西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

Object-Oriented Programming

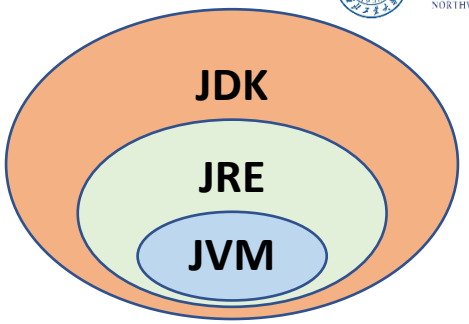
Chapter 3
Fundamental Programming Structures in Java

Dr. Helei Cui

19 Mar 2021

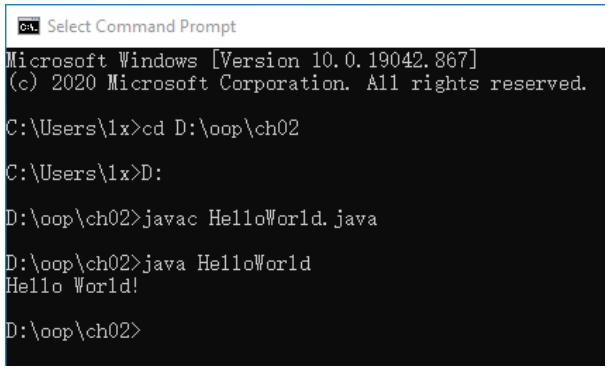
Slides partially adapted from lecture notes by Cay Horstmann

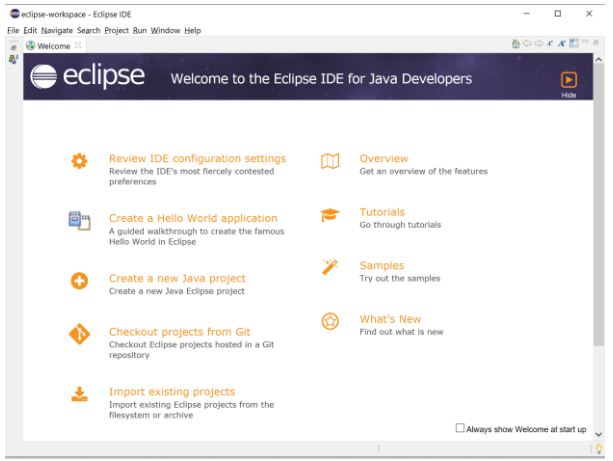
Recap



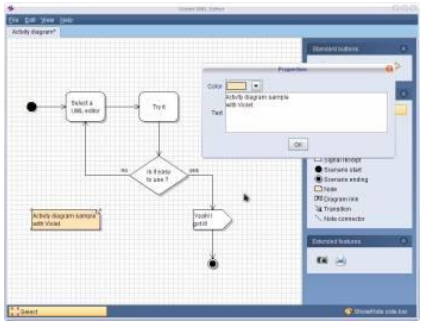
- **JDK, JRE, JVM?**
  - JDK is a software development kit
  - JRE is a software bundle that allows Java program to run
  - JVM is an environment for executing bytecode

- **Run HelloWorld.java**
  - Using Command-Line Tools
  - Using an IDE, e.g., Eclipse





- **Use Violet UML editor**



Contents

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

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

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)
`int`	4 bytes	-2,147,483,648 to 2,147,483,647 (just over 2 billion)
`short`	2 bytes	-32,768 to 32,767
`long`	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
`byte`	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 ±3.40282347E+38F (6-7 significant decimal digits)
`double`	8 bytes	Approximately ±1.7976931348623157E+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
\b	Backspace	\u0008
\t	Tab	\u0009
\n	Linefeed	\u000a
\r	Carriage return	\u000d
\"	Double quote	\u0022
\'	Single quote	\u0027
\\	Backslash	\u005c

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)

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

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)

