# Arrays Lab Exercises

## Due Date : Nov 19

Create the following programs in a single package with your First and Last Name. Submit the package to drop off folder.

**Topics**  **Lab Exercises**

One-Dimensional Arrays [15] Tracking Sales

Grading Quizzes

Reversing an Array

Searching and Sorting in an Integer List

Adding To and Removing From an Integer List

Two-Dimensional Arrays [5] Magic Squares

Arrays of Objects [5] A Shopping Cart

ArrayLists [5] A Shopping Cart Using the ArrayList Class

Arrays and Graphics [10] Drawing Circles (again)

A Polygon Person

Buttons [10] Adding Buttons to StyleGUI.java

An Array of Radio Buttons

**Tracking Sales**

File *Sales.java* contains a Java program that prompts for and reads in the sales for each of 5 salespeople in a company. It then prints out the id and amount of sales for each salesperson and the total sales. Study the code, then compile and run the program to see how it works. Now modify the program as follows:

1. Compute and print the average sale. (You can compute this directly from the total; no loop is necessary.)

2. Find and print the maximum sale. Print both the id of the salesperson with the max sale and the amount of the sale, e.g., "Salesperson 3 had the highest sale with $4500." Note that you don't need another loop for this; you can do it in the same loop where the values are read and the sum is computed.

3. Do the same for the minimum sale.

4. After the list, sum, average, max and min have been printed, ask the user to enter a value. Then print the id of each salesperson who exceeded that amount, and the amount of their sales. Also print the total number of salespeople whose sales exceeded the value entered.

5. The salespeople are objecting to having an id of 0—no one wants that designation. Modify your program so that the ids run from 1–5 instead of 0–4. **Do not modify the array**—just make the information for salesperson 1 reside in array location 0, and so on.

6. Instead of always reading in 5 sales amounts, at the beginning ask the user for the number of sales people and then create an array that is just the right size. The program can then proceed as before.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Sales.java

//

// Reads in and stores sales for each of 5 salespeople. Displays

// sales entered by salesperson id and total sales for all salespeople.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import cs1.Keyboard;

public class Sales

{

public static void main(String[] args)

{

final int SALESPEOPLE = 5;

int[] sales = new int[SALESPEOPLE];

int sum;

for (int i=0; i<sales.length; i++)

{

System.out.print("Enter sales for salesperson " + i + ": ");

sales[i] = Keyboard.readInt();

}

System.out.println("\nSalesperson Sales");

System.out.println("--------------------");

sum = 0;

for (int i=0; i<sales.length; i++)

{

System.out.println(" " + i + " " + sales[i]);

sum += sales[i];

}

System.out.println("\nTotal sales: " + sum);

}

}

**Grading Quizzes**

Write a program that grades multiple-choice quizzes as follows:

* Ask the user how many questions are in the quiz.
* Ask the user to enter the key (that is, the correct answers). There should be one answer for each question in the quiz, and each answer should be a lower-case letter. They can be entered on a single line, e.g., abbacdeacb might be the key for a 10-question quiz. You will need to store the key in an array.
* Ask the user to enter the answers for the quiz to be graded. As for the key, these can be entered on a single line. Again there needs to be one for each question. Note that these answers do not need to be stored; each answer can simply be compared to the key as it is entered.
* When the user has entered all of the answers to be graded, print the number correct and the percent correct.

When this works, add a loop so that the user can grade any number of quizzes with a single key. After the results have been printed for each quiz, ask "Grade another quiz? (y/n)."

**Reversing an Array**

Write a program that prompts the user for an integer, then asks the user to enter that many values. Store these values in an array and print the array. Then reverse the array elements so that the first element becomes the last element, the second element becomes the second to last element, and so on, with the old last element now first. Do not just reverse the order in which they are printed; actually change the way they are stored in the array. Do not create a second array; just rearrange the elements within the array you have. (Hint: Swap elements that need to change places.) When the elements have been reversed, print the array again.

**Searching and Sorting In An Integer List**

File *IntegerList.java* contains a Java class representing a list of integers. The following public methods are provided:

* IntegerList(int size)—creates a new list of *size* elements. Elements are initialized to 0.
* void randomize()—fills the list with random integers between 1 and 100, inclusive.
* void print()—prints the array elements and indices
* int search(int target)—looks for value *target* in the list using a sequential search algorithm. Returns the index where it first appears if it is found, -1 otherwise.
* void selectionSort()—sorts the lists into ascending order using the selection sort algorithm.

