Hello, World!

Mitsu Ogihara

Department of Computer Science
University of Miami

Table of Contents

What Is a Computer Program?

A **program** is a list of **instructions**, which are actions to be performed on the machine on which the program is running

A program embodies an **algorithm**, which an abstract idea about how to solve the task at hand

An **instruction** can be described at various levels of specificities to the machine on which the program is to run

The Digital Number Representation

In programming environments, the numbers are often represented using either the binary encoding or the hexadecimal encoding

- **Binary Encoding**: the denominations are powers of 2 and at each position the number is either 0 or 1
- **Hexidecimal Encoding**: the denominations are powers of 16 and at each position the number is one of 0 .. 9 and A .. F, where A - F correspond to 10 - 15

Digital Number Examples

Decimal	Binary	Hexadecimal
0	0	0
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1100	E
15	1111	F
99	1100011	63
100	1100100	64
127	1111111	7F
128	10000000	80

Memory Sizes

A group of 8 bits is called a **byte**
Memory units are counted in thousands (or 2^{10})

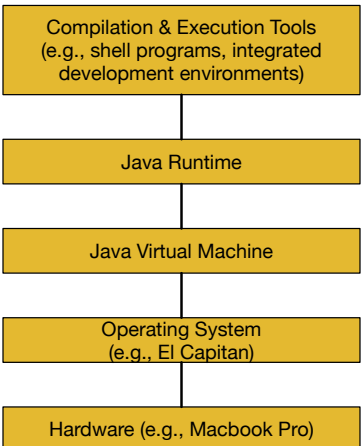
Name	2 to the power of
Kilobytes	2^{10}
Megabytes	2^{20}
Gigabytes	2^{30}
Terabytes	2^{40}
Petabytes	2^{50}

Java Virtual Machine

Java Virtual Machine (JVM) is a program that provides a machine-like environment in which to run **Java bytecodes**, the kind of program exclusively designed for Java

To run Java bytecodes you need **Java runtime**, a program that runs on JVM

The Software Layers



Advantages of Using Java

1. A large number of Java libraries that can be used in other Java programs
2. Much information and help available on the Internet (e.g., Stack Exchange)

Java Coding Process

1. Use a text editor to write the source code (with the file extension of ".java")
2. Generate the Java bytecode (with the file extension of ".class") from the source code
3. Execute the bytecode

Java Coding Process

1. Use a text editor to write the source code (with the file extension of ".java")
2. Generate the Java bytecode (with the file extension of ".class") from the source code
3. Execute the bytecode

- IDEs combine the first two steps into one
- In a shell environment, the second step is accomplished by executing a line command **javac**

Table of Contents

The First Program: HelloWorld.java

The source code of `HelloWorld`

```
1  public class HelloWorld {
2    public static void main( String[] args ) {
3      System.out.println( "Hello, World!" );
4    }
5  }
```

The First Program: HelloWorld.java

The source code of `HelloWorld`

```
1  public class HelloWorld {
2    public static void main( String[] args ) {
3      System.out.println( "Hello, World!" );
4    }
5  }
```

A **class** is a building block of Java programs
This code specifies that “HelloWorld” is a class

Outcome of Execution

In a shell environment you type a command to execute

By the command `javac <name>.java`, you compile the code for class <name>

By the command `java <name>`, you execute the bytecode for class <name>

Outcome of Execution

In a shell environment you type a command to execute

By the command `javac <name>.java`, you compile the code for class <name>

By the command `java <name>`, you execute the bytecode for class <name>

```
% javac HelloWorld.java
% java HelloWorld
Hello, World!
%
```

Here % is the shell's output to indicate the start of a new command, called **prompt**

Structure of a Class

```
public class <name> {
  <method>
  <method>
  ...
  <method>
}
```

A **method** is a named block of code

The `main` in the example is a method has attributes `public`, `static`, and `void` and has a parameter `String[] args` to run with

The class name should be followed by a pair of curly brackets in which the contents appear

Each method name followed by a pair of curly brackets in which the contents appear

System.out.println

- `System.out.println` is a method that produces a line of output on the screen by printing the thing inside the parentheses that follow
- A character sequence when sandwiched by a pair of double quotation marks is called a **String literal**

