

Forum:[Hack Forums](#)

Thread: [Basic Java Tutorial](#) | [Variables](#) | [Loops](#) | [Conditionals](#) | [Methods](#) | [Classes](#)

King of Hearts - June 3, 2012, 11:39 am

What Is Java?

Java is a programming language originally developed by James Gosling. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture. Java is a general-purpose, concurrent, class-based, object-oriented language that is specifically designed to have as few implementation dependencies as possible.

[Read more here.](#)

Hello World (code) Code public class HelloWorld{

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

}Hello World Explained

Let's look at the code line by line.public class HelloWorld{

This line defines the class name. The syntax of defining classes is: AccessType class ClassName { Notice the line ends with an opening bracket. Every time you define a class, you have to use opening and closing brackets.

When saving a Java file, the name of the file must be the ClassName. So the HelloWorld class above must be saved as HelloWorld.java

```
    public static void main(String[] args) {
```

This line is the header of the main method. Every main method has a header like this. Methods will be covered in detail later on.

```
        System.out.println("Hello World!");
```

This line is what says what's going to happen. System.out has many different type of print methods. I'll discuss two of them here.

The System.out.println method prints the parameter (what's in the parenthesis) on a new line in the console. This method can also be used without any parameter and that will make the console print an empty line. System.out.println() is useful to print new lines in the console.

The System.out.print method is similar to the println method, but this doesn't print a new line. This method will print something on an existing line in the console. Because of this, if you don't put anything in the parameter, unlike the println method, you'll get a Syntax error (an error which won't let you compile the code).

You can change what gets printed by changing the parameter. If you'd like to print Hey, Saad you'd use: System.out.println("Hey, Saad"); If you are printing a number, you don't need the quotation marks. However if you want to print a string (set of characters) you must use parenthesis.

Most statements in Java need to end in a semicolon except methods, conditionals, and class declarations.

}Remember this rule: Every bracket which is opened must be closed.
The first parenthesis is closing the main method. The second parenthesis is closing the class. Not including any of these parentheses will result in a Parsing Syntax Error.

I'll be covering classes and methods in more detail later on.

Defining Variables

A variable is an object that can hold a specific type of data. So you can basically assign a value a name and get the value by using the name instead of the value. There are many types of variables and you can also create your own types of variables. (Will be discussed later on). Here are the most common variable types:
int - An int is an integer which can hold whole numbers under 2.17 billion in size.

long - A long is just like an integer but it can hold whole numbers larger than 2.17 million.

double - A double can contain numbers with decimal values.

boolean A boolean holds only two values, true or false (0 or 1).

String - A string holds more than one character which can include numbers, symbols and characters. When assigned a value it needs to be assigned with quotation marks before and after the value ("").

char - A char holds only one character. When assigned a value it needs to be assigned with an apostrophe before and after the value('').

Defining variables is easy. There are two ways you can instantiate a variable. The first way is to declare it and assign it at the same time. For example:

Code `int num = 1;` Here we are making an int with the name num and it contains the value 1. The default way to make these types of variables is:

Code `type name = value;` Here the type is the object type (int, double, String etc). The name is any name you want. However, you can define two variables with the same name. The value is the data that is assigned to the name.

Now what if you want to declare a variable but use it later?

Well you can define a variable and set it later. For example:

Code `int num;`

`//other code (this is a one line comment, this will not be read by the compiler).`

`num = 1;` Here you are declaring the variable, but setting it later. If you try to use the variable without assigning it a value, you'll get a syntax error.

You can also assign one variable the value of another. For example:

Code `int a = 1;`

`int b = a;` Here b will also be equal to 1.

Basic Scope

Variables made in the main method will only be able to be used there. If you decide to make a different method and use variables in the main method in the other method, you'll get a syntax error. (Methods will be discussed later on). To fix this, you can define global variables. Global variables can be accessed any where in the class. When declaring global variables you can set the access level. The access level is how your variable can be accessed, meaning can other classes see it or not (classes will be covered later on).

The two common access levels are: public: The access level public means that other classes can see it.

private: Other classes cannot see a variable with access type of private.

