Object Oriented and Java Programming

Course 2

Qiong Liu

qiong.liu@cyu.fr
CY Tech, CY Cergy Paris University

October 2024

Contents

Outline

Recall on the Object Oriented Programming

Everything is an object!

- Java is more "pure" object-oriented language, compared with C++.
- Object-oriented programming has 4 important characteristics:
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism

Visibility	**Class**	**Attributes/Methods**
Public	All	All
Protected	NA	Classes and subclass in the package
Package/Default	Class of package	Class of package
Private	NA	Inside Class

Java main structure

HelloWorld.java (same as main class name)

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!"); // Hello, World!
    }
}
```

- Global variable and local variable declaration;
- Main method: start with "{" and end with "}". "main()" method must be `public static void`
- `Sting [] args`: parameter of main class. An array of String objects, where each element of the array is a string.

Java main structure

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!"); // Hello, World!
    }
}
```

- `public`: Access modifier
  - It makes the method accessible from any other class
  - Usage: the `main` method must be `public` because it is the entry point of the program, and the JVM needs to call it from outside the class
- static: Static modifier
  - The method belongs to the class, not an instance
- void: Return type
  - The method does not return any value.

Creating New Class File

- You must have the same class name as the file name.
- Java is case sensitive.
- By convention, class names begin with a capital letter and capitalize the first letter of each word they include (e.g., `SampleClassName`).
- To execute a class file, we must have `public static void main(String[] args)`.
  It serves as the starting point for the Java Virtual Machine (JVM) to begin executing your program.
  - The `public` keyword means that the method is available to the outside world.
  - The argument to `main()` is an array of `String` objects.
  - The `args` won't be used in this program, but the Java compiler insists that they be there because they hold the arguments from the command line.

Comments in Java

There are 3 types of commenting ways in Java:

1. **Single Line comment:**

```
1 // This is a line comment
```

2. **Block comment:**

```
1 /*
2  * This is a block comment
3  */
```

3. **Documentation Comment:**

```
1 /**
2  * java documentation comment
3  */
```

Data types.

Integer Types

```
int x;
int x,y;
int x = 10, y = -5;
int x = 5+ 23;
```

- `int` (4 bytes): -2^{31} to 2^{31}-1
- `byte` (1 byte): -128 to 127
- `short` (2 bytes): -32,768 to 32,767
- `long` (8 bytes): -2^{63} to 2^{63}-1

Integer Types

Pay attention:

- Decimal (Base 10):
  - Default number system.
  - No prefix required.
- Octal (Base 8):
  - Uses prefix `0`.
  - Equivalent to decimal 83.

```
1        int octalNumber = 0123;
```

- Hexadecimal (Base 16):
  - Uses prefix `0x`.
  - Equivalent to decimal 291.

```
1        int hexNumber = 0x123;
```

Float Types

```
float f1=13.23f;
double d1 = 4562.12d;
```

- **float** (4 bytes): Single precision, 7 decimal digits
- **double** (8 bytes): Double precision, 16 decimal digits

Pay attention:

- Float numbers are approximate due to limited bits for representation;
- Rounding errors occur when numbers can't by represented exactly;
- Bias refers to the small errors that can accumulate, making the final result slightly inaccurate. (double has more bits and thus a smaller bias)

Example 1: Basic char Declaration

```
public class CharExample {
    public static void main(String[] args) {
        char letter = 'A';  // A single character 'A'
        System.out.println("Character is: " + letter);
    }
}
```

Output: `Character is:  A`

Real-world Use: Handling single characters such as letters, symbols, or input validation.

Example 2: Unicode Character

'char' is used to define a single character and occupies 16 bits of memory space.

```
public class UnicodeExample {
    public static void main(String[] args) {
        char unicodeChar = '\u263A';  // Unicode for
            smiley face
        System.out.println("Unicode character is: " +
            unicodeChar);
    }
}
```

Output: Unicode character is:☺
Real-world Use: Displaying symbols or international characters in applications.

Example 3: Convert char to Numeric Value

```
public class CharToIntExample {
    public static void main(String[] args) {
        char digit = '5';
        int numericValue = Character.getNumericValue(
            digit);
        System.out.println("Numeric value is: " +
            numericValue);
    }
}
```

Output: `Numeric value is: 5`

Real-world Use: Processing numeric input from characters, such as form fields.

Example 4: char with Strings

```
public class CharInStringExample {
    public static void main(String[] args) {
        String str = "Hello";
        char firstChar = str.charAt(0);  // Get first
            character
        System.out.println("First character is: " +
            firstChar);
    }
}
```

Output: First character is: H

Real-world Use: Extracting and analyzing individual characters in strings.

Java Escape Characters

Escape Sequence	Character	Description
\n	Newline	Moves cursor to the next line
\t	Tab	Inserts a tab character
\b	Backspace	Inserts a backspace character
\f	Form Feed	Advances to the next page
\r	Carriage Return	Moves cursor to the start of the current line
\"	Double Quote	Inserts a double-quote mark
\'	Single Quote	Inserts a single-quote mark
\uXXXX	Unicode	Represents a Unicode character

Table: Common Java Escape Characters

Boolean in Java

- The ‘boolean‘ data type has two possible values: ‘true‘ and ‘false‘.
- It is used for simple flags that track true/false conditions.
- The default value for a ‘boolean‘ variable is ‘false‘.
- Example usage:

```
public class BooleanExample {
    public static void main(String[] args) {
        boolean isJavaFun = true;
        boolean isFishTasty = false;

        System.out.println("Is Java fun? " +
            isJavaFun);
        System.out.println("Is fish tasty? " +
            isFishTasty);
    }
}
```

Variables, Constants, and Scope.

Identifiers in Java and Examples

Identifiers are names used for variables, constants, methods, classes...
Rules for Identifiers:

- Must begin with a letter (A-Z, a-z), underscore (‘_‘), or dollar sign (‘$‘).
- After the first character, they can contain letters, digits (0-9), underscores, or dollar signs.
- Cannot start with a digit (e.g., ‘3days‘ is not allowed).
- Identifiers should be descriptive and meaningful to improve code readability.

Examples of Valid Identifiers:

- `age`, `salary`, `_count`, `name123`, `$value`

Examples of Invalid Identifiers:

- `my-name` (Hyphen ‘-‘ is not allowed)
- `void,byte,abstract` (Reserved keyword)
- `first name` (Spaces are not allowed)

Declaring Constants and Variables

Declaring Variables:

- A variable is a storage location in memory with a specific type.
- Variables are declared follow the **camelCase** convention.
  - The first word is in lowercase, and each subsequent word starts with an uppercase letter.

Syntax:

```
type variableName = value;
```

Example:

```
int age = 25;                    // Integer variable
double mySalary = 45000.50; // Double variable
String name = "Alice";      // String variable
```

Declaring Constants and Variables

Declaring Constants:

- A constant is a variable whose value cannot change once assigned.
- Constants are declared using the 'final' keyword.
- Usually we use UPPERCASE

Syntax:

```
final type CONSTANT_NAME = value;
```

Example:

```
final int MAX_AGE = 100;        // Constant integer
final double PI = 3.14159;      // Constant double
```

Variable: Static, Instance, and Local Variables

1. Static Variables (Class Variables):
   - Declared with the static keyword inside a class.
   - Shared among all instances of the class (only one copy exists).
   - Accessible from both static and non-static methods.
2. Instance Variables:
   - Declared inside a class but outside any method.
   - Each object of the class has its own copy of the instance variable.
   - Accessible only through an object of the class (non-static methods).
3. Local Variables:
   - Declared inside a method, constructor, or block of code.
   - Only accessible within that method or block.
   - Created when the method is called and destroyed when it returns.

