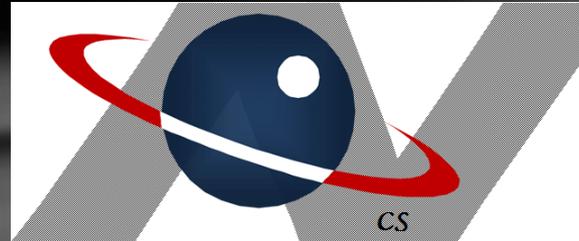


O(N) CS LESSONS

Lesson 5C – MyClass Methods



By John B. Owen

All rights reserved

©2011, revised 2014



Table of Contents



- [Objectives](#)
- [Defining your own class](#)
- [Defining and calling a static method](#)
- [Method structure](#)
- [String return method with parameter](#)
- [Integer return method with parameter](#)
- [JavaDocs](#)
- [Lesson Summary / Labs](#)
- [Contact Information for supplementary materials](#)

Objectives

Students will:

- Develop their own class of utility methods called MyClass, writing stand-alone “static” methods, similar to those found in the Math class, but dealing with a variety of tasks.
- Learn to create their own API using the Javadoc utility, making a help file for their customized class.



CodingBat.com

- The AND, OR and NOT boolean operators are briefly introduced at the end of the lesson in some special labs using a website called codingbat.com.



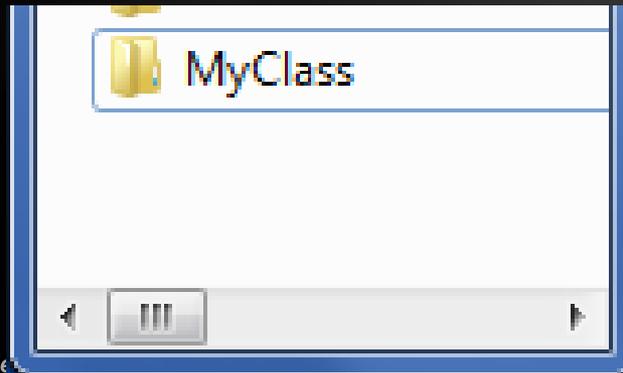
Defining your own class

- Every program in Java is a class.
- For most of what you have done so far, the class has only contained one method, the main method, in which all of your work has been done, and which is required to run most programs.
- Now you will add more methods to your class that can do various tasks.



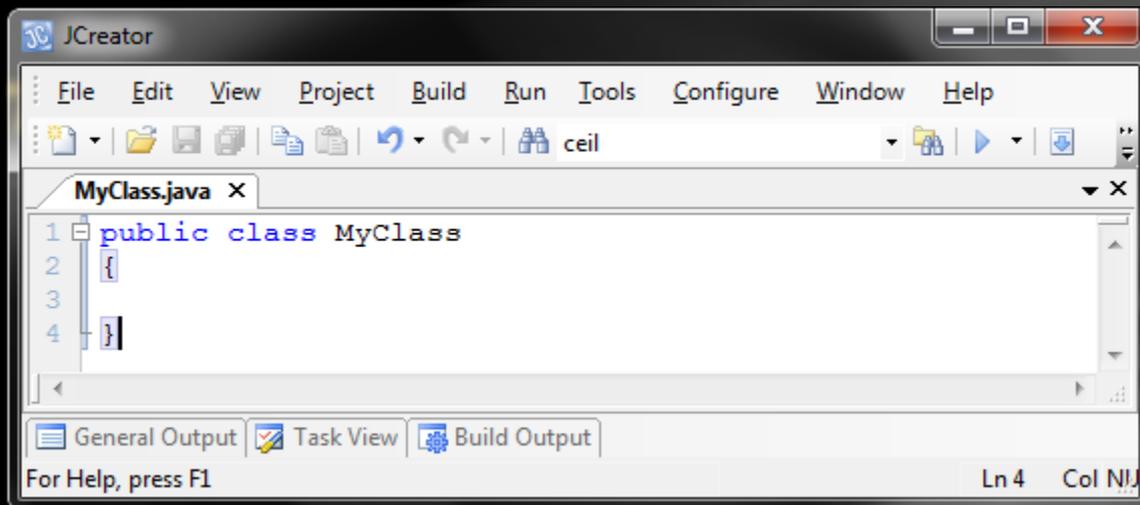
Create a separate folder

- It is good practice to create a separate folder for each lesson you do, although not essential.
- Make one called MyClass for all the work in this lesson, and save all of your Java files to this folder.



The class skeleton

- We'll start with the basic skeleton of the class, which essentially contains nothing, as you can see.
- *Be sure to save this class file in the MyClass folder you just created.*



The screenshot shows the JCreator IDE interface. The main window displays the code for `MyClass.java`. The code is a basic class skeleton:

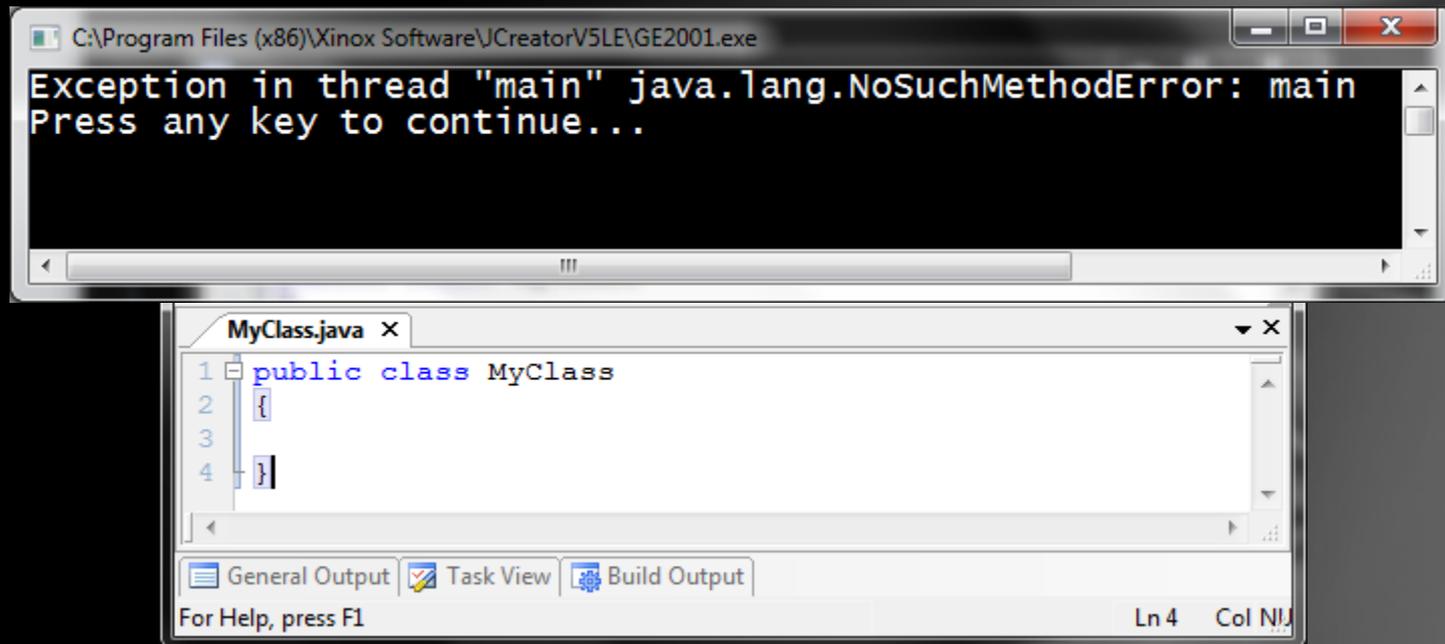
```
1 public class MyClass
2 {
3
4 }
```

The IDE includes a menu bar (File, Edit, View, Project, Build, Run, Tools, Configure, Window, Help), a toolbar, and a status bar at the bottom with the text "For Help, press F1" and "Ln 4 Col N/A".



The class skeleton

- It will compile, but will not run since there is no main method, but that is OK for now..



The screenshot shows a Java IDE window with a terminal and a code editor. The terminal window displays the following error message:

```
C:\Program Files (x86)\Xinox Software\JCreatorV5LE\GE2001.exe  
Exception in thread "main" java.lang.NoSuchMethodError: main  
Press any key to continue...
```

The code editor window shows the following code:

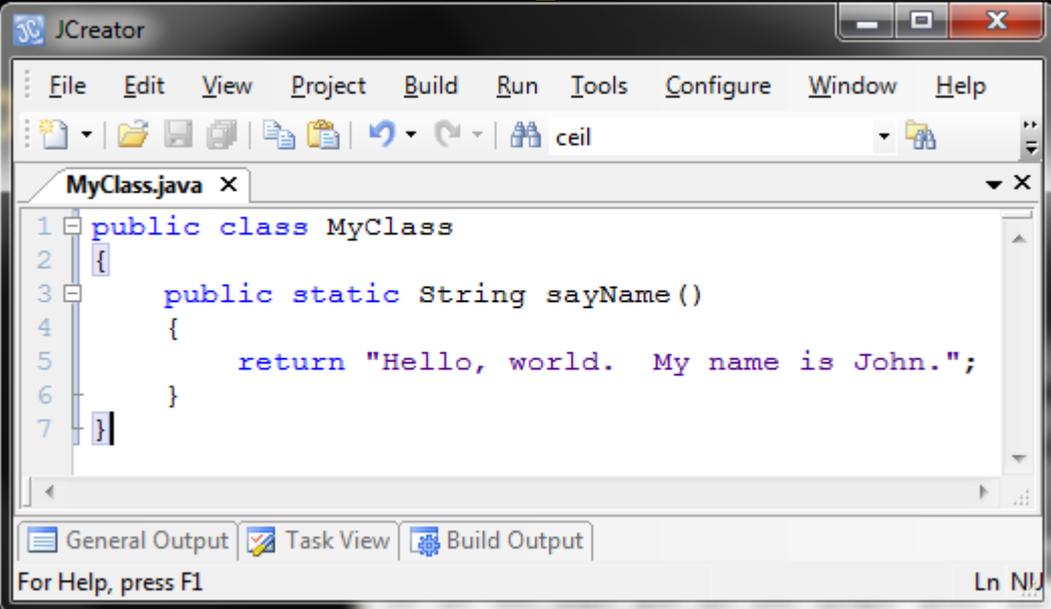
```
1 public class MyClass  
2 {  
3  
4 }
```

The IDE interface includes tabs for 'General Output', 'Task View', and 'Build Output'. The status bar at the bottom indicates 'For Help, press F1' and 'Ln 4 Col N/A'.



Defining and calling a static method

- Let's define a very simple static method, one that returns a String, like, "Hello, world. My name is John."
- We'll call the method **sayName()**
- **Here it is!**



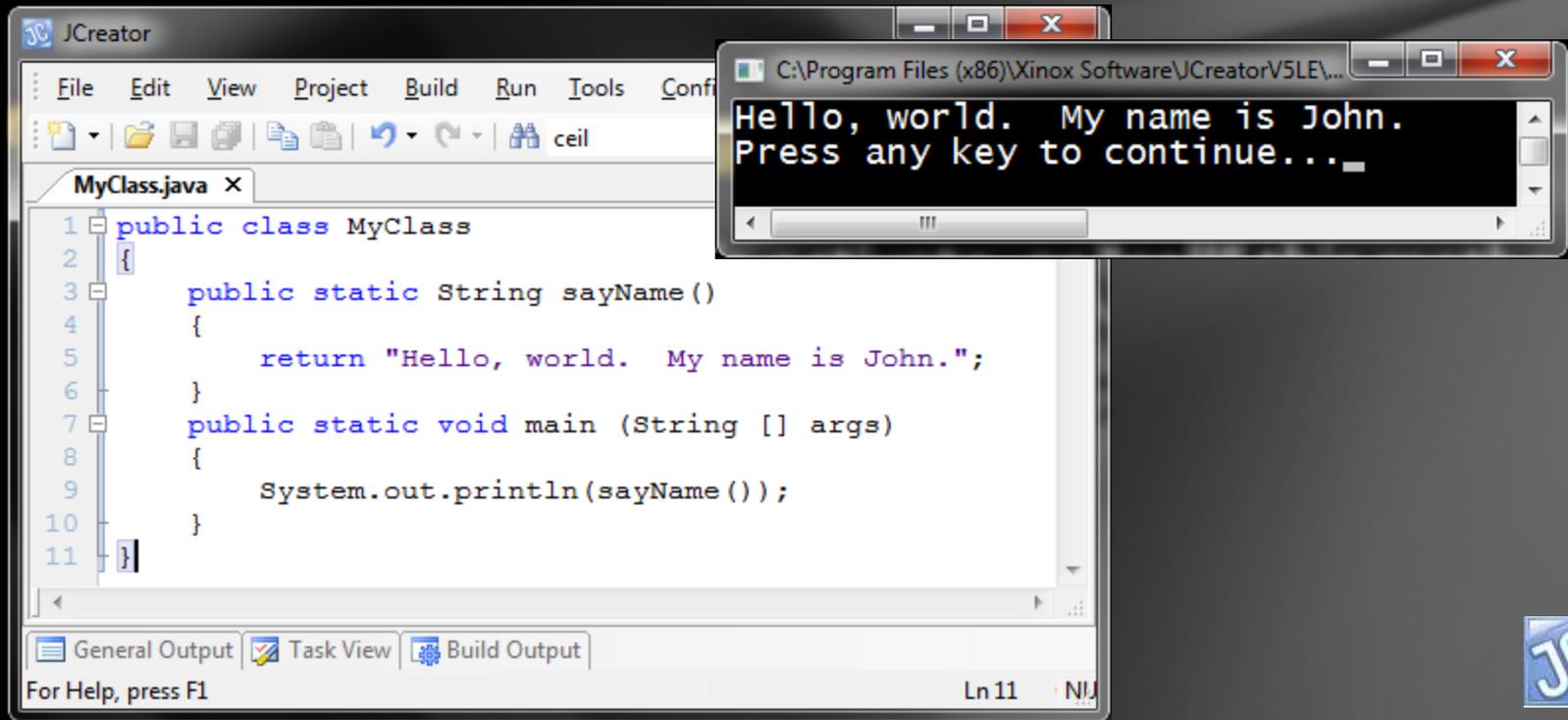
```
1 public class MyClass
2 {
3     public static String sayName ()
4     {
5         return "Hello, world. My name is John.";
6     }
7 }
```

The screenshot shows the JCreator IDE with a single file named 'MyClass.java' open. The code defines a public class 'MyClass' with a public static method 'sayName()' that returns the string 'Hello, world. My name is John.'. The IDE interface includes a menu bar (File, Edit, View, Project, Build, Run, Tools, Configure, Window, Help), a toolbar, and a status bar at the bottom with 'For Help, press F1' and 'Ln N/A'.



Defining and calling a static method

- To activate, or “call” this method, we need the main method with an output statement.



The screenshot shows the JCreator IDE with a Java file named MyClass.java. The code defines a static method sayName() that returns the string "Hello, world. My name is John." and a main method that calls sayName() and prints the result. A separate window shows the output of the program, which is "Hello, world. My name is John. Press any key to continue...".

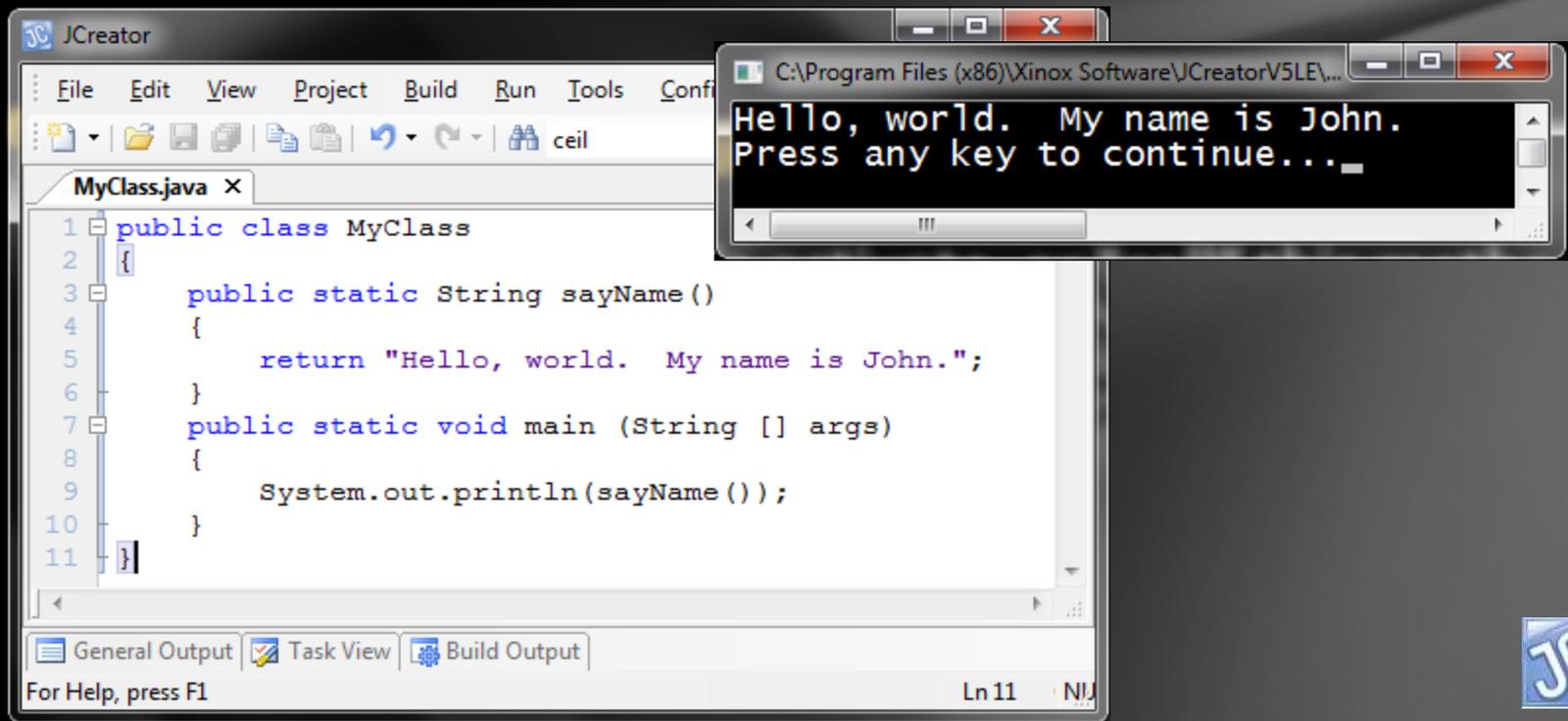
```
1 public class MyClass
2 {
3     public static String sayName ()
4     {
5         return "Hello, world. My name is John.";
6     }
7     public static void main (String [] args)
8     {
9         System.out.println (sayName ());
10    }
11 }
```

Output: Hello, world. My name is John. Press any key to continue...



Defining and calling a static method

- Since the println method requires a string, the call to **sayName()** provides that String.



The screenshot shows the JCreator IDE with a Java file named MyClass.java. The code defines a public class MyClass with two static methods: sayName() and main(). The sayName() method returns the string "Hello, world. My name is John.". The main() method calls sayName() and prints the result. A separate console window shows the output of the program: "Hello, world. My name is John. Press any key to continue...".

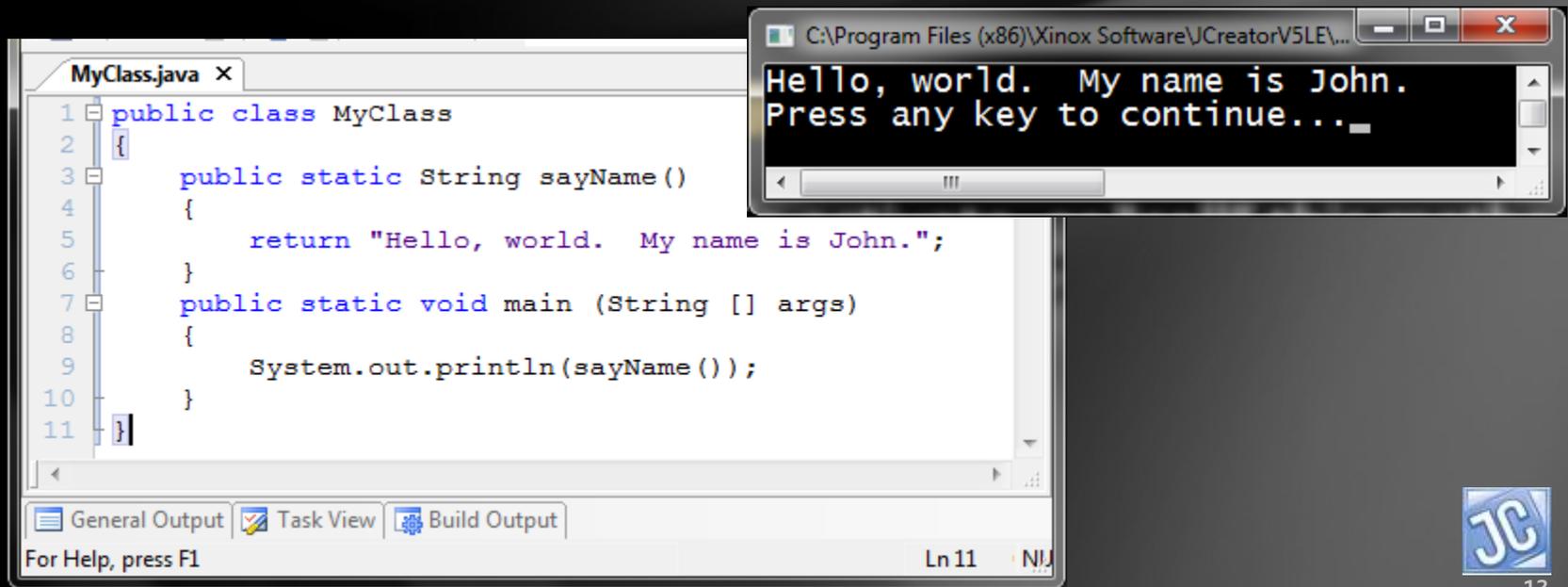
```
1 public class MyClass
2 {
3     public static String sayName ()
4     {
5         return "Hello, world. My name is John.";
6     }
7     public static void main (String [] args)
8     {
9         System.out.println (sayName ());
10    }
11 }
```

General Output Task View Build Output
For Help, press F1 Ln 11



Defining and calling a static method

- When the **sayName()** method is called, the body of the method is executed, and a String is returned to the source of the call.



The screenshot displays an IDE window titled 'MyClass.java' with the following code:

```
1 public class MyClass
2 {
3     public static String sayName()
4     {
5         return "Hello, world. My name is John.";
6     }
7     public static void main (String [] args)
8     {
9         System.out.println(sayName());
10    }
11 }
```

Overlaid on the code is a console window showing the output of the program:

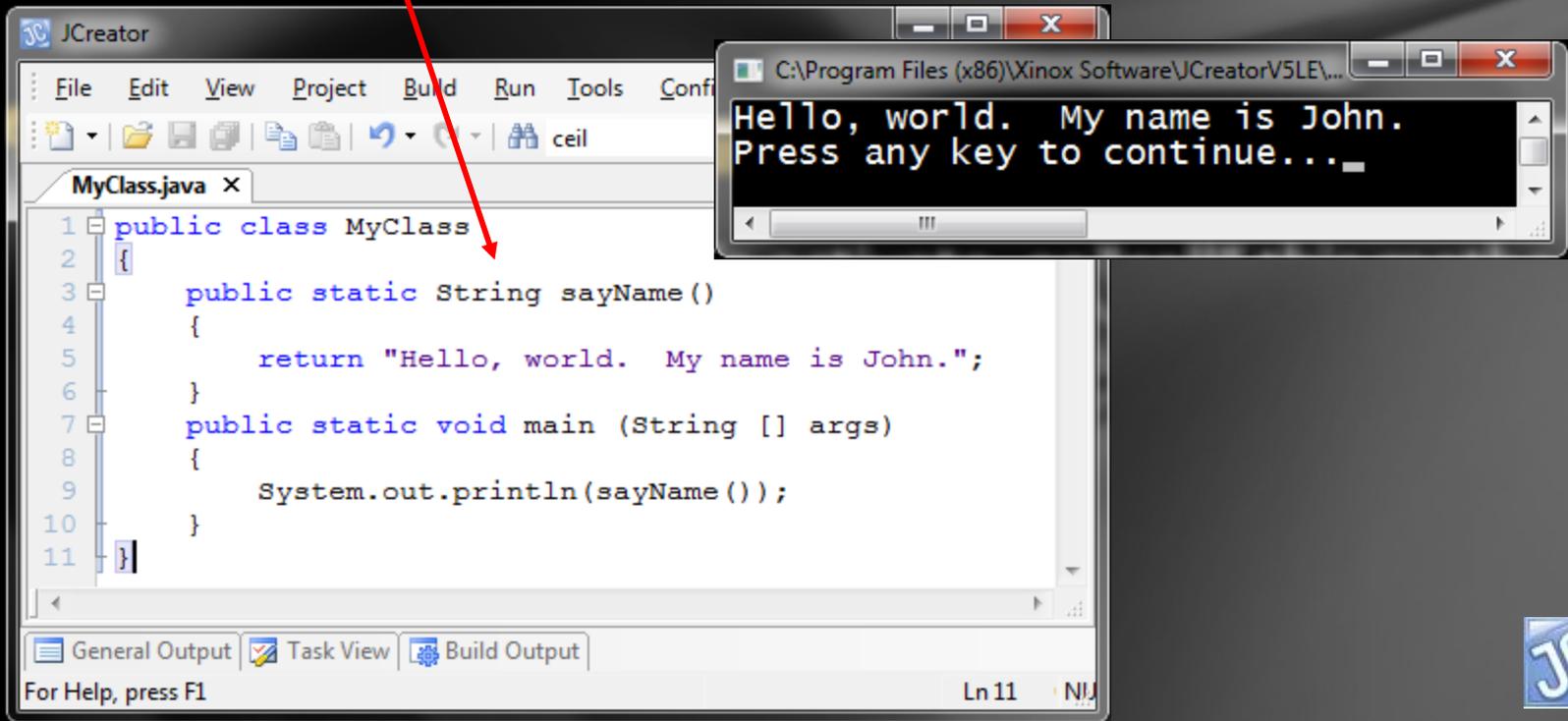
```
Hello, world. My name is John.
Press any key to continue...
```

The IDE interface includes a 'General Output' tab, a 'Task View' icon, and a 'Build Output' icon. The status bar at the bottom indicates 'Ln 11' and 'NJ'.



