Practices for Lesson 5

Practices Overview

In these practices, you create and manipulate Java technology objects and also create and use String and StringBuilder objects. In the last exercise, you become familiar with the Java API specification.

Practice 5-1: Creating and Manipulating Java Objects

Overview

In this practice, you create instances of a class and manipulate these instances in several ways. This Practice has two sections. In the first section, you create and initialize object instances. In the second section, you manipulate object references.

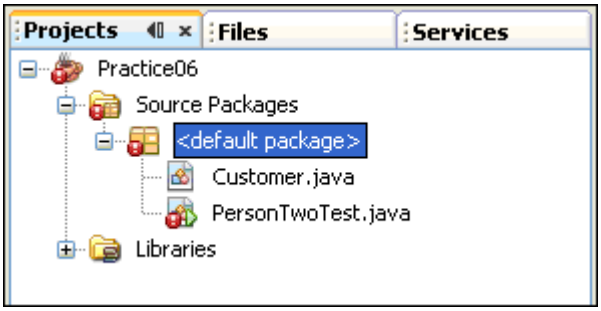
Assumptions

The `Customer.java` file appears in the practice folder for this lesson: `Lesson05`

Initializing Object Instances

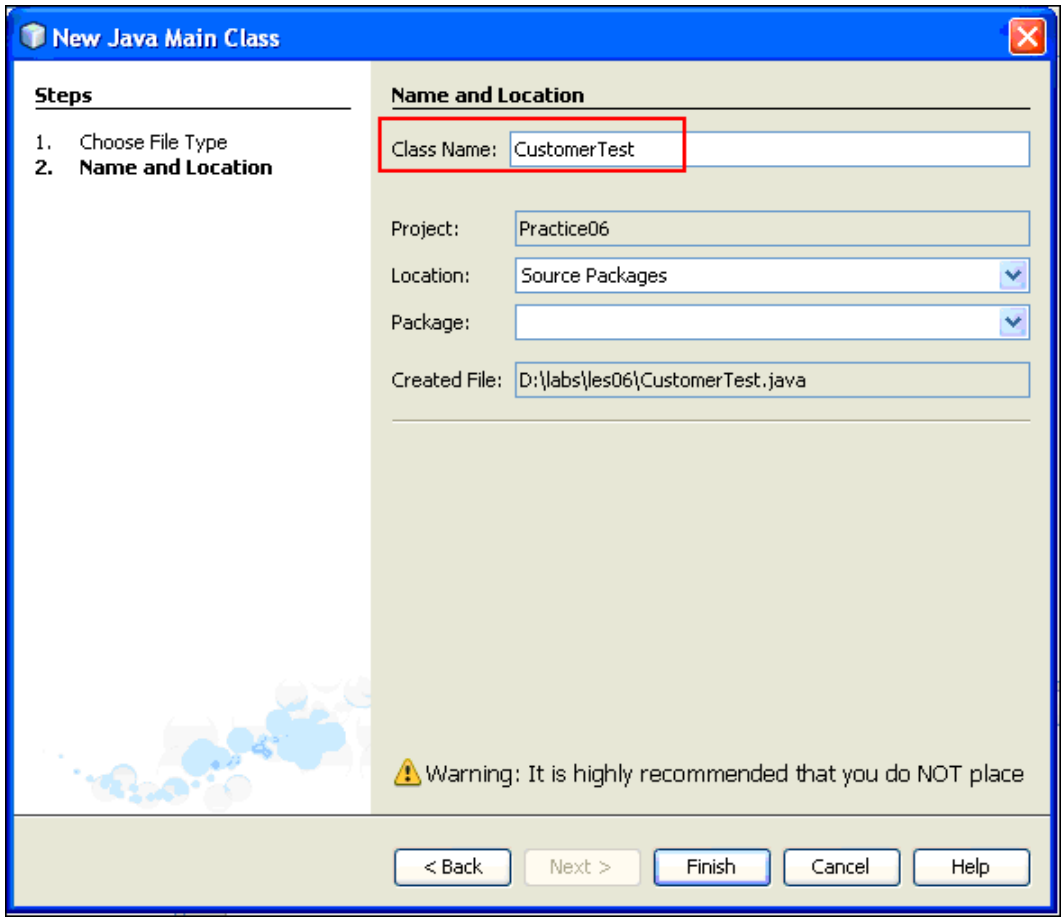
A Customer class is provided for you. In this section, you create, compile, and execute a CustomerTest class. In this test class, you create objects of the Customer class and set values to its member fields.

