

# JAVA for Beginners



An introductory course for Advanced IT Students and those who would like to learn the Java programming language.

Riccardo  
Flask

# Introduction

## About JAVA

“Java refers to a number of computer software products and specifications from Sun Microsystems (the Java™ technology) that together provide a system for developing and deploying cross-platform applications. Java is used in a wide variety of computing platforms spanning from embedded devices and mobile phones on the low end to enterprise servers and super computers on the high end. Java is fairly ubiquitous in mobile phones, Web servers and enterprise applications, and somewhat less common in desktop applications, though users may have come across Java applets when browsing the Web.

Writing in the Java programming language is the primary way to produce code that will be deployed as Java bytecode, though there are compilers available for other languages such as JavaScript, Python and Ruby, and a native Java scripting language called Groovy. Java syntax borrows heavily from C and C++ but it eliminates certain low-level constructs such as pointers and has a very simple memory model where every object is allocated on the heap and all variables of object types are references. Memory management is handled through integrated automatic garbage collection performed by the Java Virtual Machine (JVM).”<sup>1</sup>

## OOP – Object Oriented Programming

OOP is a particular style of programming which involves a particular way of designing solutions to particular problems. Most modern programming languages, including Java, support this paradigm. When speaking about OOP one has to mention:

- Inheritance
- Modularity
- Polymorphism
- Encapsulation (binding code and its data)

However at this point it is too early to try to fully understand these concepts.

This guide is divided into two major sections, the first section is an introduction to the language and illustrates various examples of code while the second part goes into more detail.

## Variables and Data Types

### Variables

A variable is a place where the program stores data temporarily. As the name implies the value stored in such a location can be changed while a program is executing (compare with constant).

```
class Example2 {  
    public static void main(String args[]) {  
        int var1; // this declares a variable  
        int var2; // this declares another variable  
        var1 = 1024; // this assigns 1024 to var1  
        System.out.println("var1 contains " + var1);  
        var2 = var1 / 2;  
        System.out.print("var2 contains var1 / 2: ");  
        System.out.println(var2);  
    }  
}
```

Predicted Output:

```
var2 contains var1 / 2: 512
```

The above program uses two variables, var1 and var2. var1 is assigned a value directly while var2 is filled up with the result of dividing var1 by 2, i.e.  $\text{var2} = \text{var1}/2$ . The words *int* refer to a particular data type, i.e. integer (whole numbers).

### Test your skills – Example3

As we saw above, we used the '/' to work out the quotient of var1 by 2. Given that '+' would perform addition, '-' subtraction and '\*' multiplication, write out a program which performs all the named operations by using two integer values which are hard coded into the program.

Hints:

- You need only two variables of type integer
- Make one variable larger and divisible by the other
- You can perform the required calculations directly in the print statements, remember to enclose the operation within brackets, e.g. (var1-var2)

## Mathematical Operators

As we saw in the preceding example there are particular symbols used to represent operators when performing calculations:

Operator	Description	Example – given a is 15 and b is 6
+	Addition	a + b, would return 21
-	Subtraction	a - b, would return 9
*	Multiplication	a * b, would return 90
/	Division	a / b, would return 2
%	Modulus	a % b, would return 3 (the remainder)

```
class Example4 {
    public static void main(String args[]) {
        int iresult, irem;
        double dresult, drem;
        iresult = 10 / 3;
        irem = 10 % 3;
        dresult = 10.0 / 3.0;
        drem = 10.0 % 3.0;
        System.out.println("Result and remainder of 10 / 3: " +
            iresult + " " + irem);
        System.out.println("Result and remainder of 10.0 / 3.0: "
            + dresult + " " + drem);
    }
}
```

Predicted Output:

```
Result and Remainder of 10/3: 3 1
Result and Remainder of 10.0/3.0: 3.3333333333333335 1
```

The difference in range is due to the data type since 'double' is a double precision 64-bit floating point value.

