Practices for Lesson 7

Practices Overview

In these practices, you create and populate arrays and ArrayLists. You also use the methods of the ArrayList class to manipulate its values. In the last exercise, you create and use a two-dimensional array.

Practice 7-1: Creating a Class with a One-Dimensional Array of Primitive Types

Overview

In this practice, you create an array containing the number of days that an employee at the Duke's Choice company receives, based on the number of years that the employee has worked for Duke's Choice. The following table shows the vacation scale for Duke's Choice:

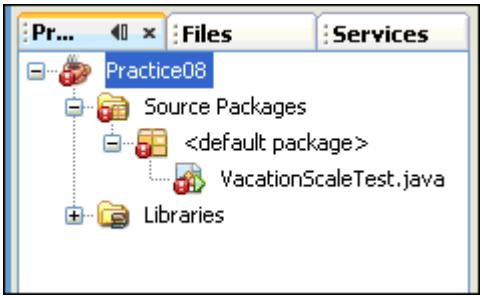
Number of Years of Employment	Number of Days of Vacation
Up to 1 year	10
1, 2, or 3 years	15
4 or 5 years	20
6 or more years	25

Assumptions

The VacationScaleTest.java file appears in the practice folder for this lesson, `Lesson07`.

Tasks

1. Create a new project from existing source called Practice08. Set the **Source Package Folder** to point to `Lesson07`. Remember to also change the Source/Binary Format property. If you need further details, refer to Practice 1-2, Steps 3 and 4.



2. Create a new Java class called VacationScale. Declare but do not initialize two public fields to this class as follows:
   - `int` array called `vacationDays`
   - `int` called `yearsOfService`

   Hint: Use the square brackets ([ ]) next to the data type to indicate that this field is an array.

3. Create a method in the VacationScale class called `setVacationScale`. The table below provides high level steps for this task. More detailed steps can be found below the table.

Step	Window/Page Description	Choices or Values
a.	Initialize the `vacationDays` array	Set size of the array to 7
b.	Populate each element of the `vacationDays` array to align a number of years of service with the correct number of vacation days.. (See the table above)	The value = number of vacation days The element index = number of years of service

   a. Use the `new` keyword to initialize the `vacationDays` array. Supply the size of the array within the square brackets as shown below.

      ```
      vacationDays = new int[7];
      ```

   b. Assign each array element, beginning with `vacationDays[0]` with the appropriate number of days of vacation from the table shown above in the overview section. For example, an employee with 0 years of service is entitled to 10 vacation days. Therefore, `vacationDays[0] = 10`. An employee with 1 year of service is entitled to 15 days of vacation. Therefore `vacationDays[1] = 15`.

   Solution:

```
public void setVacationScale(){
     vacationDays = new int[7];
     vacationDays[0] = 10;
     vacationDays[1] = 15;
     vacationDays[2] = 15;
     vacationDays[3] = 15;
     vacationDays[4] = 20;
     // … and so on through element 6
}
```

4. Create a public method called `displayVacationDays` that displays the number of vacation days due to an employee with the years of service indicated in the `yearsOfService` field. Use an `if/else` construct to check for an invalid `yearsOfService` (a negative number) and display an error message in this case.

   Hint: You can use a variable within the square brackets to represent the array index number. For example:

