# 10 JAVA LAB ACTIVITIES

**Activity 1:**

Write a program to implement overload methods to display dates. The date-displaying methods might be used by many different applications in an organization, such as those that schedule jobs, appointments, and employee reviews. The methods take one, two, or three integer arguments. If there is one argument, it is the month, and the date becomes the first day of the given month in the year 2014. If there are two arguments, they are the month and the day in the year 2014. Three arguments represent the month, day, and year.

Instead of creating your own class to store dates, you can use the built-in Java class GregorianCalendar to handle dates. This exercise illustrates how some of the built-in GregorianCalendar class was constructed by Java's creators.

1. Open a new file in your text editor.
2. Begin the following DemoOverload class with three integer variables to test the method and three calls to a displayDate() method:
3. Create the following displayDate() method that requires one parameter to represent the month and uses default values for the day and year:
4. Create the following displayDate() method that requires two parameters to represent the month and day and uses a default value for the year:
5. Create the following displayDate() method that requires three parameters used as the month, day, and year:
6. Type the closing curly brace for the DemoOverload class
7. Save the file as DemoOverload.java.
8. Compile the program, correct any errors, recompile if necessary, and then execute the program.
9. Notice that whether you call the displayDate() method using one, two, or three arguments, the date is displayed correctly because you have successfully overloaded the displayDate() method.

**Activity 2 :**

Write a program for Sacks Fifth Avenue, a nonprofit thrift shop. The program should determine which volunteer to assign, to price a donated item. To begin, you should prompt the user to answer a question about whether a donation is clothing or some other type, and then the program should display the name of the volunteer who handles such donations. Clothing donations are handled by Regina, and other donations are handled by Marco.

1. Open a new text file, and then enter the first lines of code to create a class named AssignVolunteer. You import the Scanner class so that you can use keyboard input. The class contains a main() method that performs all the work of the class.

2. On new lines, declare the variables and constants this application uses. The user will be prompted to enter one of the values stored in the two constants. That value will then be assigned to the integer donationType and compared to the CLOTHING_CODE constant. Then, based on the results of that comparison, the program will assign the value of one of the PRICER constants to the String variable volunteer.

3. Define the input device, and then add the code that prompts the user to enter a 1 or 2 for the donation type. Accept the response, and assign it to donationType:

Use an if...else statement to choose the name of the volunteer to be assigned to the volunteer String, as follows.

5. Display the chosen code and corresponding volunteer's name

6. Type the two closing curly braces to end the main() method and the AssignVolunteer class.

7. Save the program as AssignVolunteer.java, and then compile and run the program. Confirm that the program selects the correct volunteer when you

choose 1 for a clothing donation or 2 for any other donation type.

**Activity 3 :**

**Using Part of an Array**

Write a program that allows a student to enter up to 10 quiz scores and then computes and displays the average. To allow for 10 quiz scores, you create an array that can hold 10 values, but because the student might enter fewer than 10 values, you might use only part of the Array.

**Activity 4:**

**Using a Bubble Sort**

In this activity, write a program in which you enter values that you sort using the bubble sort algorithm. Display the values during each iteration of the outer sorting loop so that you can track the values as they are repositioned in the array.

1. Open a new file in your text editor, and create the shell for a BubbleSortDemo program as follows:
2. Make some declarations between the curly braces of the main() method. Declare an array of five integers and a variable to control the number of comparisons to make during the sort. Declare a Scanner object, two integers to use as subscripts for handling the array, and a temporary integer value to use during the sort.
3. Write a for loop that prompts the user for a value for each array element and accepts them.
4. Next, call a method that accepts the array and the number of sort iterations performed so far, which is 0. The purpose of the method is to display the current status of the array as it is being sorted.
5. Add the nested loops that perform the sort. The outer loop controls the

number of passes through the list, and the inner loop controls the comparisons on each pass through the list. When any two adjacent elements are out of order, they are swapped. At the end of the nested loop, the current list is output and the number of comparisons to be made on the next pass is reduced by one.

6. After the closing brace for the main() method, but before the closing brace for the class, insert the display() method. It accepts the array and the current outer loop index, and it displays the array contents.