## Logical Operators

These operators are used to evaluate an expression and depending on the operator used, a particular output is obtained. In this case the operands must be Boolean data types and the result is also Boolean. The following table shows the available logical operators:

Trying to understand the above program is a bit difficult, however the program highlights the main difference in operation between a normal AND (&) and the short-circuit version (&&). In a normal AND operation, both sides of the expression are evaluated, e.g.

if(d != 0 & (n % d) == 0) – this returns an error as first d is compared to 0 to check inequality and then the operation (n%d) is computed yielding an error! (divide by zero error)

The short circuit version is smarter since if the left hand side of the expression is false, this means that the output has to be false whatever there is on the right hand side of the expression, therefore:

if(d != 0 && (n % d) == 0) – this does not return an error as the (n%d) is not computed since d is equal to 0, and so the operation (d!=0) returns false, causing the output to be false. Same applies for the short circuit version of the OR.

## Character Escape Codes

The following codes are used to represent codes or characters which cannot be directly accessible through a keyboard:

Code	Description
\n	New Line
\t	Tab
\b	Backspace
\r	Carriage Return
\\	Backslash
\'	Single Quotation Mark
\"	Double Quotation Mark
\*	Octal - * represents a number or Hex digit
\x*	Hex
\u*	Unicode, e.g. \u2122 = ™ (trademark symbol)

```
class Example6 {
    public static void main(String args[]) {
        System.out.println("First line\nSecond line");
        System.out.println("A\tB\tC");
        System.out.println("D\tE\tF") ;
    }
}
```

Predicted Output:

```
First Line
Second Line
A      B      C
D      E      F
```

## ☀️ Test your skills – Example7

Make a program which creates a sort of truth table to show the behaviour of all the logical operators mentioned. Hints:

- You need two Boolean type variables which you will initially set both to false
- Use character escape codes to tabulate the results

The following program can be used as a guide:

```
class LogicTable {
    public static void main(String args[]) {
        boolean p, q;

        System.out.println("P\tQ\tPANDQ\tPORQ\tPXORQ\tNOTP");

        p = true; q = true;

        System.out.print(p + "\t" + q + "\t");
        System.out.print((p&q) + "\t" + (p|q) + "\t");
        System.out.println((p^q) + "\t" + (!p));

        p = true; q = false;

        System.out.print(p + "\t" + q + "\t");
        System.out.print((p&q) + "\t" + (p|q) + "\t");
        System.out.println((p^q) + "\t" + (!p));

        p = false; q = true;

        System.out.print(p + "\t" + q + "\t");
        System.out.print((p&q) + "\t" + (p|q) + "\t");
        System.out.println((p^q) + "\t" + (!p));

        p = false; q = false;

        System.out.print(p + "\t" + q + "\t");
        System.out.print((p&q) + "\t" + (p|q) + "\t");
        System.out.println((p^q) + "\t" + (!p));
    }
}
```

## Introducing Control Statements

These statements will be dealt with in more detail further on in this booklet. For now we will learn about the **if** and the **for loop**.

```
class Example11 {
    public static void main(String args[]) {
        int a,b,c;
        a = 2;
        b = 3;
        c = a - b;
        if (c >= 0) System.out.println("c is a positive number");
        if (c < 0) System.out.println("c is a negative number");
        System.out.println();
        c = b - a;
        if (c >= 0) System.out.println("c is a positive number");
        if (c < 0) System.out.println("c is a negative number");
    }
}
```

Predicted output:

```
c is a negative number
c is a positive number
```

The 'if' statement evaluates a condition and if the result is true, then the following statement/s are executed, else they are just skipped (refer to program output). The line `System.out.println()` simply inserts a blank line. Conditions use the following comparison operators:

Operator	Description
<	Smaller than
>	Greater than
<=	Smaller or equal to, (a<=3) : if a is 2 or 3, then result of comparison is TRUE
>=	Greater or equal to, (a>=3) : if a is 3 or 4, then result of comparison is TRUE
==	Equal to
!=	Not equal

The for loop is an example of an iterative code, i.e. this statement will cause the program to repeat a particular set of code for a particular number of times. In the following example we will be using a counter which starts at 0 and ends when it is smaller than 5, i.e. 4. Therefore the code following the for loop will iterate for 5 times.

```
class Example12 {
    public static void main(String args[]) {
        int count;
        for(count = 0; count < 5; count = count+1)
            System.out.println("This is count: " + count);
        System.out.println("Done!");
    }
}
```

