Introduction to Java and NetBeans: Hello World!

The Tools You Will Use

What is Java?

Java is a computer programming language — accompanied by a set of development tools, libraries, and runtime "virtual machines" for a variety of platforms — originally developed by James Gosling and his team of programmers at Sun Microsystems. (Sun continues to own the Java trademark; to declare and evolve the standard for Java; and to develop the core tools, classes, and virtual machines that make up the Java family of products.) The key goals of the Java platform are as follows:

- It is intended to be highly portable, allowing code compiled on one platform to be run on another (using the Java virtual machine architecture), even if the CPUs and operating systems of the two systems are completely different.
- It is intended to be secure. Compiled Java code can be "signed" with certificates issued by trusted agencies, to ensure that software which purports to be from a given provider is indeed from that provider; also, runtime policies can be set, granting each Java class a configurable level of access to the resources of the host system, based on whether the code is signed or unsigned (and, if signed, based on the signer), as well as the source and location of the compiled code.
- It is intended to be general-purpose. Although Java originally gained wide attention as a language and runtime platform for Web applets (actually, Java was originally intended for use in "intelligent devices" — set-top boxes, PDAs, etc.), and although its virtual machine architecture means that Java programs will generally not run as fast as code which is compiled and optimized for a specific hardware platform and operating system, the language and APIs are intended to give sufficient flexibility for a wide variety of applications, with an acceptable level of performance for most.
- Finally, it was, and is, intended to leverage and encourage good programming practices — through the evolving product itself, and through supporting materials available from Sun. A central set of practices supported by Java are referred to as "object-oriented programming".

In most of these aspects, Java succeeds pretty well. Of course, there is a price for the flexibility and power that Java offers: while its learning curve may not be as steep as that for C++ (for example), expertise in Java still requires a significant amount of time and practice. Nonetheless, a programmer can become productive in Java in a relatively short amount of time, and a casual programmer can do good work in Java without having to be a Java guru.

What is Object-Oriented Programming?

Object-oriented programming is a programming style, and a set of programming practices — usually supported to some extent by the underlying programming language and environment — through the use of which a programmer is able to do the following:

- Define and instantiate (i.e. create new variables based on) custom variable types, based on the intrinsic types in the language, and on other custom types.
- Incorporate the operations relevant to the new types (i.e. how they behave) into the definitions of the types.
- Define and use the new types as specializations of the types on which they are based — supporting the same operations, with extensions, modifications, and additions to the relevant operations as appropriate.
- Programmatically invoke the operations associated with a custom type, on an instance of that type, without the invoking code having to be aware of the internal details of that type — or even that the instance may be of a type which is a specialization of the referred-to type. In other words, the variable types can be "black boxes", and an instance of a specialized type can be treated as if it were an instance of the more general type on which the specialization is based.

The ability to encapsulate data and behavior together makes OOP a very good fit for modeling and simulation projects, where objects in the physical world (with their own attributes and behaviors) are mapped to corresponding objects (with the associated attributes and behaviors) in a computer program, and the interactions between those objects are simulated in the program.

A few programming languages have been designed from the ground-up as purely object-oriented languages. However, many more are hybrids — existing languages with object-oriented features added in, or new languages which incorporate some object-oriented, and some non-object-oriented, concepts and features. For example, Visual Basic and Object Pascal (the Pascal dialect used in Delphi) fall into the category of languages which started out as non-object-oriented, and which have had object-oriented features grafted on; on the other hand, C++ and Java fall into the category of hybrid languages which have incorporated some object-oriented elements — and some non-object-oriented elements — from their inceptions.

How do I start writing Java programs?

Given Sun's aims for Java, it would make sense for Sun to make Java compilers, libraries, and virtual machines widely available, at minimal cost. In fact, that is exactly what they have done — not just at low cost, but at no cost (in most cases). The Java Development Kit (JDK) can be downloaded free of charge from the Sun Java site (http://java.sun.com), for Windows and several flavors of Unix/Linux (including Mac OS X), and for several different CPUs. With the JDK, you can compile Java applications and applets (which can use the services provided by the classes in the extensive Java API), package compiled classes into `.jar` archives, sign archives — even develop Java programs that will run on cell phones and PDAs.