Variable Scope in Java: Static, Instance, and Local Variables

Example:

```
public class Val {
    static int times = 3;    // Static variable
    String instanceName;          // Instance variable

    public static void main(String[] args) {
        int times = 4;
        System.out.println("The value of times is  " +
            times);
    }
}
```

The result: 4

Operators.

Arithmetic Operators

Arithmetic Operators:

- Plus: +
- Subtraction: -
- Multiplication: *
- Division: /
- Remainder: %
- **Almost** all operators work only with primitive types.
- The exceptions are =, ==, and != , as they work with both primitive types and objects.
- For more advanced mathematical functions (e.g., exponents, logarithms), refer to the Java Math documentation https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/Math.html.

Generating Random Numbers

- Example of creating a Random object:

```
Random rand = new Random();  // Seeded by current time
int randomNumber = rand.nextInt(100);  // Generate a
    random integer from 0 to 99
System.out.println(randomNumber);  // Print the random
     number
```

- Each execution of the program will produce a different random number.

Pre-increment vs Post-increment

- **Pre-increment (‘++a‘)**: Increment ‘a‘, then assign the value.
- **Post-increment (‘a++‘)**: Assign the value of ‘a‘, then increment.

Example:

```
int a = 5;
int b = ++a;   // Pre-increment: b = 6, a = 6
int c = a++;   // Post-increment: c = 6, a = 7
```

Explanation:

- In ‘b = ++a‘, ‘a‘ is incremented first, then assigned to ‘b‘. Thus, ‘b = 6‘ and ‘a = 6‘.
- In ‘c = a++‘, the current value of ‘a‘ is assigned to ‘c‘ first, and then ‘a‘ is incremented. Thus, ‘c = 6‘ and ‘a = 7‘.

Comparison Operators

- == (Equal to): Compares if two values are equal.
- != (Not equal to): Compares if two values are not equal.
- < (Less than): Checks if the left operand is less than the right operand.
- > (Greater than): Checks if the left operand is greater than the right operand.
- <= (Less than or equal to): Checks if the left operand is less than or equal to the right operand.
- >= (Greater than or equal to): Checks if the left operand is greater than or equal to the right operand.

Other Operators

Logical Operators

- && (Logical AND): Returns true if both conditions are true.
- || (Logical OR): Returns true if at least one condition is true.
- ! (Logical NOT): Inverts the value of a boolean expression.

Shift Operators:

- << (Signed Left Shift): Moves all bits by a given number of bits to the left.
- >> (Signed Right Shift): Moves all bits by a given number of bits to the right.
- >>> (Unsigned Right Shift): Same as >>, but the vacant leftmost position is filled with 0 instead of the sign bit.

Logical Operators in Java

- Three main logical operators for boolean expressions:
  - AND (&&): Returns true if both operands are true.
  - OR (||): Returns true if at least one operand is true.
  - NOT (!): Reverses the logical state of the operand.
- These operators are used to combine multiple boolean conditions.

Example Code:

```
boolean a = true;
boolean b = false;

boolean result1 = a && b; // AND: false
boolean result2 = a || b; // OR: true
boolean result3 = !a;     // NOT: false
```

Data Type Conversion

- Java supports two types of type conversions:
  - **Implicit conversion (Widening)**: Automatically converts smaller data types to larger types.
    - Example: `int` to `double`
  - **Explicit conversion (Narrowing)**: Requires casting, converting larger data types to smaller ones.
    - Example: `double` to `int`

Example Code:

```
int i = 10;
double d = i; // Implicit conversion

double x = 10.5;
int y = (int) x; // Explicit conversion
```

Name rules.