Predicted Output:

```
This is count: 0
This is count: 1
This is count: 2
This is count: 3
This is count: 4
Done!
```

Instead of `count = count+1`, this increments the counter, we can use `count++`

The following table shows all the available shortcut operators:

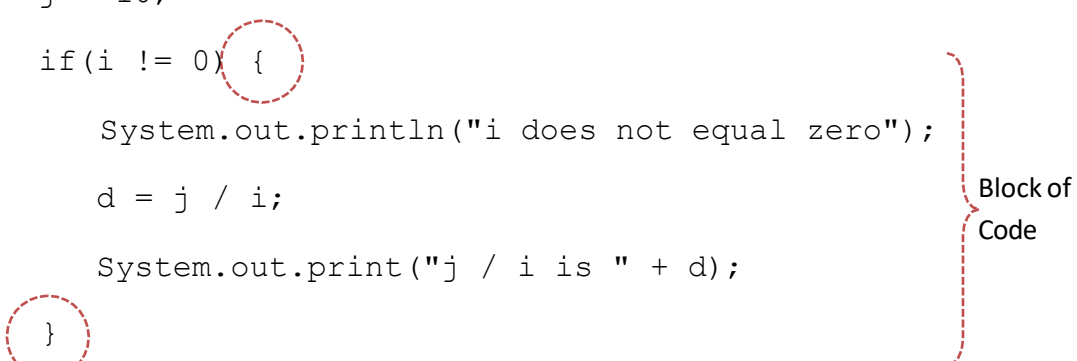
Operator	Description	Example	Description
++	Increment	a++	a = a + 1 (adds one from a)
--	Decrement	a--	a = a - 1 (subtract one from a)
+=	Add and assign	a+=2	a = a + 2
-=	Subtract and assign	a-=2	a = a - 2
*=	Multiply and assign	a*=3	a = a * 3
/=	Divide and assign	a/=4	a = a / 4
%=	Modulus and assign	a%=5	a = a mod 5



## Blocks of Code

Whenever we write an IF statement or a loop, if there is more than one statement of code which has to be executed, this has to be enclosed in braces, i.e. '....-'

```
class Example13 {
    public static void main(String args[]) {
        double i, j, d;
        i = 5;
        j = 10;
        if(i != 0) {
            System.out.println("i does not equal zero");
            d = j / i;
            System.out.print("j / i is " + d);
        }
        System.out.println();
    }
}
```



Predicted Output:

```
i does not equal to zero
j/i is 2
```

## ☼ Test your skills – Example14

Write a program which can be used to display a conversion table, e.g. Euros to Malta Liri, or Metres to Kilometres.

Hints:

- One variable is required
- You need a loop

The Euro Converter has been provided for you for guidance. Note loop starts at 1 and finishes at 100 (<101). In this case since the conversion rate does not change we did not use a variable, but assigned it directly in the print statement.

```
class EuroConv {
```

useful method is the `Math.random()` which would return a random number ranging between 0.0 and 1.0.

### Scope and Lifetime of Variables

The following simple programs, illustrate how to avoid programming errors by taking care where to initialize variables depending on the scope.

```
class Example16 {
    public static void main(String args[]) {
        int x; // known to all code within main
        x = 10;

        if(x == 10) { // start new scope
            int y = 20; // known only to this block
            // x and y both known here.
            System.out.println("x and y: " + x + " " + y);
            x = y * 2;
        }

        // y = 100; // Error! y not known here
        // x is still known here.
        System.out.println("x is " + x);
    }
}
```

Predicted Output:

```
x and y: 10 20
x is 40
```

If we had to remove the comment marks from the line, `y = 100`; we would get an error during compilation as `y` is not known since it only exists within the block of code following the 'if' statement.

The next program shows that `y` is initialized each time the code belonging to the looping sequence is executed; therefore `y` is reset to -1 each time and then set to 100. This operation is repeated for three (3) times.