Also available from the Sun Java site is complete reference documentation for Java — the language, tools, and APIs — and The Java Tutorial. The latter guides a new Java programmer, step-by-step, through exercises that explore all of the key features of the Java language, and a large portion of the API library.

What is NetBeans?

There's one very important thing missing from the JDK: a development environment. You can write Java code with any text editor (a fair number still program with `notepad.exe`), and then use the compiler in the JDK, but a good development environment can have a dramatic impact on programmer productivity — as well as on the rate at which a programmer moves up the learning curve.

Fortunately, Sun recognized this shortcoming several years ago, and in 1999 acquired an Integrated Development Environment (IDE) intended for Java development — and written in Java, as well — called NetBeans; they also acquired the company that developed the IDE. In 2000, Sun released NetBeans to open source, allowing the entire community of NetBeans users to participate directly in the IDE's evolution. Just as is the case with Java itself, Sun maintains control over the specification of NetBeans, and over the official releases; but developers oustide Sun are encouraged to contribute fixes and enhancements, and to participate in the specifications and standards processes.

NetBeans can be downloaded free of charge from netbeans.org

"Hello World" with Java and NetBeans

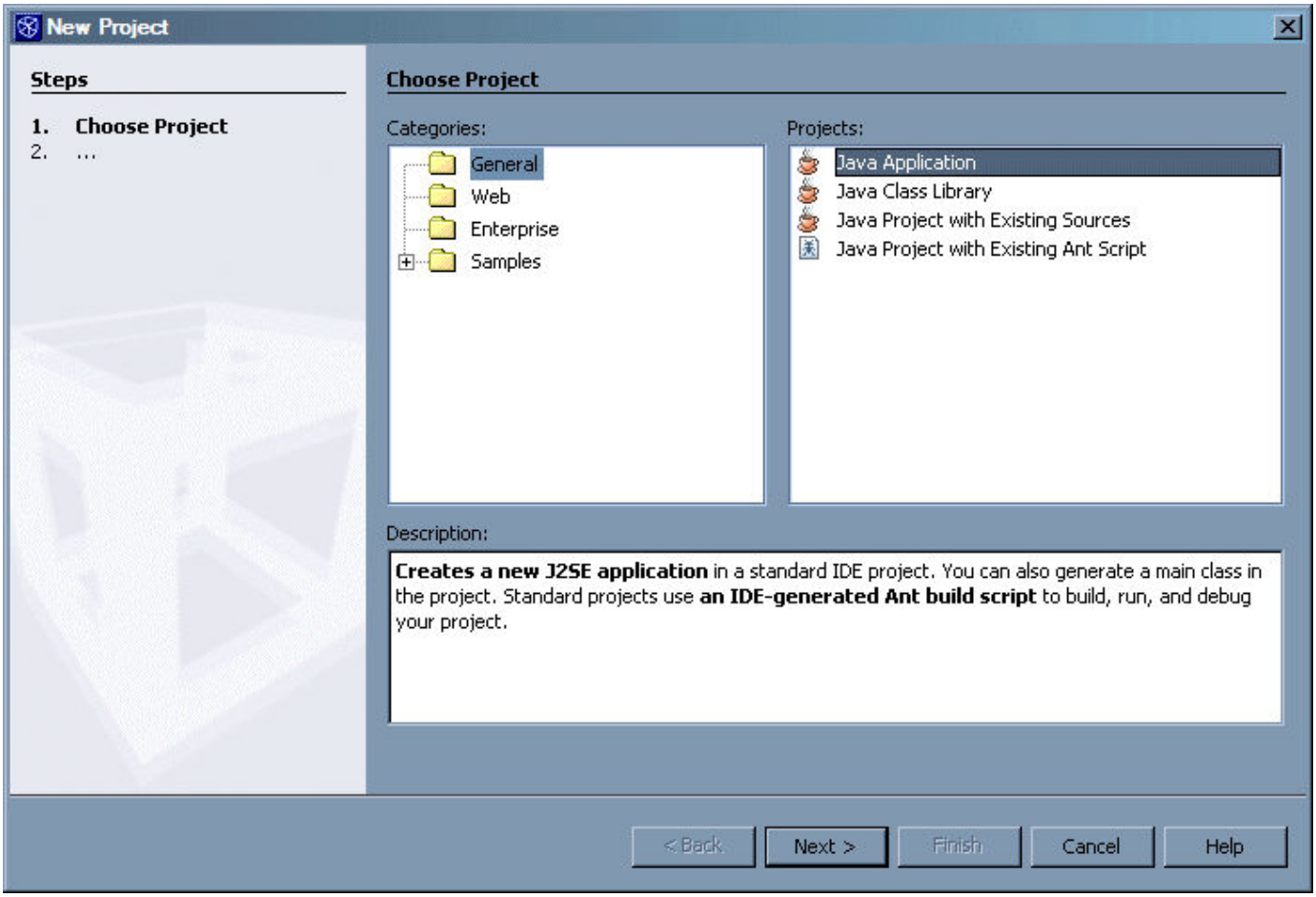
Why "Hello World"?

