Introduction to Java Programming Language

Based on Junji Zhi notes, University of Toronto

Content

- Java Language Syntax
- Program examples
- Compiling, Running and Debugging Java code

Java programming Language

- Some buzzwords for Java
  - "Write Once, Run Anywhere"
  - Simple
  - **Object oriented**
  - Distributed
  - Multithreaded
  - Dynamic
  - **Architecture neutral**
  - Portable
  - High performance
  - Robust
  - Secure

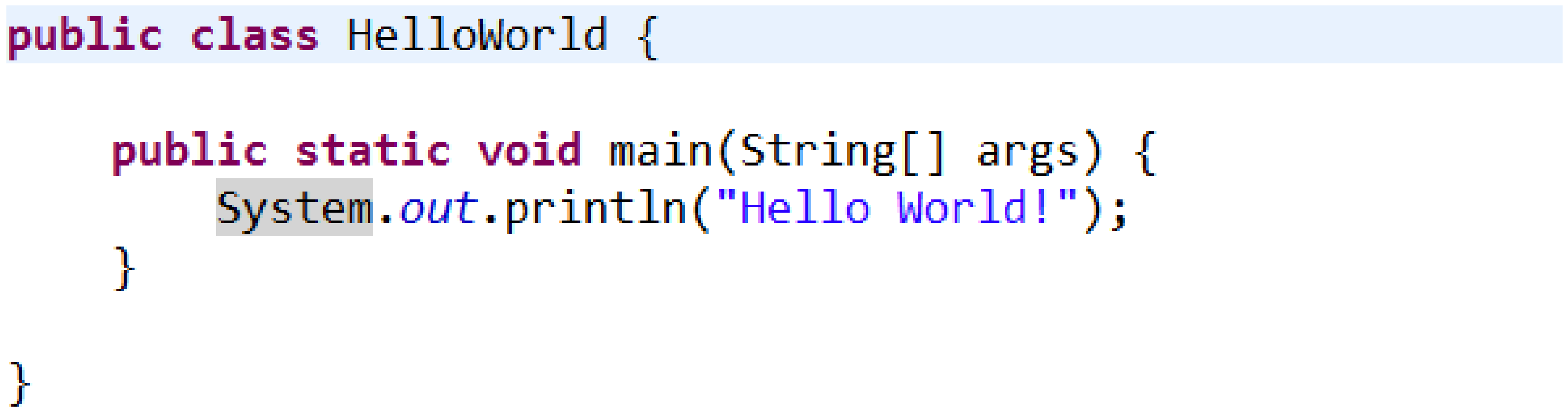
Basic Java Syntax

Hello world

```
public class HelloWorld {
    /**
     * @param args
     */
    public static void main(String[] args) {
        System.out.println("Hello World! I am new to Java.");
    }
}
```

- What is the filename of this program?
- What is the start point?
- How do you compile and run?

Example: Hello World Program



```java
public class HelloWorld {

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

}
```

- Everything is in a class
- One file, one public class
- In the runnable public class:
  - `public static void main(String [] args)`

Primitive Types and Variables

- boolean, char, byte, short, int, long, float, double etc.
- These basic (or primitive) types are the only types that are not objects (due to performance issues).
- This means that you don’t use the new operator to create a primitive variable.
- Declaring primitive variables:

```
float initVal;
int retVal, index = 2;
double gamma = 1.2, brightness;
boolean valueOk = false;
```

Primitive Data Types

- **Primitive Data Types**: byte, short, int, long, float, double, boolean, char
- **Arrays** are also a class

```
long [] a = new long[5];
```

  - You can get the length by visiting the length field of array object a, like this: `a.length`
- **String** class is very commonly used to represents character strings, for example

```
String s1 = “Hello ”, s2 = “World!”;
String s3 = s1 + s2;
```

Declaring Variables

```
int n = 1;
char ch = ‘A’;
String s = “Hello”;
Long l = new Long(100000);
boolean done = false;
final double pi = 3.14159265358979323846;
Employee joe = new Employee();
char [] a = new char[3];
Vector v = new Vector();
```

Initialisation

- If no value is assigned prior to use, then the compiler will give an error
- Java sets primitive variables to zero or false in the case of a boolean variable
- All object references are initially set to null
- An array of anything is an object
  - Set to null on declaration
  - Elements to zero, false, or null on creation

Assignment

- All Java assignments are right associative

  int a = 1, b = 2, c = 5;

  a = b = c;

  System.out.print(“a= “ + a + “b= “ + b + “c= “ + c);

- What is the value of a, b & c
- Done right to left: a = (b = c);

```
public class Variables {
    public static void main(String[] args) {
        int a = 5, b, c;
        b = a + 5;
        c = a * b;
        System.out.println("a: " + a);
        System.out.println("b: " + b);
        System.out.println("c: " + c);
    }
}
```

Basic Mathematical Operators

- `* / % + -` are the mathematical operators
- `* / %` have a higher precedence than `+` or `-`

```
double myVal = a + b % d - c * d / b;
```

- Is the same as:

```
double myVal = (a + (b % d)) -
                ((c * d) / b);
```

Statements & Blocks

- A simple statement is a command terminated by a semi-colon:

  ```
  name = “Fred”;
  ```

- A block is a compound statement enclosed in curly brackets:

  ```
  {
      name1 = “Fred”; name2 = “Bill”;
  }
  ```

- Blocks may contain other blocks

Flow of Control

- Java executes one statement after the other in the order they are written
- Many Java statements are flow control statements:

Alternation: if, if else, switch

Looping: for, while, do while

Escapes: break, continue, return

If – The Conditional Statement

- The if statement evaluates an expression and if that evaluation is true then the specified action is taken

```
if ( x < 10 ) x = 10;
```

- If the value of x is less than 10, make x equal to 10
- It could have been written:

```
if ( x < 10 )
x = 10;
```

- Or, alternatively:

```
if ( x < 10 ) { x = 10};
```

Relational Operators

==	Equal (careful)
!=	Not equal
>=	Greater than or equal
<=	Less than or equal
>	Greater than
<	Less than