   ```
   vacationDays[yearsOfService]
   ```

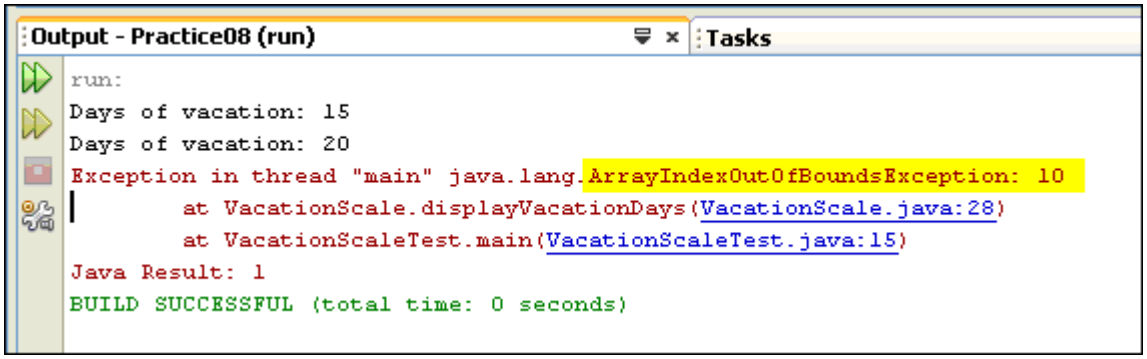
Example:

```
public void displayVacationDays(){
     if(yearsOfService >= 0){
          System.out.println("Days of Vacation: " +
             vacationDays[yearsOfService]);
     }else {
          System.out.println("Invalid years of service");
     }
}
```

5. Save and compile your program. Run the `VacationScaleTest` class to test your program.

   Note: The program, as currently written, throws an exception (an error). You fix this problem in the next few steps.

6. The exception thrown by the Java Virtual Machine (JVM) is an `ArrayIndexOutOfBounds` exception. Your Output window should look similar to the screenshot below:



   This exception is thrown when an attempt has been made to access a non-existent index of an array. Notice that the index number that caused the exception is shown in the error message: index #10. Remember that this array has 7 elements, indexed by numbers 0 through 6. Why did the program try to access index 10?

   Answer:

   If you look at the `displayVacationDays` method, you see that the `yearsOfService` field is used as the array index (as an argument to the `System.out.println` method). It is, of course, conceivable that an employee would have more than 6 (the highest index number of the array) years of service. The `displayVacationDays` method needs to be modified to account for >6 years of service.

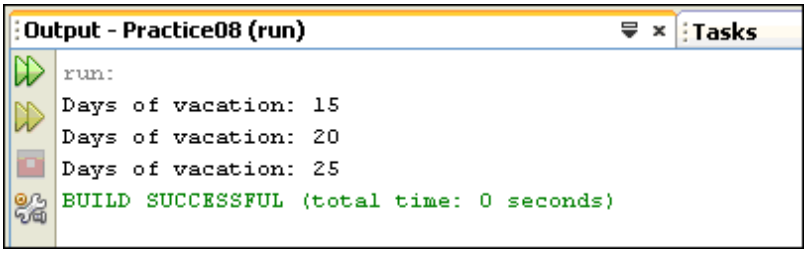
7. Change the `if/else` construct to also check for a `yearsOfService` value that is >=6. All years of service greater than or equal to 6 receive the same number of vacation days.

   Hint: For any `yearsOfService` value between 0 and 5 (inclusive), you can display the value of the array whose index corresponds to that value. For a `yearsOfService` of 6 and above, use the value referenced by the last array index.

Solution:

```
if (yearsOfService >= 0 && yearsOfService < 6) {
     System.out.println("Days of vacation: " +
        vacationDays[yearsOfService]);
} else if (yearsOfService >= 6) {
     System.out.println("Days of vacation: " +
        vacationDays[6]);
} else {
     System.out.println("Invalid years of service");
}
```

8. Save and compile the program and then test it again by running the VacationScaleTest class. You should now see all three of the test values for `yearsOfService` displayed in the output window.



Practice 7-2: Creating and Working With an ArrayList

Overview

In this practice, you create the NamesList class and the NamesListTest class in order to experiment with populating and manipulating ArrayLists. There are two sections in this practice. In the first section, you create the two classes and then add a method to the NamesList class to populate the list and display its contents. In the second section, you add a method to manipulate the values in the list.