Structure of a Method

```
public static void main( String[] args ) {
  <statement>
  <statement>
  ...
  <statement>
}
```

A **statement** is an executable unit

A Program with Multiple Statements

```
1  public class HelloWorld2 {
2    public static void main( String[] args ) {
3      System.out.println( "Hello, World!" );
4      System.out.println( "Hello, Class!" );
5      System.out.println();
6      System.out.println( "This is the fourth line of output." );
7    }
8  }
```

A Program with Multiple Statements

```
1  public class HelloWorld2 {
2    public static void main( String[] args ) {
3      System.out.println( "Hello, World!" );
4      System.out.println( "Hello, Class!" );
5      System.out.println();
6      System.out.println( "This is the fourth line of output." );
7    }
8  }
```

The program has four statements, which are executed in the order of appearance

Statements must end with a semicolon

System.out.print

System.out.print is a cousin to System.out.println that does not complete the line . . . the output ends at the last character

```
1  public class RowRow {
2    public static void main( String[] args ) {
3      System.out.print( "Row, " );
4      System.out.print( "row, " );
5      System.out.println( "row your boat" );
6      System.out.println( "Gently down the stream" );
7      System.out.print( "Merrily, " );
8      System.out.print( "merrily, " );
9      System.out.print( "merrily, " );
10     System.out.println( "merrily" );
11     System.out.println( "Life is but a dream" );
12   }
13 }
```

Table of Contents

List of Reserved Words

Reserved words cannot be used for identifier (name for a method, a class, or a variable)

abstract	default	if	private	this
boolean	do	implements	protected	throw
break	double	import	public	throws
byte	else	instanceof	return	transient
case	extends	int	short	try
catch	final	interface	static	void
char	finally	long	strictfp	volatile
class	float	native	super	while
const	for	new	switch	
continue	goto	package	synchronized	

Syntax Errors

- Errors that are found where the code is not in the format that the Java compiler expects
- The execution of compilation (javac) informs of the error, for example:
  - Spelling errors
  - Use of reserved words
  - Unbalanced braces
  - Incomplete comments
  - Missing semicolons
  - Missing magic word "static"

Buggy Hello World

```
1  public class BuggyHelloWorld
2    public static void main( String[] args ) {
3      System.out.pritnln( "Hello, World! );
4      System.out.println( Hello, Class!" );
5      System.out.println( "Hello, its' me!" ):
6    }
7  }
8  }
```

Buggy Hello World

```
1  public class BuggyHelloWorld
2    public static void main( String[] args ) {
3      System.out.pritnln( "Hello, World! );
4      System.out.printin( Hello, Class!" );
5      System.out.printin( "Hello, its' me!" ):
6    }
7  }
8  }
```

Missing {

Buggy Hello World

```
1  public class BuggyHelloWorld
2    public static void main( String[] args ) {
3      System.out.pritnln( "Hello, World! );
4      System.out.printin( Hello, Class!" );
5      System.out.printin( "Hello, its' me!" ):
6    }
7  }
8  }
```

Missing " at the end of literal

Buggy Hello World

```
1  public class BuggyHelloWorld
2    public static void main( String[] args ) {
3      System.out.pritnln( "Hello, World! );
4      System.out.printin( Hello, Class!" );
5      System.out.printin( "Hello, its' me!" ):
6    }
7  }
8  }
```

Missing " at the beginning of literal

Buggy Hello World

```
1  public class BuggyHelloWorld
2    public static void main( String[] args ) {
3      System.out.pritnln( "Hello, World! );
4      System.out.printin( Hello, Class!" );
5      System.out.printin( "Hello, its' me!" ):
6    }
7  }
8  }
```

Colon instead of semicolon

Buggy Hello World

```
1  public class BuggyHelloWorld
2    public static void main( String[] args ) {
3      System.out.pritnln( "Hello, World! );
4      System.out.printin( Hello, Class!" );
5      System.out.printin( "Hello, its' me!" ):
6    }
7  }
8  }
```