Package Naming Rules

- Use all lowercase letters.
- Use reverse domain name to avoid conflicts (e.g., `com.example.project`).
- Sub-packages reflect the project hierarchy and contents.
- Example: `com.example.util`, `com.example.network`

Class Naming Rules

- Use Pascal Case (Camel Case starting with an uppercase letter).
- Classes are usually nouns or noun phrases.
- Clearly reflect their purpose or the entity they represent.
- Example: `Student`, `CustomerService`

Method Naming Rules

- Use Camel Case starting with a lowercase letter.
- Start with a verb to indicate action.
- Should clearly indicate what the method does.
- Example: `calculateTotal`, `printDetails`

Instance and Attribute Naming Rules

- Use Camel Case starting with a lowercase letter.
- Names should be descriptive and indicative of their purpose.
- Avoid abbreviations unless they are widely understood.
- Boolean variables often start with `is`, `has`, or `can`.
- Example: `firstName`, `isAvailable`

Exercise.

Exercise 1

Exercise 1: Comparing Integers with Logical Operators

- Create a class called `Calculation`.
- In the `main` method, declare three integers:
  - `maleCount` (number of boys)
  - `femaleCount` (number of girls)
  - `totalPeople` (total number of people)
- Use comparison and logical operators to check the following conditions:
  - If the number of boys is greater than the number of girls and the total number of people is greater than 30.
  - If the number of boys is greater than the number of girls or the total number of people is greater than 30.

Exercise 1

Tipes:

```
class Calculation {
  public static void main(String[] args) {
    int maleCount = ;

    // Check: Are there more boys than girls and total
        people > 30?
    if () {
    System.out.println("More boys than girls, and
       total people > 30");
    }
    // Check: Are there more boys than girls or total
       people > 30?
    if ( ) {

    }
  }
}
```

Exercise 1

```
class Calculation {
  public static void main(String[] args) {
    int maleCount = 20;
    int femaleCount = 15;
    int totalPeople = maleCount + femaleCount;

    // Check condition 1
    if (maleCount > femaleCount && totalPeople > 30) {
      System.out.println("More boys than girls, and
          total people > 30");
    }
    // Check condition 2
    if (maleCount > femaleCount || totalPeople > 30) {
      System.out.println("More boys than girls, or
          total people > 30");
    }
  }
}
```

Exercise 2

Exercise 2: Generate Random Object and Comparing Integers with Logical Operatoors

- Change the previous class `Calculation`.
- In the `main` method:
  - Set `maleCount` to 20 (number of boys).
  - Generate a random integer between 10 and 40 for `femaleCount` (number of girls).
  - Calculate `totalPeople` as the sum of `maleCount` and `femaleCount`.
- Use logical operators to check the following conditions:
  - If the number of boys is greater than the number of girls **and** the total number of people is greater than 30.
  - If the number of boys is greater than the number of girls **or** the total number of people is greater than 30.

Exercise 2

```
import java.util.Random;

class Calculation {
  public static void main(String[] args) {
    int maleCount = 20;

    // Generate random number of girls
    Random rand = new Random();
    int femaleCount = rand.nextInt(31) + 10;

    int totalPeople = maleCount + femaleCount;

    // Check: condition 1
    if (maleCount > femaleCount && totalPeople > 30) {
      System.out.println("More boys than girls, and
          total people > 30");
    }
```

Exercise 3: Generate a Remittance Slip

- Create a class called `RemittanceSlip`.
- Use the following data types:
  - `String`: Account holder, RIB, bank name.
  - `double`: Amount of the transaction.
  - `int`: Bank ID.
  - `boolean`: Priority transfer status.
  - `char`: Transfer type (`'C'` for credit).
  - `char`: Currency symbol (EURO).