File *IntegerListTest.java* contains a Java program that provides menu-driven testing for the IntegerList class. Copy both files to your directory, and compile and run IntegerListTest to see how it works. For example, create a list, print it, and search for an element in the list. Does it return the correct index? Now look for an element that is not in the list. Now sort the list and print it to verify that it is in sorted order.

Modify the code in these files as follows:

1. Add a method *void sortDecreasing()* to the IntegerList class that sorts the list into decreasing (instead of increasing) order. Use the selection sort algorithm, but modify it to sort the other way. Be sure you change the variable names so they make sense!

Add an option to the menu in IntegerListTest to test your new method.

2. Add a method *void replaceFirst(int oldVal, int newVal)* to the IntegerList class that replaces the first occurrence of oldVal in the list with newVal. If oldVal does not appear in the list, it should do nothing (but it's not an error). If oldVal appears multiple times, only the first occurrence should be replaced. Note that you already have a method to find oldVal in the list; use it!

Add an option to the menu in IntegerListTest to test your new method.

3. Add a method *void replaceAll(int oldVal, int newVal)* to the IntegerList class that replaces all occurrences of oldVal in the list with newVal. If oldVal does not appear in the list, it should do nothing (but it's not an error). Does it still make sense to use the search method like you did for *replaceFirst*, or should you do your own searching here? Think about this.

Add an option to the menu in IntegerListTest to test your new method.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// IntegerList.java

//

// Define an IntegerList class with methods to create, fill,

// sort, and search in a list of integers.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import cs1.Keyboard;

public class IntegerList{

int[] list; //values in the list

//-------------------------------------------------------

//create a list of the given size

//-------------------------------------------------------

public IntegerList(int size)

{

list = new int[size];

}

//-------------------------------------------------------

//fill array with integers between 1 and 100, inclusive

//-------------------------------------------------------

public void randomize()

{

for (int i=0; i<list.length; i++)

list[i] = (int)(Math.random() \* 100) + 1;

}

//-------------------------------------------------------

//print array elements with indices

//-------------------------------------------------------

public void print()

{

for (int i=0; i<list.length; i++)

System.out.println(i + ":\t" + list[i]);

}

//-------------------------------------------------------

//return the index of the first occurrence of target in the list.

//return -1 if target does not appear in the list

//-------------------------------------------------------

public int search(int target)

{

int location = -1;

for (int i=0; i<list.length && location == -1; i++)

if (list[i] == target)

location = i;

return location;

}

//-------------------------------------------------------

//sort the list into ascending order using the selection sort algorithm

//-------------------------------------------------------

public void selectionSort()

{

int minIndex;

for (int i=0; i < list.length-1; i++)

{

//find smallest element in list starting at location i

minIndex = i;

for (int j = i+1; j < list.length; j++)

if (list[j] < list[minIndex])

minIndex = j;

//swap list[i] with smallest element

int temp = list[i];

list[i] = list[minIndex];

list[minIndex] = temp;

}

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// IntegerListTest.java

//

// Provide a menu-driven tester for the IntegerList class.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import cs1.Keyboard;

public class IntegerListTest{

static IntegerList list = new IntegerList(10);

//-------------------------------------------------------

// Create a list, then repeatedly print the menu and do what the

// user asks until they quit

//-------------------------------------------------------

public static void main(String[] args)

{

printMenu();

int choice = Keyboard.readInt();

while (choice != 0)

{

dispatch(choice);

printMenu();

choice = Keyboard.readInt();

}

}

//-------------------------------------------------------

// Do what the menu item calls for

//-------------------------------------------------------

public static void dispatch(int choice)

{

int loc;

switch(choice)

{

case 0:

System.out.println("Bye!");

break;

case 1:

System.out.println("How big should the list be?");

int size = Keyboard.readInt();

list = new IntegerList(size);

list.randomize();

break;

case 2:

list.selectionSort();

break;

case 3:

System.out.print("Enter the value to look for: ");

loc = list.search(Keyboard.readInt());

if (loc != -1)

System.out.println("Found at location " + loc);

else

System.out.println("Not in list");

break;

case 4:

list.print();

break;

default:

System.out.println("Sorry, invalid choice");

