CMPS 251

Please read Chapters 2, 4 and 5

Fundamentals

Dr. Abdelkarim Erradi

CSE @ QU

Outline

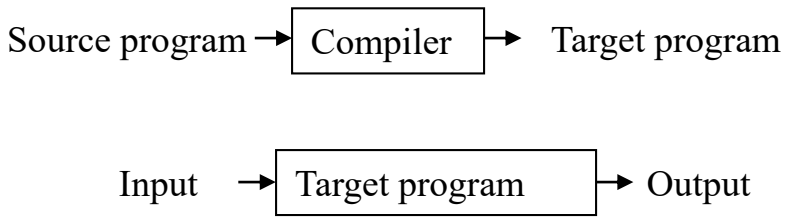
- Introduction to Java
- Data types (numeric and string)
- Expressions
- Conditional Statements (if-else and switch)
- Loops (for, while, do)
- Arrays
- Input/output

Introduction to Java

Back

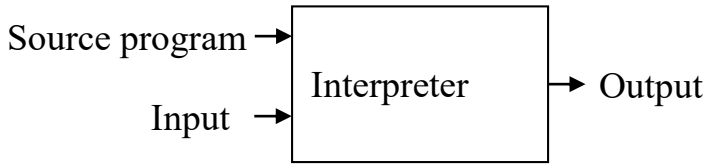
Families of Languages

Compiled translates to machine code



- **Fast**
- Example: C++

Interpreted executes source code "directly"



- **Flexible**
- But Slower execution

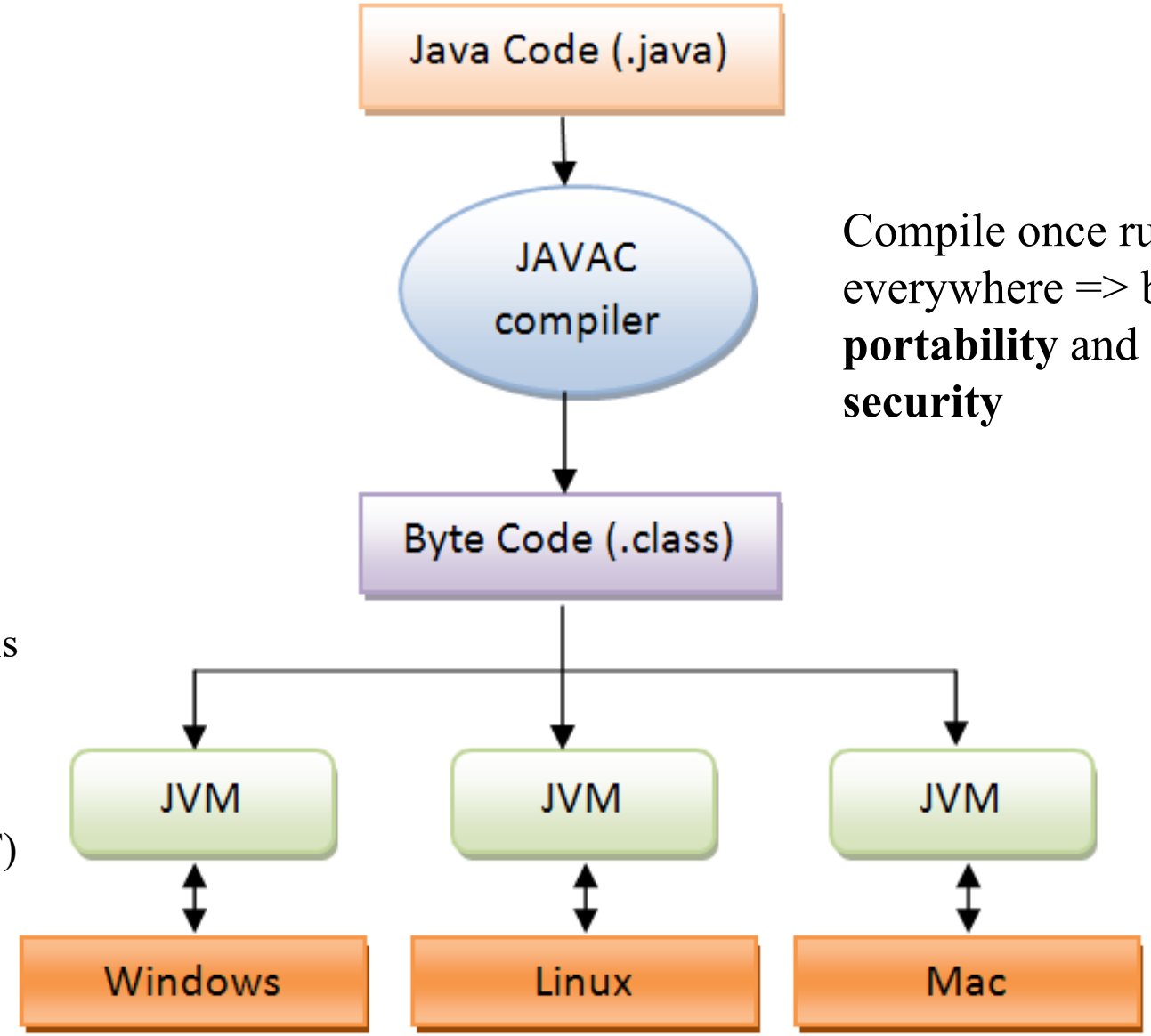
Example: JavaScript

Hybrid interpretation of intermediate code

- **Portable**
- Example: Java

Java Compilation

- **Java** source code is **compiled** into bytecode (saved in .class file)



