JAVA – BASIC SYNTAX

BASIC TERMS

When we consider a Java program it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what do class, object, methods and instance variables mean. In more deep it will be explained later.

OBJECT

Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors - wagging, barking, eating. An object is an instance of a class.

CLASS

A class can be defined as a template/ blue print that describes the behaviors/states that object of its type support.

METHODS

A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.

INSTANCE VARIABLES

Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

BASIC SYNTAX

CASE SENSITIVITY

Java is case sensitive, which means that identifier **Hello** and **hello** would have different meaning in Java.

CLASS NAMES

By convence, for all class names the first letter should be in **Upper** Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.

Example: `class MyFirstJavaClass`

METHOD NAMES

All method names should by convence start with a **Lower** Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.

Example: `public void myMethodName()`

PROGRAM FILE NAME

Name of the program file should exactly match the class name.

When saving the file, you should save it using the class name (Remember: Java is case sensitive!) and append '**.java**' to the end of the name. If the file name and the class name do not match, your program will not compile!

Example : assume '`MyFirstJavaProgram`' is the class name. Then the file should be saved as '`MyFirstJavaProgram.java`'

public static void main(string args[])

Java program processing starts from the `main()` method which is a **mandatory part of every Java program**.

JAVA IDENTIFIERS

All Java components require names. Names used for classes, variables and methods are called identifiers.

In Java, there are several points to remember about identifiers. They are as follows:

A..Z, a..z, $ and _

All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).

After the first character identifiers can have any combination of characters.

A key word cannot be used as an identifier.

Most importantly: **identifiers are case sensitive**.

EXAMPLES OF LEGAL IDENTIFIERS

age, $salary, _value, __1_value

EXAMPLES OF ILLEGAL IDENTIFIERS

123abc, -salary

JAVA MODIFIERS

Modifiers are used to change, modify interpretation of identifiers. Modifiers are often used to refine their meaning.

We will be looking into more details about modifiers in the further section.

Like other languages, it is possible to modify classes, methods, etc., by using modifiers. There are two categories of modifiers:

ACCESS MODIFIERS

default, public , protected, private

NON-ACCESS MODIFIERS

final, abstract, strictfp

JAVA VARIABLES

We would see following type of variables in Java:

- Local Variables
- Class Variables (Static Variables)
- Instance Variables (Non-static variables)

JAVA ARRAYS

Arrays are objects that store multiple variables of the same type. However, an array itself is an object on the heap. We will look into how to declare, construct and initialize in the upcoming chapters.

JAVA ENUMS

Enums were introduced in java 5.0. Enums restrict a variable to have one of only a few predefined values. The values in this enumerated list are called enums.

With the use of enums it is possible to reduce the number of bugs in your code.

EXAMPLE

For example, if we consider an application for a fresh juice shop, it would be possible to restrict the glass size to **small**, **medium** and **large**. This would make sure that it would not allow anyone to order any size other than the small, medium or large.

```
class FreshJuice {

   enum FreshJuiceSize{ SMALL, MEDIUM, LARGE }
   FreshJuiceSize size;
}

public class FreshJuiceTest {

   public static void main(String args[]){
      FreshJuice juice = new FreshJuice();
      juice.size = FreshJuice. FreshJuiceSize.MEDIUM ;
      System.out.println("Size: " + juice.size);
   }
}
```

Above example will produce the following result:

```
Size: MEDIUM
```

NOTE

Enums can be declared as their own or inside a class. Methods, variables, constructors can be defined inside enums as well.

JAVA KEYWORDS

The following list shows the reserved words in Java. These reserved words may not be used as constant or variable or any other identifier names.

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

COMMENTS IN JAVA

Java supports single-line and multi-line comments very similar to c and c++. All characters available inside any comment are ignored by Java compiler.

```
public class MyFirstJavaProgram{

   /* This is my first java program.
    * This will print 'Hello World' as the output
    * This is an example of multi-line comments.
    */
```

```
    public static void main(String []args){
       // This is an example of single line comment
       /* This is also an example of single line comment. */
       System.out.println("Hello World");
    }
}
```

DOCUMENTARY AND NON-DOCUMENTARY COMMENTS

In Java, there are two types of comments.

Standard comments mean comments, similar to other programming languages.

