

Practice 2: Introducing to Java technology

Practices Overview

In these practices, you run a Java program, first using the DOS command line and then from the NetBeans integrated development environment (IDE).

The practices for this course assume that the following software is installed:

- JDK 1.7.0
- Java API Specification (installed locally)
- Java SE7 Specification (installed locally)
- NetBeans EE Edition, 7.0.1 (GlassFish server only; Tomcat server is not used.)

Practice 2-1: Running a Java Program Using the Command Line

Overview

In this practice, you compile and run a Java program at the command line. A Java technology program is already created for you. In some cases, you may need to first set the PATH variable for the DOS session before running the program. Instructions for setting the PATH are included below.

Assumptions

The Java SE 7 development environment is installed on your computer.

Task: Compiling and Executing a Java Program

In this task, you compile and execute a Java Program.

1. Compile the CalcAverage.java program. The high-level steps for this task are shown in the table below. If you need more assistance, you can use the detailed steps that follow the table.
- 2.

Step	Description	Choices or Values
a.	Open a DOS command window and navigate to:	D:\labs\les02
b.	Check the contents of this directory listing to find:	CalcAverage.java
c.	Set the PATH variable to include:	D:\Program Files\Java\jdk1.7.0\bin
d.	Compile the CalcAverage java source file by typing:	javac CalcAverage.java

- a. From the Windows Start menu, select Start > Run. Enter `cmd` in the Open field and click **OK**. At the prompt, enter `cd D:\labs\les02`. Press Enter.

```
cmd D:\WINNT\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
D:\winnt\Profiles\Administrator>cd D:\labs\les02
```

- b. Check the directory contents by typing `dir` at the command prompt. Press enter to see the results listed.

```
D:\labs\les02>dir
Volume in drive D is WINNT
Volume Serial Number is FC5C-B059

Directory of D:\labs\les02

06/08/2011  02:27 PM    <DIR>          .
06/08/2011  02:27 PM    <DIR>          ..
04/22/2011  03:05 PM                641 CalcAverage.java
                1 File(s)                641 bytes
                2 Dir(s)  459,428,489,216 bytes free
```

- c. Confirm that the system PATH points to the correct folder location for the Java executables (the compiler and the runtime executable). Type `PATH` at the prompt and

press Enter. You should see D:\Program Files\Java\jdk1.7.0\bin appearing somewhere in the PATH string as shown below.

```
D:\winnt\Profiles\Administrator>PATH
PATH=D:\Program Files\Java\jdk1.7.0\bin;d:\winnt\system32;d:\winnt;d:\winnt\system32\ubn;c:\dos;c:\ntinst.ad;c:\utils;c:\detect;c:\net
```

- a. If it is not there, append this directory to the System PATH variable by entering the following at the command prompt. Press enter.

```
D:\labs\les02>PATH = %PATH%;D:\Program Files\Java\jdk1.7.0\bin
```

You can confirm that the PATH was changed correctly by typing PATH at the next prompt. You should see the jdk1.7.0\bin appearing at the end of the PATH string.

- d. Compile the .java file by typing `javac CalcAverage.java`. Press enter. After a slight delay the prompt will return.

```
D:\labs\les02>javac CalcAverage.java
```

3. Run the CalcAverage.java program. The high-level steps for this task are shown in the table below. If you need more assistance, you can use the detailed steps that follow the table.

Step	Window/Page Description	Choices or Values
a.	Confirm that the file was successfully compiled. List the directory content and look for:	CalcAverage.class
b.	Run the CalcAverage program. It will prompt you to enter three integers separated by spaces. Do so and press enter to see the average of the three integers.	java CalcAverage

- a. Look for the compiled class, `CalcAverage.class`, by listing the contents of the directory again. Type `dir` and press Enter.

```
D:\labs\les02>dir
Volume in drive D is WINNT
Volume Serial Number is FC5C-B059

Directory of D:\labs\les02

06/08/2011  04:57 PM    <DIR>          .
06/08/2011  04:57 PM    <DIR>          ..
06/08/2011  04:57 PM                921 CalcAverage.class
04/22/2011  03:05 PM                641 CalcAverage.java
                2 File(s)      1,562 bytes
                2 Dir(s)    459,428,415,488 bytes free
```

- b. Run the CalcAverage program by invoking the java runtime executable. You do not need to use the .class extension of the class. Type `java CalcAverage` and press Enter. The program will prompt you to enter three integers.

```
D:\labs\les02>java CalcAverage
Enter 3 Integers separated only by spaces: <example 20 30 40>
```

Type three integers separated by spaces and then press Enter.

```
D:\labs\les02>java CalcAverage
Enter 3 Integers separated only by spaces: <example 20 30 40>
2 46 88
Average = 45
```

This is how you would compile and run a Java program using only a DOS console or terminal window.

Practice 2-2: Running a Java Program Using NetBeans IDE

Overview

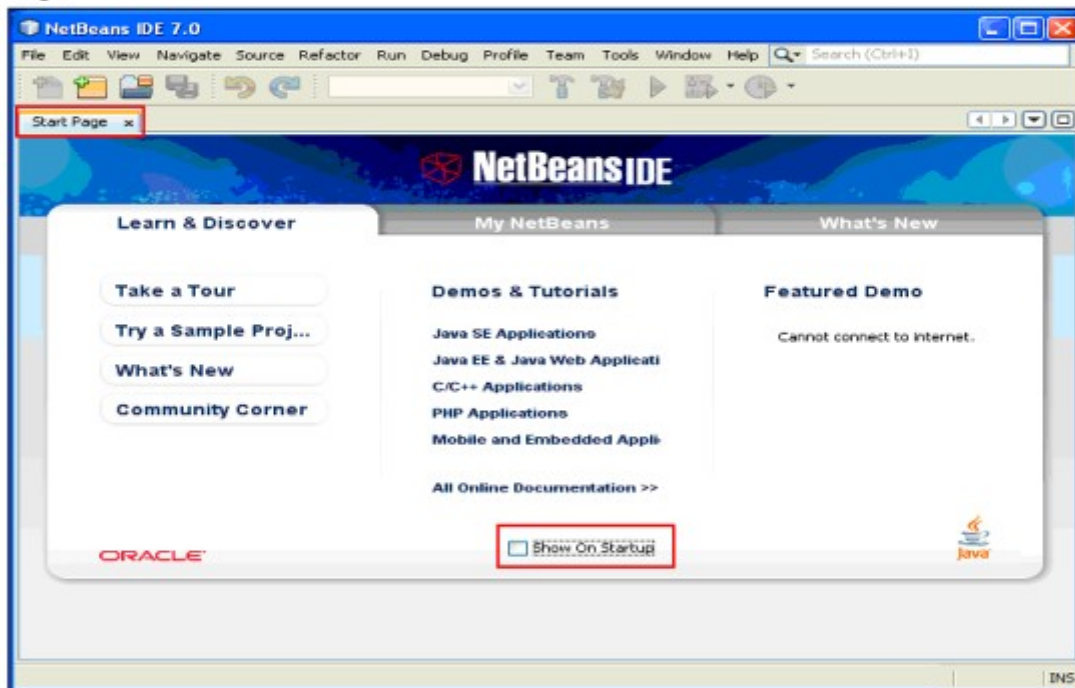
In this practice, you compile and execute a Java program using NetBeans IDE. In addition, you explore some features of an IDE that let you develop programs more quickly and easily than if you use a command line.

Assumptions

The NetBeans 7.0.1 IDE is installed on your computer.

Tasks

1. Double-click the NetBeans icon from your computer desktop.
2. When NetBeans opens, deselect the **Show On Startup** check box and close the Start Page.

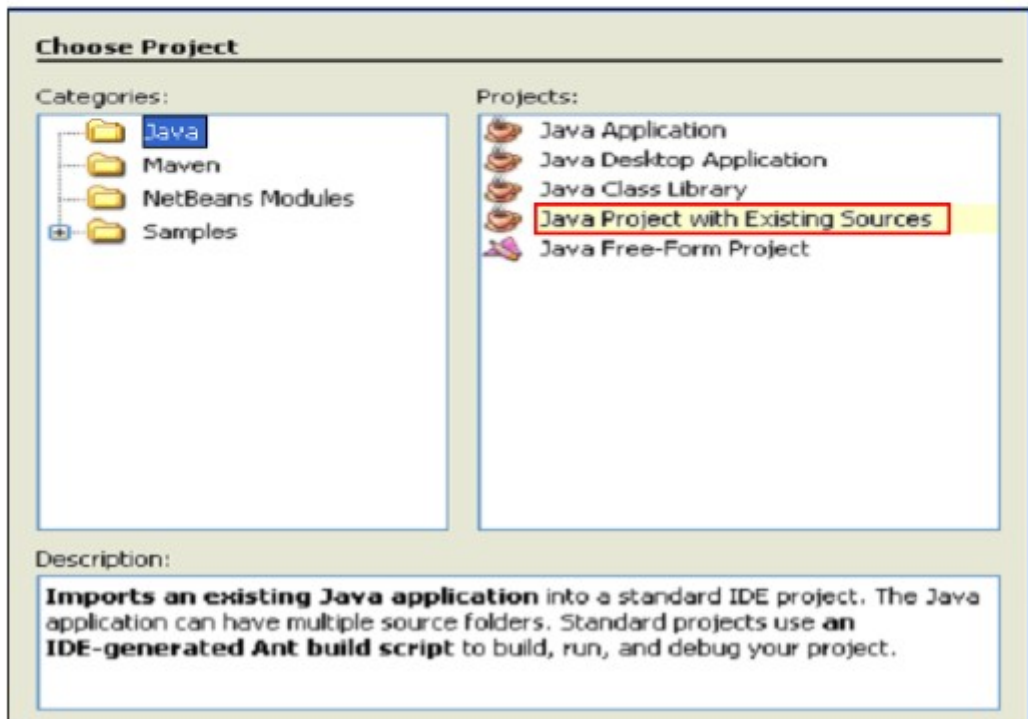


3. Create a NetBeans project that includes the CalcAverage.java file in its project source folder. The high-level steps for this task are shown in the table below. If you need more assistance, you can use the detailed steps that follow the table.