1. Create a new project from existing source called Practice06. Set the **Source Package Folder** to point to `Lesson05`. Remember to also change the Source/Binary Format property. If you need further details, refer to Practice 1-2, Steps 3 and 4.

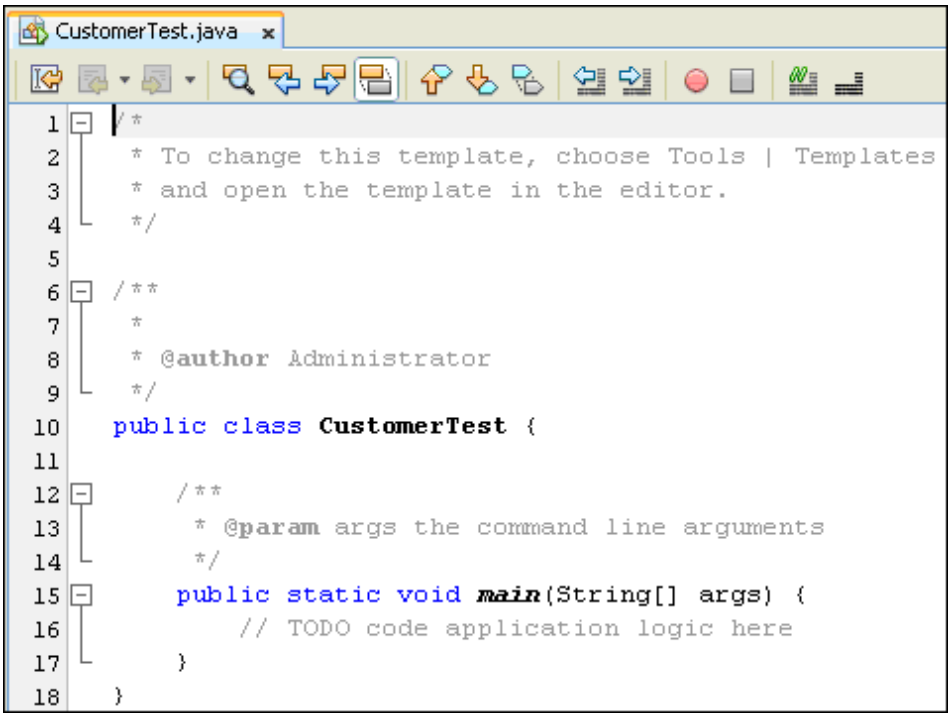


2. Open the `Customer.java` file in the editor and examine its member fields and its method. You use the field information to complete this practice.
3. Create the CustomerTest class as a “Java Main Class” type. Since this class is run (executed) by the Java executable, it must contain a `main` method. The NetBeans IDE provides the skeleton of a main class for you.
   a. Right-click the Practice06 project in the Projects window and select New > Java MainClass from the popup menu. (This is a shortcut way of creating new Java classes.)

b. Name the class CustomerTest and click Finish.



c. The CustomerTest class appears in the text editor.



4. In the `main` method of `CustomerTest`, add code to declare and initialize two instances of the `Customer` class. The table below provides high-level instructions for this task. If you need more assistance, refer to the detailed steps following the table.

Step	Window/Page Description	Choices or Values
a.	Declare two fields of type `Customer`	**cust1** **cust2**
b.	Initialize the two instances	Use the `new` operator

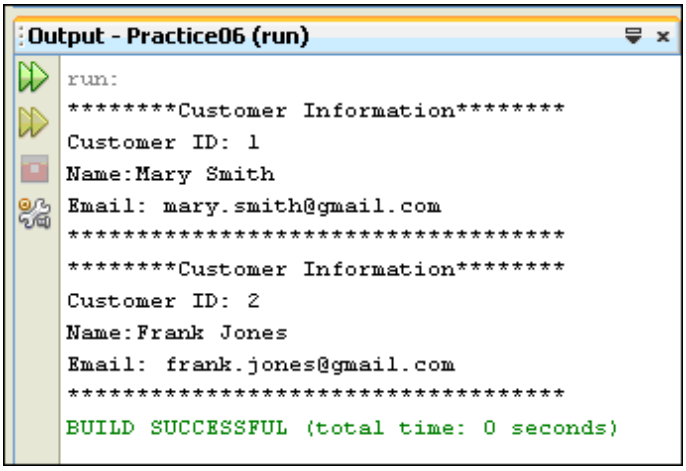
   a. Within the body of the `main` method, declare two fields of type Customer as follows:
   ```
   Customer cust1, cust2;
   ```
   b. Initialize each of the variables using this syntax:
   ```
   <variable name> = new <class name>();
   ```

5. Finish coding the `main` method as indicated in the following table. More detailed instructions are provided below the table.