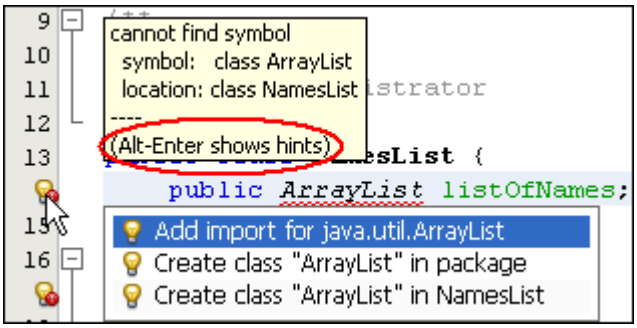
Assumptions

None

Creating and Populating an ArrayList

1. Create a new Java *main* class called NamesListTest. **Reminder:** Right-click the project name in the Projects window and select New > Java Main Class. Leave the `main` method empty for the time being. You add code later.
2. Create a new Java class called NamesList.
3. In the `NamesList` class, declare a public `ArrayList` field called `listOfNames`. Do not instantiate the `listOfNames` field.

   Note: When you type the word `ArrayList`, the editor indicates a warning in the margin of this line. It does not recognize the `ArrayList` class. You must import this class to make it visible to the compiler.
4. Put your cursor over the warning icon in the margin to see the warning description. Click Alt-Enter to view and select from a list of hints to solve this problem. Select **Add import for java.util.ArrayList** as shown here:

   The import statement is placed at the top of the NamesList class (above the class declaration).

```
import java.util.ArrayList;

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author Administrator
 */
public class NamesList {
    public ArrayList listOfNames;
```

5. Add a new method to the NamesList class called setList. Code the method as described in the table below. More detailed steps can be found below the table.

Step	Code Description	Choices or Values
a.	Instantiate the `listOfNames` object	Use the `new` keyword. Do not specify size.
b.	Add a name (first and last name) to the `listOfNames` object	Use the `add` method Example: `listOfNames.add("Joe Smith");`
c.	Repeat step b three times to add a total of four names to the `listOfNames` object.	
d.	Print the list of names with a suitable label.	You can just print the object, itself. (`listOfNames`)
e.	Print the size of the `listOfNames` ArrayList	Use the `size` method of the `listOfNames` object

a. Use the `new` keyword to instantiate `listOfNames`. Example:
```
listOfNames = new ArrayList();
```

b. Invoke the `add` method of the listOfNames object. Pass a `String` value containing first_name and last_name, separated by a space. (See example in table above)

c. Repeat step b three more times, using a different name in each method invocation.

d. Use `System.out.println` to print out all of the names within the `listOfNames` object. Use a suitable label and concatenate the `listOfNames` object to it.
```
System.out.println("Names list: " + listOfNames);
```

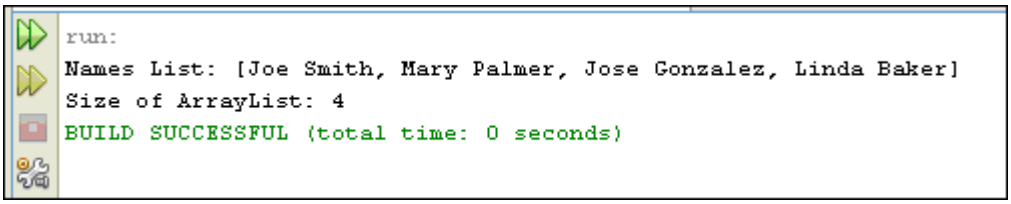
e. Use `System.out.println` to print out the size (number of elements) of the `listOfNames` object. Use the `size` method of the `listOfNames` object and concatenate a suitable label.
```
System.out.println("Size of ArrayList: " + listOfNames.size());
```

Solution:

```
public void setList(){
     listOfNames = new ArrayList();
     listOfNames.add("Joe Smith");
     listOfNames.add("Mary Palmer");
     listOfNames.add("Jose Gonzalez");
     listOfNames.add("Linda Baker");

     System.out.println("Names list: " + listOfNames);
     System.out.println("Size of ArrayList: " +
          listOfNames.size());
}
```

6. Click Save to compile.
7. Open the NamesListTest class in the editor. In the `main` method:
8. Instantiate a NamesList object called "names" using the `new` keyword.
   a. Invoke the `setList` method of the `names` object.
9. Save and compile your program. Run the NamesListTest class to test the program.



