CMPT 125: Lecture 2
Introduction to Java

Tamara Smyth, tamaras@cs.sfu.ca
School of Computing Science,
Simon Fraser University

January 3, 2009

Brief History of Java

- Java was developed by James Gosling and his team at Sun Computers in the early 1990s.
- Originally called Oak, later given the name Java:
  - inspired by a coffee bean?
  - or an acronym for the names of the team members: **J**ames Gosling, **A**rthur **V**an Hoff, and **A**ndy Bechtolsheim?
- Though now often thought of as a program designed for the world wide web, it was originally meant to be a programming language for embedded systems.
- As the web gained in popularity in 1994, the HotJava browser was a good demo of Java's capabilities: platform independence and security.
- Java released publicly in 1995.

Java Versions

- We will use Java version 1.5 (5.0) which includes some major enhancements to previous versions of Java (so watch out for this if you're using older textbooks!).
- Regularly consult the following website (keep it bookmarked!) for the API specification: http://java.sun.com/j2se/1.5.0/docs/api/.

Some Java Details

- Java syntax is (purposely) similar to C/C++ syntax.
- Java supports the object-oriented programming paradigm (OOP).
- Because of OOP, some simple programs tend to be overly complex in Java.
- Java was not designed with beginners in mind, but it is arguably easier to learn than C/C++.
- Java uses a garbage collector to help manage memory.
- For some applications Java is a little slower than C/C++ but tends to be much faster than (say) Python.

Translating and Executing Java Programs

- All source code is first written in plain text files ending with the `.java` extension.
- Source files are then compiled to produce `.class` files.
- A `.class` file does not contain code that is native to your processor. Rather, it contains **bytecode**, the machine language of the Java Virtual Machine (Java VM).
- The java launcher tool then runs your application with an instance of the Java VM.

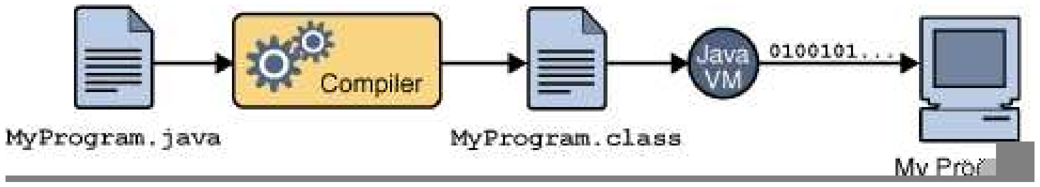

Figure 1: MyProgram.java is compiled to produce MyProgram.class, which is then interpreted by the Java VM before My Program runs on a computer.

Platform Independence

- Because the Java VM is available on many different operating systems, the same `.class` files can run on different computers without recompiling.
- The Java HotSpot virtual machine, perform additional steps at runtime to give your application a performance boost, including recompiling some frequently used code to *native* machine code.

Compiling and Running Java

- To compile `DrawWin.java` from the command line, type

  ```
  C:\> javac  DrawWin.java
  ```

  to produce a `DrawWin.class` file.

- To run `DrawWin.class`, from the the command line, type

  ```
  C:\> java DrawWin
  ```

  which will call the JVM and runs `DrawWin.class`.

Integrated Development Environment—Eclipse

- Java programs tend to use many files (one file per class), and managing all these files can get tricky.
- We will, therefore, use the Eclipse integrated development environment (IDE) to write our Java programs
- Eclipse is a modern, powerful Java IDE and can be found at: http://www.eclipse.org/.
  - It compiles as you type!
  - Errors are pointed out immediately through wavy red underlines
  - It provides automatic formatting, auto-fixing of code, intelligent searching and code modification, and many sophisticated tools for managing large projects

A Java Program

PROGRAM CODE:

```
/*
 *  WiseWords.java   CMPT 125 Sample code
 *                   (by Toby Donaldson)
 */

public class WiseWords {

   public static void main(String[] args) {

      // print a wise saying in the console window
      System.out.println("No matter where you go ...");
      System.out.println("... there you are.");
   }
}
```

OUTPUT:

```
No matter where you go ...
... there you are.
```

Understanding a WiseWords

- The first three (3) lines of the program are *comments*,

```
/*
 *  WiseWords.java   CMPT 125 Sample code
 */
```

- The rest is a *class definition*.

```
public class WiseWords {

   public static void main(String[] args) {

      // print a wise saying in the console window
      System.out.println("No matter where you go ...");
      System.out.println("... there you are.");
   }
}
```

- The name of this class is **WiseWords** (though you may give a class any name you like), and its **definition** runs from the first opening brace '{' to the final closing brace '}'.

