

PROGRAMMING IN



Wake up and smell the coffee!

Software

Java Development Kit (JDK) - <http://java.sun.com/javase/downloads/index.jsp>

Eclipse Platform - <http://www.eclipse.org/>

Reference

The Java Tutorial - <http://java.sun.com/docs/books/tutorial/index.html>

Java Language API - <http://java.sun.com/javase/reference/api.jsp>

Java SE Documentation - <http://java.sun.com/javase/downloads/index.jsp>

Java SE Source Code - <http://java.sun.com/javase/downloads/index.jsp>

Sun Microsystems



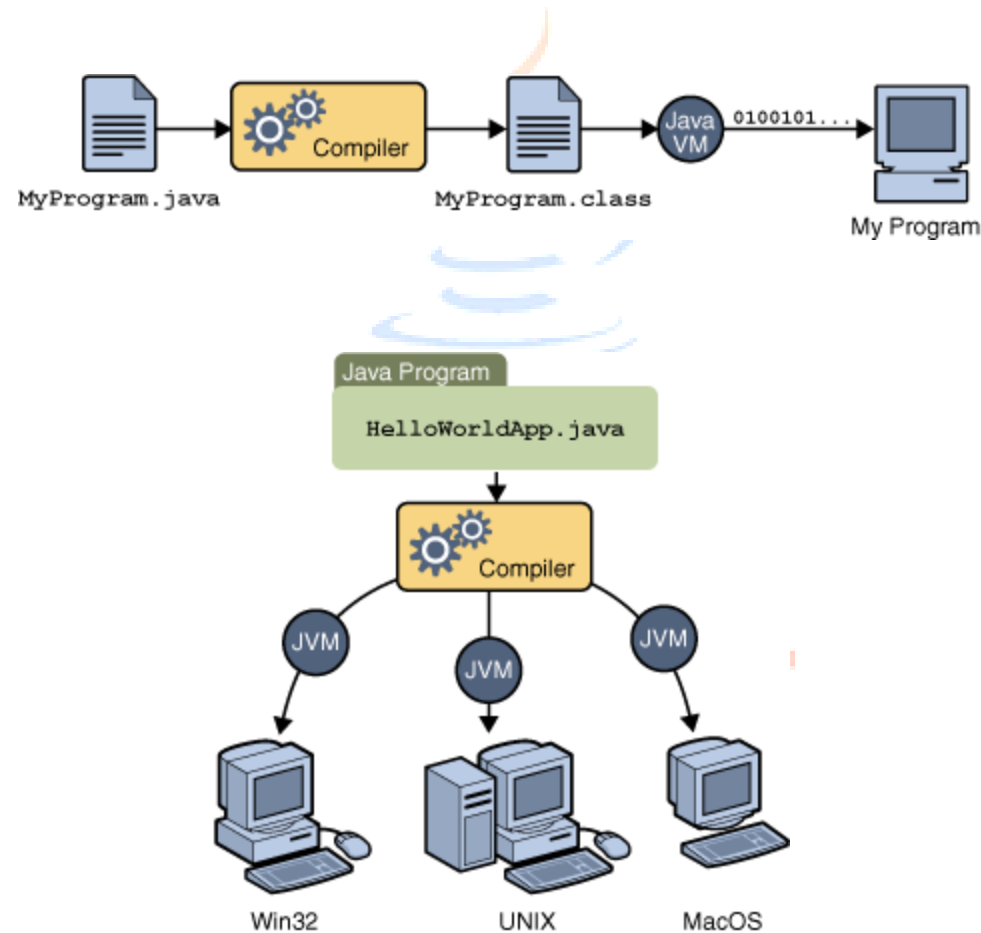
Not your daddy's cup of Joe...

There were five primary goals in the creation of the Java language:

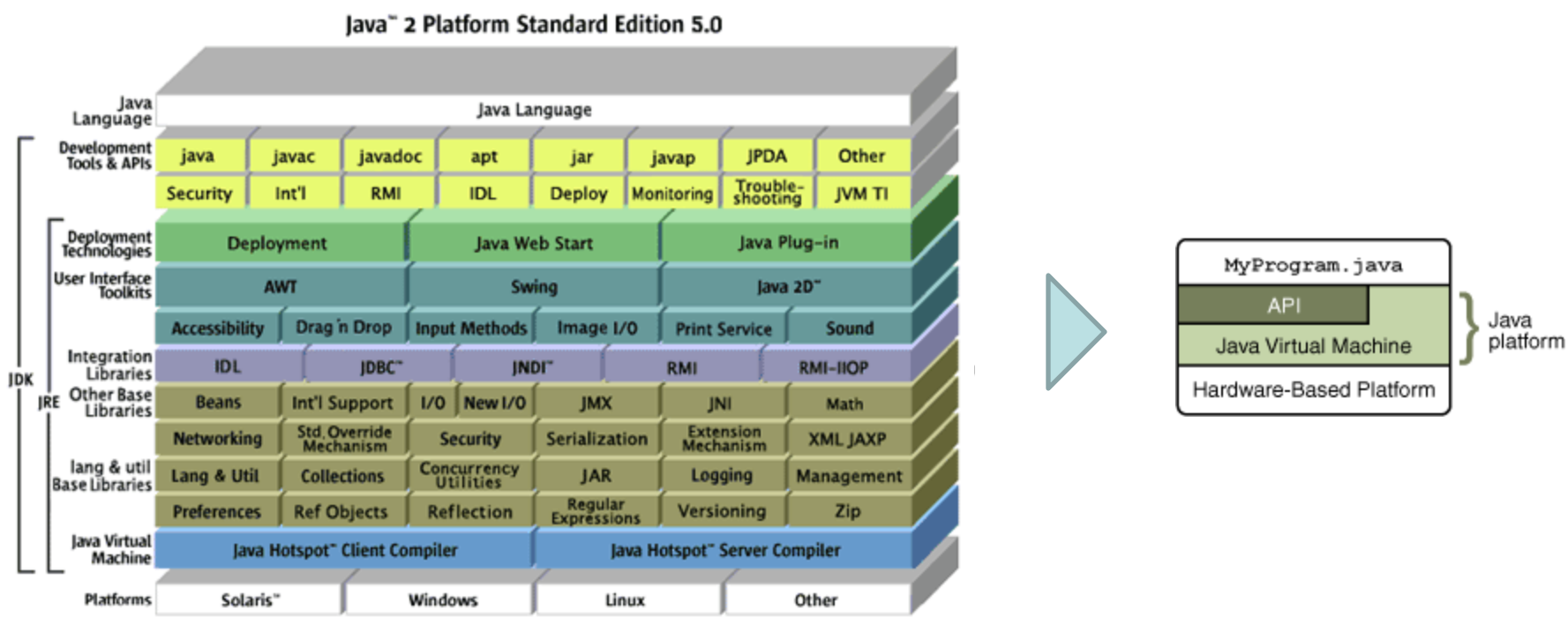
- 1) It should be "simple, object oriented, and familiar".
- 2) It should be "robust and secure".
- 3) It should be "architecture neutral and portable".
- 4) It should execute with "high performance".
- 5) It should be "interpreted, threaded, and dynamic".

Java was designed to be **safe**, **simple**, and **powerful**.

From beans to brew



What's in the cup?



Caf or decaf?

Java is awesome at...

- Cross-platform GUIs
- Large, complex applications
- Embedded systems
- Web-deployment
- Debugging

Java sucks at...

