Programming Java

STEP IT ACADEMY

Lesson 4

Loops

Contents

1. Loops

Java has the **while**, **do while**, and **for** operators for implementing looping repetitive algorithm.

The ***while*** Loop Operator.

Syntax:

```
while (expression) operation;
```

Expression in braces (loop condition) should be of boolean type. If the loop condition is true, then the "operation" will be executed, and the control will be passed back to the conditional check of expression. If the value is false, then the loop body execution will be terminated.

For example:

```
int i = 0;
while (i < 5) System.out.println(i++);
```

Outcome: 0, 1, 2, 3, 4, 5

Loop body is an operation of a block of code, which is limited by the curly braces and located immediately following the while operator.

For example:

```
int i = 0;
while (i < 5) { // beginning of the loop body
    i++;
    System.out.println(i);
} // end of the loop body
```

Outcome: 1, 2, 3, 4, 5

Note: *If the loop condition is initially false, then the loop body will never be fulfilled.*

For example:

```
int i = 0;
while (i > 5) {
    System.out.println(i++);
}
System.out.println(i);
```

Outcome: 0

If the members of the loop condition are not changed within the loop body, and the loop condition is true, then this loop will be executed infinitely.

Examples of an infinite loop:

```
while (true) {
    System.out.println("infinity");
}
```

do-while: a Loop with Postcondition

Syntax

```
do operation;
while(expression);
do {
operation1;
operation2;
}
while(expression);
```

The difference between the **do while** and the **while** loop is that the body of the **do while** loop is executed once until the check of the loop condition. Repeated execution of the loop body will depend on the value of an expression, and if it is true, then the program will return control to the beginning of the loop body.

For example:

```
int i = 0;
do
{
    i++;
    System.out.println(i);
} while (i > 10);
```

Outcome: 1

for: Execute a Loop n Times

Syntax:

```
        for (<initialization>;
    <condition>; <counter>)
            operation;
```

The "initialization" block can contain declaration and initialization of the variables that will be available in the loop body. This block is optional.

The "condition" block should contain an expression, the result of which should be a boolean value. This block is optional.

The "counter" block can contain any operations. This block is optional.

Example, in which all blocks are filled:

```
for (int i = 0; i < 10; i++)
{
    System.out.println(i);
}
```

Example without the counter variable declaration (it is required if you need access to the counter beyond the loop body):

```
int i;
for (i = 0; i < 10; i++)
{
    System.out.println(i);
}

i+=2;
System.out.println(i);
```

Example with no initialization block:

```
int i = 0;
for (; i < 10; i++)
{
    System.out.println(i);
}
```

Example with multiple counters:

```
for (int i = 1, j = 3; i < 5 & j > 0; i++, --j)
{
    System.out.println("i = " + i + " j = " + j);
}
```

Outcome: i = 1 j = 3,   i = 2 j = 2,   i = 3 j = 1

For-each

Java has no specific keyword for iteration through elements of an array or a collection, which other languages have. A special syntax of the for operator is used for implementing the for-each loop.

Syntax:
```
for (type element : collection) {
    operation;
}
```

Example for an array:

```
int [] numbers = new int[] { 3, 2, 1 };
for (int number : numbers)
{
    System.out.print(number + " ");
}
```

Example for a collection:

```
List<Integer> lists = new ArrayList<Integer>();
lists.add(1);
lists.add(3);
for (Integer value: lists)
{
    System.out.println(value);
}
```

The break and continue Operators

The **break** operator terminates loop execution and passes control to the end of the loop body.

For example:

```
int i = 0;
while (i < 10)
{
    System.out.println(i);
    if (i >= 2) break;
    i++;
}
```

Outcome: 0, 1

The **break** operator can be used in conjunction with a label, for example, for simultaneous termination of multiple nested loops. In this case, the control will be passed to the end of the block with the label specified after the **break** operator. Label is used for naming a block of code.

For example:

```
    two: for (int n = 0; n < 10; n++) {
       for (int j = 10; j > 0; j--) {
          System.out.print(j - n + " ");
             if (j + n == 5 && n > 0)
                break two;
      }
}
```

The **continue** operator terminates the loop body execution and passes control to the beginning of the loop. At that, current loop iteration is skipped.

For example:

```
int i = 0;
while (i < 10)
{
    i++;
    if (i % 2 == 0)
    continue;
    System.out.println(i);
}
```

Outcome: 1, 3, 5, 7, 9

The **continue** operator can also be used with a label.

For example:

```
one: for (i = 0; i < 10; i++) {
       for (int j = 0; j < 10; j++) {
         if (i == j)    continue one;
             System.out.printf("i=%d j=%d \n", i, j);
       }
     }
```

The Return Operator

The **return** operator terminates execution of a method, returning the control to the method call.

2. Main Provisions of Java Code Convention

Code Conventions for the Java Programming Language is an agreement on code formatting. Compliance with the provisions of the Convention makes your code more readable and understandable to other programmers. These provisions are recommendations based on personal experience of programmers when working with the code.

Class Names

The files with the source code should have the java extension.

The bytecode files should have the class extension.