}

}

//-------------------------------------------------------

// Print the user's choices

//-------------------------------------------------------

public static void printMenu()

{

System.out.println("\n Menu ");

System.out.println(" ====");

System.out.println("0: Quit");

System.out.println("1: Create a new list (\*\* do this first!! \*\*)");

System.out.println("2: Sort the list using selection sort");

System.out.println("3: Find an element in the list using sequential search");

System.out.println("4: Print the list");

System.out.print("\nEnter your choice: ");

}

}

**Adding To and Removing From an Integer List**

File *IntegerList.java* (See the previous exercises.) contains a Java class representing a list of integers. The following public methods are provided:

* IntegerList(int size)—creates a new list of *size* elements. Elements are initialized to 0.
* void randomize()—fills the list with random integers between 1 and 100, inclusive.
* void print()—prints the array elements and indices
* int search(int target)—looks for value *target* in the list using a sequential search algorithm. Returns the index where it first appears if it is found, -1 otherwise.
* void selectionSort()—sorts the lists into ascending order using the selection sort algorithm.

File *IntegerListTest.java* (See the previous exercises.) contains a Java program that provides menu-driven testing for the IntegerList class. Copy both files to your directory, and compile and run IntegerListTest to see how it works. For example, create a list, print it, and use sequential search to look for an element in the list. Does it return the correct index? Now look for an element that is not in the list. Now sort the list and print it to verify that it is in sorted order.

It is often necessary to add items to or remove items from a list. When the list is stored in an array, one way to do this is to create a new array of the appropriate size each time the number of elements changes, and copy the values over from the old array. However, this is rather inefficient. A more common strategy is to choose an initial size for the array and add elements until it is full, then double its size and continue adding elements until it is full, and so on. (It is also possible to decrease the size of the array if it falls under, say, half full, but we won't do that in this exercise.) The CDCollection class in Listing 6.8 of the text uses this strategy—it keeps track of the current size of the array and the number of elements already stored in it, and method *addCD* calls *increaseSize* if the array is full. Study that example.

1. Add this capability to the IntegerList class. You will need to add an *increaseSize* method plus instance variables to hold the current number of integers in the list and the current size of the array. Since you do not have any way to add elements to the list, you won't need to call *increaseSize* yet..

2. Add a method *void addElement(int newVal)* to the IntegerList class that adds an element to the list. At the beginning of *addElement*, check to see if the array is full. If so, call *increaseSize* before you do anything else.

Add an option to the menu in IntegerListTest to test your new method.

3. Add a method *void removeFirst(int newVal)* to the IntegerList class that removes the first occurrence of a value from the list. If the value does not appear in the list, it should do nothing (but it's not an error). Removing an item should not change the size of the array, but note that the array values do need to remain contiguous, so when you remove a value you will have to shift everything after it down to fill up its space. Also remember to decrement the variable that keeps track of the number of elements.

Add an option to the menu in IntegerListTest to test your new method.

4. Add a method *removeAll(int newVal)* to the IntegerList class that removes all occurrences of a value from the list. If the value does not appear in the list, it should do nothing (but it's not an error).

Add an option to the menu in IntegerListTest to test your new method.

5. Add a method *void addInOrder(int newVal)* to the IntegerList class that assumes that the list is sorted in increasing order and adds the given element in its correct (sorted) position. So if the list contained the values 10 20 30 40 50 and you added 25, the new list would be 10 20 25 30 40 50. Don't just stick the value on the end and then sort—sorting is expensive! Instead, look through the list, figure out where the new value should go and put it there, moving everything after it down to make room for it. As for *addElement*, you may need to increase the size of the array (determine this and do it first if necessary).

Add an option to the menu in IntegerListTest to test your new method.

**Magic Squares**

One interesting application of two-dimensional arrays is *magic squares*. A magic square is a square matrix in which the sum of every row, every column, and both diagonals is the same. Magic squares have been studied for many years, and there are some particularly famous magic squares. In this exercise you will write code to determine whether a square is magic.

File *Square.java* contains the shell for a class that represents a square matrix. It contains headers for a constructor that gives the size of the square and methods to read values into the square, print the square, find the sum of a given row, find the sum of a given column, find the sum of the main (or other) diagonal, and determine whether the square is magic. The read method is given for you; you will need to write the others.