Manipulating the ArrayList

10. Add another new method to the NamesList class called `manipulateList`. Code the method as described in the table below. More detailed steps can be found below the table.

Step	Code Description	Choices or Values
a.	Remove one of the names from the list	Use the `remove(Object obj)` method of the `ArrayList` object. Hint: a `String` literal is an object.
b.	Print the contents of the `listOfNames` object, using a suitable label	
c.	Print the size of the `listOfNames` object	Use the `size` method of the `ArrayList`
d.	Add the name you just removed back into the list in a different location	Use the `add(int index, Object obj)` method of the `ArrayList` object. Hint: Index numbers are zero-based.
e.	Print the contents of the `listOfNames` object	
f.	Print the size of the `listOfNames` object	

a. Remove one of the names in the `ArrayList` using the `remove` method and passing the full name, enclosed in double quotes.
   - Note: This method is defined as taking an `Object` as an argument. A `String` literal, such as the quote-enclosed full name, is an object.

   ```
   listOfNames.remove("Joe Smith");
   ```

b. Use `System.out.println` to print the `listOfNames` object. Use an appropriate label.

c. Use `System.out.println` to print the current size of the `ArrayList`. Use an appropriate label.

d. Use the `add` method of the `ArrayList` to add the name you just removed back into the `ArrayList`, but at a different location in the list than previously.

   Note: The `add` method is "overloaded". That is, it has two different method signatures. One of the `add` methods takes an `Object` and appends it to the end of the `ArrayList`. The other method takes an index number and an `Object`. It inserts the `Object` before the referenced index number, pushing all remaining list elements over one index number. Use the latter `add` method. An example is shown below:

   ```
   listOfNames.add(1, "Joe Smith");
   ```

e. Use a suitable label when printing the newly modified contents of the `listOfNames` object.

f. Use a suitable label when printing the new size of the `listOfNames` object.

Example Solution:

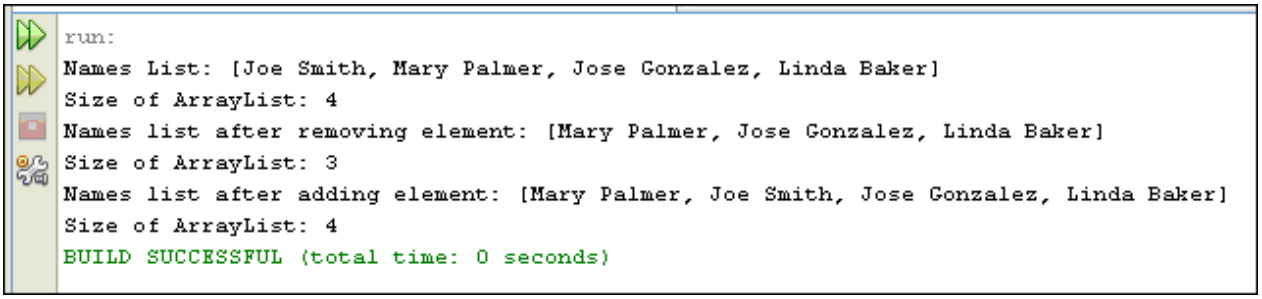
```
listOfNames.remove ("Joe Smith");
System.out.println("Names list after removing element: "
    + listOfNames);
System.out.println("Size of ArrayList: " +
    listOfNames.size());
listOfNames.add(1, "Joe Smith");
System.out.println("Names list after adding element: "
     + listOfNames);
System.out.println("Size of ArrayList: " +
    listOfNames.size());
```

11. In the `main` method of the NamesListTest class, invoke the `manipulateList` method of the `names` object.

    Note: You may need to click Save so that the compiler can resolve the reference to `manipulateList`.

12. Save and compile the program.

13. Run the NamesListTest class to test the program. The output should look similar to the screenshot below, depending upon the name you removed and added, and the index number you used in the `add` method.



Note: In the example shown above, Joe Smith was previously located at index position 0 and Mary Palmer was at index position 1.

