

Second AP\* Edition  
— with GridWorld

# *Java*

# *Methods*

Object-Oriented Programming  
and  
Data Structures

Maria Litvin

Phillips Academy, Andover, Massachusetts

Gary Litvin

Skylight Software, Inc.

Skylight Publishing  
Andover, Massachusetts

Skylight Publishing  
9 Bartlet Street, Suite 70  
Andover, MA 01810

web: <http://www.skylit.com>  
e-mail: [sales@skylit.com](mailto:sales@skylit.com)  
[support@skylit.com](mailto:support@skylit.com)

**Copyright © 2011 by Maria Litvin, Gary Litvin, and  
Skylight Publishing**

This material is provided to you as a supplement to the book *Java Methods*, second AP edition. You may print out one copy for personal use and for face-to-face teaching for each copy of the *Java Methods* book that you own or receive from your school. You are not authorized to publish or distribute this document in any form without our permission. **You are not permitted to post this document on the Internet.** Feel free to create Internet links to this document's URL on our web site from your web pages, provided this document won't be displayed in a frame surrounded by advertisement or material unrelated to teaching AP\* Computer Science or Java. You are not permitted to remove or modify this copyright notice.

Library of Congress Control Number: 2010915303

ISBN 978-0-9824775-7-1

\* AP and Advanced Placement are registered trademarks of The College Board, which was not involved in the production of and does not endorse this book.

The names of commercially available software and products mentioned in this book are used for identification purposes only and may be trademarks or registered trademarks owned by corporations and other commercial entities. Skylight Publishing and the authors have no affiliation with and disclaim any sponsorship or endorsement by any of these product manufacturers or trademark owners.

Oracle, Java, and Java logos are trademarks or registered trademarks of Oracle Corporation and/or its affiliates in the U.S. and other countries.

SCRABBLE® is the registered trademark of HASBRO in the United States and Canada and of J.W. Spear and Sons, PLC, a subsidiary of Mattel, Inc., outside the United States and Canada.

## Appendix B: Common Syntax Error Messages

---

[Exception in thread "main" java.lang.NoClassDefFoundError — wrong name](#)  
[Exception in thread "main" java.lang.NoClassDefFoundError — /java](#)  
[Exception in thread "main" java.lang.NoClassDefFoundError — /class](#)  
[Exception in thread "main" java.lang.NoSuchMethodError: main](#)  
[class is public, should be declared in a file named](#)  
[cannot return a value from method whose result type is void](#)  
[non-static method cannot be referenced from a static context](#)  
[cannot find symbol -- class](#)  
[cannot find symbol -- method](#)  
[cannot find symbol -- variable](#)  
['}' expected](#)  
['class' or 'interface' expected](#)  
[illegal character](#)  
[<identifier> expected](#)  
['\(' or '\[' expected](#)  
[variable might not have been initialized](#)  
[unclosed string literal](#)  
[missing return statement](#)  
[':' expected](#)  
[incompatible types](#)  
['\]' expected](#)  
[array required, but java.lang.String found](#)  
[possible loss of precision](#)  
['class' expected](#)  
[attempting to assign weaker access privileges](#)  
[call to super must be first statement in constructor](#)  
[invalid method declaration; return type required](#)  
[is not abstract and does not override abstract method](#)

```
C:\mywork>java hello
```

```
Exception in thread "main" java.lang.NoClassDefFoundError: hello (wrong
name: Hello)
```

This run-time error (exception) happens when you mistype a lower case letter for upper case. Normally a class name (e.g., Hello) starts with an upper case letter and the file name should be the same. Under Windows, the command

```
javac hello.java
```

will compile the file `Hello.java`, but when you try to run it, as above, it reports an exception. It should be:

```
C:\mywork>java Hello
```

❖ ❖ ❖

```
C:\mywork>java Hello.java
```

```
Exception in thread "main" java.lang.NoClassDefFoundError: Hello/java
```

or

```
C:\mywork>java Hello.class
```

```
Exception in thread "main" java.lang.NoClassDefFoundError: Hello/class
```

The command to run the Java interpreter should use the class name but should not include any extension, neither `.java` nor `.class`. An extension in the file name confuses the interpreter about the location of the source code file (the extension is interpreted as a subfolder).