Tips: char currencySymbol = '\u20AC'

Exercise 3: Generate a Remittance Slip (Part 1)

```
import java.time.LocalDate;

class RemittanceSlip {
  public static void main(String[] args) {
    // Variables
    String bankName = "Global Bank";
    int bankID = 12345;
    LocalDate date = LocalDate.now();
    String accountHolder = "Anna";
    String RIB = "FR1234567812345678";
    char transferType = 'C';  // 'C' for credit
    char currencySymbol = '\u20AC';  // Euro symbol
    double amount = 1000.00;
    boolean priorityTransfer = true;

    // Output remittance slip
```

Exercise 3: Generate a Remittance Slip (Part 2)

```
        System.out.println("----- Remittance Slip -----");
        System.out.println("Bank Name: " + bankName);
        System.out.println("Bank ID: " + bankID);
        System.out.println("Date: " + date);
        System.out.println("Account Holder: " +
            accountHolder);
        System.out.println("Account Number: " + RIB);
        System.out.println("Transfer Type: "
            + (transferType == 'D' ? "Debit" : "Credit"));
        System.out.println("Currency: " + currencySymbol);
        System.out.println("Amount: " + amount + " " +
            currencySymbol);
        System.out.println("Priority Transfer: "
            + (priorityTransfer ? "Yes" : "No"));
        System.out.println("-----------------------------");
    }
}
```

Outline

Compound Statements in Java

What is a Compound Statement?

- A compound statement in Java is a block of code that groups multiple statements together
- Enclosed within braces { }
- Treated as a single unit, typically in control flow structures like `if`, `for`, `while`, etc.

Syntax:

```
{
    // statement 1
    // ...
    // statement n
}
```

- Only one statement each line to let your code clear.

Controlling Execution

All conditional statements use the truth or falsehood of a conditional expression to determine the execution path. An example of a conditional expression is $a == b$. This uses the conditional operator $==$ to see if the value of a is equivalent to the value of b. The expression returns true or false. **Types:**

- If-else
- Switch
- Loopings
  - for loops
  - while loops
  - do-while loops

If

What is an If Statement?

- The `if` statement is used to execute a block of code only if a specified condition is true.

Syntax:

```
if (condition) {
    // code to be executed if condition is true
}
```

Example:

```
int x = 10;
if (x > 5) {
    System.out.println("x is greater than 5");
}
```

If-Else

What is an If-Else Statement?

- The `if-else` statement provides an alternative block of code to execute if the condition is false.

Syntax:

```
if (condition) {
    // code to be executed if condition is true
} else {
    // code to be executed if condition is false
}
```

If-Else

Example:

```
public void setY(int y) {
    if (y >= -500 && y <= 500) {
        this.y = y;
    } else {
        throw new IllegalArgumentException("New y
            value " + y + " out of range");
    }
}
```

Explanation:

- `IllegalArgumentException`: An exception that is part of the Java standard library and is used to indicate that a method has been passed an argument that is of an inappropriate or illegal value.

If-Else-If

What is an If-Else-If Ladder?

- The `if-else-if` ladder is used to test multiple conditions.
- The first true condition's block is executed, and the rest are skipped.

Syntax:

```
if (condition1) {
    // code to be executed if condition1 is true
} else if (condition2) {
    // code to be executed if condition1 is false and
       condition2 is true
} else {
    // code to be executed if all conditions are false
}
```

If-Else-If

Example:

```
int x = 10;
if (x < 0) {
    System.out.println("x is negative");
} else if (x == 0) {
    System.out.println("x is zero");
} else {
    System.out.println("x is positive");
}
```

Nested If Statements

What is a Nested If Statement?

- A nested `if` statement is an `if` statement inside another `if` statement.
- This allows for multiple layers of conditions to be tested.

Syntax:

```
if (condition1) {
    if (condition2) {
        // code to be executed if both conditions are
            true
    }
}
```

Example: Nested If Statements

```
int score = 85;
char grade;
if (score >= 90) {
    grade = 'A';
} else {
    if (score >= 80) {
        if (score >= 85) {
            grade = 'B+';
        } else {
            grade = 'B';
        }
    } else {
        grade = 'C';
    }
}
System.out.println("Grade: " + grade);
```

Switch

The switch is sometimes called a selection statement. The switch statement selects from among pieces of code based on the value of an integral expression.

```
switch (integral-selector) {
    case integral-value1: statement; break;
    case integral-value2: statement; break;
    case integral-value3: statement; break;
    //...
    default:
        // Code to be executed if no case matches
}
```

- Each case must end with break to prevent fall-through.
- The default case is optional and runs if no other case is matched.