Compile once run everywhere => better **portability** and **security**

- When the program is to be run, the bytecode is converted, using the just-in-time (JIT) compiler, into executable machine code

First Cup of Java

class header

method header

Class

Method

```
public class FirstCup {

   public static void main( String[] args )
   {
        System.out.println( "Hello World" );
   } // end method

} // end class
```

Comments

- Comments should be included to explain the purpose of the program and describe the processing
- They do not affect how a program works
- Java comments can take two forms:

```
// this comment runs to the end of the line


/*  this comment runs to the terminating
    symbol, even across line breaks        */
```

Data Types

Back

Variables

- A *variable* is a name for a location in memory
- A variable must be *declared* by specifying its name and the type of data that it will hold

data type

variable name

Multiple variables can be created in one declaration

```
int total;
int count, temp, result;
```

Always choose meaningful and descriptive variable names

Numeric Types

- The difference between the various numeric primitive types is their **size**, and therefore the **values they can store**:

Type	Storage	Min Value	Max Value
`byte`	8 bits	–128	127
`short`	16 bits	–32,768	32,767
`int`	32 bits	–2,147,483,648	2,147,483,647
`long`	64 bits	–9,223,372,036,854,775,808	9,223,372,036,854,775,807
`float`	32 bits	Approximately –3.4E+38 with 7 significant digits	Approximately 3.4E+38 with 7 significant digits
`double`	64 bits	Approximately –1.7E+308 with 15 significant digits	Approximately 1.7E+308 with 15 significant digits

Character

- A `char` variable stores a single character
- Character literals are delimited by single quotes:

```
'a'   'X'   '7'   '$'   ','   '\n'
```

- Example declarations

```
char topGrade = 'A';

char terminator = ';', separator = ' ';
```

- Note the distinction between a primitive character variable, which holds only one character, and a `String` object, which can hold multiple characters

Boolean

- A `boolean` value represents a true or false values
- The reserved words `true` and `false` are the only valid values for a boolean type

```
boolean done = false;
```

String

- **String is a s**equence of characters

Example:

```
String greeting = "Hello world!";
```

- Many useful builtin methods
  - contains, startsWith, endsWith, indexOf, substring, split, replace
  - toUpperCase, toLowerCase, equals

- Use `equals` (not ==) to compare strings

```
String name = "Ali";
name.equals("Ali");  // --> true
```

String Concatenation

Use + for string concatenation

```
String helloStr = "Hello," + " world!";
System.out.println( helloStr );
//The output will be     Hello world!
```

Implicit type using var

NEW

- Java 10 and above has **var** keyword to declare a variable without explicit type

e.g. instead of doing

```
String str = "Java" ;
```

You can just say

```
var str = "Java" ;
```

- Java will **implicitly** recognize the variable data type based on the initial value assigned to it
- This will be heavily used in this course!

Expressions

An *expression* is a combination of one or more operators and operands

Back

Arithmetic Operators

Java operation	Operator	Algebraic expression	Java expression
Addition	+	$f + 7$	`f + 7`
Subtraction	–	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

Relational Operators

Standard algebraic equality or relational operator	Java equality or relational operator	Sample Java condition	Meaning of Java condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Special Math Operators

Use **a <operator>= b** instead of a = a <operator> b

Assignment operator	Sample expression	Explanation	Assigns
Assume: int c = 3, d = 5, e = 4, f = 6, g = 12;			
+=	c += 7	c = c + 7	10 to c
-=	d -= 4	d = d - 4	1 to d
*=	e *= 5	e = e * 5	20 to e
/=	f /= 3	f = f / 3	2 to f
%=	g %= 9	g = g % 9	3 to g

- Another handy shortcut:
  ++ increments by 1
  -- decrements by 1

Example:

b++ is the same as **b = b + 1**

Conditional Statements

If Statement

- Single option

```
if (boolean-expression) {
  statement1;
  ...
  statementN;
}
```

- Two options

```
if (boolean-expression) {
  ...
} else {
  ...
}
```

- Multiple options

```
if (boolean-expression) {
  ...
} else if (boolean-expression) {
  ...
} else if (boolean-expression) {
  ...
} else {
  ...
}
```

- A widely accepted best practice is to use the braces even if there is only a single statement inside the if or else.