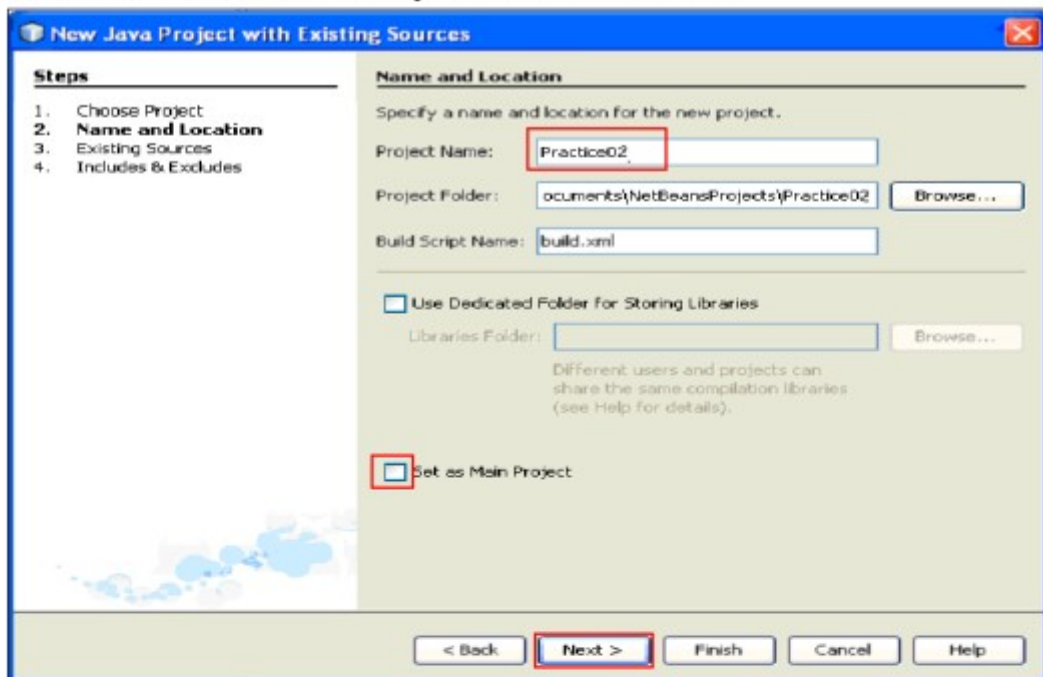
Step	Window/Page Description	Choices or Values
a.	Main menu	File > New Project ...
b.	New Project wizard: Choose Project step	Categories: Java Projects: Java project with existing source Click Next
c.	New Project wizard: Name and Location step	Project Name: Practice02 Deselect the Set as Main Project check box Click Next

Step	Window/Page Description	Choices or Values
d.	New Project wizard: Existing Sources step	Source Packages Folder: Browse to select D:\labs\les02 Click Finish
e.	Prompt window	Delete existing class files within the package folder. The new project appears in the Project's window of NetBeans.

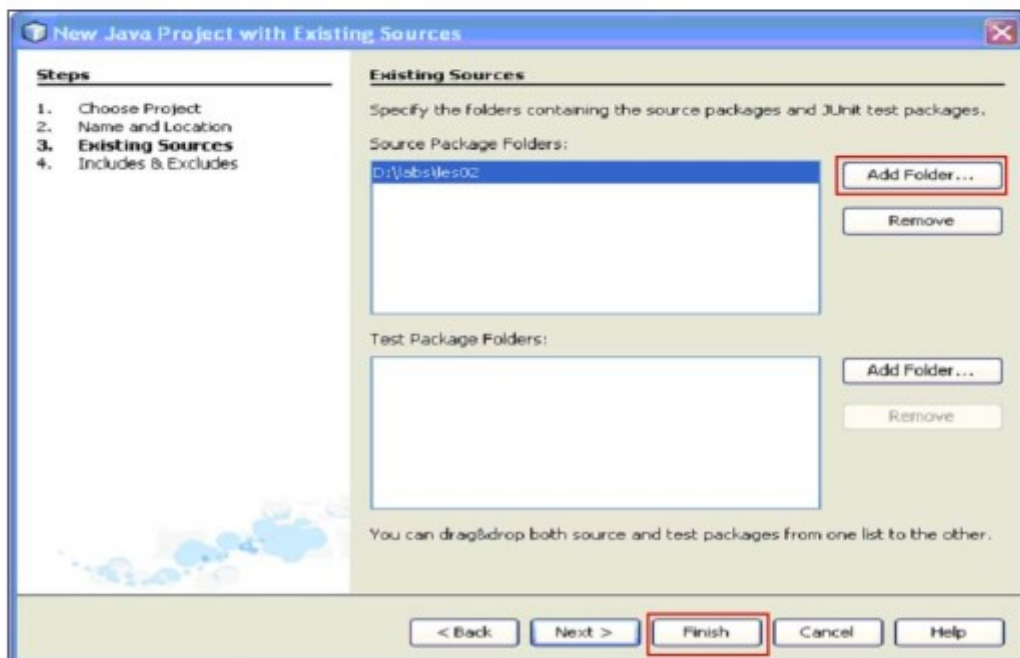
- Select **File > New Project** from the main NetBeans menu. The New Project wizard opens.
- In the **Choose Project** step of the wizard (shown in the left column), select "Java" from the Categories column. Select "Java project with existing source" from the Projects column. Click **Next**.



- c. In the **Name and Location** step of the wizard, enter "Practice02" for the Project Name and deselect the **Set as Main Project** check box. Click **Next**.



- d. In the **Existing Sources** step of the wizard, add D:\labs\les02 to the Source Packages Folder panel by clicking **Add Folder** and browsing to the desired directory. Click **Finish**.

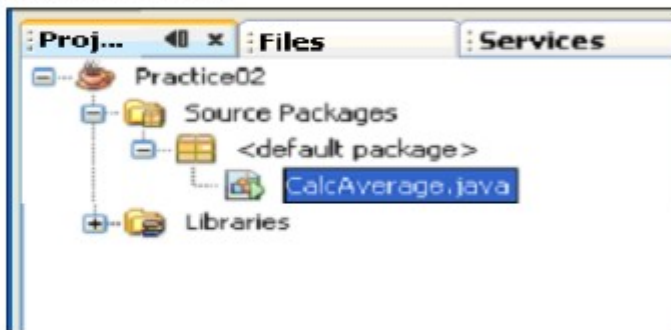


- e. You are now prompted with the message "The specified package folders contain compiled class files". Click **Delete** to delete the CalcAverage.class file that was

generated in the previous practice when you compiled the CalcAverage.java file from the DOS console. NetBeans will generate a new class file for you in this practice.



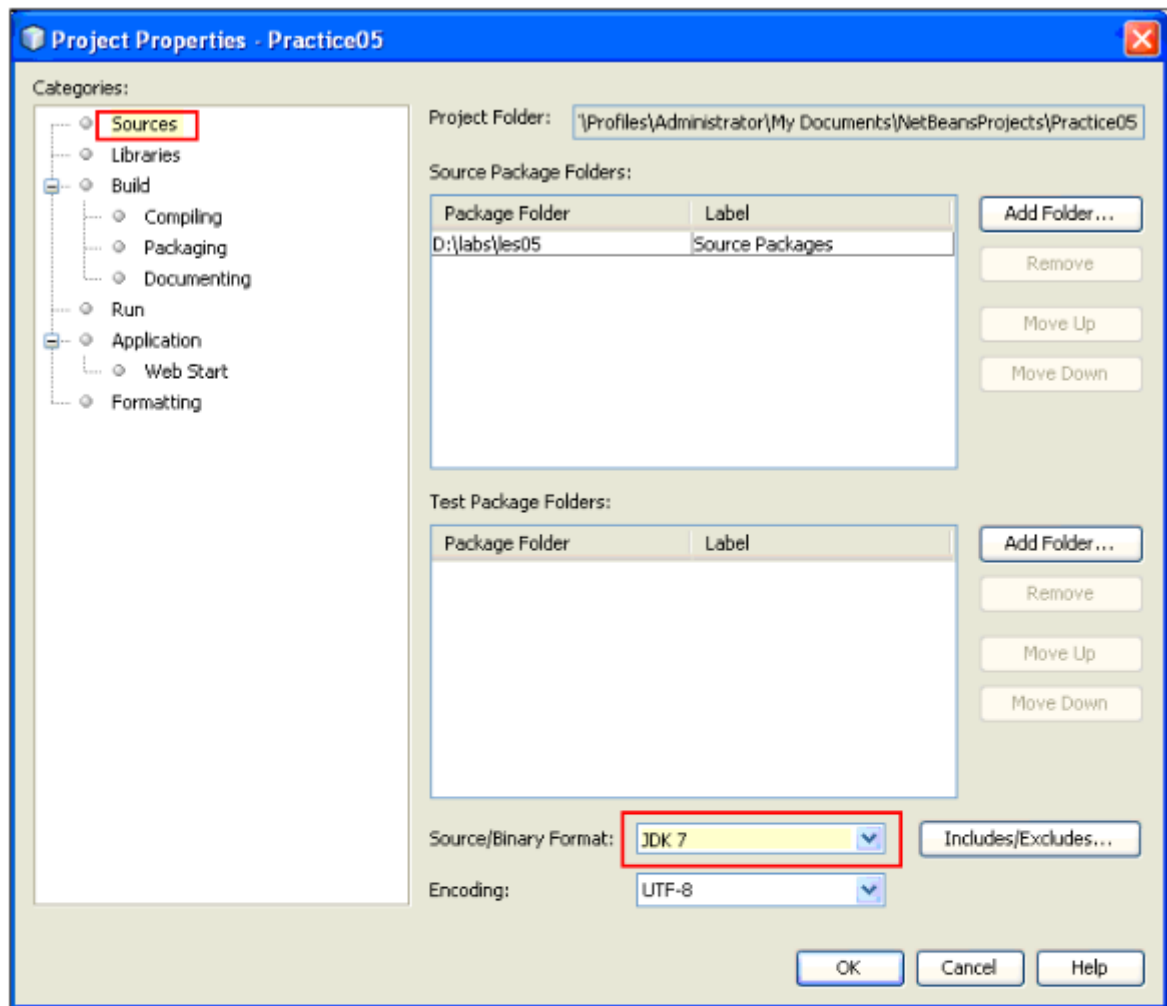
The contents of the project are now displayed in the **Projects** window within upper left pane of NetBeans. Click the Projects tab if necessary to view the Projects window. Here you see the project name at the root node. Expand the nodes beneath that to find CalcAverage.java.



4. Modify the properties of this project to set the Source/Binary Format property to JDK 7. This will allow you to use any new language features of Java SE 7 without getting an error message from NetBeans. The table below provides the high-level steps. If you need more details, follow the steps below the table.

Step	Window/Page Description	Choices or Values
a.	Main menu	File > Project Properties (Practice02)
b.	Project Properties window Source category	Source/Binary Format field = JDK 7
c.	Project Properties window Libraries category	Confirm that Java 7 is listed as the Java Platform
d.	Project properties window	Click OK

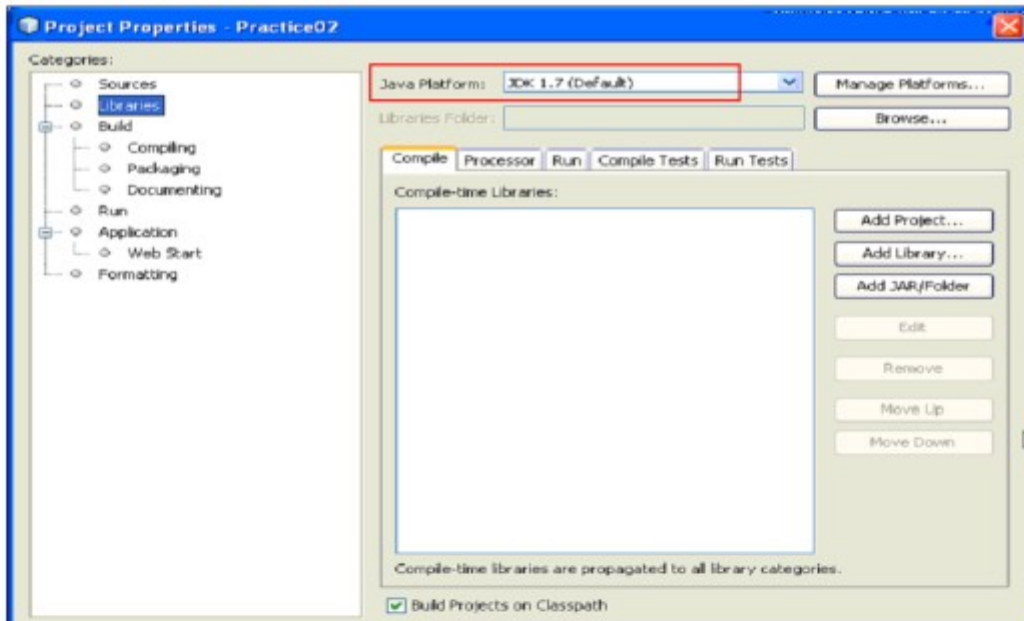
- a. Select **File > Project Properties (Practice02)** from the main menu. (Alternatively, right-click the **Practice02** project node in the Projects window and select **Properties**). The Project Properties window opens.
- b. Select **Sources** in the Categories column. Set the **Source/Binary Format** field to "JDK 7".