Method structure - sayName

- Let's talk about the method structure.
- This is a *return* method since it returns something, in this case, a String.



The screenshot shows the JCreator IDE with the following code in MyClass.java:

```
1 public class MyClass
2 {
3     public static String sayName ()
4     {
5         return "Hello, world. My name is John.";
6     }
7     public static void main (String [] args)
8     {
9         System.out.println (sayName ());
10    }
11 }
```

The execution output window shows the following text:

```
Hello, world. My name is John.
Press any key to continue...
```

A red arrow points from the text "something, in this case, a String." in the list above to the `return` statement in the `sayName` method.



Method structure - sayName

- The method header contains four parts: public static String sayName()

```
public static String sayName()  
{  
    return "Hello, world. My name is John.";  
}
```



Method structure - sayName

- The “public” designation makes the method available to use anywhere the class is “visible” (more on that later).

```
public static String sayName()  
{  
    return "Hello, world. My name is John.";  
}
```

- There are two other access designations we'll also discuss later on: *private* and *protected*



Method structure - sayName

- The word “static” makes it a “stand-alone” method, belonging to the class, not requiring an object to call it.
- This is like a Math class method because it is “stand-alone”, but not like a String class method, which requires an object.

```
public static String sayName ()  
{  
    return "Hello, world. My name is John."  
}
```



Method structure - sayName

- The word "String" indicates the type of data it will return.
- Finally, the name of the method, **sayName()**, completes the header as the *identifier* of the method.

```
public static String sayName ()  
{  
    return "Hello, world. My name is John."  
}
```



Method structure - sayName

- The empty parentheses are necessary to indicate to the compiler that it is a *method*, even though no parameter (variable inside parentheses) is required for this one.

```
public static String sayName()  
{  
    return "Hello, world. My name is John.";  
}
```



Method structure - sayName

- The body of the method is contained inside the pair of curly braces... { }
- For this method, the only statement in the body is the return statement, which returns the String.

```
public static String sayName()  
{  
    return "Hello, world. My name is John.";  
}
```



Method structure - sayName

- One final but important part is the documentation, which is a simple comment that precedes the method definition, but is important, especially if a method is particularly complex.

```
//returns a greeting sentence
public static String sayName()
{
    return "Hello, world. My name is John.";
}
public static void main (String [] args)
{
    System.out.println(sayName());
}
```



Parameter method - *firstHalf*

- Now let's define a method that requires a parameter.

Study carefully this new method definition for *firstHalf*, and the output shown below.

```
MyClass.java x
1 public class MyClass
2 {
3     //returns a greeting sentence
4     public static String sayName()
5     {
6         return "Hello, world. My name is John.";
7     }
8     //returns the first half of the parameter word
9     public static String firstHalf(String word)
10    {
11        int halfway = word.length()/2;
12        String answer = word.substring(0, halfway);
13        return answer;
14    }
15    public static void main (String [] args)
16    {
17        //System.out.println(sayName());
18        System.out.println(firstHalf("rhinoceros"));
19        System.out.println(firstHalf("elephants"));
20        System.out.println(firstHalf(sayName()));
21    }
22 }
```

```
C:\Program Files (x86)\Xinox Software...
rhino
elep
Hello, world.
Press any key to continue...
```



Parameter method - *firstHalf*

```
MyClass.java x
1 public class MyClass
2 {
3     //returns a greeting sentence
4     public static String sayName()
5     {
6         return "Hello, world.  My name is John.";
7     }
8     //returns the first half of the parameter word
9     public static String firstHalf(String word)
10    {
11        int halfway = word.length()/2;
12        String answer = word.substring(0, halfway);
13        return answer;
14    }
15    public static void main (String [] args)
16    {
17        //System.out.println(sayName());
18        System.out.println(firstHalf("rhinoceros"));
19        System.out.println(firstHalf("elephants"));
20        System.out.println(firstHalf(sayName()));
21    }
22 }
```

The *firstHalf* method will receive a String parameter, calculate the first half of it, and then returns that calculated answer.

```
C:\Program Files (x86)\Xinox Software...
rhino
elep
Hello, world.
Press any key to continue...
```



Parameter method - firstHalf

Notice carefully that other String class methods were used in the definition of the new custom method.

Using previously defined methods is an important technique and should be done as often as possible to save work and avoid duplication of effort.

```
MyClass.java x
1 public class MyClass
2 {
3     //returns a greeting sentence
4     public static String sayName()
5     {
6         return "Hello, world.  My name is John.";
7     }
8     //returns the first half of the parameter word
9     public static String firstHalf(String word)
10    {
11        int halfway = word.length()/2;
12        String answer = word.substring(0, halfway);
13        return answer;
14    }
15    public static void main (String [] args)
16    {
17        //System.out.println(sayName());
18        System.out.println(firstHalf("rhinoceros"));
19        System.out.println(firstHalf("elephants"));
20        System.out.println(firstHalf(sayName()));
21    }
22 }
```

```
C:\Program Files (x86)\Xinox Software...
rhino
elep
Hello, world.
Press any key to continue...
```



Parameter method - firstHalf

Also notice that each calculation step was done separately.

Although it is possible to imbed all the steps into one statement, it makes your code clearer if you limit the actions to one or two steps per command.

```
MyClass.java x
1 public class MyClass
2 {
3     //returns a greeting sentence
4     public static String sayName()
5     {
6         return "Hello, world.  My name is John.";
7     }
8     //returns the first half of the parameter word
9     public static String firstHalf(String word)
10    {
11        int halfway = word.length()/2;
12        String answer = word.substring(0, halfway);
13        return answer;
14    }
15    public static void main (String [] args)
16    {
17        //System.out.println(sayName());
18        System.out.println(firstHalf("rhinoceros"));
19        System.out.println(firstHalf("elephants"));
20        System.out.println(firstHalf(sayName()));
21    }
22 }
```

```
C:\Program Files (x86)\Xinox Software...
rhino
elep
Hello, world.
Press any key to continue...
```



Parameter method - first

Just in case you are curious, below is the “condensed” version of the method, where everything *is* done in one command.

The same steps are all there, just in one longer command, which in this case is not too bad, but is just about as long as you would want and still have good programming style.

```
MyClass.java x
1 public class MyClass
2 {
3     //returns a greeting sentence
4     public static String sayName()
5     {
6         return "Hello, world.  My name is John.";
7     }
8     //returns the first half of the parameter word
9     public static String firstHalf(String word)
10    {
11        int halfway = word.length()/2;
12        String answer = word.substring(0, halfway);
13        return answer;
14    }
15 }
16 //returns the first half of the parameter word
17 public static String firstHalf(String word)
18 {
19     return word.substring(0, word.length()/2);
20 }
21 System.out.println(firstHalf(sayName()));
22 }
```

```
C:\Program Files (x86)\Xinox Software...
rhino
elep
Hello, world.
Press any key to continue...
```



Math method - firstLastValue

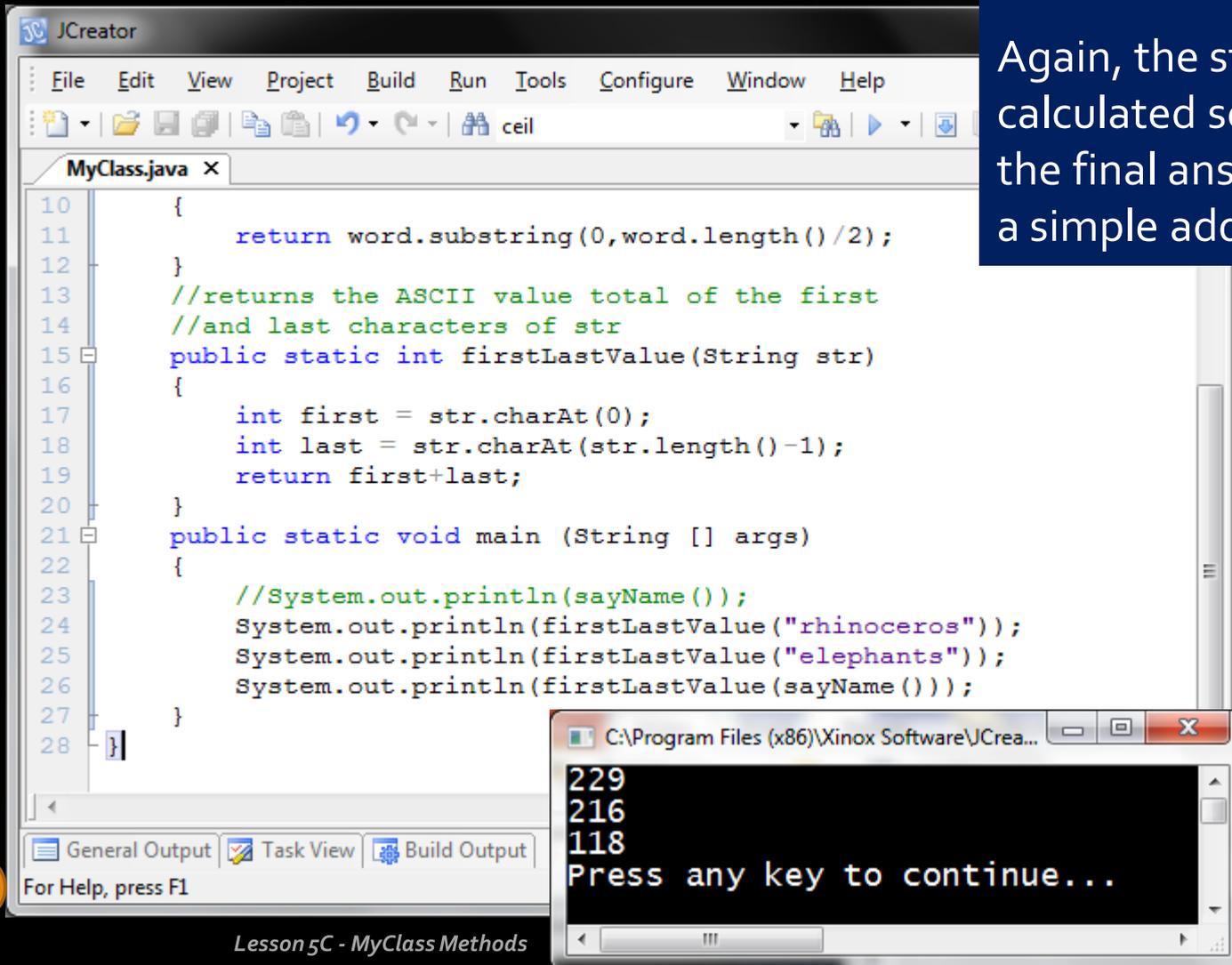
- Now let's define a method that calculates and returns an integer.
- We'll call it *firstLastValue(String str)*, which will receive a String, add together the ASCII values of the first and last letters in the String, and return that total.



Math method - firstLastValue

Here it is!

Again, the steps are all calculated separately, with the final answer returned as a simple addition step.



```
10 {
11     return word.substring(0,word.length()/2);
12 }
13 //returns the ASCII value total of the first
14 //and last characters of str
15 public static int firstLastValue(String str)
16 {
17     int first = str.charAt(0);
18     int last = str.charAt(str.length()-1);
19     return first+last;
20 }
21 public static void main (String [] args)
22 {
23     //System.out.println(sayName());
24     System.out.println(firstLastValue("rhinoceros"));
25     System.out.println(firstLastValue("elephants"));
26     System.out.println(firstLastValue(sayName()));
27 }
28 }
```

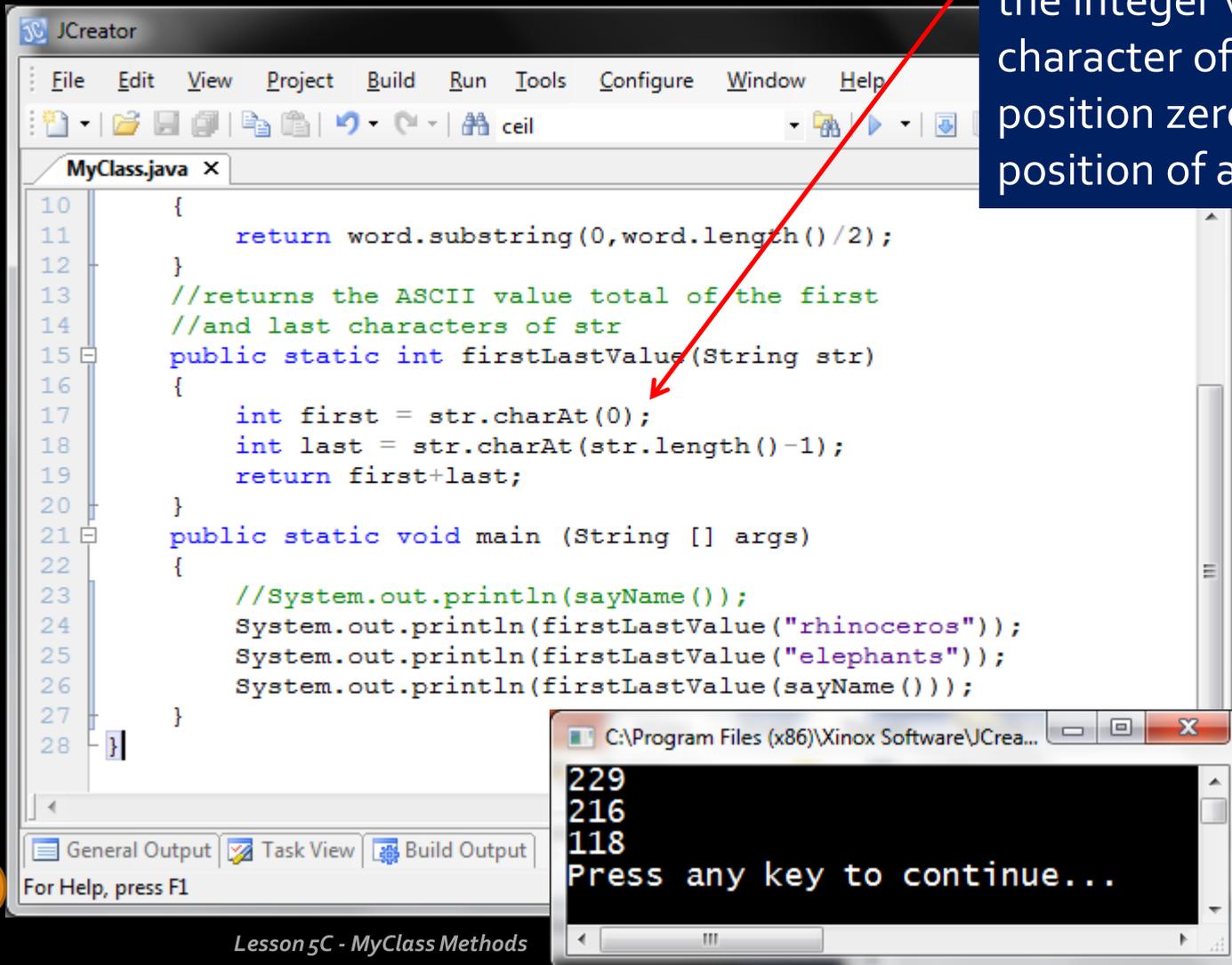
General Output Task View Build Output
For Help, press F1

```
C:\Program Files (x86)\Xinox Software\JCrea...
229
216
118
Press any key to continue...
```



Math method - firstLastValue

The first calculation finds the integer value of the first character of str, the one at position zero, the first position of any string.



```
10 {
11     return word.substring(0,word.length()/2);
12 }
13 //returns the ASCII value total of the first
14 //and last characters of str
15 public static int firstLastValue(String str)
16 {
17     int first = str.charAt(0);
18     int last = str.charAt(str.length()-1);
19     return first+last;
20 }
21 public static void main (String [] args)
22 {
23     //System.out.println(sayName());
24     System.out.println(firstLastValue("rhinoceros"));
25     System.out.println(firstLastValue("elephants"));
26     System.out.println(firstLastValue(sayName()));
27 }
28 }
```

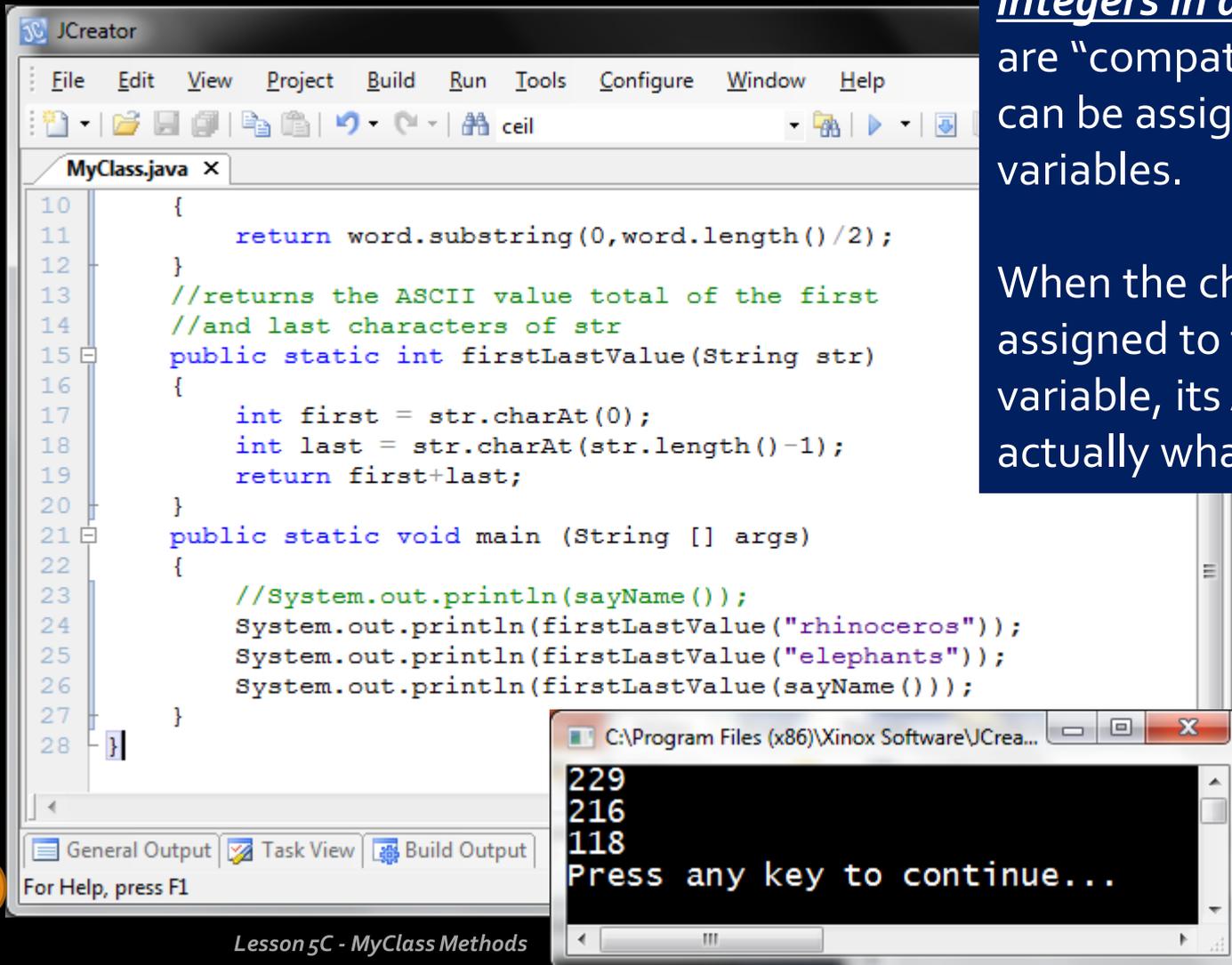
229
216
118
Press any key to continue...



Math method - firstLastValue

Since characters ARE integers in disguise, they are “compatible” with and can be assigned to integer variables.

When the character is assigned to the integer variable, its ASCII value is actually what is stored.



```
10 {
11     return word.substring(0,word.length()/2);
12 }
13 //returns the ASCII value total of the first
14 //and last characters of str
15 public static int firstLastValue(String str)
16 {
17     int first = str.charAt(0);
18     int last = str.charAt(str.length()-1);
19     return first+last;
20 }
21 public static void main (String [] args)
22 {
23     //System.out.println(sayName());
24     System.out.println(firstLastValue("rhinoceros"));
25     System.out.println(firstLastValue("elephants"));
26     System.out.println(firstLastValue(sayName()));
27 }
28 }
```

229
216
118
Press any key to continue...

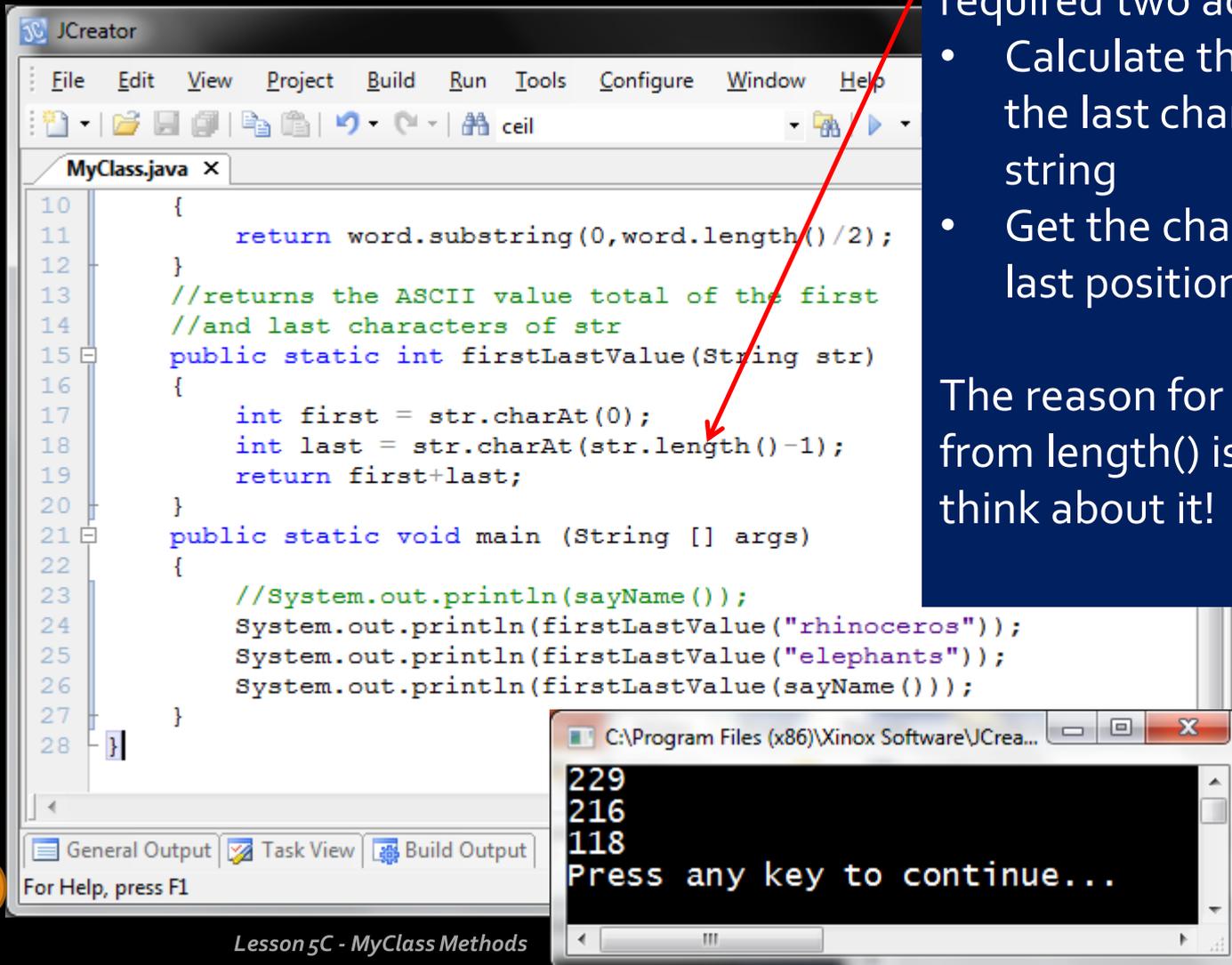


Math method - firstLastValue

The second calculation required two actions:

- Calculate the position of the last character in the string
- Get the character at that last position

The reason for subtracting 1 from length() is important... think about it!



```
10 {
11     return word.substring(0,word.length()/2);
12 }
13 //returns the ASCII value total of the first
14 //and last characters of str
15 public static int firstLastValue(String str)
16 {
17     int first = str.charAt(0);
18     int last = str.charAt(str.length()-1);
19     return first+last;
20 }
21 public static void main (String [] args)
22 {
23     //System.out.println(sayName());
24     System.out.println(firstLastValue("rhinoceros"));
25     System.out.println(firstLastValue("elephants"));
26     System.out.println(firstLastValue(sayName()));
27 }
28 }
```

Output:

```
229
216
118
Press any key to continue...
```



Math method - firstLastValue

```
JCreator
File Edit View Project Build Run Tools Configure Window Help
ceil
MyClass.java x
10 {
11     return word.substring(0,word.length()/2);
12 }
13 //returns the ASCII value total of the first
14 //and last characters of str
15 public static int firstLastValue(String str)
16 {
17     int first = str.charAt(0);
18     int last = str.charAt(str.length()-1);
19     return first+last;
20 }
21 public static void main (String [] args)
22 {
23     //System.out.println(sayName ());
24     System.out.println(firstLastValue ("rhinocer
25     System.out.println(firstLastValue ("elephant
26     System.out.println(firstLastValue (sayName (
27 }
28 }
```

Here's why:

- For the string "rhinoceros", the length command returns the value 10, the number of characters in this string.
- However, the last letter of "rhinoceros" is in position 9 because of "zero-indexing" (we start counting at zero).
- Therefore, for any string, the last position is always its length, minus 1.

```
C:\Program Files (x86)\Xinox Software\JCrea...
229
216
118
Press any key to continue...
```



JavaDocs for JCreator

- Now it is time to create your own API, or “help” file, using a built-in Java tool called Javadoc.
- The steps to customize your JCreator IDE for this capability are covered in the next few slides.
- Follow these steps VERY CAREFULLY!
 - javadoc - creates the help file
 - show javadoc - opens the help file



JavaDocs for JCreator

- We'll create two new tools in JCreator
 - javadoc - creates the help file
 - show javadoc - opens the help file
- The "javadoc" tool will create an html file from your class definitions, as long as you have included the correct documentation.

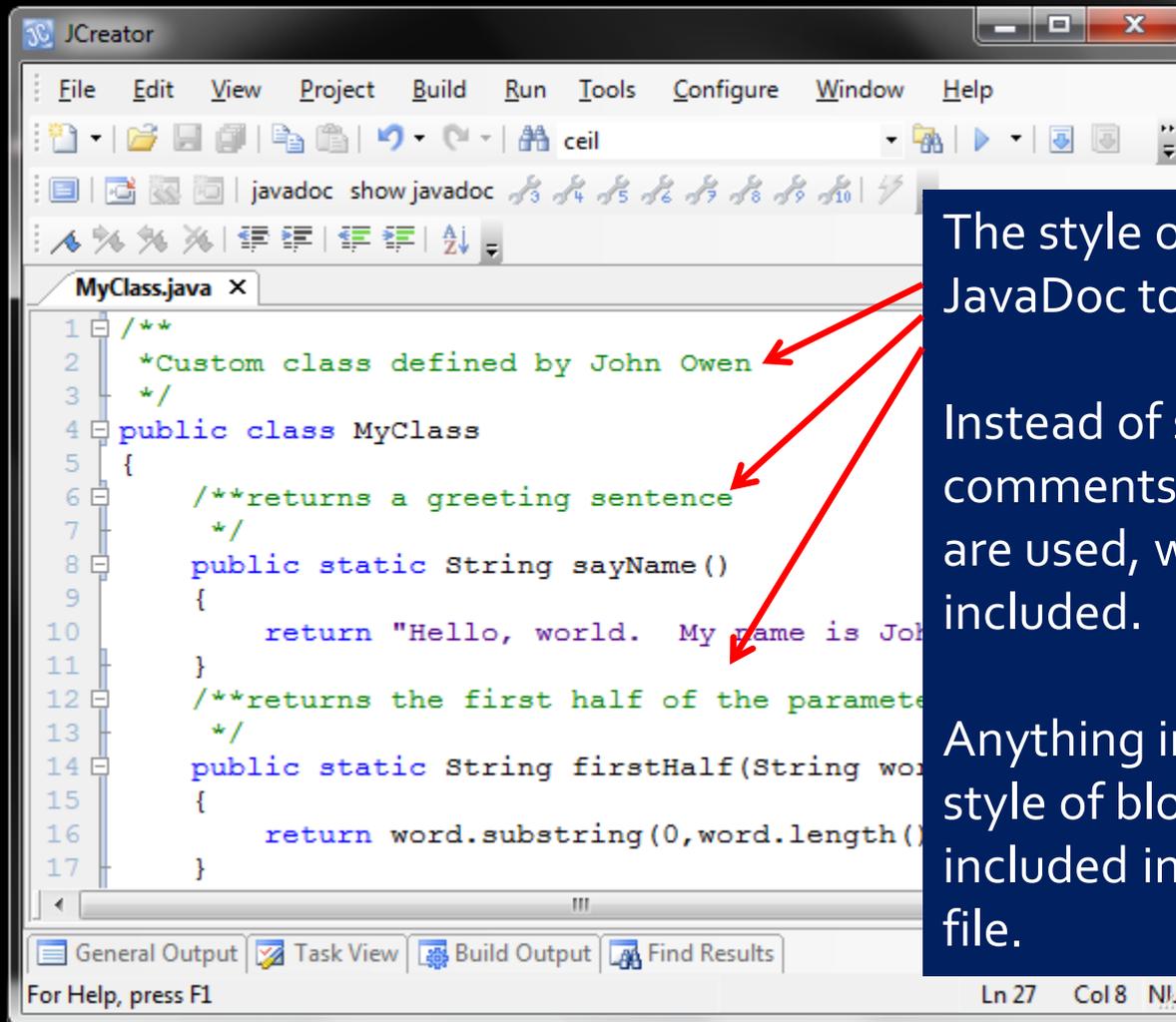


JavaDocs for JCreator

- The “show javadoc” tool will actually open that html file for you in a new browser window.
- **Oh, one more thing...**
- The style of documentation needs to be correct for JavaDoc to work.
- See the next slide for this...



JavaDocs for JCreator



The screenshot shows the JCreator IDE with a code editor window titled 'MyClass.java'. The code contains two methods with JavaDoc comments. Red arrows point from the text boxes on the right to the comment lines in the code.

```
1  /**
2   *Custom class defined by John Owen
3   */
4  public class MyClass
5  {
6     /**returns a greeting sentence
7     */
8     public static String sayName()
9     {
10        return "Hello, world. My name is John";
11    }
12    /**returns the first half of the parameter
13    */
14    public static String firstHalf(String word)
15    {
16        return word.substring(0,word.length()/2);
17    }
18 }
```

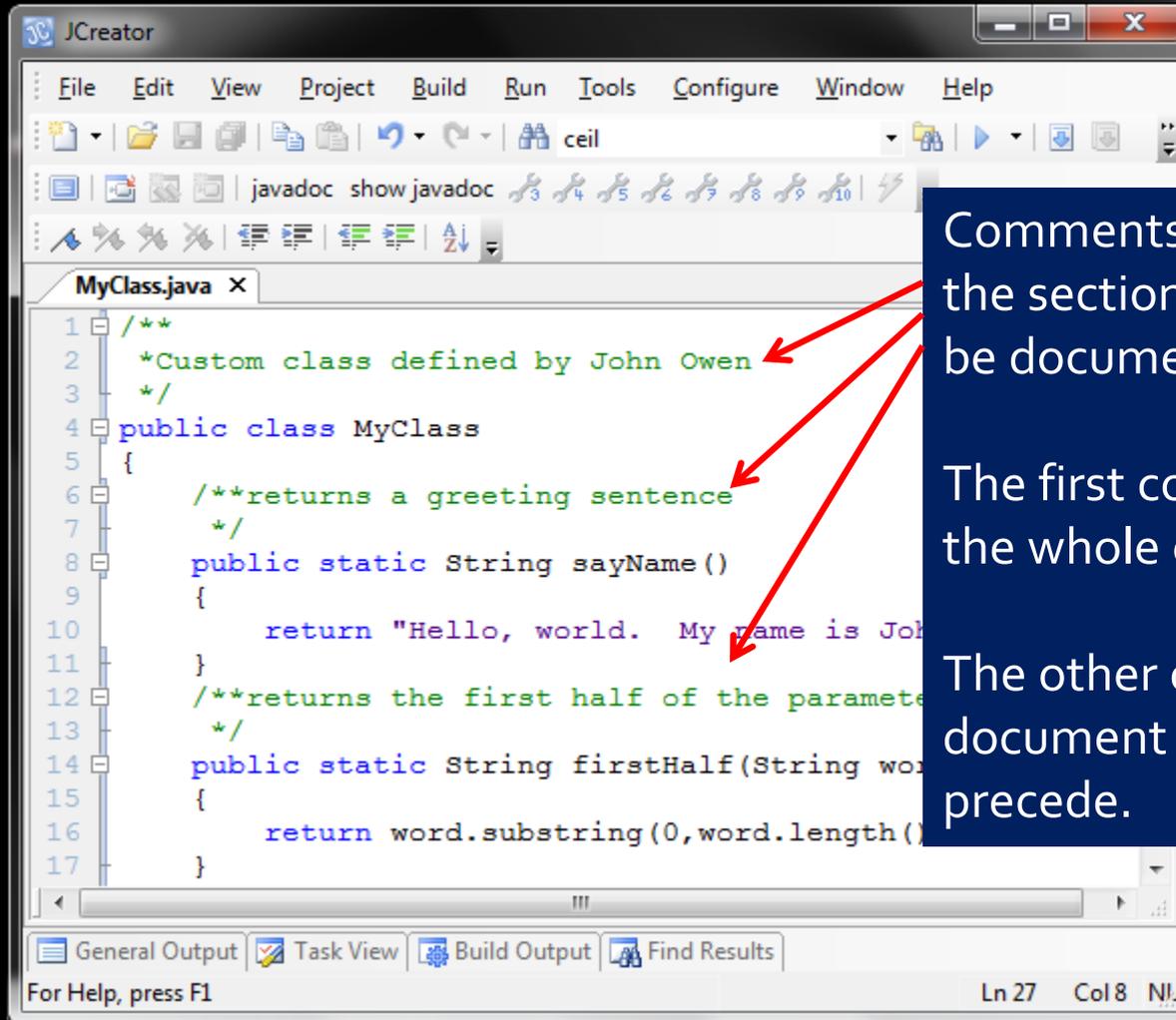
The style of comments for JavaDoc to use is as shown.

Instead of single line comments, block comments are used, with extra `*`s included.

Anything imbedded within this style of block comment will be included in the JavaDoc help file.



JavaDocs for JCreator



```
1  /**
2   *Custom class defined by John Owen
3   */
4  public class MyClass
5  {
6   /**returns a greeting sentence
7   */
8   public static String sayName()
9   {
10    return "Hello, world. My name is John";
11  }
12  /**returns the first half of the parameter
13  */
14  public static String firstHalf(String word)
15  {
16    return word.substring(0,word.length()/2);
17  }
```

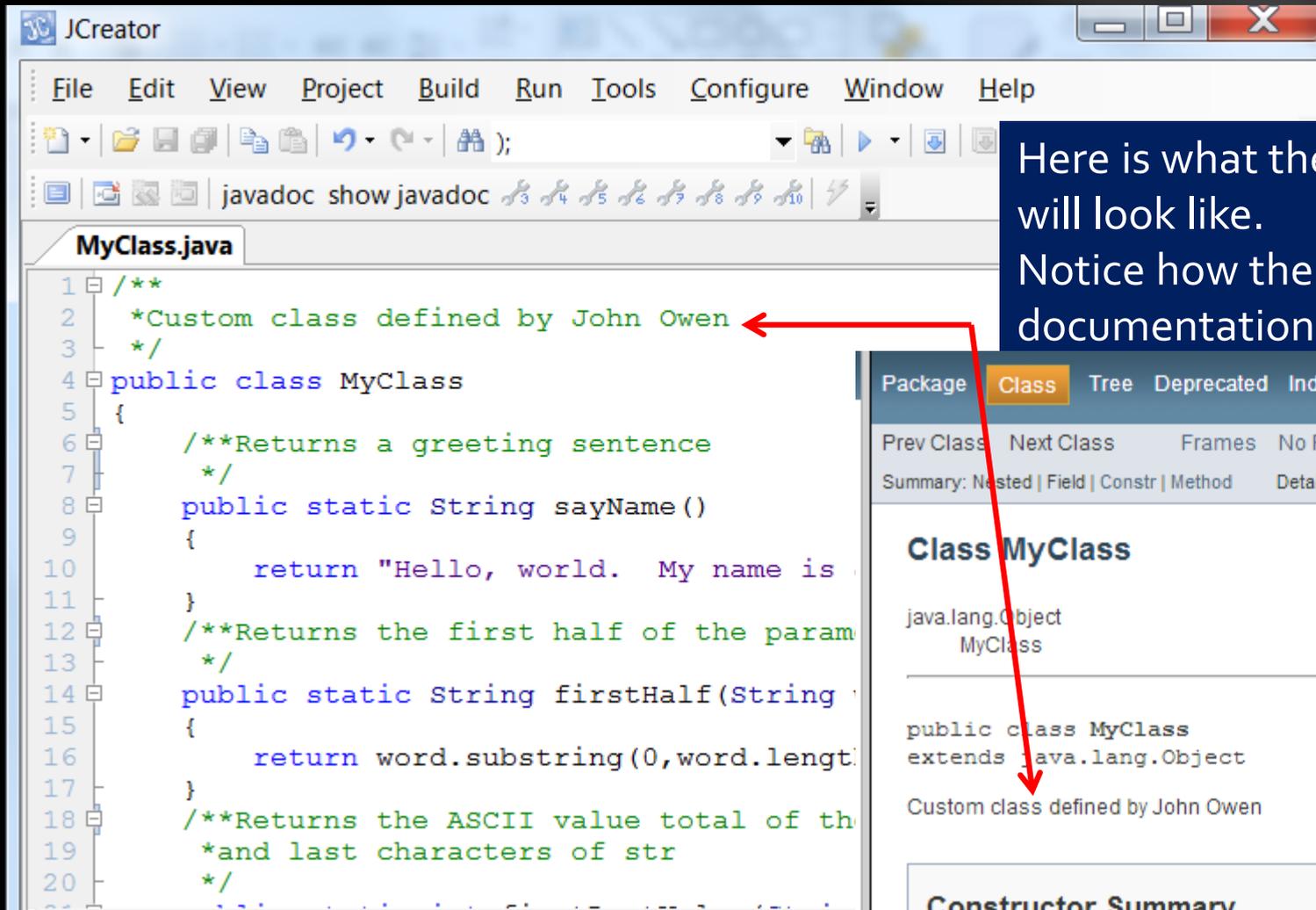
Comments ALWAYS precede the section of the program to be documented.

The first comment describes the whole class.

The other comments document each method they precede.

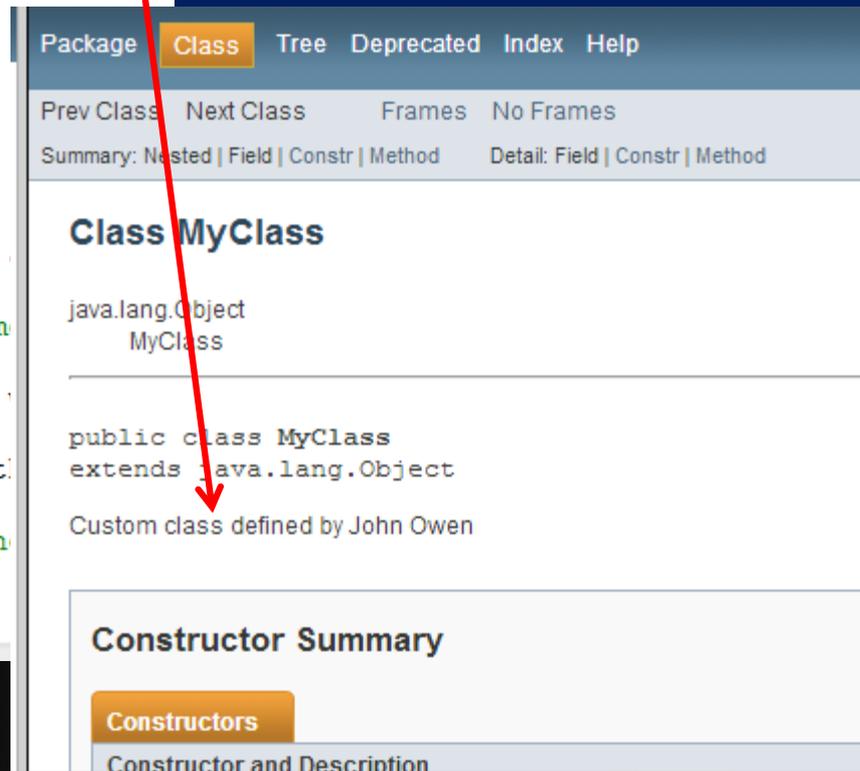


JavaDocs for JCreator



```
1 /**
2  *Custom class defined by John Owen
3  */
4 public class MyClass
5 {
6     /**Returns a greeting sentence
7     */
8     public static String sayName ()
9     {
10         return "Hello, world.  My name is
11     }
12     /**Returns the first half of the param
13     */
14     public static String firstHalf(String
15     {
16         return word.substring(0,word.length
17     }
18     /**Returns the ASCII value total of th
19     *and last characters of str
20     */
```

Here is what the final product will look like. Notice how the documentation matches.



Package Class Tree Deprecated Index Help

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

Class MyClass

java.lang.Object
MyClass

```
public class MyClass
extends java.lang.Object
```

Custom class defined by John Owen

Constructor Summary

Constructors

Constructor and Description



JavaDocs for JCreator

Here is what the final product will look like. Notice how the documentation matches.

```
1 /**
2  *Custom class defined by John Owen
3  */
4 public class MyClass
5 {
6     /**Returns a greeting sentence
7     */
8     public static String sayName()
9     {
10         return "Hello, world. My name is John.";
11     }
12     /**Returns the first half of the parameter word
13     */
14     public static String firstHalf(String word)
15     {
16         return word.substring(0,word.length()/2);
17     }
18     /**Returns the ASC
19     *and last charact
20     */
```

Method Summary

Methods

Modifier and Type	Method and Description
static java.lang.String	firstHalf(java.lang.String word) Returns the first half of the parameter word
static int	firstLastValue(java.lang.String str)

General Output | Build Output | Help, press F1

Lesson 5C - MyClass



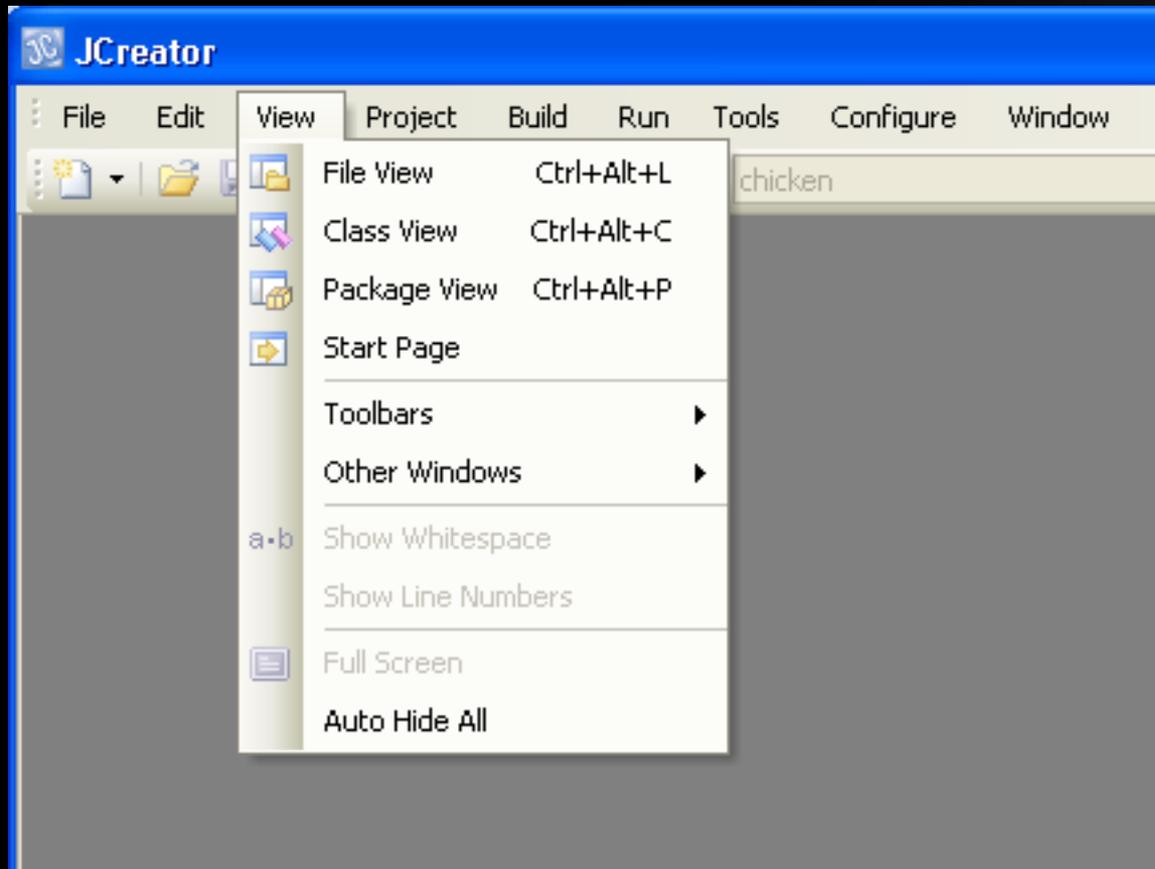
JavaDocs for JCreator

- Now let's create the JavaDoc tools...



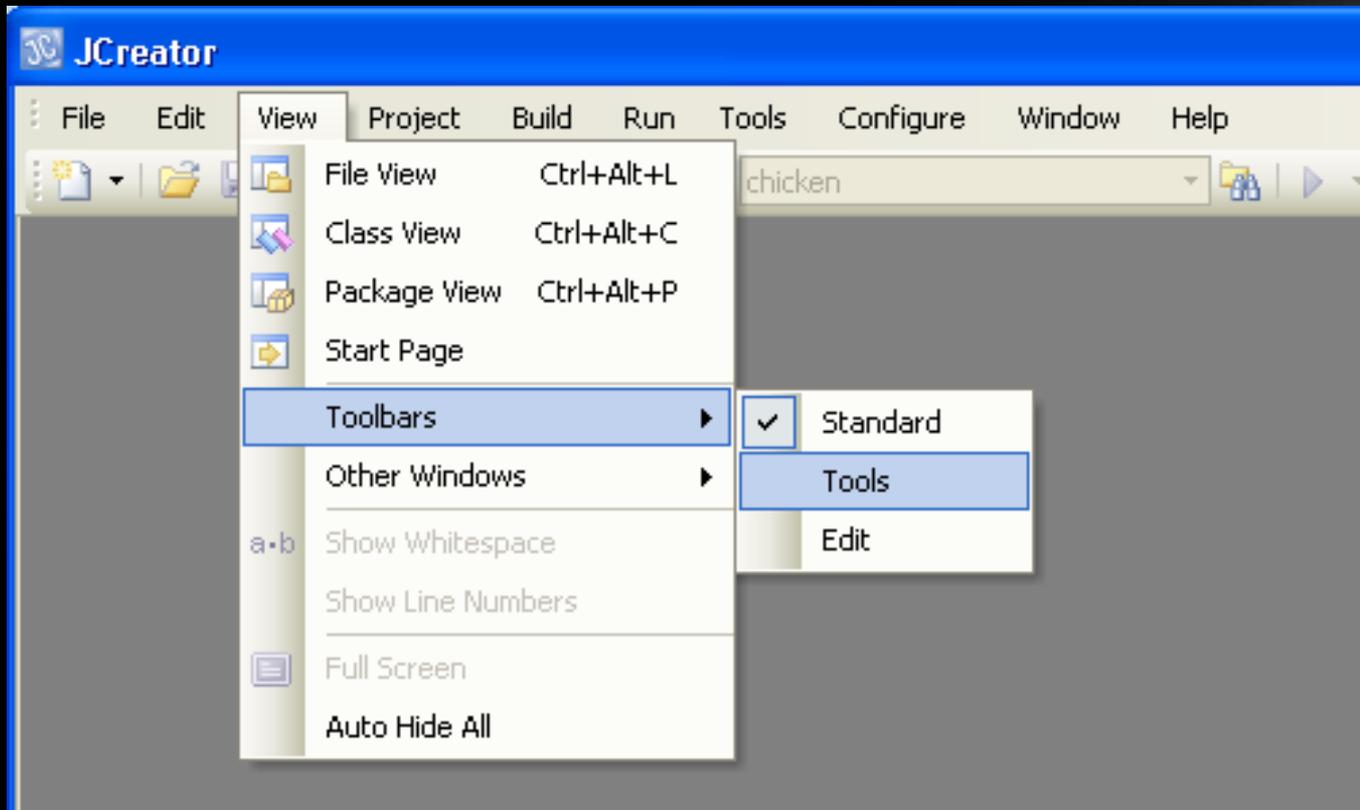
JavaDocs for JCreator

- Open Jcreator and click on the View tab at the top.



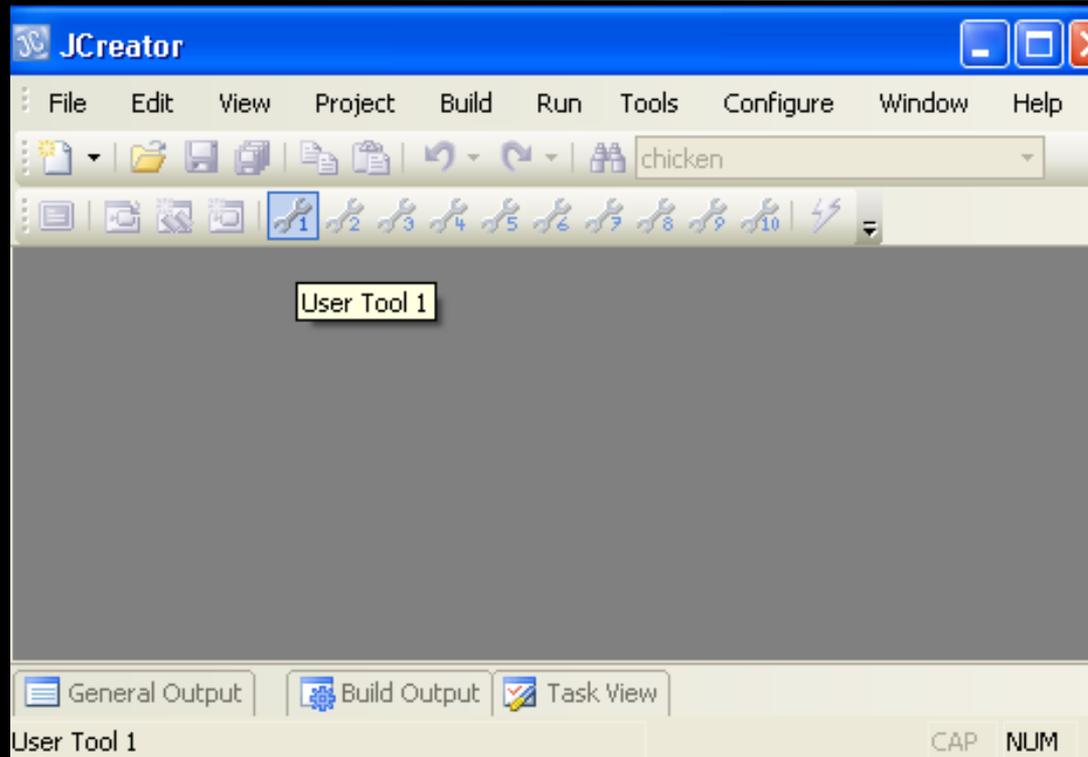
JavaDocs for JCreator

- Then click on the ToolBars black triangle and click on the Tools button.