"Hello World" is the generic name for a program which does nothing more than display or print the text: "Hello, world!" Since the 1970s (the earliest documented appearance was in an internal Bell Laboratories memo on C programming), the "Hello World" program has been widely used as a first program — a first program for someone just learning to program, a first program for a programmer learning a new language, a first program used by a compiler writer to test a new build, etc. "Hello World" programs can be very simple (like the one we will write today), or much fancier (for example, using advanced features of a GUI to present an animated "Hello World" message).

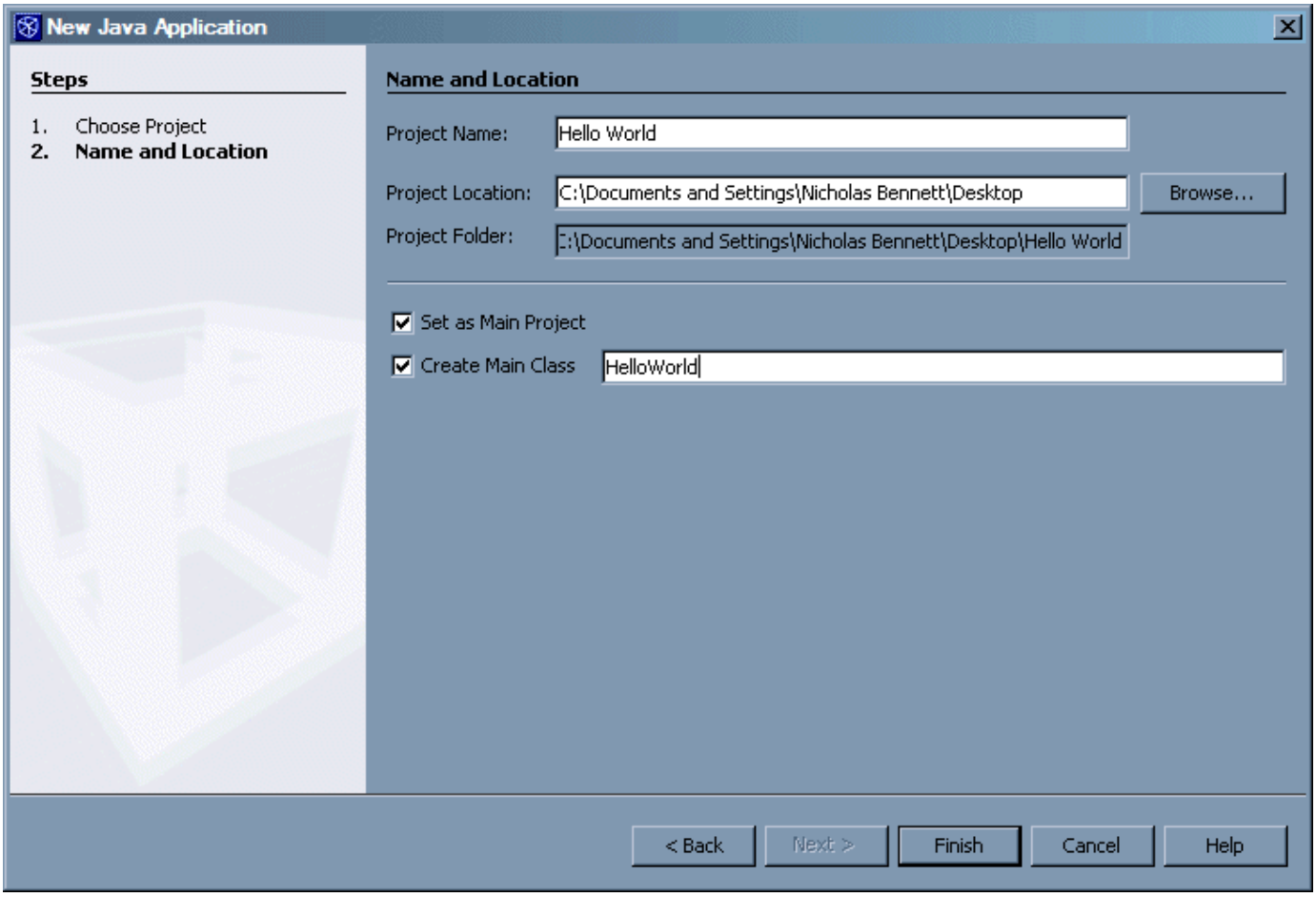
Many of you have never programmed in Java before; some that have, have never used NetBeans. So, "Hello World" is probably not a bad place to start.