Source File Organization

Files longer than 2000 lines are cumbersome and should be avoided. Files with a larger number of lines make it difficult to find the right block of code, increase the time of compilation of and loading a class to a java machine.

The file should be divided into sections that are separated by an empty string.

A single file can contain only one class with the public modifier (base class).

It is not desirable to have other (non-public) classes in a single file.

The order of sections in a class:

- JavaDoc;
- Class header;
- Static fields;
- Non-static fields;

- Constructors;
- Methods.

Indentations

Four spaces should be used as a unit of indentation (tabulation can be used).

Lines should not be longer than 80 characters.

Naming of Classes and Interfaces

Class names should be capitalized, consisting of letters of the Latin aphabet without spaces. Class name should correspond to an object of this class.

For example:

```
class  Cats // bad
class  Cat // good
```

If the class name contains more than one word, then each word should be capitalized (Camel notation). Class name should be corresponding.

For example:

```
class  TileIterator
```

Interface naming rules are the same as class naming rules. It is not desirable to use additional indications of belonging to an interface in the interface name.

```
interface IClonable // bad
interface Clonable // good
```

If the class name contains an abbreviation of more than two letters, then all the letters of this abbreviation, except the first one, should be lowercase.

For example:

```
class  FPSRenderer // bad
class  FpsRenderer // good
```

Variable Naming

Multiple variables should be declared in different lines each.

```
int a, b; // bad
int a; // good
int b; // good
```

Variables should be initialized at declaration where possible.

```
int a;
...
a = 255;  // bad
int a = 255; // good
```

Variables should be declared as close as possible to where they are used.

```
int a = 255; // bad
System.out.println("file")
a + = 127;
System.out.println("file")
int a = 255; // good
int c = a + 127;
```

Arrays

When declaring an array, brackets should be specified after the variable type, not the name.

```
int a [] = new int[3]; // bad
int[] a = new int[3]; // good

int [] a [] = new int[3]; // bad
int[][] a = new int[3][3]; // good
```

Constants

It is commonly accepted to name constants with all letters capitalized, and to use underscore as a separator.

Examples:

```
final int MAX_STEP = 3;// good
final String DELIMITER = ";";// good
```

Methods

Method names should consist of lowercase Latin letters. The first word in the method name should be a verb of an adverb. If the method name contains more than one word, then each word, except for the first one, should be capitalized (Camel notation).

Examples:

```
void startprocess() {} // bad
void Start() {} // bad
void start_process() {} // bad
void sProcess() {} // bad
void sp() {} // bad
void start() {} // good
void startProcess() {} // good
```

3. Working with the Eclipse integrated debugger

Eclipse is a free integrated development environment for modular cross-platform applications. It is developed and supported by the Eclipse Foundation.

Alternative Development Environments

Table 1. Rating of currently used Java-IDE

N	Name	% of votes	Evaluation
1	Eclipse	19.77	4.6
2	IntelliJ IDEA	19.06	4.7
3	NetBean	7.11	4.1
4	JBuilder	5.68	4.2
5	JDeveloper	2.13	4.0
6	JCreator	1.70	3.9

Table 2. Rating of previously used Java-IDE

N	Name	% of votes	Evaluation
1	Eclipse	21.47	3.0
2	IntelliJ IDEA	16.64	3.3
3	NetBean	14.22	2.9
4	JBuilder	11.66	3.5
5	JDeveloper	7.11	2.8
6	Visual J++	5.26	1.8
7	JCreator	4.26	2.3
8	VisualAge for Java	3.69	2.8
9	JCreator	3.41	2.0

Eclipse Installation

From the http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/indigor page.

Download Eclipse Indigo based on the type and size of the operating system.

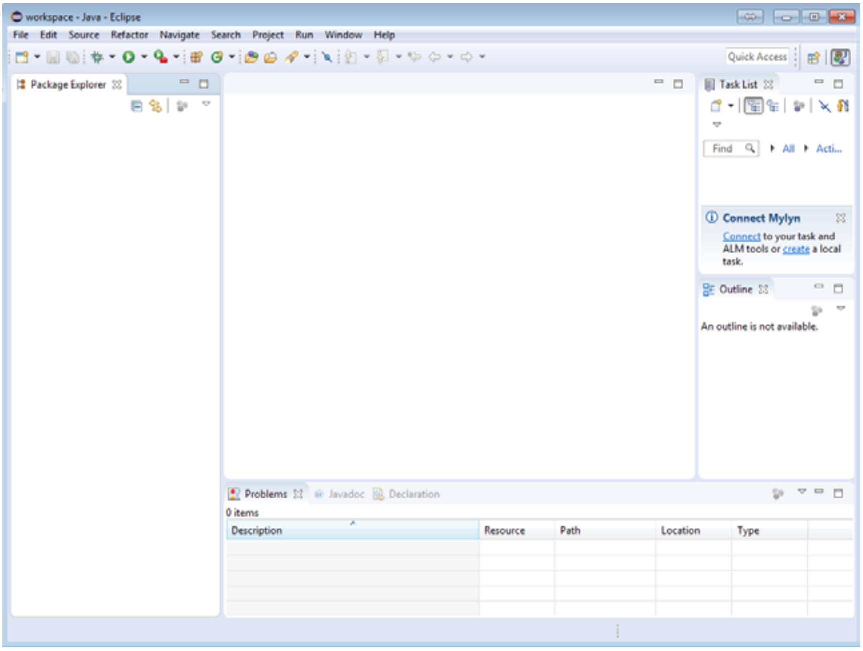
Download the archive. Unpack it in a convenient place: Eclipse is not installed through the installer; it is portable, unlike Netbeans.

