Self-Learning Material #4

Scopes and IO

By Kevin 2022/2023

Variable Scope

What is a variable scope

- A project is usually combining code segments written by multiple programmers.
- Each piece of code segments have their own variables.

👨‍🏫💬 What if these code segments share the same variable name?

- A variable scope makes **variables does not interferer with each other**.

image credit: https://codeweek.eu/resources/CodingAtHome

Understanding a class

- All codes are inside a class (a project can have many classes).
- Variables can be declared as
  - ① a **member variable** or a **field**.
  - ② a **parameter**.
  - ③ a **local variable**.

```
public class ScopeEx {
    int field_x; ①
    String field_y; ①

    public static void main(String[] a) ② {
      new ScopeEx().method(10);
    }

    public void method(int para_x) ② {
        int local_x; ③
        {
            int local_inner_x;  ③
        }
    }
}
```

- The scope of a variable means the area where the variable is visible.

① Scope of member variables

- The scope of a **member variable/field** is the entire class.

```
public class ScopeEx {
  int field_x = 10;
  String field_y = field_x + "-string" ; //OK

  public static void main(String[] a) {
    new ScopeEx().method(10);
  }
  public void method(int para_X) {
    System.out.println(field_y); //OK
    ...
  }
}
```

② Scope of parameters

- Parameters are closely related to methods (known as function in other programming language).
- The scope of a **parameter** is the entire method.

```
public class ScopeEx {

  public static void main(String[] para_a) {
    System.out.println(para_a); //OK
    new ScopeEx().method(10);
    System.out.println(para_x); //invalid, para_x is invisible here
  } //end of scope of para_a

  public void method(int para_x) {
    System.out.println(para_x); //OK
    System.out.println(para_a); //invalid
    ...
  } //end of scope of para_x
}
```

③ Scope of local variables

- The scope of a local variable is defined within its immediate parent `{ }`.

```
public void method(int para_x) {
  int local_x = 10;
  {
    int local_y = 5;
    System.out.println(local_x + local_y); //OK
  } //end of scope of local_y
  System.out.println(local_x); //OK
  System.out.println(local_y); //invalid
} //end of scope of local_x
```

③ Scope of local variables - Example 2

```
public void method2() {
  int local_x = 10;
  if (local_x > 0) { //OK
    int local_y = 5;
    if (local_x > local_y) { //OK
      int local_z = 2;
    } else { //end of local_z 's scope
      local_y = local_z; // invalid, local_z not visible
    }
  } //end of local_y's scope
  if (local_x < 20) {
    int local_y = 5; //this is another local_y!

  } //end of local_y's scope
  local_x = local_y; //invalid. Both local_y end
} //end of local_x's scope
```

Rules about Scopes

Rules to remember when handling variable scope:

1. Local variables and parameters are **invisible outside** their scope.
2. **Reuse a variable name** outside its scope is always **allowed**.
3. The value of the variable will be lost when it is outside its scope.
4. **Defining local variables** with the same variable name in an overlapped scope is **disallowed**.
5. Local variables/parameters has **a higher precedence** than fields of a class.

Same variable name in overlapped scopes

- You cannot define a **local variable** if a **parameter** or another **local variable** of the **same name** is visible here.

```
public void method2() {
  int local_x = 10;
  if (local_x > 0) {
    int local_x; //invalid
  }
  int local_x; //invalid
} //end of local_x's scope
```

```
public void method3(int x) {
  int x; //invalid, x is already defined as parameter
}
```

- You can, however, name a local variable or a parameter even if the name has been taken by **a field**.

Fields of a class being preceded

- Name crash between a field and a local variable: local variable is referred.
- Name crash between a field and a parameter: parameter is referred.
- Name crash between a local variable and a parameter: not allowed.

```
int x = 1;
public void method3(int x) { //OK
  System.out.println(x); //print the parameter x
}

public void method4() {
  int x = 10; //OK
  System.out.println(x); //print 10
}
```

this

- You can use the keyword `this` to explicitly refer a field.
- e.g. `this.field_x`;

```
int x = 1;

public void method4() {
  int x = 10; //OK
  System.out.println(x); //print 10
  System.out.println(this.x); //print 1
}
```

Simple I/O

Inputs and Outputs

Java supports the following Input and Output (I/O) methods:

- Command Line Interface (CLI)
- Graphic User Interface (GUI) - see Javax.swing, javafx
- File I/O - see Java.io
- Network I/O - see Java.net and Java.io

We will mainly cover CLI and probably some file I/O too.