Inside the Class Definition

- All Java programs are defined using **class definitions** which contain the description of functions/methods and variables.
- Inside `WiseWords` there is a single *method* called `main`:

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

   // print a wise saying in the console window
   System.out.println("No matter where you go ...");
   System.out.println("... there you are.");
}
```

  which, like the class is delimited by by curly braces.

- Generally, classes can contain any number of methods, variables, and even other classes.

The `main` method

- All Java programs must have a method named `main`:

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

- The Java Virtual Machine (JVM) automatically calls the `main` method to start your program.
- Each statement of the `main` method is executed in the order it occurs, (with the exception of control structures such as loops or conditional statements).

  First statement executed:

  ```
  System.out.println("No matter where you go ...");
  ```

  Last statement executed:

  ```
  System.out.println("... there you are.");
  ```

Parsing the `main` Header

- The **method header** for the `main` method is given by

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

  with the **method name**, $\boxed{\texttt{main}}$, always being preceded by the words $\boxed{\texttt{public}}$, $\boxed{\texttt{static}}$, $\boxed{\texttt{void}}$, and including the **input argument** $\boxed{\texttt{String[] args}}$.

  – $\boxed{\texttt{void}}$ is the **return type**. All methods must return a **type** of value. If no value is returned, the return type is `void`.

  – $\boxed{\texttt{static}}$ essentially says that `main` is part of the class, and not a member of an object (more on this when we discuss the difference between classes and objects).

  – $\boxed{\texttt{public}}$ specifies that other methods have permission to call this method. This gives the JVM the ability to call `main`.

Input Parameter for `main`

- A `main` method always takes in a single **input parameter** specified by

  ```
  String[] args
  ```

  which is an array of strings[1].
- If the program `WiseWords` is run from a console command-line,

  ```
  >> WiseWords candy is dandy
  ```

  then `args` will hold any string appearing after the program name, which in this case is

  $\boxed{\texttt{candy is dandy}}$.
- Whether or not you use `args`, the `main` method must always include it as a parameter.

[1]This is not generally true of all methods. Methods may have several input parameters, or alternatively, none at all.

Parsing the `main` body

- The **method body** is given by

  ```
  {
  // print a wise saying in the console window
  System.out.println("No matter where you go ...");
  System.out.println("... there you are.");
  }
  ```

- The body of a method is a block of code, and so must begin with an open brace, and end with a matching closing brace.
- The JVM executes the statements of a code block in the order they occur, unless a *control structure* such as a loop or if-statement specifies otherwise.
- Every statement in Java must end with a semi-colon.
- The line

  ```
  // print a wise saying in the console window
  ```

  is another way of inserting a comment.

`main` **Method in** `WiseWords`

- The `WiseWords` class has only one method, `main`, from which it *invokes*, or calls, another method, $\boxed{\texttt{println}}$ ("print line"):

```
public static void main(String[] args) {
    // print a wise saying in the console window
    System.out.println("No matter where you go ...");
    System.out.println("... there you are.");
}
```

- The `println` method is from another object called `System.out`, which is part of the Java standard class library.
- The `println` method prints the specified *character string*, enclosed in double quotes (`"`), to the screen.

Comments

- Comments provide insight into the programmer's original intent. Anything that is "commented out", is not seen by the compiler and exists only for the benefit of the programmer.
- Comments may take on two different forms:
  1. Multi-line comments begin with '/*' and end with '*/'

     ```
     /*
      This is a longish comment that spans multiple
      lines.  Comments in this form can provide a
      good deal of information about the code.  You
      may consider using this comment style to head
      your code, providing details of authorship,
      when and where the code was written, and perhaps
      even for what  purpose---such as a CMPT 125
      assignment!
     */
     ```

  2. Single line comments begin with // and continues to the end of line

     ```
     // This is a comment
     ```

White Space

- Whitespace is used to separate the words and symbols used in a program. It consists of blanks, tabs, and newline characters.
- Whitespace does not matter in Java (except in that it separates words). Though appropriate use