The basic structure of writing document comments is to embed them inside /** ... */. The Javadoc is written next to the items without any separating newline. The class declaration usually contains:

```
/**
 * @author      Firstname Lastname <address @ example.com>
 * @version     1.6          (current version number of program)
 * @since       2010-03-31   (the version of the package this
                              class was first added to)
 */
public class Test {
    // class body
}
```

USING BLANK LINES

A line containing only whitespace, possibly with a comment, is known as a blank line, and Java totally ignores it.

INHERITANCE

In Java, classes can be derived from classes. Basically if you need to create a new class and here is already a class that has some of the code you require, then it is possible to derive your new class from the already existing code.

This concept allows you to reuse the fields and methods of the existing class without having to rewrite the code in a new class. In this scenario the existing class is called the superclass and the derived class is called the subclass.

INTERFACES

In Java language, an interface can be defined as a contract between objects on how to communicate with each other. Interfaces play a vital role when it comes to the concept of inheritance.

An interface defines the methods, a deriving class(subclass) should use. But the implementation of the methods is totally up to the subclass.

EXCERCISES

FRESHJUICETEST

Create and run program with class FreshJuice (see above).

DOCUMENTARY COMMENTS

Create documentary comments for your program. You may change documentary comments as follows:

```
/**
 *
 * @author      Josef Kokes <josef.kokes@fs.cvut.cz>
 * @version     1.6          (current version number of program)
 * @since       2013-10-01   (the version of my program)
 *
 */
```

After you save your changes, try Run → Generate JavaDoc to generate documentation.

Surprisingly, there appears only „args“ information, nothing more. See Results to discover, why.