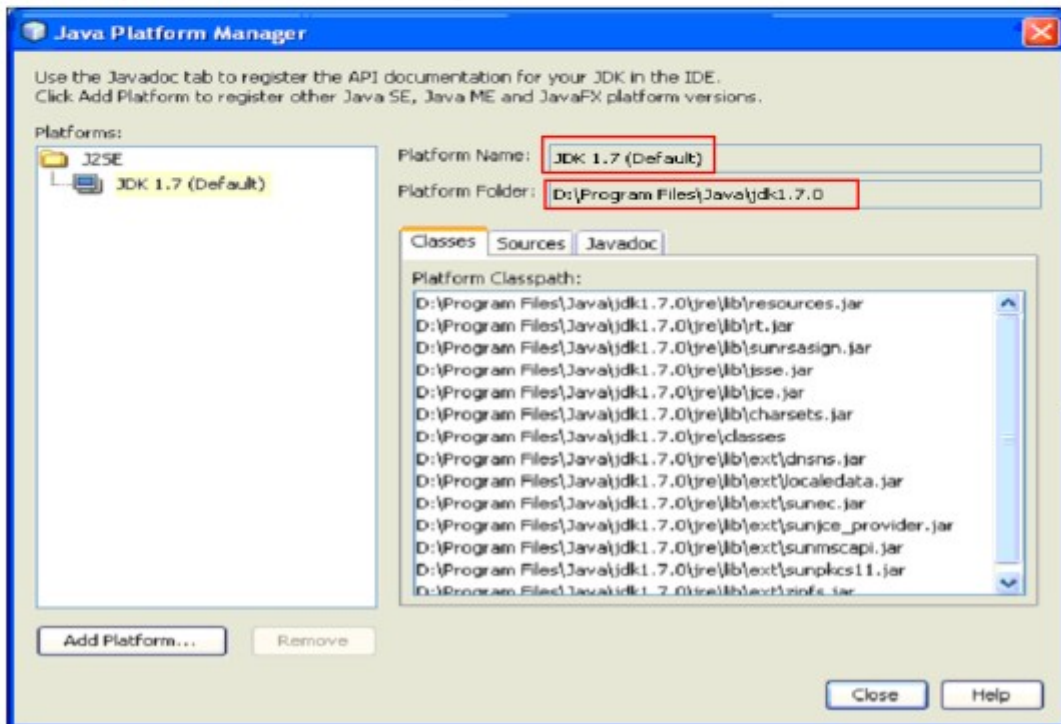
Note: NetBeans allows you to specify the lowest Java platform version with which the generated code should be compatible. For example, if you had not changed this setting to JDK 7, you would have seen error messages when using any of the core language changes included in JDK 7. NetBeans would warn you that the code would be incompatible with an earlier version.

Remember that when you compiled and ran this java file from the command prompt, you had to manually set the PATH to point to the JDK 7 installation. When you use an IDE, it automatically sets a default JDK runtime environment for each NetBeans project.

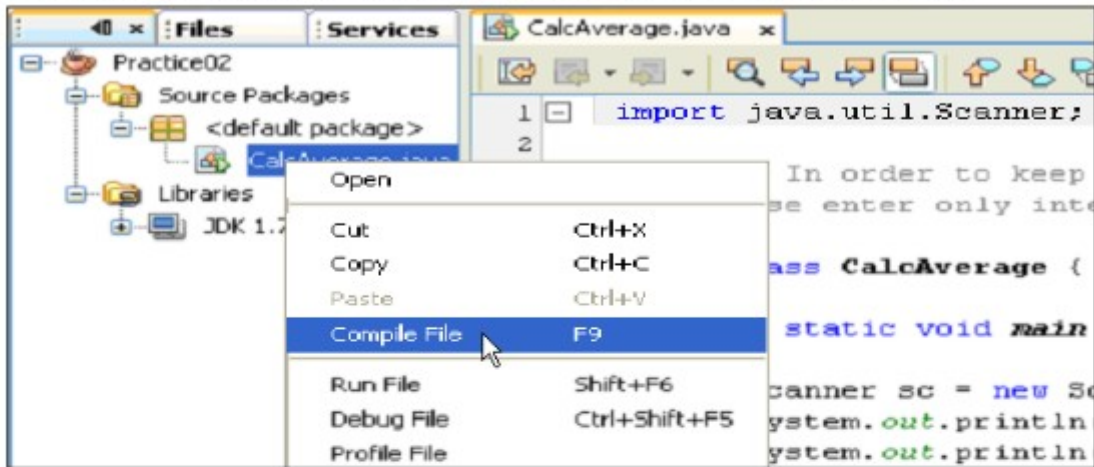
- c. Confirm that the Java Platform setting for the Practice02 project is **JDK 7**. Select the **Libraries** node in the Categories column. On the right, the JDK 7 is listed as the Java Platform for this project. Notice that you could select a different platform (JDK version) if you wished (assuming other platforms had been properly installed on this machine).



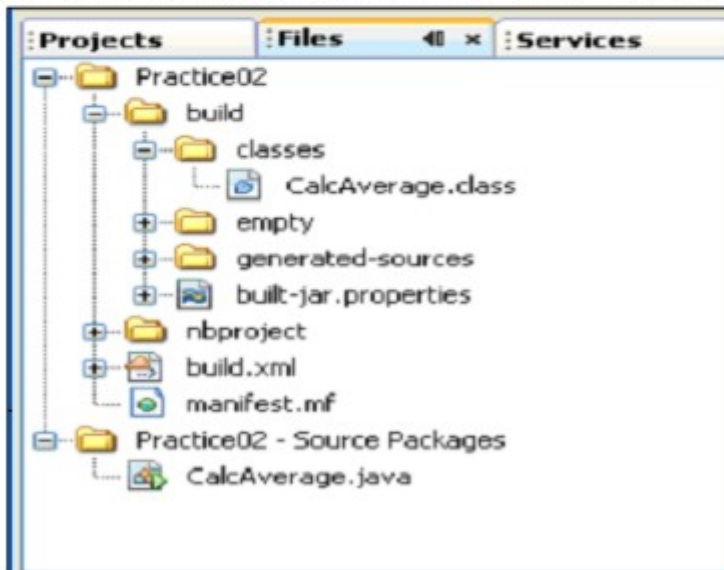
- d. Click **OK** to save the change you made in step b to the project properties.
5. To determine or change the default Java Platform for NetBeans, select **Tools > Java Platforms** on the main menu. This window shows all versions of the JDK that have been properly installed on this machine. In our case, only the JDK 7 (a.k.a. JDK 1.7) has been installed so it is marked as the "Default" platform in the Platforms column. On the right, the directory location for the JDK 7 installation is shown. Close the Java Platform Manager window when you have finished examining it.



- To view and edit the code for the CalcAverage.java file, double click it in the Project's window. It opens in the Editor pane. Notice the color coding used by the editor. (For example, keywords are in blue, string literals are in red.) This makes working with and reading you code much easier. You learn more about using this editor in upcoming practices.
- In the Projects window, right-click CalcAverage.java, and choose **Compile File**.

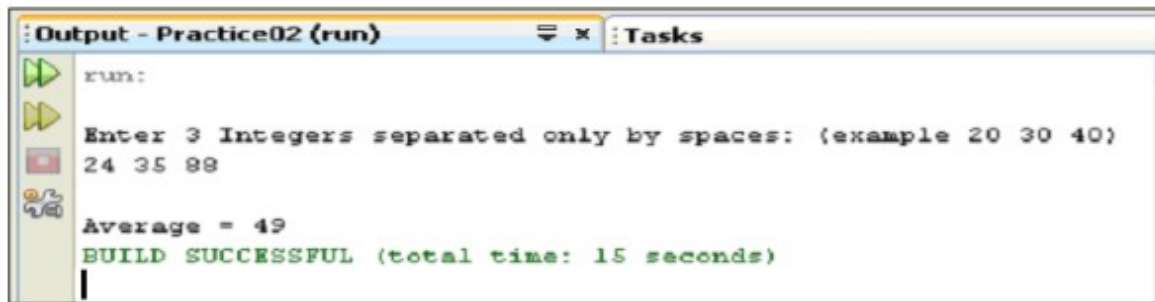


- Assuming you had no compilation errors, you can now find the .class file by clicking the Files window and expanding **Practice02 > build > classes**.



Note: If you had made any changes to the java file, the Save button would have become enabled. By default, compilation occurs automatically with a Save.

- Click the Projects window again. Right-click the file and choose **Run File**. The output from the program appears in the Output window. Enter the three integer values in the line beneath the output message and press Enter to see the result.



The screenshot shows the NetBeans IDE's Output window for a project named 'Practice02'. The window title is 'Output - Practice02 (run)'. The output text is as follows:

```
run:  
Enter 3 Integers separated only by spaces: (example 20 30 40)  
24 35 88  
Average = 49  
BUILD SUCCESSFUL (total time: 15 seconds)
```

Now you have seen how to run a simple Java program using both the DOS command prompt and the NetBeans IDE.

10. Close the Practice02 project in NetBeans. In the Projects window, right-click Practice02 and select **Close** from the context menu.

Practice 3: Thinking in Objects

Practices Overview

In these practices, you first analyze a problem using object-oriented analysis, and then you design a possible solution by using UML-like notation. Solutions for these practices can be found in `D:\labs\soln\les03`.

Practice 3-1: Analyzing a Problem Using Object-oriented Analysis

Overview

In this practice, you analyze a case study and use object-oriented analysis to list the objects, attributes, and operations in the case study.

Preparation

Read the following case study, and then model the system by choosing objects and their attributes and operations.

Case Study

A soccer league needs a system to track team and player standings.

At any moment, administrators want to be able to report a list of games played with results, a list of teams ranked by wins, and a list of players on each team ranked by goals scored.