- Short scripts
- Low-level system code
- Malicious code
- Web-deployment
- High-volume data processing



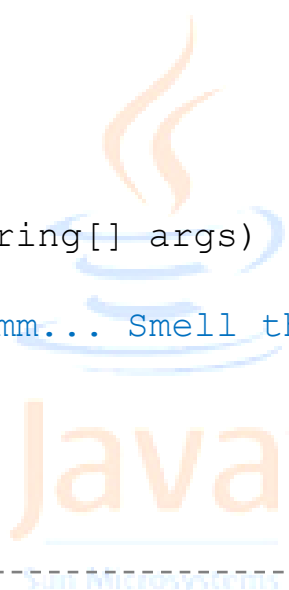
```
package tvald.intro2java;
/**
 * This is a demo program.
 * @author tvald
 * @date 1/5/2009
 */
public class Hello {

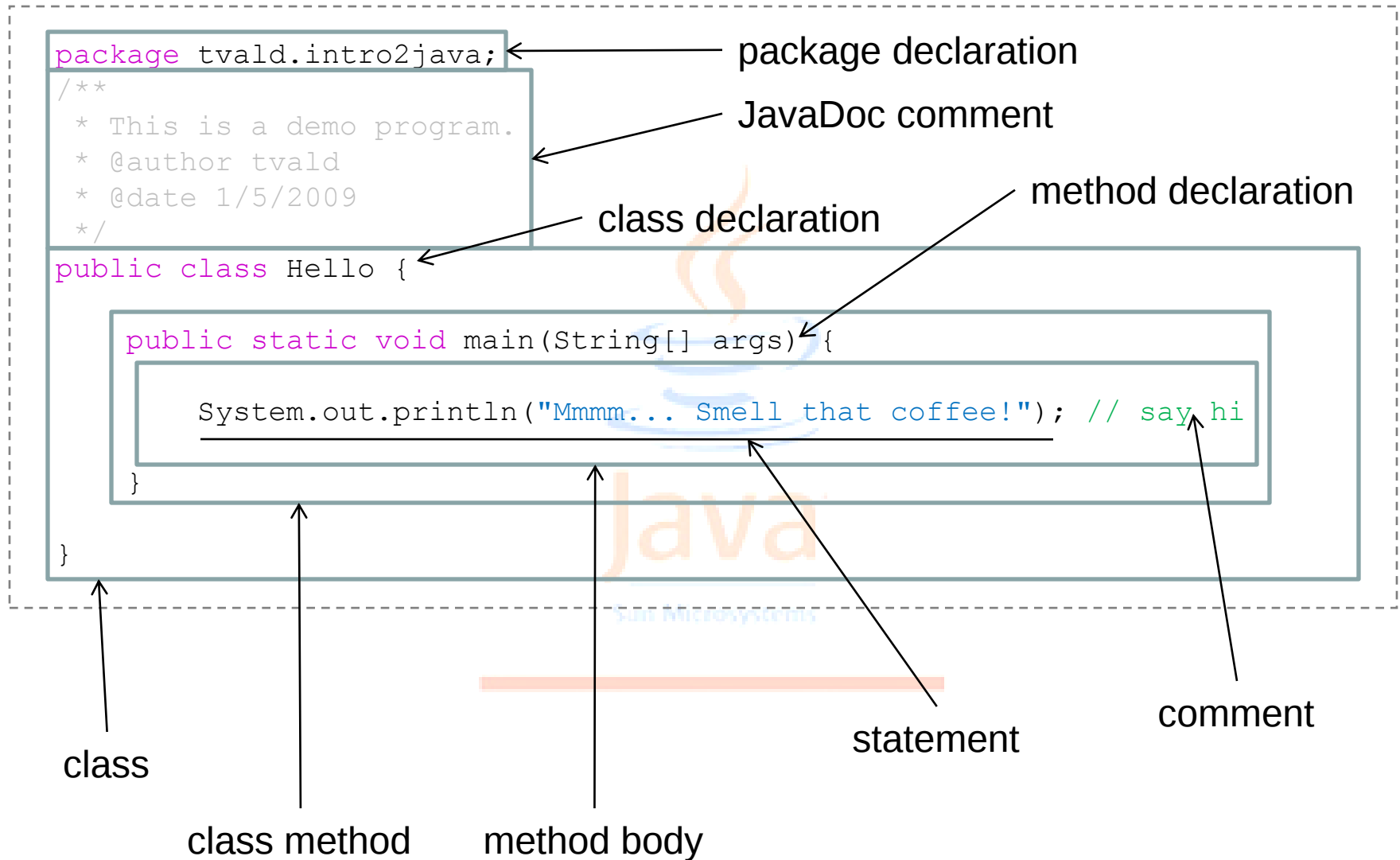
    public static void main(String[] args) {

        System.out.println("Mmmm... Smell that coffee!"); // say hi

    }

}
```