Step	Window/Page Description	Choices or Values
a.	Assign values to the member fields of one of the `Customer` objects	Example: `cust1.customerID = 1;`
b.	Repeat for the other Customer object but use different values for the fields.	
c.	Invoke the `displayCustomerInfo` method of each object	Use the object reference variable to qualify the method as you did in step a.

   a. Assign values to all of the member fields of one of the Customer objects. Use the object reference variable to qualify the field name as shown below:
   ```
   cust1.customerID = 1;
   ```
   b. Assign different values to each member field of the other `Customer` object.
   c. Invoke the `displayCustomerInfo` method of each object. Example:
   ```
   cust1.displayCustomerInfo();
   ```

6. Click Save to compile.
7. Run the `CustomerTest.java` file. Check the output to be sure that each Customer object displays the distinct values you assigned.



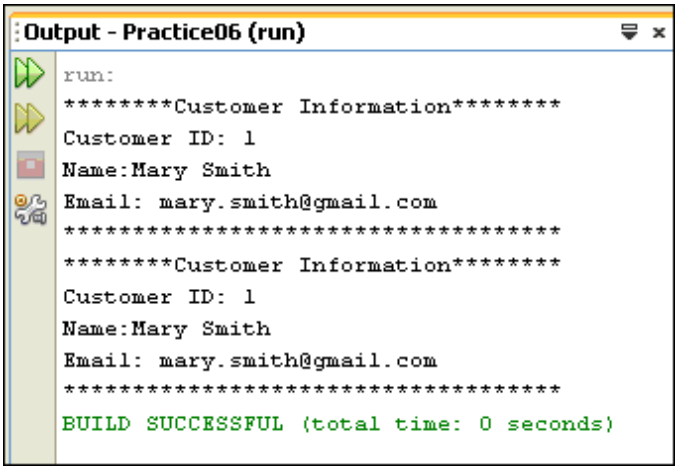
Manipulating Object References

In this section, you assign the value of one object reference to another object reference.

8. Edit the `main` method of CustomerTest to assign one object reference to another object reference just above the first line of code that invokes the `displayCustomerInfo` method. For example (assuming that `cust1` and `cust2` are instances of the Customer class):

```
cust2 = cust1;
```

9. Save and run the `CustomerTest.java` file. Check the output of the `displayCustomerInfo` methods for both objects. Both of the object references now point to the same object in memory so both of the `displayCustomerInfo` method outputs should be identical.



Practice 5-2: Using the StringBuilder Class

Overview

In this practice, you create, initialize, and manipulate StringBuilder objects.

Assumptions

The `PersonTwoTest.java` file appears in the practice folder for this lesson: `Lesson05`

Creating and Using String Objects

1. Create a new Java class called “PersonTwo”.
2. Declare and instantiate two member fields of type `StringBuilder` to hold the person’s name and phone number, respectively. For the `name` field, initialize the capacity of the `StringBuilder` object to 8. Use meaningful field names.

   Example Solution:

   ```
   public class PersonTwo {
        public StringBuilder name = new StringBuilder(8);
        public StringBuilder phoneNumber = new StringBuilder();
   }
   ```

3. Create a new method called “displayPersonInfo”.
4. In the body of the `displayPersonInfo` method, populate and then display the `name` object. Ensure that the total number of characters in the name exceeds the initial capacity of the object (8). The following table provides high-level steps for this task. More detailed instructions can be found below the table.

Step	Window/Page Description	Choices or Values
a.	Add a first name to the `StringBuilder` object	Use the `append` method of the `StringBuilder` class
b.	Append two more values to the `name` object	a space: “ “ a last name **Note:** The total number of characters appended should exceed 8
c.	Display the `String` value of the `name` object	Use the `toString` method of the `StringBuilder` class
d.	Display the capacity of the `name` object with a suitable label	Use the `capacity` method of the `StringBuilder` class
e.	Compile and run the program	Run the `PersonTwoTest.java` file

   a. Use the `append` method of the `StringBuilder` class to append a first name. Example:

      ```
      name.append("Fernando");
      ```

   b. Use the same method in two separate invocations to add first a space (“ “), and then a last name. Ensure that total number of characters that you have added to the `name` object exceeds 8.