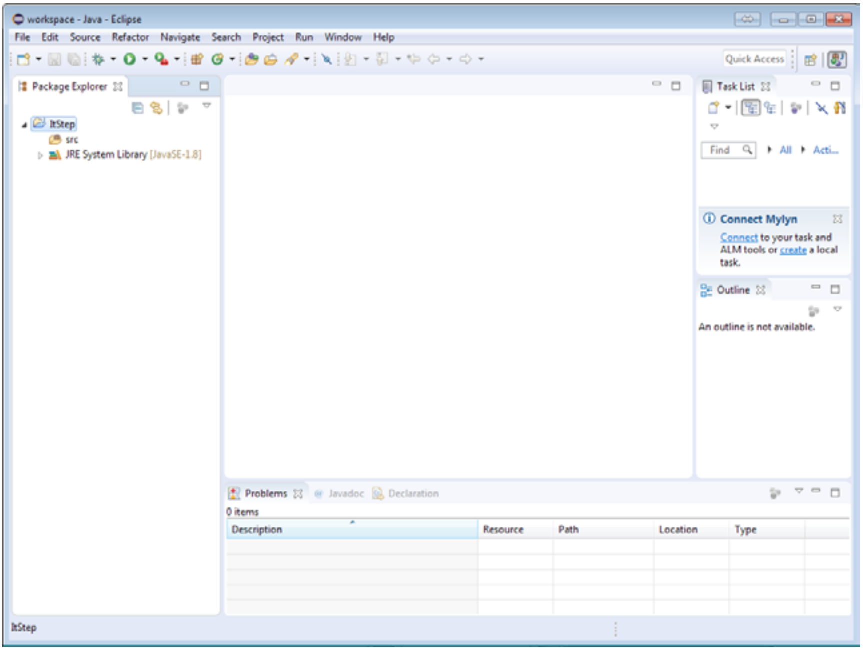
Creating a Project in Eclipse

Run the Eclipse SDK and make sure that the Java projection is open.

In the dialog box that opens, enter the name of the project. The project will be located in the directory installed as Workspace when configuring Eclipse. In the next dialog box, go to the "Libraries" tab. We will not change anything here, but remember that on this tab you can add additional Java libraries to the project, and at the moment the standard API, which is supplied with the JRE, is connected to the project.

Click "Finish".


workspace - Java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Package Explorer
Task List
Find
All
Acti...
Connect Mylyn
Connect to your task and ALM tools or create a local task.
Outline
An outline is not available.
Problems
Javadoc
Declaration
0 items
Description
Resource
Path
Location
Type


workspace - Java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Package Explorer
ItStep
src
JRE System Library [JavaSE-1.8]
Task List
Find
All
Acti...
Connect Mylyn
Connect to your task and ALM tools or create a local task.
Outline
An outline is not available.
Problems
Javadoc
Declaration
0 items
Description
Resource
Path
Location
Type
ItStep

An empty project is created, in order to continue to work, you need to add packages and classes. Create a package through the context menu. Right click on the "src" folder and select **"New" -> "Package"**. The same can be done by clicking the "New Java Package" button on the toolbar. In the dialog box that appears, enter the name of the package, it must be unique to avoid collisions of names, as a rule, developers invert the name of their domain, you too can do so. Create a class through the context menu. Right click on the package and select **"New" -> "Class"**. The same can be done by clicking the "New Java Class" button on the toolbar. In the dialog box of creating a class, enter its name, it can be anything, according to the rules for naming classes in Java. Note the "public static void main (String [] args)" option, thus we will tell the IDE to create for us the same function. Click "Finish". Now on the right, we see the project structure and our class file with the JAVA extension. The source code of the class is in the center, and the class browser, showing the structure of packages and classes in the form of a tree, is on the right. Edit the source code, enter the instruction for outputting the line to the console — System.out.println ("Your line");. Do not forget to use hints and autocompletion — start entering the code and press **Ctrl + Space**. Do not forget to end the line with ";".

Save the changes by pressing the **Ctrl + S** keys.

Running the Java project in Eclipse

To check the efficiency of our program, click the **"Run"** button on the toolbar or through the main menu. When you first start, you need to select either to run the program as a normal application, or as an applet. Select "Java Application".

Your first Java console application will be compiled and executed. In the "Console" view that opens, in the bottom panel of the main window of the IDE, we will see the output of the program, namely our line. Compiled class files with the CLASS extension can be found in the folder with the -> "bin" project.

Lesson 4

Loops