## Console Input

Most students at this point would be wondering how to enter data while a program is executing. This would definitely make programs more interesting as it adds an element of interactivity at run-time. This is not that straight forward in Java, since Java was not designed to handle console input. The following are the three most commonly used methods to cater for input:

## Using the Keyboard Class

One can create a class, which would contain methods to cater for input of the various data types. Another option is to search the internet for the Keyboard Class. This class is easily found as it is used in beginners Java courses. This class is usually found in compiled version, i.e. keyboard.class. This file has to be put in the project folder or else placed directly in the Java JDK. The following is the source code for the Keyboard class just in case it is not available online!

```
import java.io.*;

import java.util.*;

public class Keyboard {

    /******* Error Handling Section

        private static boolean printErrors = true;

        private static int errorCount = 0;

        // Returns the current error count.

        public static int getErrorCount(){

            return errorCount;

        }

        // Resets the current error count to zero.

        public static void resetErrorCount (int count){

            errorCount = 0;

        }

        // Returns a boolean indicating whether input errors are

        // currently printed to standard output.

        public static boolean getPrintErrors(){

            return printErrors;

        }

    }
```

## Using Swing Components

This is probably the most exciting version, since the Swing package offers a graphical user interface (GUI) which allows the user to perform input into a program via the mouse, keyboard and other input devices.

```
import javax.swing.*; // * means „all“

public class SwingInput {

    public static void main(String[] args) {

        String temp; // Temporary storage for input.

        temp = JOptionPane.showInputDialog(null, "First
number");

        int a = Integer.parseInt(temp); // String to int

        temp = JOptionPane.showInputDialog(null, "Second
number");

        int b = Integer.parseInt(temp);

        temp = JOptionPane.showInputDialog(null, "Third
number");

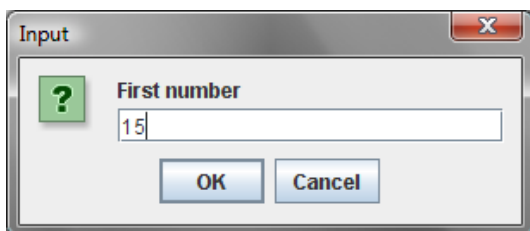
        int c = Integer.parseInt(temp);

        JOptionPane.showMessageDialog(null, "Average is " +
(a+b+c)/3);

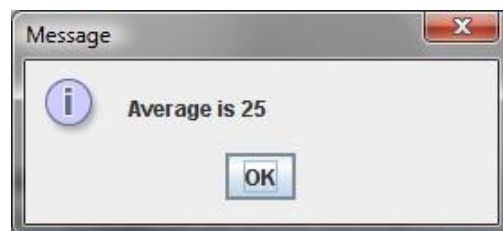
    }

}
```

One has to note that the input is stored as a string, **temp**, and then parsed to integer using the method **parseInt()**. This time the method accepts a parameter, temp, and returns an integer. When the above program is executed, a dialog box will appear on screen with a field to accept input from user via keyboard (JOptionPane.showInputDialog). This is repeated three times and finally the result is again displayed in a dialog box (JOptionPane.showMessageDialog).



JOptionPane.showInputDialog



JOptionPane.showMessageDialog

## Part 2 - Advanced Java program

### Control Statements - The if Statement

```
if(condition) statement;  
else statement;
```

Note:

- **else** clause is optional
- targets of both the **if** and **else** can be blocks of statements.

The general form of the **if**, using blocks of statements, is:

```
if(condition)  
    {  
        statement sequence  
    }  
else  
    {  
        statement sequence  
    }
```

If the conditional expression is true, the target of the **if** will be executed; otherwise, if it exists, the target of the **else** will be executed. At no time will both of them be executed. The conditional expression controlling the **if** must produce a **boolean** result.