This is where you would declare global variables:

Code `public class HelloWorld{`

`public int a = 1;`

```

public int b;

public static void main(String[] args) {

}
}

```

Using Variables

There are many uses for variables. You can use them to add. For example:

```

Code int a = 1;
int b = 2;
int c = a + b;

```

Here c will be equal to 3 because $1 + 2 = 3$.

Here are some operators which can be used with ints, floats, doubles, and longs in Java:

- +: This is the addition operator.

- : This is the subtraction operator.

- /: This is the division operator.

- *: This is the multiplication operator.

- ?: This is the modulus operator. This will give you the remainder of two numbers. E.g $5 \% 2 = 1$ because when dividing 5 by 2 you have a remainder of 1.

- ++: This is the increment operator. This will increase the value by 1. E.g If we have a variable x with the value 2, after x++ the value will be 3.

- : This is the decrement operator. This will decrease the value by 1. E.g If we have a variable x with the value 2, after x-- the value will be 1.

Another important thing you can do with variables is print them.

```

Code String abc = "Hey";
int num = 3;
System.out.println(abc);
System.out.println(num);

```

This will print:
Hey
3

Conditional Statements

Conditional statements basically say that if a condition is met, only then will a piece of code get executed. There are three types of conditional statements. The if statement This basically means that if a condition is true then the code will get executed. If the condition isn't true, nothing will happen. For example:

```

Code int num = 1;
if(num == 1){
    System.out.println("Num = 1");
}

```

Here if num is assigned to be 1, there will be a print statement saying num is 1. If not nothing will happen. Now notice how the if statement is worded:
Code if(VariableNameOrValue == AnotherVariableNameOrValue) {
//code to be executed if the statement is true.
}

Notice the two equal signs. You have to have the two equal signs or you'll get an error.

The if else statement This lets you execute one piece of code if something is true and another if it isn't. For example (I'll use booleans since I have shown you how to define them):

```

Code boolean a = false;
if(a == false){
    System.out.println("A is false");
} else{
    System.out.println("A isn't false.");
}

```

//you can also use this with numbers. For example:

```
int b = 2;
if(b == 2){
    System.out.println("B is 2");
} else{
```

System.out.println("B isn't 2");
 } The if elseif StatementThe if elseif Statement goes through two or more statements one by one and if any of them is true, it executes the code in the statement and stops. For example:

```
Code int b = 2;
if(b == 1){
    System.out.println("B is 1");
} else if(b == 2) {
    System.out.println("B is 2");
} else if(b == 3) {
    System.out.println("B is 3");
} else {
    System.out.println("B isn't 1, 2 or 3");
}
Notice how you can use else statements with an else if statement.
```

Other operators

Now what if you don't want to see if something is equal. What if you want to check is something is bigger then something else? Well there are more operators you can use to compare two values.>: This is the greater than operator. This will be true if the left value is greater than the right.

<: This is the less than operator. This will be true if the left value is smaller than the right.

<=: This is the less than or equal to operator. This will be true if the left value is smaller than or equal to the right.

>=: This is the more than or equal to operator. This will be true if the left value is bigger than or equal then the right.

!: This is the not operator. This will be true if the value isn't something.

.equals(""): This is the equals operator for strings. YOU CAN'T DO == WITH STRINGS. You have to use .equals().

Here's how you use some of these operators:

```
Code int b = 2;
if(b > 1){ //true
if(b < 1){ //false
if(b >= 1){ //true
if(b <= 1){ //false
if(b != 1){ //true
String a = "Saad";
String b = "a";
if(a.equals(b)){//if a equals b -- False
if(!a.equals(b)){//if a does not equal b -- True
```

Loop Statements

A loop is a statement which allows code to be repeatedly executed. There are three types of loops. I'll be covering two types and the third type later on. The While LoopThe while loop goes on till the condition inside it becomes false. Typically we use a counter or a boolean to control loops. Here is an example:

```
Code int counter = 0;
while(counter <= 10){
    System.out.println(counter);
    counter++;
}
```

//Heres a loop controlled by a boolean

```

boolean b = false;
int a = 0;
while(b == false){
    System.out.println(a);
    if(a == 10){
        b = true;
    }
}

```

}These two loops are exactly the same thing and they both will print the numbers 0 to 10, each on a new line. Here's how most loops are formatted:

```

Code int counter = 0;
while(conditional which usually involves counter){
    //code
    //change counter so we don't result in an infinite loop.
}

```

An infinite loop goes on forever and will eventually slow down your computer.