If… else

- The if … else statement evaluates an expression and performs one action if that evaluation is true or a different action if it is false.

```
if (x != oldx) {
   System.out.print("x was changed");
}
else {
   System.out.print("x is unchanged");
}
```

Nested if … else

```
if ( myVal > 100 ) {
   if ( remainderOn == true) {
        myVal = mVal % 100;
   }
   else {
    myVal = myVal / 100.0;
   }
}
else
{
   System.out.print("myVal is in range");
}
```

else if

- Useful for choosing between alternatives:

```
if ( n == 1 ) {
  // execute code block #1
}
else if ( j == 2 ) {
  // execute code block #2
}
else {
  // if all previous tests have failed,
  execute code block #3
}
```

A Warning…

WRONG!

```
if( i == j )
    if ( j == k )
     System.out.print(
        "i equals k");
    else
     System.out.print(
     "i is not equal
     to j");
```

CORRECT!

```
if( i == j ) {
   if ( j == k )
   System.out.print(
         "i equals k");
}
else
   System.out.print("i
   is not equal to j");
     // Correct!
```

The switch Statement

```
switch ( n ) {
  case 1:
   // execute code block #1
   break;
  case 2:
   // execute code block #2
   break;
   default:
   // if all previous tests fail then
   //execute code block #4
   break;
}
```

The for loop

- Loop n times

```
for ( i = 0; i < n; i++ ) {
  // this code body will execute n times
  // i from  0 to n-1
}
```

- Nested for:

```
for ( j = 0; j < 10; j++ ) {
  for ( i = 0; i < 20; i++ ){
   // this code body will execute 200 times
  }
}
```

while loops

```
while(response == 1) {
  System.out.print( "ID =" + userID[n]);
  n++;
  response = readInt( "Enter ");
}
```

What is the minimum number of times the loop is executed?

What is the maximum number of times?

do {… } while loops

```
do {
  System.out.print( "ID =" + userID[n] );
  n++;
  response = readInt( "Enter " );
}while (response == 1);
```

What is the minimum number of times the loop is executed?

What is the maximum number of times?

Break

- A break statement causes an exit from the <u>innermost loop</u> containing while, do, for or switch statement.

```
for ( int i = 0; i < maxID, i++ ) {
  if ( userID[i] == targetID ) {
   index = i;
   break;
  }
} // program jumps here after break
```

Continue

- Can only be used with while, do or for.
- The continue statement causes the innermost loop to start the next iteration immediately

```
for ( int i = 0; i < maxID; i++ ) {
   if ( userID[i] == -1 ) continue;
   System.out.print( “UserID ” + i + “ :” +
     userID);
}
```

Arrays

- An array is a list of things of the same type.
- An array has a fixed:
  - name
  - type
  - length
- These must be declared when the array is created.
- Array sizes cannot be changed during the execution of the code.

myArray =	**3**	**6**	**3**	**1**	**6**	**3**	**4**	**1**
	0	1	2	3	4	5	6	7

myArray has room for 8 elements

- the elements are accessed by their index
- in Java, array indices start at 0

Declaring Arrays

```
int myArray[];
```

declares *myArray* to be an array of integers

```
int[] myArray;
```

is also possible (and preferred in Java)

```
myArray = new int[8];
```

sets up 8 integer-sized spaces in memory, labelled *myArray[0]* to *myArray[7]*

```
int myArray[] = new int[8];
```

combines the two statements in one line

Assigning Values

- refer to the array elements by index to store values in them.

```
myArray[0] = 3;
myArray[1] = 6;
myArray[2] = 3;   ...
```

- can create and initialise in one step:

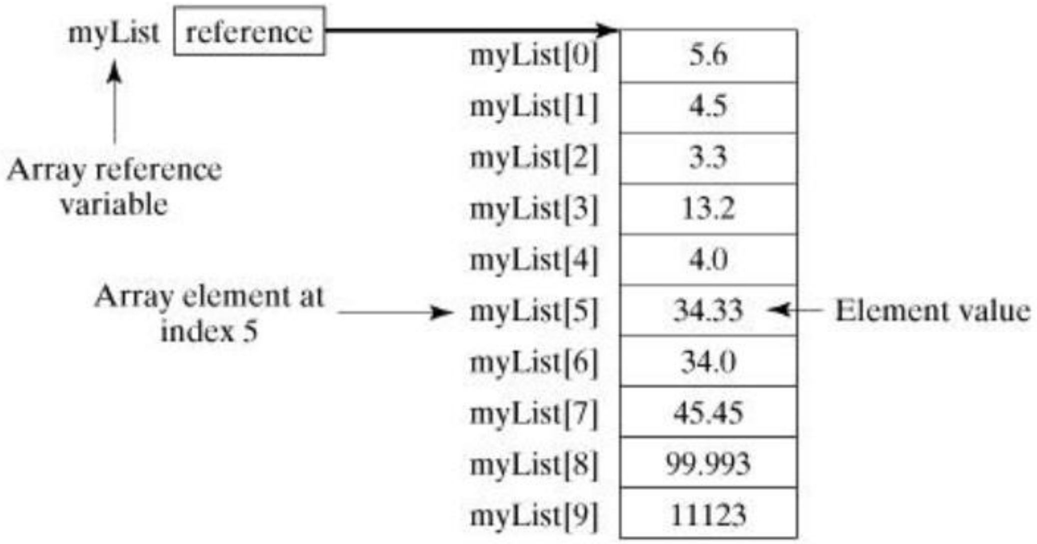
```
int myArray[] = {3, 6, 3, 1, 6, 3, 4, 1};
```

Example

Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList −

```
double[] myList = new double[10];
```

Following picture represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.



Iterating Through Arrays

- *for* loops are useful when dealing with arrays:

```
for (int i=0; i<myArray.length; i++) {
   myArray[i] = getsomevalue();
}
```

Arrays of Objects

- So far we have looked at an array of primitive types.
  - integers
  - could also use doubles, floats, characters…
- Often want to have an array of objects
  - Students, Books, Loans ……
- Need to follow 3 steps.

