Unit 01

Introduction to Java

CMPS 251, Fall 2020, Dr. Abdulaziz Al-Ali

Announcements

- Syllabus week distribution changed, redownload

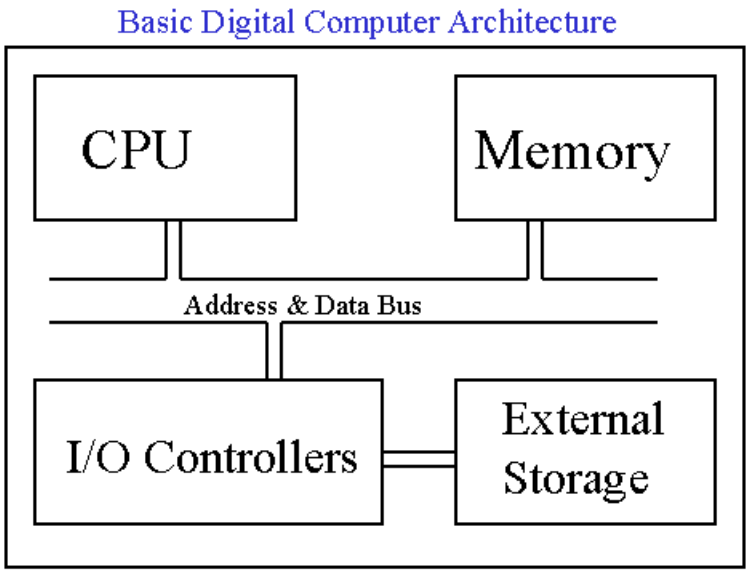
- Instructions for accessing eBooks now under *Course Content → How-To, Guides, and Manuals*

Objectives

- Introduction to Java
- Data types (numeric and String)
- Expressions
- Conditional statements
- Loops
- Arrays
- Input/Output

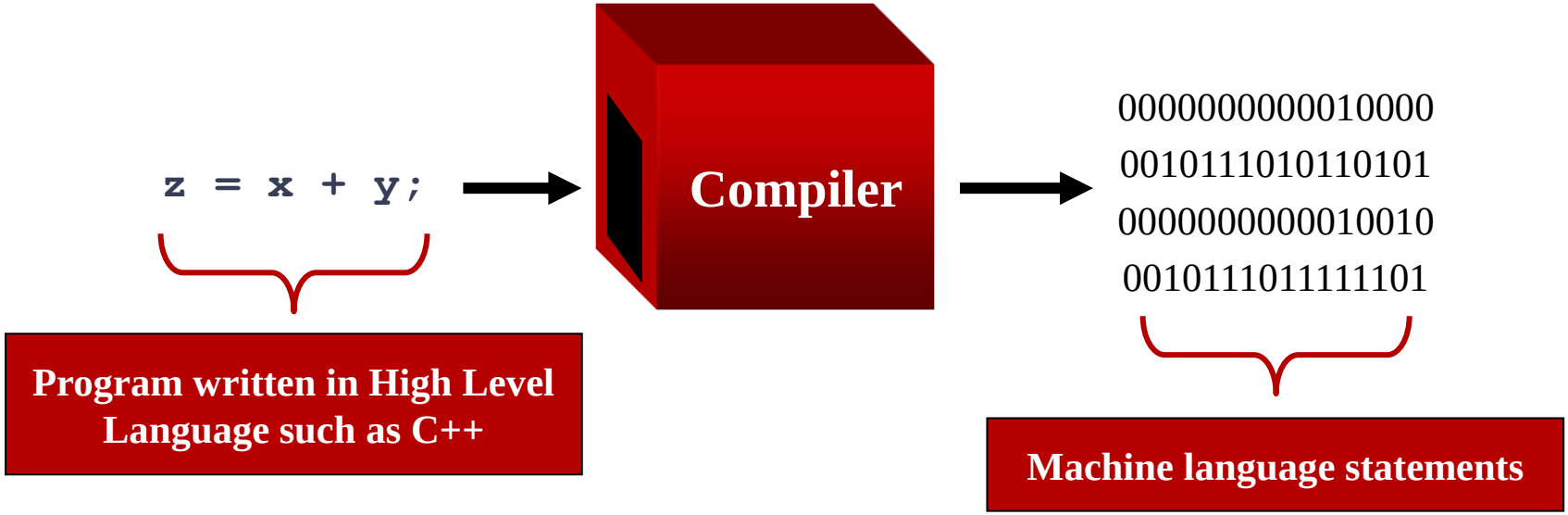
Getting to know Java

- Computer Architecture
- Processor Instructions
  - From code to Machine language
- Interpretation
- JVM
  - Java Virtual Machine