Exercise #1: Creating a new NetBeans project

Launch the NetBeans program (it may take a few minutes to fully start up, the first time after installation). When it has completed its startup process, select **New Project…** from the **File** menu, to open the New Project wizard:



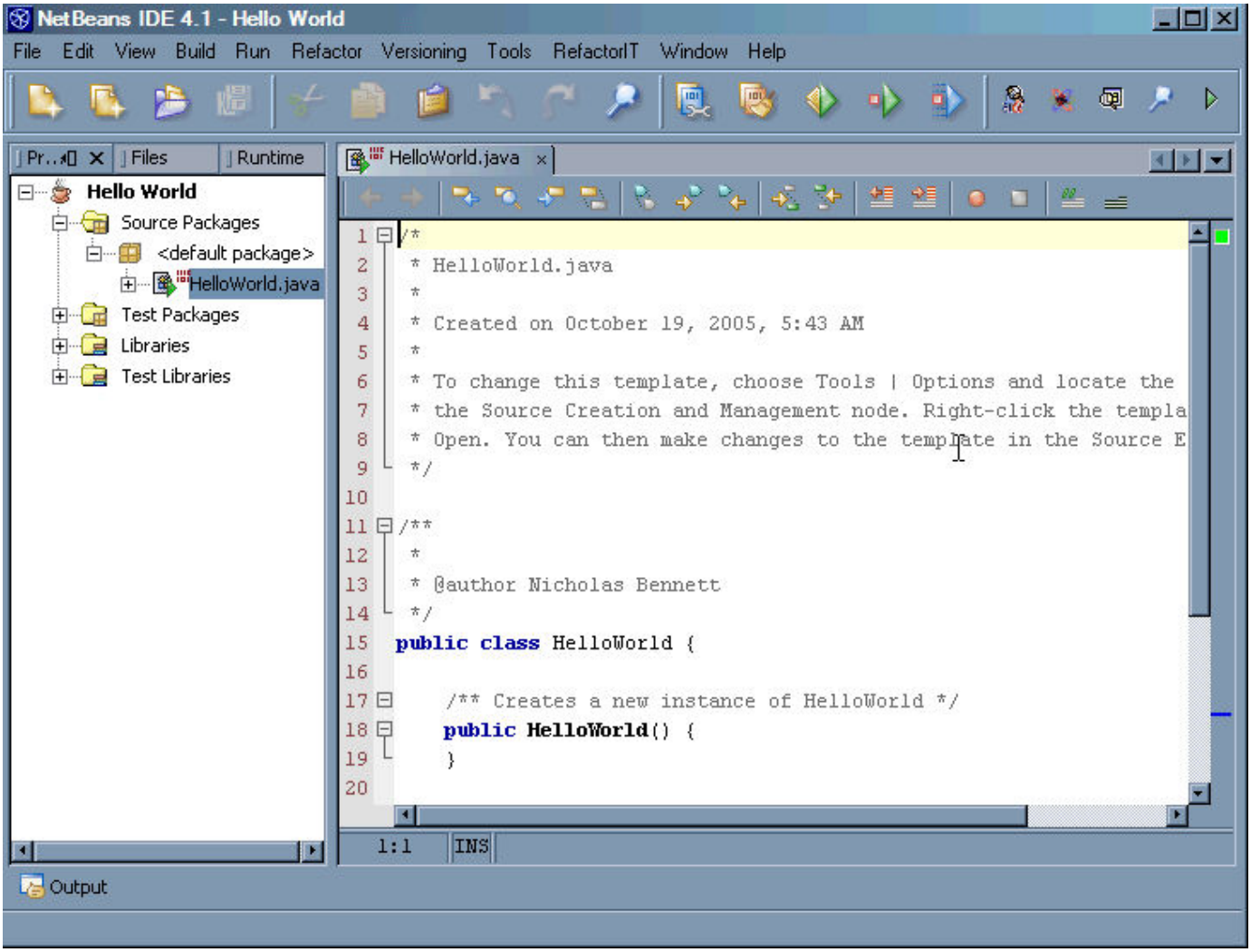
Select **General** from the list of categories, and **Java Application** from the list of projects. Then, press the **Next** button, to go to the following screen:



Follow these steps to complete the **New Java Application** screen:

1. For **Project Name**, type "Hello World" (don't worry about spelling or case for this one — the name you type is the name NetBeans will use when displaying the project in its workspace, but is has little to do with the compilation and execution of your program).
2. The value provided by default in **Project Location** is probably fine; but if you want to be able to copy your work to a floppy or USB drive, it might be easier if you specify a directory (by clicking the **Browse…** button, and navigating from there) that will be easily found later. (The Desktop is pretty good for this purpose.) In any event, you will want to write this value down for later use.
3. Make sure that **Set as Main Project** is checked.
4. Make sure that **Create Main Class** is checked, and type "HelloWorld" for the class name (this time, spelling and case are *very* important — also make sure you don't have spaces between the words).
5. Click the **Finish** button.

Now, you should see a new NetBeans project, with the source file for the class you specified (in step #4) open and ready for editing. (Don't worry if your NetBeans screen isn't laid out in exactly the same way: as long as you see your new project on the left, and the `HelloWorld.java` file open on the right, you are doing fine.)



Here are some things to notice about this what is displayed in the NetBeans workspace:

- The Java class for our program is `HelloWorld` (hint: look for the line that begins with the words `public class`); similarly, the file in which we define the class is `HelloWorld.java`. It is *very* important that these two match — not just in name (except for the `.java` file extension), but also in case (more on that below).
- The `HelloWorld` class contains a method called `main`. *Every* Java standalone application must have at least one class which has a `main` method, declared in this way. (Note that there are many other kinds of classes in Java, which don't need a `main` method; also, there are different kinds of Java programs — applets, for example — which use different methods as starting points.)
- Note that NetBeans has automatically included a line (a comment line) in the `main` method, which reads: "`// TODO code application logic here`"; we will be replacing this line in a few minutes.
- The `HelloWorld` class contains a constructor; this is the code that begins with `public HelloWorld()`. A constructor is used for creating individual instances (objects) based on the class definition. Since our first program will be so simple, we can ignore this for now.
- As it stands, the code compiles and runs cleanly (even though it doesn't really do anything when it runs); we can check this by selecting **Run Main Project** from the **Run** menu (if the code has been modified, this will also cause the code to be recompiled before it is run). In general, it is a good practice to write code in such a way that you can compile and test often — even before the code does everything it is supposed to do. Similarly, when changing existing code, make and test small changes — rather than making a large number of changes, and only then trying to compile and test, and finding out that there are several errors in the code.

