Java Fundamentals

Han-fen Hu

Tony Gaddis (2019) Starting Out with Java: From Control Structures through Data Structures, 4th Edition

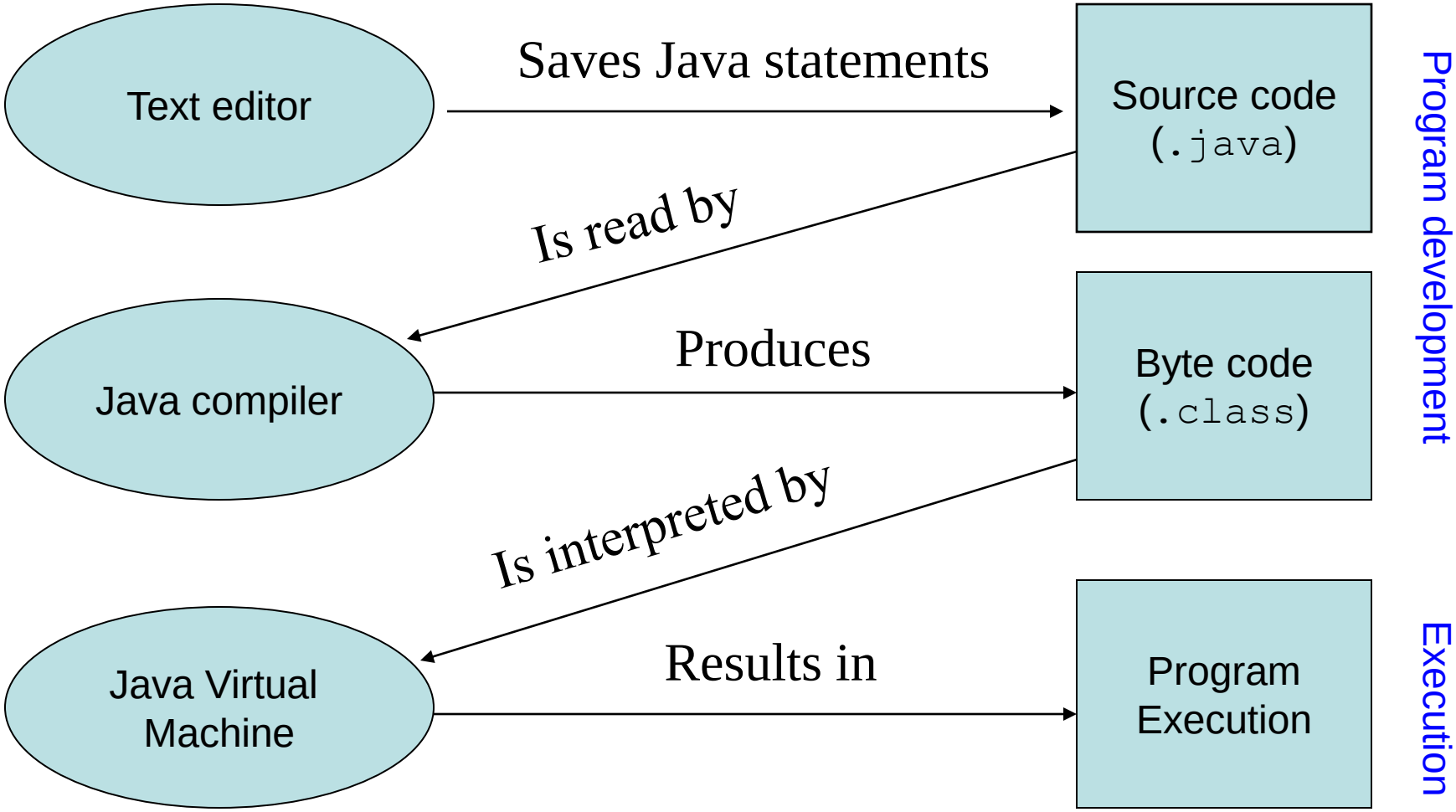
UNLV

Outline

- Java Program Development Process
- Parts of a Java Program
- Variables and Literals
  - Primitive Data Types
  - Arithmetic Operators
  - Combined Assignment Operators
- String
- Programming style