After removing Joe Smith, Mary Palmer moved to index position 0 and Jose Gonzalez was at index position 1.

Joe Smith was then added at index position 1, pushing Jose Gonzalez to index position 2.

Practice 7-3: Using Runtime Arguments and Parsing the `args` Array

Overview

In this practice, you write a guessing game that accepts an argument and displays an associated message. You create a class that accepts a runtime argument between 1 and 5, inclusive. You also randomly generate a number between 1 and 5 in the class and compare the value of the argument with the randomly generated number.

Assumptions

None

Tasks

1. Create a new Java Main Class called GuessingGame.
2. In the `main` method, declare two `int` variables as shown below:

```
int randomNum = 0;
int guess;
```

3. Add code to the `main` method to accept a single argument of any number in the range of 1 to 5, inclusive, or the word "help". The high level steps are described in the pseudo code below, followed by helpful hints. If you need additional assistance, follow the steps below the pseudo code and hints. Remember, the solution can also be found in `Lesson07`
4. **Pseudo Code for main Method:**

```
if length of args array == 0 or value of args[0] = "help"
     print a Correct Usage message
else
     randomNum = a generated random number 1 - 5
     guess = integer value of args[0]

     if argument < 1 or > 5
          print an error message (invalid argument)
     else
          if argument == randomNum
               print congratulations message
          else
               print a "Sorry; try again" message
```

Hints

- Use the `compareTo` method of the `String` class (elements of the `args` array are always Strings) to match the `args[0]` to "help".
- To generate the random number 1 – 5, use the following code snippet:
  `randomNum = ((int)(Math.random()*5)+1);`
- Convert the runtime argument to an `int` before assigning it to the `guess` variable. Use the `Integer.parseInt` method to do the conversion.

Detailed Steps

a. If the first argument in the `args` array equals "help" or if the `args` array is empty, display the usage of the program. For example:

"Usage: java GuessingGame [argument]"

"Enter a number from 1 to 5 as your guess"

b. If a 1, 2, 3, 4, or 5 is entered:

- Generate a random number (as shown in Hint above)
- Convert the `arg[0]` to an `int` and assign it to the `guess` variable.

```
guess = Integer.parseInt(args[0]);
```

- Compare the `guess` to `randomNum` using a nested `if/else` construct.
- If they match, display a "Congratulations" message.
- Else, tell them what the random number was and ask them to "Try again."

Solution:

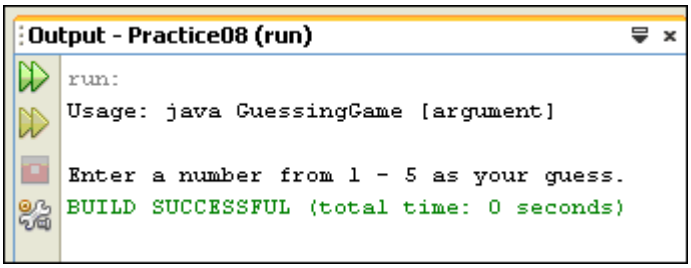
```
public static void main(String[] args){
     int randomNum = 0;
     int guess;
     if(args.length == 0 || args[0].compareTo("help") == 0){
          System.out.println
               ("Usage: java GuessingGame [argument]");
          System.out.println();
          System.out.println
               ("Enter a number from 1 – 5 as your guess.");
     }else {
          randomNum = ((int)(Math.random()*5) +1);
          guess = Integer.parseInt(args[0]);

          if(guess < 1 || guess > 5){
               System.out.println
                ("Invalid argument: Enter a number from 1 - 5");
          }else {
               if(guess == randomNum){
                  System.out.println
                     ("Great guess! You got it right!");
               }else {
                     System.out.println
                       ("Sorry. The number was " + randomNum +
                          ". Try again.");
               }//end of innermost if/else
          } // end of first nested if/else
     }// end of outer if/else
}// end of main method
```

