

Forum:[Hack Forums](#)

Thread: [\[Updated\] Java Tutorial #2](#)

Psycho - April 16, 2009, 10:29 pm

Java Tutorial #2

Outline

1. Order of Operation
 - ..1.1 Operator Precedence
2. Nested Flow Control Statements
 - ..2.1 Nested If Statements
 - ..2.2 Nested While/For Loops
3. Using Classes
 - ..3.1 Instantiating a Class
 - ..3.2 String Class
 - ..3.3 Math Class
 - ..3.4 Arrays
4. Methods
 - ..4.1 Using Methods
 - ..4.2 Creating Methods
5. File Input/Output
 - ..5.1 File Input
 - ..5.2 File Output
6. End

1. Order of Operation

It is important to understand operator precedence.

Go [here](#) for a

complete list of

operator precedence.

2. Nested Flow Control Statements

It is very useful to know about nested flow control statements.

2.1 Nested If Statements

An example of a nested If statement:

Code if (a)

```
    if (b)
        do();
```

This nested If statement first tests whether or not a

is true.

If so, it proceeds and then tests whether or not b is

true.

If so, it proceeds to call the do() method.

This can't be written as if (a && b)?

Yes, it could be written that way; however, there may be special cases where someone will find

it practical to use

a nested if statement.

Such as:

Code if (a)

```
{
    c++; // we want to increment C regardless of /b/'s value.
    if (b)
        die();
}
```

2.2 Nested While/For Loops

An example of a nested For loop inside a While loop:

Code boolean input = true;

while (input)

```
    for (int i = 0; i < 3; i++)
```

```
        System.out.println(3-i);
```

This code block will print out 3 to 1, each number on a separate line.

You may have noticed that this will continuously print out:

Code 3

2

1

Every pass through the while loop, it makes 3 passes through the for loop.

Since input is always true, while's condition will always return true; thus, we have an infinite

loop.

3. Using Classes

Using classes is another very important part of Java programming.

3.1 Instantiating a Class

Most classes need to be instantiated before being used.

Instantiate?

Instantiating is the act of creating an instance of a class; an object.

The syntax for instantiation is:

Code `ClassName identifier = new ClassName;`

However, if the class requires any parameters to be passed, it could look this way:

Code `ClassName identifier = new ClassName(1, 2, 3);`

The following is also valid:

Code `ClassName identifier;`

`identifier = new ClassName;`

The first line only creates a reference variable, a variable that points to an object.

Since we have not instantiated an object, there is nothing to point to; thus, we have a

null pointer

.

You may have guessed that a null pointer points to nothing.

This would be a good opening to explain Java's garbage collection.

When a reference variable stops pointing to an object, and no other reference variables are

pointing to it, that

portion of the memory is automatically freed up and available.

This is called garbage collection.

3.2 String Class

Java uses a String class for string-type variables; however, it is quite different from normal classes.

While the new operator can be used to instantiate a String object, it is not

required.

(Only true with the String class. No other classes can be instantiated without the

new

operator.)

So this makes creating String objects much like creating regular variables of primitive data

type.

A String object can also be represented as an array of characters.

Code `String name = "name";`

`String lastName;`

`lastName = "last name";`

3.3 Math Class

The Math class is also unique. The Math class is static; therefore, its methods can be invoked directly through dot notation, without an object being instantiated.

Example of square root function:

```
Code System.out.println(Math.sqrt(16)); // displays 4
System.out.println(Math.sqrt(4)); // displays 2
```

3.4 Arrays

Arrays can be difficult to grasp for a newborn programmer. I will try to explain them the best I can.

An array holds multiple values under one identifier. To reference these values, you use an index number (or

subscript) between two brackets, [].

Example:

```
Code int[] grades = new int[2];
grades[0] = 10;
grades[1] = 50;
grades[2] = 100;
```

The above example instantiates an array of int type, and of size 2.

Array indexes start at 0, so it can technically hold 3 values. In index 0, 1, 2 and 3.

You can also instantiate an array like so:

```
Code int grades[] = new int[2];
```

I will be instantiating arrays this way from now on.

Arrays are also easier to manage than multiple variables.

Let's say you had 100 values you wanted to add 1 to.

Which of the following would you prefer?

```
Code var1++;
var2++;
var3++;
...
var99++;
var100++;
or
```

```
Code for (int i = 0; i < 100; i++)
    var[i]++;
```

I think 100 lines vs 2 lines speaks for itself here. :)

4. Methods

Now, on to methods. Isn't this fun? :D

Methods are used to break up your driver's code and used for segments of code

that would

usually be repeated.

4.1 Using Methods

Methods are fairly simple to use. They may return any type of data, or they may return nothing. A method with the return type void will not return anything.

To call a method, you simply use the name of the method and if it has any parameters, include them.