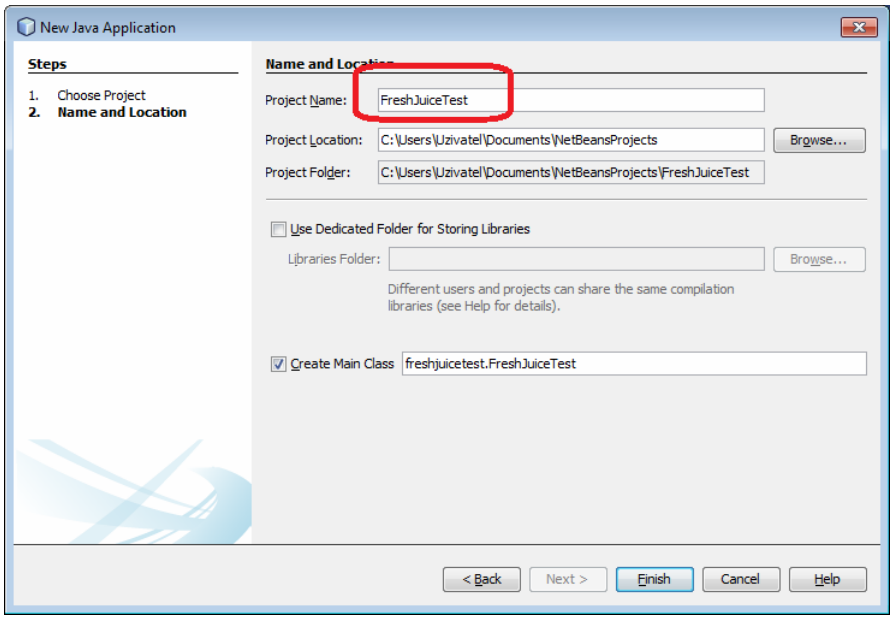
RESULTS

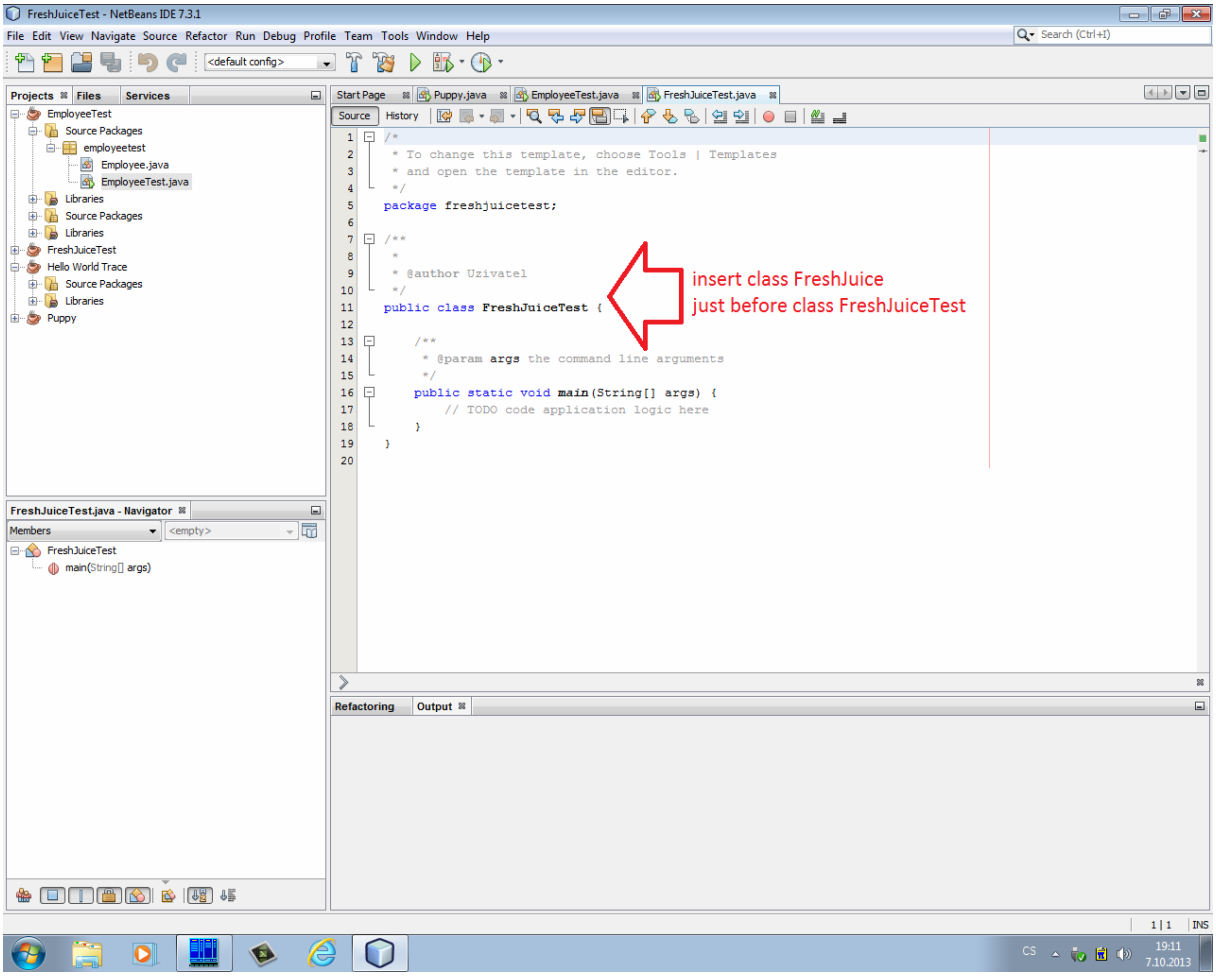
FRESHJUICETEST

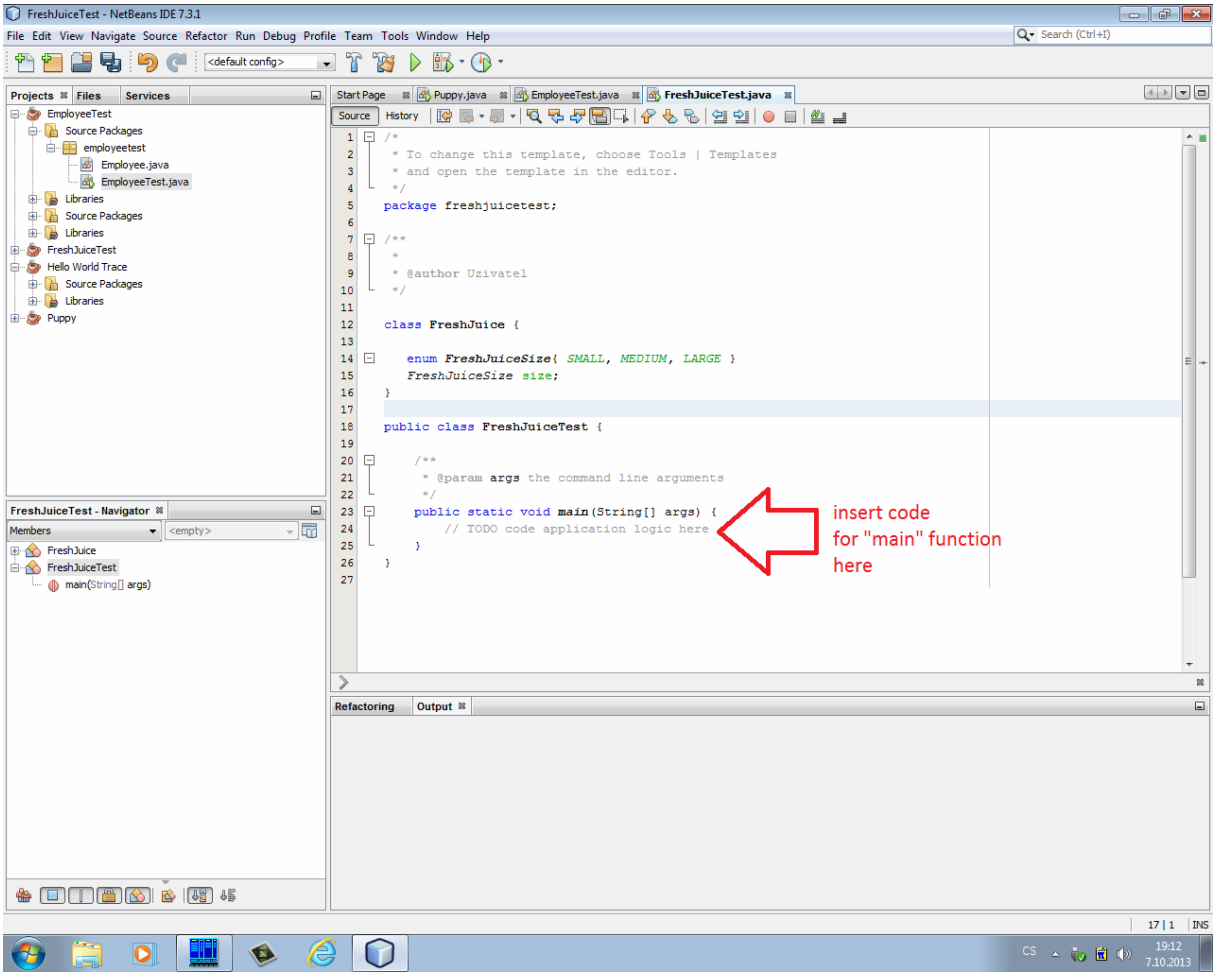
You must surround your class FreshJuice by main procedure and other stuff. Notice that there are **two** distinct classes, `class FreshJuice` (see above) and a general, public `class FreshJuiceTest`.