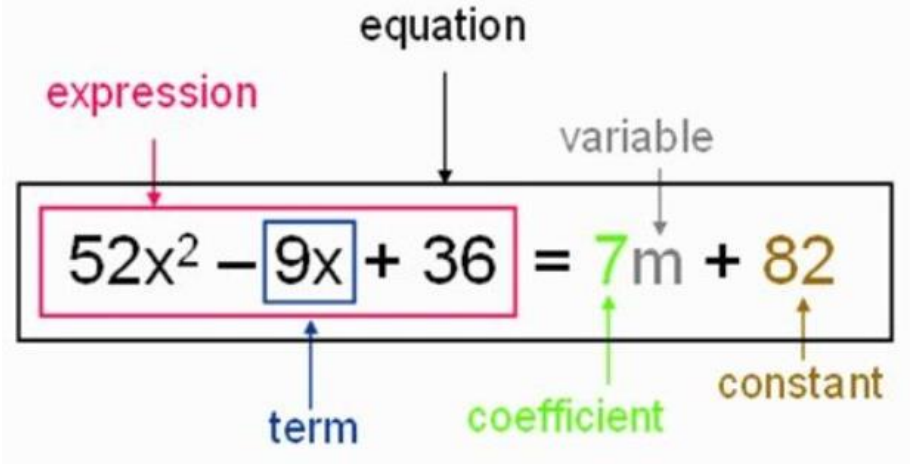
Memory Handling in Java

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

Contents

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 ("_").

Class constants

- Using `static final` to create a constant so it's available to multiple methods inside a single class.

```
public class Constants2 {
    public static final double CM_PER_INCH = 2.54;
    public static void main(String[] args) {
        double paperWidth = 8.5;
        System.out.println("Paper width in centimeters: "
           + paperWidth * CM_PER_INCH);
    }
}
```

- If the constant is declared `public`, other classes can use it like *`Constants2.CM_PER_INCH`*.

3.4.4 Enumerated types

- To describe a variable that only hold a restricted set of values.
  - E.g., sizes of pizza: small, medium, large, and extra large.
  - Only a finite number of named values.

```
public class Example {
    enum Size { SMALL, MEDIUM, LARGE, EXTRA_LARGE };
    public static void main(String[] args) {
        // Now you can declare variables of this type:
        Size s = Size.MEDIUM;
        System.out.println(s); // Outputs: MEDIUM
    }
}
```

  - Variable of type `s` can only hold `size` values or `null`.

Contents

3.5.1 Arithmetic operators

- The usual arithmetic operators:
  - Addition +
  - Subtraction -
  - Multiplication *
  - Division /
    - Integer division if both arguments are integers, and floating-point division otherwise.
    - *E.g., 15 / 2 is 7, 15.0 / 2 is 7.5.*
  - Integer remainder (a.k.a., modulus) %
    - *E.g., 15 % 2 is 1.*

Quick question 1

-7 % 3 = ?

A. 1
B. -1
C. -2

7 % -3 = ?

A. 1
B. -1
C. -2

% in Java

7 % 3 = 1

-7 % 3 = ?

7 % -3 = ?

- Using the formula **a % b = a – a / b * b**, you can get

  7 % 3 = 7 – 7 / 3 * 3 = 7 – 2 * 3 = 1

  -7 % 3 = -7 – (-7) / 3 * 3 = -7 – (-2) * 3 = -1

  7 % -3 = 7 – 7 / (-3) * (-3) = 7 – (-2) * (-3) = 1

Division /

- **In Python, the integer division uses "floor function":**

  -7 % 3 = -7 - floor(-7 / 3) * 3 = -7 - (-3) * 3 = -7 + 9 = 2

  7 % (-3) = 7 - floor(7 / (-3)) * (-3) = 7 - (-3) * (-3) = 7 - 9 = -2

The floor function takes as input a real number x, and gives as output the greatest integer less than or equal to x, denoted floor(x).

- **In Java or C, the integer division uses "truncation":**

  -7 % 3 = -7 - trunc(-7 / 3) * 3 = -7 - (-2) * 3 = -7 + 6 = -1

  7 % (-3) = 7 - trunc(7 / (-3)) * (-3) = 7 - (-2) * (-3) = 7 - 6 = 1

For positive real numbers, truncation is done using the floor function. But for negative numbers, truncation always rounds towards zero.

Quick question 2

How to get the ones, tens, hundreds, and thousands digit in the number 1,234?