**Activity 5:**

**Inheritance**

i.Create a Superclass and an Application to Use It

1. Open a new file in your text editor, and enter the following first few lines for a simple Party class. The class, hosts one integer data field—the number of guests expected at the party.
2. Add the following methods that get and set the number of guests.
3. Add a method that displays a party invitation
4. Add the closing curly brace for the class, and then save the file as Party.java. Compile the class; if necessary, correct any errors and compile again.

ii. Write an Application That Uses the Party Class

Now that you have created a class, you can use it in an application. A very simple application creates a Party object, prompts the user for the number of guests at the party, sets the data field, and displays the results.

1. Open a new file in your text editor.
2. Write a UseParty application that has one method—a main() method. Enter the beginning of the class, including the start of the main() method, which declares a variable for guests, a Party object, and a Scanner object to use for input.

3. Continue the main() method by prompting the user for a number of guests and accepting the value from the keyboard. Set the number of guests in the Party object, and then display the value

4. Add a statement to display the party invitation, and then add the closing curly braces for the main() method and for the class.

5.Save the file as UseParty.java, then compile and run the application.

**Activity 6:**

**Using an Interface**

In this section, create an Insured interface for use with classes that represent objects that can be insured. For example, you might use this interface with classes such as Jewelry or House. Extend Vehicle to create an InsuredCar class that implements the Insured interface, and then you write a short program that instantiates an InsuredCar object.

1. Open a new file in your text editor, and type the following Insured interface. A concrete class that implements Insured will be required to contain setCoverage() and getCoverage() methods.

2. Save the file as Insured.java and compile it.

3. Open a new file in your text editor, and start the InsuredCar class that extends Vehicle and implements Insured.

4. Add a variable to hold the amount covered by the insurance.

5. Add a constructor that calls the Vehicle superclass constructor, passing arguments for the InsuredCar's power source and number of wheels.

6. Implement the setPrice() method required by the Vehicle class. The method accepts the car's price from the user and enforces a maximum value of $60,000.

7. Implement the setCoverage() and getCoverage() methods required by the Insured class. The setCoverage() method sets the coverage value for an insured car to 90 percent of the car's price.

8. Create a toString() method, followed by a closing brace for the class.

9. Save the file as InsuredCar.java and compile it.
10. Create a demonstration program that instantiates an InsuredCar object and displays its values as follows.
11. Save the file as InsuredCarDemo.java. Compile and execute it. It will display a prompt to enter the car's price.

**Activity 7:**

**Throwing and Catching an Exception**

Create an application in which the user enters two values to be divided. The application catches an exception if either of the entered values is not an integer.

1. Open a new file in your text editor, and type the first few lines of an interactive application named ExceptionDemo.
2. Declare three integers—two to be input by the user and a third to hold the result after dividing the first two. The numerator and denominator variables must be assigned starting values because their values will be entered within a try block. The compiler understands that a try block might not complete; that is, it might throw an exception before it is through. Also declare an input String to hold the return value of the JOptionPane showInputDialog() method.
3. Add a try block that prompts the user for two values, converts each entered String to an integer, and divides the values, producing result.
4. Add a catch block that catches an ArithmeticException object if division by 0 is attempted. If this block executes, display an error message, and force result to 0.
5. Whether or not the try block succeeds, display the result (which might have been set to 0). Include closing curly braces for the main() method and for the class.
6. Save the file as ExceptionDemo.java, and then compile and execute the application. Enter two nonzero integer values. For example, the first execution in shows the output when the user enters 12 and 3 as the two input values. The application completes successfully. Click OK to end the application,

and execute the ExceptionDemo application again. This time, enter 0 for the second value; Click OK to end the application.

Activity 8:

**Introduction to Swing Components**

Create a Swing application that displays a JFrame that holds a JLabel, JTextField, and JButton.

1. Open a new file in your text editor, then type the following first few lines of an application. The import statements make the Swing and AWT components available, and the class header indicates that the class is a JFrame. The class contains several components: a label, field, and button.
2. In the JFrameWithComponents constructor, set the JFrame title to "Frame with Components" and the default close operation to exit the program when the JFrame is closed. Set the layout manager. Add the label, field, and button to the JFrame.
3. Add a closing curly brace for the class, and then save the file as JFrameWithComponents.java.
4. Compile the class and correct any errors.
5. Next, write an application that creates a new JFrameWithComponents named aFrame, sizes it using the setSize() method, and then sets its visible property to true.
6. Save the file as CreateJFrameWithComponents.java. Compile and then execute the application.
7. Click the JButton. It acts like a button should—that is, it appears to be pressed when you click it, but nothing happens because you have not yet written instructions for the button clicks to execute.
8. Close the application.