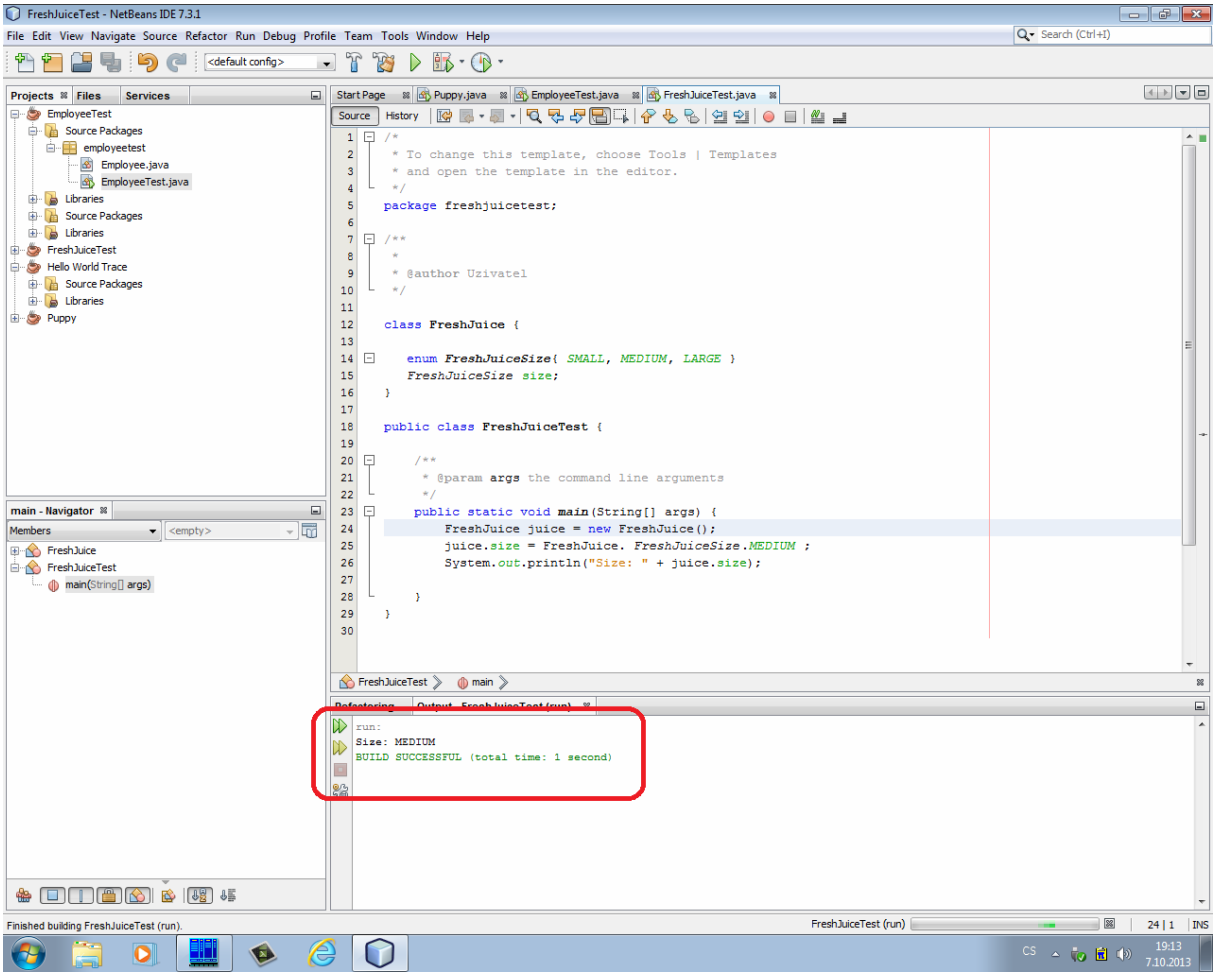
Class `FreshJuice` contains the `enum` type.

Class `FreshJuiceTest` encapsulates both class `FreshJuice` and ist function `main`.




FreshJuiceTest - NetBeans IDE 7.3.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Projects Files Services
EmployeeTest
Source Packages
employeetest
Employee.java
EmployeeTest.java
Libraries
Source Packages
Libraries
FreshJuiceTest
Hello World Trace
Source Packages
Libraries
Puppy
FreshJuiceTest.java - Navigator
Members
<empty>
FreshJuiceTest
main(String[] args)
Start Page
Puppy.java
EmployeeTest.java
FreshJuiceTest.java
Source
History
1 /*
2 * To change this template, choose Tools | Templates
3 * and open the template in the editor.
4 */
5 package freshjuicetest;
6
7 /**
8 *
9 * @author Uzivatel
10 */
11 public class FreshJuiceTest {
12
13 /**
14 * @param args the command line arguments
15 */
16 public static void main(String[] args) {
17 // TODO code application logic here
18 }
19 }
20
insert class FreshJuice
just before class FreshJuiceTest
Refactoring
Output
1 | 1
INS
CS
19:11
7.10.2013