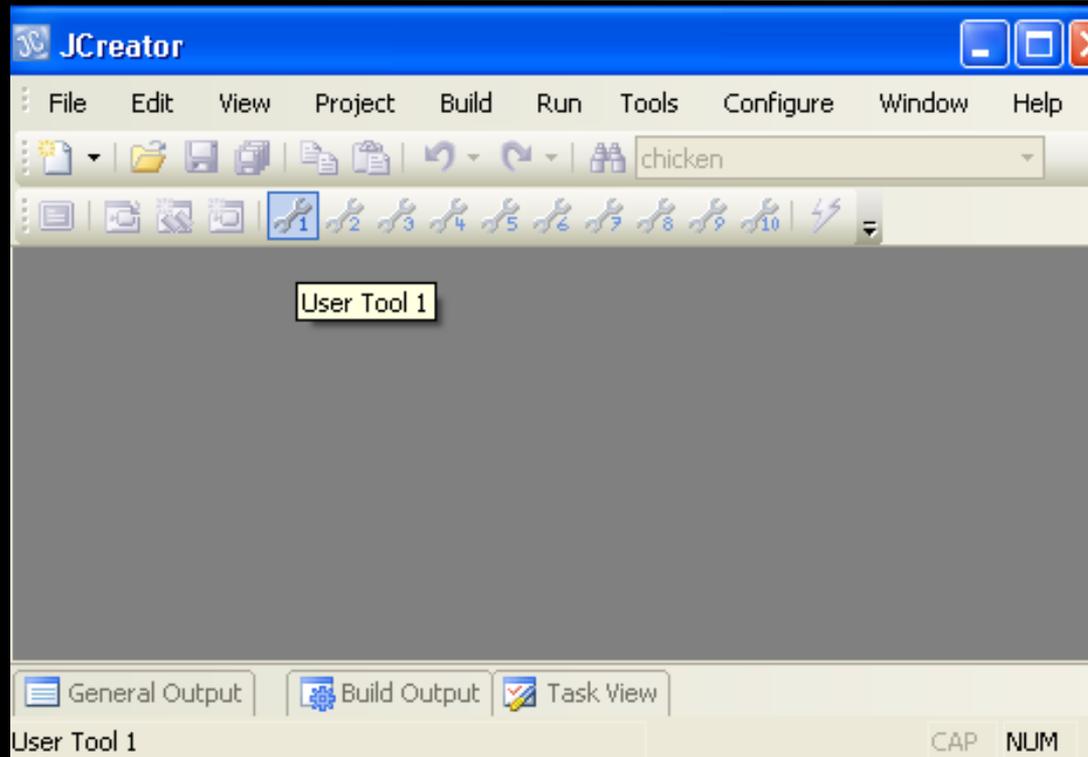
JavaDocs for JCreator

- You should now see a row of Tools that look like wrenches. Hover over the first one to see its name – User Tool 1



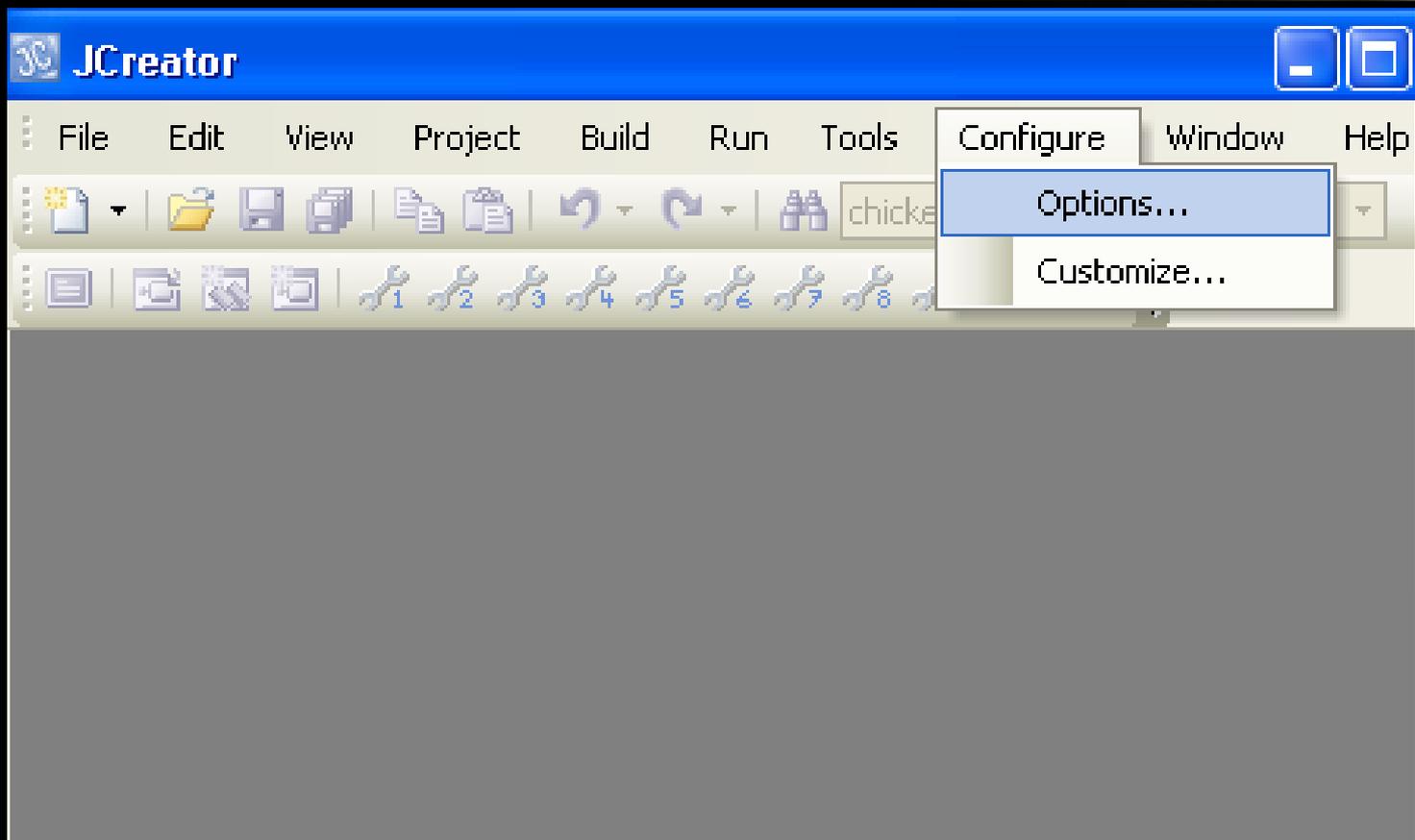
JavaDocs for JCreator

- We will now redefine this tool as the “javadoc” tool. Here’s how...



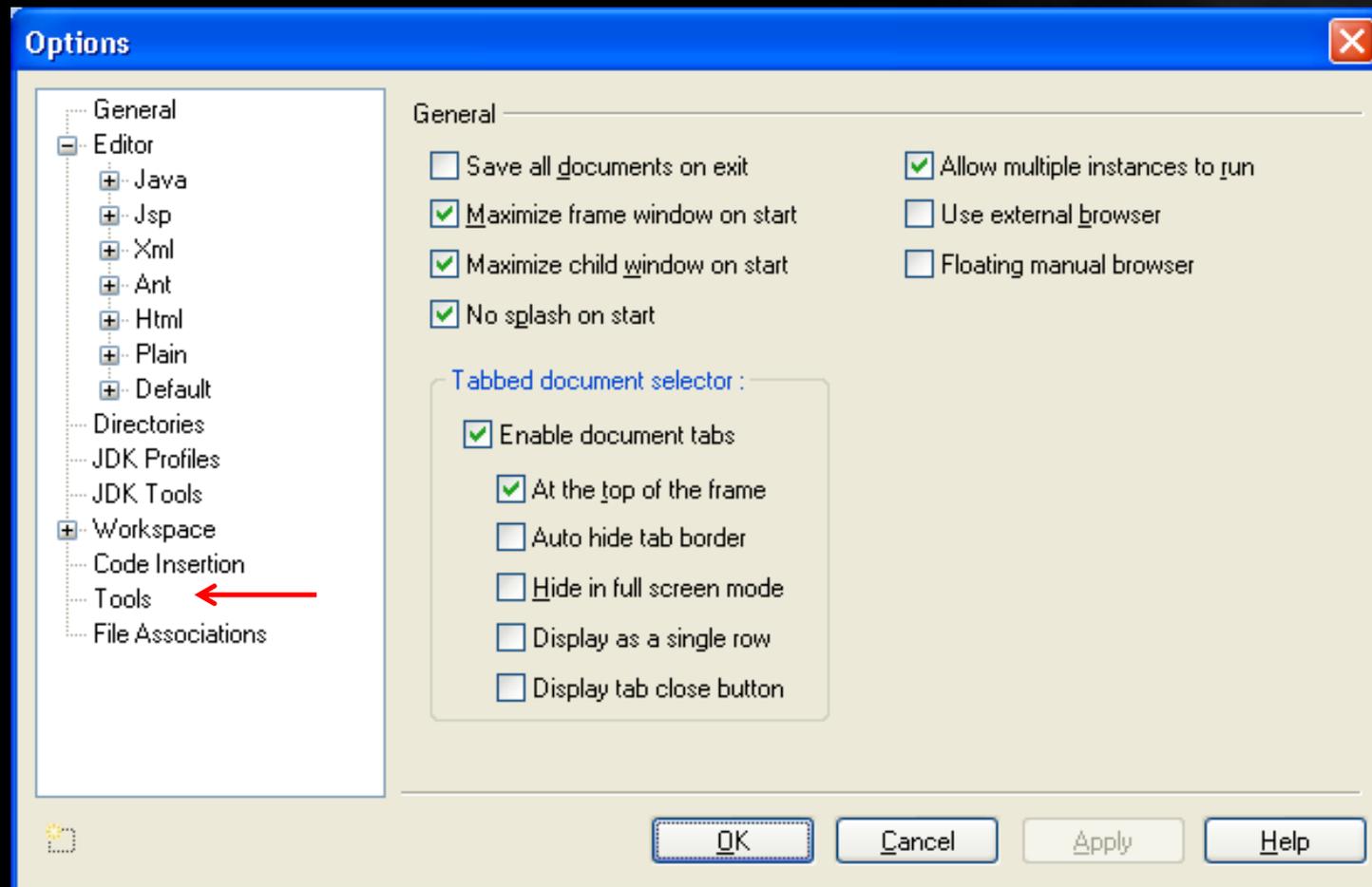
JavaDocs for JCreator

- Click on Configure, then Options...



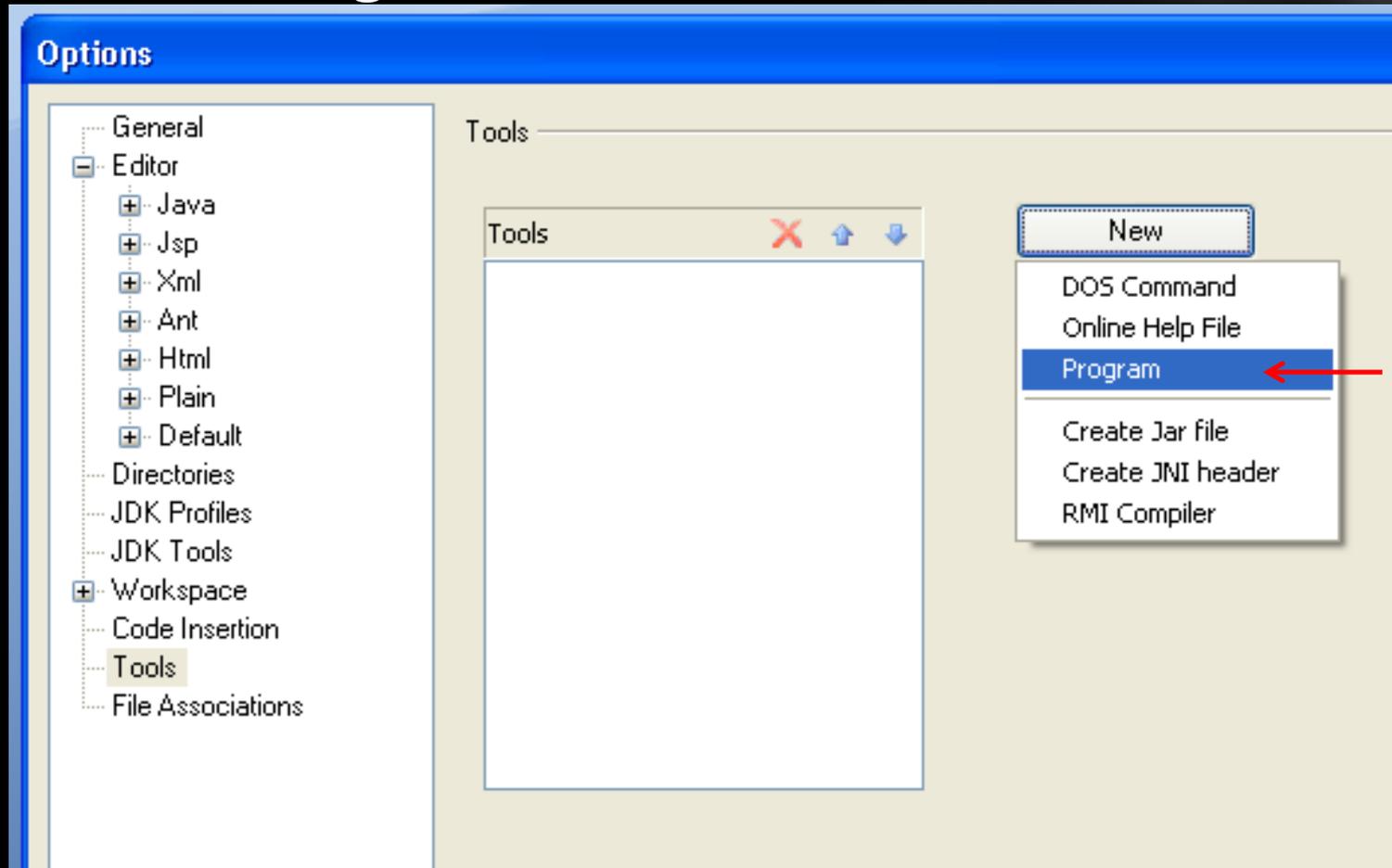
JavaDocs for JCreator

- From here, select Tools...



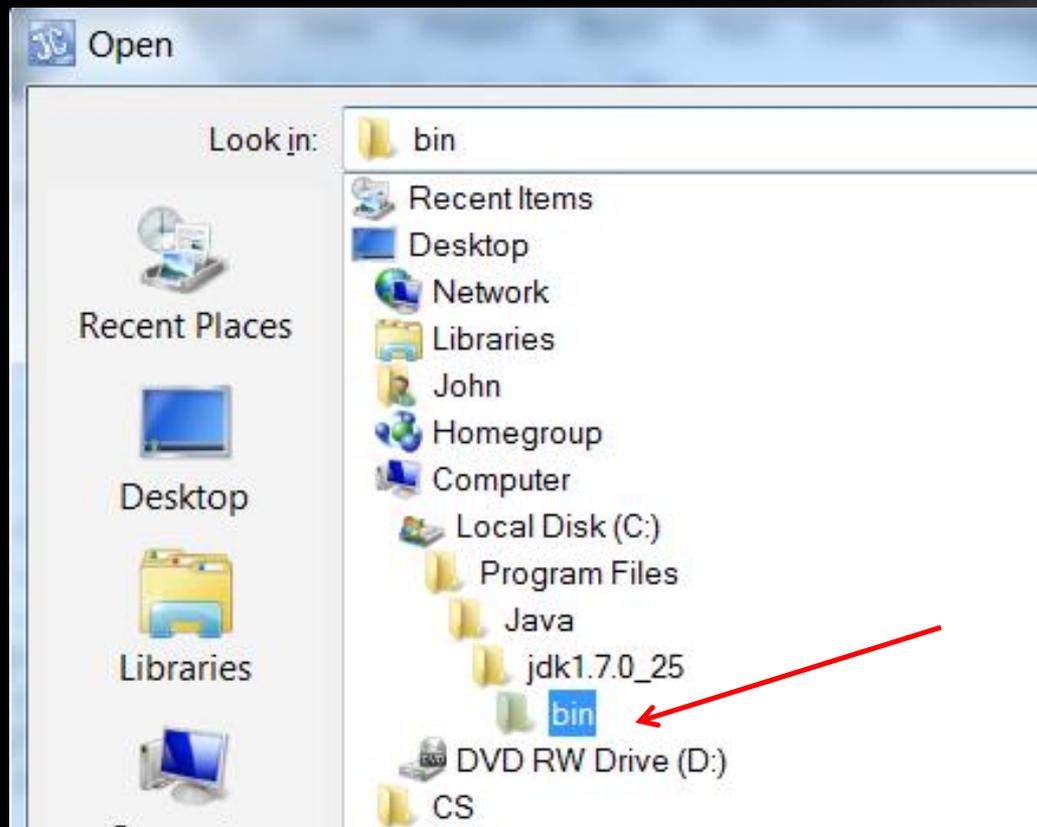
JavaDocs for JCreator

- ...then Program...



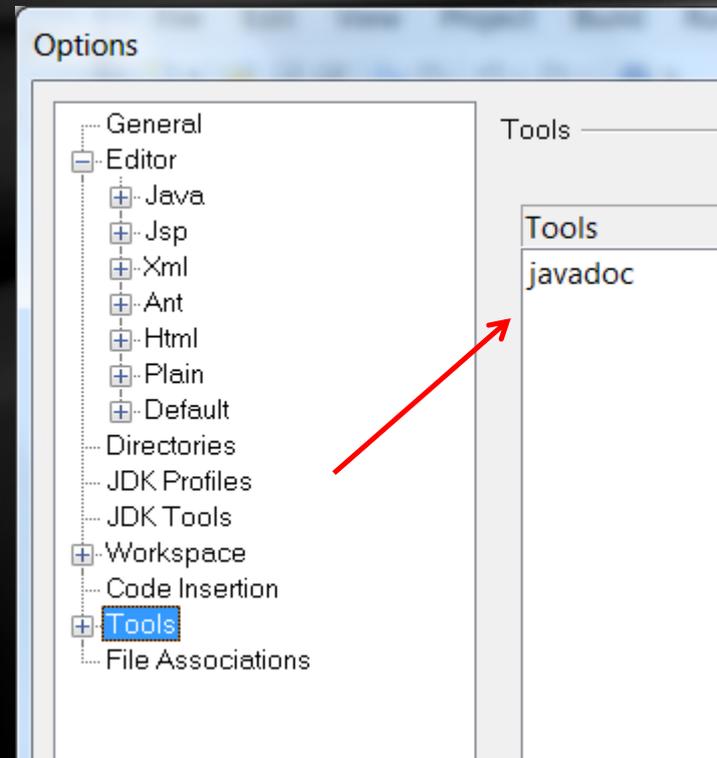
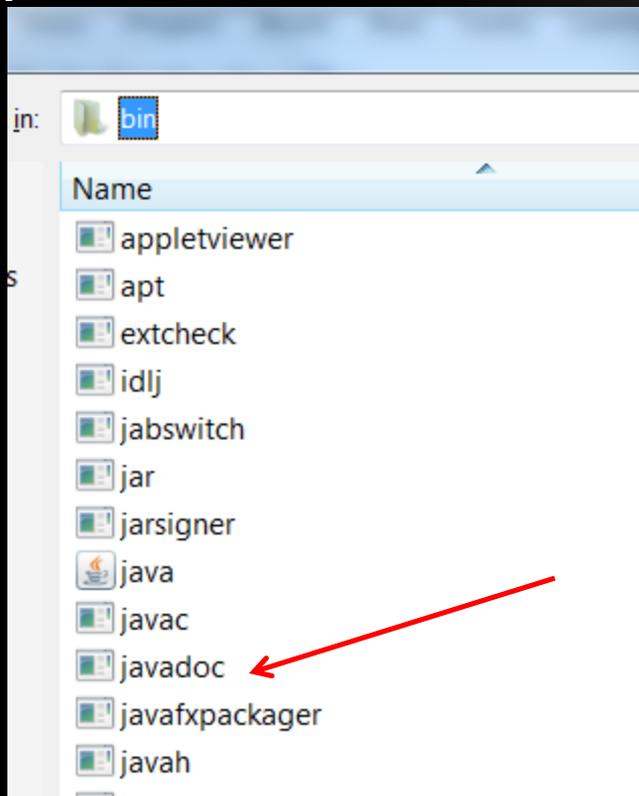
JavaDocs for JCreator

- ...then navigate to wherever Java “lives” on your computer, specifically to the **bin** folder...



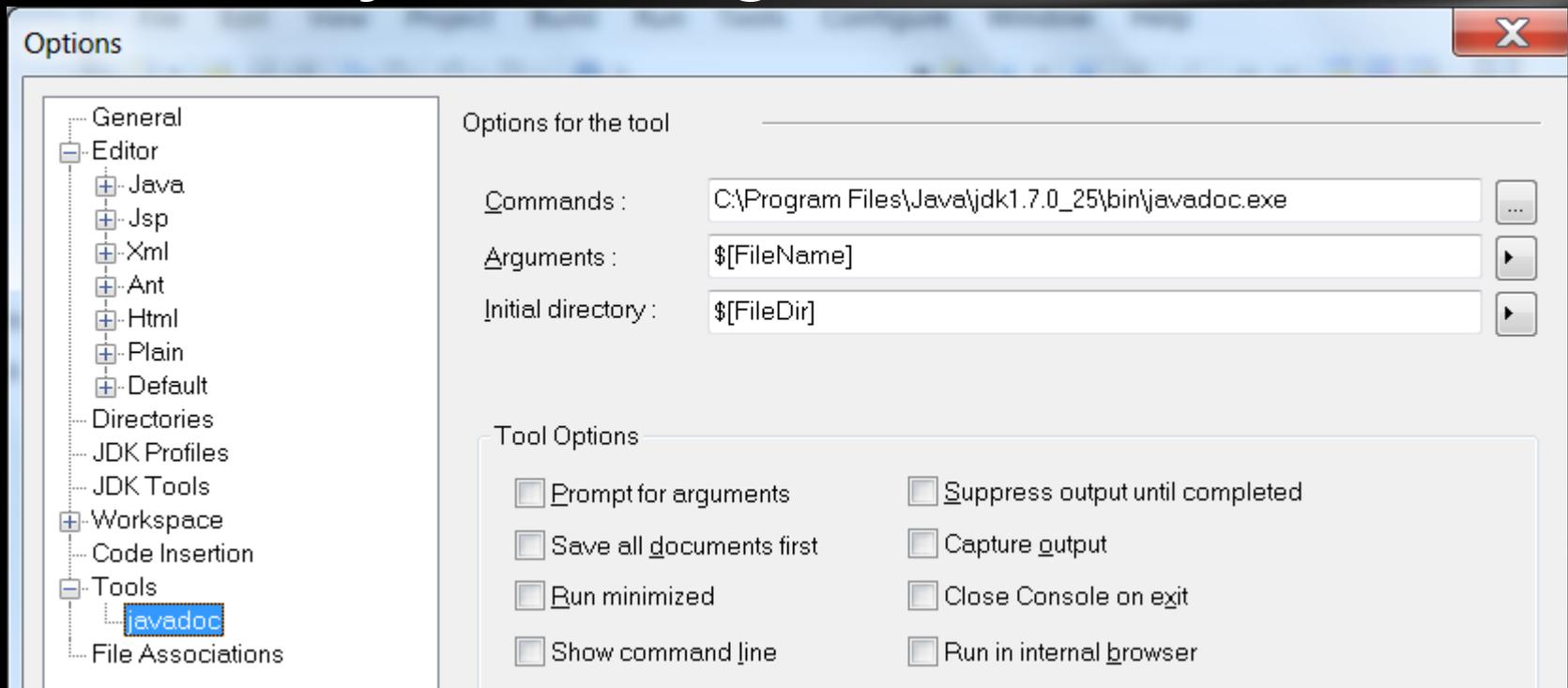
JavaDocs for JCreator

- Inside the **bin** folder is a file called javadoc.exe...double click on it and you'll see a new tool appear in JCreator



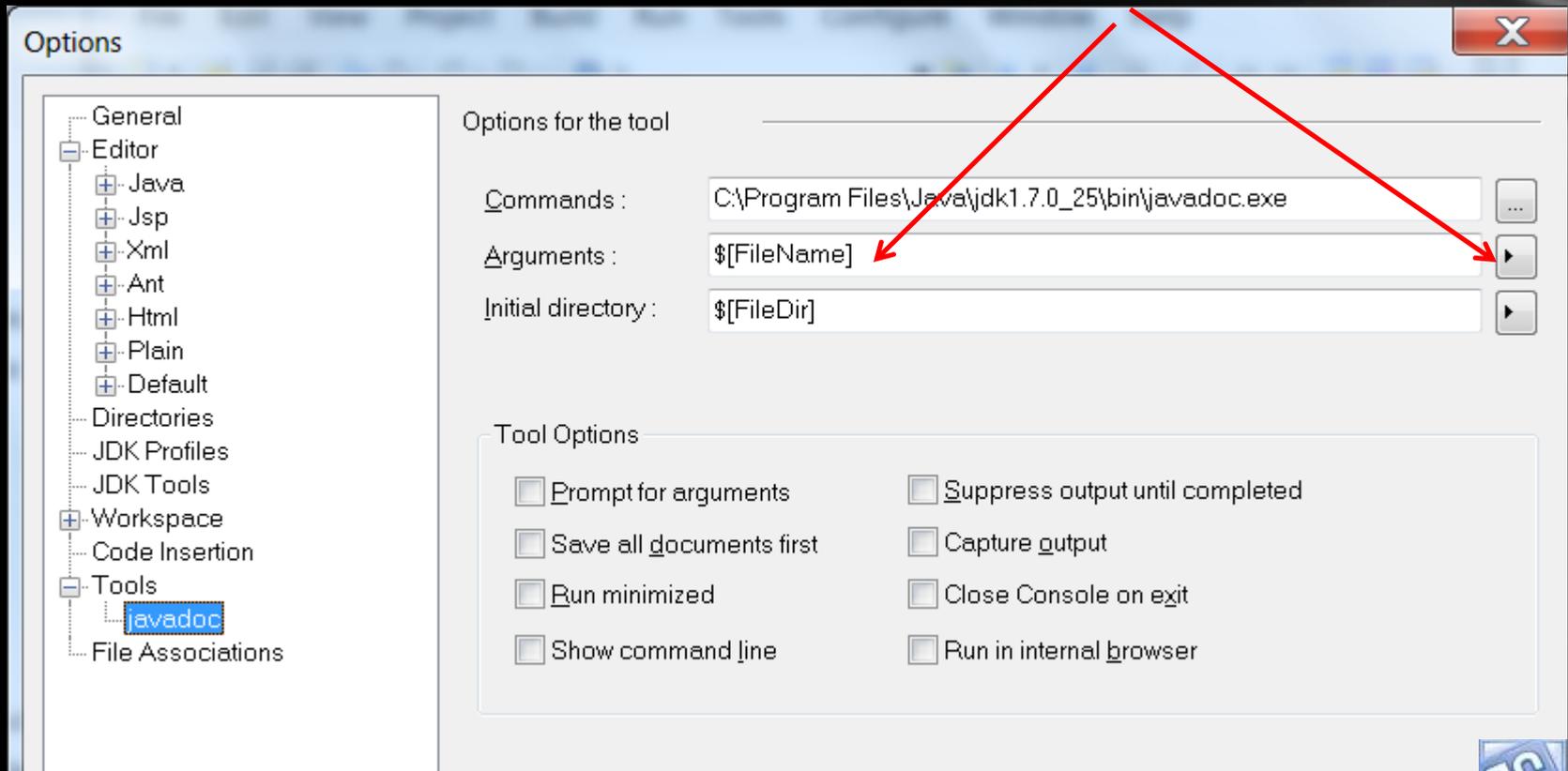
JavaDocs for JCreator

- To complete the configuration of this tool, click on Configure, Options, Tools, and then javadoc to get to this window.