❖ ❖ ❖

```
C:\mywork>java Hello
```

```
Exception in thread "main" java.lang.NoSuchMethodError: main
```

This exception may be reported when the main method is missing or its signature is incorrect. The correct signature is

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

Possible mistakes:

```
private static void main (String[] args)
public void main (String[] args)
public static int main (String[] args)
public static void main (String args)
```



```
C:\mywork>javac Test.java
class Hello is public, should be declared in a file named Hello.java
public class Hello
    ^
```

A mismatch between the file name (`Test`) and the class name (`Hello`) — they must be the same.



```
cannot return a value from method whose result type is void
    return 0;
        ^
```

In Java, `main` is `void`, so `return` is not necessary and you can't use `return 0` or `return "Hello, World!"` in it.



```
non-static method printMsg() cannot be referenced from a static context
    printMsg();
        ^
```

The method `printMsg` is called directly from `main`, without any dot-prefix, and the keyword `static` is missing in the `printMsg` header:

```
public void printMsg(String msg)
```

should be:

```
public static void printMsg(String msg)
```

Since `main` is a static method and it calls `printMsg` with no "something-dot" prefix, `printMsg` is assumed to be another static method of the same class. Another way of handling this is to create an object of the `Hello` class in `main` and call that object's `printMsg`:

```
Hello test = new Hello();
test.printMsg();
```



```
cannot find symbol
symbol   : class Scanner
location: class Hello
    Scanner kboard = new Scanner(System.in);
        ^
```

The compiler automatically finds classes, either source or compiled, used by your class `Hello`, as long as they are located in the same folder as `Hello.java`. Library classes must be properly “imported” in your class. For example:

```
import java.util.Scanner;
```

at the top of your program.

Another possible reason for this error message is incorrect or misspelled primitive data type name. For example:

```
private bool match(String word, String pattern)
```

gives

```
cannot find symbol
symbol   : class bool
location: class Hello
    private bool match(String word, String pattern)
            ^
```

It should be `boolean`.



```
cannot find symbol
symbol   : method printMessage()
location: class Hello
    test.printMessage();
        ^
```

This error occurs when a method is called incorrectly: either its name is misspelled or upper-lower case is misplaced. Here the method name should be `printMsg`.

The same error is reported when you call a method for a wrong type of object. For example:

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

instead of

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

You will get:

```
cannot find symbol
symbol   : method println(java.lang.String)
location: class java.lang.System
    System.println("Hello, World!");
           ^
```

Another example:

```
cannot find symbol
symbol   : method println(java.lang.String,java.lang.String)
location: class java.io.PrintStream
    System.out.println("Hello, World!", name);
                   ^
```

Here a comma is used instead of a + in the `println` call. This makes it a call with two parameters instead of one and `System.out` does not have a `println` method that takes two `String` parameters.



```
cannot find symbol
symbol   : variable name
location: class Hello
    name = kboard.next();
           ^
```

A very common error “cannot find symbol” may result from an undeclared variable or a misspelled local variable or field name. Here it should be

```
String name = kboard.next();
```

or name should be declared earlier.



```
'}' expected
  }
  ^
```

An extra opening brace or a missing closing brace may produce several errors, including

```
illegal start of expression
```

```
';' expected
```

and finally

```
'}' expected
```

or

```
reached end of file while parsing
}
^
```

❖ ❖ ❖

```
class, interface, or enum expected
}
^
```

This error often results from an extra closing brace (or a missing opening brace) or a method declared outside a class.

❖ ❖ ❖

```
illegal character: \8220
  System.out.println("Hello, World!");
                        ^
```

“Smart quote” characters accidentally left in the source file by a word processor instead of straight single or double quotes may cause this error. The same error is reported when the source file contains any non-ASCII character in the code (outside comments).