UNLV

Java Program Development Process



Compiler and the Java Virtual Machine (1)

- Source code
  - A programmer writes Java programming statements (i.e., source code) for a program
  - A text editor is used to edit and save a Java source code file.
  - Source code files have a *.java* file extension

Compiler and the Java Virtual Machine (2)

- Byte code
  - A compiler is a program that translates source code into an executable form
  - Syntax errors that may be in the program will be discovered during compilation
    - Syntax errors are mistakes that the programmer has made that violate the rules of the programming language
  - The compiler translates source code into executable files containing machine code; i.e., the byte code instructions
    - Byte code instructions are the machine language of the Java Virtual Machine (JVM) and cannot be directly executed directly by the CPU
    - Byte code files end with the *.class* file extension

Compiler and the Java Virtual Machine (3)

- Program execution
  - The JVM is a program that *emulates* a micro-processor.
  - The JVM executes instructions as they are read.
  - JVM is often called an *interpreter.*
  - Java is often referred to as an interpreted language.

JVM and Portability (1)

- *Portable* means that a program may be written on one type of computer and then run on a wide variety of computers, with little or no modification
- With most programming languages, portability is achieved by compiling a program for each CPU it will run on

JVM and Portability (2)

- Java
  - Byte code runs on the JVM and not on any particular CPU; therefore, compiled Java programs are highly portable.
  - Java provides an JVM for each platform so that programmers do not have to recompile for different platforms
- JVMs exist on many platforms
  - Windows, Mac, Linux, Unix, BSD, etc.
  - Java Runtime Environment (JRE) creates the JVM.

UNLV

Java Development Kit

- Tool you use to write Java programs is called the Java Development Kit, or JDK.
- There are different editions of the JDK:
  - Java SE - Java Platform, Standard Edition
  - Java EE - Java Platform, Enterprise Edition
  - Java ME - Java Platform, Micro Edition
- Available for download at https://www.oracle.com/java/technologies/

Parts of a Java Program (1)

- Java is a case-sensitive language.
- A Java source code file contains one or more Java classes.
- The class and the filename of the source code file must match.
  - A class named Simple must be in a file named Simple.java

Example: Employee.java

A model class example

```
public class Employee {
    private String fullName;
    private double payRate;

    /* The constructor sets the default value for the fields.
     * fullName is default to an empty string
     * payRate is default to 13.5
     */
    public Employee() {
        fullName = "";
        payRate = 13.5;
    }


    public String getFullName() {
        return fullName;
    }

    public void setFullName(String fullName) {
        this.fullName = fullName;
    }

    public double getPayrate() {
        return payRate;
    }

    public void setPayrate(double payrate) {
        this.payRate = payrate;
    }

    public double calDailyPay(double hours) {
        return payRate* hours;
    }

}
```

Fields

Program comments

Constructor

Methods

UNLV

Example: Simple.java

An application class example

Main routine

Program comments

```
public class Simple {
    public static void main(String[] args) {
        // This is a simple Java program

        // declare variables
        double salary = 0;

        System.out.println("Programming is great fun");

        Employee someone = new Employee();

        someone.setFullName("Alex Smith");
        salary = someone.calDailyPay(8);
        System.out.print(someone.getFullName()+"'s daily salary is $"+salary);
    }
}
```


UNLV

Java Statements

```
System.out.println("Programming is great
  fun!");
```

- This line uses the **`System`** class from the standard Java library.
  - The **`System`** class contains methods and objects that perform system level tasks.
  - The **`out`** object, a member of the **`System`** class, contains the methods **`print`** and **`println`**.
  - The **`print`** and **`println`** methods actually perform the task of sending characters to the output device.