Declaring the Array

1. Declare the array

    `private Student studentList[];`

    – this declares studentList

2 .Create the array

    `studentList = new Student[10];`

    – this sets up 10 spaces in memory that can hold references to Student objects

3. Create Student objects and add them to the array:

```
studentList[0] =
new Student("Cathy", "Computing");
```

```
int [] arrayOfInts;
int arrayOfInts [];

equivalent
```

```
ChessPiece [][] chessBoard;
chessBoard = new ChessPiece [8][8];
chessBoard[0][0] = new ChessPiece(
"Rook" ); chessBoard[1][0] = new
ChessPiece( "Pawn" );


boolean [][] checkerBoard;
```

Java Methods & Classes

Classes ARE Object Definitions

- OOP - object oriented programming
- code built from objects
- Java these are called ***classes***
- Each class definition is coded in a separate .java file
- Name of the object must match the class/object name

Simple Class and Method

```
Class Fruit{
    int grams;
    int cals_per_gram;

    int total_calories() {
          return(grams*cals_per_gram);
    }
}
```

Methods

- A method is a named sequence of code that can be invoked by other Java code.
- A method takes some parameters, performs some computations and then optionally returns a value (or object).
- Methods can be used as part of an expression statement.

```
public float convertCelsius(float tempC) {
      return( ((tempC * 9.0f) / 5.0f) + 32.0 );
   }
```

Method Signatures

- A method signature specifies:
  - The name of the method.
  - The type and name of each parameter.
  - The type of the value (or object) returned by the method.
  - The checked exceptions thrown by the method.
  - Various method modifiers.
  - *modifiers type name (parameter list) [throws exceptions]*

```
public float convertCelsius (float tCelsius ) {}
public boolean setUserInfo ( int i, int j, String name ) throws
    IndexOutOfBoundsException {}
```

Public/private

- Methods/data may be declared ***public*** or ***private*** meaning they may or may not be accessed by code in other classes …
- Good practice:
  - keep data private
  - keep most methods private
- well-defined interface between classes - helps to eliminate errors

Using objects

- Here, code in one class creates an instance of another class and does something with it …

```
Fruit plum=new Fruit();
int cals;
cals = plum.total_calories();
```

- ***Dot operator*** allows you to access (public) data/methods inside Fruit class

Constructors

```
class Date {
        long time;
        Date( ) {
                time = currentTime( );
        }

        Date( String date ) {
                time = parseDate( date );
        } ...
}

Date now = new Date( );
Date newYear = new Date(”Jan 1, 2019");
```

- The line

  plum = new Fruit();

- invokes a constructor method with which you can set the initial data of an object
- You may choose several different type of constructor with different argument lists

  eg Fruit(), Fruit(a) ...

Overloading

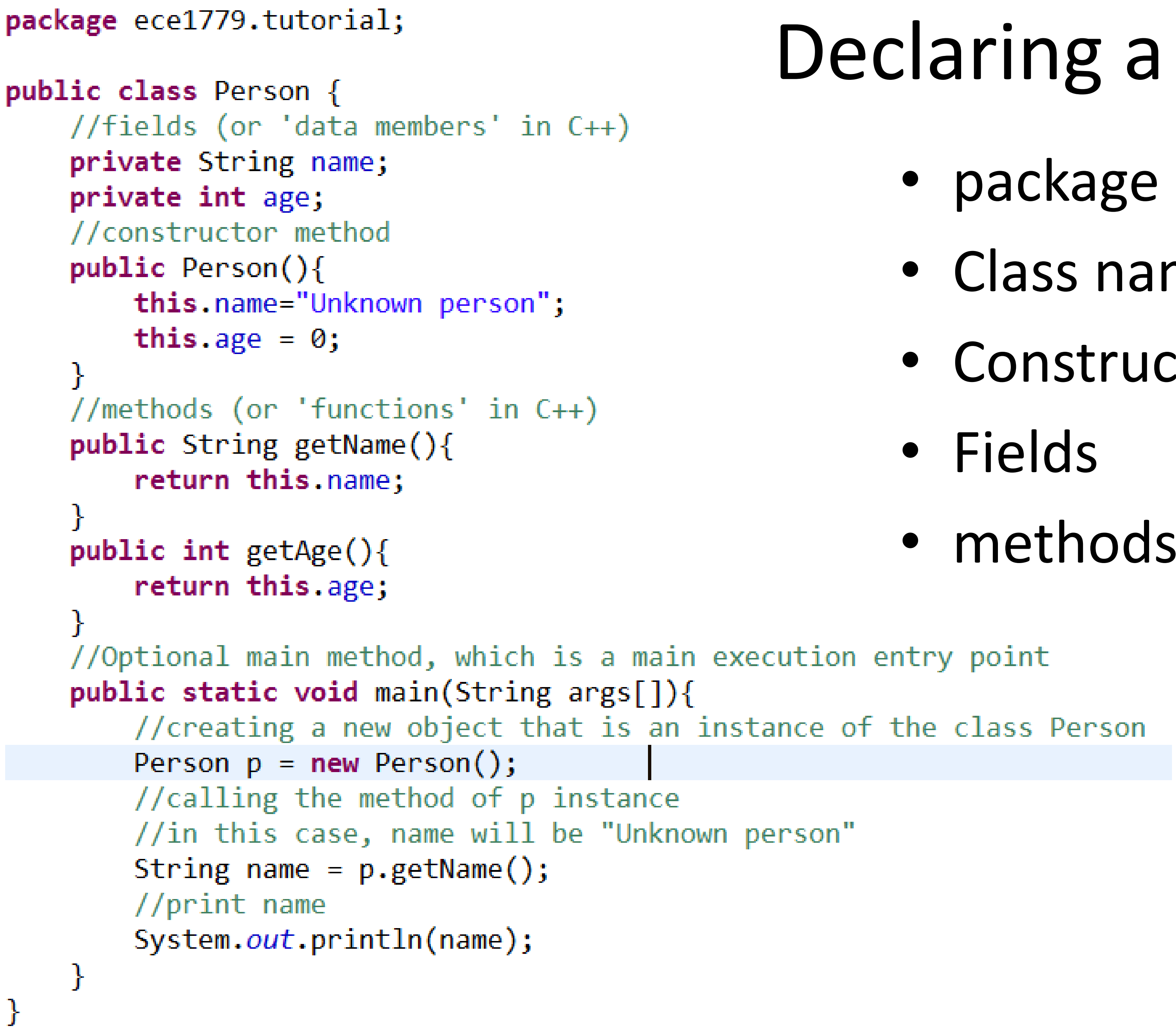
- Can have several versions of a method in class with different types/numbers of arguments

```
Fruit() {grams=50;}
Fruit(a,b) { grams=a; cals_per_gram=b;}
```