Exercise #2: Adding the "Hello, World!" code

As mentioned above, every Java standalone application has a method called `main`, which is declared just like our own method by that name. There is a reason for this: the Java "executive" (the software responsible for loading and running programs written in Java) needs to be able to find this method, to know where our program starts. In other words, the `main` method is the starting point for a Java standalone application. Since all we need to do in our program is display a single line of text, it makes sense to keep things simple, and put the Java code to do that in the `main` method.

Actually, NetBeans has made our task pretty easy: all we need to do is replace the line that reads "`// TODO code application logic here`" with the line of code that will display "Hello, world!" The question is, what code will do that?

There is a class in the Java API which is always available to us, called `System`. This class provides many services to Java applications; one of the most important is access to the standard input and output devices (i.e. the keyboard and screen — more specifically, a text-mode window). The `System.out` object is the Java encapsulation of the standard output device. The type definition for this object (remember the description of object-oriented programming, above?) includes a number of operations — called "methods", in Java — for sending data to the standard output device (i.e. displaying data in a text-mode window). For fun (but mostly because we'll make use of it in a few minutes), we're going to use a method that was only recently added to the Java API, but which has been available to C and C++ programmers for years: `printf`.

So, we will be calling the `printf` method of the `System.out` object. And, just as the we write `System.out`, to refer to the `out` object in the `System` class, we will write `System.out.printf`, to refer to the `printf` method in the `System.out` object.

The only thing left is to tell `System.out.printf` what it should display. In Java, we put string literals (i.e. text that we know when we write the code, rather than text that is computed dynamically when the program is running) inside double quotes. So, the way we would write the text we will display is exactly what you might expect: `"Hello, world!"` And the way we tell `System.out.printf` to display that text is to pass it to the method, inside parentheses: `System.out.printf("Hello, world!")`. Finally, we need to end the line of code with a semicolon (all simple statements in Java need a semicolon at the end), giving us this complete line of code:

Example: *Statement to display "Hello, world!"*

```
System.out.printf("Hello, world!");
```

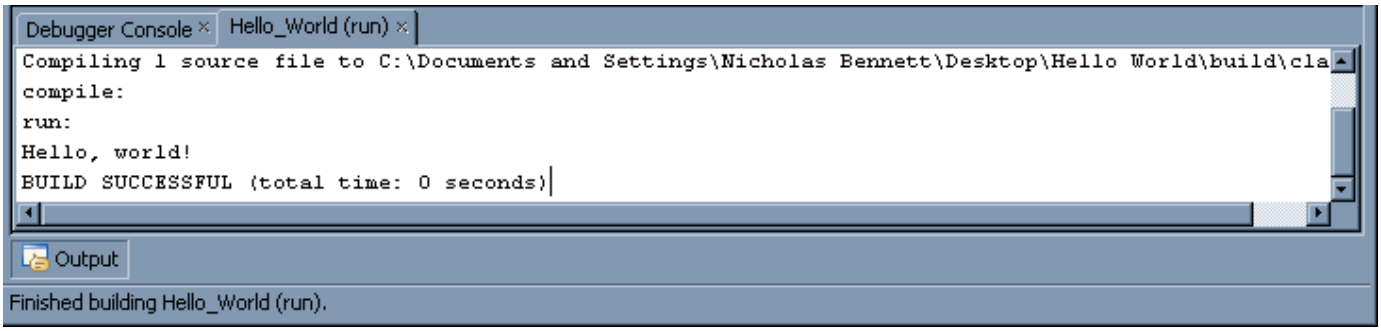
Find the line of code in the `HelloWorld.java` file which reads "`// TODO code application logic here`", and replace the entire line (but nothing else) with the line shown above.

When you are done, your `main` method should read as follows:

Example: *"Hello World"* `main` *method*

```
public static void main(String[] args) {
    System.out.printf("Hello, world!");
}
```

Select **Run Main Project** from the **Run** menu, and you should see your "Hello World" message, in the output pane of the NetBeans workspace:



Exercise #3: Making it a bit more interesting

So far, so good — but let's face it: "Hello World" programs are pretty boring. We don't have time to do anything really exciting with our program, but we can make the text displayed a little less static. In fact, that was why we used the `System.out.printf` method: it makes it easy to output dynamic data, along with static text. We haven't made use of those capabilities yet — but we will now.