Compiler (from C++)

- Compiler translates high level statements into Machine Language (ML) statements

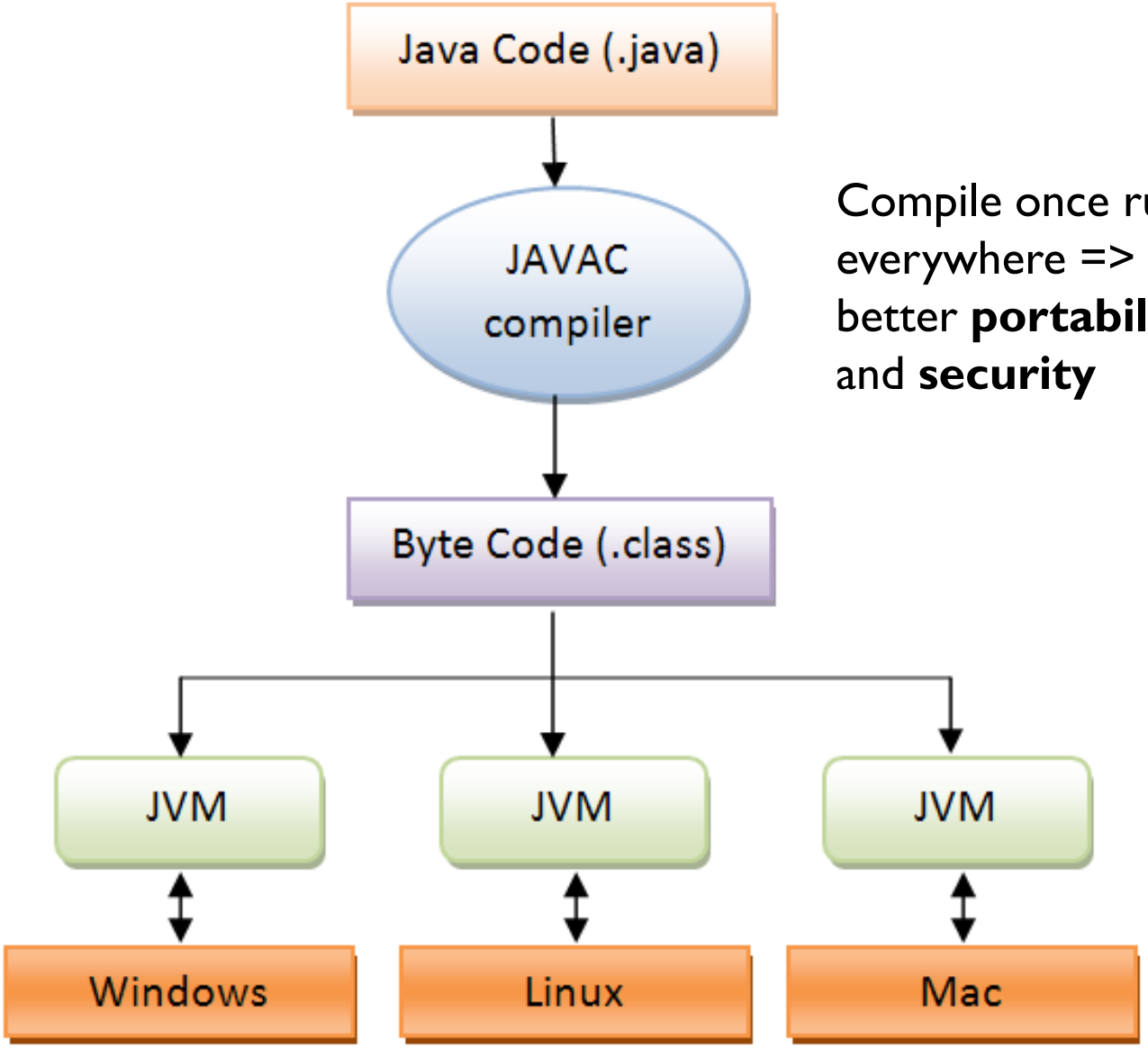


Java

- Write it once, runs everywhere!
- How is that achieved?