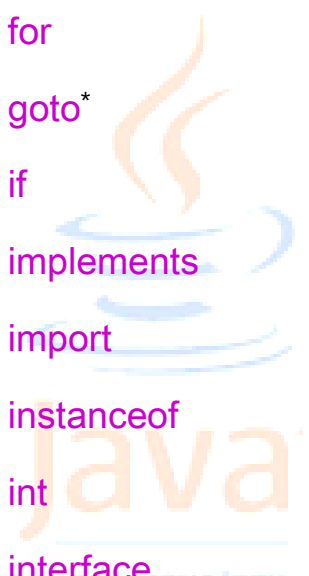
The Java logo is centered in the background. It features a stylized orange coffee cup with steam rising from it, positioned above the word "Java" in a large, orange, sans-serif font. Below "Java" is the text "Sun Microsystems" in a smaller, blue, sans-serif font.



SYNTAX



Keywords



abstract	continue	for	new	switch
assert	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const*	float	native	super	while

Literals: true false null

* not used

Basics

```
// Single line comment

/*
 * Multi-line
 * comment
 */

/**
 * JavaDocs documentation
 */

{
    // blocks contain statements
    statement;
    another statement;
}
```



Java is *imperative*

The importance of being consistent...

```

class                               Hello
    { public
        static
        void    main
            (
                String    [
                    ]
                System
                    out    .
                    println (
                        "Mmm... Coffee!"
                    )
            ;
        }
    }

```



Variables

```
<type> name = expression; // declaration, with optional initialization  
name = expression; // assignment, by value
```

```
char c;  
int i = 2;  
int j = i+3;  
String string;
```

```
//Declarations of the same type may be strung together with commas  
int a, A = 2, a_ = 3;
```

Convention:

All variable names begin with lowercase letter.

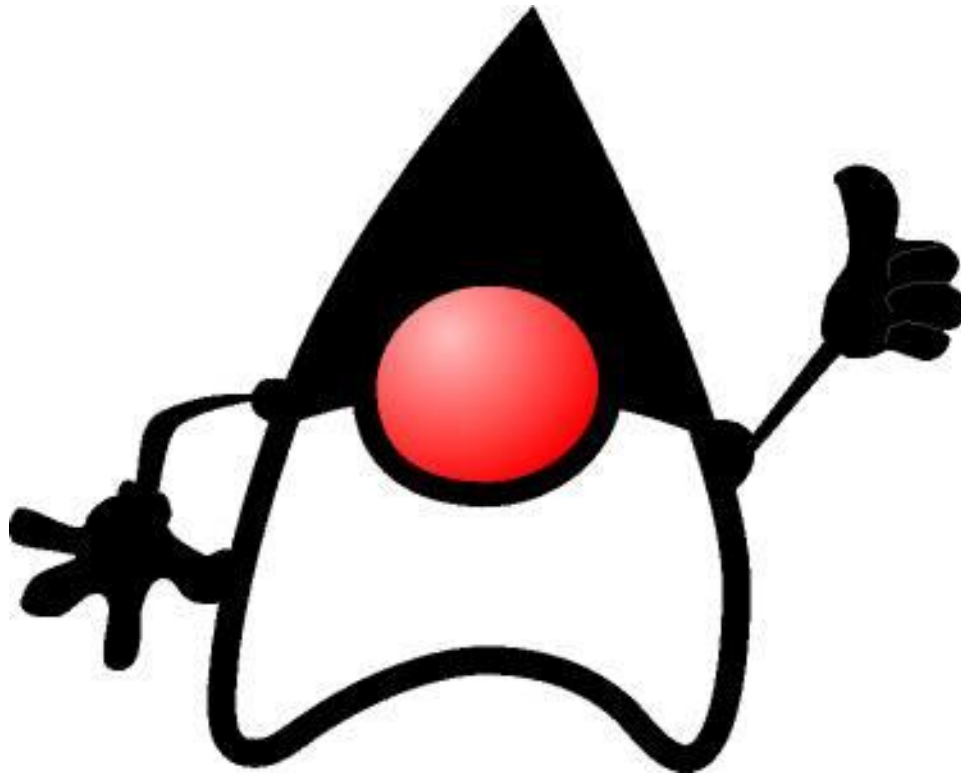
Use “camel case” - firstSecondThird.

