2023-02-06

Unit 7 Test Review

_____ 9) Consider the following correct implementation of the insertion sort algorithm.

```
public static void insertionSort(int[] elements)
{
     for (int j = 1; j < elements.length; j++)
     {
          int temp = elements[j];
          int possibleIndex = j;
          while (possibleIndex > 0 && temp < elements[possibleIndex - 1])
          {
               elements[possibleIndex] = elements[possibleIndex - 1];
               possibleIndex--;    // Line 10
          }
          elements[possibleIndex] = temp;

     }
}
```

The following declaration and method call appear in a method in the same class as `insertionSort`.

```
int[] arr = {4, 12, 4, 7, 19, 6};
insertionSort(arr);
```

How many times is the statement `possibleIndex--;` in line 10 of the method executed as a result of the call to `insertionSort` ?

(A) 2 (B) 3 (C) 4 (D) 5 (E) 6

int[] arr = {4, 12, 4, 7, 19, 6};

j=1 {4, 12, 4, 7, 19, 6} a[j]=12
0 swaps

j=2 {4, 12, 4, 7, 19, 6} a[j]=4
1 swap

j =3 {4, 4, 12, 7, 19, 6} a[j]=7
1 swap

j=4 {4, 4, 7, 12, 19, 6} a[j]=19
0 swaps

j=5 {4, 4, 7, 12, 19, 6} a[j]=6
3 swaps

TOTAL - 5 swaps

```
public static void insertionSort(int[] elements)
{
     for (int j = 1; j < elements.length; j++)
     {
          int temp = elements[j];
          int possibleIndex = j;
          while (possibleIndex > 0 && temp < elements[possibleIndex - 1])
          {
               elements[possibleIndex] = elements[possibleIndex - 1];
               possibleIndex--;    // Line 10
          }
          elements[possibleIndex] = temp;

     }
}
```

```java
/** Creates and returns a new Animal object, as described in part (a) */
public Animal createNewAnimal(String name, String type, double age) {
  int cost = 15;
  if (age < 1.0) {
    int count = 0;
    for (Animal a : allAnimals) {
      if (a.getType().equals(type)) {
        count++;
      }
    }
    if (count < 5) {
      cost = 25;
    } else {
      cost = 20;
    }
  }
  return new Animal(name, type, age, cost);
}
```

This is not a constructor!

This is an example of the Factory pattern: AnimalShelter knows best how much Animals cost, based on its own state. So the cost "business logic" lives in AnimalShelter, not Animal.

Animal's constructor is probably very simple and just copies its parameters into instance variables.

Some folks noticed you don't have to count up the animals of a certain type if age >= 1.0. Good optimization!

```java
/** Adds an animal to the list allAnimals, as described in part (b) */
public void addAnimal(String name, String type, double age) {
  int i = 0;
  while (i < allAnimals.size() && age > allAnimals.get(i).getAge()) {
    i++;
  }
  allAnimals.add(i, createNewAnimal(name, type, age));
}
```

Many folks wrote an Insertion Sort, or even a Selection Sort. Hard to get 100% right!

If you're asked an ArrayList question, you CAN bust out ArrayList.add(index, object), and then you just need to find the insertion point, not code a full-blown sort.

Remember that ArrayList.add(0, object) works fine when size == 0.
(But ArrayList.set(0, object) does NOT work when size == 0.)

8.2
Traversing
Two-Dimensional Arrays

for loops

Review: Traversing Arrays with `for` loops

- Remember that the range of valid Array indexes (for non-empty Arrays) is `0` to `Array.length - 1`

```
int scores[] = {95, 100, 91, 85 };
for (int idx = 0; idx < scores.length; idx++)
{
    System.out.println(scores[ idx]);
}
```

Review: Traversing Arrays with `for` loops

- Remember that the range of valid Array indexes (for non-empty Arrays) is `0` to `Array.length - 1`

```
int scores[] = {95, 100, 91, 85 };
for (int idx = 0; idx < scores.length; idx++)
{
    System.out.println(scores[ idx]);
}
```

```
int scores[] = {95, 100, 91, 85 };
for (int idx = 1; idx <= scores.length;
idx++) {
    System.out.println(scores[ idx]);
}
```

Review: Traversing Arrays with `for` loops

- Remember that the range of valid Array indexes (for non-empty Arrays) is `0` to `Array.length - 1`

```
int scores[] = {95, 100, 91, 85 };
for (int idx = 0; idx < scores.length; idx++)
{
    System.out.println(scores[ idx]);
}
```

```
int scores[] = {95, 100, 91, 85 };
for (int idx = 1; idx <= scores.length;
idx++) {
    System.out.println(scores[ idx]);
}
```

Note: *Passing an out of range index will cause a* `ArrayIndexOutOfBoundsException`*!*

Review: Traversing Arrays with `for` loops

- Remember that the range of valid Array indexes (for non-empty Arrays) is `0` to `Array.length - 1`

```
int scores[] = {95, 100, 91, 85 };
for (int idx = 0; idx < scores.length; idx++)
{
    System.out.println(scores[idx]);
}
```

```
int scores[] = {95, 100, 91, 85 };
for (int idx = 1; idx <= scores.length;
idx++) {
    System.out.println(scores[idx]);
}
```

This loop also skips the first element in the Array!

Note: *Passing an out of range index will cause a* `ArrayIndexOutOfBoundsException`*!*

Review: Traversing Arrays with `for` loops

- You can use a `for` loop to traverse an Array from back to front!

```
int scores[] = {95, 100, 91, 85 };
for (int idx = scores.length - 1; idx >= 0; idx--) {
    System.out.println(scores[ idx]);
}
```

- ...or to traverse any arbitrary range of elements

```
int scores[] = {95, 100, 91, 85 };
for (int idx = 1; idx <= 2; idx++) {
    System.out.println(scores[ idx]);
}
```

Traversing Two-Dimensional Arrays with `for` loops

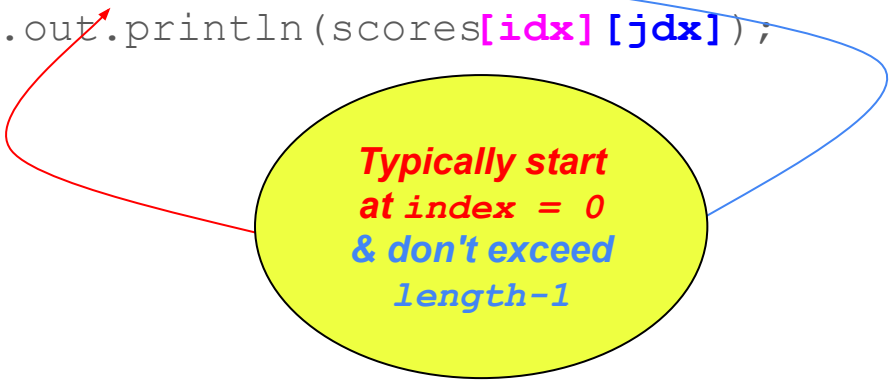
- Traversing Two-Dimensional Arrays is very similar

```
int scores[ ][ ] = { {10,20,30},{40,50,60} };
for (int idx = 0; idx < scores.length; idx++) {
  for (int jdx = 0; jdx < scores[idx].length; jdx++) {
    System.out.println(scores[idx][jdx]);
  }
}
```

Traversing Two-Dimensional Arrays with `for` loops

- