Windows Programming with Java

Oğuzhan Öztaş

Java Books

-Java Examples in a nutshell
O'REILLY
-Java Cookbook
O'REILLY
-Java 2 Core Language Little Black Book
Paraglyph Press
-core java 2, volume 1 fundamentals 5th
The Sun Microsystems Press - Prentice Hall
-Java Tutorial 3th Edition - A Short Course on the Basics
(2000) - Addison Wesley
-Javascript Bible Gold.Edition
Hungry Minder

Contents

- what is java?
- How java programs work?
- Standalone java program in DOS mode
- Standalone java program in Windows mode
- Java Applets
- Java Events
- Graphical User Interface
- Input/Output
- Database Access with SQL

Java Features

- independent from operating systems
- Object-oriented language
- internet programming
- Pocet PC and mobile phone programming
- And new technology

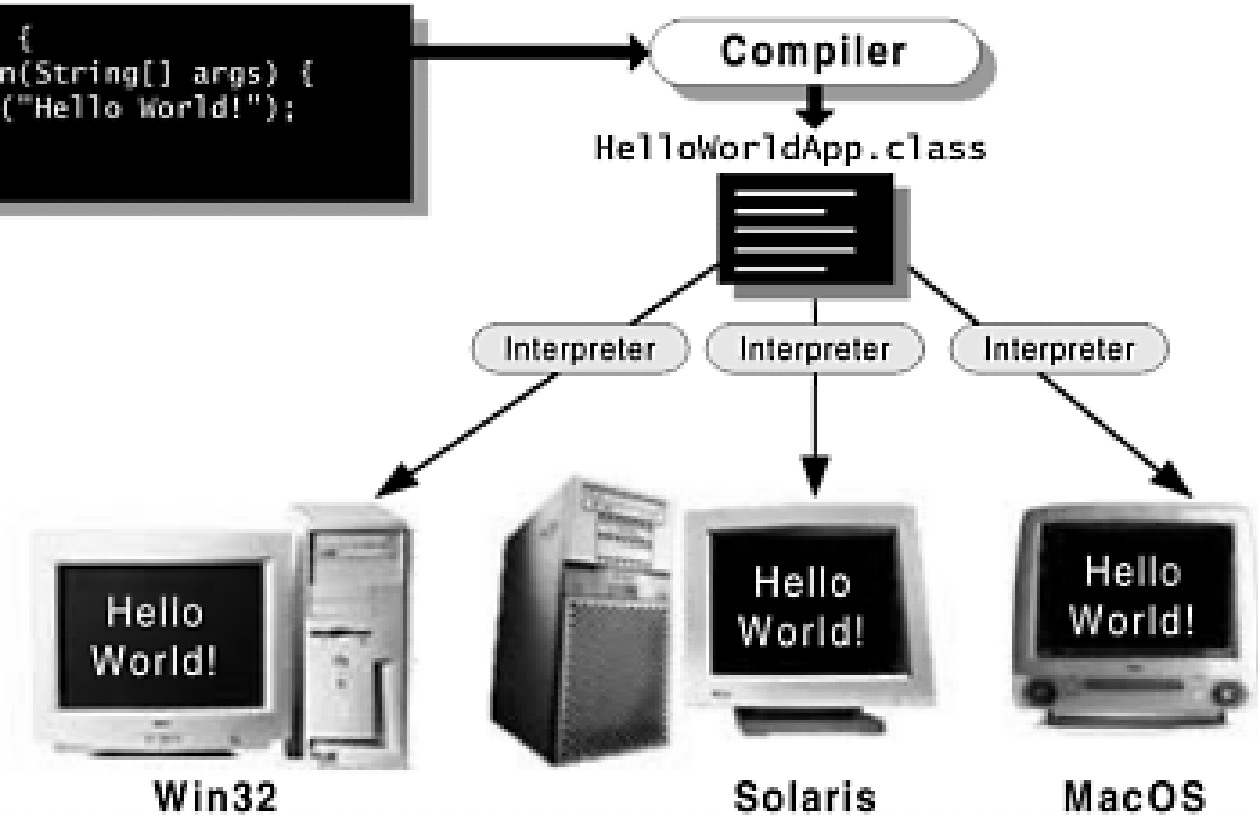
Java Features

- **Get started quickly**: Although the Java programming language is a powerful object-oriented language, it's easy to learn, especially for programmers already familiar with C or C++.
- **Write less code**: Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in the Java programming language can be four times smaller than the same program in C++.
- **Write better code**: The Java programming language encourages good coding practices, and its garbage collection helps you avoid memory leaks. Its object orientation, its JavaBeans component architecture, and its wide-ranging, extensible API let you reuse other people's code and introduce fewer bugs.
- **Develop programs more quickly**: Your development time may be twice as fast as writing the same program in C++. Why? You write fewer lines of code with the Java programming language, and it is a simpler programming language than C++.
