every object contains
  - functions (methods) that can act on the object
- objects are instances of a particular class

the java language (2)

- all data is either
  - an object *i.e.* an instance of some class
  - a primitive data type
    - int
    - float, double
    - char
    - boolean

the java language (3)

- java is *strongly, statically typed*
  - strongly typed: all data has a type
  - statically typed: all types must be declared before use
- type declarations can occur anywhere in source code

```
int foo;  // foo has type int
```

the java language (4)

- methods have
  - a name
  - a set of arguments with their types
  - a return type
  - some optional modifiers
- methods written inside class definition
- methods have implicit extra argument: the object they're part of (called this)

"hello world" program (1)

- in file "HelloWorld.java":

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

"hello world" program (2)

- class definition:

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

- file must be called "HelloWorld.java"

"hello world" program (3)

- method definition:

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

"hello world" program (4)

- method name:

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

- program always starts executing with main

"hello world" program (5)

- method arguments:

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

- String[] = array of strings (command line args)

"hello world" program (6)

- method return type:

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

- void means "doesn't return anything"

"hello world" program (7)

- method modifiers:

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

- we'll discuss these later

"hello world" program (8)

- method body:

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

- print "Hello, world!" to the terminal (System.out)

"hello world" program (9)

- compile:

```
% javac HelloWorld.java
```

➔ HelloWorld.class

- run:

```
% java HelloWorld
Hello, world!
%
```

data types

- int → integers
- float → single precision floating point
- double → double precision floating point
- char → Unicode characters (16 bit)
- boolean → true or false (not 0 or 1)
- byte → 8 bits; "raw data"
- String → character strings

operators

- like in C:
  - + - * / % = ++ -- += -= etc.
- precedence:
  - a + b * c → a + (b * c) NOT (a + b) * c
  - use parentheses if need to override defaults

comments

- three kinds:

```
// This comment goes to the end of the line.
/* This comment can span
 * multiple lines. */
/**
 * This comment is for documentation.
 */
```

conditionals

- **if / else if / else** like in C:

```
int i = 10;
if (i < 20) {
    System.out.println("less than 20");
} else if (i == 20) {
    System.out.println("equal to 20");
} else {
    System.out.println("greater than 20");
}
```

loops (1)

- **for** and **while** loops like in C:

```
int i;
for (i = 0; i < 10; i++) {
   // do something with i
}
while (i < 20) {
   // do something with i
   // increment i
}
```

loops (2)

- can declare types at first use:

```
for (int i = 0; i < 10; i++) {
  // do something with i
}
```

- now "i" only usable inside the loop
- judgment call; usually the right thing to do

that's all for now

- this is enough for 1st assignment
- lots more to come!