- By looking at arguments Java decides which version to use

Declaring a class

- package
- Class name
- Constructor
- Fields
- methods

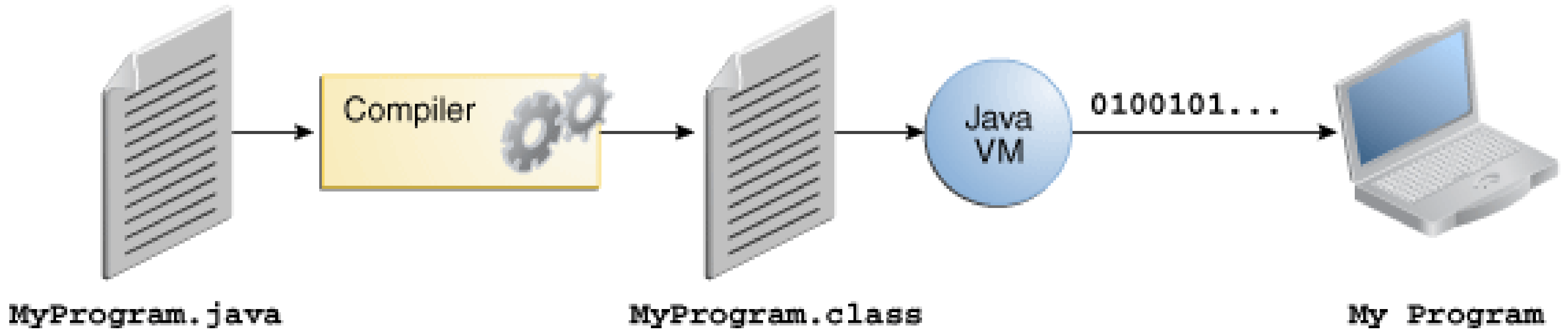


```java
package ece1779.tutorial;

public class Person {
    //fields (or 'data members' in C++)
    private String name;
    private int age;
    //constructor method
    public Person(){
        this.name="Unknown person";
        this.age = 0;
    }
    //methods (or 'functions' in C++)
    public String getName(){
        return this.name;
    }
    public int getAge(){
        return this.age;
    }
    //Optional main method, which is a main execution entry point
    public static void main(String args[]){
        //creating a new object that is an instance of the class Person
        Person p = new Person();
        //calling the method of p instance
        //in this case, name will be "Unknown person"
        String name = p.getName();
        //print name
        System.out.println(name);
    }
}
```

Compiling, Running and Debugging Java Programs

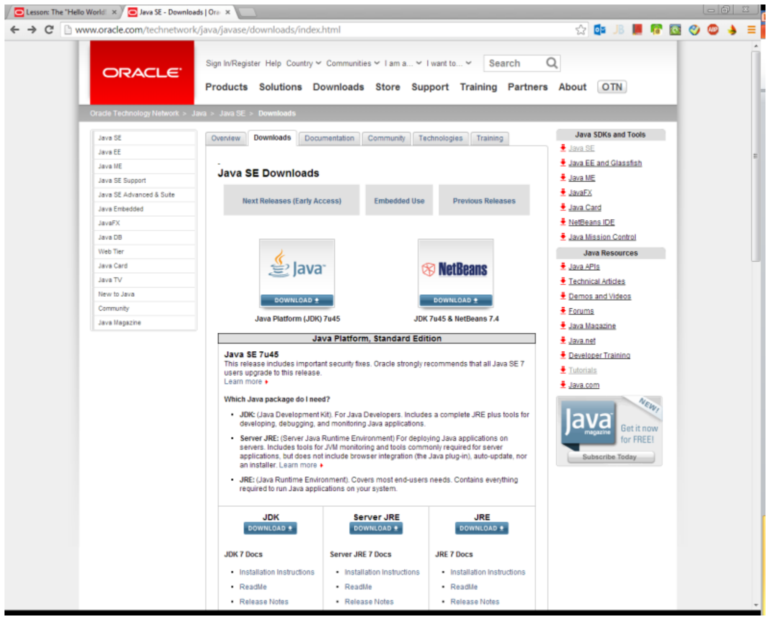
Java Development Process

.java => .class => JVM execution



Installing Java in your machine (1)

- Downloading Java Development Kit (JDK) from Oracle
- Java Runtime Environment (JRE) is usually included in the JDK installation file.



Java SE Downloads

Java Platform, Standard Edition

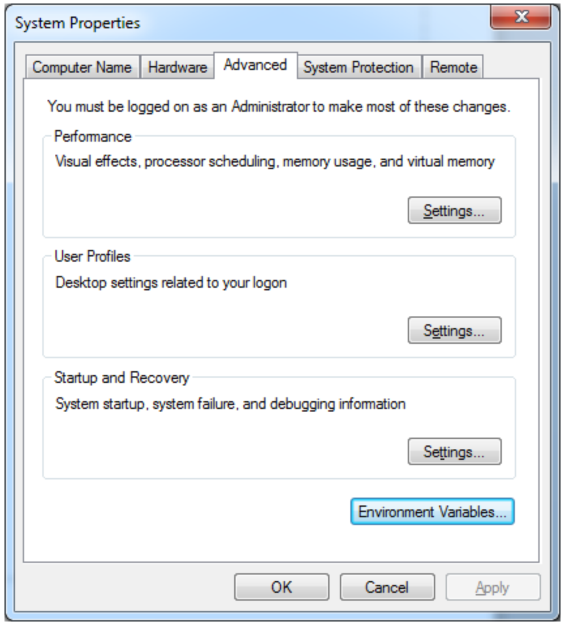
Java SE 7u45
This release includes important security fixes. Oracle strongly recommends that all Java SE 7 users upgrade to this release.
Learn more

Which Java package do I need?

