



西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

Object-Oriented Programming

Chapter 3 Fundamental Programming Structures in Java

Dr. Helei Cui

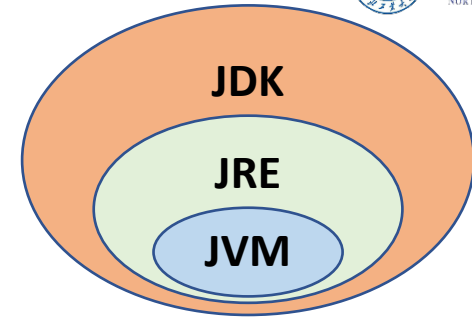
7 Mar 2022

*Slides partially adapted from lecture
notes by Cay Horstmann*

Recap

➤ JDK, JRE, JVM?

- JDK is a software development kit
- JRE is a software bundle that allows Java program to run
- JVM is an environment for executing bytecode

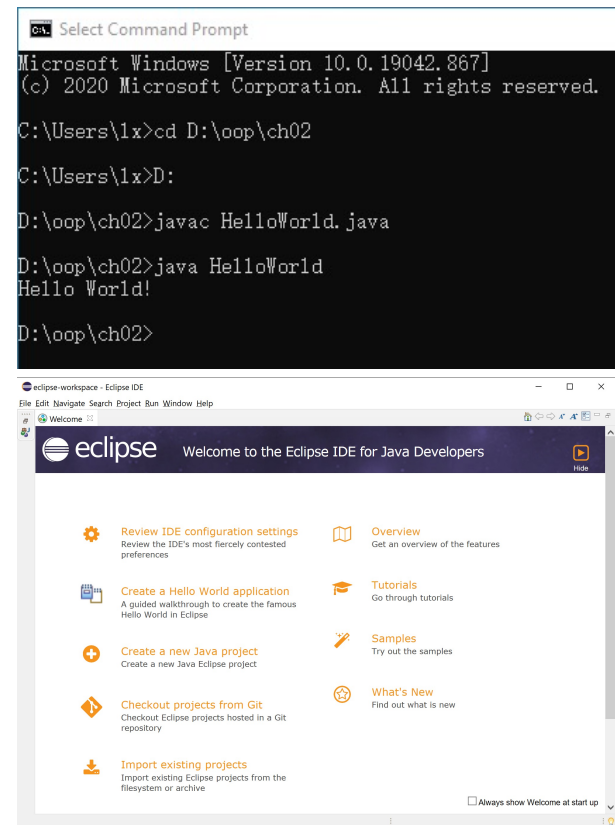


➤ Run **HelloWorld.java**

- Using Command-Line Tools
- Using an IDE, e.g., Eclipse

➤ Use UML editor

➤ Violet or diagrams.net



Contents

- 3.1 A Simple Java Program
- 3.2 Comments
- 3.3 Data Types
- 3.4 Variables and Constants
- 3.5 Operators
- 3.6 Strings
- 3.7 Input and Output
- 3.8 Control Flow
- 3.9 Big Numbers
- 3.10 Arrays

FirstSample.java

```
public class FirstSample {  
    public static void main(String[] args) {  
        System.out.println("We will not use 'Hello, World!'");  
    }  
}
```

- **“public”**: access modifier
 - Define the accessibility of a class.
- **“class”**: “a container for the program logic”
 - Everything in a Java program must be inside a class.
- **“FirstSample”**: name of the class
 - The file name for the source code must be **the same** as the name of the public class, with **“.java”** appended.

Rules for class names in Java

1. Names must **begin with a letter**, and after that, they can have **any combination of letters and digits**.
2. The length is essentially **unlimited**.
3. Do not use a **Java reserved word** for a class name.
 - *E.g., public, class, static, void, etc.*
4. The standard naming convention:
 - Class names are **nouns** that start with an uppercase letter.
 - **Camel Case**: If a name consists of multiple words, use an initial uppercase letter in each of the words.
 - *E.g., HelloWorld, FirstSample.*

The braces { }

```
public class FirstSample
{
    public static void main(String[] args)
    {
        System.out.println("We will not use 'Hello, World!'");
    }
}
```

```
public class FirstSample {
    public static void main(String[] args) {
        System.out.println("We will not use 'Hello, World!'");
    }
}
```

FirstSample.java

```
public class FirstSample {  
    public static void main(String[] args) {  
        System.out.println("We will not use 'Hello, World!'");  
    }  
}
```

- The body of the **main** method contains a statement that outputs a single line of text to the console.
 - Here, we are using the **System.out** object and calling its **println** method.
 - The periods (".") are used to invoke a method.
 - A method can have zero, one or more parameters (arguments).
 - Parentheses are always needed even there is no parameters.
 - E.g., **System.out.println();**

Contents

- 3.1 A Simple Java Program
- 3.2 Comments
- 3.3 Data Types
- 3.4 Variables and Constants
- 3.5 Operators
- 3.6 Strings
- 3.7 Input and Output
- 3.8 Control Flow
- 3.9 Big Numbers
- 3.10 Arrays

Comments

- Three types:

1. `//` Single-line comments

2. `/*` Multi-line Comments

The second line of this comment `*/`

3. `/**`

`* This is used to generate documentation automatically`

`* @version 1.0 2021-03-19`

`* @author Harry Cui`

`*/`

- Comments can be used to explain Java code, and to make it more readable.
- Comments do not show up in the executable program.

Contents

- 3.1 A Simple Java Program
- 3.2 Comments
- 3.3 Data Types
- 3.4 Variables and Constants
- 3.5 Operators
- 3.6 Strings
- 3.7 Input and Output
- 3.8 Control Flow
- 3.9 Big Numbers
- 3.10 Arrays

Java is strongly typed

- All variable must be declared.
 - `<type> <variable>;`
 - *E.g., `int x;`*
- After a variable is declared, you can assign to it.
 - *E.g., `x = 4;`*
- We call a variable which has a class for a type an object.
 - *E.g., `Car c;`*
- Once an object is declared, you can
 - assign to it, often with a creation statement,
 - access its data members, and
 - call its methods.
 - *E.g., `c = new Car("BMW"); c.make = "Audi"; c.getMake();`*

3.3.1 Integer types

- For numbers without fractional parts.

Type	Storage	Range (Inclusive)
<code>int</code>	4 bytes	-2,147,483,648 to 2,147,483,647 (just over 2 billion)
<code>short</code>	2 bytes	-32,768 to 32,767
<code>long</code>	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
<code>byte</code>	1 byte	-128 to 127

- In Java, the ranges of the integer types do not depend on the machine on which you will be running the Java code.

Literals in Java

- A literal is a source code representation of a fixed value.
 - They are represented directly in the code without any computation.
 - Literals can be assigned to any primitive type variable.
 - **byte**, **int**, **long**, and **short** can be expressed in *decimal* (base 10), *hexadecimal* (base 16), *octal* (base 8) or *binary* (base 2) number systems.

```
byte a      = 65;  
int  decimal = 100;  
long num    = 100L;    // with suffix L or l  
int  octal   = 0144;    // with prefix 0  
int  hex     = 0x64;    // with prefix 0x or 0X  
int  bin     = 0b1100100; // with prefix 0b or 0B
```

Data type

Literal

3.3.2 Floating-point types

- For numbers with fractional parts.

Type	Storage	Range
float	4 bytes	Approximately $\pm 3.40282347\text{E}+38\text{F}$ (6-7 significant decimal digits)
double	8 bytes	Approximately $\pm 1.79769313486231570\text{E}+308$ (15 significant decimal digits)

- The name double refers to the fact that these numbers have twice the precision of the float type.

```
float  fNum = 3.14F; // with suffix F or f
double dNum = 3.14D; // with suffix D or d (optionally)
```

Caution: round-off error

```
System.out.println(2.0 - 1.1); // result: 0.8999999999999999 not 0.9
```

- Reason:
 - **Floating-point numbers are represented in the binary number system.** There is no precise binary representation of the fraction $1/10$, just as there is no accurate representation of the fraction $1/3$ in the decimal system.
- Solution:
 - Using the **BigDecimal** class if you need precise numerical computations.

```
System.out.println(BigDecimal.valueOf(2.0).subtract(BigDecimal.valueOf(1.1)));  
// result: 0.9
```

3.3.3 The char type

- Used for describing individual characters.

```
char ch = 'A';  
char tm = '\u2122'; // Unicode for the trademark symbol (™)
```

- 'A' is a character constant with a value of 65.
- "A" is a string containing a single character.

```
char ch1 = 'A';  
char ch2 = '\u0041'; // Unicode for the character A
```


Escape sequences for special characters

- Escape sequence is a character preceded by a backslash (\) and has a special meaning to the compiler.

Escape Sequence	Name	Unicode Value
<code>\b</code>	Backspace	<code>\u0008</code>
<code>\t</code>	Tab	<code>\u0009</code>
<code>\n</code>	Linefeed	<code>\u000a</code>
<code>\r</code>	Carriage return	<code>\u000d</code>
<code>\"</code>	Double quote	<code>\u0022</code>
<code>\'</code>	Single quote	<code>\u0027</code>
<code>\\</code>	Backslash	<code>\u005c</code>

Java backspace escape doesn't work?

<https://stackoverflow.com/questions/3328824/java-backspace-escape>

```
System.out.println("She said \"Hello!\" to me.");  
// output: She said "Hello!" to me.
```

3.3.4 Unicode and the char type

- Unicode is an information technology standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems.
 - It was invented to overcome the limitations of traditional character encoding schemes.
 - Before Unicode, there were many different standards: ASCII in the United States, BIG-5 for Chinese, etc.
 - *See more on “Unicode Encoding! UTF-32, UCS-2, UTF-16, & UTF-8!”*
<https://www.youtube.com/watch?v=uTJoJtNYcaQ>

Strong recommendation:

Not to use the char type in your programs unless you are actually manipulating UTF-16 code units. You are almost always better off treating strings as abstract data types.