Note: You can accomplish the same thing by using a String object and concatenating additional values. However, this would be inefficient because a new String object is created with each concatenation. String object capacity cannot be increased as Strings are immutable.

c. Use the `System.out.println` method to display the entire name value. You can embed the `toString` method of `name` object within the `System.out.println` method.

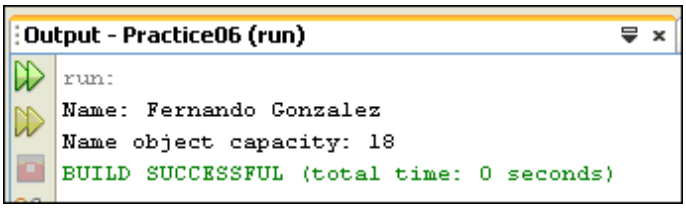
```
System.out.println("Name: " + name.toString());
```

d. Display the capacity of the `name` object, using the `capacity` method. The `StringBuilder` object has dynamically increased the capacity to contain all of the values that you have appended.

Example Solution:

```
public void displayPersonInfo(){
     name.append("Fernando");
     name.append(" ");
     name.append("Gonzalez");
     // Display the name object
     System.out.println("Name: + name.toString());
     // Display the capacity
     System.out.println("Name object capacity: " +
          name.capacity());
}
```

e. Click Save to compile. Run the `PersonTwoTest.java` file. The output should look similar to the screenshot below. Notice that the capacity has been increased from the initial setting of 8 to accommodate the full name.



5. Populate and manipulate the `phoneNumber` object. Here you append a string of digits and then use the `insert` method to insert dashes at various index locations, achieving the format "nnn-nnn-nnnn". The table below provides high-level instructions for this task. More detailed instructions can be found below the table.

Step	Window/Page Description	Choices or Values
a.	Append a 10 digit `String` value to the `phoneNumber` object	Example: "5551234567"
b.	Insert a dash ("-") after the first three characters of the `phoneNumber`.	Use the `insert` method that takes an `int` value for the offset and inserts a `String` value. (Use offset number 3)
c.	Insert another dash after the first seven characters of the `phoneNumber`	Reminder: The previous insertion pushed the remaining characters over one index.
d.	Display the `phoneNumber` object	Use the `toString` method of the `StringBuilder` class

a. Use the `append` method of the `StringBuilder` class to append a `String` value consisting of ten numbers.

b. Insert a dash ("-") at offset position 3. This puts the dash at the 4th position in the String, pushing all of the remaining characters over one position. The syntax for this method is shown below:

```
<object reference>.insert(int offset, String str);
```

Example: Consider the following string,

"5551234567"

The offset position 3 occurs at the number 1. (Index numbers begin at 0.) If the dash is inserted at offset position 3, it pushes the number currently at that position and all remaining numbers over to the next offset position.

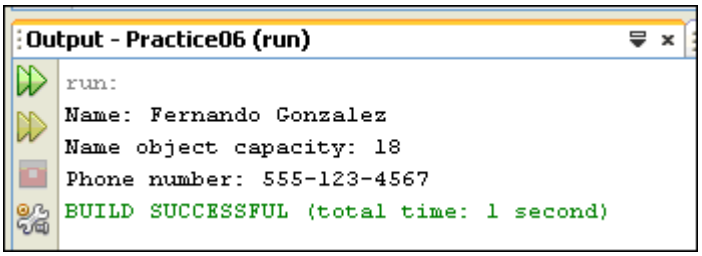
c. Insert a dash at offset position 7 (where the number 4 is currently placed).

d. Use `System.out.println` to display the output from the `StringBuilder` object's `toString` method.

Solution:

```
phoneNumber.append("5551234567");
phoneNumber.insert(3, "-");
phoneNumber.insert(7, "-");
System.out.println("Phone number: " +
   phoneNumber.toString());
```

6. Click Save to compile. Run the `PersonTwoTest.java` file. Check the output from the `displayPersonInfo` method. Ensure that the dashes appear between the third and fourth digits and between the sixth and seventh digits.



7. Use the `substring` method of the `StringBuilder` class to get just the first name value in the `name` object. Use the `substring` method that takes the start index and the end index for the substring. Display this value using `System.out.println`.