- JDK: (Java Development Kit). For Java Developers. Includes a complete JRE plus tools for developing, debugging, and monitoring Java applications.
- Server JRE: (Server Java Runtime Environment) For deploying Java applications on servers. Includes tools for JVM monitoring and tools commonly required for server applications, but does not include browser integration (the Java plug-in), auto-update, nor an installer. Learn more
- JRE: (Java Runtime Environment). Covers most end-users needs. Contains everything required to run Java applications on your system.

Installing Java in your machine (2)

- Setting JAVA_HOME (Windows):
  - E.g., *C:\Program Files\Java\jdk1.7.0_45*
- Setting **path** and **classpath**



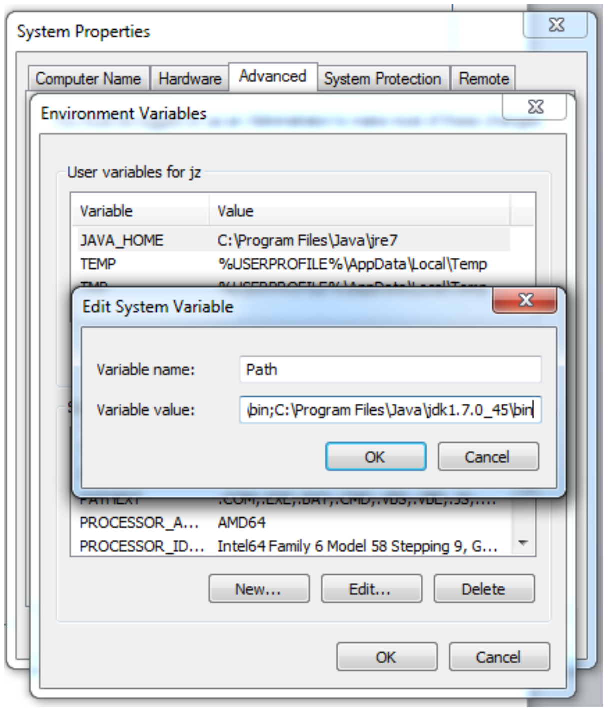
System Properties

You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your logon

Startup and Recovery
System startup, system failure, and debugging information



System Properties

Environment Variables

User variables for jz

Variable	Value
JAVA_HOME	C:\Program Files\Java\jre7
TEMP	%USERPROFILE%\AppData\Local\Temp

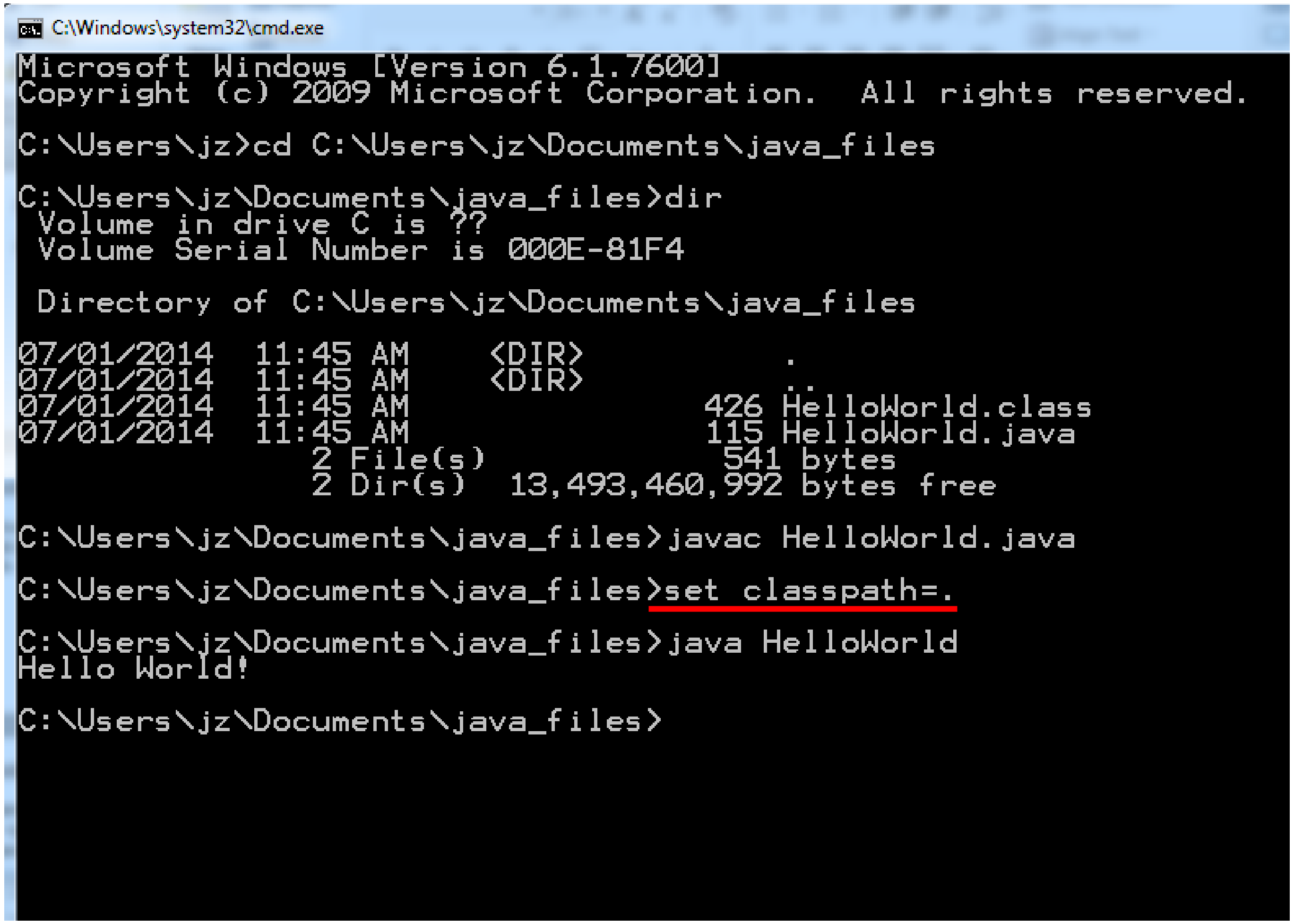
Edit System Variable

Variable name: Path

Variable value: bin;C:\Program Files\Java\jdk1.7.0_45\bin

PROCESSOR_A...	AMD64
PROCESSOR_ID...	Intel64 Family 6 Model 58 Stepping 9, G...