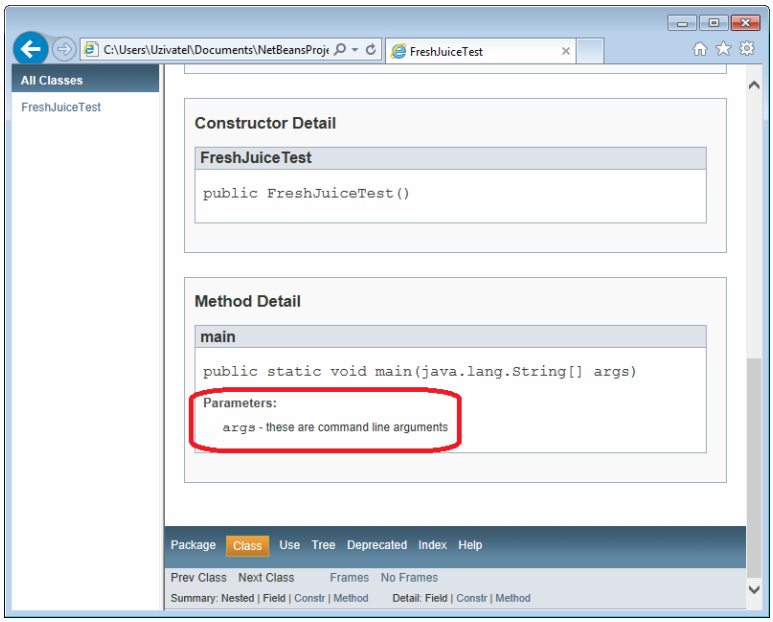

FreshJuiceTest - NetBeans IDE 7.3.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Projects Files Services
EmployeeTest
Source Packages
employeetest
Employee.java
EmployeeTest.java
Libraries
Source Packages
Libraries
FreshJuiceTest
Hello World Trace
Source Packages
Libraries
Puppy
FreshJuiceTest - Navigator
Members
<empty>
FreshJuice
FreshJuiceTest
main(String[] args)
Start Page
Puppy.java
EmployeeTest.java
FreshJuiceTest.java
Source
History
1 /*
2 * To change this template, choose Tools | Templates
3 * and open the template in the editor.
4 */
5 package freshjuicetest;
6
7 /**
8 *
9 * @author Uzivatel
10 */
11
12 class FreshJuice {
13
14 enum FreshJuiceSize{ SMALL, MEDIUM, LARGE }
15 FreshJuiceSize size;
16 }
17
18 public class FreshJuiceTest {
19
20 /**
21 * @param args the command line arguments
22 */
23 public static void main(String[] args) {
24 // TODO code application logic here
25 }
26 }
27
insert code
for "main" function
here
Refactoring
Output
17 | 1
INS
CS
19:12
7.10.2013