Parts of a Java Program (2)

- Comments
  - The line is ignored by the compiler.
  - The comment in the example is a single-line comment.
- Class Header
  - The class header tells the compiler things about the class such as what other classes can use it (public) and that it is a Java class (class), and the name of that class.
- Curly Braces
  - When associated with the class header, they define the scope of the class.
  - When associated with a method, they define the scope of the method.

Parts of a Java Program (3)

- The **`main`** Method
  - This line must be exactly as shown in the example (except the *args* variable name can be programmer defined).
  - This is the line of code that the *java* command will run first.
  - This method starts the Java program.
  - Every Java *application* must have a **`main`** method.

Java Package (1)

- A Java package is a mechanism for organizing Java classes into
- Java packages can be stored in compressed files called JAR files, allowing classes to be downloaded faster as groups rather than individually.
- Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality.
- Classes in the same package can access each other's package-access members.

Java Package (2)

Naming Conventions

- Package names are written in all lower case
- Companies use their reversed Internet domain name to begin their package names
  - for example, com.example.mypackage
- Name collisions that occur within a single company need to be handled by convention within that company, perhaps by including the region or the project name after the company name
  - for example, com.example.region.mypackage
- Packages in the Java language itself begin with java. or javax.

Java Package (3)

- Importing a Package Member
  - import graphics.Rectangle;
- Importing an Entire Package
  - import graphics.*;

Managing Source and Class Files

- Many implementations of the Java platform rely on hierarchical file systems to manage source and class files
- For example
  - Put the source code for a class in a text file whose extension is .java
  - Then, put the source file in a directory whose name reflects the name of the package
  - The qualified name of the package member and the path name to the file are parallel

class name: graphics.Rectangle

pathname to file: graphics\Rectangle.java

Lab (1)

- Purposes
  - Create a Java Project a Package
  - Create a Java Class
  - Run Your Project in Eclipse

Special Characters

Symbol	Description	Usage
//	double slash	Marks the beginning of a single line comment.
( )	open and close parenthesis	Used in a method header to mark the parameter list.
{ }	open and close curly braces	Encloses a group of statements, such as the contents of a class or a method.
“ ”	quotation marks	Encloses a string of characters, such as a message that is to be printed on the screen
;	semi-colon	Marks the end of a complete programming statement

Identifiers (1)

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

Identifiers (2)

- 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.
  - Identifiers cannot include spaces.

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

Why Follow the Naming Standards?

- Reduce the effort needed to read and understand source code
- Enable other programmers to focus on more important issues than arguing over syntax and naming standards
- Enable code quality review tools to focus their reporting mainly on significant issues other than syntax and style preferences

Source: https://www.tutorialspoint.com/why-we-should-follow-the-naming-conventions-in-java

Java Naming Conventions (1)

- Classes
  - Nouns, in mixed case with the first letter of each internal word capitalized.
  - simple and descriptive.
  - Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used)
- Methods
  - Verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.

Source: https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html

Java Naming Conventions (2)

- Variables
  - mixed case with a lowercase first letter
  - Variable names should not start with underscore _ or dollar sign $ characters, even though both are allowed
  - Short yet meaningful
  - One-character variable names should be avoided
- Constants
  - All uppercase with words separated by underscores ("_")

Variable Names

- Variable names should be descriptive.
- Descriptive names allow the code to be more readable; therefore, the code is more maintainable.

```
double tr = 0.0725;
double salesTaxRate = 0.0725;
```

- Java programs should be *self-documenting*.

Variable Declarations

- Variable Declarations take the following form:
  - *DataType VariableName;*
    - `byte inches;`
    - `short month;`
    - `int speed;`
    - `long timeStamp;`
    - `float salesCommission;`
    - `double distance;`

Primitive Data Types (1)

- Primitive data types are built into the Java language and are not derived from classes.
- There are 8 Java primitive data types.
  - **`float`**
  - **`double`**
  - **`boolean`**
  - **`char`**
  - **`byte`**
  - **`short`**
  - **`int`**
  - **`long`**

Primitive Data Types (2)