- **Avoid platform dependencies with 100% Pure Java™:** You can keep your program portable by avoiding the use of libraries written in other languages. The 100% Pure Java™ Product Certification Program has a repository of historical process manuals, white papers, brochures, and similar materials online at: http://java.sun.com/100percent/
- **Write once, run anywhere**: Because 100% Pure Java programs are compiled into machine-independent bytecodes, they run consistently on any Java platform.
- **Distribute software more easily**: You can easily upgrade certain types of programs, such as applets, from a central server. Applets take advantage of the feature of allowing new classes to be loaded "on the fly," without recompiling the entire program.

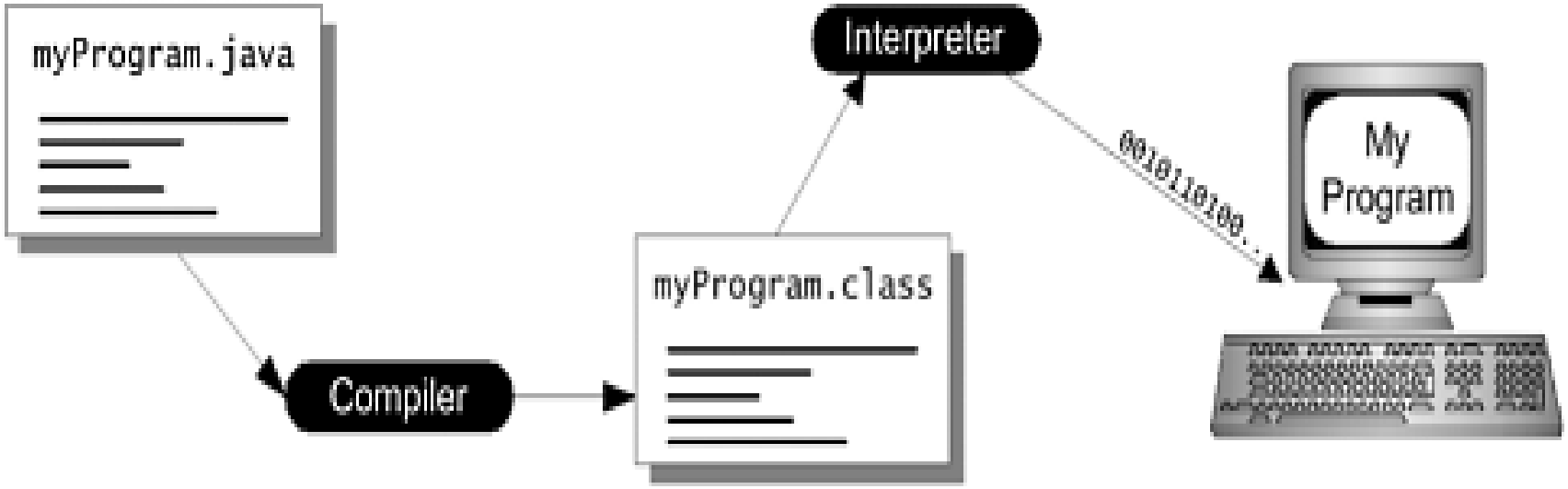
independent from operating systems

HelloWorldApp.java

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



How java programs work?



How java programs work?

1. Write simple java program using any editor.
2. Save it to the directory with giving a name.
3. The name as “sample.java”.
4. in the command line write (in two step)
   - Javac sample.java
   - Java sample

Java Commands

Java Compiler : **javac**

compiles Java programs into bytecodes

Java Interpreter : **java**

executes Java bytecodes

Java Runtime Interpreter : **jre**

for end-users not requiring all of the development-related options available with the **java** tool

Java AppletViewer : **appletviewer**

for testing and running applets

A simple java program

```
public class Hello {
    public static void main(String[] args) {
      system.out.println(“Hello World!”);
  }
}
```

To compile and run it

C:\> Javac Hello.java

C:\> Java Hello

Hello World!