The For LoopThe for loop is just like a while loop but here you make the counter, add the condition and change it in one line. It's basically a way to write a while loop horizontally instead of vertically. Here's the basic syntax:

```

Code for( create counter; condition; change counter ){
    //do code
}

```

Here's how you'd print numbers from 0 to on using a for loop instead of a while:

```

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

```

Some tips:

- Anything variable declared in the loop will only be accessible inside the loop. It will also be reset after the loop increments. The way to avoid this is to not declare variables inside the loop but declare it before the loop.

Methods

Suppose you want to do the same logic of code many times but with different values. Let's say you want to add two different values but the values will be different most of the time (like in a calculator). To do this you use methods. Methods have to be placed outside the main method but inside the class: For example:

```

Code public class HelloWorld{

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
    //method 1
    //method 2
}

```

Methods can be classified into two types. void: This type of method does not return any value. It only changes existing values. Here's how void methods look like:

```

Code AccessLevel void name(Parameters){
    //code
}

```

Here's a completed method which displays numbers from 0 to 10 and how you can call the mehtod.

```

Code public class HelloWorld{

    public static void main(String[] args) {
        displayNumbers();
    }

    public void displayNumbers(){
        for(int counter = 0; counter <= 10; counter++ ){

```

```

        System.out.println(counter);
    }
}

```

}This code will display numbers from 0 to 10. Notice this method has no parameter. A parameter is a way of giving our method values to use.

Let's say you wanted a method to print numbers from 0 till x, where x can be any number. Here's how the method would look like:

Code public class HelloWorld{

```

    public static void main(String[] args) {
        displayNumbers(12);
    }

    public void displayNumbers(int x){
        for(int counter = 0; counter <= x; counter++){
            System.out.println(counter);
        }
    }
}

```

}Notice three things here. I now have something inside the parenthesis when calling the method. I have to have a number inside the parenthesis because the method has an int parameter. If I try to do displayNumbers("String"); I'll get a syntax error. I have to have an int inside the parenthesis.

The second thing is that I have included a parameter when defining a method. To add a parameter you have to add Type name in the parenthesis. To add multiple parameters you separate them with commas.

The third thing which I changed is the 10 in the counter. The 10 is now x meaning the loop will go on till counter is equal to x.

Here's a method which displays numbers from a to b. Where a and b are parameters and can be anything.

Code public class HelloWorld{

```

    public static void main(String[] args) {
        displayNumbers(1, 12);
    }

    public void displayNumbers(int a, int b){
        for(int counter = a; counter <= b; counter++){
            System.out.println(counter);
        }
    }
}

```

} The return statement

The return statement can be used in void methods to stop a method early. It's optional but it can come in handy. For example let's say you want a parameter to be more than 0. So to implement this you can add a if statement and if it's true you can add a return statement. Here's an example of how you'd check to see if the parameter a is less than 0 and if it is how to stop it.

Code public class HelloWorld{

```

    public static void main(String[] args) {
        displayNumbers(1, 12);
    }

    public void displayNumbers(int a, int b){
        if(a < 0){
            return;
        }
    }
}

```

```

    }
    for(int counter = a; counter <= b; counter++){
        System.out.println(counter);
    }
}

```

} Methods that return something

You can also use methods to get values. Here's the basic syntax of defining return methods:

Code AccessLevel returnType name(Parameters){

 returnType a = ...;

 return a;

 }The return type can be any variable type. It can be a String, int, double or any type of variable.

Notice how I have a variable (it can also be a value) after the return statement.

The return statement basically stops the method and sends a value/object back to the caller. (You'll understand later on with examples).

You have to have a return statement in a method that isn't a void method.

For example if you wanted to get what a + b is you'd make a method like this:

Code public int add(int a, int b){

 int c = a + b;

 return c;

 }See how I'm returning c? You can assign returned values to a variable of the same type. For

example:

Code public class HelloWorld{

 public static void main(String[] args) {

 int c = add(1, 12);//here c will be 13

 }

 public int add(int a, int b){

 int c = a + b;

 return c;

 }

} This is just a beginner's guide to methods. Methods can get more complex and there are many types of methods (recursive, overloaded, overridden etc) which I may cover in a later tutorial.

