

Java Basics

Compiling

- A “compiler” is a program that translates from one language to another.
- typically from easy-to-read to fast-to-run
- eg. from C or Fortran to assembly code
- Java must be (explicitly) compiled before it can be run.
- The Java compiler turns Java source code (.java) into Java bytecode (.class).

The Java Platform

- The Java Virtual Machine (JVM) is responsible for running Java bytecode.
- The idea: bytecode can be interpreted quickly.
- The same bytecode can be interpreted on any machine architecture: write once, run anywhere.
- Code (C, C++) compiled to machine code is specific to an architecture (Windows, Mac, Linux, ...)
 - must be recompiled for each system

The Java Language

- ... is a high-level programming language
- ... is very object-oriented.
- ... is similar to C++ and C.
- ... typically compiled to Java bytecode
- ... is often confused with the Java Platform, but these are two distinct aspects of “Java”.

Hello World

```
// hello.java
```

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

Hello World

```
// hello.java
```

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

- first line is a comment. These are comments in Java:

```
// comment  
/* comment */
```

- creates a “class” called `HelloWorld`.
 - generates a `.class` file when compiled (`HelloWorld.class`)
 - classes are used to create objects... later.
 - wrapped in curly braces: `{ ... }`

Hello World

```
// hello.java
```

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

- The `main` function is where it starts when run.
 - ignore “`public static void`” and “`String[] args`” for now.
- contains one “statement”
 - prints the text “`Hello World!`” on the screen
 - the `System.out.println` function comes from the Java “class library”
 - ends with a semicolon: all statements in Java do.

Compiling & Running

- Create the program `hello.java` in a text editor.

- Compile:

```
javac hello.java
```

- Run:

```
java HelloWorld
```

- Output:

```
Hello World!
```


Strong Typing

- Java is a “strongly typed” language.
- All variables and values have a specific type.
- The type is known when the program is compiled: *before* it’s run.
- So, all variables must be declared as having a particular type.
- declaration must occur before the variable is used.

Declaring Variables

- syntax for a variable declaration:

`<variable declaration> ::= <type> <declarator>, <declarator>, ...;`

`<declarator> ::= <identifier>`

`<declarator> ::= <identifier> = <expression>`

- built-in types: `int`, `float`, `String`.
 - others can be defined (as we'll see later)
- The optional expression is used to initialize the variable.
- examples:

```
float length;  
int count = 0;  
String course1 = new String("CMPT 125");
```

Primitive Types

- `int`: a subset of the integers from -2^{31} to $2^{31}-1$
- `double`: a subset of the reals, with approx. 15 significant digits
- `boolean`: either `true` or `false`.
- ...and five others.
- some operators are defined on the primitive types.
 - eg. `+` and `-` for `int` and `double`, `&` for `boolean`

Variable Assignment

- syntax for variable assignment statement:

$\langle \text{assignment statement} \rangle ::= \langle \text{identifier} \rangle = \langle \text{expression} \rangle ;$

- The type of the expression result must match the variable type.
- If not, it can be converted...

Type Conversion

- Non-matching types can be converted.
- A “widening conversion” is automatic.
- A “narrowing conversion” must be done explicitly.
 - a “cast operator” indicates the conversion
 - ... because it might lose information.
- eg.

```
long longint = 10;
int regint = 20;
longint = regint;           // widening
regint = (int) longint;    // narrowing
```

Object Types

- There aren't many primitive types in Java
- Other types are object types or “classes”.
 - typically Capitalized (primitives are lower case)
- Object variables always hold *references* to an object.
 - The declaration only creates a reference, eg.
`String course;`
- No `String` exists in memory, just a null reference.

Object Instances

- We must create a new “instance” of the object type to store something.
 - Each object type has a “constructor”.
 - The constructor creates an “instance” of the type.


```
course = new String("CMPT 125");
```
- This creates a new `String` in memory;
 - ... stores the eight characters “CMPT 125”;
 - ... and the assignment sets `course` to refer to this instance.

References and Instances

```
String course;
```

```
course: 
```

```
new String("CMPT 125")
```



CMPT 125

```
course = new String("CMPT 125")
```

```
course:   
```

CMPT 125