Java applet

First, start NotePad. In a new document, type the following code:

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet {
    public void paint(Graphics g) {
        // Display "Hello world!"
        g.drawString("Hello world!", 50, 25);
    }
}
```

Java applet

Second, you also need an HTML file to accompany your applet. Type the following code into a new NotePad document:

```
<HTML>
    <HEAD>
        <TITLE>A Simple Program</TITLE>
    </HEAD>
     <BODY>
            Here is the output of my program:
      <APPLET CODE="HelloWorld.class" WIDTH=150 HEIGHT=25>
       </APPLET>
     </BODY>
</HTML>
```

Save this code to a file called Hello.html.

What Is an Object?

Objects are key to understanding object-oriented technology. You can look around you now and see many examples of real-world objects: your dog, your desk, your television set, your bicycle.

These real-world objects share two characteristics: They all have state and behavior. For example, dogs have state (name, color, breed, hungry) and behavior (barking, fetching, and wagging tail). Bicycles have state (current gear, current pedal cadence, two wheels, number of gears) and behavior (braking, accelerating, slowing down, changing gears).

Software objects are modeled after real-world objects in that they too have state and behavior. A software object maintains its state in one or more variables. A variable is an item of data named by an identifier. A software object implements its behavior with methods. A method is a function (subroutine) associated with an object.

Definition

An object is a software bundle of variables and related methods.

What Is a Class?

In the real world, you often have many objects of the same kind. For example, your bicycle is just one of many bicycles in the world. Using object-oriented terminology, we say that your bicycle object is an instance of the class of objects known as bicycles. Bicycles have some state (current gear, current cadence, two wheels) and behavior (change gears, brake) in common. However, each bicycle's state is independent of and can be different from that of other bicycles.

When building bicycles, manufacturers take advantage of the fact that bicycles share characteristics, building many bicycles from the same blueprint. It would be very inefficient to produce a new blueprint for every individual bicycle manufactured.

In object-oriented software, it's also possible to have many objects of the same kind that share characteristics: rectangles, employee records, video clips, and so on. Like the bicycle manufacturers, you can take advantage of the fact that objects of the same kind are similar and you can create a blueprint for those objects. A software blueprint for objects is called a class

Definition:

A class is a blueprint, or prototype, that defines the variables and the methods common to all objects of a certain kind.

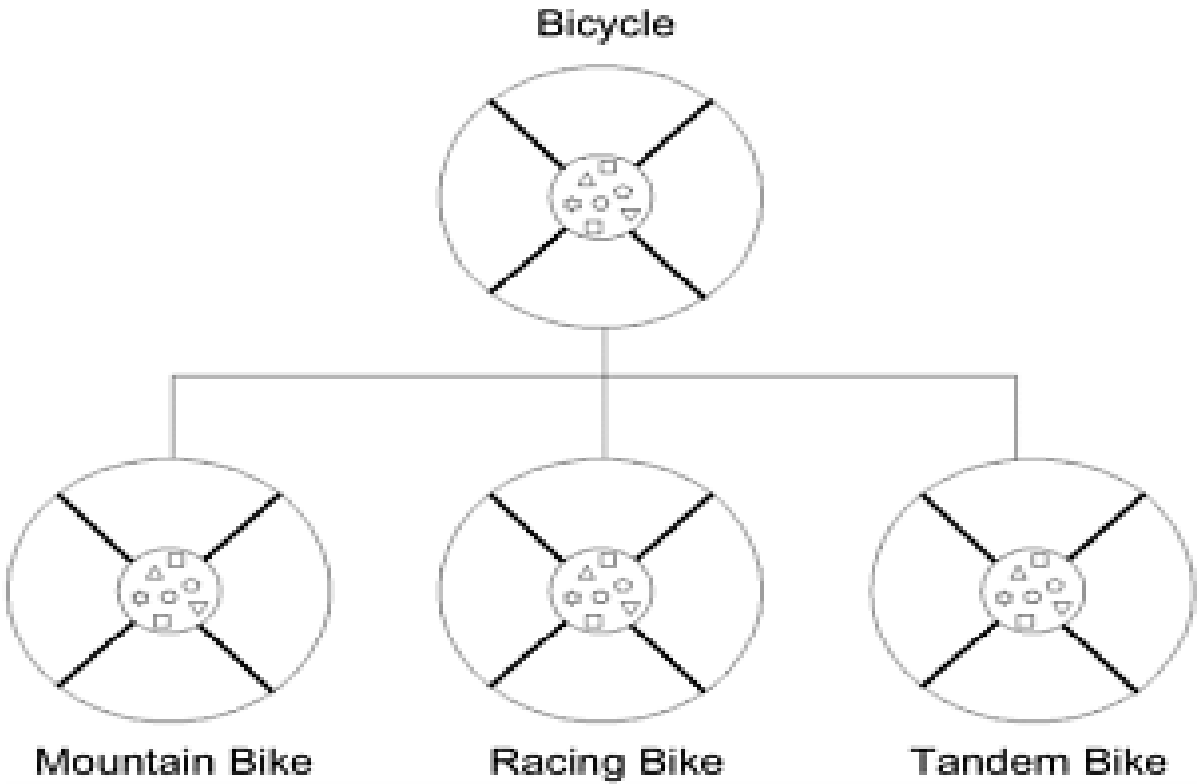
What Is Inheritance?

Generally speaking, objects are defined in terms of classes. You know a lot about an object by knowing its class. Even if you don't know what a penny-farthing is, if I told you it was a bicycle, you would know that it had two wheels, handle bars, and pedals.

Object-oriented systems take this a step further and allow classes to be defined in terms of other classes. For example, mountain bikes, racing bikes, and tandems are all kinds of bicycles. In object-oriented terminology, mountain bikes, racing bikes, and tandems are all subclasses of the bicycle class. Similarly, the bicycle class is the superclass of mountain bikes, racing bikes, and tandems. This relationship is shown in Figure.

What Is Inheritance?

The hierarchy of bicycle classes.



Java integer types

Java integer types		
Type	Storage Requirement	Range (inclusive)
`int`	4 bytes	-2,147,483,648 to 2,147,483, 647 (just over 2 billion)
`short`	2 bytes	-32,768 to 32,767
`long`	8 bytes	-9,223,372,036,854,775,808L to 9,223,372,036,854,775,807L
`byte`	1 byte	-128 to 127

Floating-point types

Floating-point types		
Type	Storage Requirement	Range
`float`	4 bytes	approximately ±3.40282347E+38F (6–7 significant decimal digits)
`double`	8 bytes	approximately ±1.79769313486231570E+308 (15 significant decimal digits)

Special characters

Special characters		
Escape Sequence	**Name**	**Unicode Value**
\b	backspace	\u0008
\t	tab	\u0009
\n	linefeed	\u000a
\r	carriage return	\u000d
\"	double quote	\u0022
\'	single quote	\u0027
\\	backslash	\u005c

Variables

In Java, every variable has a *type.* You declare a variable by placing the type first, followed by the name of the variable. Here are some examples:

```
double salary;
int vacationDays;
long earthPopulation;
char yesChar;
boolean done;
```

Notice the semicolon at the end of each declaration. The semicolon is necessary because a declaration is a complete Java statement.

Assignments and Initializations

```
int vacationDays; // this is a declaration
vacationDays = 12; // this is an assignment
int vacationDays = 12; // this is an initialization
```

Operators

x += 4;

is equivalent to

x = x + 4;

(In general, place the operator to the left of the = sign, such as *= or %=.)

Shortcut Increment and Decrement Operators

Shortcut Increment and Decrement Operators

Operator	Use	Description
++	op++	Increments op by 1; evaluates to the value of op before it was incremented
++	++op	Increments op by 1; evaluates to the value of op after it was incremented
--	op--	Decrements op by 1; evaluates to the value of op before it was decremented
--	--op	Decrements op by 1; evaluates to the value of op after it was decremented

Increment and Decrement Operators

```
int m = 7;
int n = 7;

int a = 2 * ++m; // now a is 16, m is 8
int b = 2 * n++; // now b is 14, n is 8
```

Relational Operators

Relational Operators

Operator	Use	Description
>	op1 > op2	Returns true if op1 is greater than op2
>=	op1 >= op2	Returns true if op1 is greater than or equal to op2
<	op1 < op2	Returns true if op1 is less than op2
<=	op1 <= op2	Returns true if op1 is less than or equal to op2
==	op1 == op2	Returns true if op1 and op2 are equal
!=	op1 != op2	Returns true if op1 and op2 are not equal

Conditional Operators

Conditional Operators

Operator	Use	Description
`&&`	`op1 && op2`	Returns `true` if `op1` and `op2` are both `true`; conditionally evaluates `op2`