Tasks

Your task is to produce an object-oriented analysis for a Java technology application that tracks soccer scores. The program should track:

- The list of *players* on each *team* ranked by *goals* scored
- The list of *games* played with results
- The list of teams in the *league* ranked by wins

Hint: You can think of the objects as nouns, attributes as adjectives, and operations as verbs. As an example, a *Player* is a noun, the player's name is an adjective that describes that noun, and *add goal* is a verb.

The application should be able to generate statistics for teams, players, and seasons.

1. Open the text editor by selecting Start > Programs > Accessories > Notepad.
2. Save the file as D:\labs\les03\oo-analysis.txt.
3. To get started, list the high-level classes that are included in this problem. You can list them in the text editor and use dashed lines to separate the objects, attributes, and operations as shown in the screenshot.

```

oo_analysis.txt - Notepad
File Edit Format View Help

Player
-----
id
name
number
*Team

Team
-----
id
name
*Player(s)
-----
Get ranked player

```



4. (Optional) You can use the UMLet tool if you choose. Double-click the UMLet icon from the Windows desktop to launch the program.

Solution

Player	Team	Game	League	Goal
id name number *Team	id name *Player(s)	id team one score team two score *Goal	*Team(s) *Game(s)	id *Team *Player time
	Get ranked player	Get results	Get game results Get ranked teams	

The asterisk (*) denotes attributes that are also objects.

Practice 3-2: Designing a Programming Solution

Overview

In this practice, you continue with Practice 3-1 by using UML-like notation to represent the classes you identified.

Assumptions

You have completed identifying the objects, attributes, and operations that you found in Practice 3-1.

Tasks

Your task is to produce a design for each of the classes in the earlier system for tracking soccer scores. Remember to:

- Use camelCase to name your classes, attribute variables, and methods
 - Identify a valid range of values for each attribute (where a range is known)
 - Use square brackets to indicate an attribute that represents a collection of values (players[])
 - Use parentheses to identify methods
1. Open `D:\labs\les03\oo-analysis.txt` and save it as `D:\labs\les03\oo-design.txt`.
 2. Use the classes, variables, and operations that you identified in the previous practice, and develop method names for the operations. The screenshot below is an example.



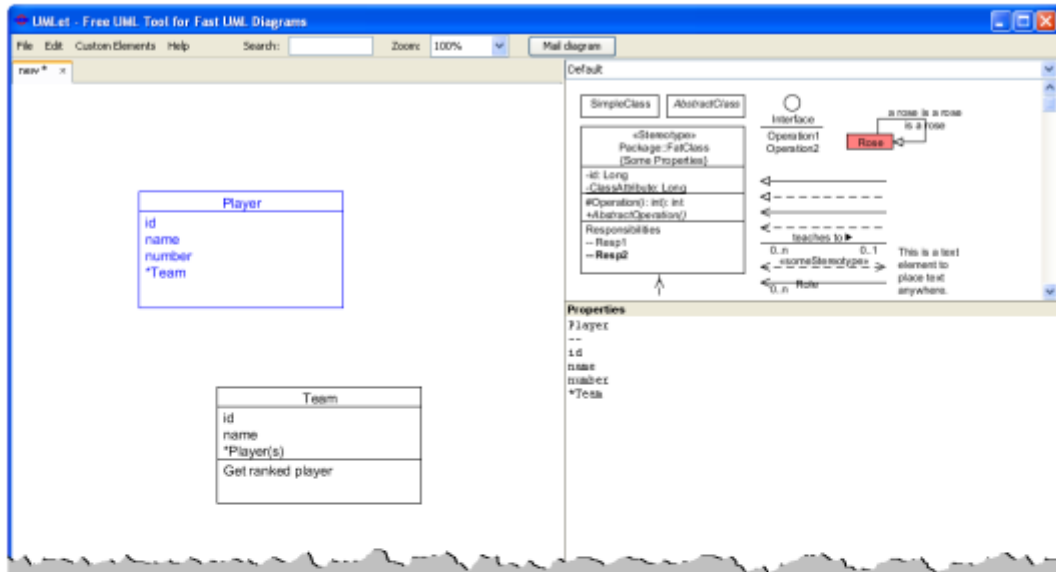
```
oo-design.txt - Notepad
File Edit Format View Help

Player
-----
id
name
number
team

Team
-----
id
name
players[ ]
-----
getRankedPlayers()
```




- (Optional) You can use the UMLet tool if you choose. Double-click the UMLet icon from the Windows desktop to launch the program.



Solution

Player	Team	Game	League	Goal
id name number team	id name players[]	id team one score team two score goals[]	teams[] games[]	id team player time
	getRankedPlayers()	getResults()	getGameResults() getRankedTeams()	

Note: Although not shown in the solution, you need add/remove methods for each collection attribute and get/set methods for all other attributes. We have not discussed those methods at this point in the course.

Your solutions might look different from the suggested solution. The purpose of this lesson is to help you continue thinking in terms of objects, attributes, and operations. You have another opportunity during this course to practice modeling a programming solution.

Practice 4: Introducing the Java Language

Practices Overview

In these practices, you examine and modify existing Java programs and also run them to test the program. Solutions for these practices can be found in `D:\labs\soln\les04`.

Practice 4-1: Viewing and Adding Code to an Existing Java Program

Overview

In this practice, you are given a completed Java program. You open it, examine the lines of code, modify it, compile it, and then test it by executing the program.

Assumptions

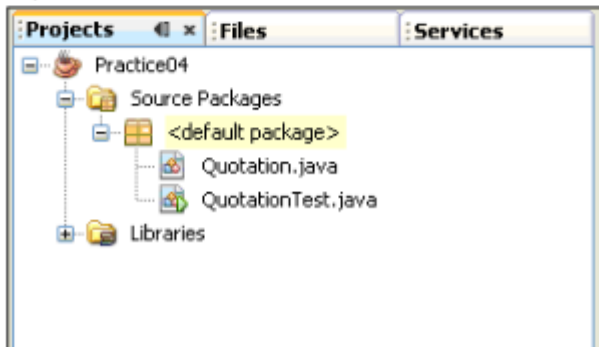
- Quotation.java and QuotationTest.java appear in the folder for this practice:
D:\labs\les04

Tasks

1. Create a new project from existing Java source, just as you did in Practice 2-2. The high-level steps are shown in the table below. If you need further detail, refer to Practice 2-2, steps 3 and 4.

Step	Window/Page Description	Choices or Values
a.	Menu	File > New Project
b.	New Project wizard Choose Project step	Category: Java Project: Java Project with Existing Sources Next
c.	New Project with Existing Sources wizard Name and Location step	Project Name: Practice04 Next
d.	New Project with Existing Sources wizard Existing Sources step	Add Folder: D:\labs\les04 Finish
e.	Project Properties window Source category	Source/Binary Format: JDK 7 OK

Note: The Projects window should now look like this when the **<default package>** node is expanded:



2. Double-click the `Quotation.java` file in the Projects window to open it for editing.

- Identify the field and the method contained within this class, using the table below:

Member	Variable or Name
Field variable:	
Method name:	

Solution: Field variable – `quote`; Method name – `display`.

- In the `display` method, write the code to display the `quote` field. *Hint:* Use the `System.out.println` method shown in the Student Guide for this lesson. Be sure to finish the line of code with a semicolon.

Note: You will notice, as you type the code, that NetBeans' code assist feature provides feedback and help whenever you pause in your typing. For instance, if you stop at some point at which the code, as is, would not compile successfully, it displays a red exclamation mark in the left margin. If you pause after typing the dot (".") following `System` or `out`, it gives you context sensitive help in the form of a list of methods and fields that would be valid for the particular class to the left of the dot. You can select from the list instead of typing.

Solution:

```
System.out.println(quote);
```

```

1 public class Quotation {
2
3     public String quote = "Welcome to Oracle, the new home of Java!";
4
5     public void display() {
6         // display the member variable here
7         System.out.println(quote);
8     }
9 }

```

- Click the Save button to save and compile `Quotation.java`.
- Open the `QuotationTest.java` file in the editor and examine its `main` method. It creates an instance of the `Quotation` class and then calls its `display` method.
- Run the `QuotationTest` class by right-clicking `QuotationTest.java` in the Projects window and selecting **Run File**. The output from the `display` method appears in the Output window.

Note: You were able to skip the Compile step because when you select Run File, NetBeans first compiles not only the class you selected to run, but also any referenced classes within that class (`Quotation.java`).

```

run:
Welcome to Oracle, the new home of Java!
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Edit the `Quotation.java` file now to change the default value of the `quote` field.

9. Run `QuotationTest` again to verify the output.
10. In the Editor pane, close `Quotation.java` and `QuotationTest.java`.

Practice 4-2: Creating and Compiling a Java Class

Overview

In this practice, you create a Java class and compile it. You also create another Java class to test the previous class.

Assumptions

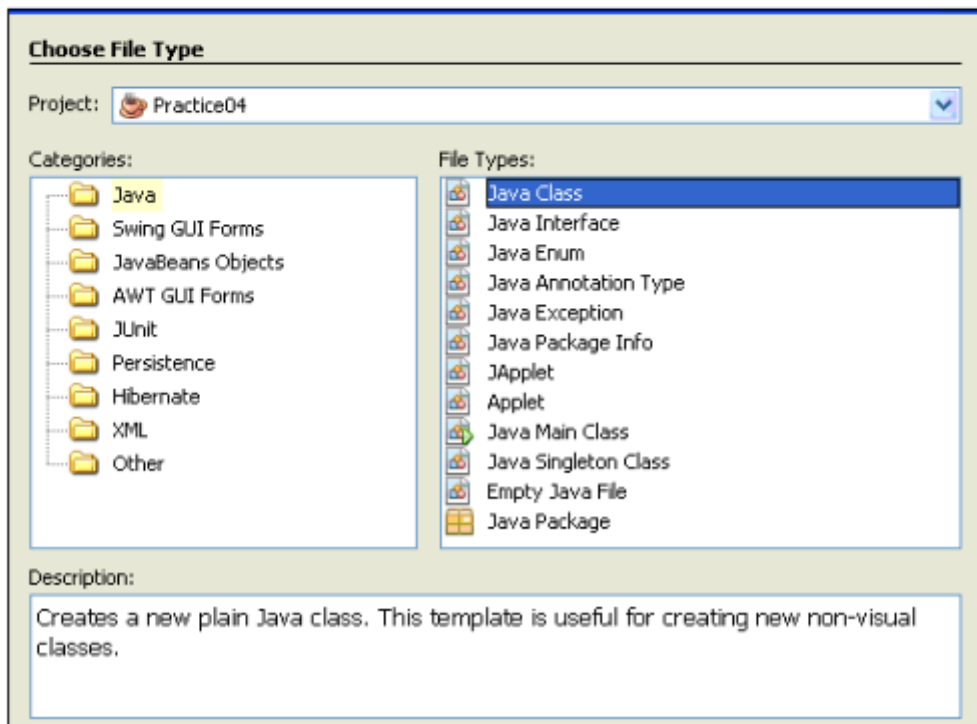
None

Tasks

1. Create a new Java class in the Practice04 project using the NetBeans wizard. The high-level steps for this task are shown in the table below. If you need more assistance, you can use the detailed steps that follow the table.

Step	Window/Page Description	Choices or Values
a.	Menu	File > New File
b.	New File window Choose File Type step	Category: Java File Types: Java Class Next
c.	New Java Class window Name and Location step	Class Name: <code>Shirt</code> Finish

- a. From the main menu, select **File > New File**.
- b. The New File wizard opens and you are on step 1 "Choose File Type". Select **Java** in the Category column. Select **Java Class** in the File Types column. Click **Next**.



- c. In the New Java Class window, you are on step 2 "Name and Location". Enter "Shirt" as the Class Name. Click **Finish**.

Name and Location

Class Name:

Project:

Location:

Package:

Created File:

Warning: It is highly recommended that you do NOT place Java

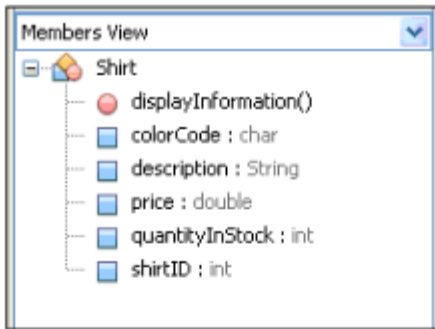
The Java source file for the new class now appears in the editor ready for you to fill in the details.

2. Enter the Java code syntax for the Shirt class shown in this lesson of the Student Guide.

Solution: You can find the solution code for the Shirt class in `D:\labs\soln\les04`

3. Click the Save button to save and compile the Shirt class. Any red error icons in the left margin should disappear after saving if there were no compilation errors. If necessary, fix any errors that appear in the Output window and save again.

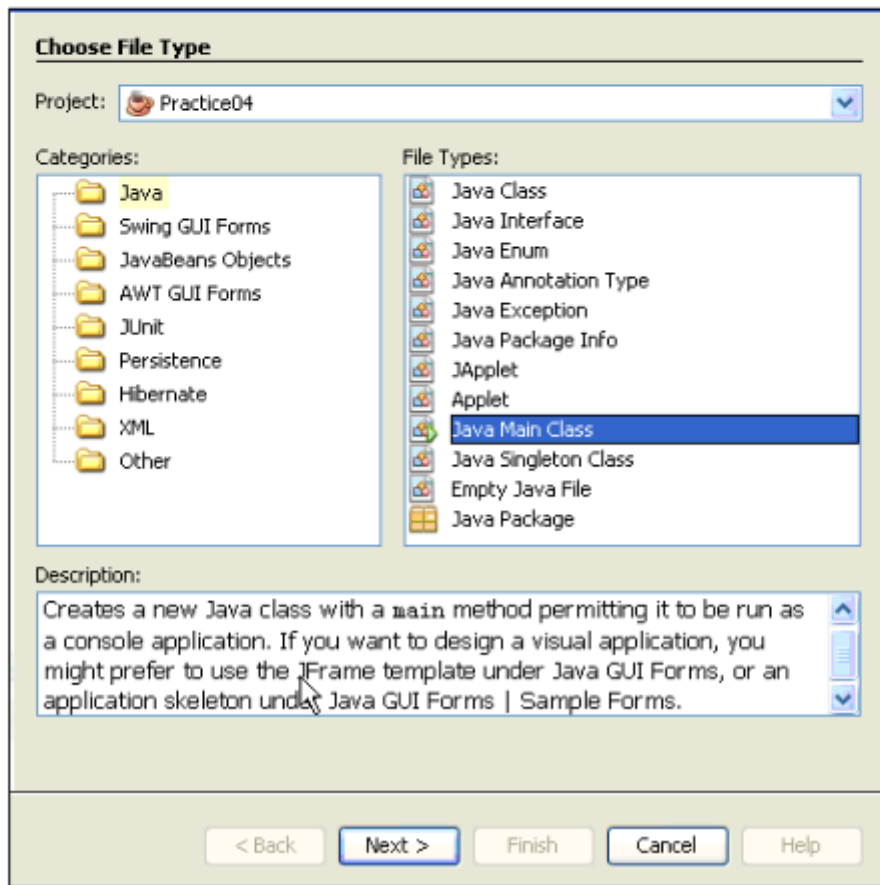
Note: The Navigator pane (lower left corner of NetBeans) for the `Shirt` class now shows the Members view of the class. Notice the color coding that distinguishes between fields and methods. Both of these are considered "Members" of the class.



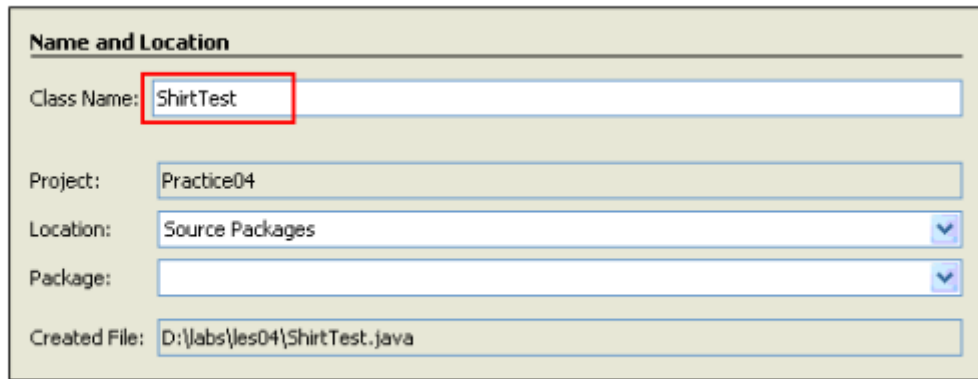
4. Follow the instructions from Step 1 to create another new class. This will be a Test class, so it will need a `main` method. To accommodate that change, the table below shows the substitutions in the Step 1 instructions you should make as you go through the New Class wizard. For more detail, see the screenshots following the table.

Step	Window/Page Description	Choices or Values
a.	New File window Choose File Type step	File Types: Java Main Class
b.	New File window Name and Location step	Name: ShirtTest

- a. In the Choose File Type step, select **Java Main Class** instead of Java Class.



- b. In the Name and Location step, enter **ShirtTest** as the name.



Name and Location

Class Name:

Project:

Location:

Package:

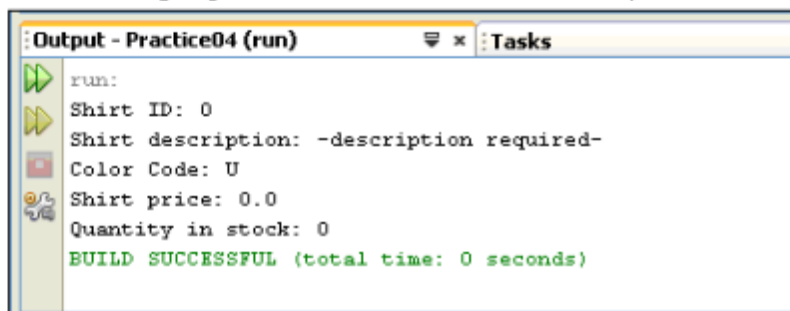
Created File:

5. Replace the **To Do:** comment in the `main` method with the two lines of code that appear in the `main` method for the `ShirtTest` class shown in this lesson of the Student Guide.

Solution: You can find the solution code for the `ShirtTest` class in `D:\labs\soln\les04`

```
10 public class ShirtTest {
11     /**
12      * @param args the command line arguments
13      */
14     public static void main(String[] args) {
15         Shirt myShirt;
16         myShirt = new Shirt();
17         myShirt.displayInformation();
18     }
19 }
```

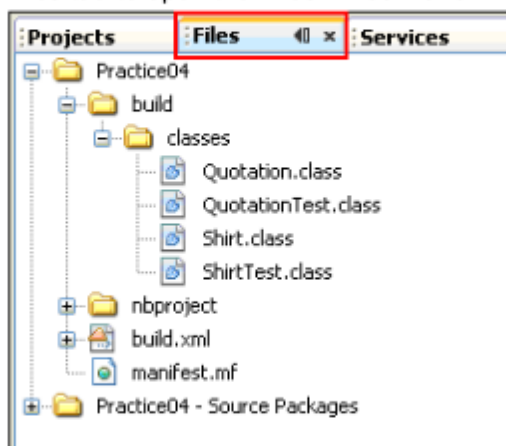
6. Save and compile the code by clicking Save.
7. Run the `ShirtTest` class by right-clicking `ShirtTest.java` in the Projects window. Look for the output of the `displayInformation` method in the Output window.



The screenshot shows the 'Output - Practice04 (run)' window with the following text:

```
run:
Shirt ID: 0
Shirt description: -description required-
Color Code: U
Shirt price: 0.0
Quantity in stock: 0
BUILD SUCCESSFUL (total time: 0 seconds)
```

8. Find the class files that were generated by NetBeans when you ran the program. Click the Files tab to open the Files window and find `Shirt.class` and `ShirtTest.class` as shown below.



9. Open (or return focus to) the `Shirt.java` file. Modify the values of `Shirt ID` and `price`.
10. Run the `ShirtTest` class again. Verify that the modified values are shown in the Output window.

Practice 4-3: Exploring the Debugger

Overview

Virtually every Java IDE provides a debugger. They tend to offer the same core features and work very similarly. In this practice, you debug the ShirtTest program using the NetBeans debugger. You set breakpoints, examine field values, and modify them as you step through each line of code.

Assumptions

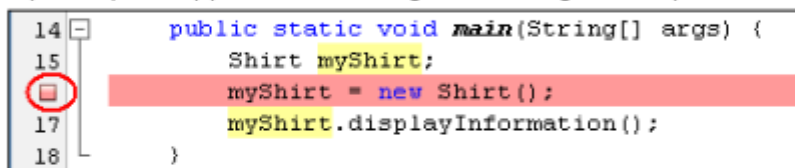
None

Tasks

1. Set a breakpoint in the ShirtTest class. Click in the left margin of the editor, next to the following line of code:

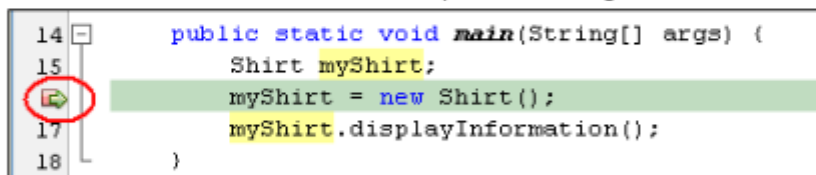
```
myShirt = new Shirt();
```

A pink square appears in the margin, indicating a breakpoint.



```
14 public static void main(String[] args) {
15     Shirt myShirt;
16     myShirt = new Shirt();
17     myShirt.displayInformation();
18 }
```

2. Run the debugger by right-clicking on the ShirtTest file in the Projects window and selecting **Debug File**.
3. The debugger starts the program and stops at the breakpoint. In the Editor panel you should now see a different icon that points with a green arrow to the line of code.



```
14 public static void main(String[] args) {
15     Shirt myShirt;
16     myShirt = new Shirt();
17     myShirt.displayInformation();
18 }
```

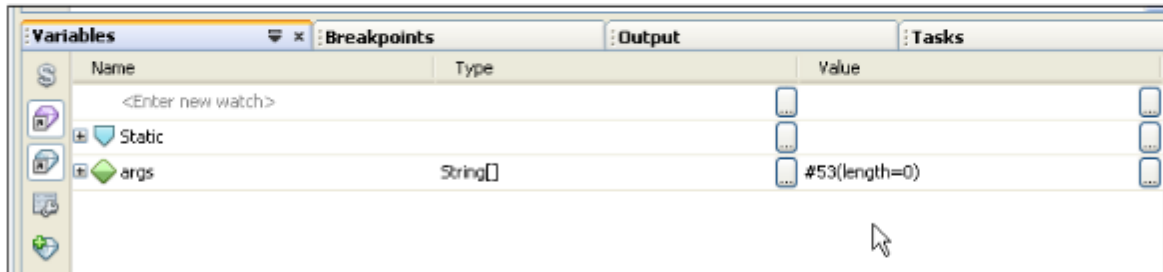
This line of code has not yet been executed.

4. Several other changes have occurred in the NetBeans window.
 - A new toolbar appears, containing buttons that you use when debugging.




- Move your cursor over each of the toolbar buttons to read the toolbar tip explaining what each button does. The buttons are described below.
 - The first button, **Finish Debugger Session**, stops the debugging session.
 - The second button, **Pause**, pauses the execution of the debugger.
 - The third button **Continue** the execution, either to the next breakpoint or to the end of the program.
 - The fourth button, **Step Over**, moves the program forward to the next line of code in the current class (in this case, the ShirtTest class).
 - The fifth button, **Step Over Expression**, allows you to step over an entire expression to the next line of code in the current class.

- The sixth button, **Step Into**, allows you to step into another class referenced in this current line of code.
- The seventh button, **Step Out**, allows you to step back out of a class that you stepped into.
- The last button, **Run to Cursor**, takes execution to the line of code where the cursor appears.
- The panel at the bottom of the window changes to show debugging output and variables and other useful information during a debug session.



- In the Variables panel, you see all variables that are visible to the current class. Remember that the execution was stopped *before* the `Shirt` class object has been instantiated. Consequently, you do not see the `myShirt` variable in this panel.


5. Click the Step Over button to move to the next line of code. 
6. The arrow now points to the line of code that calls the `displayInformation` method on the `myShirt` object. In the Values window, you now see the `myShirt` variable. Expand it to see all of the fields of this `Shirt` object.

```

14 public static void main(String[] args) {
15     Shirt myShirt;
16     myShirt = new Shirt();
17     myShirt.displayInformation();
18 }
19 }

```

At this point, the `displayInformation` method has not yet been executed. You could change the values of the object's fields right now, using the Variables window if you wanted to. However, instead, you "step into" the `myShirt` object and change the values during the execution of the `displayInformation` method.

7. Click the Step Into button to step into the `displayInformation` method. 

- The arrow icon is pointing to the first executable line of code within the `displayInformation` of the `Shirt` class. In the Variables window, expand **this** to see the fields of this object.

```

19 public void displayInformation() {
20     System.out.println("Shirt ID: " + shirtID);
21     System.out.println("Description: " + description);
22     System.out.println("Color Code: " + colorCode);
23     System.out.println("Price: " + price);
24     System.out.println("Quantity in stock: " + quantityInStock);

```

Name	Type	Value
<Enter new watch>		
this	Shirt	#57
shirtID	int	0
description	String	"-description required-"
colorCode	char	'\u'
price	double	0.0
quantityInStock	int	0

- In the Value column double-click each field's value and edit it to change the value. Ensure that you use the correct value for the data type expected and enclose any character data types with the type of quote mark indicated. After editing the final field, click the tab button so that the text you typed into the edit buffer is accepted.

Name	Type	Value
<Enter new watch>		
this	Shirt	#58
shirtID	int	1
description	String	"T Shirt"
colorCode	char	'R'
price	double	15.0
quantityInStock	int	3

- Click the Step Out button to return to the next line of code in the `ShirtTest` class. The `displayInformation` method will have completed.

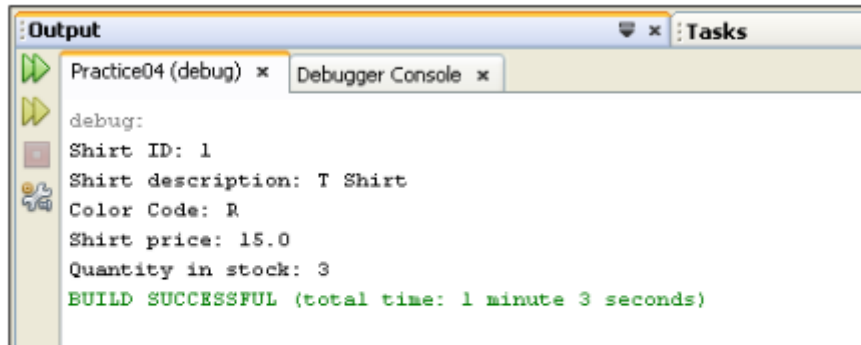
```

14 public static void main(String[] args) {
15     Shirt myShirt;
16     myShirt = new Shirt();
17     myShirt.displayInformation();
18 }

```

- Notice that the `myShirt` object field variables reflect the changes you made while in the method.
- Click the Continue button now to finish execution and end the debug session.

13. Click the Output tab to view the output.



You have now experienced some of the most commonly used features of a typical IDE Debugger. You may wish to use the debugger in remaining labs to help you diagnose and fix problems you may experience in your programs.

14. Close the Practice04 project in NetBeans. In the Projects window, right-click Practice04 and select **Close** from the context menu.

Practice 5: Declaring, Initializing, and Using Variables

Practices Overview

In these practices, you create several Java classes that declare, initialize and manipulate field variables. Solutions for these practices can be found in `D:\labs\soln\les05`.

Practice 5-1: Declaring Field Variables in a Class

Overview

In this practice, you create a class containing several fields. You declare the fields, initialize them, and then test the class by running the CustomerTest program.

Assumptions

This practice assumes that the CustomerTest Java source file appears in the practice folder for this lesson: `D:\labs\les05`

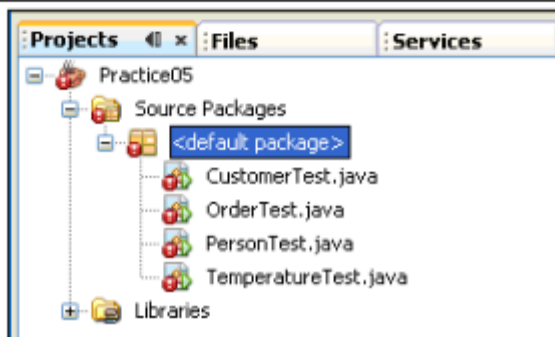
Tasks

1. Close any open project in NetBeans. In the Projects window, right-click the project name and select **Close** from the context menu.
2. Create a new project from existing Java source, using the values in the table below when you complete the New Project wizard.

Step	Window/Page Description	Choices or Values
a.	Choose Project step	Category: Java Project: Java Project with Existing Sources
b.	Name and Location step	Project Name: Practice05
c.	Existing Sources step	Add Folder: <code>D:\labs\les05</code>
d.	Project Properties window	Set the Source/Binary Format property to JDK 7

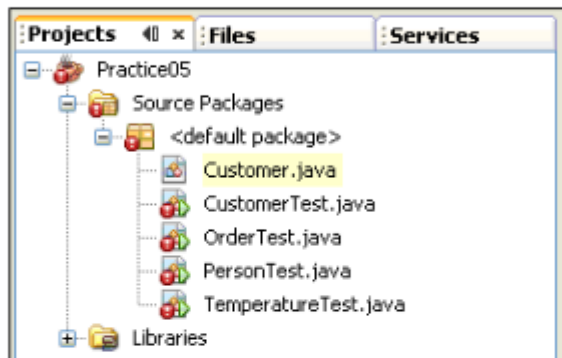
Note: If you need a more detailed reminder of how to create a new project, refer to Practice 2-2, steps 3 and 4.

Solution: The Projects window should show four Java source files beneath the <default package> node.



3. Create a new Java class. The table below provides the high level steps. If you need more assistance, refer to Practice 4-2, step 1.

Step	Window/Page Description	Choices or Values
a.	Menu	File > New File
b.	New File window Choose File Type step	Category: Java File Types: Java class Next
c.	New Java Class window Name and Location step	Class Name: Customer Finish



4. With `Customer.java` open for editing in the Editor pane, declare and initialize the fields described in the table below. If you need more assistance, more detailed steps are provided following the table.

Field Name	Data Type	Default Value
<code>customerID</code>	<code>int</code>	<your choice>
<code>status</code>	<code>char</code>	<your choice> 'N' for new, 'O' for old
<code>totalPurchases</code>	<code>double</code>	0.0

- The syntax of a variable declaration and initialization is:
`modifier type variable = <value>;`
- Assume that all fields are `public`.
- Include a comment at the end of each line describing the field.

Solution: This shows one possible solution for the `customerID` declaration and initialization. The others are similar.

```
public int customerID = 0; // Default ID for a customer
```

5. Add a method within the `Customer` class called `displayCustomerInfo`. This method uses the `System.out.println` method to print each field to the screen with a corresponding label (such as "Purchases are: ").

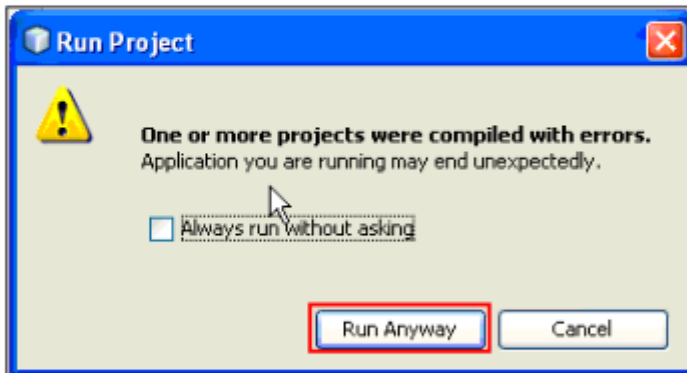
Solution:

```
public void displayCustomerInfo () {  
    System.out.println("Customer ID: " + customerID);  
    // continue in a similar fashion for all other fields  
}
```

6. Click Save to compile the class.

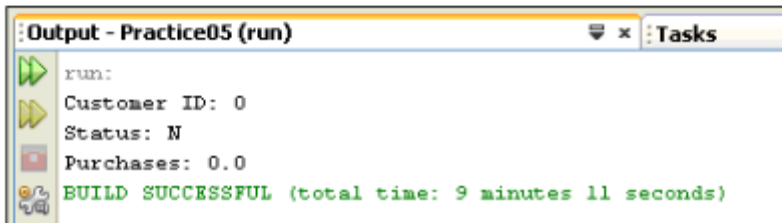
Note: You will notice that the red error indicator next to the CustomerTest class in the Projects window disappears after saving the Customer class. The reason is that the CustomerTest class references the displayCustomerInfo method, which did not exist before you saved the file. NetBeans recognized a potential compilation error in the CustomerTest class, due to the missing method.

7. Run the CustomerTest class to test your code. If you are prompted with a warning indicating that there are compilation errors within the project, click **Run Anyway**.



Note: All of the examples and practices in this course require a test class. In most situations, the test class is provided. However, in some situations, you create the class.

8. Check the output to be sure that it contains the values you assigned.



Practice 5-2: Using Operators and Performing Type Casting to Prevent Data Loss

Overview

In this practice, you use operators and type casting. This exercise has three sections. In each section you create one Java class, compile it, and test it.

Assumptions

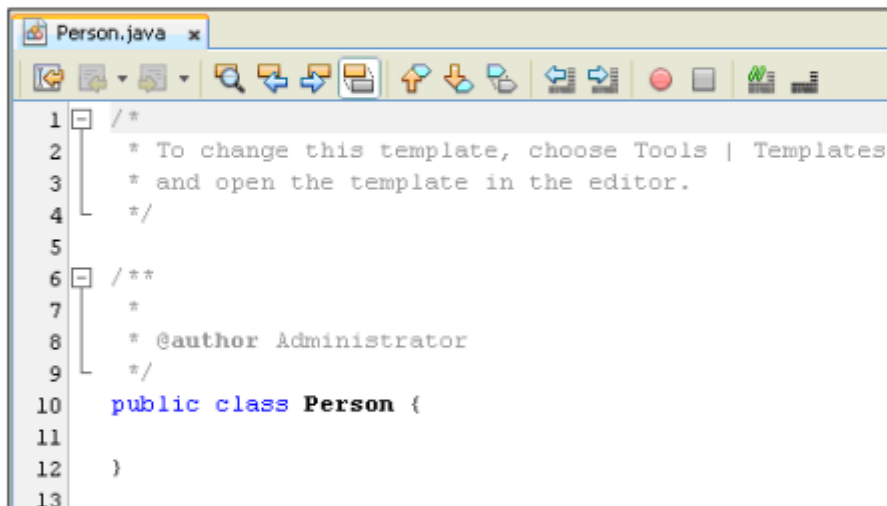
The following Java source files appear in the practice folder for this lesson: D:\labs\les05

- PersonTest.java
- OrderTest.java
- TemperatureTest.java

Calculating Age Using Operators

In this task, you use operators to calculate age in days, minutes, seconds, and milliseconds.

1. Select File > New File from the menu to create a new Java class called Person.



```
1  /*
2  |  * To change this template, choose Tools | Templates
3  |  * and open the template in the editor.
4  |  */
5
6  /**
7  |  *
8  |  * @author Administrator
9  |  */
10 public class Person {
11
12 }
13
```

2. Using the editor, add the following fields to store age in years, days, minutes, seconds, and milliseconds. Provide meaningful names for all the fields. The table below provides more detailed information:

Year Part	Data Type	Additional Info
Years	int	Initialize to 1
Days	int	Do not initialize
Minutes	long	Do not initialize
Seconds	long	Do not initialize
Milliseconds	long	Do not initialize

Hint: You can declare multiple variables of the same type in one line by separating the variables by a comma. Be sure to end the line with a semicolon, just as you would any other line of code.

3. Create a new public method in this class called `calculateAge`.
 - a. The method should calculate age in days, minutes, seconds, and milliseconds, assigning the value to the relevant field. The following table gives you the calculations:

Year Part	Calculated By:
Days	Year * 365
Seconds	Days * 24 * 60 * 60
Minutes	Seconds / 60
Milliseconds	Seconds * 1000

- b. Print out all the ages in various units, each in a separate line with an appropriate message. For example "You are 3156000 seconds old."

Solution:

```
public void calculateAge () {
    ageDays = ageYears * 365;
    ageSeconds = ageDays * 24 * 60 * 60;
    ageMinutes = ageSeconds / 60;
    ageMilliseconds = ageSeconds * 1000;

    System.out.println ("You are " + ageDays + " days old.");
    System.out.println("You are " + ageMinutes +
        " minutes old.");
    System.out.println("You are " + ageSeconds +
        " seconds old.");
    System.out.println("You are " + ageMilliseconds +
        " milliseconds old.");
}
```

4. Save to compile the class and then run the `PersonTest.java` file.
5. Perform several tests, by setting the value of age as 1, 24, and 80 in the `Person` class.

Solution:

For one year, the results should be: You are 365 days old. You are 31536000 seconds old. You are 525600 minutes old. You are 31536000000 milliseconds old.

Using Casting to Prevent Data Loss

In this section you use casting to ensure that data loss does not occur in your programs.

6. Create a new Java class called `Order`

7. Add three fields to the `Order` class as follows:

Field Name	Data Type	Initialized Value
<code>orderValue</code>	<code>long</code>	0L (zero L)
<code>itemQuantity</code>	<code>int</code>	10_000_000
<code>itemPrice</code>	<code>int</code>	555_500

Note: The underscores used to initialize the `int` values improve the readability of your code. They have no effect on the actual numeric value of the field. The compiler strips them out. This is one of the new language features of Java 7.

8. Create a `calculateTotal` method that calculates the total order value (`itemQuantity * itemPrice`) and print it. Be sure to type cast either `itemQuantity` or `itemPrice` to a `long` so that the temp storage used to hold the outcome of the multiplication is large enough to contain a `long` value.

Solution:

```
public void calculateTotal(){
    orderValue = (long)itemQuantity * itemPrice;
    System.out.println("Order total: "+ orderValue);
}
```

9. Save `Order.java` and then test it by running `OrderTest.java`. Verify the result by using a calculator.

Solution: Result should be 5555000000000

10. Edit the `Order.java` file to remove the type casting done in the `calculateTotal` method.
11. Compile and run `OrderTest` again to see the resulting data loss that occurs without type casting.

Creating a Temperature Program

In this section, you write a program to convert temperature from Fahrenheit to Celsius.

12. Create a new Java class called `Temperature`. Add a member field to the `Temperature` class that stores the temperature in Fahrenheit. Declare the field variable with an appropriate data type, such as `int`, `float`, or `double`.
13. Create a `calculateCelsius` method. Convert the Fahrenheit temperature to Celsius by subtracting 32, multiplying by 5, and dividing by 9. Be sure to observe the rules of precedence when typing this expression.

Hint: The rules of precedence are listed here for your convenience.

- Operators within a pair of parentheses
- Increment and decrement operators
- Multiplication and division operators, evaluated left to right
- Addition and subtraction operators, evaluated left to right

Solution: This is one possible solution.

```
public class Temperature {
    public float fahrenheitTemp = 78.9F;

    public void calculateCelsius() {
        System.out.println ((fahrenheitTemp - 32) * 5 / 9);
    }
}
```

14. Compile the Temperature class and test it using the TemperatureTest class. Confirm that you get the same result running the program as you do when doing this calculation using a calculator.
15. Test the program using several values of temperature.
16. When you have finished experimenting with different values, close the Practice05 project.

Practice 6: Working with Objects

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. Solutions for these practices can be found in `D:\labs\soln\les06`.

Practice 6-1: 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.

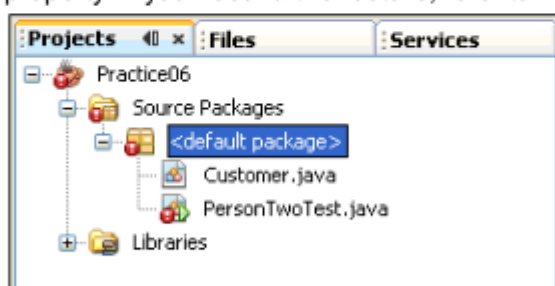
Assumptions

The `Customer.java` file appears in the practice folder for this lesson: `D:\labs\les06`

Initializing Object Instances

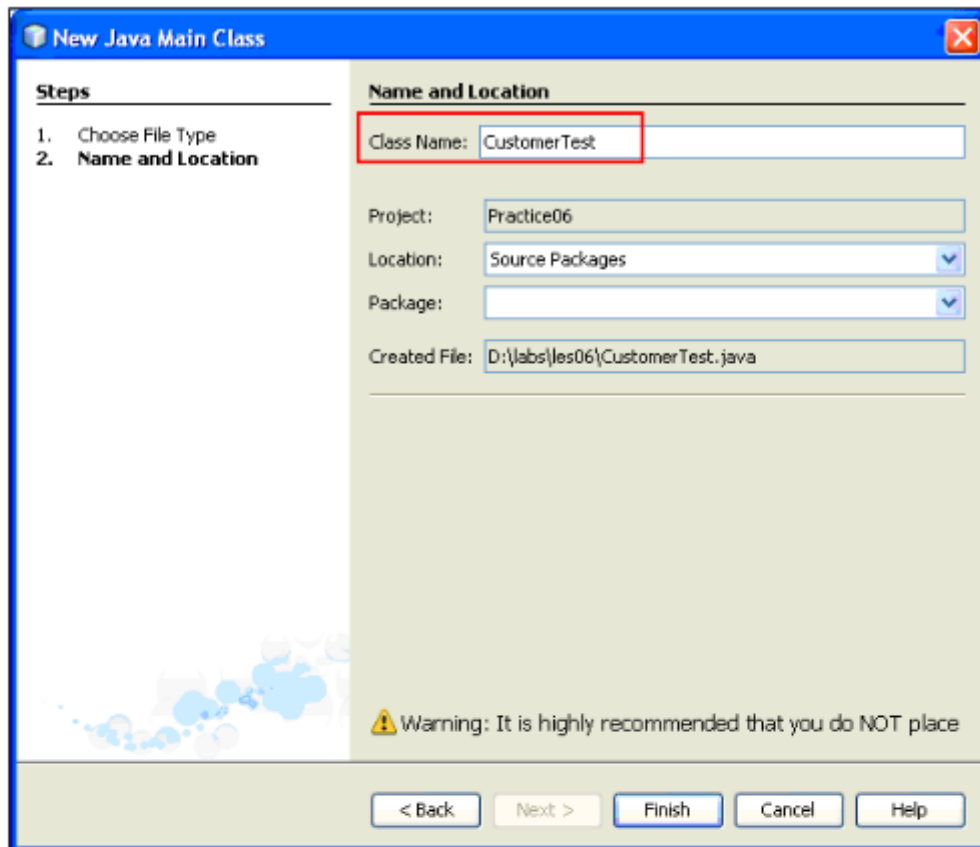
A `Customer` class is provided for you. In this section, you create, compile, and execute a `CustomerTest` class. In this test class, you create objects of the `Customer` class and set values to its member fields.

1. Create a new project from existing source called `Practice06`. Set the **Source Package Folder** to point to `D:\labs\les06`. Remember to also change the **Source/Binary Format** property. If you need further details, refer to Practice 2-2, Steps 3 and 4.

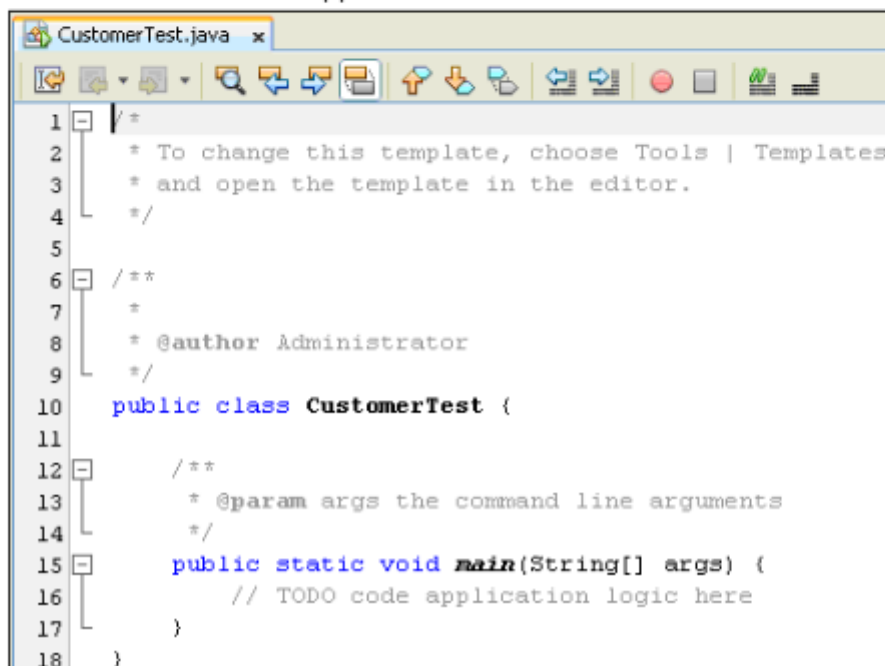


2. Open the `Customer.java` file in the editor and examine its member fields and its method. You use the field information to complete this practice.
3. Create the `CustomerTest` class as a "Java Main Class" type. Since this class is run (executed) by the Java executable, it must contain a `main` method. 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.



- 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 <code>Customer</code>	cust1 cust2
b.	Initialize the two instances	Use the <code>new</code> operator

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

- 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 <code>Customer</code> objects	Example: <code>cust1.customerID = 1;</code>
b.	Repeat for the other <code>Customer</code> object but use different values for the fields.	
c.	Invoke the <code>displayCustomerInfo</code> method of each object	Use the object reference variable to qualify the method as you did in step 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;`
- Assign different values to each member field of the other `Customer` object.
- Invoke the `displayCustomerInfo` method of each object. Example:
`cust1.displayCustomerInfo();`

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

```

Output - Practice06 (run)
run:
*****Customer Information*****
Customer ID: 1
Name:Mary Smith
Email: mary.smith@gmail.com
*****
*****Customer Information*****
Customer ID: 2
Name:Frank Jones
Email: frank.jones@gmail.com
*****
BUILD SUCCESSFUL (total time: 0 seconds)

```

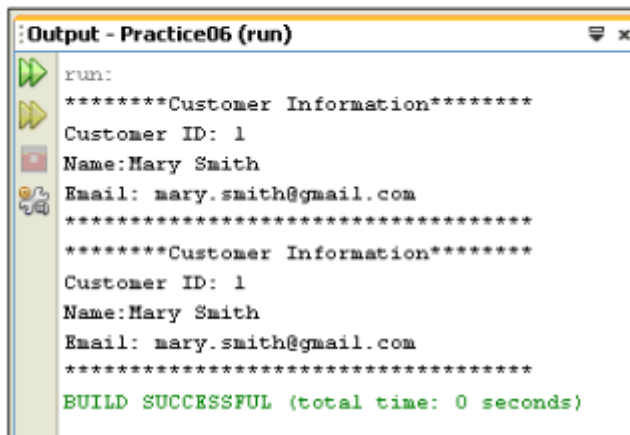
Manipulating Object References

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.



```
Output - Practice06 (run)
run:
*****Customer Information*****
Customer ID: 1
Name:Mary Smith
Email: mary.smith@gmail.com
*****
*****Customer Information*****
Customer ID: 1
Name:Mary Smith
Email: mary.smith@gmail.com
*****
BUILD SUCCESSFUL (total time: 0 seconds)
```

Practice 6-2: 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: `D:\labs\les06`

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 <code>StringBuilder</code> object	Use the <code>append</code> method of the <code>StringBuilder</code> class
b.	Append two more values to the <code>name</code> object	a space: " " a last name Note: The total number of characters appended should exceed 8
c.	Display the <code>String</code> value of the <code>name</code> object	Use the <code>toString</code> method of the <code>StringBuilder</code> class
d.	Display the capacity of the <code>name</code> object with a suitable label	Use the <code>capacity</code> method of the <code>StringBuilder</code> class
e.	Compile and run the program	Run the <code>PersonTwoTest.java</code> 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.

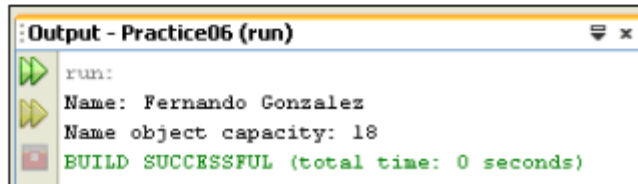
Note: You can accomplish the same thing by using a String object and concatenating additional values. However, this would be inefficient because a new String object is created with each concatenation. String object capacity cannot be increased as Strings are immutable.

- 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 <code>String</code> value to the <code>phoneNumber</code> object	Example: "5551234567"
b.	Insert a dash ("-") after the first three characters of the <code>phoneNumber</code> .	Use the <code>insert</code> method that takes an <code>int</code> value for the offset and inserts a <code>String</code> value. (Use offset number 3)
c.	Insert another dash after the first seven characters of the <code>phoneNumber</code>	Reminder: The previous insertion pushed the remaining characters over one index.
d.	Display the <code>phoneNumber</code> object	Use the <code>toString</code> method of the <code>StringBuilder</code> class

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

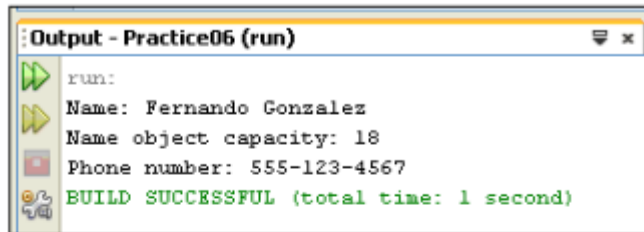
The offset position 3 occurs at the number 1. (Index numbers begin at 0.) If the dash is inserted at offset position 3, it pushes the number currently at that position and all remaining numbers over to the next offset position.

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

- 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.



```
Output - Practice06 (run)
run:
Name: Fernando Gonzalez
Name object capacity: 18
Phone number: 555-123-4567
BUILD SUCCESSFUL (total time: 1 second)
```

- 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:

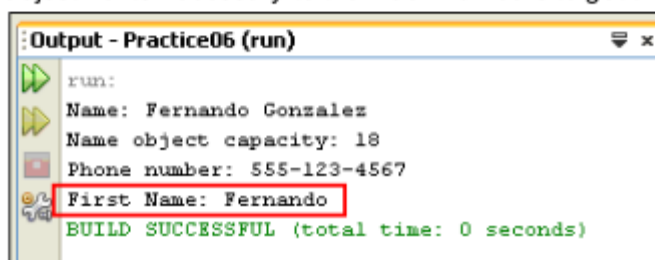
```
<object reference>.substring(int start, int end);
```

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));
```

- 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.



```
Output - Practice06 (run)
run:
Name: Fernando Gonzalez
Name object capacity: 18
Phone number: 555-123-4567
First Name: Fernando
BUILD SUCCESSFUL (total time: 0 seconds)
```

Practice 6-3: 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

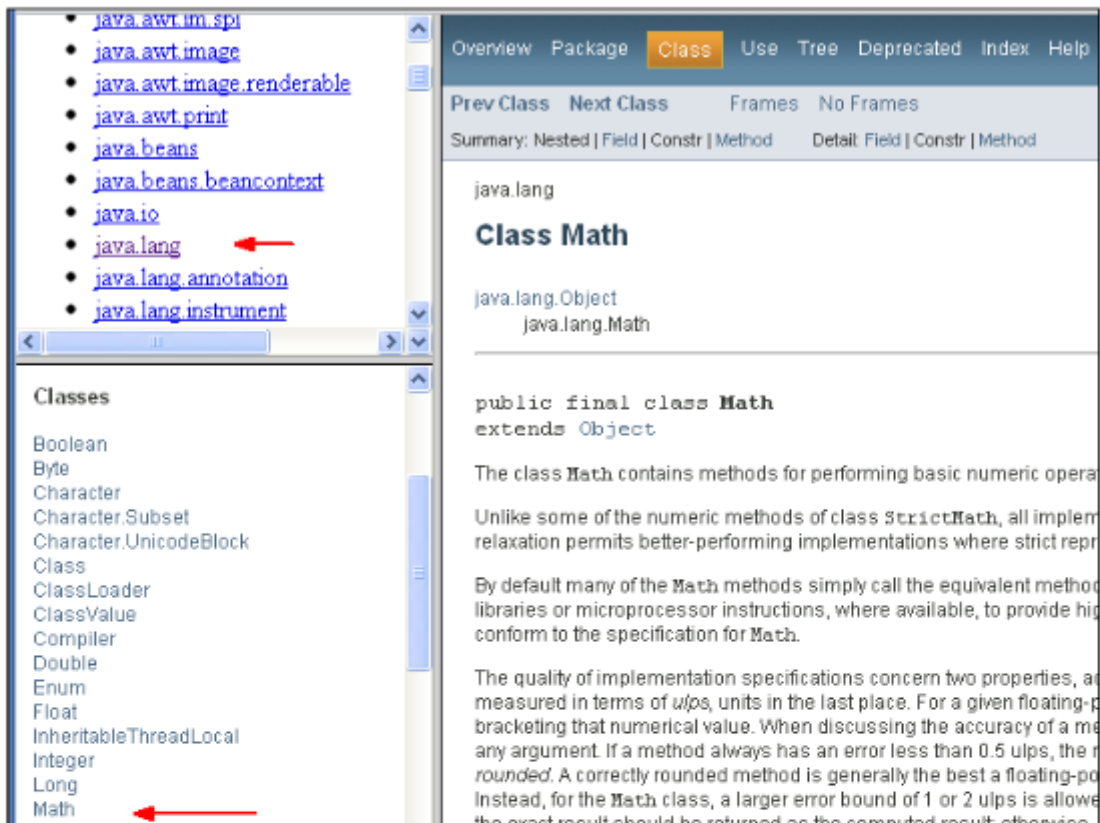
1. To view the Java SE7 API specification (also referred to as “javadocs”), double-click the shortcut on your desktop (entitled “Java JDK7 1.7.0 API Docs”).

Package	Description
java.applet	Provides the classes nece
java.awt	Contains all of the classes
java.awt.color	Provides classes for color
java.awt.datatransfer	Provides interfaces and cla
java.awt.dnd	Drag and Drop is a direct n mechanism to transfer info
java.awt.event	Provides interfaces and cla
java.awt.font	Provides classes and inter
java.awt.geom	Provides the Java 2D class
java.awt.im	Provides classes and inter

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.