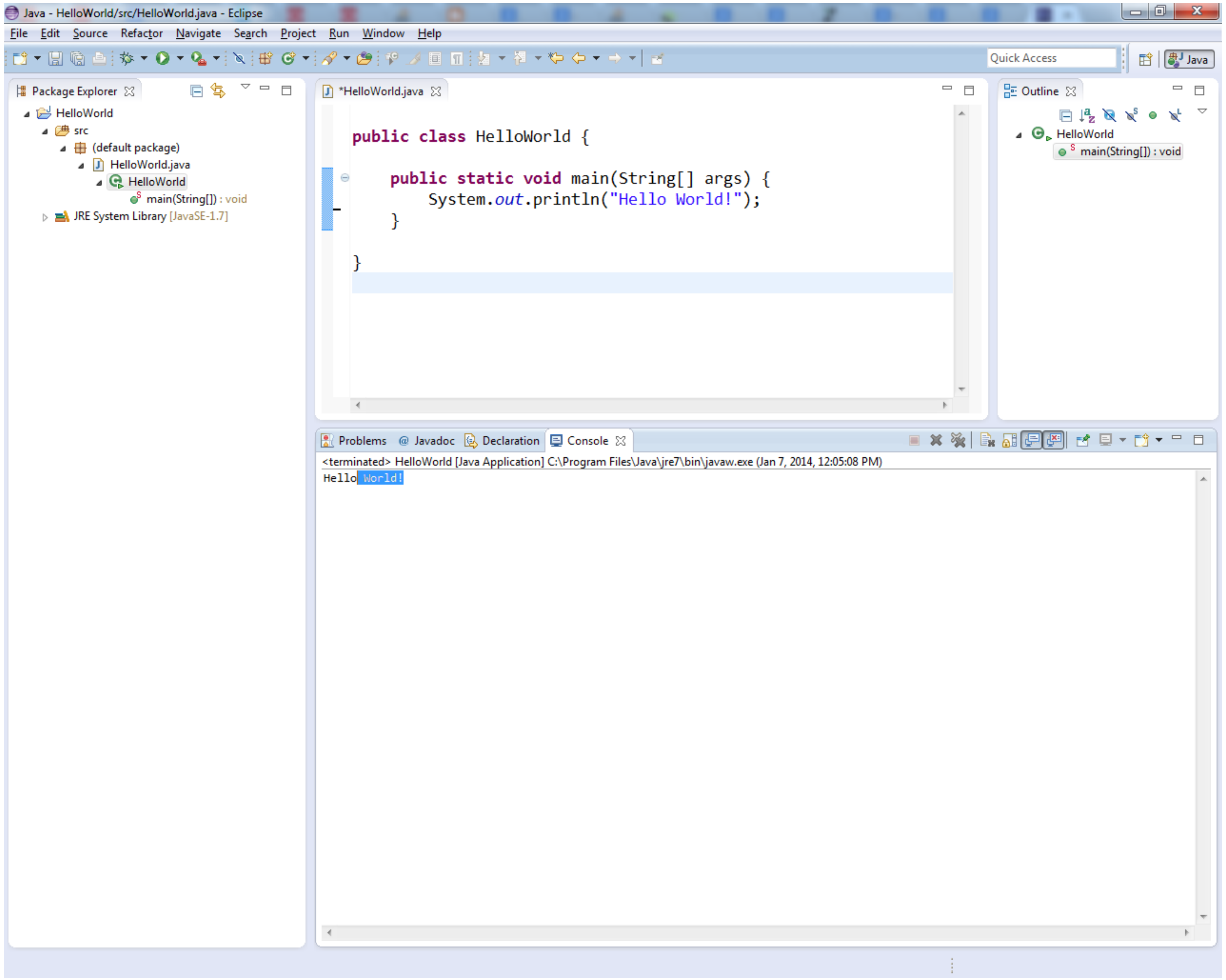
Compile .java File into a .class File (Command Line)



C:\Windows\system32\cmd.exe

```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\jz>cd C:\Users\jz\Documents\java_files

C:\Users\jz\Documents\java_files>dir
 Volume in drive C is ??
 Volume Serial Number is 000E-81F4

 Directory of C:\Users\jz\Documents\java_files

07/01/2014  11:45 AM    <DIR>          .
07/01/2014  11:45 AM    <DIR>          ..
07/01/2014  11:45 AM               426 HelloWorld.class
07/01/2014  11:45 AM               115 HelloWorld.java
               2 File(s)            541 bytes
               2 Dir(s)  13,493,460,992 bytes free

C:\Users\jz\Documents\java_files>javac HelloWorld.java

C:\Users\jz\Documents\java_files>set classpath=.

C:\Users\jz\Documents\java_files>java HelloWorld
Hello World!

C:\Users\jz\Documents\java_files>
```