Example of a void method:

Code `displayNumbers();`

This method obviously displays some numbers. It requires no parameters, and has nothing to return.

Example of an int method:

Code `int x;`

`x = add(1, 3); // x = 4`

This method returns an integer, and in this case, it returns 4, the sum of 1 and 3.

If it's not obvious already, to store the value a method returns, you need to store it in a

variable of the same data

type.

4.2 Creating Methods

Methods consist of a method header and a body of code.

In the method header, we have a visibility modifier, a return

type, a method

name and a list of parameters

`public int add(int a,`

`int b)`

The above is the header for a method that adds two numbers together and returns the sum. `public` being the visibility modifier, which decides who can access

this method.

More on visibility modifiers will be discussed in Encapsulation.

int is the data type of the value we return when the method is finished.
"add" is the name of our method, the name we will use to call it.
And then we have the parameter list, which defines variables that are local to the method.
More on variable scopes will be discussed in Scopes.

The full method looks like:

```
Code public int add(int a, int b)
{
    return a + b; // return sum of a and b
}
```

The return statement returns the value given to it and ends the method

regardless of its position.

H3LL - April 17, 2009, 2:48 am

You know what... i came in and looked at the screen... immediately i pressed the back button... then come to think of it, this guy has put some effort... so i came back in...

Thanks for the tutorial, hope it helps those beginners

KING-T - April 17, 2009, 3:38 am

Lol at the post above

and nice tutorial i hate java but still this is very good.

scaryblackguy - April 17, 2009, 2:23 pm

nice tut again ^_^

(tooshort)

Psycho - April 17, 2009, 5:24 pm

Thanks and sorry for it being incomplete.

Things offline aren't going so well, and busier than usual.

I'll complete it in time.

Itachi Uchiha - April 17, 2009, 5:26 pm

Omg dude +rep. First TUT I seen in a while that someone made themselves.

hurmony - April 17, 2009, 9:04 pm

This is a nice tutorial, very easy to understand but i despise java.

Psycho - April 27, 2009, 7:49 pm

Tutorial updated and completed.

Sorry for the long delay. I finished the rest of the tutorial.

It was too large to put into the same post, so I made a new post.

5. File Input/Output

Ah, file input and output. Usually a dreaded topic for any beginner. No worries, file input and output is quite simple when you understand it.

5.1 File Input

We will start with file input. In Java, inputting a file required the use of two classes, both can be found in the java.io package. These two classes are `FileReader` and `BufferedReader`.

How do we use them? Well, I'm getting to that.

Let us assume you have a program that needs to print every line of a text document into the console.

First, your code would look something like this:

```
Code import java.util.Scanner;
public class FileIO
{
    public static void main(String [] args)
    {
        Scanner keyboard = new Scanner(System.in);

        // Input file name
        System.out.print("Enter a file name: ");
        String fileName = keyboard.nextLine();

        // read from the file
        // read
        // display
    }
}
```

Obviously, we have not implemented the part where we read from the file and display the contents.

Before dumping all the code on you, let me explain the two classes we will be using. The `FileReader` class requires one parameter to be instantiated. That is, the file

name/path.

As you may have guessed, the syntax for instantiating a `FileReader` object would look like:

```
FileReader fr = new FileReader ("document.txt");
```

As for the `BufferedReader` class, this one accepts one parameter, too. It wants an object of type `FileReader`.

Looks like we are in luck, we just instantiated one of those. ;)

The syntax for instantiating a `BufferedReader` object would look like:

```
BufferedReader br = new BufferedReader (fr);
```

We pass in a `FileReader` object, and the `BufferedReader` class is happy. :)

How do we make use of these? That is exactly what I am about to teach you.

The `BufferedReader` class has a method that allows us to read a line from a file, close the file, and all of that good stuff.

How do we invoke this method? Simple! Using dot notation, you can invoke the `readLine()` method.

Example: `br.readLine();`

Yes, it is that simple to read a line from our file. ^_~

--Wait, we just dropped the line we read on the floor. Since `readLine()` returns a `String` and we never stored it, we simply read the line and threw the `String` away. :(

Let's fix that: `String line = br.readLine();`

What if there are no more lines left to read? Then the `readLine()` method returns `null`

.
How do we handle that? I am getting to that right now.

Now, because we want to read every single line and display them, we'll need a loop.
And because we have no idea how large the file is, or how many lines we will be reading, we will be using the `While` loop.

Our plan is to develop a loop that reads every single line (except the `null`, non-existing ones).
So, a basic outline:

```
Read our first line
WHILE (Our line is not null)
{
    Display the line
    Read another line
}
```

The structure of that loop will successfully read every line in the file.