Java Compilation

- **Java** source code is **compiled** into bytecode (saved in .class file)
- When the program is to be run, the bytecode is converted, using the just-in-time (JIT) compiler, into executable machine code
- JVM means Java Virtual Machine



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

Java JVM disadvantages

- Speed!
- Memory consumption (for small devices)
- JVM is a requirement (for the end-user)

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
```

Every program must have at least one class.

Packages and Import statements

- Packages
  - A method to categorize classes and interfaces
  - Package name comes first in your java file

- Import statements
  - Bring in external source code into your file
  - Usually has definitions of external classes/methods/variables
  - They are usually defined after the package statement (if one exists) and before class declarations

Demo of running a java program

- Public classes
- Naming of java files/classes
- Package statement /location
- Import statement /location

Last Lecture

- What is a JVM? Why is it needed in Java?
- Are classes required in Java?
- What is the name of the tool we'll use for writing Java code?
- Which function did Dr. Abdulaziz forget to write last lecture? Why is it needed?

Warm up

- When you create a new class in Java, what does it do in the project folder?

- What about a package?

Java Syntax Overview

- Let's look at an overview of Java syntax
- You'll notice a lot of similarities to C++
- This is abbreviated because you already know C++

Commenting Code

`//`	Single line comment. Anything after the // on the line will be ignored by the compiler.
`/* … */`	Block comment. Everything beginning with /* and ending with the first */ will be ignored by the compiler. This comment type cannot be nested.
`/** … */`	Javadoc comment. This is a special version of the previous block comment that allows comments to be documented by the javadoc utility program. Everything beginning with the /** and ending with the first */ will be ignored by the compiler. This comment type cannot be nested.

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

Identifiers

- Identifiers are programmer-defined names for:
  - variables
  - methods
  - classes
- Identifiers may not be any of the Java reserved keywords.

Java Reserved Keywords

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

***Refer to Appendix C in the textbook**

Identifiers

- Identifiers must follow certain rules:
  - An identifier may only contain:
    - letters a–z or A–Z,
    - the digits 0–9,
    - underscores (_), or
    - the dollar sign ($)
  - The first character may not be a digit.
  - Identifiers are case sensitive.
    - `itemsOrdered` is not the same as `itemsordered`.
  - Identifiers cannot include spaces.

Test your knowledge

- Which of the following identifiers will not compile and why?
  - _frequentFlyer
  - 33dollars
  - Dollar45
  - moneyIn$
  - heavy Monkey
  - public

Java Naming Conventions

- Variable names should begin with a lower case letter and then switch to title case thereafter:

  Ex: `int reducedItemPrice`

- Class names should be all title case.

  Ex: `public class BabyElephant`

- Read the naming conventions at:

  http://www.oracle.com/technetwork/java/codeconventions-135099.html

  A general rule of thumb about naming variables and classes are that, with some exceptions, their names tend to be nouns or noun phrases.

Test your knowledge

- What are the following identifiers?
  - StudentCounter
  - countStudents
  - STUDENTS_COUNT
  - student.count

See!

class header

method header

Class

Method

```
public class FirstCup {

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

} // end class
```

Every program must have at least one class.

Arithmetic Operations

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`

Q: Can you add two strings in Java ?

e.g.

```
String s1 = "Hi";
String s2 = "There";
String s3 = s1+s2; //what is stored in s3?
```

Relational Operations

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

Special Math Operators

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

Special Math Operators

Operator	Operator name	Sample expression	Explanation
`++`	prefix increment	`++a`	Increment `a` by `1`, then use the new value of `a` in the expression in which `a` resides.
`++`	postfix increment	`a++`	Use the current value of `a` in the expression in which `a` resides, then increment `a` by `1`.
`--`	prefix decrement	`--b`	Decrement `b` by `1`, then use the new value of `b` in the expression in which `b` resides.
`--`	postfix decrement	`b--`	Use the current value of `b` in the expression in which `b` resides, then decrement `b` by `1`.

Hint: read from left to right to know whether you should use, or increment first.

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

Promotions in Java

Type	Valid promotions
`double`	None
`float`	`double`
`long`	`float` or `double`
`int`	`long`, `float` or `double`
`char`	`int`, `long`, `float` or `double`
`short`	`int`, `long`, `float` or `double` (but not `char`)
`byte`	`short`, `int`, `long`, `float` or `double` (but not `char`)
`boolean`	None (`boolean` values are not considered to be numbers in Java)

Fig. 6.4 | Promotions allowed for primitive types.

Demotion of variables

- Java does not allow <u>demotion</u> of variables automatically.

- What happens if we write the following:

  - int x = 3.55;
  - System.out.println(x);

- How can we demote variables?
  - Use casting: int x = (int) 3.55;

Check Point

- Which one of the following is an invalid identifier?
  - grades34
  - $twoGrades
  - _Grades
  - 6elephants

Example

- Promotions / Demotions

```
private static void displayNum(int a)
{
        System.out.println(a);
}
```

- Consider the above function, are these statements valid?

```
double y = 33.5f;
displayNum(y);
```

displayNum(y) is illegal (not correct), because java does not automatically demote variables, you have to use casting:
displayNum((int) y);

String

- **String** is a sequence of characters

Example:

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

- Useful tip:
  - In Java use **equals** (not ==) to compare strings

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

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.
- We will be using this style in certain places.

Control Statements

- `if`
- `if… else if`
- `if… else if… else`
- `for()`
- `while()`
- `do… while()`
- `switch()`

Switch statements in Java

- Can you see what is different about the switch statement in the following code that you have not seen before in C++?

```
String abc;
……..(code to take user input in abc)……..
switch (abc)
{
        case "Doha":
                System.out.println("Qatar");
                break;
        case "Cairo":
                System.out.println("Egypt");
                break;
}
```

Loops: 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

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(String formatString, argumentList…)**

Formatted print

```
Specifying format:   %<options><data type>
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 Strings

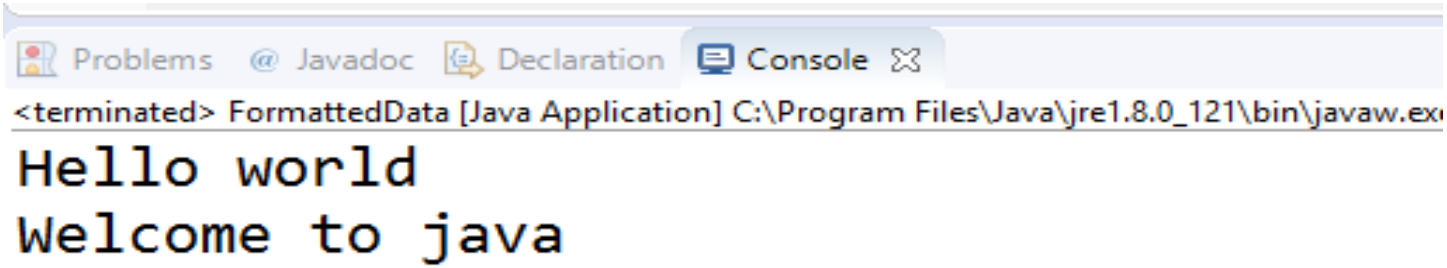
```
public class FormattedData {
 public static void main(String arg[])
 {
  System.out.printf("%s\n%s","Hello world","Welcome to java");
 }

}
```

Format

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

\n inserts a line break



Formatted Strings

- Excellent printf tutorial on: http://alvinalexander.com/programming/printf-format-cheat-sheet

- See Unit 1 sample code and practice on TODO 1

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)

Practice at home

- Write a program to print the following

```
*
**
***
****
*****
```

- Write a program that will ask the user to enter a PIN code. The program should display "Access granted!" if the PIN code is 1234. Otherwise, it will ask the user to enter the PIN code again.

Java API

- Java has lots of classes available for your use in the Java API: https://docs.oracle.com/en/java/javase/13/docs/api/index.html

Overview Package Class Use Tree Deprecated Index Help

Java™ Platform Standard Ed. 7

Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.util

Class Scanner

java.lang.Object
  java.util.Scanner

All Implemented Interfaces:

Closeable, AutoCloseable, Iterator<String>

```
public final class Scanner
extends Object
implements Iterator<String>, Closeable
```

A simple text scanner which can parse primitive types and strings using regular expressions.

A `Scanner` breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various `next` methods.

For example, this code allows a user to read a number from `System.in`:

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
```

Next is Object Oriented Concepts!