File *SquareTest.java* contains the shell for a program that reads input for squares and tells whether each is a magic square. Following the comments, fill in the remaining code. Run your program on file *magicData* (the program reads from the standard input, so you'll need to redirect the standard input to the file). You should find that the first, second, and third squares in the input are magic, and that the rest (fourth through seventh) are not. Note that the -1 at the bottom tells the test program to stop reading.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Square.java

//

// Define a Square class with methods to create and read in

// info for a square matrix and to compute the sum of a row,

// a col, either diagonal, and whether it is magic.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import cs1.Keyboard;

public class Square

{

int[][] square;

//--------------------------------------

//create new square of given size

//--------------------------------------

public Square(int size)

{

}

//--------------------------------------

//return the sum of the values in the given row

//--------------------------------------

public int sumRow(int row)

{

}

//--------------------------------------

//return the sum of the values in the given column

//--------------------------------------

public int sumCol(int col)

{

}

//--------------------------------------

//return the sum of the values in the main diagonal

//--------------------------------------

public int sumMainDiag()

{

}

//--------------------------------------

//return the sum of the values in the other ("reverse") diagonal

//--------------------------------------

public int sumOtherDiag()

{

}

//--------------------------------------

//return true if the square is magic (all rows, cols, and diags have

//same sum), false otherwise

//--------------------------------------

public boolean magic()

{

}

//--------------------------------------

//read info into the square from the standard input.

//--------------------------------------

public void readSquare()

{

for (int row = 0; row < square.length; row++)

for (int col = 0; col < square.length; col ++)

square[row][col] = Keyboard.readInt();

}

//--------------------------------------

//print the contents of the square, neatly formatted

//--------------------------------------

public void printSquare()

{

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// SquareTest.java

//

// Uses the Square class to read in square data and tell if

// each square is magic.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import cs1.Keyboard;

public class SquareTest

{

public static void main(String[] args)

{

int count = 1; //count which square we're on

int size = Keyboard.readInt(); //size of next square

//Expecting -1 at bottom of input file

while (size != -1)

{

//create a new Square of the given size

//call its read method to read the values of the square

System.out.println("\n\*\*\*\*\*\*\*\* Square " + count + " \*\*\*\*\*\*\*\*");

//print the square

//print the sums of its rows

//print the sums of its columns

//print the sum of the main diagonal

//print the sum of the other diagonal

//determine and print whether it is a magic square

//get size of next square

size = Keyboard.readInt();

}

}

}

**magicData**

3

8 1 6

3 5 7

4 9 2

7

30 39 48 1 10 19 28

38 47 7 9 18 27 29

46 6 8 17 26 35 37

5 14 16 25 34 36 45

13 15 24 33 42 44 4

21 23 32 41 43 3 12

22 31 40 49 2 11 20

4

48 9 6 39

27 18 21 36

15 30 33 24

12 45 42 3

3

6 2 7

1 5 3

2 9 4

4

3 16 2 13

6 9 7 12

10 5 11 8

15 4 14 1

5

17 24 15 8 1

23 5 16 14 7

4 6 22 13 20

10 12 3 21 19

11 18 9 2 25

7

30 39 48 1 10 28 19

38 47 7 9 18 29 27

46 6 8 17 26 37 35

5 14 16 25 34 45 36

13 15 24 33 42 4 44

21 23 32 41 43 12 3

22 31 40 49 2 20 11

-1

**A Shopping Cart**

In this exercise you will complete a class that implements a shopping cart as an array of items. The file *Item.java* contains the definition of a class named *Item* that models an item one would purchase. An item has a name, price, and quantity (the quantity purchased). The file *ShoppingCart.java* implements the shopping cart as an array of Item objects.

1. Complete the *ShoppingCart* class by doing the following:

a. Declare an instance variable *cart* to be an array of Items and instantiate *cart* in the constructor to be an array holding *capacity* Items.

b. Fill in the code for the *increaseSize* method. Your code should be similar to that in Listing 6.8 of the text but instead of doubling the size just increase it by 3 elements.

c. Fill in the code for the *addToCart* method. This method should add the item to the cart and update the *totalPrice* instance variable (note this variable takes into account the quantity).

d. Compile your class.

1. Write a program that simulates shopping. The program should have a loop that continues as long as the user wants to shop. Each time through the loop read in the name, price, and quantity of the item the user wants add to the cart. After adding an item to the cart, the cart contents should be printed. After the loop print a "Please pay ..." message with the total price of the items in the cart.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Item.java

//

// Represents an item in a shopping cart.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.text.NumberFormat;

public class Item

{

private String name;

private double price;

private int quantity;

// -------------------------------------------------------

// Create a new item with the given attributes.