Let's implement this!

```
Code import java.util.Scanner;
import java.io.*; // ***Remember to include your IO package!***
public class FileIO
{
    public static void main(String [] args)
    {
        Scanner keyboard = new Scanner(System.in);

        // Input file name
        System.out.print("Enter a file name: ");
        String fileName = keyboard.nextLine();
```

```

    FileReader fr = new FileReader(fileName);
    BufferedReader br = new BufferedReader(fr);

    String line = br.readLine();

    while(line != null)
    {
        // display
        System.out.println(line);

        // get next line
        line = br.readLine();
    }

    br.close();
}

```

Now let's create a file to read from. Name it whatever you'd like. I'm naming mine file.txt with

"Hello" on one line and "world!" on the second line.

Now run your new Java program! ^_^

--Hm? Errors?

Ah, yes! We have errors! Why? Because we did not put our file input statements in a try/catch block.

All file input/output statements must have some form of exception handling.

You do not even have to handle the exception, if you do not wish to.

I, however, do not recommend you let exceptions occur without either letting it crash with a stack trace or notifying the user and handling it.

So, let's fix our program.

```

Code import java.util.Scanner;
import java.io.*; // ***Remember to include your IO package!***
public class FileIO
{
    public static String newline = System.getProperty("line.separator");
    public static void main(String [] args)
    {
        Scanner keyboard = new Scanner(System.in);

        // Input file name
        System.out.print("Enter a file name for input: ");
        String fileName = keyboard.nextLine();

        try
        {

```

```

    FileReader fr = new FileReader(fileName);
    BufferedReader br = new BufferedReader(fr);

    String line = br.readLine();

    while(line != null)
    {
        // display
        System.out.println(line);

        // get next line
        line = br.readLine();
    }

    br.close();
}
catch(Exception e)
{
    System.out.println(e);
}
}

```

It is now safe to run your program. :)
 Your output should be something like:
 Code Enter a file name: file.txt
 Hello
 world!
 Press any key to continue. . .
 See, that wasn't so bad, was it?

5.2 File Output

Whew, now time for File Output.

First, let me start by saying the new line character may vary from OS to OS. It may work on one, but not the other.

You want your program to work on all operating systems, don't you? :D

When writing to a file, you should use the system independent new line character.

We can easily do this by using the following statement:

```
Code public static String newline = System.getProperty("line.separator");
```

The String newLine now has a newline character that will work for whichever OS the program is run

from.

Now, let's move on to file output.

In this section, we will assume you want to take the contents of one file and transfer them all to another file.

This requires us to both read a file and write to a file. Good practice! ;)

Let's assume you half of the program implemented (which looks much like the last one):

```
Code import java.util.Scanner;
import java.io.*; // ***Remember to include your IO package!***
public class FileIO
{
    public static String newline = System.getProperty("line.separator");
    public static void main(String [] args)
    {
        Scanner keyboard = new Scanner(System.in);

        // Input file name
        System.out.print("Enter a file name: ");
        String fileName = keyboard.nextLine();

        FileReader fr = new FileReader(fileName);
        BufferedReader br = new BufferedReader(fr);

        String line = br.readLine();

        while(line != null)
        {
            // instead of displaying, we will write
            //write

            // get next line
            line = br.readLine();
        }

        br.close();
    }
}
```

Notice the newline String instantiated right above the main method.

Also, like the comment says, rather than displaying the lines, we will write them.

So what classes do we need to write to a file? We need two, again. `FileWriter`
`BufferedWriter`

Notice these two are File and Buffered Writer classes, not Reader.

Setting up these two objects is much like before. `FileWriter` needs a file, `BufferedWriter` needs a

`FileWriter`.

So, let's move on to the `write()` method. We simply call this method and pass a value to it, and it will write it to the desired file.

So, basically, our code would look something like:

```
Code import java.util.Scanner;
import java.io.*; // ***Remember to include your IO package!***
public class FileIO
```

```

{
    public static String newline = System.getProperty("line.separator");
    public static void main(String [] args)
    {
        Scanner keyboard = new Scanner(System.in);

        // Input file name
        System.out.print("Enter a file name for input: ");
        String fileName = keyboard.nextLine();

        System.out.print("\n\nEnter a file name for output: ");
        String wFileName = keyboard.nextLine();
        try
        {
            FileReader fr = new FileReader(fileName);
            BufferedReader br = new BufferedReader(fr);
            FileWriter fw = new FileWriter(wFileName);
            BufferedWriter bw = new BufferedWriter(fw);

            String line = br.readLine();

            while(line != null)
            {
                // instead of displaying, we will write
                bw.write(line + newline);

                // get next line
                line = br.readLine();
            }

            br.close();
            bw.close();
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}

```

Note: I went over this fairly quickly, please do ask questions if you do not

understand something. I am always happy to answer a question about my tutorials.