Running HelloWorld in Eclipse IDE

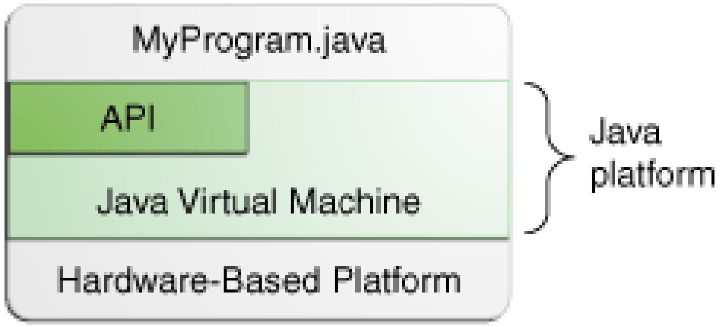


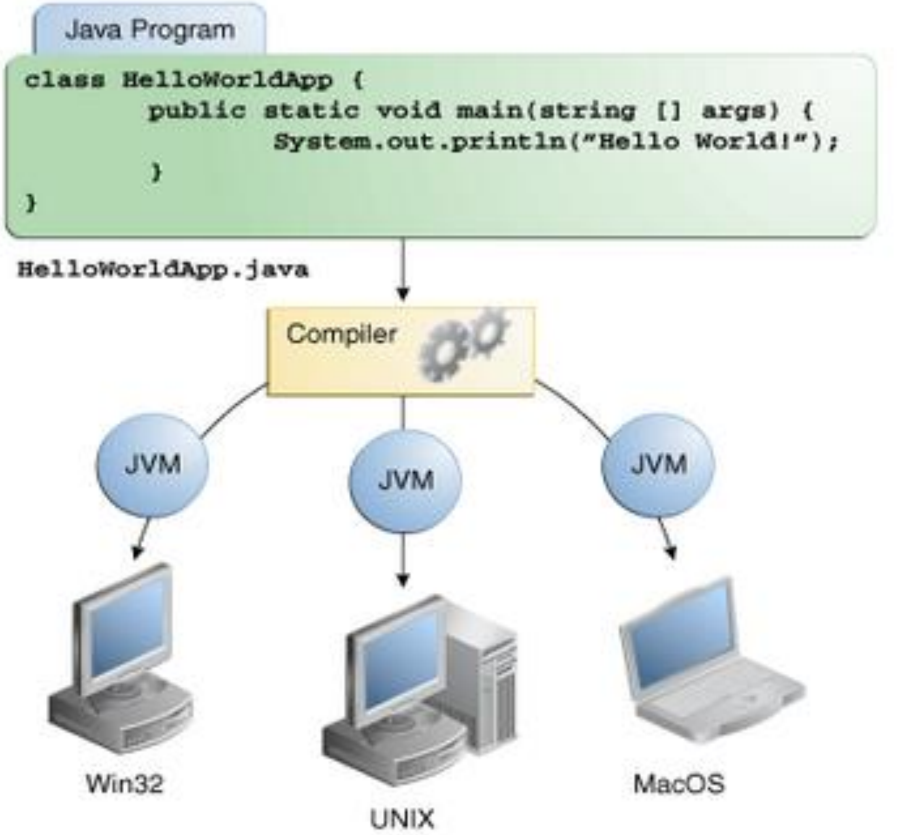
Java - HelloWorld/src/HelloWorld.java - Eclipse

```java
public class HelloWorld {

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

}
```

```
<terminated> HelloWorld [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Jan 7, 2014, 12:05:08 PM)
Hello World!
```

Java platform



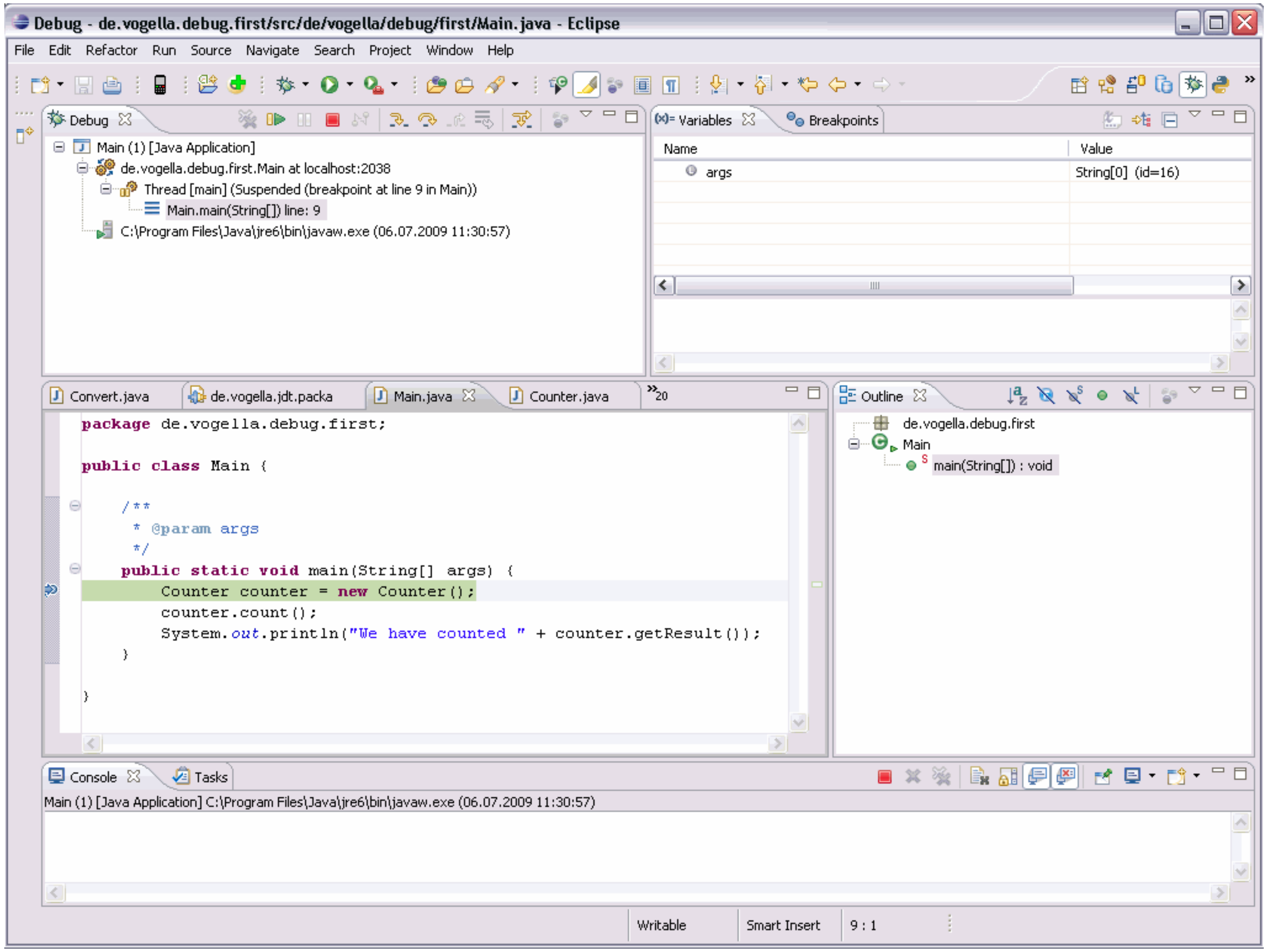


```
class HelloWorldApp {
        public static void main(string [] args) {
                System.out.println("Hello World!");
        }
}
```