Classes

Classes in Java is the last thing I'll be covering today. Classes are basically object types in Java.

Remember how I said you can make your own variable types? Well this is how. You can have many classes in your program, but usually only one has a main method.

Classes are made to keep similar methods in one place or to make a new object type.

Classes are made like this:

Code accessType class NameOfClass{

}Classes are made to keep similar methods in one place or to make a new object type.Here's a class which has similiar methods together.

Code public class MathOps{

 public static int add(int num1, int num2){

 int answer = num1 + num2;

 return answer;

 }

 public static int subtract(int num1, int num2){

 int answer = num1 - num2;

 return answer;

 }

}Here you see a new word: static. static methods can be used from other classes without making an instance of the class. (You'll understand better when I show you the other type of class).

To use these methods from another class you'd do: MathOps.add(1, 2);

For example:

```
Code public class HelloWorld{

    public static void main(String[] args) {
        int c = MathOps.add(1, 12);//here c will be 13
    }
}
```

} Classes which can be instantiated.

Here's an example of a class which can be instantiated:

```
Code public class Car{
    private String make;
    private String model;

    public Car(String m, String m2){
        make = m;
        model = m2;
    }
    public void Drive(){
        //drive code
    }
}
```

}Here you'll notice something that should ring bells in your brain. I HAVE A METHOD WITH NO RETURN TYPE? WHAT???

Well, notice how it has the same name as the class name. This is a constructor. A constructor is like a method but it doesn't have a return type.

It allows you make an instance of the object. You can have as many constructors as you want (they have to have different parameters) and usually the parameters are assigned to the global variables (which are usually private).

To create instances of objects you use the following syntax:

Code ObjectType name = new ObjectType(Parameters if any);

Here's how you'd make a new Car object in another class.

```
Code public class HelloWorld{

    public static void main(String[] args) {
        Car a = new Car("BMW", "X5");
    }
}
```

}Basically that's making a new car.

I can make as many cars as I want:

```
Code public class HelloWorld{

    public static void main(String[] args) {
        Car a = new Car("BMW", "X5");
        Car a = new Car("Mercedes", "S550");
        Car a = new Car("Lexus", "RX400");
    }
}
```

}Back to the car class. Notice I have a drive method there. The drive method isn't static. That means I can't do Car.drive();

I can however do this:

```
Code public class HelloWorld{

    public static void main(String[] args) {
```

```
Car a = new Car("BMW", "X5");
a.drive();
}
```

}If I had a static method in the Car class, doing a.staticMethod would give me an error. I'd have to do Car.staticMethod

Well that's the end of my tutorial. I'm more than willing to take constructive criticism. However please don't flame because this is my first tutorial and it took me two days to make and perfect. If you'd like to use this on other forums please pm me for permission. If you need any help feel free to pm me (Saad).

It took me 2 days to make this. It only takes 2 seconds to say thanks.

InuyashaITB - June 3, 2012, 11:42 am

I am going to be learning Java over the summer, but i've already learned C++... and i've gotta say, the syntac looks pretty much identical... what is the difference in the two languages?

(don't say stuff like c++ is more powerful, that doesn't help... tell me like difference in memory allocations or like how the objects are referenced and stuff like that)

King of Hearts - June 3, 2012, 11:44 am

(Jun 3, 2012 11:42 AM)InuyashaITB Wrote: [CITE](#) I am going to be learning Java over the summer, but i've already learned C++... and i've gotta say, the syntac looks pretty much identical... what is the difference in the two languages?

(don't say stuff like c++ is more powerful, that doesn't help... tell me like difference in memory allocations or like how the objects are referenced and stuff like that)

I don't know c++, sorry. But Java's syntax is derived from C and C++ so the syntax is quite similar. The main difference is that Java is run on a java virtual machine.

FlyingManiac - June 3, 2012, 4:08 pm

Java is 100% object orientated, C++ is not.

You can still write c++ in the style of c, which is totally procedural.