// -------------------------------------------------------

public Item (String itemName, double itemPrice, int numPurchased)

{

name = itemName;

price = itemPrice;

quantity = numPurchased;

}

// -------------------------------------------------------

// Return a string with the information about the item

// -------------------------------------------------------

public String toString ()

{

NumberFormat fmt = NumberFormat.getCurrencyInstance();

return (name + "\t" + fmt.format(price) + "\t" + quantity + "\t"

+ fmt.format(price\*quantity));

}

// -------------------------------------------------

// Returns the unit price of the item

// -------------------------------------------------

public double getPrice()

{

return price;

}

// -------------------------------------------------

// Returns the name of the item

// -------------------------------------------------

public String getName()

{

return name;

}

// -------------------------------------------------

// Returns the quantity of the item

// -------------------------------------------------

public int getQuantity()

{

return quantity;

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// ShoppingCart.java

//

// Represents a shopping cart as an array of items

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import Item;

import java.text.NumberFormat;

public class ShoppingCart

{

private int itemCount; // total number of items in the cart

private double totalPrice; // total price of items in the cart

private int capacity; // current cart capacity

// -----------------------------------------------------------

// Creates an empty shopping cart with a capacity of 5 items.

// -----------------------------------------------------------

public ShoppingCart()

{

capacity = 5;

itemCount = 0;

totalPrice = 0.0;

}

// -------------------------------------------------------

// Adds an item to the shopping cart.

// -------------------------------------------------------

public void addToCart(String itemName, double price, int quantity)

{

}

// -------------------------------------------------------

// Returns the contents of the cart together with

// summary information.

// -------------------------------------------------------

public String toString()

{

NumberFormat fmt = NumberFormat.getCurrencyInstance();

String contents = "\nShopping Cart\n";

contents += "\nItem\t\tUnit Price\tQuantity\tTotal\n";

for (int i = 0; i < itemCount; i++)

contents += cart[i].toString() + "\n";

contents += "\nTotal Price: " + fmt.format(totalPrice);

contents += "\n";

return contents;

}

// ---------------------------------------------------------

// Increases the capacity of the shopping cart by 3

// ---------------------------------------------------------

private void increaseSize()

{

}

}

**A Shopping Cart Using the ArrayList Class**

In this exercise you will implements a shopping cart using the ArrayList class. The file *Item.java* contains the definition of a class named *Item* that models an item one would purchase. An item has a name, price, and quantity (the quantity purchased). The file *Shop.java* is an incomplete program that models shopping.

1. Complete Shop.java as follows:

a. Declare and instantiate a variable *cart* to be an empty ArrayList.

b. Fill in the statements in the loop to add an item to the cart and to print the cart contents (using the default *toString* in the ArrayList class). Comments in the code indicate where these statements go.

c. Compile your program and run it.

2. You should have observed two problems with using the default printing for the cart object. The output doesn't look very good and the total price of the goods in the cart is not computed or printed. Modify the program to correct these problems by replacing the print statement with a loop that does the following:

a. gets each item from the cart and prints the item

b. computes the total price of the items in the cart (you need to use the *getPrice* and *getQuantity* methods of the Item class). The total price should be printed after the loop.

3. Compile and run your program.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Shop.java

//

// Uses the Item class to create items and add them to a shopping

// cart stored in an ArrayList.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.util.ArrayList;

import Item;

import cs1.Keyboard;

public class Shop

{

public static void main (String[] args)

{

ArrayList cart = new ArrayList();

Item item;

String itemName;

double itemPrice;

int quantity;

String keepShopping = "y";

do

{

System.out.print ("Enter the name of the item: ");

itemName = Keyboard.readString();

System.out.print ("Enter the unit price: ");

itemPrice = Keyboard.readDouble();

System.out.print ("Enter the quantity: ");

quantity = Keyboard.readInt();

// \*\*\* create a new item and add it to the cart

// \*\*\* print the contents of the cart object using println

System.out.print ("Continue shopping (y/n)? ");

keepShopping = Keyboard.readString();

}

while (keepShopping.equals("y"));

}

}

**Drawing Circles (Again)**

File *DrawCircles.java* contains an applet to create and draw 10 circles.(This was a lab exercise from Chapter 4.) It uses class *Circle.java* and is called from *DrawCircles.html*. Save all of these files to your directory. Use the appletviewer and DrawCircles.html to run DrawCircles.java. Note that when the applet is redrawn (e.g., when the window is popped) a different set of circles is drawn. This is because the circles are randomly created in *paint*, which is called every time the applet is drawn. The applet can't draw the same circles as before because it doesn't store them.