Example: Switch

```
import java.util.Scanner; // introduce Scanner class
public class Grade {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter your score: ");

        int grade = sc.nextInt();
        String result = switch (grade) {
            case 10, 9 -> "Good";
            case 8 -> "Well";
            case 7, 6 -> "Middle";
            case 5, 4, 3, 2, 1, 0 -> "Do not pass";
            default -> "Nul";
        };
        System.out.println(result);
        sc.close();  //  the sc.close statement
    }
}
```

Branching Statements

Break Statement:

- Used to exit a loop prematurely or terminate a `switch` statement.

Continue Statement:

- Skips the current iteration of a loop and proceeds to the next one.

Return Statement:

- Used to exit a method and optionally return a value to the caller.

Throw Statement:

- Explicitly throws an exception that can be caught by an exception handler.

While and Do-While

What is a While Loop?

- The `while` loop repeatedly executes a block of code as long as a specified condition is true.
- It checks the condition before each iteration, so if the condition is false at the start, the loop body won't execute at all.

Syntax:

```
while (Boolean-expression) {
    // Code to be executed while condition is true
}
```

Key Points:

- Make sure the condition eventually becomes false to avoid infinite loops.

While and Do-While

What is a Do-While Loop?

- The `do-while` loop is similar to the `while` loop, but the condition is checked after the code block is executed.
- This guarantees that the code block is executed at least once, regardless of the condition.

Syntax:

```
do {
    // Code to be executed
} while (Boolean-expression);
```

Key Points:

- The block of code is always executed at least once.
- The condition is evaluated after the execution of the block.

While vs Do-While

While Loop Example

```
// While loop
int count1 = 5;
while (count1 < 5) {
    System.out.println("While Count: " + count1);
    count1++;
}
```

Do-While Loop Example

```
// Do-while loop
int count2 = 5;
do {
    System.out.println("Do-While Count: " + count2);
    count2++;
} while (count2 < 5);
```

For

What is a For Loop?

- A `for` loop is used to execute a block of code a fixed number of times.
- It is commonly used for iterating over arrays or performing a task with a known number of iterations.

Syntax:

```
for (initialization; Boolean-expression; step) {
    // Code to be executed
}
```

Using the comma operator, you can define multiple variables within a for statement, but they must be of the same type:

```
for (int i = 1; j=i + 10; i < 5; i++, j= i *2) {
    System.out.println("i= " + i + " j = " + j);
}
```

Nested Loops

What are Nested Loops?

- A nested loop is a loop inside another loop.
- The inner loop is executed fully each time the outer loop runs once.
- Useful for working with multi-dimensional arrays or performing repeated tasks within repeated tasks.

Syntax:

```
for (initialization; Boolean-expression; step) {
    for (initialization; Boolean-expression; step) {
        // Code to be executed
    }
}
```

Nested Loops

Example:

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 3; j++) {
        System.out.println("i = " + i + ", j = " + j);
    }
}
```

Explanation:

- The outer loop controls the variable `i`, running from 1 to 3.
- For each iteration of the outer loop, the inner loop runs fully, controlling the variable `j` from 1 to 3.
- The result prints every combination of `i` and `j`.

Exercise.

Exercise

- Create a class "compute", it calculate the area of a triangle, given parameters: length of base and height.
- Following "compute", implement calculation of area for 3 scenarios, triangle, rectangle, and square, using switch.
- Use loops to simulate throwing a dice. A dice has 6 faces, containing value from 1 to 6. Assuming the dice has been thrown 10 times. Show the output of each throw.
- Use While to simulate: Exit when you get a value of 6. And print the number of times required to get 6. *Hint: use class Random. and Incremental Operator.
- Generate two variables, representing values obtained from two dices. Compare their value, and show which one wins.