   Syntax:

   ```
   <object reference>.substring(int start, int end);
   ```

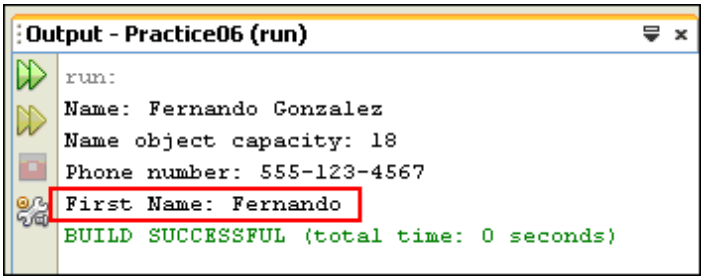
   Note: Indexes for characters in the StringBuilder class, much like array indexes, are zero-based. The first character in the StringBuilder is located at position (or index) 0.

   While the start index of the `substring` method is inclusive (it is the *actual* index of the first character you want returned), the end index is exclusive (it is the index of the character just to the right of the last character of your substring.)

   Example Solution:

   ```
   // Assumes the first name "Fernando"
   System.out.println("First name: " + name.substring(0,8));
   ```

8. Save and again run the `PersonTwoTest.java`. Check the output and make any adjustments necessary to the index numbers to get the correct first name value.



Practice 5-3: Examining the Java API Specification

Overview

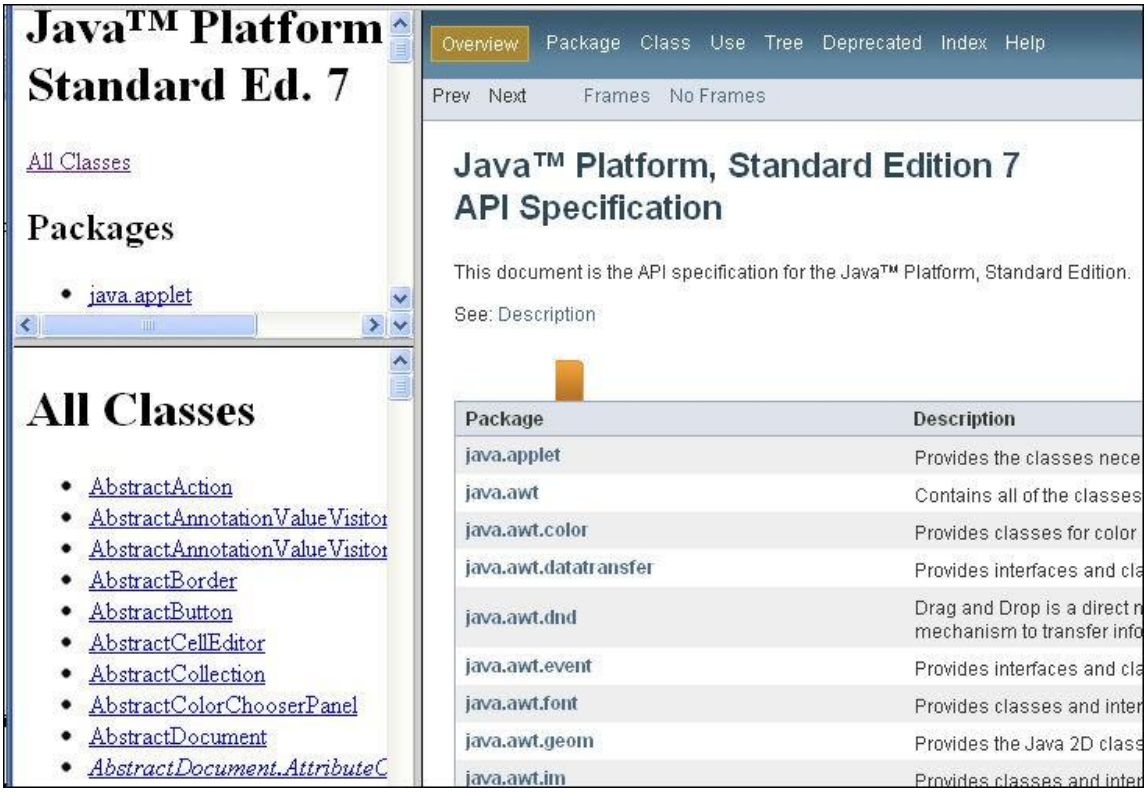
In this practice, you examine the Java API specification to become familiar with the documentation and how to look up classes and methods. You are not expected to understand everything you see. As you progress through this course, the Java API documentation should make more sense.