Data Type	Length	Range
byte	1 byte	Integers in the range -128 to +127
short	2 bytes	Integers in the range of -32,768 to +32,767
int	4 bytes	Integers in the range of -2,147,483,648 to +2,147,483,647
long	8 bytes	Integers in the range of -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
float	4 bytes	Floating-point numbers in the range of ±3.410-38 to ±3.41038, with 7 digits of accuracy
double	8 bytes	Floating-point numbers in the range of ±1.710-308 to ±1.710308, with 15 digits of accuracy

Variable Assignment and Initialization (1)

- In order to store a value in a variable, an *assignment statement* must be used.
  - The *assignment operator* is the equal (=) sign.
  - The operand on the left side of the assignment operator must be a variable name.
  - The *operand* on the right side must be either a literal or expression that evaluates to a type
  - The operand on the right side should be compatible with the type of the variable.

Variable Assignment and Initialization (2)

```
3  public class ExampleClass {
4
5      public static void main(String[] args) {
6          // variable declaration
7          int month, days;
8
9          month = 2;
10         days = 28;
11         System.out.println("Month "+month+" has "+days+" days.");
12
13     }
14
15 }
```


UNLV

Variable Assignment and Initialization (3)

- The variables must be declared before they can be used.
- Once declared, they can then receive a value (initialization); however the value must be compatible with the variable's declared type.
- After receiving a value, the variables can then be used in output statements or in other calculations.
- Local variables can be declared and initialized on the same line.

Variable Assignment and Initialization (4)

```
3  public class ExampleClass {
4
5      public static void main(String[] args) {
6          // variable declaration
7          int month=2, days=28;
8
9          System.out.println("Month "+month+" has "+days+" days.");
10
11     }
12
13 }
```

Arithmetic Operators (1)

- Java has 5 arithmetic operators.

Operator	Meaning	Type	Example
+	Addition	Binary	total = cost + tax;
-	Subtraction	Binary	cost = total – tax;
*	Multiplication	Binary	tax = cost * rate;
/	Division	Binary	salePrice = original / 2;
%	Modulus	Binary	remainder = value % 5;

Arithmetic Operators (2)

- Division can be tricky.

  In a Java program, what is the value of 1/2?

- The answer is 0.

- Integer division will truncate any decimal remainder.

Combined Assignment Operators

- Java has some combined assignment operators.

Operator	Example	Equivalent	Value of variable after operation
+=	x += 5;	x = x + 5;	The old value of x plus 5.
-=	y -= 2;	y = y – 2;	The old value of y minus 2
*=	z *= 10;	z = z * 10;	The old value of z times 10
/=	a /= b;	a = a / b;	The old value of a divided by b.
%=	c %= 3;	c = c % 3;	The remainder of the division of the old value of c divided by 3.

Constants

- Constants allow the programmer to use a name rather than a value throughout the program.
  - Constants also give a singular point for changing those values when needed.
  - Constants keep the program organized and easier to maintain.
- Constants are declared using the keyword **`final`**.
- By convention, constants are all upper case and words are separated by the underscore character.

```
final double CAL_SALES_TAX = 0.725;
```

Lab (2)

- Purposes: Complete
  - Wage Application
  - Employee class and EmployeeDemo

`String` Class

- Java has no primitive data type that holds a series of characters.
- The **`String`** class from the Java standard library is used for this purpose.
- In order to be useful, a variable must be created to reference a **`String`** object.

```
String number;
```

  - Notice the **`S`** in **`String`** is upper case.
  - By convention, class names should always begin with an upper-case character.

`String` Objects

- A variable can be assigned a String literal.

```
String value = "Hello";
```

- **`Strings`** are the only objects that can be created in this way.
- A variable can be created using the *new* keyword.

```
String value = new String("Hello");
```

- This is the method that all other objects must use when they are created.

Programming Style

- Although Java has a strict syntax, whitespace characters are ignored by the compiler.
- The Java whitespace characters are space, tab, newline, carriage return
- Indentation
  - Programs should use proper indentation.
  - Each block of code should be indented a few spaces from its surrounding block.