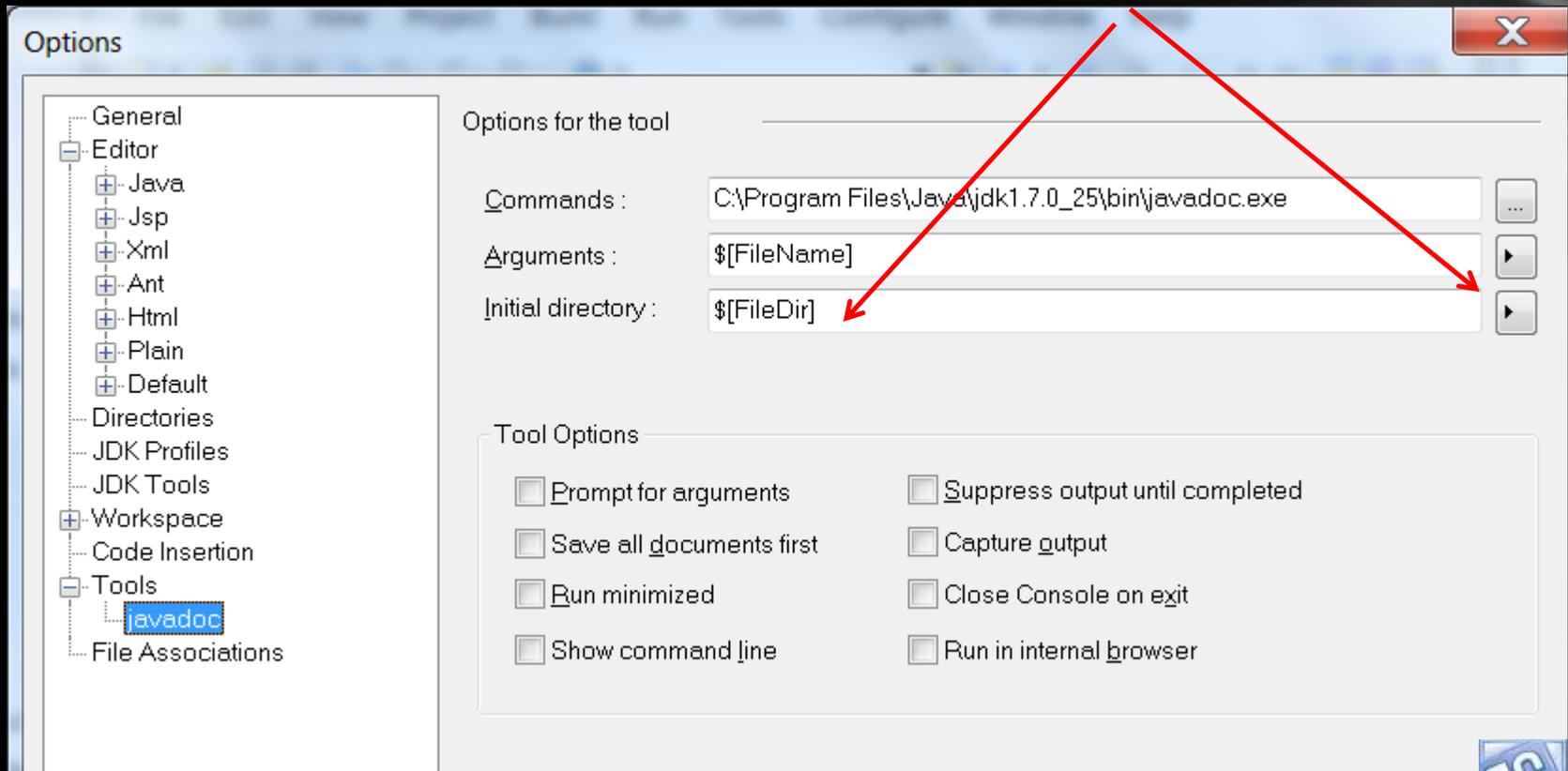
JavaDocs for JCreator

- For the Arguments window, click on the arrow button and select **File Name**

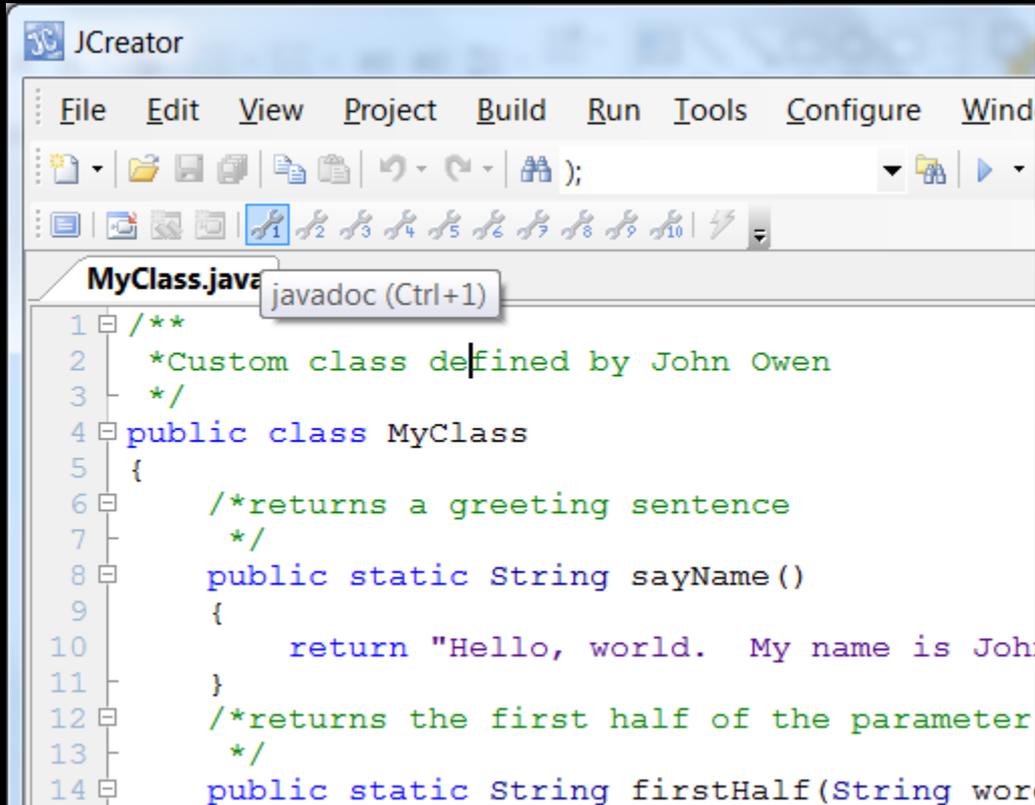


JavaDocs for JCreator

- For the Initial Directory, click on the arrow button and select **File Directory**



JavaDocs for JCreator



The screenshot shows the JCreator IDE interface. The title bar reads "JCreator". The menu bar includes "File", "Edit", "View", "Project", "Build", "Run", "Tools", "Configure", and "Wind". The toolbar contains various icons, including a wrench icon. A tooltip is displayed over the first wrench icon, containing the text "javadoc (Ctrl+1)". The main editor window shows the code for "MyClass.java":

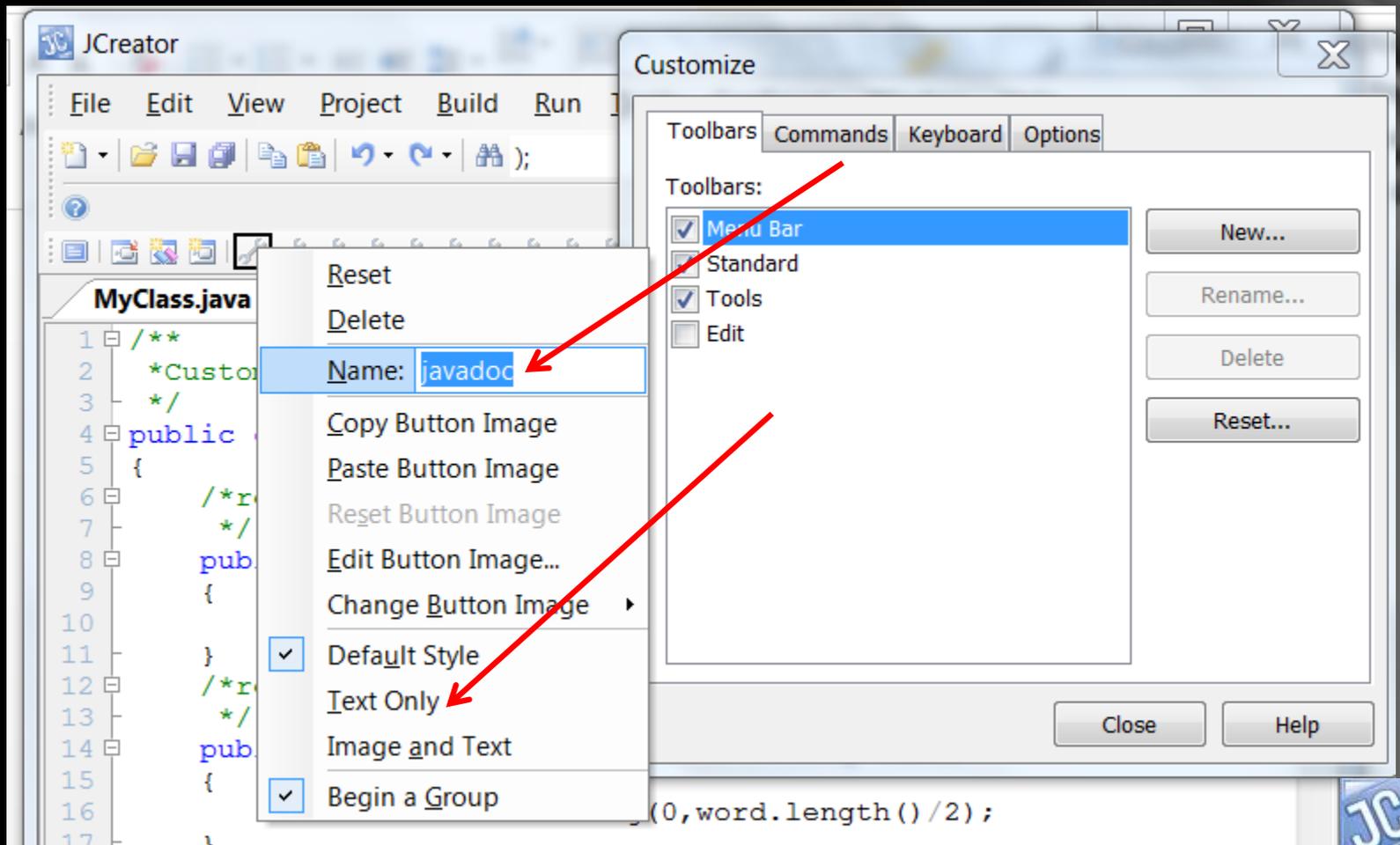
```
1  /**
2   *Custom class defined by John Owen
3   */
4  public class MyClass
5  {
6     /*returns a greeting sentence
7     */
8     public static String sayName()
9     {
10        return "Hello, world. My name is John";
11    }
12    /*returns the first half of the parameter
13    */
14    public static String firstHalf(String word)
```

Now when you hover over the first wrench, it says "javadoc" as the name of the tool.



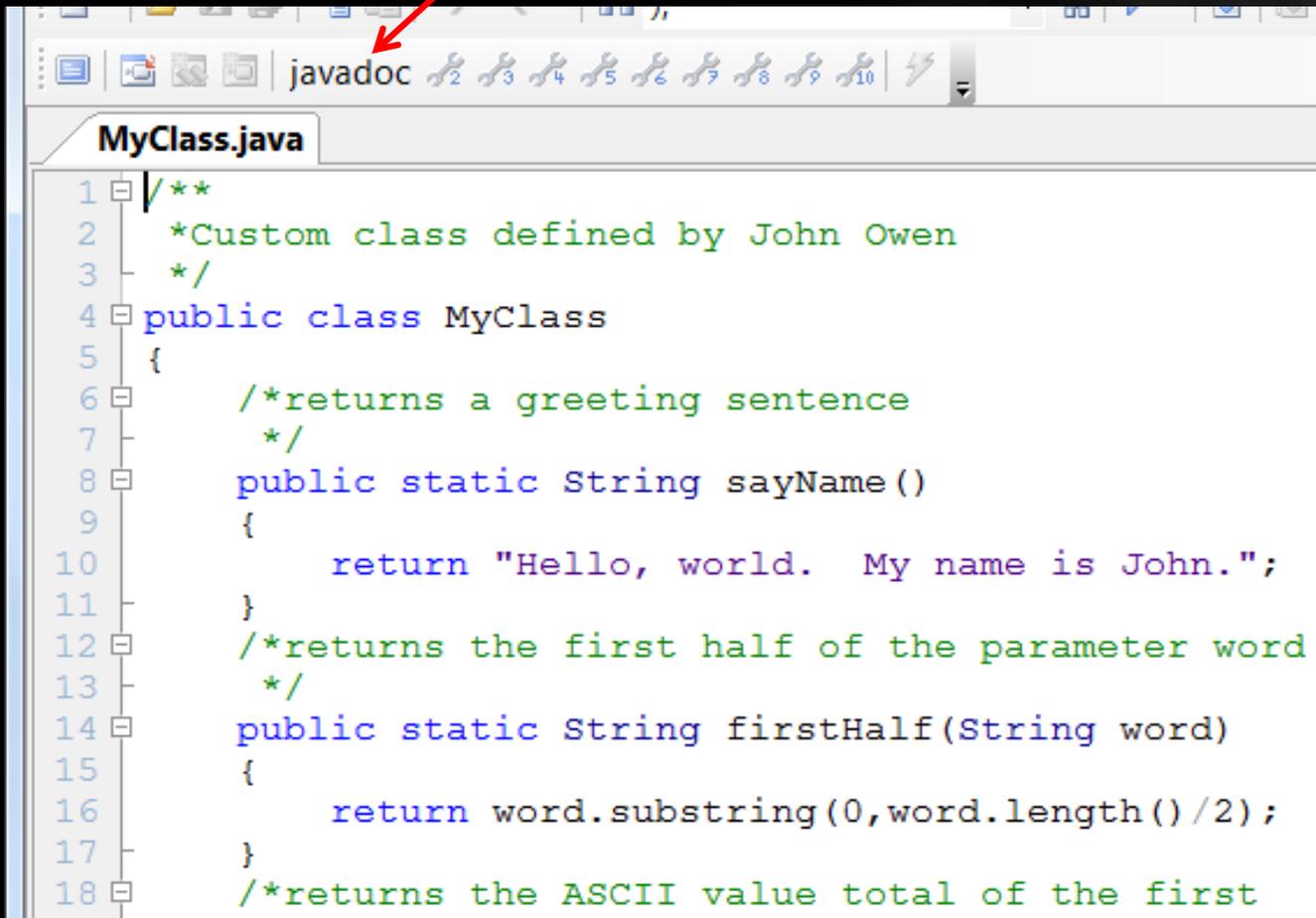
JavaDocs

To change the tool appearance from a wrench to the label "javadoc", right click once, click Customize, move that window to the side and ignore it, then right click again, change the name to "javadoc", and then click **Text Only**



JavaDocs for JCreator

Now you can see the tool name as you changed it.



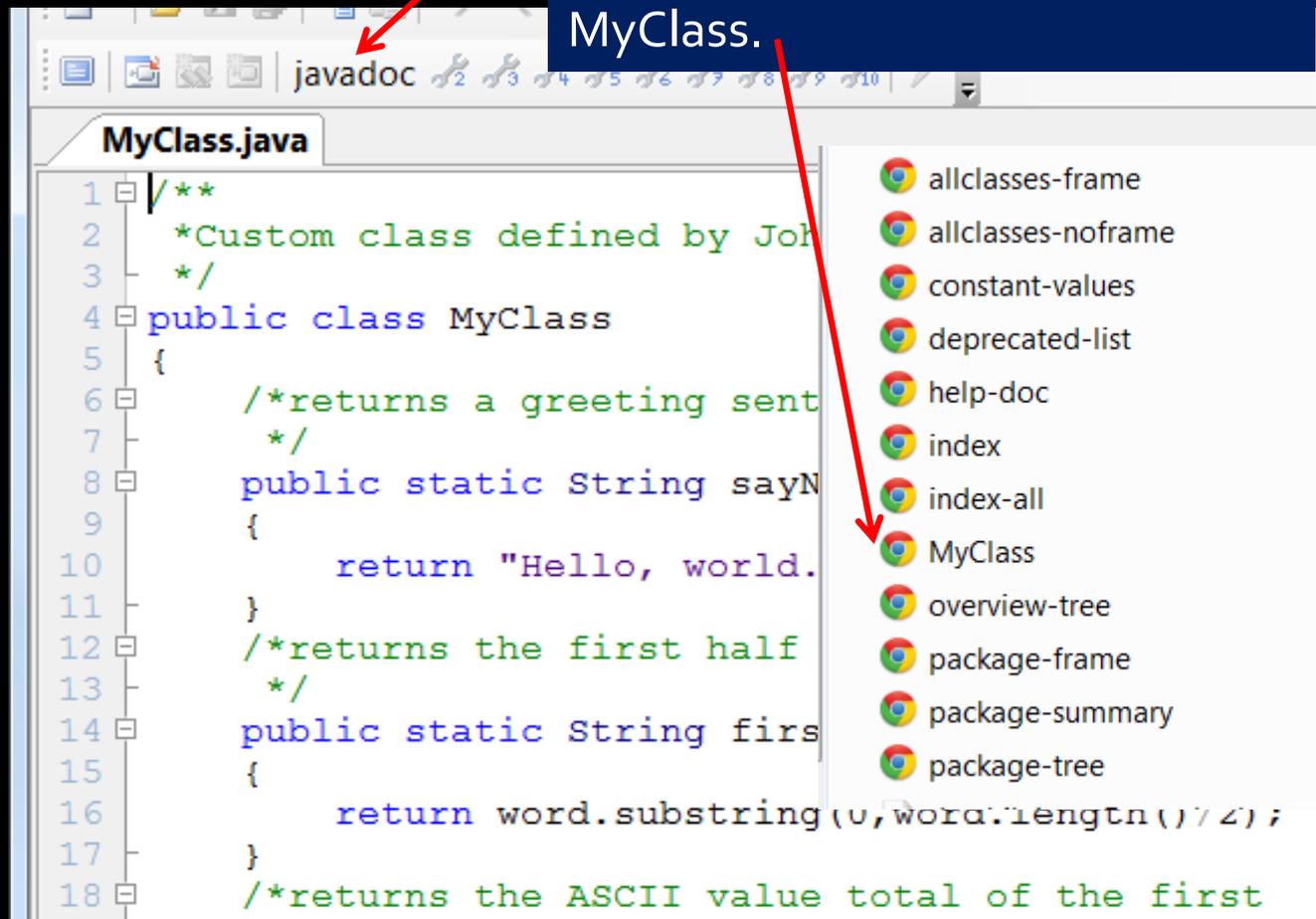
The screenshot shows the JCreator IDE interface. The toolbar at the top contains several icons, including a 'javadoc' icon which is highlighted with a red arrow. Below the toolbar, the editor window displays the code for 'MyClass.java'. The code includes a class definition with two methods: 'sayName()' and 'firstHalf(String word)'. The code is as follows:

```
1  /**
2   *Custom class defined by John Owen
3   */
4  public class MyClass
5  {
6     /*returns a greeting sentence
7     */
8     public static String sayName ()
9     {
10        return "Hello, world.  My name is John.";
11    }
12    /*returns the first half of the parameter word
13    */
14    public static String firstHalf(String word)
15    {
16        return word.substring(0,word.length()/2);
17    }
18    /*returns the ASCII value total of the first
```



JavaDocs for JCreator

When you click on it, a series of html files will be created, as shown below, including one for MyClass.



The screenshot shows the JCreator IDE with a 'javadoc' window open. The main editor displays the source code for 'MyClass.java'. The javadoc window shows a list of generated HTML files, with 'MyClass' selected. A red arrow points from the text box to the 'javadoc' window, and another red arrow points from the text box to the 'MyClass' entry in the file list.