Modify this applet so that it creates 10 circles and stores them in an array, then simply draws each circle in the array in *paint.* You will need to do the following:

1. Declare and initialize an array to hold ten Circle objects. This should be an instance variable—it should not be local to any method.

2. Add an *init* method (public void init()). Remember that *init* is called only once, when the applet is loaded, which is what you want—this way you only create one set of circles.

3. In *init*, write a loop that fills the array with Circle objects. Each time through the loop you will need to create a new circle and store it in the next slot in the array.

4. Modify *paint* to run through the array and draw each circle. No new circles should be created in *paint*!!

When you run your program now, you should see the same set of circles when you redraw, but a new set if you choose Restart or Reload from the applet menu.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// DrawCircles.java

//

// An applet that uses the Circle class to create and draw

// 10 random circles.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.awt.\*;

import java.applet.Applet;

import Circle;

public class DrawCircles extends Applet

{

//---------------------------------------------------------

// Create 10 circles and draw each one. Note that the circles

// are not saved, so they will be different each time

// paint is called.

//---------------------------------------------------------

public void paint (Graphics page)

{

for (int i=0;i<10;i++)

{

Circle c = new Circle();

c.draw(page);

}

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Circle.java

//

// Define a Circle class with methods to create and draw

// a circle of random size, color, and location.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.awt.\*;

import java.util.Random;

public class Circle{

private int x, y;

private int radius;

private Color color;

static Random generator = new Random();

//---------------------------------------------------------

// Creates a random circle with properties in ranges given:

// -- radius 25..74

// -- color RGB value 0..16777215 (24-bit)

// -- x-coord of upper left-hand corner 0..399

// -- y-coord of upper left-hand corner 0..199

//---------------------------------------------------------

public Circle()

{

radius = Math.abs(generator.nextInt())%50 + 25;

color = new Color(Math.abs(generator.nextInt())% 16777216);

x = Math.abs(generator.nextInt())%400;

y = Math.abs(generator.nextInt())%200;

}

//---------------------------------------------------------

// Draws circle on graphics object given

//---------------------------------------------------------

public void draw(Graphics page)

{

page.setColor(color);

page.fillOval(x,y,radius\*2,radius\*2);

}

}

**Drawcircles.html**

<html>

<applet width=600 height=400 code=DrawCircles.class>

</applet>

</html>

**A Polygon Person**

A polygon is a multisided closed figure; a polyline is a line with an arbitrary number of segments. Both polygons and polylines are defined by a set of points, and Java provides graphics methods for both that are based on arrays. Read section 6.5 in the text and study the Rocket example in Listing 6.17.

File *DrawPerson.java* contains an applet that draws a blue shirt. Copy it and the associated *html file* to your directory, compile DrawPerson.java, and run it using the appletviewer to see what it does. Now modify it as follows:

1. Draw pants to go with the shirt (they should be a different color). You will need to declare pantsX and pantsY arrays like the shirtX and shirtY arrays and figure out what should go in them. Then make the paint method draw the pants as well as the shirt.

2. Draw a head. This can just be a circle (or oval), so you won't need to use the Polygon methods. Declare variables headX and headY to hold the position of the head (its upper lefthand corner), and use them when you draw the circle.

3. Draw hair on the head. This is probably best done with a polygon, so again you'll need two arrays to hold the points.

4. Draw a zigzag across the front of the shirt. Use a polyline.

5. Write a method *movePersocd .n(int x, int y)* that moves the person by the given number of pixels in the x and y direction. This method should just go through the shirt, pants, hair and zigzag arrays and the head x and y coords and increment all of the coordinates by the x or y value as appropriate. (This isn't necessarily the cleanest way to do this, but it's very straightforward).

6. Now put a loop in your paint method that draws the person three times, moving him (her?) 150 or so pixels each time (you decide how far).