Also you use Functions from objects for the most simple things like

```
std::cout << "hello world";
```

is in Java

```
System.out.println("Hello World");
```

Every Class in Java inherits from the Object class and can acess its functions like .equals();

Only primitive Datatypes are no Objects.

Therre is no code outside of classes like in C++

Also, try some wiki:

http://en.wikipedia.org/wiki/Comparison_of_Java_and_C%2B%2B

@Op: very nice tutorial, haven't read everything, but seems very detailed

Nov - June 3, 2012, 6:59 pm

Good tutorial for a beginner. Nice job OP.

King of Hearts - June 4, 2012, 3:05 pm

Thanks for the good reviews....If anyone needs help, feel free to pm me.

Frostbite™ - June 4, 2012, 6:32 pm

This really is an amazing tutorial, especially for a first time.

One question though, would it be possible to make a root kit in java? I've heard people say it is, but the fact it is run in a java VM should mean that it isn't. I'm confused.

King of Hearts - June 4, 2012, 6:54 pm

(Jun 4, 2012 06:32 PM)Frostbite™ Wrote: [\[CITE\]](#) This really is an amazing tutorial, especially for a first time.

One question though, would it be possible to make a root kit in java? I've heard people say it is, but the fact it is run in a java VM should mean that it isn't. I'm confused.

To be honest I'm not really into black hat much...However, if rootkit involves editing registry or system files, that can be done. You can also use native C libraries from Java using JNI (so if C can make a rootkit then you can use JNI to accomplish that task using JNI).

Frostbite™ - June 5, 2012, 1:06 am

(Jun 4, 2012 06:54 PM)saadmboy Wrote: [\[CITE\]](#) To be honest I'm not really into black hat much...However, if rootkit involves editing registry or system files, that can be done. You can also use native C libraries from Java using JNI (so if C can make a rootkit then you can use JNI to accomplish that task using JNI).

Ok thanks for your response, but im going to go with C++, it is easier to understand :P

MindOfLogic - June 5, 2012, 6:31 am

Pretty good tutorial here considering it's just a post. For the people who want to understand all the aspects of Java methods or classes, I suggest reading an ebook or looking on the oracle website.

Shadows - June 5, 2012, 5:42 pm

After reading half last night, I've just read the second half.

Confused but I suppose I'm learning. Rep+ man it's a great tutorial.

King of Hearts - June 5, 2012, 7:02 pm

(Jun 5, 2012 05:42 PM)Shadows Wrote: [\[CITE\]](#) After reading half last night, I've just read the second half.

Confused but I suppose I'm learning. Rep+ man it's a great tutorial.

What was confusing? I'm glad you liked it.

Automatic Coding - June 5, 2012, 7:07 pm

Just skimmed over it, never knew about:

%: This is the modulus operator. This will give you the remainder of two numbers. E.g $5 \% 2 = 1$ because when dividing 5 by 2 you have a remainder of 1.

Although, I've never needed it. You learn something every day ^_^

DustinPwnzer - June 5, 2012, 7:53 pm

Great Post, Helped me out lots! Very Simple And Down To The Point!

King of Hearts - June 7, 2012, 3:01 pm

Glad this helped so many people out. If you need any help feel free to pm me.

King of Hearts - September 13, 2012, 6:11 pm

(Jun 5, 2012 07:53 PM)DustinPwnzer Wrote: [\[CITE\]](#) Great Post, Helped me out lots! Very Simple And Down To The Point!

No problem, glad you liked it....This might help others who need help in java, so I'm ing it after a few months.

Jire - September 14, 2012, 3:42 am

Maximum value of 32-bit signed data (int) is 2.147 billion.

King of Hearts - November 8, 2012, 7:03 pm

(Sep 14, 2012 03:42 AM)Ignatius Wrote: [\[CITE\]](#) Maximum value of 32-bit signed data (int) is 2.147 billion.

Alright, I can't edit the tutorial though.

vexia - November 10, 2012, 2:20 pm

Beautiful tutorials for begginers :).

Sparta. - November 10, 2012, 2:39 pm

It is a good tutorial, but it's not a tutorial, it's multiple tutorials put in one thread. Understanding loops isn't basic, so I don't suggest any newbies at Java to start at that point.