Use only alphanumeric characters, and occasionally underscore.

Use a noun!

Java typing is *static*, *strict*, and *nominative*.

The Java Compiler is Your Friend!



Primitives

```
/* integers */
byte b = 127; // 8 bits signed; -128 to 127
short s = 127; // 16 bits signed; -32,768 to 32,767
int i = 127; // 32 bits signed; -2,147,483,648 to 2,147,483,647
long l = 200000000000000L; // 64 bits signed

int dec = 26, oct = 032, hex = 0x1a, HeX = 0X1Ab0;

/* real numbers */
float f = 1.0f; // 32 bits signed
double d = 2.0; // 64 bits signed

double dec = 123.4, sci = 1.234e2;

/* booleans */
boolean t = true, f = false;

/* text */
char c = 'c', unicode = '\u00F1', tab='\t';
String s = "Hello world!\nAnd hello again!";
```

Operators

Operators	
postfix	<i>expr++ expr--</i>
unary	<i>++expr --expr +expr -expr ~ !</i> ←
multiplicative	<i>* / %</i>
additive	<i>+ -</i>
shift	<i><< >> >>></i>
relational	<i>< > <= >= instanceof</i>
equality	<i>== !=</i>
bitwise AND	<i>&</i>
bitwise exclusive OR	<i>^</i>
bitwise inclusive OR	<i> </i>
logical AND	<i>&&</i> ←
logical OR	<i> </i> ←
ternary	<i>? :</i>
assignment	<i>= += -= *= /= %= &= ^= = <<= >>= >>>=</i>

~ - bitwise NOT
! - logical NOT

“short-circuit” as soon
as result is determined

Strings

A String is an Object, NOT an array of characters.

A String is immutable.

Any character is valid between quotes, except a CR / LF

```
String s = "Hello World!";  
String longText = " Supercalifragilisticexpialidocious is " +  
    "a very long word."; // concatenation  
  
String numstr = "I can count to " + 3 + '.';
```

Special characters can be escaped

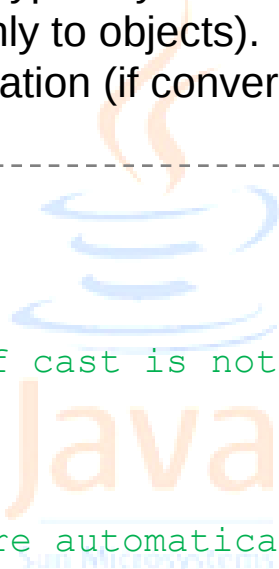
- \b (backspace)
- \t (tab)
- \n (line feed)
- \f (form feed)
- \r (carriage return)
- \" (double quote)
- ' (single quote)
- \\ (backslash)

Casting

Variables can be converted to different types by *casting*.

Must be compatible types (applies mainly to objects).

Number conversion is handled by truncation (if converting to integer) then modulus size.



```
int i;  
double d = 2.75;  
  
//compiler generates warning if cast is not present:  
i = (int)d; // i is 2  
  
i = 2;  
d = i; // less exact numbers are automatically promoted
```

Conditionals



```
if (conditional expression) {
    statement;
    ...
} else if (condition2) {
    statement;
    ...
} else {
    statement;
    ...
}

if (condition) statement;

(condition ? expression : elseExpression) // ternary inline conditional
System.out.println("Good " + (Time.isMorning() ?
    "morning" : "afternoon") + "!");

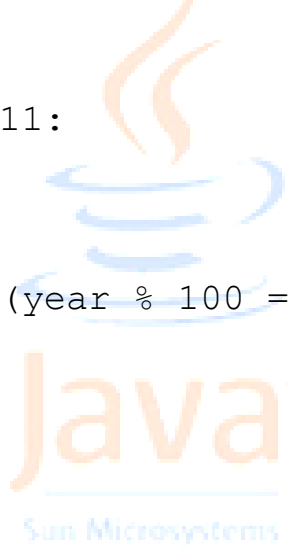
int i;
if ((i = 1) == 0) statement;
if (i = 1) statement; // invalid, number != boolean
```

Switch

```
switch (month) {  
    case 1: System.out.println("January"); break;  
    case 2: System.out.println("February"); break;  
    case 3: System.out.println("March"); break;  
    case 4: System.out.println("April"); break;  
    case 5: System.out.println("May"); break;  
    case 6: System.out.println("June"); break;  
    case 7: System.out.println("July"); break;  
    case 8: System.out.println("August"); break;  
    case 9: System.out.println("September"); break;  
    case 10: System.out.println("October"); break;  
    case 11: System.out.println("November"); break;  
    case 12: System.out.println("December"); break;  
    default: System.out.println("Invalid month."); break;  
}
```

Switch

```
switch (month) {  
    case 1: case 3: case 5: case 7: case 8: case 10: case 12:  
        numDays = 31;  
        break;  
    case 4: case 6: case 9: case 11:  
        numDays = 30;  
        break;  
    case 2:  
        if ( ((year % 4 == 0) && !(year % 100 == 0)) || (year % 400 == 0) )  
            numDays = 29;  
        else  
            numDays = 28;  
        break;  
    default:  
        System.out.println("Invalid month.");  
        break;  
}
```



Loops

```
while (expression) {  
    statement(s)  
}  
  
while (true){  
    // infinite loop  
    if (exitCondition) break;  
}  
  
do {  
    statement(s)  
} while (expression);
```



Scope

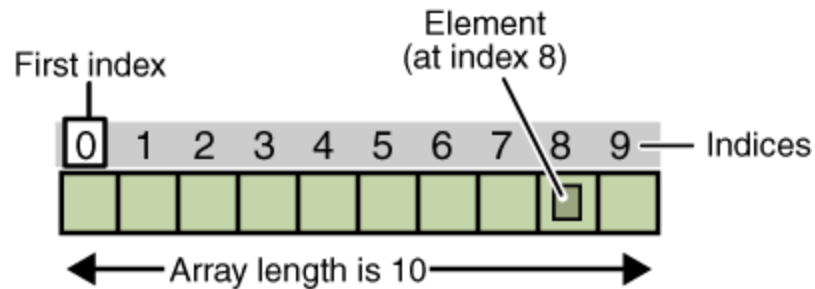
Where can I access a variable?

Inside the block where you declare the variable.

```
if (a < 10) {  
    int c = 2;  
} else {  
    int c = 4;  
}  
System.out.println(c); // ERROR! C isn't in scope!
```



Arrays



```
<type>[] name = new <type>[length];
```

```
int[] intArray = new int[]; // length not necessary
```

```
int[] intArray = new int[10]; // 10 elements, #0 - 9
```

```
String[] stringArray = new String[3]; // Objects work too.
```

```
stringArray.length; // 3
```

```
intArray.length; // 10
```

NOTE: Java Arrays are first-class Objects, so initialized to null.