`x/1000` ---> thousands digit

`x%10` ---> ones digit

`x/10%10` ---> tens digit

`x/100%10` ---> hundreds digit

3.5.2 Mathematical functions and constants

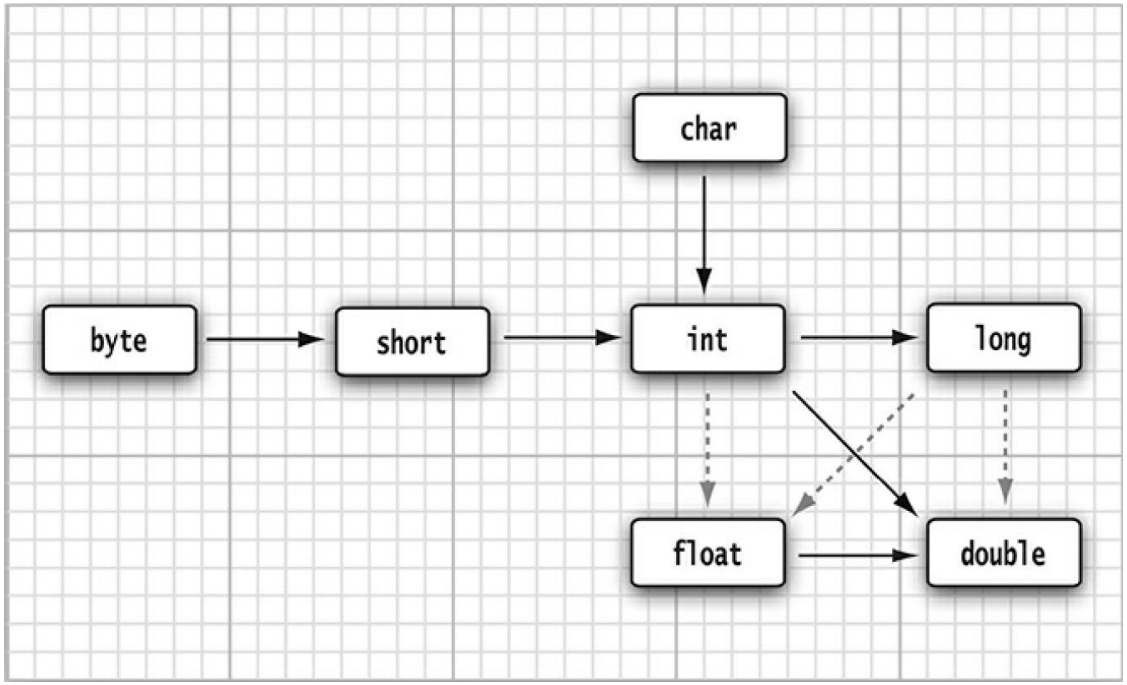
- The `Math` class contains methods for performing basic numeric operations and constants π and e.
  - E.g., the elementary exponential, logarithm, square root, and trigonometric functions.

```
double x = 4;
double y = Math.sqrt(x);
System.out.println(y);        // prints 2.0
System.out.println(Math.PI); // prints 3.141592653589793
System.out.println(Math.E);  // prints 2.718281828459045
```

The `println` method operates on the `System.out` object. But the `sqrt` method in the `Math` class does not operate on any object, which is called a **static method**.

3.5.3 Conversions between numeric types

- These conversions are automatic:



Dotted arrows indicate possible precision loss.

```
int n = 123456789;
float f = n; // f is 1.23456792E8
```

Rules

- When two values are combined with a binary operator (such as `n + f` where `n` is an integer and `f` is a floating-point value), both operands are converted to a common type before the operation is carried out.
  - If either of the operands is of type `double`, the other one will be converted to a `double`.
  - Otherwise, if either of the operands is of type `float`, the other one will be converted to a `float`.
  - Otherwise, if either of the operands is of type `long`, the other one will be converted to a `long`.
  - Otherwise, both operands will be converted to an `int`.

3.5.4 Casts

- Conversions in which loss of information is possible are done by means of casts.
  - The syntax for casting is to give the target type in parentheses, followed by the variable name.

```
double x = 9.997;
int nx = (int) x; // nx is 9
```

  - Use the `Math.round` method to round a floating-point number to the nearest integer.

```
double x = 9.997;
int nx = (int) Math.round(x); // nx is 10
```

https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/Math.html

Caution

- If you try to cast a number of one type to another that is out of range for the target type, the result will be a truncated number that has a different value.

```
byte x = (byte) 300;
System.out.println(x); // Outputs 44
```

- The number 300 in binary form is 100101100, then byte type only gets 8 digits. So, the x only gets 00101100, i.e., 44.