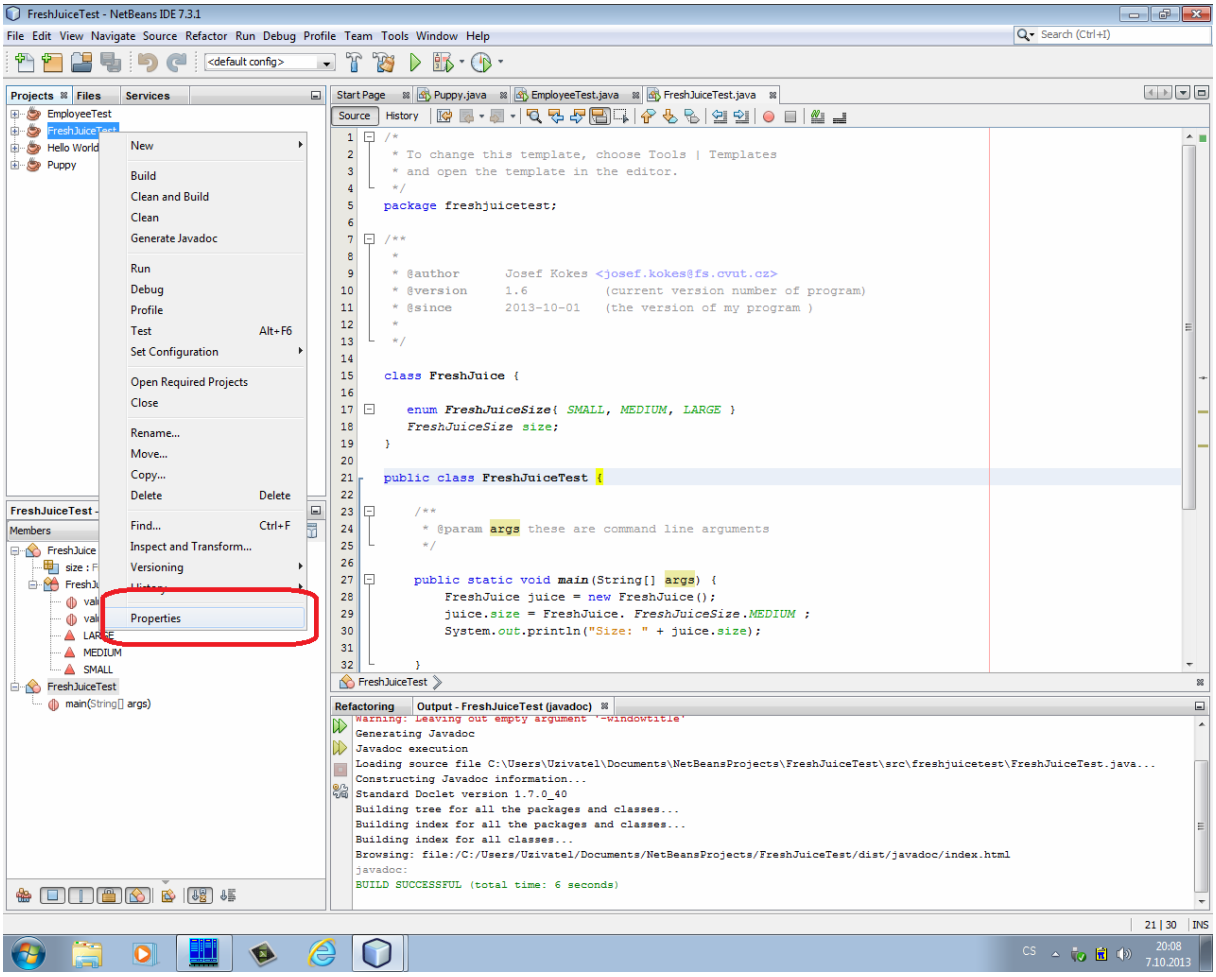
JAVADOC

After you have saved your changes, try Run → Generate JavaDoc to generate documentation, surprisingly, there appears only „args“ information, nothing more:

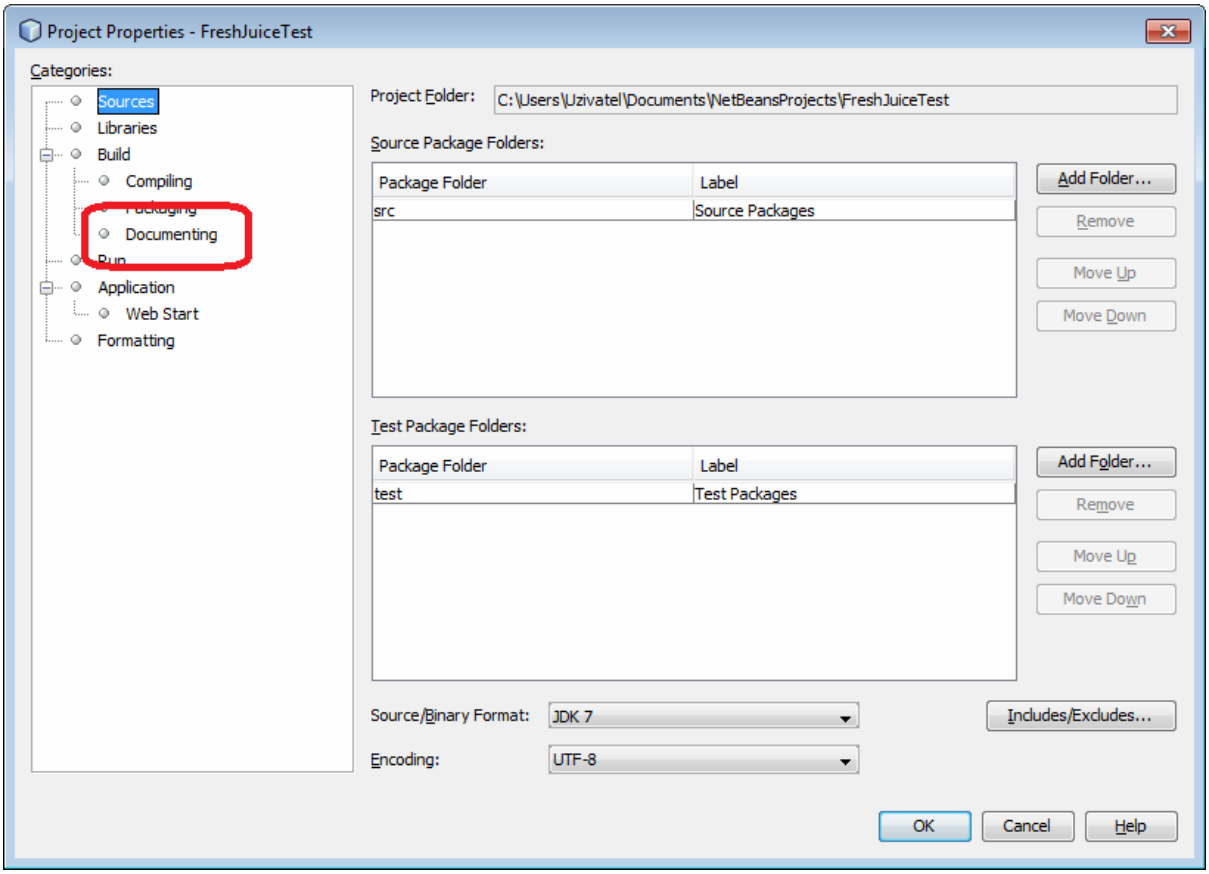
The reason is, that we did not set properties for JavaDoc.