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// DrawPerson.java

//

// An applet that uses the Graphics draw methods to draw a person.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.awt.\*;

import java.applet.Applet;

public class DrawPerson extends Applet{

private final int APPLET\_WIDTH = 600;

private final int APPLET\_HEIGHT = 400;

private int[] shirtX = {60,0,20,60,50,130,120,160,180,120};

private int[] shirtY = {100,150,180,160,250,250,160,180,150,100};

//--------------------------------------

// Set applet size

//--------------------------------------

public void init()

{

setSize(APPLET\_WIDTH, APPLET\_HEIGHT);

}

//--------------------------------------

// Draw person

//--------------------------------------

public void paint (Graphics page)

{

page.setColor(Color.blue);

page.fillPolygon(shirtX, shirtY, shirtX.length);

}

}

**html file**

<html>

<applet width=600 height=400 code=DrawPerson.class>

</applet>

</html>

**Adding Buttons to *StyleGUI.java***

The files *StyleOptions.java* and *StyleGUI.java* are from Listings 6.18 and 6.19 of the text (with a couple of slight changes—a variable *fontSize* is used rather than the constant FONT\_SIZE and the variable *style* is an instance variable rather than local to the itemStateChanged method). The program demonstrates checkboxes and ItemListeners. In this exercise you will add a set of 3 radio buttons to let the user choose among three font sizes. The method of adding the radio buttons will be very similar to that in the QuoteGUI class (Listing 6.21 of the text). Before modifying the program compile and run the current version to see how it works and study the QuoteGUI example.

Do the following to add the radio buttons to the GUI:

1. Declare three objects *small*, *medium*, and *large* of type JRadioButton.

2. Instantiate the button objects labeling them "Small Font," "Medium Font," "Large Font." Initialize the large font button to true. Set the background color of the buttons to cyan.

3. Instantiate a button group object and add the buttons to it.

4. Radio buttons produce action events so you need to add an inner class (name it *SizeListener*) to implement ActionListener and listen for radio button clicks. The code for *actionPerformed* will be similar to that in the QuoteListener in Listing 6.21. In this case you need to set the *fontSize* variable (use 12 for small, 24 for medium, and 36 for large) in the if statement, then call the *setFont* method to set the font for the *saying* object.

5. In *StyleGUI()* instantiate a SizeListener and add it to each button. Also add each button to the panel.

6. Compile and run the program. Note that as the font size changes the checkboxes and buttons re-arrange themselves in the panel. You will learn how to control layout later in the course.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// StyleOptions.java Author: Lewis/Loftus

//

// Demonstrates the use of check boxes.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import javax.swing.\*;

public class StyleOptions

{

//-----------------------------------------------------------------

// Creates and presents the program frame.

//-----------------------------------------------------------------

public static void main (String[] args)

{

JFrame styleFrame = new JFrame ("Style Options");

styleFrame.setDefaultCloseOperation (JFrame.EXIT\_ON\_CLOSE);

StyleGUI gui = new StyleGUI();

styleFrame.getContentPane().add (gui.getPanel());

styleFrame.pack();

styleFrame.show();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// StyleGUI.java Author: Lewis/Loftus

//

// Represents the user interface for the StyleOptions program.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

public class StyleGUI

{

private final int WIDTH = 300, HEIGHT = 150;

private int fontSize = 36;

private int style = Font.PLAIN;

private JLabel saying;

private JCheckBox bold, italic;

private JPanel primary;

//-----------------------------------------------------------------

// Sets up a panel with a label and some check boxes that

// control the style of the label's font.

//-----------------------------------------------------------------

public StyleGUI()

{

saying = new JLabel ("Say it with style!");

saying.setFont (new Font ("Helvetica", style, fontSize));

bold = new JCheckBox ("Bold");

bold.setBackground (Color.cyan);

italic = new JCheckBox ("Italic");

italic.setBackground (Color.cyan);

StyleListener listener = new StyleListener();

bold.addItemListener (listener);

italic.addItemListener (listener);

primary = new JPanel();

primary.add (saying);

primary.add (bold);

primary.add (italic);

primary.setBackground (Color.cyan);

primary.setPreferredSize (new Dimension(WIDTH, HEIGHT));

}

//-----------------------------------------------------------------

// Returns the primary panel containing the GUI.

//-----------------------------------------------------------------

public JPanel getPanel()

{

return primary;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Represents the listener for both check boxes.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

private class StyleListener implements ItemListener

{

//--------------------------------------------------------------

// Updates the style of the label font style.