Print

Three main commands that outputs text and number in on a CLI.

1. `System.out.print` - print content
2. `System.out.println` - print content followed by a new line
3. `System.out.printf` - print content with a specific format

`System.out.print` and `System.out.println` can be used like

```
String a = "Hello", b = "COMP";
int c = 2026;
System.out.print(a); //no need line
System.out.println(b + c);
System.out.print("new line");
```

```
Hello_
```

```
HelloCOMP2026
_
```

```
HelloCOMP2026
new line_
```

_ : the cursor.

Output

- `System.out.printf` support a different syntax:

```
System.out.printf(<formatted string>, var1, var2, ...);
```

- Example

```
String b = "COMP"; int c = 2026;
System.out.printf("Hello %s%d", b, c);
//output Hello COMP2026
```

- A formatted string contains different **format specifier** (e.g. `%s`, `%d`, `%c`...).
- The variables are filled into the formatted string *in order*.

Common Formatted String

Format Specifier	Data Type
%d	Integer
%5d	Integer, output has at least 5 characters aligned to right.
%f	Decimal number, by default 6 decimal places.
%.5f	Decimal number with 5 decimal places
%7f	Decimal number, output has at least 7 characters aligned to right.
%s	String
%b	Boolean
%c	Character

```
int i = 4; float f = 1.445f; String s = "String";
System.out.printf("i: %d f: %.2f s : %s\n", i, f, s);
System.out.println("i: " + i + " f: " + f + " s: " + s);
```

```
i: 4 f: 1.45 s : String
i: 4 f: 1.445 s: String
```

Escape Character

- Apart from using unicode encoding (e.g. `\u0060`), we can also represent certain character using escape characters `\`.

Code	Output	Example	Example Output
`\"`	" Double quote	`"I \"know\" Java"`	I "know" Java
`\\`	\ Backslash	`"True\\False"`	True\False
`\'`	' Single quote	`"I don't know"` or `"I don\'t know"`	I don't know
`\n`	new line symbol	`"So... \nNew line!"	So... New line!
`\t`	tab symbol. Fill with spaces until tab stop	`"A\tBcd"`	A Bcd

Formatted String

```
int d = 123456;
System.out.printf("d: |%d|\n", d);
System.out.printf("9d |%9d|\n", d);
System.out.printf("6d |%6d|\n", d);
System.out.printf("5d |%5d|\n", d);
```

```
d: |123456|
9d |   123456|
6d |123456|
5d |123456|
```

```
int d = 123456; float f = 0.123456789f;
System.out.printf("f    : |%f|\n", f);
System.out.printf(".5f  : |%.5f|\n", f);
System.out.printf("9f   : |%9f|\n", f);
System.out.printf("9.6f : |%9.6f|\n", f);
System.out.printf("10.2f: |%10.2f|\n", f + d);
```

```
f    : |0.123457|
.5f  : |0.12346|
9f   : | 0.123457|
9.6f : | 0.123457|
10.2f: | 123456.13|
```

Input

- You are seeing your first *Object* in Java.
- To accept inputs, do:

1. Add `import java.util.Scanner` in the first line of your program
2. Create a scanner **once** in your program.
3. Use `scanner.next()` or relevant commands to take the inputs.

```
import java.util.Scanner; // add this line to the top of your program
...

Scanner scanner = new Scanner(System.in); //create scanner
String s = scanner.next(); // the text typed will be stored in s as a string.
```

Input

- Scanner reacts when `enter` key is pressed.
- Inputs are separated by `enter` or spaces .
- A line may contain more than one inputs.
- For string input, use `next()`.
- For an integer input, use `nextInt()`
- For double input, use `nextDouble()`
- For boolean input, use `nextBoolean()`
- For float input, use `nextFloat()`
- Program goes crazy if the input is unexpected.

```
String name = scanner.next();
int age = scanner.nextInt();
float weight = scanner.nextFloat();
System.out.printf(name + " %d %.2f", age, weight);
```

```
Kevin
30 220
Kevin 30 220.00
```

Summary

- Variable Scopes
- Basic CLI IO