King of Hearts - November 13, 2012, 7:23 pm

(Nov 10, 2012 03:20 PM)vexia Wrote: [\[CITE\]](#) Beautiful tutorials for begginers :).

Thanks.

@ SPARTA

I know, but this is just more like reference in my opinion if you stop developing and then decide to start again.

iDontHustle - December 25, 2012, 10:01 am

Very helpful tutorial. Thanks for the share!

King of Hearts - December 28, 2012, 4:14 pm

(Dec 25, 2012 11:01 AM)iDontHustle Wrote: [\[CITE\]](#) Very helpful tutorial. Thanks for the share!

Glad you like the tutorial. :)

-Saad

tenthfeb - January 3, 2013, 12:32 pm

thank you for this, i always wanted to learn Java

King of Hearts - May 8, 2013, 5:22 pm

(Jan 3, 2013 01:32 PM)tenthfeb Wrote: [\[CITE\]](#) thank you for this, i always wanted to learn Java

No problem. I hope others find this useful also.

-Saad

Purple BMX - June 2, 2013, 5:39 pm

Who ever rated this one star, go fuck yourself. This is a good tutorial, thanks Saad.

dog135 - June 2, 2013, 7:04 pm

I started out with c++ and switched to Java. There's actually a few big differences.

Pointers are transparent. For instance, instead of using "class->function()" you would use "class.function()".

You do not manage your memory allocation, Java does. Once a pointer no longer has references, it's marked for collection. You never "free" a pointer, instead you either let it go out of scope, or set your local pointer to null.

Some things like multithreading is easier. You can:

```
new Thread(new Runnable(){public void run(){doSomething();}}).start();
```

or just have your class inherit from Runnable, override the run function, then call

"yourFunction.start();" This is usefull if you want to give your thread a constructor.

Java also has a class loader class. This allows you to load external code and call it internally.

Java is designed to run on pretty much any system. Once you save your code as a jar file, (a type of zip file) your code will run on Linux, Mac, Windows, Android, etc.

If you already write C++, I suggest reading "Java in a nutshell". It's a down to business book that's written for people who already write in other languages. I still use my copy as a reference guide. It has the complete list of Java classes taking up over half the book.

BTW, speeds can be comparable to compiled code. The Java VM uses a "just in time" (JIT) compiler to compile your Java code for cpu intensive tasks.

Infobesity - June 3, 2013, 11:53 pm

I was always encouraged to keep my instance variables private. Also for loops you don't have to do

```
Code boolean b = true;
```

```
while(b == true){
```

```
//
```

```
}that defeats the whole purpose of boolean haha, you can simply do while(b). '==' should be used
```

when comparing non-boolean values. Other that, after a quick skim, a very nice basic tutorial on

Java. Though you should include starter tips like, how to setup java for coding and etc.

Lie.

King of Hearts - June 4, 2013, 3:22 pm

(Jun 3, 2013 11:53 PM)Lie. Wrote: [\[CITE\]](#) I was always encouraged to keep my instance variables private. Also for loops you don't have to do

```
Code boolean b = true;
```

```
while(b == true){
```

```
//
```

```
}that defeats the whole purpose of boolean haha, you can simply do while(b). '==' should be used
```

when comparing non-boolean values. Other that, after a quick skim, a very nice basic tutorial on

Java. Though you should include starter tips like, how to setup java for coding and etc.

Lie.

Thanks for the review, you're right about the boolean thing in the while statement. I'll fix that (if the message isn't too long). The reason why I didn't add basic tutorials like setting up Java and JDK is because there are many others out there on HF and the interwebs. I wanted to make a tutorial on HF that gives a taste of everything and helps potential Java programmers to get a feel of Java.

Also M, thanks for 5 stars and for liking the tutorial, now why don't you start programming :P

Purple BMX - June 4, 2013, 5:32 pm

(Jun 4, 2013 03:22 PM)SaadTronics Wrote: [\[CITE\]](#) Thanks for the review, you're right about the boolean thing in the while statement. I'll fix that (if the message isn't too long). The reason why I didn't add basic tutorials like setting up Java and JDK is because there are many others out there on HF and the interwebs. I wanted to make a tutorial on HF that gives a taste of everything and helps potential Java programmers to get a feel of Java.