Assumptions

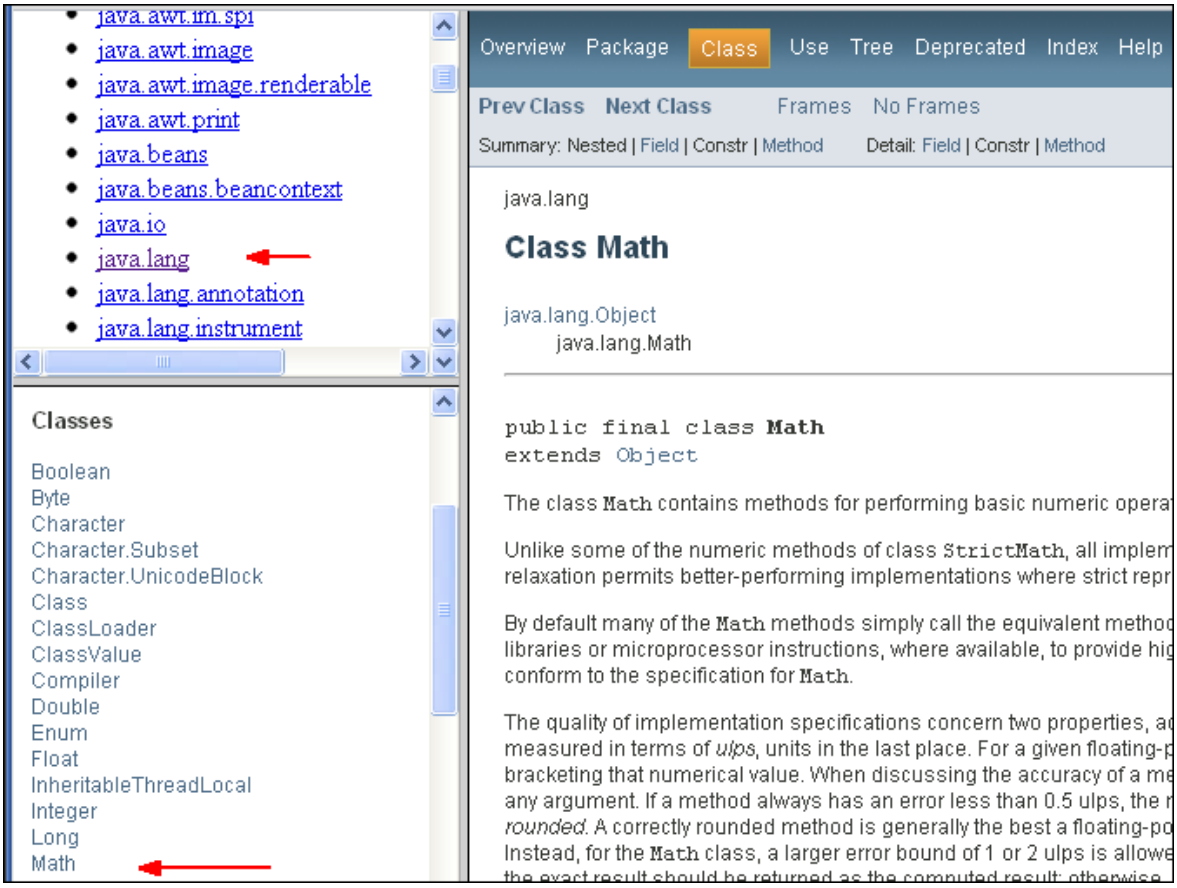
The Java SE 7 API specification is installed locally on your machine.

Tasks

1. To view the Java SE7 API specification (also referred to as “javadocs”), double-click the shortcut on your desktop (entitled “Java JDK7 1.7.0 API Docs”).



   The opening page of the javadocs consists of three frames as shown above. It allows you to navigate through the hierarchy of classes in the API by class name or by package. (**Note:** you learn about packages later in this course)
2. Using the **Packages** frame, select the `java.lang` package. The **All Classes** frame now changes to display only classes within that package.
3. Find the `Math` class and click it to display its documentation in the main frame.



4. Answer the following questions about the Math class:
   a. How many methods are there in the `Math` class?_____
   b. How many fields are there in the `Math` class?_____

   Answer:
   a.) 54
   b.) 2

5. Select several other classes in the Classes panel to answer this question: What class does every class refer to at the top of the page? **Hint:** What class is the superclass to all classes? _____

   Answer: `Object`

6. Find the `String` class and identify the methods of this class. Which methods enable you to compare two strings?_____

   Answer: `compareTo` and `compareToIgnoreCase`

7. Close the Practice06 project in NetBeans.