## Guessing Game (Guess.java)

The program asks the player for a letter between A and Z. If the player presses the correct letter on the keyboard, the program responds by printing the message **\*\*Right\*\***.

```
// Guess the letter game.

class Guess {

    public static void main(String args[])
        throws java.io.IOException {

        char ch, answer = 'K';

        System.out.println("I'm thinking of a letter between A
and Z.");

        System.out.print("Can you guess it: ");

        ch = (char) System.in.read(); // read a char from the
        keyboard

        if(ch == answer) System.out.println("** Right **");

    }

}
```

Extending the above program to use the else statement:

```
// Guess the letter game, 2nd version.

class Guess2 {

    public static void main(String args[])
        throws java.io.IOException {

        char ch, answer = 'K';

        System.out.println("I'm thinking of a letter between A
and Z.");

        System.out.print("Can you guess it: ");

        ch = (char) System.in.read(); // get a char

        if(ch == answer) System.out.println("** Right **");
        else System.out.println("...Sorry, you're wrong.");

    }

}
```

## Nested if

The main thing to remember about nested **ifs** in Java is that an **else** statement always refers to the nearest **if** statement that is within the same block as the **else** and not already associated with an **else**. Here is an example:

```
if(i == 10) {
    if(j < 20) a = b;
    if(k > 100) c = d;
    else a = c; // this else refers to if(k > 100)
}
else a = d; // this else refers to if(i == 10)
```

## Guessing Game v.3

```
// Guess the letter game, 3rd version.
class Guess3 {
    public static void main(String args[])
        throws java.io.IOException {
        char ch, answer = 'K';
        System.out.println("I'm thinking of a letter between A
and Z.");
        System.out.print("Can you guess it: ");
        ch = (char) System.in.read(); // get a char
        if(ch == answer) System.out.println("*** Right ***");
        else {
            System.out.print("...Sorry, you're ");
            // a nested if
            if(ch < answer) System.out.println("too low");
            else System.out.println("too high");
        }
    }
}
```

A sample run is shown here:

I'm thinking of a letter between A and Z.

Can you guess it: Z

...Sorry, you're too high

### if-else-if Ladder

```
if(condition)
    statement;
else if(condition)
    statement;
else if(condition)
    statement;
...
else
    statement;
```

The conditional expressions are evaluated from the top downward. As soon as a true condition is found, the statement associated with it is executed, and the rest of the ladder is bypassed. If none of the conditions is true, the final **else** statement will be executed. The final **else** often acts as a default condition; that is, if all other conditional tests fail, the last **else** statement is performed. If there is no final **else** and all other conditions are false, no action will take place.

```
// Demonstrate an if-else-if ladder.

class Ladder {
public static void main(String args[]) {
    int x;

    for(x=0; x<6; x++) {
        if(x==1)
            System.out.println("x is one");
        else if(x==2)
            System.out.println("x is two");
```



## Multiple Loop Control Variable

Using more than one variable in the same loop is possible:

```
// Use commas in a for statement.
```

```
class Comma {
    public static void main(String args[]) {
        int i, j;
        for(i=0, j=10; i < j; i++, j--)
            System.out.println("i and j: " + i + " " + j);
    }
}
```

*'i' and 'j' are the  
two variables used  
in the same loop*

Expected output:

```
i and j: 0 10
i and j: 1 9
i and j: 2 8
i and j: 3 7
i and j: 4 6
```

## Terminating a loop via user intervention

Let us write a program which involves a loop and this is stopped when the user types 's' on the keyboard:

```
// Loop until an S is typed.
```

```
class ForTest {
    public static void main(String args[])
        throws java.io.IOException {
        int i;
        System.out.println("Press S to stop.");
        for(i = 0; (char) System.in.read() != 'S'; i++)
            System.out.println("Pass #" + i);
    }
}
```

## Interesting For Loop Variations

It is possible to leave out parts of the loop declaration:

// Example 1 - Parts of the for can be empty.

```
class Empty {  
    public static void main(String args[]) {  
        int i;  
        for(i = 0; i < 10; ) {  
            System.out.println("Pass #" + i);  
            i++; // increment loop control var  
        }  
    }  
}
```

// Example 2 - Parts of the for can be empty.

```
class Empty2 {  
    public static void main(String args[]) {  
        int i;  
        i = 0; // move initialization out of loop  
        for(; i < 10; ) {  
            System.out.println("Pass #" + i);  
            i++; // increment loop control var  
        }  
    }  
}
```

Initialising the loop out of the 'for' statement is only required when the value needs to be a result of another complex process which cannot be written inside the declaration.