Extra }

```
public class BuggyHelloWorld
                           ^
BuggyHelloWorld.java:3: error: unclosed string literal
    System.out.pritnln( "Hello, World! );
                        ^
BuggyHelloWorld.java:3: error: ';' expected
    System.out.pritnln( "Hello, World! );
                                       ^
BuggyHelloWorld.java:4: error: illegal start of expression
    System.out.printin( Hello, Class!" );
         ^
BuggyHelloWorld.java:4: error: ';' expected
    System.out.printin( Hello, Class!" );
              ^
BuggyHelloWorld.java:4: error: ')' expected
    System.out.printin( Hello, Class!" );
                                ^
BuggyHelloWorld.java:4: error: unclosed string literal
    System.out.printin( Hello, Class!" );
                                  ^
BuggyHelloWorld.java:5: error: ';' expected
    System.out.printin( "Hello, its' me!" ):
         ^
BuggyHelloWorld.java:5: error: ';' expected
    System.out.printin( "Hello, its' me!" ):
                                         ^
BuggyHelloWorld.java:8: error: class, interface, or enum expected
}
^
10 errors
```

How to deal with compilation errors

- It is often the case that one typo causes multiple error messages
- Find the very first error message and fix the error

Two Other Types of Errors

- **Runtime errors** : generated during the execution of the code
  - Not naming the main method "main"
- **Logic Errors** : due to errors in logic for architecting the code; do not necessarily generate runtime errors; the results are different from those that are anticipated

Table of Contents

White Space Can Be Inserted Pretty Much Everywhere

Normal-looking use of white space:

```
1  public class HelloWorld2 {
2    public static void main( String[] args ) {
3      System.out.println( "Hello, World!" );
4      System.out.println( "Hello, Class!" );
5      System.out.println();
6      System.out.println( "This is the fourth line of output." );
7    }
8  }
```

Insertion of White space

Whitespace, tab, and newline characters can be added surrounding identifiers, attributes, and other symbols (parentheses, single-letter math signs, punctuations)

```
1  public  class  HelloWorldWithSpace
2  {
3    public  static  void   main   ( String [ ] args )
4    {
5       System . out . println (  "Hello, World!"     ) ;
6       System . out . println (  "Hello, Class!"   ) ;
7       System . out . println (                    ) ;
8       System . out . println (  "This is the last line." );
9    }
10 }
```

Escape Characters

How can I print a double quotation mark as part of a String literal?

Escape Characters

How can I print a double quotation mark as part of a String literal?

- System.out.println(""""); does not work because the Java compiler thinks that the second one matches the first and so the third one is extraneous

Escape Characters

How can I print a double quotation mark as part of a String literal?

- System.out.println("""); does not work because the Java compiler thinks that the second one matches the first and so the third one is extraneous
- Solution: attach a backslash in front of the second to inform Java that it is not the double quote that is to surround the characters to be printed
  System.out.println("\"");

Escape Characters

How can I print a double quotation mark as part of a String literal?

- System.out.println(""""); does not work because the Java compiler thinks that the second one matches the first and so the third one is extraneous
- Solution: attach a backslash in front of the second to inform Java that it is not the double quote that is to surround the characters to be printed System.out.println("\"");

Then how can I print a backslash?

Escape Characters

How can I print a double quotation mark as part of a String literal?

- System.out.println(""""); does not work because the Java compiler thinks that the second one matches the first and so the third one is extraneous
- Solution: attach a backslash in front of the second to inform Java that it is not the double quote that is to surround the characters to be printed System.out.println("\"");

Then how can I print a backslash?

- System.out.println("\\"); does it.

Escape Characters

Within a String literal, \" and \\ respectively represent the double quotation mark and the backslash character, e.g., "She said, \"How are you?\""

Also, \t and \n respectively represent the tab and the linefeed (in the case of Unix, Linux, and Mac OS X \n works fine)

Printing Some Text with Quotes

```
1  public class HuckleberryFinn {
2    public static void main(String[] args) {
3      System.out.println("\\Quoted from Huckleberry Finn\\");
4      System.out.println("I broke in and says:");
5      System.out.println("\"They're in an awful peck of trouble, and\"");
6      System.out.println("\"Who is?\"");
7      System.out.println("\"Why, pap and mam and sis and Miss Hooker;");
8      System.out.println("\tand if you'd take your ferryboat" );
9      System.out.println("\tand go up there\"");
10     System.out.println("\"Up where?  Where are they?\"");
11     System.out.println("\"On the wreck.\"");
12     System.out.println("\"What wreck?\"");
13     System.out.println("\"Why, there ain't but one.\"");
14     System.out.println("\"What, you don't mean the Walter Scott?\"");
15     System.out.println("\"Yes.\"");
16     System.out.println("\"Good land! what are they doin' there, ");
17     System.out.println("\tfor gracious sakes?\"");
18     System.out.println("\"Well, they didn't go there a-purpose.\"");
19   }
20 }
```