ASCII vs Unicode in Java (13 min)

STANDARDS: ASCII vs UNICODE

```
public class DataTypes101
{
    public static void main(String[ ] args)
    {
        int    variable1;
        double variable2;
        char   variable3;
    }
}
```

Output:

NO Output

<https://www.youtube.com/watch?v=61Bs7-ycL64>

See more on <https://www.youtube.com/watch?v=ut74oHojxqo>

3.3.5 The boolean type

- Used for evaluating logical conditions.
 - Only two values: **true** or **false**
 - No conversion between integers and Boolean values.

```
boolean isJavaFun    = true;
boolean isJavaBoring = false;
System.out.println(isJavaFun);    // Outputs true
System.out.println(isJavaBoring); // Outputs false

// Boolean Expression
int x = 10;
int y = 9;
System.out.println(x > y);        // Outputs true
```

Primitive vs reference

- Types in Java are divided into two categories - primitive types and reference types.
 - Primitive types: **boolean**, **byte**, **char**, **short**, **int**, **long**, **float**, **double**.
 - All other types are reference types, so classes, which specify the types of objects, are reference types.
- A primitive-type variable can store exactly one value of its declared type at a time.
 - Primitive-type instance variables are initialized by default.
 - Variables of types **byte**, **char**, **short**, **int**, **long**, **float** and **double** are initialized to **0**.
 - Variables of type **boolean** are initialized to **false**.
- Reference-type variables (called references) store the location (address) of an object in the computer's memory.
 - Such variables refer to objects.



Primitive and Reference Types in Memory (5 min)

Memory Handling in Java



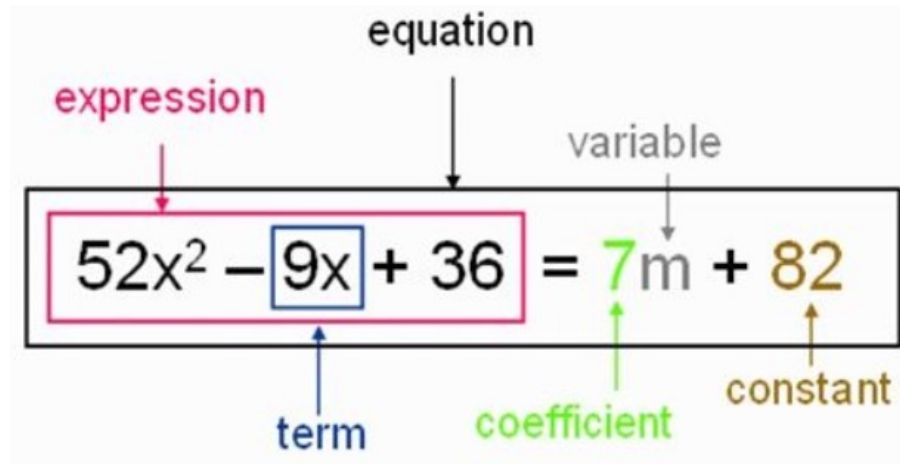
<https://www.youtube.com/watch?v=LTnp79Ke8FI>

Contents

- 3.1 A Simple Java Program
- 3.2 Comments
- 3.3 Data Types
- 3.4 Variables and Constants
- 3.5 Operators
- 3.6 Strings
- 3.7 Input and Output
- 3.8 Control Flow
- 3.9 Big Numbers
- 3.10 Arrays

Variables vs Constants

- A **constant** is a data item whose value cannot change during the program's execution.
 - Thus, as its name implies - the value is constant.
- A **variable** is a data item whose value can change during the program's execution.
 - Thus, as its name implies - the value can vary.



<https://byjus.com/maths/variables-and-constants-in-algebraic-expressions/>

3.4.1 Declaring variables

- General form: **type** **variableName**;

```
double salary;  
long earthPopulation;  
boolean done;  
int i, j;           // correct but not recommended
```

- A variable name must *begin with a letter* and must be a sequence of letters or digits.
 - **letter**: 'A'–'Z', 'a'–'z', '_', '\$', or any Unicode character that denotes a letter in a language.
 - **digit**: '0'–'9' and any Unicode characters that denote a digit in a language.
 - Symbols like '+' or '@' cannot be used inside variable names, nor can spaces.
 - **Case-sensitive**, e.g., *"aNum"* and *"ANum"* are different.

3.4.2 Initializing variables

- You must **explicitly initialize** it by means of an assignment statement.
 - You can never use the value of an uninitialized variable.
 - Otherwise, you would see an **ERROR**, “*variable not initialized*”.
- Using an equal sign =

```
int vacationDays;  
vacationDays = 12;  
int vacationDays = 12; // correct but not recommended
```

- Good style: declare variables as closely as possible to the point where they are first used.

Local type inference in Java 10

- Can use **var** instead type for local variables:

```
var counter = 0; // an int  
var message = "Greetings, earthlings!"; // a String
```

- Still strongly typed:

```
counter = 0.5; // Error: can't assign a double to an int
```

- Useful for unwieldy type names:

```
var traces = Thread.getAllStackTraces(); // a Map<Thread,  
StackTraceElement[]>
```

3.4.3 Constants

- Using **final** to denote a constant.

```
final double PI = 3.14;  
final int VACATION_DAYS = 12;
```

- Good style: name it in all uppercase with words separated by underscores ("_").