Our last Java programming task in this session will be to include the current date and time (to the nearest second) in the text we display.

First, let's see how we will need to change the "Hello World" text. The `System.out.printf` method lets us include placeholders in the text; at runtime, these placeholders are replaced with values that we pass to the method (in addition to the text itself). These placeholders not only mark the position in the text where we want the dynamic values to appear — they also indicate the kind of values we are passing, and any conversions we want to apply to those values, before displaying them. In this case, the placeholder for a date/time value, formatted in a fairly universal format, is "%tc"; for now, let's assume that we will put this placeholder in the text right after the words `"Hello, world!"`, and in parentheses: `"Hello, world! (%tc)"`.

The placeholder tells the `System.out.printf` method where and how to display the value we pass — but it doesn't tell it the value we want to display. We know we want to display the current date and time, but how can we refer to that value, and pass it to the `System.out.printf` method? As it turns out, there is a class in the standard Java API library, which will do most of the work for us. That class is one of the custom datatypes we discussed earlier — the class we want is called `Date`, and it is located in the package (packages are collections of classes) `java.util`; we can thus refer to the class as `java.util.Date`. When we create a new object from this class, that new `Date` object will have as its value the current date and time, taken from the computer's system clock; the way we create this new object is with the operator `new`, like this: `new java.util.Date()`.

Finally, we can pass this new object (containing the current date and time) to the `System.out.printf` method, simply by including it in the parentheses, after the text, with a comma between them. After modifying the text to include the placeholder, and adding the new `Date` object, our "Hello World" line of code should read as follows:

Example: *Code to display "Hello, world!", with the current date & time*

```
System.out.printf("Hello, world! (%tc)", new java.util.Date());
```

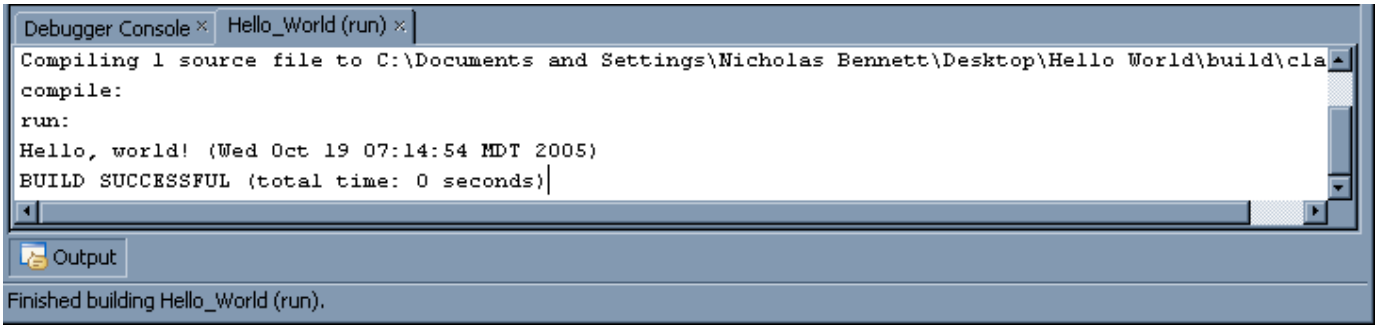
Find the line of code that you previously modified in `HelloWorld.java`, and replace it with the line shown above. Be *very* careful about your parentheses and quotation marks, when typing this new line.

When the change is complete, your main method should look like the following:

Example: *"Hello World"* `main` *method, with display of current date & time*

```
public static void main(String[] args) {
    System.out.printf("Hello, world! (%tc)", new java.util.Date());
}
```

Select **Run Main Project** from the **Run** menu, and your new "Hello World" message, with the date and time, should appear in the NetBeans output pane:



Congratulations on completing your "Hello World" (with a twist) program in Java, with NetBeans!