Imperative Languages

This material covered in Chapter 4 of the text

- We're not going to cover everything in this chapter – these slides indicate the material that you should know for the course.

Turing Completeness

- A Turing complete language is one that has computational power equivalent to a Turing machine.
- No programming language run on an actual, physical computer can completely meet this requirements, since a Turing machine has unlimited storage capacity.
- Except for this, any reasonable programming language is Turing complete.
  - Corollary: it's not interesting to compare whether you can compute more things in one programming language than another.
- (Note: book's definition of Turing complete is completely bogus – although in fact Jay is Turing complete. Take CSE 322.)

Names in Programming Languages

- Java uses *reserved words*, such as while, if, int, etc.
- You can't declare a variable named while, or redefine int.
- The alternative (predefined variables, no reserved words) is more confusing and error-prone.
- For example, this is legal PL/I:

```
IF IF>THEN THEN THEN=0;
ELSE;
THEN = 1;
```


Primitive types in Java (was also in Java slides)

- boolean
- char (16-bit) //unicode
- byte (8-bit signed)
- short (16-bit signed)
- int (32-bit signed) } Integer types
- long (64-bit signed) }
- float (32-bit signed) } Floating point types
- double (64-bit signed) }

Maximum and minimum values

- The maximum and mimimum values for Java's primitive types are available from static methods in the wrapper classes:
  - Float.MAX_VALUE (largest positive floating point value)
  - Float.MIN_VALUE (smallest positive value)
  - Byte.MAX_VALUE (largest byte value, i.e. 127)
  - Byte.MIN_VALUE (smallest byte value, i.e. -128)
  - etc.

Overloading and overriding

- Operators (+, -, *, …) in Java and most other languages are overloaded. Thus, there are actually several different + operations, and the compiler decides which one is needed based on the types of the arguments
  - i+j (for integers i and j)
  - x+y (for floats x and y)

Overloading and overriding (2)

- Method names in Java can also be overloaded. For example:
  - println(int I)
  - println(float x)
  - println(Object a)
  - … etc
- Overriding is a separate concept – we can override an inherited method from a superclass with a method with the same signature in the subclass
  - equal(Object x) in class Object
  - equal(Object x) in class Point

Coercion

- In Java (as in many other languages) the programmer can write mixed mode expressions, containing different numeric types.
- If there wouldn't be a loss of information, Java will automatically coerce some of the values so that the types are the same. Example:
  - i + x for int i and float x
  - This is the same as (new Integer(i).floatValue()) + x
- You get a compile-time error if information would be lost. Example:
  - f = d for float f and double d will give a compile-time error

Scope

- Java (and C, C++, Ada, Pascal, Modula, Scheme, Haskell, etc) all use static scoping.
  - Also called lexical scoping
  - Which variable is referenced depends on the static structure of the program, not where the procedure was called from
- A variable is *global* if it is declared in the outermost scope, and *local* if it is declared in the current scope.
  - "non-local" means (surprise) "not local"

Scope Example 1

This is legal Java code (and considered good style):

```
public class Test {
  int x;

  void set(int x) {
    this.x = x;
    }
}
```


Scope Example 2

This is also legal Java code (although anyone who wrote this deserves an unpleasant fate):

```
public class Test {
  int x;

  void set (int x) {
    this.x = x;
    }

  class Inside {
    int x;
    void set(int x) {
      this.x = x;
    }
  }

}
```


Scoping (to be continued)

- We'll return to a more complete discussion of scoping when we get to Scheme
- Static scoping is very general and elegant in Scheme – it is made complicated in Java by having several different kinds of variables (static fields, ordinary fields, method parameters, local variables, variables in inner classes)