Also M, thanks for 5 stars and for liking the tutorial, now why don't you start programming :P

I tried java script one time...

I'll stick to marketing, haha!

Acad3m1cM1n3r - June 4, 2013, 7:17 pm

Nice and organized tutorial for beginners, just one small thing that bothered me.

Code public class HelloWorld{There should be a space after HelloWorld/Before the opening brace.
:P

sixfighter - June 16, 2013, 11:47 am

My 2 seconds to say thank you!

P.D: My first post, I'm completely noob. I want to know more about this world so I'll be around.

P.D2: sorry for my english!

King of Hearts - July 17, 2013, 5:59 pm

(Jun 16, 2013 11:47 AM)sixfighter Wrote: [\[CITE\]](#) My 2 seconds to say thank you!

P.D: My first post, I'm completely noob. I want to know more about this world so I'll be around.

P.D2: sorry for my english!

No problem. Glad you like it. Hope others find this useful as well.

Sparta. - July 17, 2013, 8:17 pm

(Jun 4, 2013 07:17 PM)Matthew?? Wrote: [\[CITE\]](#) Nice and organized tutorial for beginners, just one small thing that bothered me.

Code public class HelloWorld{There should be a space after HelloWorld/Before the opening brace.
:P

That's not really a rule, just a small little problem that has to do with habits in coding.

Natecat - September 28, 2013, 9:59 pm

You confused brackets with parenthesis in the Hello World! section of your tutorial. Parenthesis are (), brackets are either [] or {}

King of Hearts - December 2, 2013, 5:41 pm

Hope others find this useful....

KoH

King of Hearts - January 1, 2014, 11:30 am

Happy new year everyone. Hope this tutorial helps you all.

dashahman - January 5, 2014, 9:42 pm

Thanks alot man, really learned alot about classes in this one.

vulgarX - January 6, 2014, 6:24 pm

Java is an interesting language. It's like C# with more portability. It has a longer shelf life than any language I know still in existence.

King of Hearts - January 19, 2014, 10:30 am

I'm glad you guys found this tutorial useful.

luv2dai - January 19, 2014, 2:08 pm

Great job with the tutorial man; Thanks for the contribution to the community.

King of Hearts - January 28, 2014, 8:46 pm

(Jan 19, 2014 03:08 PM)Jixen Wrote: [\[CITE\]](#) Great job with the tutorial man; Thanks for the

contribution to the community.

I'm glad you liked it. Hope others find it useful as well.

King of Hearts - May 25, 2014, 6:58 pm

I hope people have found this useful.

AlbinoKush - May 25, 2014, 7:52 pm

I noticed you check the value of booleans with the "==" operator. Of course this works, but it's not the common industry practice, which looks like:

```
Code boolean foo = false;
```

```
//check if true
```

```
if (foo) {
```

```
    //logic
```

```
}
```

```
//check if false
```

```
if (!foo) {
```

```
    //logic
```

```
}
```

King of Hearts - May 25, 2014, 8:36 pm

(May 25, 2014 07:52 PM)AlbinoKush Wrote: [\[CITE\]](#) I noticed you check the value of booleans with the "==" operator. Of course this works, but it's not the common industry practice, which looks like:

```
Code boolean foo = false;
```

```
//check if true
```

```
if (foo) {
```

```
    //logic
```

```
}
```

```
//check if false
```

```
if (!foo) {
```

```
    //logic
```

```
}
```

I use the equal operators so that it's more intuitive for those new to programming. If people see if(!bool) they won't know it means that if(bool == false)

Thanks for the suggestion though.

AlbinoKush - May 25, 2014, 9:25 pm

(May 25, 2014 08:36 PM)King of Hearts Wrote: [\[CITE\]](#) I use the equal operators so that it's more intuitive for those new to programming. If people see if(!bool) they won't know it means that if(bool == false)

Thanks for the suggestion though.

Then you should include it in the tutorial. They should know everything within the parentheses of an if statement is essentially a boolean.