Debugging Java in Eclipse (1)

- ***Debugging*** means “run a program interactively while watching the source code and the variables during the execution.” [5]
- Set ***breakpoints*** to stop the program at the middle of execution
- Eclipse has a *Debug Mode*

Debugging Java in Eclipse(2)


Debug - de.vogella.debug.first/src/de/vogella/debug/first/Main.java - Eclipse
File Edit Refactor Run Source Navigate Search Project Window Help
Debug
Variables
Breakpoints
Convert.java
de.vogella.jdt.packa
Main.java
Counter.java
Outline
Console
Tasks
Writable
Smart Insert
9 : 1

```
Main (1) [Java Application]
  de.vogella.debug.first.Main at localhost:2038
    Thread [main] (Suspended (breakpoint at line 9 in Main))
      Main.main(String[]) line: 9
  C:\Program Files\Java\jre6\bin\javaw.exe (06.07.2009 11:30:57)
```

Name	Value
args	String[0] (id=16)

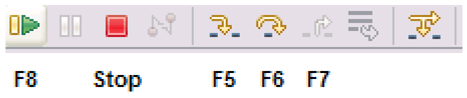
```
package de.vogella.debug.first;

public class Main {

    /**
     * @param args
     */
    public static void main(String[] args) {
        Counter counter = new Counter();
        counter.count();
        System.out.println("We have counted " + counter.getResult());
    }

}
```

```
de.vogella.debug.first
  Main
    main(String[]) : void
```

```
Main (1) [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (06.07.2009 11:30:57)
```

Image courtesy: http://www.vogella.com/tutorials/EclipseDebugging/images/xdebugstart20.gif.pagespeed.ic.SqCELlNeCm.png

Debugging Java in Eclipse(3)



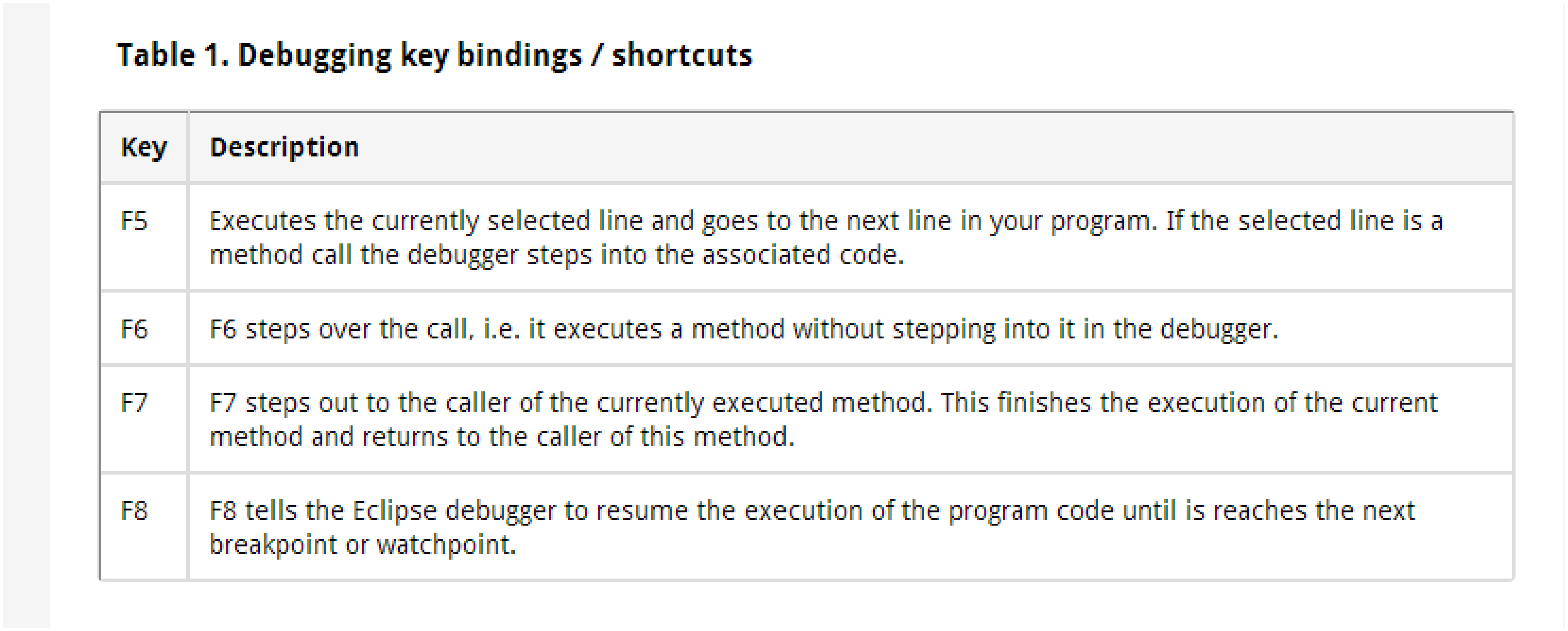


Table 1. Debugging key bindings / shortcuts

Key	Description
F5	Executes the currently selected line and goes to the next line in your program. If the selected line is a method call the debugger steps into the associated code.
F6	F6 steps over the call, i.e. it executes a method without stepping into it in the debugger.
F7	F7 steps out to the caller of the currently executed method. This finishes the execution of the current method and returns to the caller of this method.
F8	F8 tells the Eclipse debugger to resume the execution of the program code until is reaches the next breakpoint or watchpoint.

Table courtesy: http://www.vogella.com/tutorials/EclipseDebugging/article.html