Arrays

```
int[] intArray = new int[3] = { 1, 2, 3};
int[][] intArray2 = new int[3][4] = { {1,2,3,4},
                                       {5,6,7,8},
                                       {9,10,11,12} };

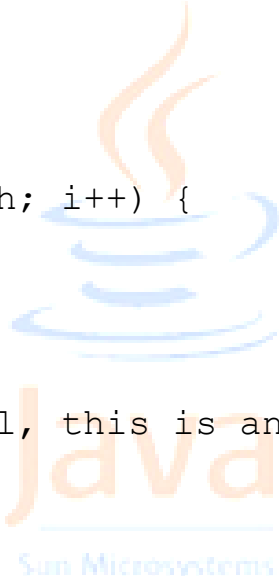
intArray2 = new int[3][]; // element arrays are now null
intArray2[0] = new int[1];
intArray2[1] = new int[2]; // element arrays can have different lengths
intArray2.length; // 3
intArray2[0].length; // 1;

// efficient copying
public static void arraycopy(
    Object src, int srcPos, Object dest, int destPos, int length);

System.arraycopy(intArray, 1, intArray2[1], 0, 2);
intArray2[1][1]; // 3
```

For Loops

```
for (initialization; termination; increment) {  
    statement(s)  
}  
  
int[] array = new int[10];  
for (int i = 0; i < array.length; i++) {  
    statements;  
}  
  
for ( ; ; ) {  
    // expressions are optional, this is an infinite loop  
}  
  
for (int element : array) {  
    // enhanced for loop (also works for objects in a collection)  
}
```



Branching

```
out: // label any kind of loop
while (condition1) {

    myloop: // nested label
    for ( ; condition2 ; ) {

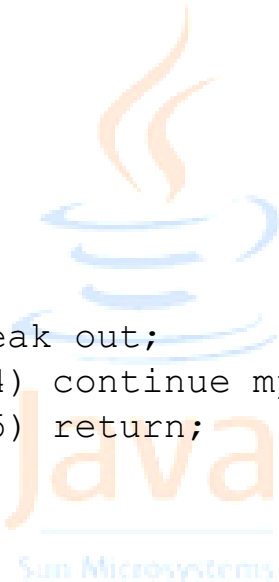
        while (true) {

            if (condition3) break out;
            else if (condition4) continue myloop;
            else if (condition5) return;

        }

    }

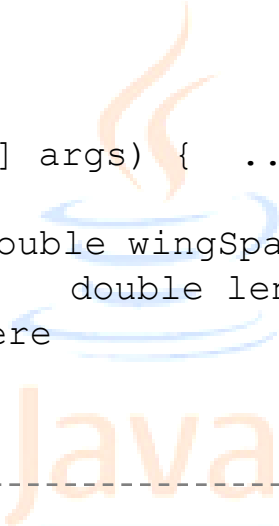
}
```



```
// return MUST match method type
boolean foo() { return false; }
```

Method Declaration

```
modifiers <type> name (parameter list) throws exceptionlist {  
    // method body  
}  
  
public static void main(String[] args) { ... }  
  
public double calculateAnswer(double wingSpan, int numberOfEngines,  
                             double length, double grossTons) {  
    //do the calculation here  
    return result;  
}
```



Convention:

Method names begin with an lowercase letter

Use “camel case” - firstSecondThird.

Use only letters.

Use a verb!

Method Overloading

Method Signature: *name (parameters)*

Java compiler will choose the right method based on the supplied name and parameters

```
public void println(String s) {  
    ...  
}  
public void println(int i) {  
    ...  
}  
public void println(double f) {  
    ...  
}  
public void println(boolean b) {  
    ...  
}
```



Parameters

```
public void foo (parameters) { }
```

Parameters may be of *any* data type - arrays, primitives, objects, whatever...

Can't pass a method, but you can pass an object and invoke its methods.