```
              can

    make                       a program

          easier            to

read                and

       understand!
```

Programming Language Levels

- Programming languages can be categorized into four groups:
  1. **machine language**:
     - the language to to which all programs must be translated to run on a specific type of CPU (a 'mac' program won't run on a 'PC').
     - expressed as a series of binary digits, and extremely difficult for humans to read and write.
  2. **assembly language**:
     - replaced binary digits of machine language with *mnemonics*, short English-like words that represent commands or data.
     - must be translated to machine language
  3. **high-level language**:
     - expressed in English-like phrases and easier to read and write
     - Java, C, C++, etc. are high level languages.
  4. **fourth-generation language**:
     - an even higher level language, with "special facilities".

Program Development

- "Programming" involves various activities:
  1. **coding**: writing the program in a chosen language, such as Java.
  2. **compiling**: the program much be translated into a form the compute "understands".
  3. **debugging**: programming errors (bugs) must be fixed.
- Software tools are available to help us in the process.

Software Tools

- **Editors**:
  - The tool with which you write and edit your programs. Familiarity, with your editor can greatly increase the speed at which you write and edit programs.
- **Compiler**:
  - The program that translates code in one language, the *source code*, to equivalent code in another language, the *target code*.
  - For many traditional compilers, the source code is translated directly to machine language.
- **Interpreter**:
  - Similar to a compiler, but *combines* the translation and execution activities, eliminating a separate compilation phase.
  - One small part of code, such as a code statement, is translated and executed at a time, making it run more slowly.

Identifiers and Reserved Words

- Identifier are certain words used when writing programs consisting of
  1. **Words that we invent** (`WiseWords`, `args` etc.).
  2. **Words chosen by another developer** (`String`, `System`, `out`, `println` and `main`:
     - These words are not part of the Java language but rather the Java standard library of predefined code: a set of classes and methods that have been written to make our job easier.
  3. **Words reserved for special purposes** (`class`, `public`, `static`, and `void`):
     - *Reserved Words* are identifiers that have a specific meaning and can only be used in predefined ways. They cannot by used to name a class or method. See JSS p. 32 for a complete list.

Rules for Creating Java Identifiers

- An identifier may begin with
  1. any "Java Letter" (any one of 26 English alphabetic characters)
  2. either upper or lower case
  3. the '$' and the underscore '_'.

Identifier	Permitted
`total`	✓
`label17`	✓
`$amount`	✓
`4th_type`	×

- The remaining characters, and order, may be freely chosen.
- Identifiers may be any length.
- Identifiers are sensitive. The following are all considered different:
  1. `taxRate`
  2. `TaxRate`
  3. `taxrate`

Errors

There are 3 different types of errors you will likely encounter:

1. **compile-time error:**
   - Syntax errors: Every language has a syntax dictating the rules of how the language elements are combined to be meaningful. A program must my syntactically correct or the compiler will not produce bytecode.
   - the compiler will also look for incompatible data types.
2. **run-time error:**
   - Will cause the program to terminate abnormally. Eg: a division by zero causes a crash
   - In Java, run-time errors are called exceptions.
3. **logical error:**
   - software compiles and executes without complaining, but it produces incorrect results.

- The process of finding errors in a program is called *debugging*.

Object Oriented Programming Principles

- *object*: a fundamental element in a program with defining characteristics and associated activities or behaviours. Usually more complex then *primitive data*.
- *class*: definition of an object, the blueprint from which an object is created.
- *attribute*: internal values, consisting of both primitive data and other objects, defining the current state of the object.
- *method*: a group of statements that is given an name which when invoked, executes the statements.
- *encapsulation*: the ability for the object to protect and manage its own information.
- *inheritance*: if classes share similarities, one may be created by inheriting characteristics from the other (called a parent class).
- *polymorphism*: allowing reference to different types of related objects at different points in time.