❖ ❖ ❖

```
<identifier> expected
  public name;
                ^
```

“<identifier> expected” is a rather common error message. Here name is a variable, but the compiler thinks it is a class name. Actually, it is the data type designation that is missing. It should be:

```
private String name;
```

The same happens here:



```
private nRows, nCols;
```

It gives an error:

```
<identifier> expected
static nRows, nCols;
      ^
```

thinking that `nRows` is a data type. Same here:

```
public static void printMsg(msg)
{
    ...
}
```

— a missing type designator for the parameter in a method's header. It produces:

```
<identifier> expected
public static void printMsg(msg)
                        ^
```

It should be:

```
public static void printMsg(String msg)
```

❖ ❖ ❖

```
(' or '[' expected
Hello test = new Hello;
                ^
```

This error is reported when a parenthesis or square bracket is missing. In the above example it should be:

```
Hello test = new Hello();
```

❖ ❖ ❖

```
variable kboard might not have been initialized
String name = kboard.next();
                ^
```

This error happens if you use a local variable before initializing it.

```
Scanner kboard;
```

declares the variable `kboard` but you also need to initialize it before you use it:

```
Scanner kboard = new Scanner(System.in);
or
Scanner kboard;
...
kboard = new Scanner(System.in);
```

❖ ❖ ❖

```
' )' expected
System.out.print(Enter your name: ");
                ^
```

```
unclosed string literal
System.out.print(Enter your name: ");
                ^
```

2 errors

A missing opening double quote in a literal string produces these two errors.

❖ ❖ ❖

```
missing return statement
}
^
```

A method, other than `void`, must return a value.

❖ ❖ ❖

```
 ';' expected
System.out.println("Hello, World!" + name)
                                     ^
```

A few compiler error messages are actually self-explanatory!

❖ ❖ ❖

```
incompatible types
found   : int
required: boolean
    if (i = 0)
        ^
```

It is supposed to be

```
if (i == 0)
```

Single `=` is assignment operator, which returns an `int` value (assuming `i` is an `int`). The `if` statement, on the other hand, expects a `boolean`. Similarly,

```
public boolean isInRange(int i)
{
    return i = 0 || i > 100;
}
```

gives:

```
operator || cannot be applied to int,boolean
    return i = 0 || i > 100;
                   ^
incompatible types
found   : int
required: boolean
    return i = 0 || i > 100;
                   ^
2 errors
```

Another situation with “incompatible types” is when a literal string is used in place of a char constant or vice-versa. For example:

```
incompatible types
found   : java.lang.String
required: char
    char letter = "A";
                   ^
```

Should be:

```
char letter = 'A';
```

❖ ❖ ❖

```
'[' expected
    int[] counts = new int(10);
                   ^
```

An array should be created using brackets, not parentheses:

```
int[] counts = new int[10];
```

❖ ❖ ❖

```
array required, but java.lang.String found
    char letter = str[k];
                   ^
```

Use `str.charAt(k)` method, not `[k]` with strings.

❖ ❖ ❖

```
possible loss of precision
found   : double
required: int
   int x = 2.5;
         ^
```

This happens when a double value is assigned to an int variable.



```
'.' expected
   double y = double(x);
             ^

not a statement
   double y = double(x);
                   ^

';' expected
   double y = double(x);
                   ^

3 errors
```

Incorrect syntax in the cast operator causes these errors. Should be:

```
double y = (double)x;
```



```
actionPerformed(java.awt.event.ActionEvent) in Hello cannot implement
actionPerformed(java.awt.event.ActionEvent) in
java.awt.event.ActionListener; attempting to assign weaker access
privileges; was public
   void actionPerformed(ActionEvent e)
         ^
```

This error is reported when the keyword `public` is missing in the `actionPerformed` method's header:

```
void actionPerformed(ActionEvent e)
```

Should be:

```
public void actionPerformed(ActionEvent e)
```



```
call to super must be first statement in constructor
  super("Hello");
  ^
```

This error is reported when the call `super` is not the first statement in a constructor or if it is mistakenly placed in a method. In particular, this happens when you accidentally put `void` in a constructor's header.



```
invalid method declaration; return type required
public hello() // Constructor
  ^
```

This error is reported when a constructor's name is misspelled or is different from the name of the class. The compiler then thinks it is a method with a missing return type. It can also happen if indeed a return type is missing in a method header.



```
Hello is not abstract and does not override abstract method
compareTo(Hello) in java.lang.Comparable
public class Hello implements Comparable<Hello>
  ^
```

This error is reported when a class claims to implement an interface (in this case `Comparable`) but does not supply all the necessary methods or misspells a method name, or has a wrong number or types of parameters in one of the interface methods.