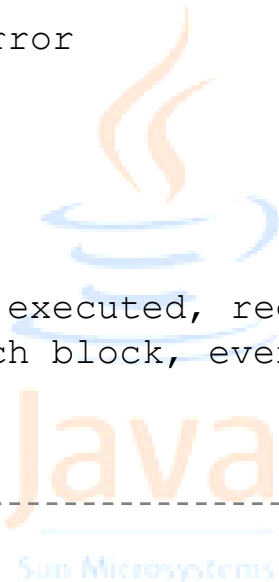
Varargs (shortcut to create an array):

```
public Polygon polygonFrom(int... corners) {  
    for (int i = 0; i < corners.length; i++) {  
        // more method body code follows that creates  
        // and returns a polygon connecting the Points  
    }  
}
```

Both primitives and object references are passed *by value*.

Error Handling

```
try {  
    // this may cause an error  
} catch (Exception e) {  
    // error recovery  
    e.printStackTrace();  
    return;  
} finally {  
    /* this will ALWAYS be executed, regardless of what happens  
     * within a try or catch block, even with a return or break.  
     */  
}
```



Three types of problems:

- **Checked Exceptions:** most problems, follow the Catch or Specify requirement.
- **Runtime Exceptions:** exceptional cases, usually indicate a logic flaw or misuse of API.
- **Error:** computer unplugged, processor melts, Armageddon, whatever... things you can't really do anything about, but you can try if you want.

Error Handling

```
>java MyClass  
  
java.lang.NullPointerException  
    at MyClass.mash(MyClass.java:9)  
    at MyClass.crunch(MyClass.java:6)  
    at MyClass.main(MyClass.java:3)
```



Method Declaration with Exceptions

```
modifiers type name (parameter list) throws exceptionlist {  
    // method body  
}  
  
public void readFile(String filename) throws FileNotFoundException,  
                                           IOException {  
}
```



What is a package?

A package is a namespace for organizing classes and interfaces in a logical manner. Placing your code into packages makes large software projects easier to manage.

```
package edu.mit.sipb.iap;

...

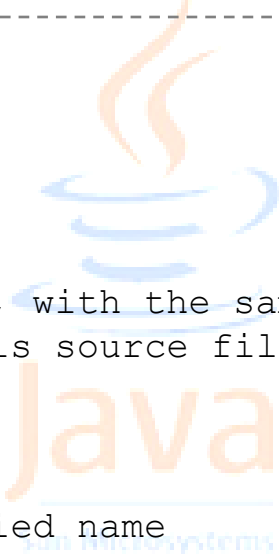
// class declarations
// NOTE: only one public class, with the same name as the file,
// is accessible outside of this source file

...

java.lang.Math // fully qualified name

...

import static java.lang.Math.PI; // import constant (static import)
double rad = 2 * PI;
```



Java™ 2 Platform

Standard Ed. 5.0

All Classes

Packages

- [java.applet](#)
- [java.awt](#)
- [java.awt.color](#)
- [java.awt.datatransfer](#)
- [java.awt.dnd](#)

All Classes

- [AbstractAction](#)
- [AbstractBorder](#)
- [AbstractButton](#)
- [AbstractCellEditor](#)
- [AbstractCollection](#)
- [AbstractColorChooserPanel](#)
- [AbstractDocument](#)
- [AbstractDocument.AttributeCo](#)

Overview Package Class Use Tree Deprecated Index Help

[PREV](#) [NEXT](#)
[FRAMES](#) [NO FRAMES](#)

Java™ 2 Platform Standard Edition 5.0 API Specification

This document is the API specification for the Java 2 Platform Standard Edition 5.0.

See:

[Description](#)

Java 2 Platform Packages	
<u>java.applet</u>	Provides the classes necessary to create an applet and the class uses to communicate with its applet context.
<u>java.awt</u>	Contains all of the classes for creating user interfaces and for and images.
<u>java.awt.color</u>	Provides classes for color spaces.