Using if…else

```
import java.util.Scanner;

public class OddEven {
 public static void main(String arg[])
 {
   Scanner input = new Scanner( System.in );
   int x;
   System.out.print( "Enter an integer: " );
    x = input.nextInt();

    if (x%2==0)
      System.out.printf("%d %s",x,"is even number");
    else
      System.out.printf("%d %s",x,"is odd number");
 }
}
```

Switch Statement

- Example

```
int month = ...;
String monthString;
switch(month) {
  case 1: monthString = "January";  break;
  case 2: monthString = "February"; break;
  case 3: monthString = "March";    break;
  ...
  default: monthString = "Invalid month";
 break;
}
```

Loops

Loops

A loop is a programming construct that **repeats** an action

- **for**

```
for(init; continueTest; updateOp) {
  body;
}
```

- **for/each**

```
for(variable: collection) {
  body;
}
```

- **while**

```
while (continueTest) {
  body;
}
```

- **do**

```
do {
  body;
} while (continueTest);
```

For Loop

```
public static void listNums1(int max) {
  for(int i=0; i<max; i++) {
    System.out.println("Number: " + i);
  }
}
```

- Result

```
listNums1(4);

Number: 0
Number: 1
Number: 2
Number: 3
```

For/Each Loop

```
public static void listEntries(String[] entries) {
  for(var entry: entries) {
    System.out.println(entry);
  }
}
```

- Result

```
String[] test = {"This", "is", "a", "test"};
listEntries(test);

This
is
a
test
```

While Loop

```
public static void listNums2(int max) {
  int i = 0;
  while (i < max) {
    System.out.println("Number: " + i);
    i++; // "++" means "add one"
  }
}
```

- Result

```
listNums2(5);

Number: 0
Number: 1
Number: 2
Number: 3
Number: 4
```

Do Loop

```
public static void listNums3(int max) {
  int i = 0;
  do {
    System.out.println("Number: " + i);
    i++;
  } while (i < max);
                  // ^ Don’t forget semicolon
}
```

- Result

```
listNums3(3);
Number: 0
Number: 1
Number: 2
```

Deciding which Loop to Use

- **`while`**: pretest loop (loop body may not be executed at all)
- **`do-while`**: post test loop (loop body will always be executed at least once)
- **`for`**: pretest loop (loop body may not be executed at all); has initialization and update code; is useful with counters when precise number of repetitions is known
- **for/each**: used to iterate through a list / array

Arrays

Arrays

- Array = a sequence of values of the same type

- Declare and initialize an array

```
type[] var = { val1, val2, ... , valN };
```

- Examples:

```
int[] values = { 10, 100, 1000 };
String[] names = {"Joe", "Jane", "Juan"};
```

Declare then populate arrays

- **Step 1: Declare the array:**

```
const int size = 10;
type[] var = new type[size];
```

E.g.:

```
int[] primes = new int[size];
String[] names = new String[size];
```

- **Step 2: populate the array**

```
primes[0] = 2;
primes[1] = 3;
primes[2] = 5;
primes[9] = 7;
```

```
names[0] = "Ali";
names[1] = "Ahmed";
names[2] = "Mariam";
names[3] = "Sarah";
```

etc.

Input / Output

Using System.out for Screen Output

- Commonly used methods:
  - **print(String *line*)** will send *line* to output
  - **println(String line)** will send line to output, followed by a line break
    println() will send just the line break to output
  - **printf** (formatString, argumentList)

```
double price = 19.8;
String name = "magic apple";
System.out.printf("$%.2f for each %s.", price, name);
```

will output

```
$19.80 for each magic apple
```

  - The formatString contains 2 format specifiers (`%.2f` and `%s`) that match the two arguments (`price` and `name`)
  - The format specifier `"%.2f"` indicates displaying 2 digits after the decimal point

Display formatted Data

```
public class FormattedData {
 public static void main(String arg[])
 {
  System.out.printf("%s\n%s","Hello world","Welcome to java");
 }

}
```

Format

The format **%s** is a placeholder for a string

\n inserts a line break

Problems @ Javadoc Declaration Console

```
<terminated> FormattedData [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.ex
Hello world
Welcome to java
```

Reading Input from the Keyboard

- Read input from the keyboard using the **Scanner** class

```
Scanner input = new Scanner(System.in);
int i = input.nextInt();
double d = input.nextDouble();
```

- In real applications, use a Graphical User Interface (GUI)

JOptionPane for Input/Output

- You may use **JOptionPane** for windows-based Input/Output
  - **Input**

```
inputString =
JOptionPane.showInputDialogue
(stringExpression);
```

  - **Output**

```
JOptionPane.showMessageDialog
(null, stringExpression);
```

Summary

- Primitive types, variables declaration, assignment, expressions, conditional statements, loops, and array are similar to C++
  - Java has a special **for** each loop to iterate trough a list

```
for(var s: stringArray) { … }
```

- Use **`equals`** (not ==) to compare strings
- Use **System.out.println** for output to the screen
- Use the **Scanner** class to read values from the keyboard
- More info @ https://www.w3schools.com/java/