5. Save and compile the program.

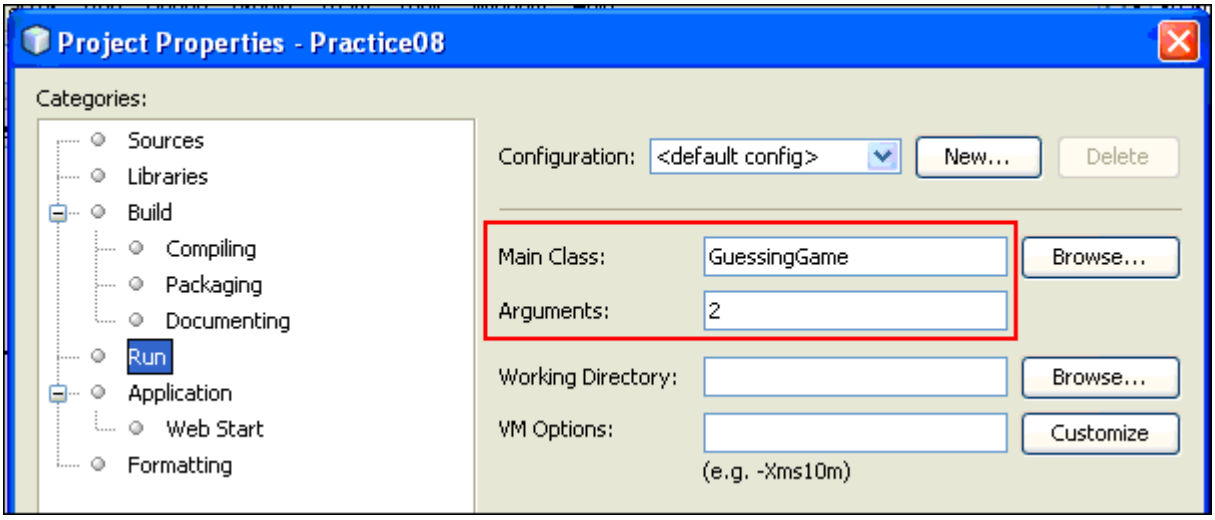
6. Test it by running the GuessingGame class.

   Note: Since no runtime parameter was passed to the `args` array, you should get the Usage message as shown here.



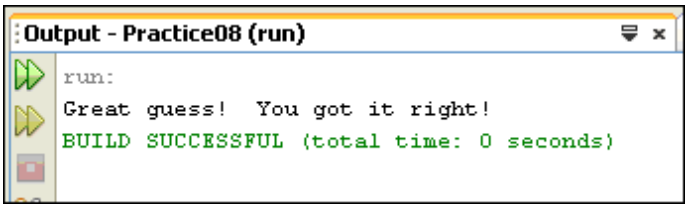
   Note: When using an IDE, you don't have access to the command line to provide runtime parameters. Therefore, you enter your "guess" (runtime parameter) as a runtime property of the project and then run the entire project, rather than just running an individual file.

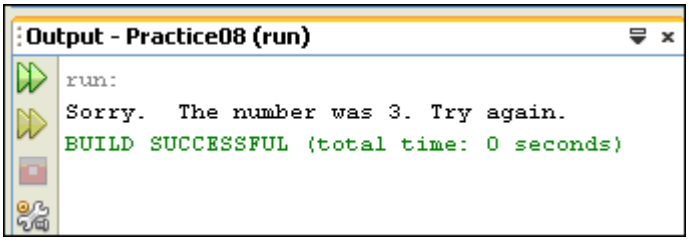
7. Right-click the project name in the Projects window and select Properties from the menu.
8. In the Project Properties window, select the **Run** category. Change the **Main Class** to GuessingGame and enter a number from 1 to 5 in the **Arguments** field. Click **OK**.



9. Now run the project by clicking the Run button on the main toolbar.

10. You should receive either the "Great guess…" message or the "Sorry. …Try again." message. Continue to click the run button to see the different random numbers generated and how the program responds by comparing it with your guess.





11. Close the Practice08 project in NetBeans.

    This completes the Lesson 7 practices. In the practices for the next lesson, you have an opportunity to work with two-dimensional arrays.