Activity 9:

**Graphics**

Create a Graphics object named pen and use the object to draw a String on the screen. The text of the String will appear to move each time a JButton is clicked.

1. Open a new text file in your text editor, and type import statements
2. Start typing the following class that extends JFrame and uses the mouse. The class defines a String, a JButton, a Font, and four integers: two to hold x- and ycoordinates, one to act as a constant size to measure the gap between lines displayed on the screen, and one to hold the size of the JFrame.
3. Type the following constructor; it changes the background color and sets the layout of the Container, adds the JButton, prepares the JFrame to listen for JButton events, sets the close operation, and sets the size of the frame.
4. Within the actionPerformed() method, you can create a Graphics object and use it to draw the String on the screen. Each time a user clicks the JButton, the x- and y-coordinates both increase, so a copy of the movie quote appears slightly below and to the right of the previous one. Type the following actionPerformed() method to accomplish this processing.

5. Add a main() method to instantiate a JDemoCreateGraphicsObject object and give it visibility. Add a closing curly brace for the class.
6. Save the file as JDemoCreateGraphicsObject.java, and then compile and run the program. Click the Move It button several times to see the String message appear and move on the screen.
7. When you finish clicking the button, close the JFrame to end the application

Activity 10:

**i.Creating an HTML Document to Host an Applet**

Create a simple HTML document that you will use to display the applet you will create in the next section. You will name the applet JGreet, and it will occupy a screen area of 450 by 100 pixels.

1. Open a new file in your text editor. Type the opening HTML tag
2. On the next line, type the opening object tag that contains the applet's name and dimensions
3. On the next line, type the applet's closing tag.
4. On the next line, type the closing HTML tag.

5. Save the file as TestJGreet.html. Just as when you create a Java application, be certain that you save the file as text only and use an .html extension. The .html file extension is required and makes the file easy to identify as an HTML file.

### ii.Creating and Running a JApplet

Next, you create the JGreet applet for which you prepared the HTML document.

1. Open a new text file in your text editor. Enter the following import statements you need for the JApplet. You need the javax.swing package because it defines JApplet, and you need the java.awt package because it defines Container.
2. Next, enter the JGreet JApplet. It contains a Container that holds a JLabel. The init() method adds the JLabel to the Container.
3. Save the file as JGreet.java. Then compile the class. If necessary, correct any errors and compile again.
4. At the command line, type appletviewer TestJGreet.html, and then press Enter. The applet appears on your screen.
5. Use the mouse pointer to drag any corner of the Applet Viewer window to resize it. Notice that if you increase or decrease the window's height, the window is redrawn on the screen and the JLabel is automatically repositioned to remain centered within the window. If you make the window narrower by dragging its right border to the left, the JLabel eventually becomes partially obscured when the window becomes too narrow for the display.

*#If your operating system does not allow you to make the window narrow enough to obscure part of the greeting, make the string in the label longer—for example, "Greetings to you and all your family!" Recompile the applet and then use the appletviewer command to display the HTML document again. The string displayed will be long enough for you to observe the effects when you narrow the width of the window.*

6. Close the Applet Viewer by clicking the Close button in the upper-right corner of the window.

### iii.Running a JApplet in Your Web Browser

1. Open any Web browser, such as Internet Explorer. You do not have to connect to the Internet; you will use the browser locally. (If you do not have a Web browser installed on your computer, skip to the end of Step 3.)
2. Click File on the menu bar, click Open, type the complete path for the

HTML document that you created to access JGreet.class (for example, C:\Java\Chapter17\TestJGreet.html), and then press Enter. Instead of typing the filename, you can click the Browse button, browse to the file location, and then click OK. You might have to agree to several security messages before you can view the applet. The applet should appear in the browser on your screen. If you receive an error message, verify that the path and spelling of the HTML file are correct.

3. Close the browser.

*#Some applets might not work correctly with your browser. Java was designed with a number of security features so that when an applet is displayed on the Internet, the applet cannot perform malicious tasks,such as deleting a file from your hard drive. If an applet does nothing to compromise security, testing it using the Web browser or the appletviewer command achieves the same results. For now, you can get your applets to perform better by using the Applet Viewer window because the output does not depend on the browser type or version.*