## Infinite Loops

Sometimes one needs to create an infinite loop, i.e. a loop which never ends! (However it can be stopped using the break statement). An example of an infinite loop declaration is as follows:

```
for(;;)
{
    // ... statements
}
```

*N.B. Using **break** to terminate an infinite loop will be discussed later on in the course.*

## No 'Body' Loops

Loops can be declared without a body. This can be useful in particular situations, consider the following example:

```
// Loop without body.
class Empty3 {
    public static void main(String args[]) {
        int i;
        int sum = 0;
        // sum the numbers through 5
        for(i = 1; i <= 5; sum += i++) ;
        System.out.println("Sum is " + sum);
    }
}
```

*Two operations are carried  
on,  $sum = sum + i$  and  
 $i = i + 1$*

Predicted Output:

Sum is 15

## Declaring variables inside the loop

Variables can be declared inside the loop itself but one must remember that in such case the variable exists only inside the loop!

## The While Loop

`while (condition) statement; //or more than one statement`

The condition could be any valid Boolean expression. The loop will function only if the condition is true. If false it will move on to the next line of code.

```
// Demonstrate the while loop.

class WhileDemo {

    public static void main(String args[]) {

        char ch;

        // print the alphabet using a while loop

        ch = 'a';

        while(ch <= 'z') {

            System.out.print(ch);

            ch++;

        }

    }

}
```

The above program will output the alphabet. As can be seen in the code the while loop will result false when the character is greater than 'z'. The condition is tested at the beginning of the program.

```
// Compute integer powers of 2.

class Power {

    public static void main(String args[]) {

        int e;

        int result;

        for(int i=0; i < 10; i++) {

            result = 1;

            e = i;

            while(e > 0) {

                result *= 2;

                e--;

            }

        }

    }

}
```

## ☼ Mini-Project 2– Java Help System (Help2.java)

We are going to work on our previous project. Copy all the code and add the following code:

```
do {
    System.out.println("Help on:");
    System.out.println(" 1. if");
    System.out.println(" 2. switch");
    System.out.println(" 3. for");
    System.out.println(" 4. while");
    System.out.println(" 5. do-while\n");
    System.out.print("Choose one: ");
    do {
        choice = (char) System.in.read();
    } while(choice == '\n' | choice == '\r');
} while( choice < '1' | choice > '5');
```

Now extend the switch as follows:

```
switch(choice) {
    case '1':
        System.out.println("The if:\n");
        System.out.println("if(condition) statement;");
        System.out.println("else statement;");
        break;
    case '2':
        System.out.println("The switch:\n");
        System.out.println("switch(expression) {");
        System.out.println(" case constant:");
        System.out.println(" statement sequence");
        System.out.println(" break;");
        System.out.println(" // ...");
        System.out.println("}");
```

## Using Break to Terminate a Loop

One can use the 'break' command to terminate voluntarily a loop. Execution will continue from the next line following the loop statements,

e.g. 1 Automatic termination (hard-coded)

```
class BreakDemo {
    public static void main(String args[]) {
        int num;
        num = 100;
        for(int i=0; i < num; i++) {
            if(i*i >= num) break; // terminate loop if i*i >= 100
            System.out.print(i + " ");
        }
        System.out.println("Loop complete.");
    }
}
```

*When i = 10, i\*i = 100. Therefore the 'if' condition is satisfied and the loop terminates before i = 100*

Expected Output:

0 1 2 3 4 5 6 7 8 9 Loop complete.

e.g. 2 Termination via user intervention

```
class Break2 {
    public static void main(String args[])
        throws java.io.IOException {
        char ch;
        for( ; ; ) {
            ch = (char) System.in.read(); // get a char
            if(ch == 'q') break;
        }
        System.out.println("You pressed q!");
    }
}
```

In the above program there is an infinite loop, `for( ; ; )`. This means that the program will never terminate unless the user presses a particular letter on the keyboard, in this case 'q'.

If we have nested loops, i.e. a loop within a loop, the 'break' will terminate the inner loop. It is not recommended to use many 'break' statement in nested loops as it could lead to bad programs. However there could be more than one 'break' statement in a loop. If there is a switch statement in a loop, the 'break' statement will affect the switch statement **only**. The following code shows an example of nested loops using the 'break' to terminate the inner loop;

### ☼ Mini-Project 3– Java Help System (Help3.java)

This project puts the finishing touches on the Java help system that was created in the previous projects. This version adds the syntax for 'break' and 'continue'. It also allows the user to request the syntax for more than one statement. It does this by adding an outer loop that runs until the user enters 'q' as a menu selection.

1. Copy all the code from Help2.java into a new file, Help3.java
2. Create an outer loop which covers the whole code. This loop should be declared as infinite but terminate when the user presses 'q' (use the break)
3. Your menu should look like this:

```
do {
    System.out.println("Help on:");
    System.out.println(" 1. if");
    System.out.println(" 2. switch");
    System.out.println(" 3. for");
    System.out.println(" 4. while");
    System.out.println(" 5. do-while");
    System.out.println(" 6. break");
    System.out.println(" 7. continue\n");
    System.out.print("Choose one (q to quit): ");
    do {
        choice = (char) System.in.read();
    } while(choice == '\n' | choice == '\r');
} while( choice < '1' | choice > '7' & choice != 'q');
```

4. Adjust the switch statement to include the 'break' and 'continue' features.

### Complete Listing

```
class Help3 {

    public static void main(String args[])

        throws java.io.IOException {

        char choice;

        for(;;) {

            do {

                System.out.println("Help on:");

                System.out.println(" 1. if");

                System.out.println(" 2. switch");

                System.out.println(" 3. for");

                System.out.println(" 4. while");
```

## Nested Loops

A nested loop is a loop within a loop. The previous examples already included such loops. Another example to consider is the following:

```
class FindFac {  
    public static void main(String args[]) {  
        for(int i=2; i <= 100; i++) {  
            System.out.print("Factors of " + i + ": ");  
            for(int j = 2; j < i; j++)  
                if((i%j) == 0) System.out.print(j + " ");  
            System.out.println();  
        }  
    }  
}
```

The above code prints the factors of each number starting from 2 up to 100. Part of the output is as follows:

```
Factors of 2:  
Factors of 3:  
Factors of 4: 2  
Factors of 5:  
Factors of 6: 2 3  
Factors of 7:  
Factors of 8: 2 4  
Factors of 9: 3  
Factors of 10: 2 5  
...
```

Can you think of a way to make the above code more efficient? (Reduce the number of iterations in the inner loop).



## Class Fundamentals

### Definition

A class is a sort of template which has attributes and methods. An object is an instance of a class, e.g. Riccardo is an object of type Person. A class is defined as follows:

```
class classname {
    // declare instance variables
    type var1;
    type var2;
    // ...
    type varN;
    // declare methods
    type method1(parameters) {
        // body of method
    }
    type method2(parameters) {
        // body of method
    }
    // ...
    type methodN(parameters) {
        // body of method
    }
}
```

The classes we have used so far had only one method, **main()**, however not all classes specify a main method. The main method is found in the main class of a program (starting point of program).

### The Vehicle Class

The following is a class named 'Vehicle' having three attributes, 'passengers' – the number of passengers the vehicle can carry, 'fuelcap' – the fuel capacity of the vehicle and 'mpg' – the fuel consumption of the vehicle (miles per gallon).

```
class Vehicle {
    int passengers; //number of passengers
    int fuelcap; //fuel capacity in gallons
    int mpg; //fuel consumption
}
```

Please note that up to this point there is no OBJECT. By typing the above code a new data type is created which takes three parameters. To create an instance of the Vehicle class we use the following statement:

```
Vehicle minivan = new Vehicle ();
```

To set the values of the parameters we use the following syntax:

```
minivan.fuelcap = 16; //sets value of fuel capacity to 16
```