I also noticed you fail to follow the conventions a few times. Please make reference to:

<http://www.oracle.com/technetwork/java/index-135089.html>

imahacker97 - May 27, 2014, 2:18 pm

Nice guide! This is not only well written but it is EXTREMELY in depth considering how short it is! I was reading through this and if I had this when I was learning the very basics of Java, I would have learned it so much easier. This is short and to the point with nice examples of all things listed.

King of Hearts - June 2, 2014, 12:57 pm

(May 27, 2014 02:18 PM)imahacker97 Wrote: [\[CITE\]](#) Nice guide! This is not only well written but it is EXTREMELY in depth considering how short it is! I was reading through this and if I had this when I was learning the very basics of Java, I would have learned it so much easier. This is short and to the point with nice examples of all things listed.

Thanks a lot. I'm glad you found it useful.

King of Hearts - January 4, 2015, 10:45 am

Hasn't been posted on since June. More people should be able to find it.

maxsmith333 - January 5, 2015, 8:23 pm

can wait to try this our thanks a lot man

King of Hearts - January 13, 2015, 7:35 am

(Jan 5, 2015 09:23 PM)maxsmith333 Wrote: [\[CITE\]](#) can wait to try this our thanks a lot man
No problem. If you need help, feel free to pm me or post a thread in the subforum.

TheNeutron - January 17, 2015, 1:27 pm

This is really an informative and well explained thread... I'd really love more from you mate :)

King of Hearts - June 6, 2015, 12:39 am

(Jan 17, 2015 02:27 PM)-=Hack0pz=- Wrote: [\[CITE\]](#) This is really an informative and well explained thread... I'd really love more from you mate :)

No problem. If you need help, feel free to pm me or post a thread in the subforum.

Vanilja - June 6, 2015, 6:27 am

Classes part helped me a lot, always had a little trouble understanding constructors, but this was clear and easy to understand.

King of Hearts - September 2, 2015, 9:45 pm

(Jun 6, 2015 06:27 AM)Glindis Wrote: [\[CITE\]](#) Classes part helped me a lot, always had a little trouble understanding constructors, but this was clear and easy to understand.

Thanks a lot. If you need help, feel free to pm me or post a thread in the subforum.

FlowDesigns - May 18, 2016, 7:29 am

Hi King of Hearths

thank you for explaining all this stuff!

Found some little mistakes:

Quote:--: This is the decrement operator. This will decrease the value by 1. E.g If we have a variable x with the value 2, after x++ the value will be 1.

Quote:long - A long is just like an integer but it can hold whole numbers larger then 2.17 million. Theoretically the sentence is correct but you were obviously refering to the limit of positive integer values (32 bit) wich is billion not million.

int (32 bit)

$-(2^{(32)/2})$ to $(2^{(32)/2})-1$ total of $2^{(32)}-1$ values

long (64 bit)

$-(2^{(64)/2})$ to $(2^{(64)/2})-1$ total of $2^{(64)}-1$ values

```
Code int counter = 0;
while(counter <= 10){
    System.out.println(counter);
    counter++;
}
```

//Heres a loop controlled by a boolean

```
boolean b = false;
int a = 0;
while(b == false){
    System.out.println(a);
    if(a == 10){
        b = true;
    }
}
```

}Quote:These two loops are exactly the same thing and they both will print the numbers 0 to 10, each on a new line.

The second loop is an infinite loop. I was confused while trying to understand it so I tried it out on eclipse and shit went crazy :P

Don Genovese - May 28, 2016, 4:48 pm

Thanks great tutorial! Amazing how you do all of this for other people. Great job man!

King of Hearts - June 4, 2016, 9:44 pm

@FlowDesigns, I'll take a look.

I totally forgot that I made this thread. I might redo some if it later and make it look nice (although there's already a great beginner's thread already stickied).

unfinishedsentenc - August 10, 2016, 5:04 pm

your avatar is making me cringe dawg just sayin, im just lookin out for a brotha bap bap remove that avatar shoutout to my boi

good luck with ur future endavours

elninocartel - February 19, 2017, 6:08 pm

helpful post.. quick intro read on Java..

RengarCode - February 24, 2017, 10:35 pm

Helped out and understood majority of it. Many thanks.

Marwix - February 25, 2017, 5:38 am

Really beautiful guide.

I'm sure it will help a lot of people get started with Java.