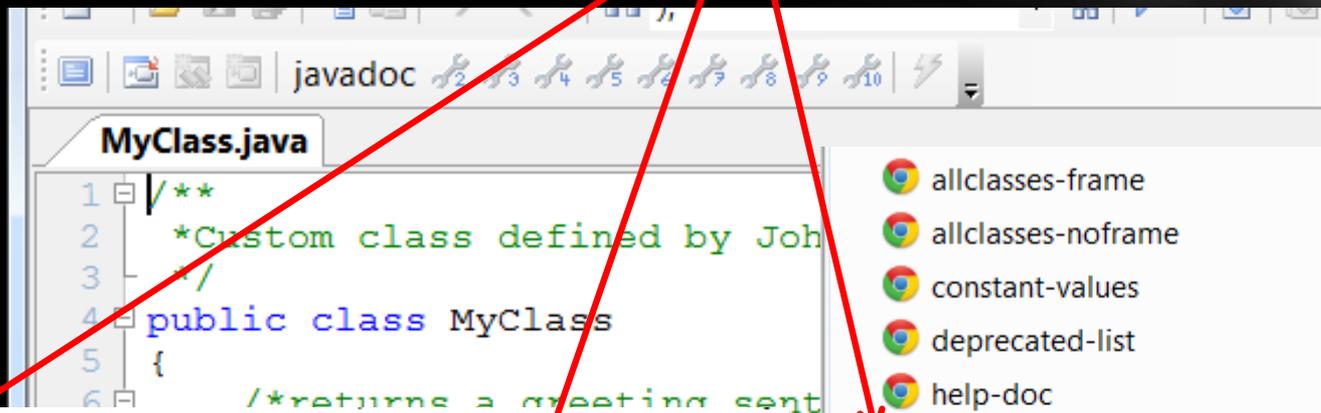
```
1  /**
2   *Custom class defined by Job
3   */
4  public class MyClass
5  {
6   /*returns a greeting sent
7   */
8   public static String sayN
9   {
10    return "Hello, world.
11   }
12  /*returns the first half
13  */
14  public static String firs
15  {
16    return word.substring(0,word.length()/2);
17  }
18  /*returns the ASCII value total of the first
```

- allclasses-frame
- allclasses-noframe
- constant-values
- deprecated-list
- help-doc
- index
- index-all
- MyClass
- overview-tree
- package-frame
- package-summary
- package-tree



JavaDocs for JCre

Open the one called "index" for the main page to see the Java Documentation for this file.



```
1 | /**
2 |  *Custom class defined by John Owen
3 |  */
4 | public class MyClass
5 | {
6 |     /**returns a greeting sent
```

All Classes
MyClass

Package	Class	Tree	Deprecated	Index	Help
java.lang	Object				
	MyClass				

Class MyClass

java.lang.Object
MyClass

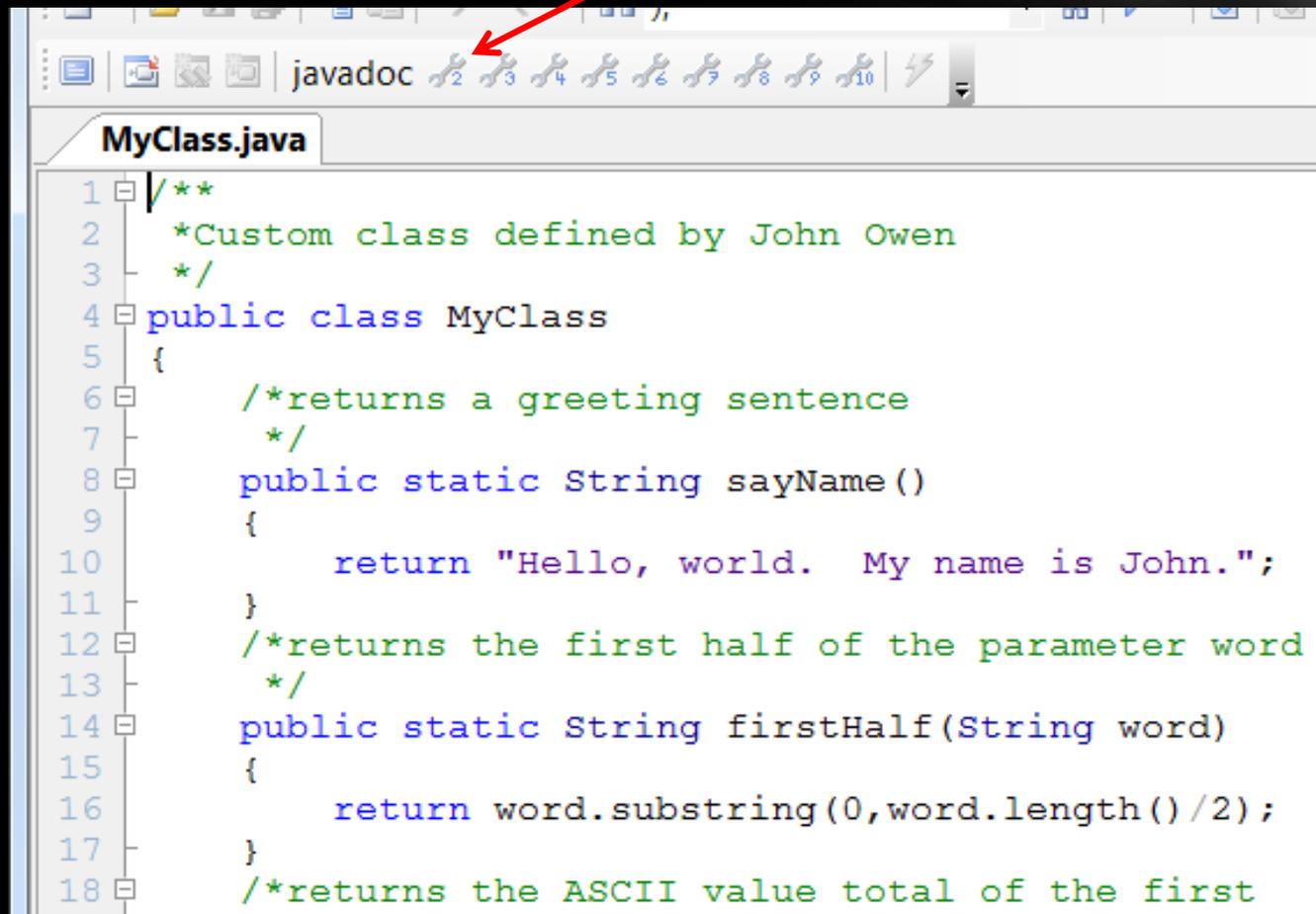
```
public class MyClass
extends java.lang.Object
```

Custom class defined by John Owen

- allclasses-frame
- allclasses-noframe
- constant-values
- deprecated-list
- help-doc
- index
- index-all
- MyClass
- overview-tree
- package-frame
- package-summary
- package-tree

JavaDocs fo

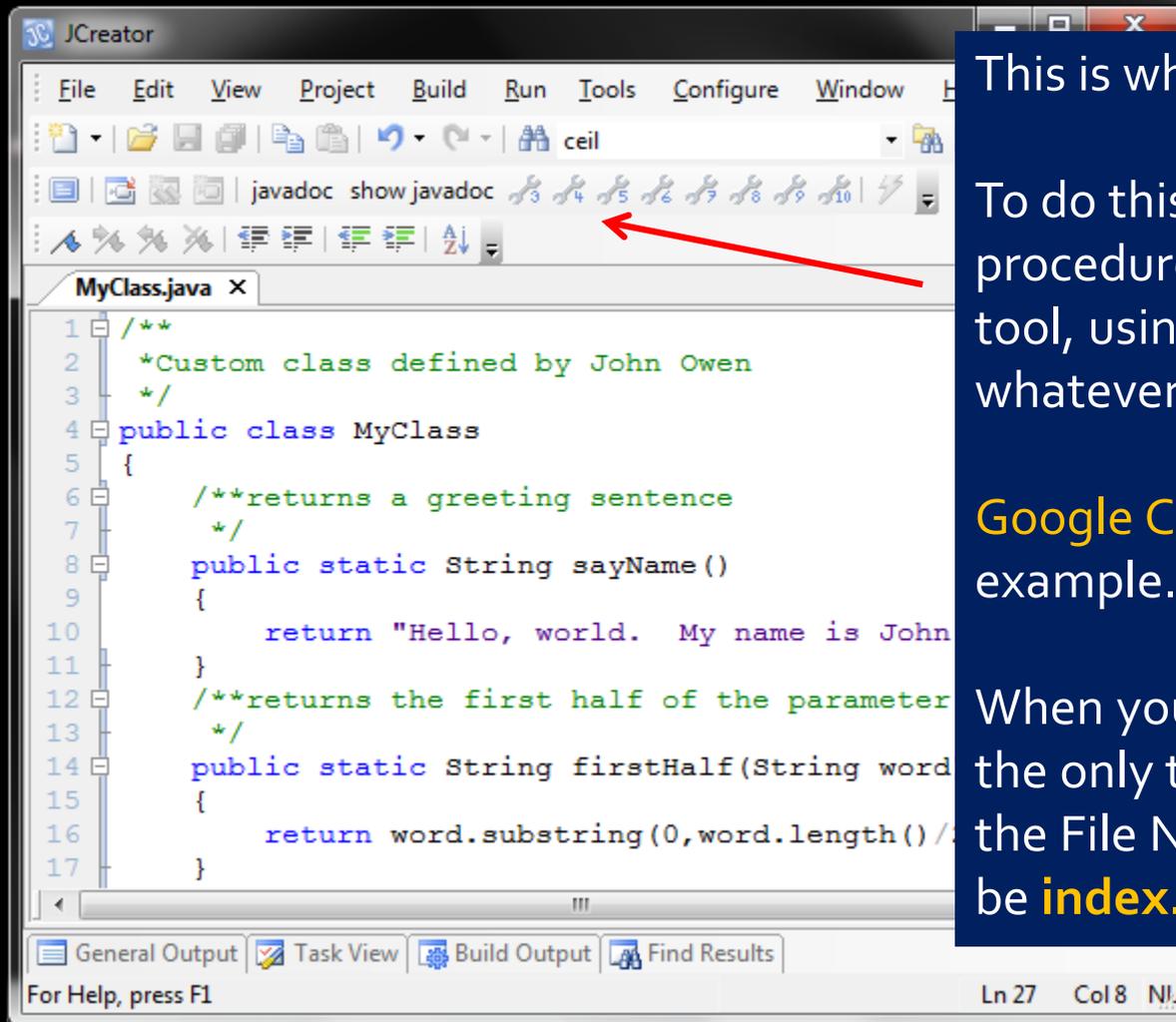
Now we'll create a second tool that can be clicked to automatically bring up the JavaDoc for the current file. We'll call it **show JavaDoc**



```
1  /**
2   *Custom class defined by John Owen
3   */
4  public class MyClass
5  {
6     /*returns a greeting sentence
7     */
8     public static String sayName ()
9     {
10        return "Hello, world.  My name is John.";
11    }
12    /*returns the first half of the parameter word
13    */
14    public static String firstHalf(String word)
15    {
16        return word.substring(0,word.length()/2);
17    }
18    /*returns the ASCII value total of the first
```



JavaDocs for JCreator



This is what it should look like.

To do this, follow the same procedure you did for the first tool, using the program for whatever browser you prefer.

Google Chrome is used for this example.

When you get to the Options, the only thing different will be the File Name, which should be **index.html**



JavaDocs for JCreator

The image shows two screenshots from the JCreator IDE. The top screenshot shows the main window with the 'javadoc show javadoc' toolbar icons highlighted by a red arrow. A blue box with the text 'Like this...' is positioned next to the arrow. The bottom screenshot shows the 'Options' dialog for the 'javadoc' tool. A red arrow points from the 'chrome' sub-option in the 'Tools' tree to the 'Arguments' field, which contains 'index.html'. Another red arrow points from the 'chrome' sub-option to the 'Commands' field, which contains 'C:\Program Files (x86)\Google\Chrome\Application\chrome.exe'. The 'Tool Options' section includes checkboxes for 'Prompt for arguments', 'Save all documents first', 'Run minimized', 'Show command line', 'Suppress output until completed', 'Capture output', 'Close Console on exit' (checked), and 'Run in internal browser'.

Like this...

Options

Options for the tool

Commands : C:\Program Files (x86)\Google\Chrome\Application\chrome.exe

Arguments : index.html

Initial directory : \${FileDir}

Tool Options

- Prompt for arguments
- Save all documents first
- Run minimized
- Show command line
- Suppress output until completed
- Capture output
- Close Console on exit
- Run in internal browser

Tools

- javadoc
- chrome



JavaDocs for

To run the JavaDoc tool, simply click on it in the JCreator window.

It will automatically generate an html help file system for the current file in the current directory.

The screenshot shows the JCreator IDE with the 'MyClass.java' file open. The 'javadoc' button in the toolbar is highlighted with a red arrow. A terminal window in the foreground displays the output of the javadoc command, showing the generation of various HTML files and a stylesheet. A file explorer window on the right shows the generated files, with a red arrow pointing to the 'help-doc' file.

```
MyClass.java x
1 /**
2  *Custom class defined by John Owen
3  */
4 public class MyClass
5 {
6     /**returns a green
7     */
8     public static String
9     {
10        return "Hello
11    }
12    /**returns the file
13    */
14    public static String
15    {
16        return word.s
17    }
```

```
C:\Program Files (x86)\Xinox Software\JCreatorV5LE\GE2001.exe
Loading source file MyClass.java...
Constructing Javadoc information...
Standard Doclet version 1.6.0_24
Building tree for all the packages and classes...
Generating MyClass.html...
Generating package-frame.html...
Generating package-summary.html...
Generating package-tree.html...
Generating constant-values.html...
Building index for all the packages and classes...
Generating overview-tree.html...
Generating index-all.html...
Generating deprecated-list.html...
Building index for all classes...
Generating allclasses-frame.html...
Generating allclasses-noframe.html...
Generating index.html...
Generating help-doc.html...
Generating stylesheet.css...
Press any key to continue...
```

- resources
- stylesheet
- MyClass.class
- package-list
- allclasses-frame
- allclasses-noframe
- constant-values
- deprecated-list
- help-doc
- index
- index-all
- MyClass
- overview-tree
- package-frame
- package-summary
- package-tree



JavaDocs for JCreator

When the "show javadoc" tool is clicked, your web browser opens the help file that was just created.

The screenshot shows the JCreator IDE with a Java class named `MyClass.java` open. The code includes a class comment and two static methods. A toolbar button labeled "show javadoc" is highlighted with a red arrow. A web browser window titled "MyClass" is open in the foreground, displaying the generated Javadoc. The browser window shows navigation links for Package, Class, Tree, Deprecated, Index, and Help. The main content of the browser displays the class name "Class MyClass", its inheritance from `java.lang.Object`, and the class comment "Custom class defined by John Owen". A "Constructor Summary" section is visible at the bottom of the browser window.

```
1 /**
2  *Custom class defined by John Owen
3  */
4 public class MyClass
5 {
6     /**returns a greet
7     */
8     public static Stri
9     {
10         return "Hello,
11     }
12     /**returns the fir
13     */
14     public static Stri
15     {
16         return word.su
17     }
```

Package **Class** Tree Deprecated Index Help
PREV CLASS NEXT CLASS [FRAMES](#) [NO FRAMES](#) [All Classes](#)
SUMMARY: NESTED | FIELD | [CONSTR](#) | [METHOD](#) DETAIL: FIELD | [CONSTR](#) | [METHOD](#)

Class MyClass

java.lang.Object
└─ MyClass

```
public class MyClass
extends java.lang.Object
```

Custom class defined by John Owen

Constructor Summary



JavaDocs for JCreator

You now have a complete help file of all the work you have done so far!!!

Cool, huh!!!!???

The screenshot shows the JCreator IDE with the following components:

- Code Editor (MyClass.java):**

```
1  /**
2   *Custom class defined by John Owen
3   */
4  public class MyClass
5  {
6     /**returns a greet
7     */
8     public static String
9     {
10        return "Hello,
11    }
12    /**returns the fir
13    */
14    public static String
15    {
16        return word.su
17    }
```
- JavaDoc Window (MyClass):**
 - Navigation: [Package](#) **[Class](#)** [Tree](#) [Deprecated](#) [Index](#) [Help](#)
 - Links: [PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#) [All Classes](#)
 - Summary: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)
 - Detail: [FIELD](#) | [CONSTR](#) | [METHOD](#)
 - Class MyClass**
 - java.lang.Object
└─ MyClass
 - public class **MyClass**
extends java.lang.Object
 - Custom class defined by John Owen
 - Constructor Summary**

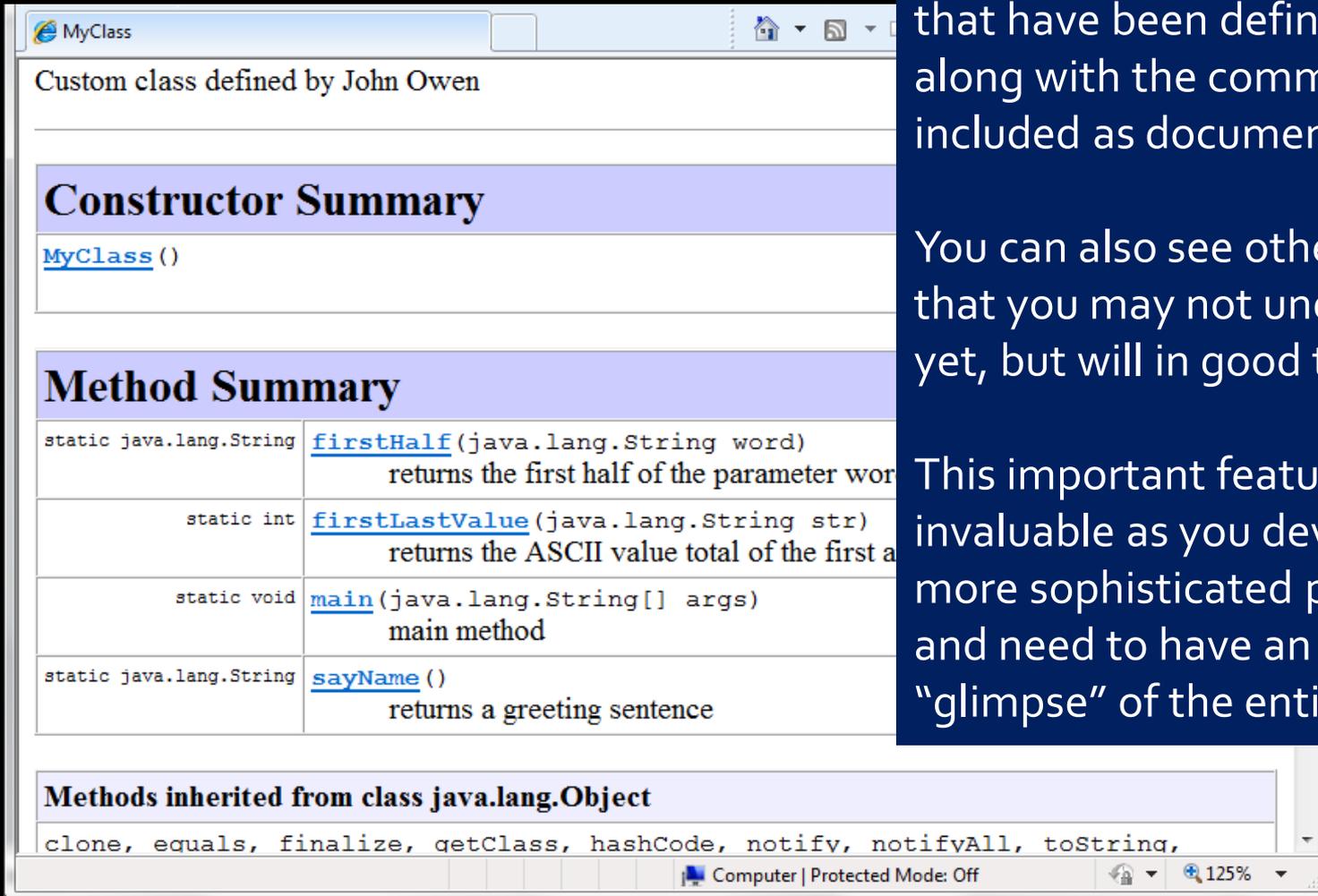


JavaDocs for JCreator

You can see the four methods that have been defined so far, along with the comments included as documentation.

You can also see other “stuff” that you may not understand yet, but will in good time.

This important feature is invaluable as you develop more sophisticated projects and need to have an organized “glimpse” of the entire project.



The screenshot shows the JavaDoc for a class named `MyClass`. The window title is "MyClass". The text "Custom class defined by John Owen" is displayed at the top. Below this, there are three main sections:

- Constructor Summary**: Shows the constructor `MyClass ()`.
- Method Summary**: A table listing four methods:

static java.lang.String	firstHalf (java.lang.String word) returns the first half of the parameter word
static int	firstLastValue (java.lang.String str) returns the ASCII value total of the first a
static void	main (java.lang.String[] args) main method
static java.lang.String	sayName () returns a greeting sentence
- Methods inherited from class java.lang.Object**: Lists `clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, and `toString`.

The bottom of the window shows the system tray with "Computer | Protected Mode: Off" and a zoom level of "125%".



Lesson Summary

- In this lesson, you learned how to create your own class and define your own static return methods.
- You also learned how to create your own help file using the JavaDoc utility within the JCreator IDE.



Labs

- Now you will add more utility methods to your class, some String related, and some Math related.
- When possible, use methods already defined to help write the new method, either from the Math class, String class, or one previously defined from your own class.



Method 1 - *upperLower(String word)*

WAM (write a method) called *upperLower* that returns a received String with the first half uppercased and the last half lowercased.

```
MyClass.java myClass1.in x
1 rhinoceros
2 elephants
3 dogs
4 antidisestablishmentarianism
5
```

```
59 *and the last half lowercased
60 */
61 public static String upperLower(String word)
```

```
63 rhinoceros ==> RHINOCeros
64 elephants ==> ELEPhants
65 dogs ==> DOgs
66 antidisestablishmentarianism ==> ANTIDISESTABLishmentarianism
67 Press any key to continue
```

```
68
69 public static void main (String [] args) throws IOException
70 {
71     Scanner f = new Scanner(new File("myClass1.in"));
72     String str;
73     while (f.hasNext())
74     {
75         str=f.next();
76         System.out.printf("%s ==> %s\n",
77             str,upperLower(str));
78     }
79 }
80 }
```



Method 2 –

diffWords(String one, String two)

WAM called *diffWords* that receives two Strings and returns an integer that represents the positive difference between the lengths of two Strings.

```
MyClass.java  myClass2.in x
1 rhinoceros
2 elephants
3 dogs
4 antidisestablishmentarianism
5
```

```
MyClass.java x  myClass2.in
59  /**returns an integer that represents the positive difference
60  *between the lengths of two words
61  */
62  public static int diffWords(String one, String two)
63
64
65
66
67
68
69  public static void main (String [] args) throws IOException
70  {
71      Scanner f = new Scanner(new File("myClass2.in"));
72      String s1,s2;
73      while(f.hasNext())
74      {
75          s1=f.next();
76          s2=f.next();
77          System.out.printf("%s %s Length difference = %s\n",
78                          s1,s2,diffWords(s1,s2));
79      }
80  }
81
```

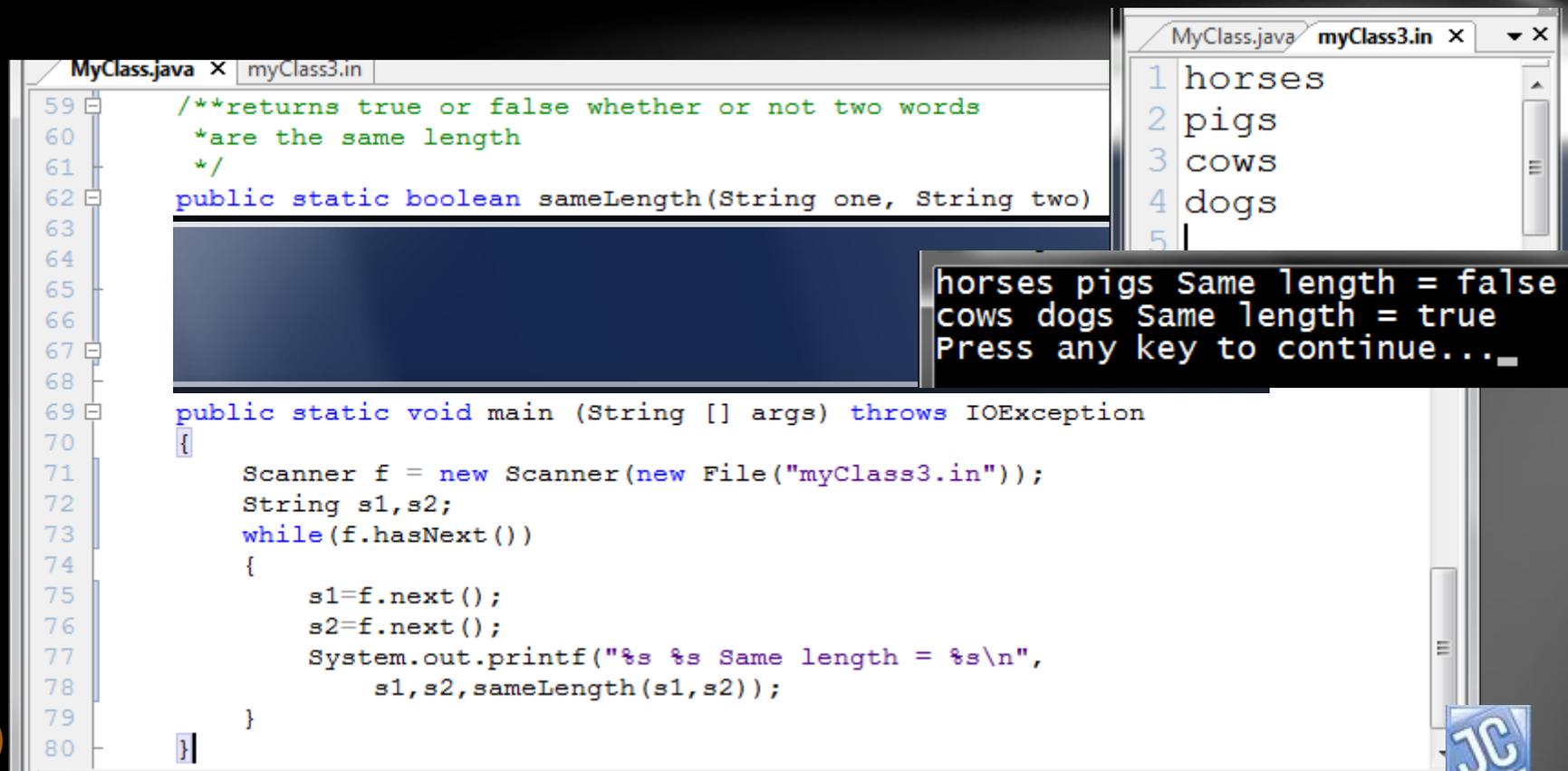
```
rhinoceros elephants Length difference = 1
dogs antidisestablishmentarianism Length difference = 24
Press any key to continue..._
```



Method 3 –

sameLength(String one, String two)

WAM called *sameLength* that receives two Strings and returns a boolean value to indicate if two words are the same length.