//--------------------------------------------------------------

public void itemStateChanged (ItemEvent event)

{

style = Font.PLAIN;

if (bold.isSelected())

style = Font.BOLD;

if (italic.isSelected())

style += Font.ITALIC;

saying.setFont (new Font ("Helvetica", style, fontSize));

}

}

}

**An Array of Radio Buttons**

File *ColorOptions.java* contains a program that will display a set of radio buttons that let the user change the background color of the GUI. The file *ColorGUI.java* contains the skeleton of the GUI for this program . Open the files and study the code that is already there. You will note that in ColorGUI.java there is an array *color* containing 5 colors already defined. Your task is to add an array of radio buttons so that a click of a radio button will cause the background of the GUI to change to the corresponding color in the *color* array.

1. Define *colorButton* to be an array of NUM\_COLORS objects of type JRadioButton.

2. Instantiate each *colorButton* with the appropriate color as the label (for example, the first button should be labeled "Yellow"). The first button (corresponding to yellow) should be on (*true*) initially.

3. Recall that radio buttons must be grouped and that the selection of a radio button produces an action event. Hence you must have a ButtonGroup object and an ActionListener. Note that the skeleton of an ActionListener named *ColorListener* is already provided. So, you need to:

a. Instantiate a ButtonGroup object and a ColorListener object. Comments in the code indicate where to do this.

b. Each radio button needs to be added to your ButtonGroup object, the background color needs to be set (use white), your ColorListener needs to be added, and the button needs to be added to the panel. All of these can be done using a single for loop. So, add a for loop that goes through the radio buttons adding each to your ButtonGroup object, setting the background of each to white, adding your ColorListener to each, and adding each to the panel.

4. Fill in the body of the actionPerformed method. This method needs to go through the buttons to determine which is selected and then set the background color accordingly. A simple for loop can do this. Use the *isSelected* method to determine if a button is selected (for example, if (colorButton[i].isSelected())....). Use the *color* array to set the background color.

5. Test your program!

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// ColorOptions.java

//

// Uses an array of radio buttons to change the background color.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import javax.swing.\*;

public class ColorOptions

{

// -------------------------------------------------------------

// Creates and presents the frame for the color change GUI.

// -------------------------------------------------------------

public static void main (String[] args)

{

JFrame colorFrame = new JFrame ("Color Options");

colorFrame.setDefaultCloseOperation (JFrame.EXIT\_ON\_CLOSE);

ColorGUI gui = new ColorGUI();

colorFrame.getContentPane().add (gui.getPanel());

colorFrame.pack();

colorFrame.show();

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// ColorGUI.java

//

// Represents the user interface for the ColorOptions program that lets

// the user change background color by selecting a radio button.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

public class ColorGUI

{

private final int WIDTH = 350, HEIGHT = 100, FONT\_SIZE = 20;

private final int NUM\_COLORS = 5;

private JPanel primary;

private Color [] color = new Color[NUM\_COLORS];

private JLabel heading;

// ------------------------------------------------------------------

// Sets up a GUI with a label at the top and a set of radio buttons

// that control the background color of the panel.

// ------------------------------------------------------------------

public ColorGUI ()

{

// Set up heading and colors

heading = new JLabel ("Choose the background color!");

heading.setFont (new Font ("Helvetica", Font.BOLD, FONT\_SIZE));

color[0] = Color.yellow;

color[1] = Color.cyan;

color[2] = Color.red;

color[3] = Color.green;

color[4] = Color.magenta;

// Set up the panel

primary = new JPanel ();

primary.add (heading);

primary.setBackground (Color.yellow);

primary.setPreferredSize (new Dimension (WIDTH, HEIGHT));

// Instantiate a ButtonGroup object and a ColorListener object

// Group the radio buttons, add a ColorListener to each,

// set the background color of each and add each to the panel.

}

// --------------------------------------------------------------

// Returns the primary panel containing the GUI.

// --------------------------------------------------------------

public JPanel getPanel()

{

return primary;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Represents the listener for the radio buttons.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

private class ColorListener implements ActionListener

{

// --------------------------------------------------------

// Updates the background color of the panel based on

// which radio button is selected.

// --------------------------------------------------------

public void actionPerformed (ActionEvent event)

{

}

}

}