Results

```
% java HuckleberryFinn
\Quoted from Huckleberry Finn\
I broke in and says:
"They're in an awful peck of trouble, and"
"Who is?"
"Why, pap and mam and sis and Miss Hooker;
        and if you'd take your ferryboat
        and go up there"
"Up where?  Where are they?"
"On the wreck."
"What wreck?"
"Why, there ain't but one."
"What, you don't mean the Walter Scott?"
"Yes."
"Good land! what are they doin' there,
        for gracious sakes?"
"Well, they didn't go there a-purpose."
```

Commenting

Java has three types of commenting

1. If // appears in a line the // and whatever remaining in the line are comments
2. A block of lines in which the first line starts with /* and the last last either starts with */ or starts with * and ends with */ and everything in-between starts with *
3. The same as before but the first line starts with /** instead of /*

The last type is used for "javadoc" - record of information about a method; it must precede the declaration of the method it annotates

Comments Example

```
1  /*
2   * Class for showing comment examples
3   * Written by Mitsunori Ogihara
4   */
5  public class Comments {
6    /**
7     * main method
8     * @param args the arguments
9     */
10   public static void main( String[] args ) {
11     // There are two lines in the program
12     System.out.println( "A code needs comments!" );
13     System.out.println( "A code needs indentation!" );
14   }
15 }
```

Multiline comment

Comments Example

```
1  /*
2   * Class for showing comment examples
3   * Written by Mitsunori Ogihara
4   */
5  public class Comments {
6    /**
7     * main method
8     * @param args the arguments
9     */
10   public static void main( String[] args ) {
11     // There are two lines in the program
12     System.out.println( "A code needs comments!" );
13     System.out.println( "A code needs indentation!" );
14   }
15 }
```

The "javadoc" comment

Comments Example

```
1  /*
2   * Class for showing comment examples
3   * Written by Mitsunori Ogihara
4   */
5  public class Comments {
6    /**
7     * main method
8     * @param args the arguments
9     */
10   public static void main( String[] args ) {
11     // There are two lines in the program
12     System.out.println( "A code needs comments!" );
13     System.out.println( "A code needs indentation!" );
14   }
15 }
```

Regular comment

Code Writing Principles

1. Indent further whenever a new level of is generated – Java ignores all the white space characters at the beginning of line

Code Writing Principles

1. Indent further whenever a new level of is generated – Java ignores all the white space characters at the beginning of line
2. Provide comments so as to remember what the code is supposed to be doing

Code Writing Principles

1. Indent further whenever a new level of is generated – Java ignores all the white space characters at the beginning of line
2. Provide comments so as to remember what the code is supposed to be doing
3. Insert space between lines for readability, if necessary

Code Writing Principles

1. Indent further whenever a new level of is generated – Java ignores all the white space characters at the beginning of line
2. Provide comments so as to remember what the code is supposed to be doing
3. Insert space between lines for readability, if necessary
4. Try to assign meaningful variable names and method names, all have to start with a lowercase letter, possibly using numerics and underscores

Code Writing Principles

1. Indent further whenever a new level of is generated – Java ignores all the white space characters at the beginning of line
2. Provide comments so as to remember what the code is supposed to be doing
3. Insert space between lines for readability, if necessary
4. Try to assign meaningful variable names and method names, all have to start with a lowercase letter, possibly using numerics and underscores
5. No more than one statement per line

Code Without Indentation

```
1  /*
2  * Class for showing comment examples
3  * Written by Mitsunori Ogihara
4  */
5  public class CommentsWithNoIndent {
6  /**
7  * main method
8  * @param args the arguments
9  */
10 public static void main( String[] args ) {
11 // There are two lines in the program
12 System.out.println( "A code needs comments!" );
13 System.out.println( "A code needs indentation!" );
14 }
15 }
```