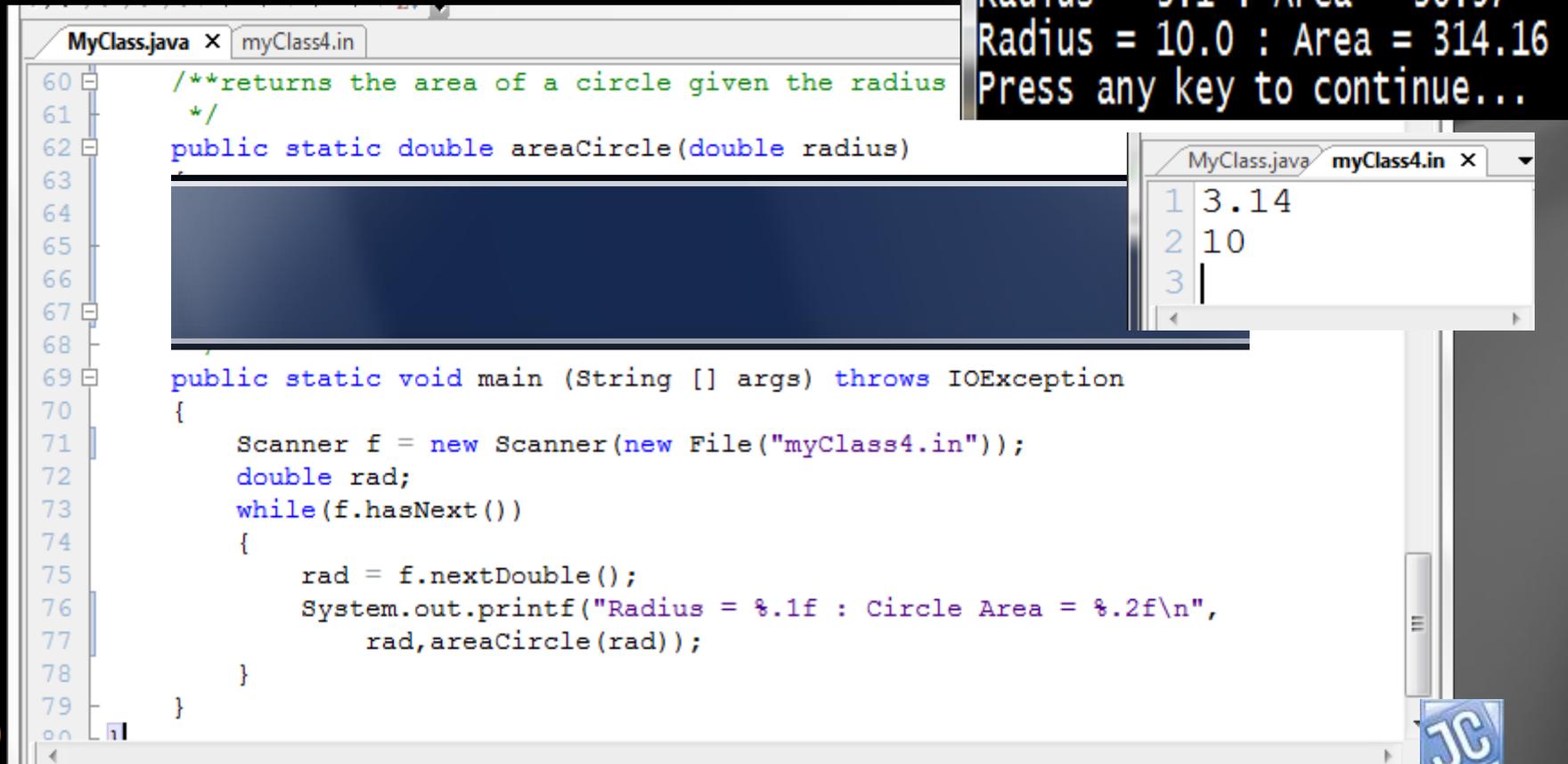
```
MyClass.java x myClass3.in
59  /**returns true or false whether or not two words
60  *are the same length
61  */
62  public static boolean sameLength(String one, String two)
63
64
65
66
67
68
69  public static void main (String [] args) throws IOException
70  {
71      Scanner f = new Scanner(new File("myClass3.in"));
72      String s1,s2;
73      while (f.hasNext ())
74      {
75          s1=f.next ();
76          s2=f.next ();
77          System.out.printf("%s %s Same length = %s\n",
78                          s1,s2,sameLength (s1,s2) );
79      }
80  }
```

```
MyClass.java x myClass3.in x
1 horses
2 pigs
3 cows
4 dogs
5
horses pigs Same length = false
cows dogs Same length = true
Press any key to continue...
```



Method 4 – *areaCircle(double radius)*

WAM called *areaCircle* that receives a radius as a double value and returns the area of a circle with the given radius.



```
MyClass.java x myClass4.in
60 /**returns the area of a circle given the radius
61 */
62 public static double areaCircle(double radius)
63
64
65
66
67
68
69 public static void main (String [] args) throws IOException
70 {
71     Scanner f = new Scanner(new File("myClass4.in"));
72     double rad;
73     while(f.hasNext())
74     {
75         rad = f.nextDouble();
76         System.out.printf("Radius = %.1f : Circle Area = %.2f\n",
77             rad,areaCircle(rad));
78     }
79 }
80
```

```
Radius = 3.1 : Area = 30.97
Radius = 10.0 : Area = 314.16
Press any key to continue...
```

```
MyClass.java myClass4.in x
1 3.14
2 10
3
```



Method 5 – *surfAreaSphere(double radius)*

WAM called *surfAreaSphere* that receives a radius and returns the surface area of a sphere with the given radius. Use the *areaCircle* method you wrote previously to help with this one.

```
MyClass.java x myClass5.in
60 /**returns the surface area of a sphere given the radius
61 */
62 public static double surfAreaSphere(double radius)
63 {
64
65
66
67
68
69 public static void main (String [] args) throws IOException
70 {
71     Scanner f = new Scanner(new File("myClass5.in"));
72     double rad;
73     while (f.hasNext ())
74     {
75         rad = f.nextDouble ();
76         System.out.printf ("Radius = %.1f : Surface Area = %.2f\n",
77                             rad, surfAreaSphere (rad) );
78     }

```

```
MyClass.java x myClass5.in x
1 3.14
2 10
3 |

```

```
Radius = 3.1 : Surface Area = 123.90
Radius = 10.0 : Surface Area = 1256.64
Press any key to continue...

```



Method 6 –

delta(int num1, int num2)

WAM called *delta* that returns the difference between two received integers. *Delta* “ Δ ” is the Greek letter mathematicians use to refer to *difference*.

```
MyClass.java x myClass6.in
74 /**returns the difference between two integers
75 */
76 public static int delta(int num1, int num2)
77
78
79
80
81
82 public static void main (String [] args) throws IOException
83 {
84     Scanner f = new Scanner(new File("myClass6.in"));
85     int n1,n2;
86     while(f.hasNext())
87     {
88         n1=f.nextInt();
89         n2=f.nextInt();
90         System.out.printf("delta(%d,%d)==> %d\n",
91             n1,n2,delta(n1,n2));
92     }
93 }
94 }
```

```
MyClass.java x myClass6.in x
1 3 5 6 10
2 -6 3 -6 7
3 4 -5 1 -5
```

```
delta(3,5)==> -2
delta(6,10)==> -4
delta(-6,3)==> -9
delta(-6,7)==> -13
delta(4,-5)==> 9
delta(1,-5)==> 6
Press any key to contin
```



Method 7 –

slope(int x1,int y1, int x2, int y2)

WAM called *slope* that returns the slope of a line as a String, given four integers representing two points. If the slope is undefined, return the word “undefined”, otherwise return a decimal value formatted to 1 decimal place, but as a String.

Hint: You will need to use an if statement and the String.format command (see Lesson 1C).

```
MyClass.java  myClass7.in x
1 3 5 6 10
2 3 5 6 -10
3 -6 3 -6 7
4 4 -5 1 -5
5 |
```

```
For the line passing through the points
(3,5) and (6,10), the slope is 1.7
For the line passing through the points
(3,5) and (6,-10), the slope is -5.0
For the line passing through the points
(-6,3) and (-6,7), the slope is undefined
For the line passing through the points
(4,-5) and (1,-5), the slope is 0.0
Press any key to continue...
```



Method 7 –

slope(int x1,int y1, int x2, int y2)

```
MyClass.java x myClass7.in
72  /**returns as a String the slope of a line given four integers
73  *representing two points in a coordinate plane,
74  *formatted to one decimal place.
75  *Return "undefined" if appropriate
76  */
77  public static String slope(int x1,int y1, int x2, int y2)
78  {
79  }
80
81
82  }
83  /
84
85  public static void main (String [] args) throws IOException
86  {
87      Scanner f = new Scanner(new File("myClass7.in"));
88      int x1,y1,x2,y2;
89      while(f.hasNext())
90      {
91          x1=f.nextInt();
92          y1=f.nextInt();
93          x2=f.nextInt();
94          y2=f.nextInt();
95          System.out.printf("For the line passing through the points\n"
96              +"(%d,%d) and (%d,%d), the slope is %s\n",
97              x1,y1,x2,y2, slope(x1,y1,x2,y2));
98      }
99  }
100 }
```



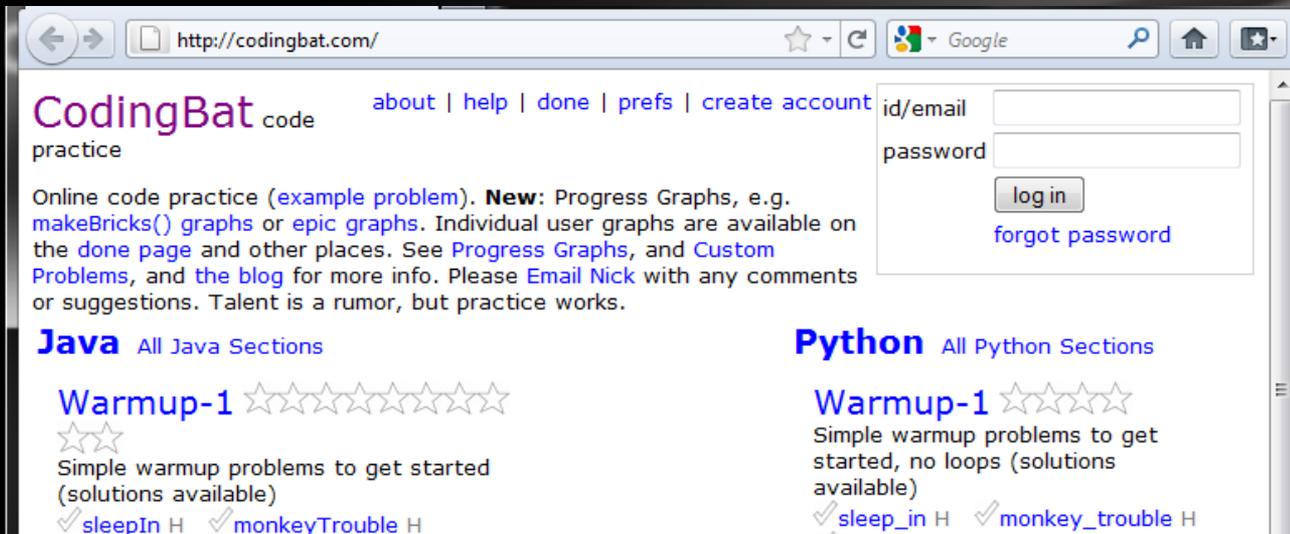
JavaDoc Review

- Once you have completed these first seven methods, run the JavaDoc tool and see the documentation for the work you did.
- Consider revising the documentation if you feel it needs clarification or adjustment.



CodingBat Labs

- These next three methods will be submitted using a Java practice web site called codingbat.com.

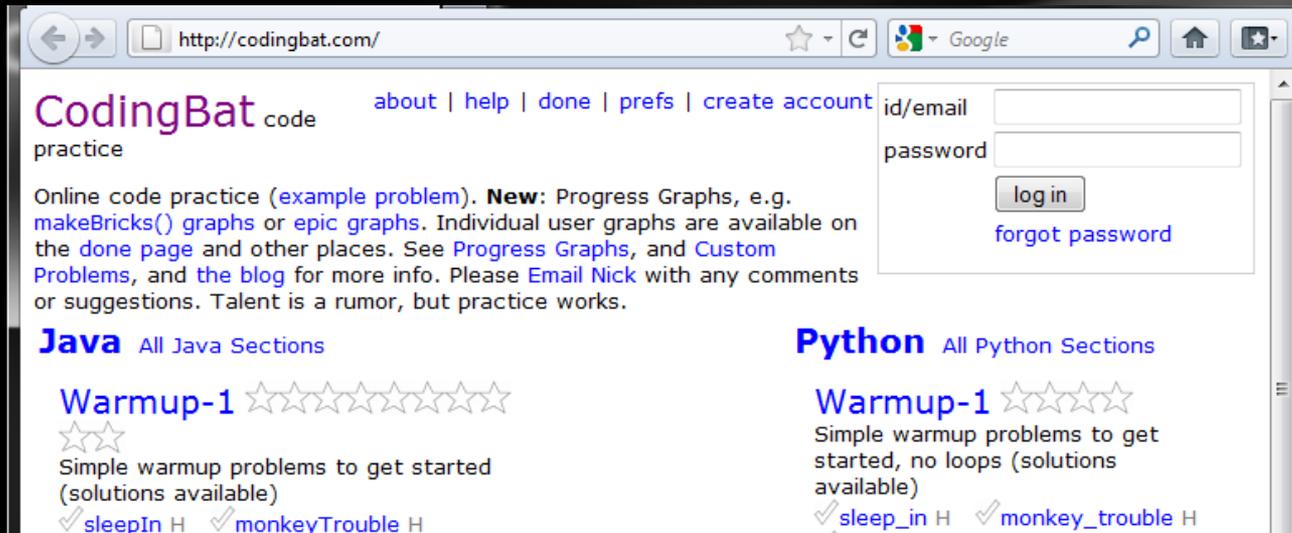


The screenshot shows the CodingBat website homepage. The browser address bar displays "http://codingbat.com/". The page header includes the site name "CodingBat code practice" and navigation links: "about | help | done | prefs | create account". A login form on the right contains fields for "id/email" and "password", a "log in" button, and a "forgot password" link. The main content area features two columns of problem sections. The left column is for "Java" and lists "Warmup-1" with a 5-star rating and a description: "Simple warmup problems to get started (solutions available)". Below it are links for "sleepIn H" and "monkeyTrouble H". The right column is for "Python" and lists "Warmup-1" with a 5-star rating and a description: "Simple warmup problems to get started, no loops (solutions available)". Below it are links for "sleep_in H" and "monkey_trouble H".



CodingBat Labs

- To use this website you need to register (free) with an email address.

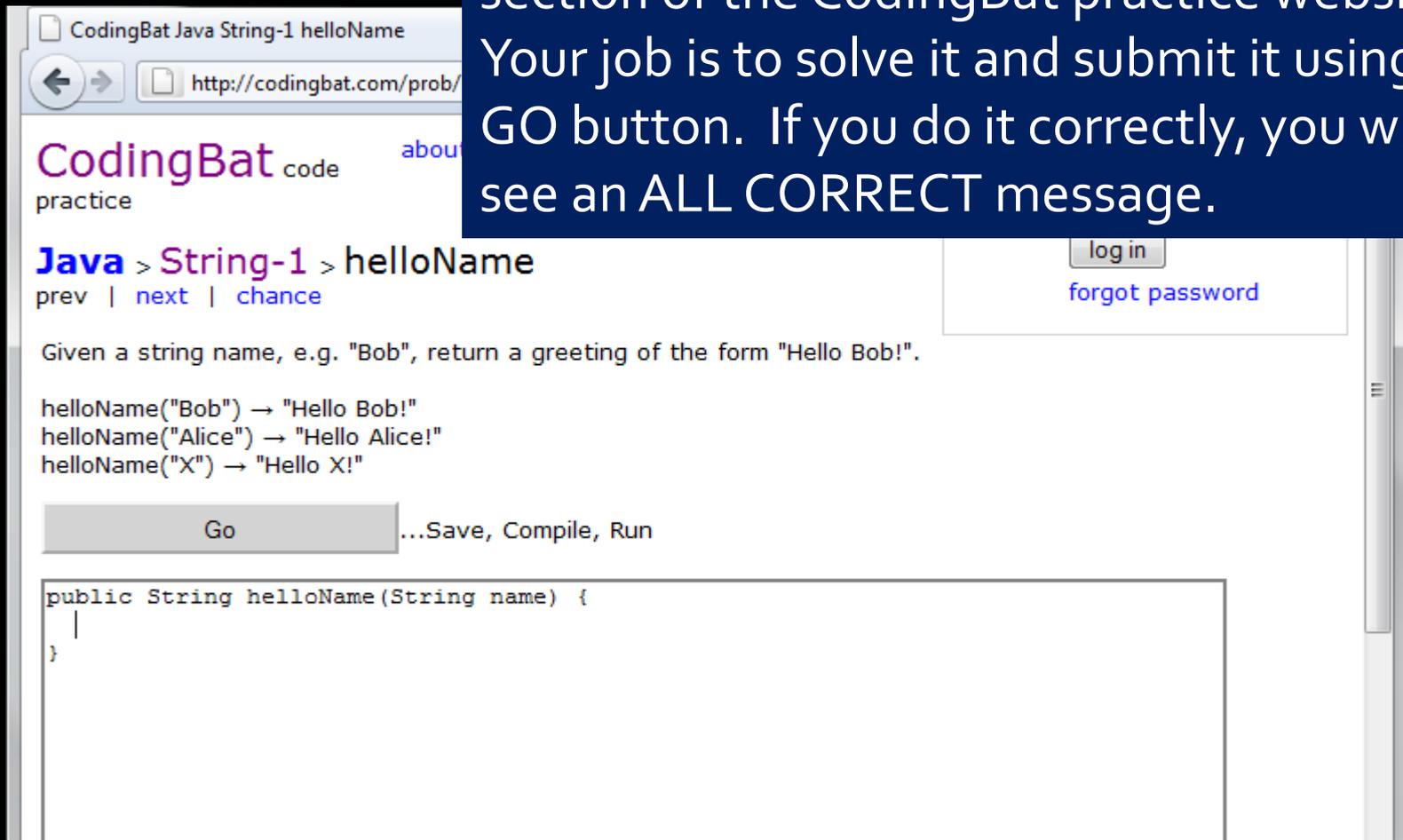


The screenshot shows the CodingBat website homepage. The browser address bar displays `http://codingbat.com/`. The page features a navigation menu with links for [about](#), [help](#), [done](#), [prefs](#), and [create account](#). The main content area includes a login form with fields for `id/email` and `password`, a `log in` button, and a [forgot password](#) link. Below the login form, there is a section for **Java** with a link to [All Java Sections](#) and a featured problem **Warmup-1** with a 5-star rating. The description for **Warmup-1** is "Simple warmup problems to get started (solutions available)" and it lists two problems: `sleepIn` and `monkeyTrouble`. A similar section for **Python** is also visible, with a link to [All Python Sections](#) and a featured **Warmup-1** problem with a 5-star rating, described as "Simple warmup problems to get started, no loops (solutions available)", listing `sleep_in` and `monkey_trouble`.



Method 8 - *helloName*

This is the first example in the `String-1` section of the CodingBat practice website. Your job is to solve it and submit it using the GO button. If you do it correctly, you will see an ALL CORRECT message.



CodingBat Java String-1 helloName

http://codingbat.com/prob/

CodingBat code about
practice

Java > String-1 > helloName
prev | next | chance

log in
forgot password

Given a string name, e.g. "Bob", return a greeting of the form "Hello Bob!".

helloName("Bob") → "Hello Bob!"
helloName("Alice") → "Hello Alice!"
helloName("X") → "Hello X!"

Go ...Save, Compile, Run

```
public String helloName(String name) {  
    |  
}
```



Method 8 - *helloName*



CodingBat Java String-1 helloName

http://codingbat.com/prob/p171896

CodingBat code about | help | done | prefs | create account

practice

Java > String-1 > helloName

prev | next | chance

Given a string name, e.g. "Bob", return a greeting of the form "Hello Bob!".

helloName("Bob") → "Hello Bob!"
helloName("Alice") → "Hello Alice!"
helloName("X") → "Hello X!"

Go ...Save, Compile, Run

```
public String helloName(String name) {  
    |  
}
```

This method is very easy, and represents the basic process CodingBat uses. Notice several important features.

- First, a description of the method and its expected outcome is given.



Method 8 - *helloName*

CodingBat Java String-1 helloName

http://codingbat.com/prob/p171896

CodingBat code about | help | done | prefs | create account

practice

Java > String-1 > helloName
prev | next | chance

Given a string name, e.g. "Bob", return a greeting of the form "Hello Bob!"

helloName("Bob") → "Hello Bob!"
helloName("Alice") → "Hello Alice!"
helloName("X") → "Hello X!"

Go ...Save Compile, Run

```
public String helloName(String name) {  
    |  
}
```

- Next, some sample data and expected output is shown.
- The method header showing the return type and parameter is provided.



Method 8 - *helloName*

CodingBat [code](#) [about](#) | [help](#) | [done](#) | [prefs](#) | [create account](#)

practice

Java > **String-1** > **helloName**

[prev](#) | [next](#) | [chance](#)

Given a string name, e.g. "Bob", return a greeting of the form "Hello Bob!".

helloName("Bob") → "Hello Bob!"
helloName("Alice") → "Hello Alice!"
helloName("X") → "Hello X!"

Go ...Save, Compile, Run

```
public String helloName(String name) {  
    |  
}
```

Your job is to enter the solution code between the curly braces.



Method 8 - *helloName*

CodingBat Java String-1 helloName

http://codingbat.com/prob/p171896

CodingBat code about | help | done | prefs | create account
practice

Java > String-1 > helloName
prev | next | chance

Given a string name, e.g. "Bob", return a greeting of the form "Hello Bob!".

helloName("Bob") → "Hello Bob!"
helloName("Alice") → "Hello Alice!"
helloName("X") → "Hello X!"

Go ...Save, Compile, Run

```
public String helloName(String name) {  
    |  
}
```

- Finally, you will click the GO button to submit your work.



Method 8 - *helloName*



CodingBat Java String-1 helloName

http://codingbat.com/prob/p171896

CodingBat code about | help | done | prefs | create account
practice

Java > String-1 > helloName
prev | next | chance

id/email
password
log in
forgot password

Given a string name, e.g. "Bob", return a greeting of the form "Hello Bob!".

helloName("Bob") → "Hello Bob!"
helloName("Alice") → "Hello Alice!"
helloName("X") → "Hello X!"

Go ...Save, Compile, Run

```
public String helloName(String name) {  
    |  
}
```

- Try it yourself first, and if you need to, take a look at the solution on the next slide.



Method 8 - *helloName*

Java > **String-1** > helloName

prev | next | chance

Given a string name, e.g. "Bob", return a greet

helloName("Bob") → "Hello Bob!"

helloName("Alice") → "Hello Alice!"

helloName("X") → "Hello X!"

Go

...Save, Compile, Run

```
public String helloName(String name) {  
    return "Hello "+name+"!";  
}
```

- The solution is a simple concatenation of the word "Hello" (note the space), the parameter *name*, and the exclamation mark.



Method 8 - *helloName*

Java > **String-1** > helloName

prev | next | chance

Given a string name, e.g. "Bob", return a greeting

helloName("Bob") → "Hello Bob!"

helloName("Alice") → "Hello Alice!"

helloName("X") → "Hello X!"

Go

...Save, Compile,

```
public String helloName(String name) {  
    return "Hello "+name+"!";  
}
```

- When you click the GO button, you will receive a report like the one shown below, hopefully with the words, "All Correct".

Expected	This Run		
helloName("Bob") → "Hello Bob!"	"Hello Bob!"	OK	
helloName("Alice") → "Hello Alice!"	"Hello Alice!"	OK	
helloName("X") → "Hello X!"	"Hello X!"	OK	
helloName("Hello") → "Hello Hello!"	"Hello Hello!"	OK	

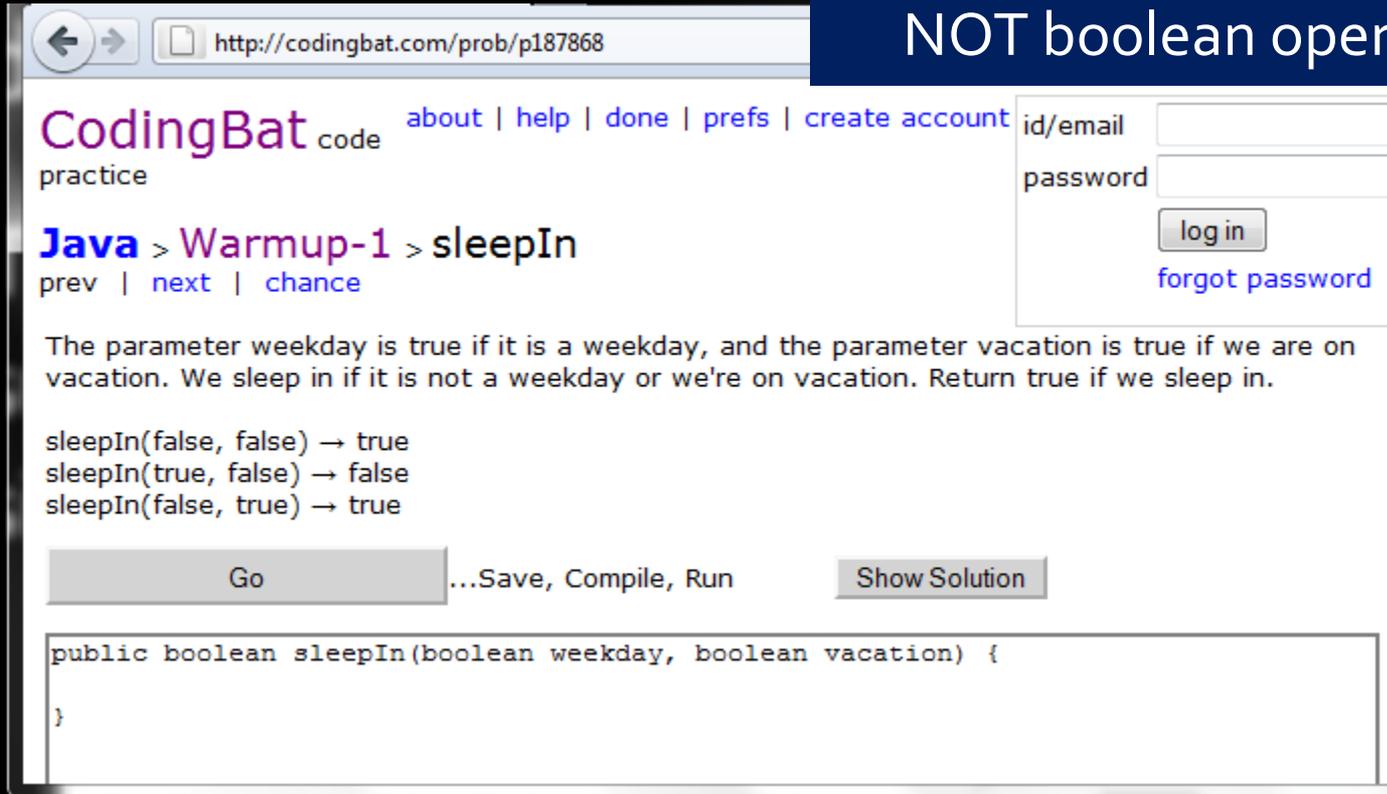


All Correct



Method 9 - *sleepIn*

- This next method is the first one in the **Warmup-1** section of the CodingBat website.
- We will work through several ways to solve this problem and introduce the AND, OR, and NOT boolean operators.



CodingBat code [about](#) | [help](#) | [done](#) | [prefs](#) | [create account](#)

practice

Java > **Warmup-1** > **sleepIn**

[prev](#) | [next](#) | [chance](#)

The parameter `weekday` is true if it is a weekday, and the parameter `vacation` is true if we are on vacation. We sleep in if it is not a weekday or we're on vacation. Return true if we sleep in.