7. End

Thank you for reading my tutorial. And as I said above, ask as many questions as you'd like. Also, I would appreciate it if you could notify me of any mistakes/errors.

scaryblackguy - April 27, 2009, 9:18 pm

(Apr 16, 2009 10:29 PM)Psycho Wrote: [\[CITE\]](#) Java Tutorial #2

The syntax for instantiation is:

```
Code ClassName identifier = new ClassName;
```

However, if the class requires any parameters to be passed, it could look this way:

```
Code ClassName identifier = new ClassName(1, 2, 3);
```

The following is also valid:

```
Code ClassName identifier;
```

```
identifier = new ClassName;
```

You just cleared up so much, thank you!

scaryblackguy - September 26, 2009, 7:51 pm

bump for better tutorial

TehPhelix - January 23, 2010, 10:28 pm

Just a heads up on an error I found. I haven't finished reading yet though so I will edit if I find more.

Under arrays you wrote

```
Quote:Code int[] grades = new int[2];
```

```
grades[0] = 10;
```

```
grades[1] = 50;
```

```
grades[2] = 100;
```

The above example instantiates an array of int type, and of size 2.

Array indexes start at 0, so it can technically hold 3 values. In index 0, 1, 2 and 3.

Which is false. Since you made it size 2, you can only hold two variables on int type. In index 0, and index 1. You can not hold anything in indexes 2 or 3 as it will give you a size mismatch error. Just a heads up. Your program would also run an error above.

Thanks though for the tutorial, help me expand on some things I was lacking in.

Psycho - February 3, 2010, 11:56 am

(Jan 23, 2010 11:28 PM)TehPhelix Wrote: [\[CITE\]](#) Just a heads up on

Which is false. Since you made it size 2, you can only hold two variables on int type. In index 0, and index 1. You can not hold anything in indexes 2 or 3 as it will give you a size mismatch error. Just a heads up. Your program would also run an error above.

Thanks though for the tutorial, help me expand on some things I was lacking in.

You are absolutely right...my mistake. I apologize if I misinformed anyone with that. I will make changes.

TheRien - January 4, 2011, 9:39 am

I knew that part was wrong somehow :P My conclusion was slightly different though but it's good to know the way it is now, thanks TehPelix.

Even more (way more) thanks to you though Psycho, once again a great tutorial.

Will there be any more coming from your hand? They fit my entry exactly the way you write them right now (not for advanced people nor for complete retards :P)

Glyceryl - January 4, 2011, 4:07 pm

Not a bad tutorial , but I think it would be wise to add a part on ArrayList in the array section . Also you should show how to create classes , and go over instance fields . Another part to add would be on the different data types such as int , boolean , char , double ..ect . Nonetheless good work .

Morphene - January 4, 2011, 8:04 pm

Very clear I think but I think that you could of spent a little bit more time on the Array Section. Otherwise great tut.

Hi Psycho,

thats a nice tutorial. But I see one litte problem in the code for file input and file output.

Code try

```
{
  FileReader fr = new FileReader(fileName);
  BufferedReader br = new BufferedReader(fr);
```

```
String line = br.readLine();
```

```
while(line != null)
{
  // display
  System.out.println(line);
```

```
    // get next line
    line = br.readLine();
}
```

```
br.close();
}
catch(Exception e)
```

```
{
  System.out.println(e);
```

}You should close the reader within a finally-clause to get shure that it will also be closed after an exception.

Man, I had really trouble with this in a bigger project because someone didn't close streams like this. Some files couldn't be deleted sometimes and it took a while to figure out that open streams where blocking them.

This may not be a problem in a little program which stops running after the exception occured. But such things are forgotten soon, if one doesn't it right everytime.

This is a solution:

Code

```
FileReader fr;
BufferedReader br = null;
try {
  fr = new FileReader(fileName);
  br = new BufferedReader(fr);
```

```
String line = br.readLine();
```

```
while (line != null) {
  // display
  System.out.println(line);
```

```
    // get next line
    line = br.readLine();
}
```

```
} catch (Exception e) {
```

```
        System.out.println(e);
    } finally{
        try {
            br.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}Deque
```

Psycho - January 5, 2011, 9:21 am

Thanks Deque. At the time of writing the original post (almost 2 years ago), it was even a learning experience for myself.

seasnake_730 - January 14, 2011, 4:24 am

Great tutorial :)

I'm learning Java in college atm, but I'm trying to work ahead so I can get more experience. This is the first tutorial I've read that helped me understand input.

Thank you :D

ajayu1992 - December 7, 2011, 10:45 am

thankyou soo soo much for this wonderful tutorial... The first tutorial i read and understood regarding file operations :)

Funn1k - September 6, 2012, 5:50 pm

wow psycho!U just rocked !help me on exceptional handling pls :)

mr.jitterbud - March 29, 2014, 7:30 am

Another great tut. I just relized php seems to be similar to java because this looks kinda the same. Noob thoughts