`||`	`op1 || op2`	Returns `true` if either `op1` or `op2` is `true`; conditionally evaluates `op2`
`!`	`!op`	Returns `true` if `op` is `false`
`&`	`op1 & op2`	Returns `true` if `op1` and `op2` are both boolean and both `true`; always evaluates op1 and `op2` If both operands are numbers, performs bitwise `AND` operation
`|`	`op1 | op2`	Returns `true` if both `op1` and `op2` are boolean, and either `op1` or `op2` is `true`; always evaluates `op1` and `op2` If both operands are numbers, performs bitwise inclusive `OR` operation
`^`	`op1 ^ op2`	Returns true if `op1` and `op2` are different, that is, if one or the other of the operands, but not both, is true

Relational and boolean Operators

uses **&&** for the logical “and” operator and **||** for the logical “or” operator.

To test for equality you use a double equal sign, **==**.

Use a **!=** for inequality.

you have the usual **<** (less than), **>** (greater than), **<=** (less than or equal), and **>=** (greater than or equal) operators.

Java supports the ternary ?: operator that is occasionally useful. The expression

condition ? *e1* : *e2*

x < y ? x : y

Mathematical Functions and Constants

Mathematical functions in Math class.
(import java.math.*;)

```
double x = 4;
y = Math.sqrt(x);
System.out.println(y); // prints 2.0

double y = Math.pow(x, a);
```

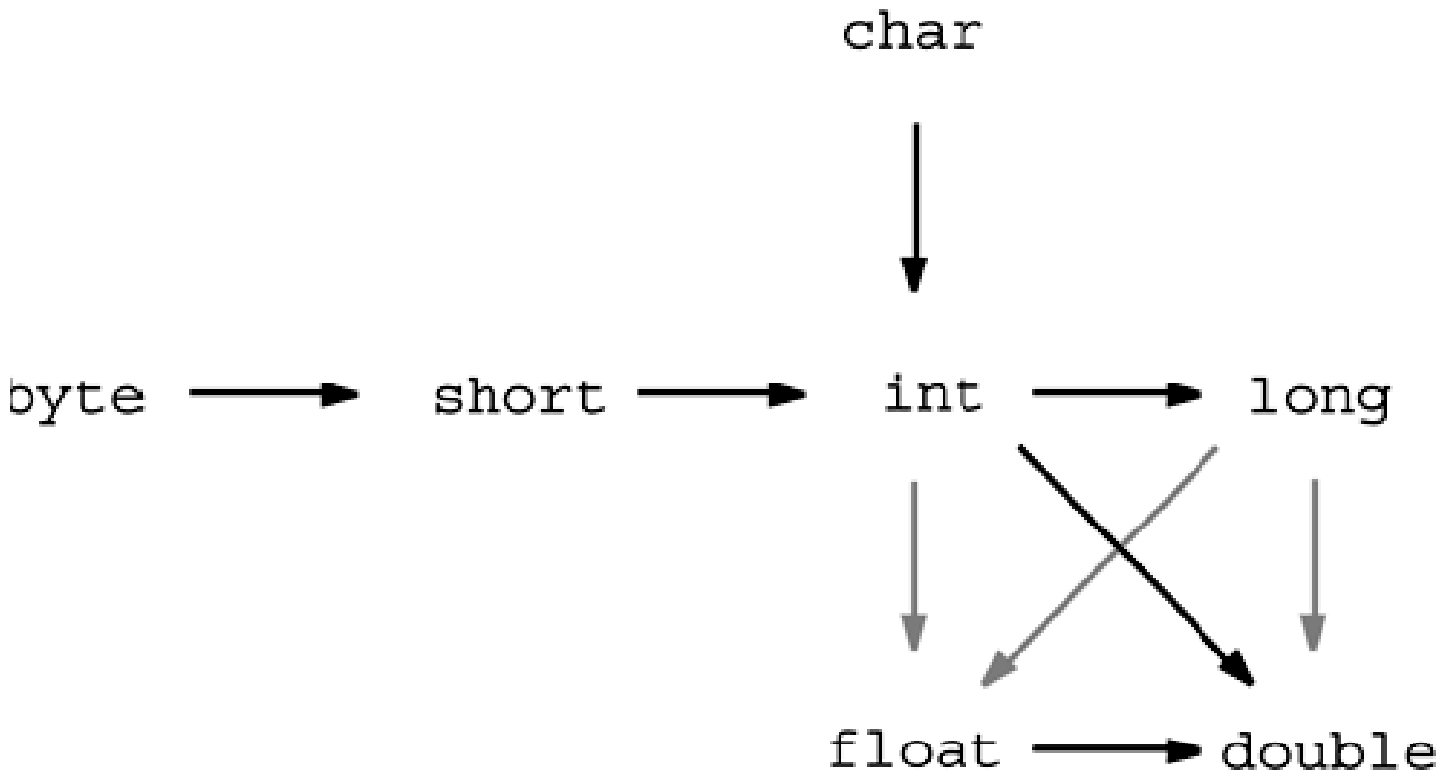
Mathematical Functions and Constants

The Math class supplies the usual trigonometric functions
Math.sin
Math.cos
Math.tan
Math.atan
Math.atan2

and the exponential function and its inverse, the natural log:
Math.exp
Math.log

Finally, there are two constants
Math.PI
Math.E

Legal conversions between numeric types



Casts

in a formula, you can redefine any variables type.