`sleepIn(false, false) → true`
`sleepIn(true, false) → false`
`sleepIn(false, true) → true`

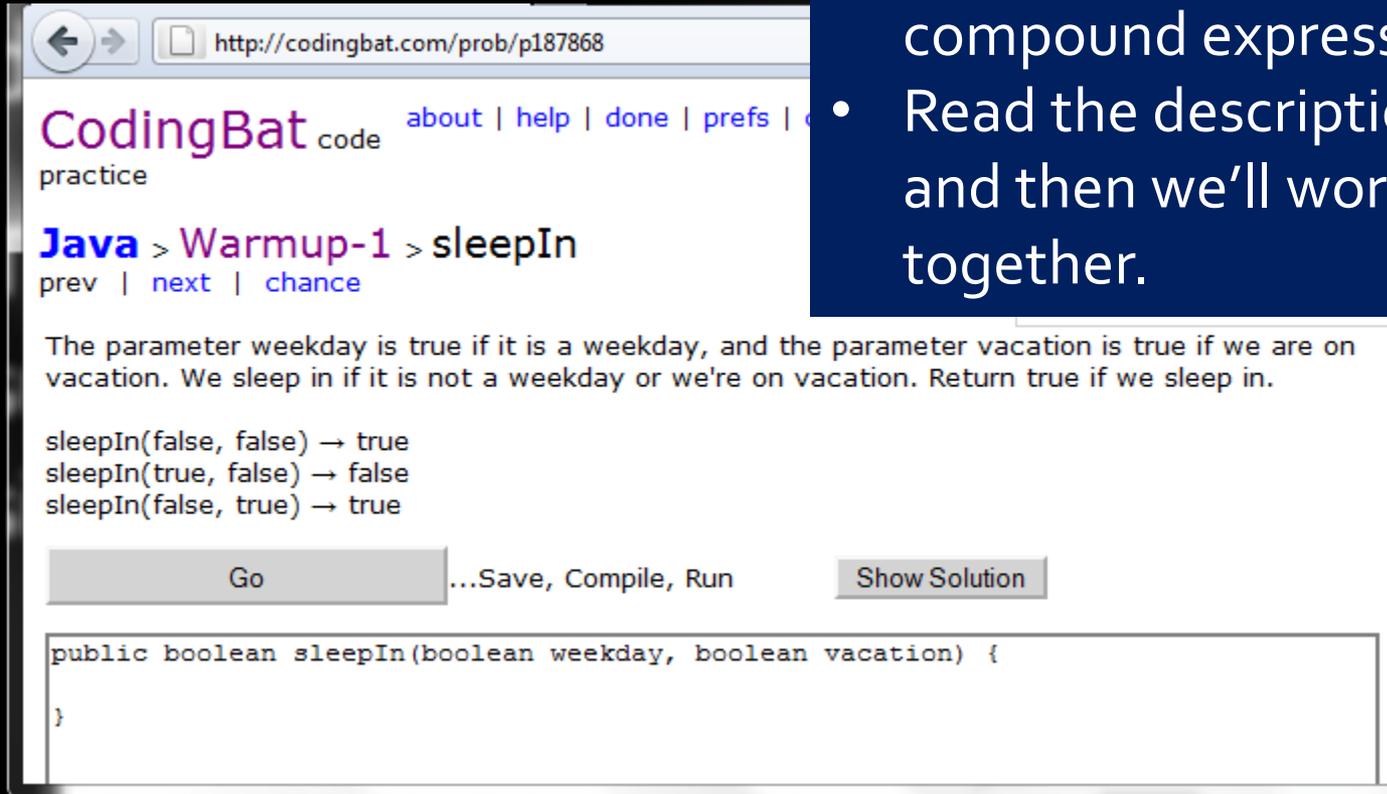
...Save, Compile, Run

```
public boolean sleepIn(boolean weekday, boolean vacation) {  
  
}
```



Method 9 - *sleepIn*

- In this method, you will return a boolean value based on two other boolean values, and you will use a new feature called AND to connect two boolean expressions into one compound expression.
- Read the description carefully, and then we'll work through it together.



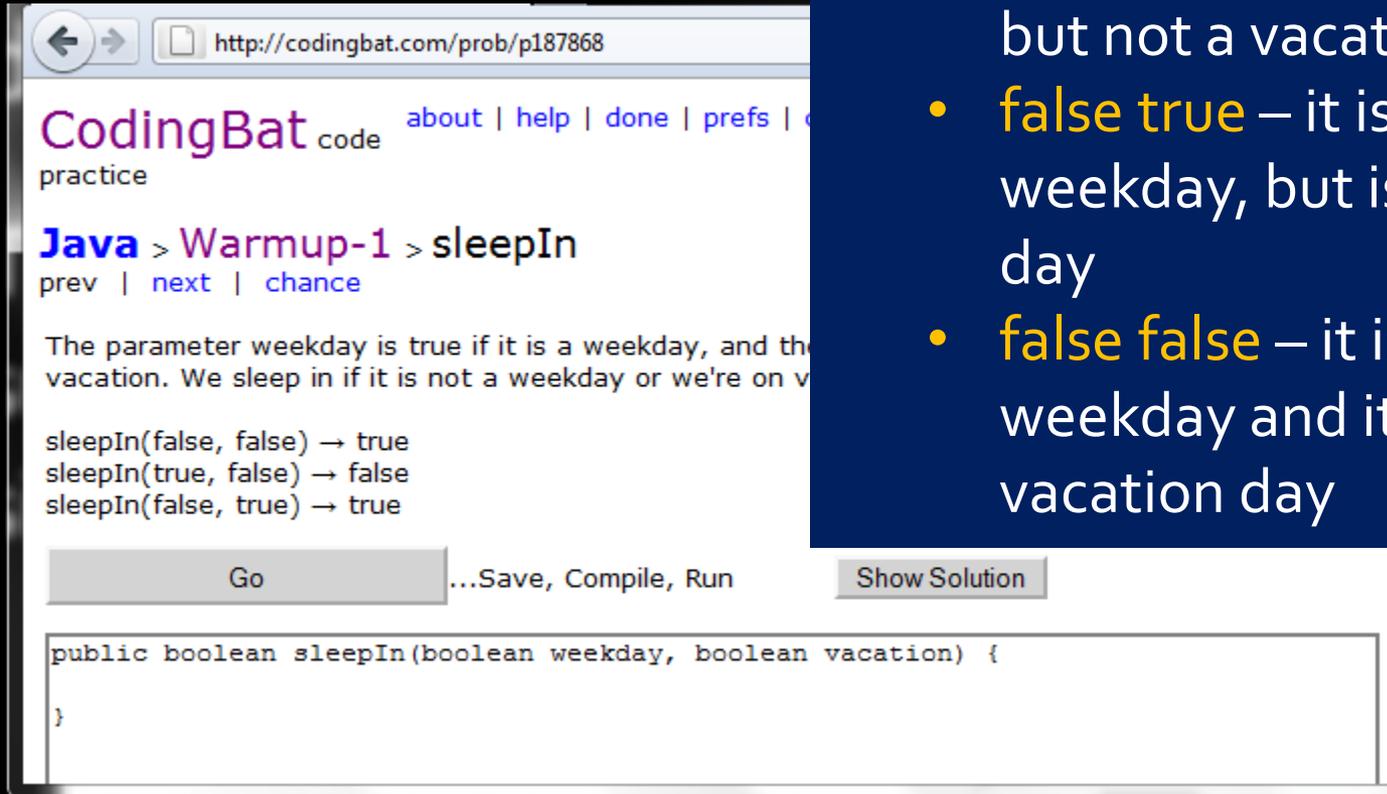
The screenshot shows a web browser window with the URL `http://codingbat.com/prob/p187868`. The page title is "CodingBat code" and the sub-page is "practice". The current problem is "Java > Warmup-1 > sleepIn". The page includes navigation links: "prev", "next", and "chance". The problem description states: "The parameter weekday is true if it is a weekday, and the parameter vacation is true if we are on vacation. We sleep in if it is not a weekday or we're on vacation. Return true if we sleep in." Below the description are three test cases: `sleepIn(false, false) → true`, `sleepIn(true, false) → false`, and `sleepIn(false, true) → true`. There are three buttons: "Go", "...Save, Compile, Run", and "Show Solution". At the bottom, there is a code editor with the following code:

```
public boolean sleepIn(boolean weekday, boolean vacation) {  
    }  
}
```



Method 9 - *sleepIn*

- There are only four possible combinations of the two boolean parameters:
 - **true true** – it is a weekday and it is a vacation day
 - **true false** – it is a weekday, but not a vacation day
 - **false true** – it is not a weekday, but is a vacation day
 - **false false** – it is not a weekday and it is not a vacation day



http://codingbat.com/prob/p187868

CodingBat code [about](#) | [help](#) | [done](#) | [prefs](#) | [code](#)

practice

Java > **Warmup-1** > **sleepIn**

[prev](#) | [next](#) | [chance](#)

The parameter `weekday` is `true` if it is a weekday, and the parameter `vacation` is `true` if we're on vacation. We sleep in if it is not a weekday or we're on vacation.

`sleepIn(false, false) → true`
`sleepIn(true, false) → false`
`sleepIn(false, true) → true`

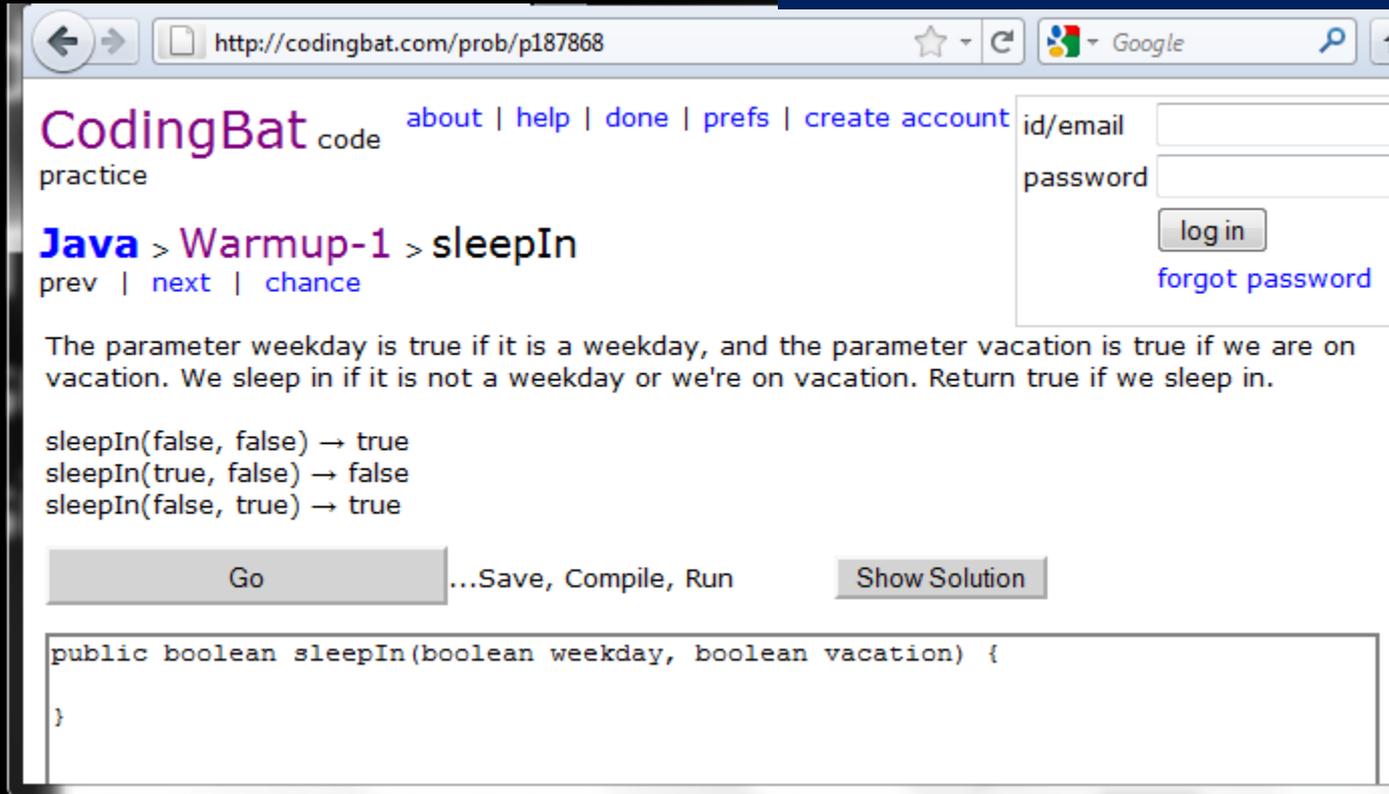
...Save, Compile, Run

```
public boolean sleepIn(boolean weekday, boolean vacation) {  
  
}
```



Method 9 - *sleepIn*

- Your job is to design a return statement, or combination of several return statements, that will correctly return **true** or **false**, indicating whether or not you can sleep in (legally, that is).



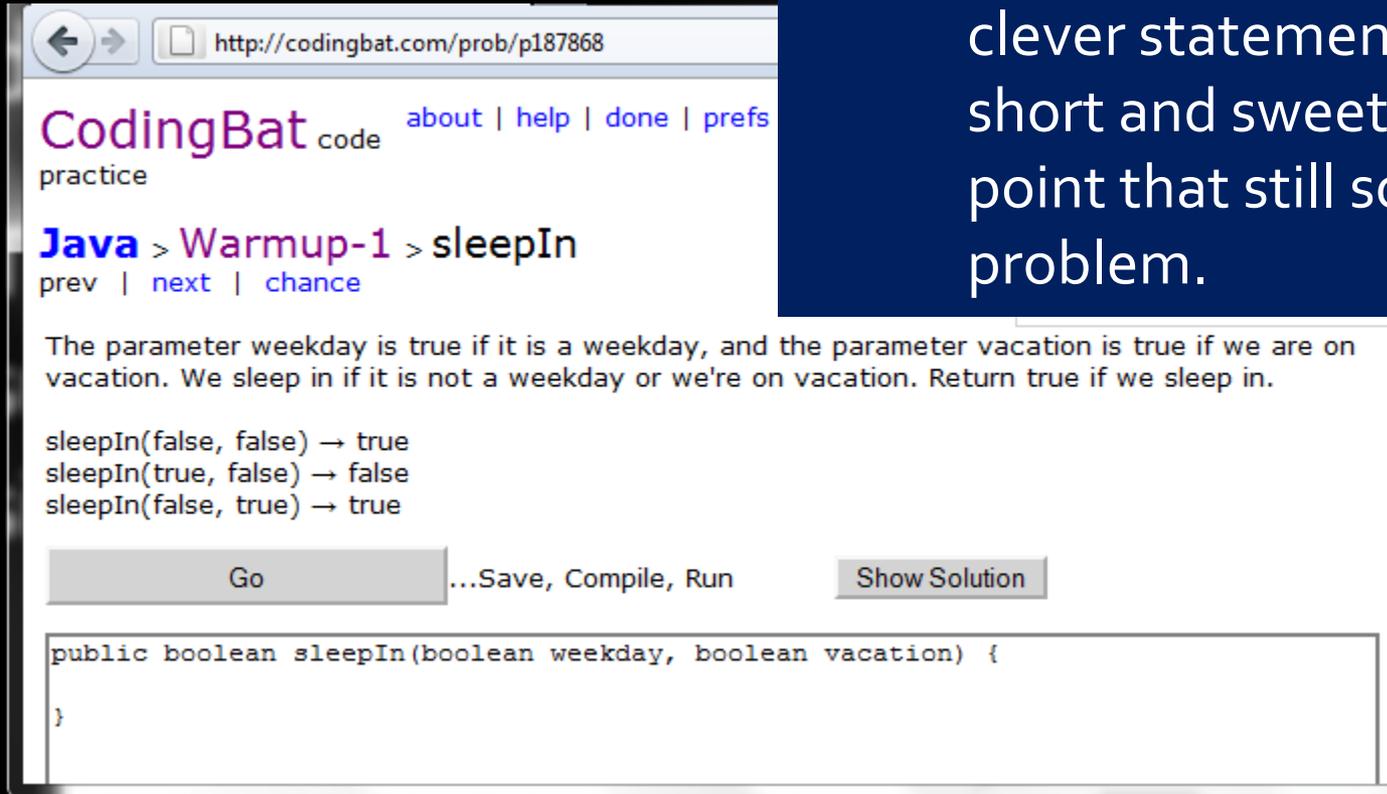
The screenshot shows a web browser window with the URL `http://codingbat.com/prob/p187868`. The page title is "CodingBat code practice" and it features navigation links: [about](#), [help](#), [done](#), [prefs](#), and [create account](#). The main content area displays the problem "Java > Warmup-1 > sleepIn" with sub-links: [prev](#), [next](#), and [chance](#). A description of the problem is provided: "The parameter weekday is true if it is a weekday, and the parameter vacation is true if we are on vacation. We sleep in if it is not a weekday or we're on vacation. Return true if we sleep in." Below the description are three test cases: `sleepIn(false, false) → true`, `sleepIn(true, false) → false`, and `sleepIn(false, true) → true`. At the bottom of the problem area are three buttons: "Go", "...Save, Compile, Run", and "Show Solution". A code editor is visible at the bottom with the following code:

```
public boolean sleepIn(boolean weekday, boolean vacation) {  
    }  
}
```



Method 9 - *sleepIn*

- There are several ways to do this:
 - The “brute force” way...set up an if statement for each possibility.
 - The “elegant” way...some clever statement that is short and sweet and to the point that still solves the problem.



The screenshot shows a web browser window with the URL `http://codingbat.com/prob/p187868`. The page title is "CodingBat code" with links for "about", "help", "done", and "prefs". Below the title is the word "practice". The main heading is "Java > Warmup-1 > sleepIn" with links for "prev", "next", and "chance". The problem description states: "The parameter weekday is true if it is a weekday, and the parameter vacation is true if we are on vacation. We sleep in if it is not a weekday or we're on vacation. Return true if we sleep in." Below this are three test cases: `sleepIn(false, false) → true`, `sleepIn(true, false) → false`, and `sleepIn(false, true) → true`. There are three buttons: "Go", "...Save, Compile, Run", and "Show Solution". At the bottom, a code editor contains the following code:

```
public boolean sleepIn(boolean weekday, boolean vacation) {  
    }  
}
```



Method 9 - *sleepIn*

- Below is the “brute force” method.
- The AND connector is used to connect each compound situation, represented by the “&&” symbol.

CodingBat code practice

Java > Warmup-1 > sleepIn

prev | next | chance

The parameter weekday is true if it is a weekday, and the parameter vacation is true if we are on vacation. We sleep in if it is not a weekday or we're on vacation. Return true if we sleep in.

sleepIn(false, false) → true
sleepIn(true, false) → false
sleepIn(false, true) → true

Go

...Save, Compile, Run

Show Solution

```
public boolean sleepIn(boolean weekday, boolean vacation) {  
    if (weekday == true && vacation == true)  
        return true;  
    if (weekday == true && vacation == false)  
        return false;  
    if (weekday == false && vacation == true)  
        return true;  
    if (weekday == false && vacation == false)  
        return true;  
    return true;  
}
```

Expected	Run		
sleepIn(false, false) → true	true	OK	
sleepIn(true, false) → false	false	OK	
sleepIn(false, true) → true	true	OK	
sleepIn(true, true) → true	true	OK	



All Correct

next | chance

CodingBat > Warmup



Method 9 - *sleepIn*

- Every combination is represented, and the method works.
- You might be wondering why there is an extra return statement...
- Here's why...there must always be one return statement that is free and clear of an if statement...more on that in a later lesson...for now, just do it.

CodingBat code practice

Java > Warmup-1 > sleepIn
prev | next | chance

The parameter weekday is true if it is a weekday, and the vacation. We sleep in if it is not a weekday or we're on v

sleepIn(false, false) → true
sleepIn(true, false) → false
sleepIn(false, true) → true

Go ...Save, Compile, Run

```
public boolean sleepIn(boolean weekday, boolean
    if (weekday == true && vacation == true)
        return true;
    if (weekday == true && vacation == false)
        return false;
    if (weekday == false && vacation == true)
        return true;
    if (weekday == false && vacation == false)
        return true;
    return true;
```

next | chance
CodingBat > Warmup-1



 All Correct

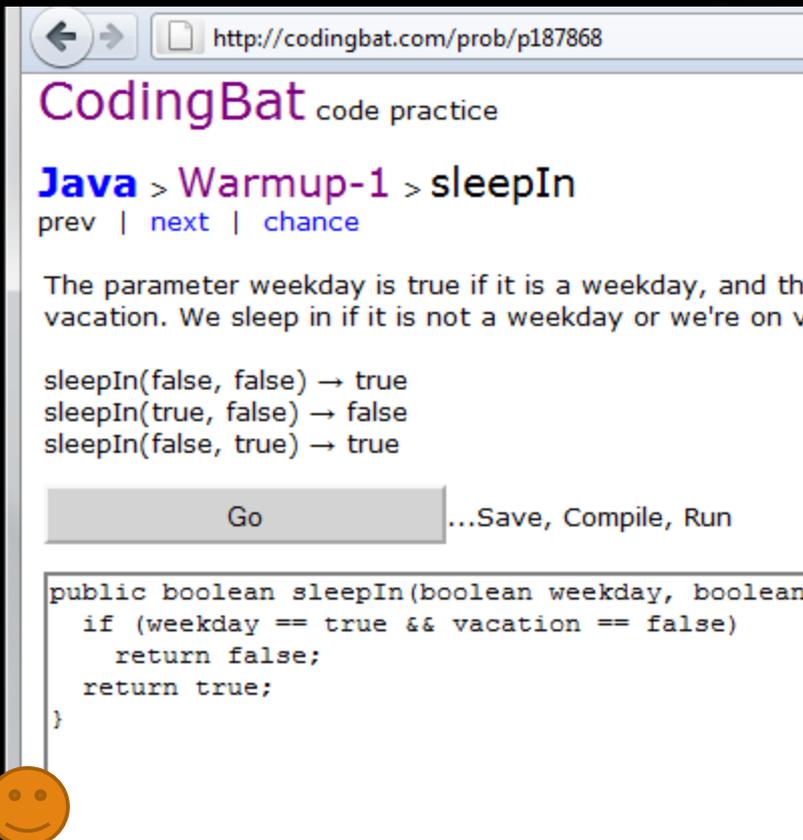
next | chance

true) → true true OK



Method 9 - *sleepIn*

- There is a “more elegant” way...
- In the previous solution, you might have noticed that there was only one false situation... *weekday and not a vacation day*, which certainly makes sense. *You must go to school on that day!* 
- Below is a solution that simplifies the logic considerably.
- Take a look at it carefully.



http://codingbat.com/prob/p187868

CodingBat code practice

Java > Warmup-1 > sleepIn

prev | next | chance

The parameter weekday is true if it is a weekday, and the vacation is true if we're on vacation. We sleep in if it is not a weekday or we're on vacation.

sleepIn(false, false) → true
sleepIn(true, false) → false
sleepIn(false, true) → true

Go ...Save, Compile, Run

```
public boolean sleepIn(boolean weekday, boolean vacation) {  
    if (weekday == true && vacation == false)  
        return false;  
    return true;  
}
```



sleepIn(false, true) → true	true	OK	
sleepIn(true, true) → true	true	OK	



All Correct



Method 9 - *sleepIn*

- Here is an even more “elegant” solution that uses the OR connector, “||”, which is two “pipe” symbols, located just above the ENTER key (use shift).
- It is essentially the same logic, approached from an “OR” point of view...think about it.

http://codingbat.com/prob/p187868

CodingBat

code practice

Java > Warmup-1 > sleepIn

prev | next | chance

The parameter weekday is true if it is a weekday, and the parameter vacation is true if we are on vacation. We sleep in if it is not a weekday or we're on vacation.

sleepIn(false, false) → true
sleepIn(true, false) → false
sleepIn(false, true) → true

Go ...Save, Compile, Run Show Solution

```
public boolean sleepIn(boolean weekday, boolean vacation) {  
    return weekday == false || vacation == true;  
}
```

sleepIn(true, false) → false	false	OK	
sleepIn(false, true) → true	true	OK	
sleepIn(true, true) → true	true	OK	

All Correct

Method 9 - *sleepIn*

- Here is the “most elegant” solution that uses the OR connector and the NOT symbol, which when used in this way implies the OPPOSITE of the boolean value.
- *!weekday* is the same as stating *weekday == false*, and simply stating *vacation* implies *vacation == true*.

http://codingbat.com/

CodingBat code practice

Java > Warmup-1 > Sleep In

prev | next | chance

The parameter weekday is true and the parameter vacation is false. We sleep in if it is not a weekday, or we are on vacation.

sleepIn(false, false) → true
sleepIn(true, false) → false
sleepIn(false, true) → true

Go ...Save, Compile, Run Show Solution

```
public boolean sleepIn(boolean weekday, boolean vacation) {  
    return !weekday || vacation;  
}
```

	Run	
sleepIn(false, false) → true	true	OK
sleepIn(true, false) → false	false	OK
sleepIn(false, true) → true	true	OK
sleepIn(true, true) → true	true	OK

✓ All Correct

Method 10 - *cigarParty*

- This final method is the first one in the **Logic-1** section, and uses similar logic techniques as you saw in the previous method. Read the description carefully, and even click on the Show Hint button if you need to. Good luck!

Java > Logic-1 > cigarParty

prev | next | chance

forgot

When squirrels get together for a party, they like to have cigars. A squirrel party is successful when the number of cigars is between 40 and 60, inclusive. Unless it is the weekend, in which case there is no upper bound on the number of cigars. Return true if the party with the given values is successful, or false otherwise.

cigarParty(30, false) → false

cigarParty(50, false) → true

cigarParty(70, true) → true

Go

...Save, Compile, Run

Show Hint

```
public boolean cigarParty(int cigars, boolean isWeekend) {  
  
}
```



one

Internet | Protected Mode: On

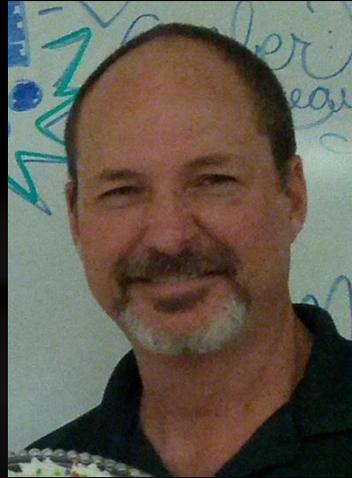


CONGRATULATIONS!

- You now know how to write your own static return methods.
- You also were introduced to the AND, OR, and NOT boolean operators using the CodingBat website labs.
- *The Lesson 6A will tell you about loops and how they work.*



Thank you, and have fun!



To order supplementary materials for all the lessons in this package, such as lab solutions, quizzes, tests, and unit reviews, visit the [O\(N\)CS Lessons](#) website, or contact me at

[John B. Owen](#)
captainjbo@gmail.com



10/10/2014

