Lab # 1

Introduction to Java

Lab Engr. Fazlullah Khan

CS Dept, Military College of Signals
NUST Rawalpindi

Lecture Objectives

- Learn about Java basics.
- Know the usage of Java programming language.
- Designing and running Java programs.

Java Development Kit (JDK)

- Sort of software development kit that allows to create software applications
- NETBEANS/JBUILDER is Integrated development environment

Chapter 2: Introduction to Java Applications

Outline

2.1 Introduction

- In this chapter
  - Introduce examples to illustrate features of Java
  - Two program styles - applications and applets

How to run program

- Create a batch file which contains the following line:
  - set PATH=C:\Program Files\Java\jdk1.6.0\bin;
- Now you have set the path. You can use the statements javac and java.
- javac statement compiles the code into byte code and generates the .class file.
- java statement executes the code.

2.2 A First Program in Java: Printing a Line of Text

- Application
  - Program that executes using the `java` interpreter
- Sample program
  - Show program, then analyze each line

`Welcome.java` Program Output

```
Welcome to Java Programming!
```

```
1  // Fig. 2.1: Welcome1.java
2  // Text-printing program.
3
4  public class Welcome1 {
5
6     // main method begins execution of Java application
7     public static void main( String args[] )
8     {
9        System.out.println( "Welcome to Java Programming!" );
10
11    } // end method main
12
13 } // end class Welcome1
```


2.2 A First Program in Java: Printing a Line of Text

```
1  // Fig. 2.1: Welcome1.java
```

- Comments start with: `//`
  - Comments ignored during program execution
  - Document and describe code
  - Provides code readability
- Traditional comments: `/* ... */`

```
/* This is a traditional
   comment. It can be
   split over many lines */
```

```
2  // Text-printing program.
```

- Another line of comments
- Note: line numbers not part of program, added for reference

2.2 A Simple Program: Printing a Line of Text

```
3
```

- Blank line
  - Makes program more readable
  - Blank lines, spaces, and tabs are white-space characters
    - Ignored by compiler

```
4  public class Welcome1 {
```

- Begins class declaration for class `Welcome`
  - Every Java program has at least one user-defined class
  - Keyword: words reserved for use by Java
    - `class` keyword followed by class name
  - Naming classes: capitalize every word
    - `SampleClassName`

2.2 A Simple Program: Printing a Line of Text

```
4  public class Welcome1 {
```

- Name of class called identifier
  - Series of characters consisting of letters, digits, underscores (_) and dollar signs ($)
  - Does not begin with a digit, has no spaces
  - Examples: `Welcome1`, `$value`, `_value`, `button7`
    - **`7button`** is invalid
  - Java is case sensitive (capitalization matters)
    - `a1` and `A1` are different

2.2 A Simple Program: Printing a Line of Text

```
4  public class Welcome1 {
```

- Saving files
  - File name must be class name with `.java` extension
  - `Welcome.java`
- Left brace {
  - Begins body of every class
  - Right brace ends declarations (line 13)

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

- Part of every Java application
  - Applications begin executing at `main`
    - Parenthesis indicate `main` is a method
    - Java applications contain one or more methods

2.2 A Simple Program: Printing a Line of Text

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

- Exactly one method must be called `main`

– Methods can perform tasks and return information
  - `void` means `main` returns no information
  - For now, mimic `main`'s first line

```
8    {
```

– Left brace begins body of method declaration
  - Ended by right brace } (line 11)

2.2 A Simple Program: Printing a Line of Text

```
9       System.out.println( "Welcome to Java Programming!" );
```

– Instructs computer to perform an action
  - Prints string of characters
    - String - series characters inside double quotes
  - White-spaces in strings are not ignored by compiler

– `System.out`
  - Standard output object
  - Print to command window (i.e., MS-DOS prompt)

– Method `System.out.println`
  - Displays line of text
  - Argument inside parenthesis

– This line known as a statement
  - Statements must end with semicolon ;

2.2 A Simple Program: Printing a Line of Text

```
11   } // end method main
```

– Ends method declaration

```
13 } // end class Welcome1
```

– Ends class declaration
– Can add comments to keep track of ending braces
– Lines 8 and 9 could be rewritten as:
– Remember, compiler ignores comments
– Comments can start on same line after code

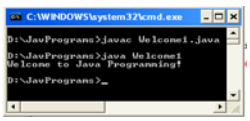
2.2 A Simple Program: Printing a Line of Text

- Compiling a program
  – Open a command prompt window, go to directory where program is stored
  – Type `javac Welcome1.java`
  – If no errors, `Welcome1.class` created
    - Has bytecodes that represent application
    - Bytecodes passed to Java interpreter

2.2 A Simple Program: Printing a Line of Text

- Executing a program
  – Type `java Welcome1`
    - Interpreter loads `.class` file for class `Welcome`
    - `.class` extension omitted from command
  – Interpreter calls method `main`

Executing Welcome1 in a Microsoft Windows Command Prompt.



`public static void main`

- **`public`**: This method can be accessed (called) from outside the class.
- **`static`**: This method does not require that an object of the class exists. static methods are sort-of like "global" methods, they are always available (but you have to use the class name to get at them).
- **`void`**: no return value.

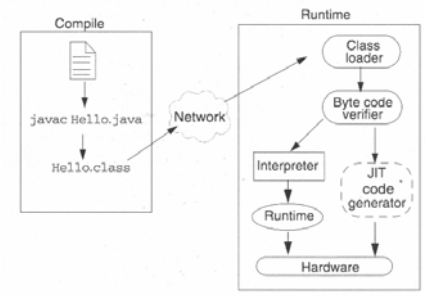
`main( Strings[] arg)`

- **`main()`** will be passed an array of Strings corresponding to any command line parameter values.
- If you ran a program (class) like this:

  `java ShowArgs hi there dave`
- then **`ShowArgs.main()`** would be passed an array (with length 3) of String objects. The values would be "hi", "there" and "dave".
  - **`arg[0]`** is the String "hi", …

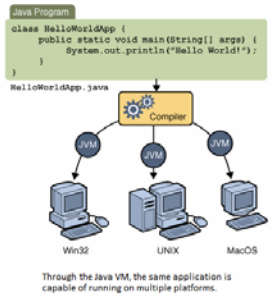
Show Command Line Program

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



A Fact about Java

Write once, Compiled once, run any where!



SDK Tools

- **`javac`**: the Java compiler.
  - Reads source code and generates bytecode.
- **`java`**: the Java interpreter
  - Runs bytecode.
- **`jar`**: Java Archive utility
- **`javadoc`**: create documentation from code.
- **`jdb`**: Java debugger (command line).
- There are others…

The Java Compiler

- Usage: **`javac *filename*.java`**
  - You can also do: **`javac *.java`**
  - Creates ***`filename`*`.class`** (if things work)
  - Use "-g" to compile for use with the debugger.

The Java Interpreter

- Usage: **`java classname`**
  - You tell the interpreter a class to run, not a file to run!
    - It uses the CLASSPATH to find the named class.
    - The named class should have a method with prototype like:
      - **`public static void main()`**

jar

- Like Unix tar comand.
- Used to create (and extract from) an archive file:
  - collection of files.
  - compressed.
- Java can find classes (bytecode) that are stored in jar files.

jar usage

- To extract files:

```
jar xf filename.jar
```

- To list files:

```
jar tf filename.jar
```

- To create and archive:

```
jar cf filename.jar file1 file2 dir1 dir2
                    …
```


javadoc

- Creates documentation from properly commented Java source code.
- The output of javadoc includes HTML files in the same format as the Java SDK documentation.
  - we all need to get used to this format…
  - learning to find and understand the documentation on classes/methods is 1/2 of learning Java!

2.3 Modifying Our First Java Program

- Modify example in Fig. 2.1 to print same contents using different code

2.3 Modifying Our First Java Program

- Modifying programs
  - Welcome2.java (Fig. 2.3) produces same output as Welcome.java (Fig. 2.1)

```
9         System.out.print( "Welcome to " );
10        System.out.println( "Java Programming!" );
```

```
1  // Fig. 2.3: Welcome2.java
2  // Printing a line of text with multiple statements.
3
4  public class Welcome2 {
5
6     // main method begins execution of Java application
7     public static void main( String args[] )
8     {
9        System.out.print( "Welcome to " );
10       System.out.println( "Java Programming!" );
11
12    } // end method main
13
14 } // end class Welcome2
```

`System.out.print` keeps the cursor on the same line, so `System.out.println` continues on the same line.

```
Welcome to Java Programming!
```


2.3 Modifying Our First Java Program

- Newline characters (`\n`)
  - Interpreted as "special characters" by methods `System.out.print` and `System.out.println`
  - Indicates cursor should be on next line
  - `Welcome3.java` (Fig. 2.4)

```
9        System.out.println( "Welcome\nto\nJava\nProgramming!" );
```

  - Line breaks at \n
- Usage
  - Can use in `System.out.println` or `System.out.print` to create new lines
    - `System.out.println( "Welcome\nto\nJava\nProgramming!" );`


```
1  // Fig. 2.4: Welcome3.java
2  // Printing multiple lines of text with a single
statement.
3
4  public class Welcome3 {
5
6     // main method begins execution of Java application
7     public static void main( String args[] )
8     {
9        System.out.println( "Welcome\nto\nJava\nProgramming!" );
10
11    } // end method main
12
13 } // end class Welcome3
```

```
Welcome
to
Java
Programming!
```


2.3 Modifying Our First Java Program

Escape characters

- Backslash (\ )
- Indicates special characters be output

Escape sequence	Description
`\n`	Newline. Position the screen cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor at the beginning of the current line; do not advance to the next line. Any characters output after the carriage return overwrite the characters previously output on that line.
`\\`	Backslash. Used to print a backslash character.
`\"`	Double quote. Used to print a double-quote character. For example, `System.out.println( "\"in quotes\"" );` displays `"in quotes"`

Fig. 2.5 Some common escape sequences.

Welcome1.java - Notepad

File Edit Format View Help

```
public class Welcome1
{

        // main method begins execution of Java application
        public static void main( String args[] )
        {
           System.out.println( "Welcome to\rJava \nProgramming!" );

           System.out.println( " \" Wlecome to javac \" " );

     } // end method main

  } // end class Welcome1
```