```
double x = 9.997;
int nx = (int)x;
```

Round function

If you want to *round* a floating-point number to the *nearest* integer (which is the more useful operation in most cases), use the Math.round method:

```
double x = 9.997;
int nx = (int)Math.round(x);
```

Parentheses and Operator Hierarchy

Operator precedence	
Operators	**Associativity**
`[] . ()` (method call)	left to right
`! ~ ++ --` + (unary) – (unary) `()` (cast) `new`	right to left
`* / %`	left to right
`+ -`	left to right
`<< >> >>>`	left to right
`< <= > >= instanceof`	left to right
`== !=`	left to right
`&`	left to right
`^`	left to right
`|`	left to right
`&&`	left to right
`||`	left to right
`?:`	left to right
`= += -= *= /= %= &= |= ^= <<= >>= >>>=`	right to left

Parentheses and Operator Hierarchy

If no parentheses are used, operations are performed in the hierarchical order indicated. Operators on the same level are processed from left to right, except for those that are right associative, as indicated in the table.

Strings

```
String e = ""; // an empty string
String greeting = "Hello";
String expletive = "Expletive";
String PG13 = "deleted";
String message = expletive + PG13;
int age = 13;
String rating = "PG" + age;
```

This feature is commonly used in output statements; for example,

```
System.out.println("The answer is " + answer);
```

Substring,length

```
String greeting = "Hello";
String s = greeting.substring(0, 4); // is ‘Hell’
int n = greeting.length(); // is 5.
char last = greeting.charAt(4); // fourth is 'o'
greeting = greeting.substring(0, 4) + "!";// is ‘Hell!’
```

Formatting Output

The NumberFormat class in the java.text package has three methods that yield standard *formatters* for (import java.text.*;)

- numbers
- currency values
- percentage values

These functions are in the same order:

- NumberFormat.getNumberInstance()
- NumberFormat.getCurrencyInstance()
- NumberFormat.getPercentInstance()

Formatting Output

x = 10000.0 / 3.0;
System.out.print(x);
prints
3333.3333333333335
That is a problem if you want to display, for example, dollars and cents.
3,333.333
$3,333.33
333,333%

double x = 10000.0 / 3.0;
NumberFormat formatter = NumberFormat.getNumberInstance();
formatter.setMaximumFractionDigits(4);
formatter.setMinimumIntegerDigits(6);
String s = formatter.format(x); // the string "003,333.3333"