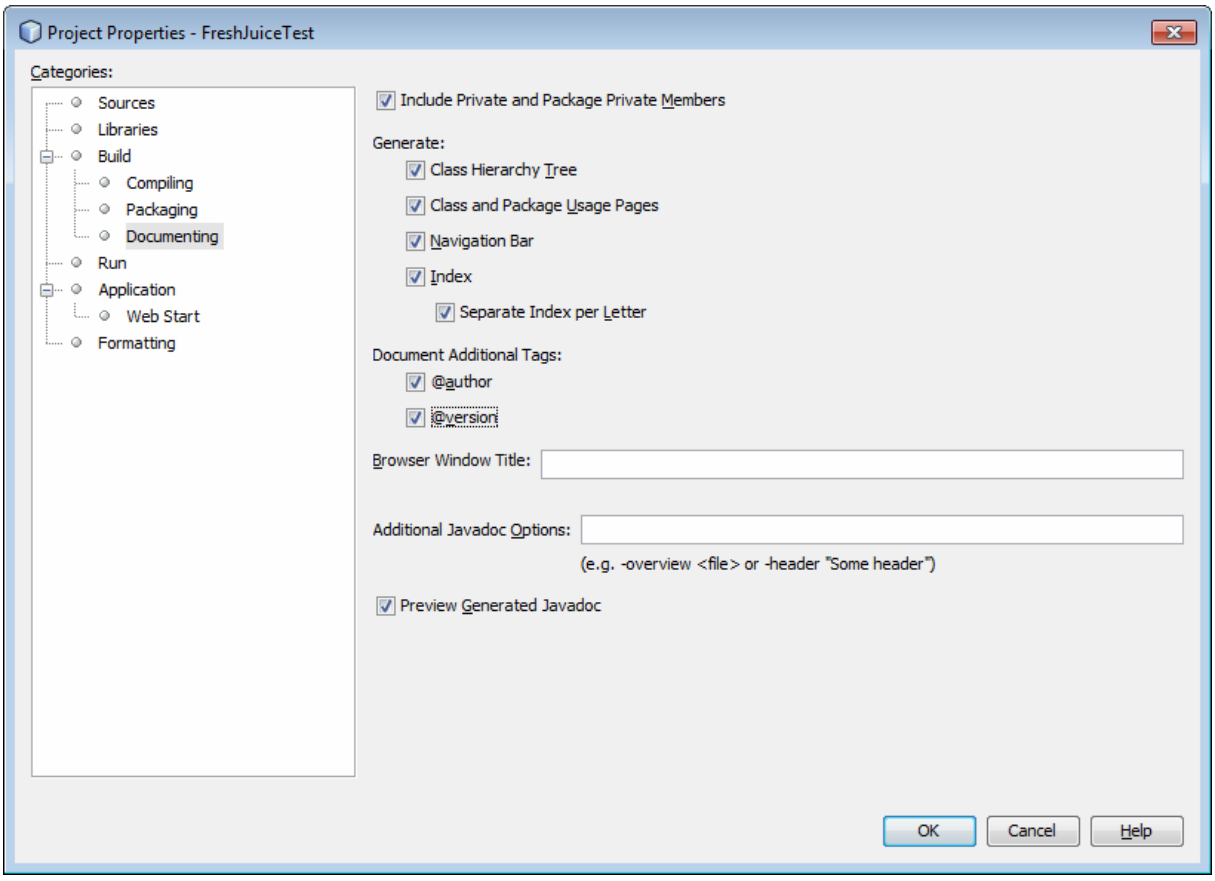
Go to Projects window, right click FreshJuiceTest and click Properties.



Then, expand Documenting



and check everything:



Now, try to generate documentation again. Everything works OK: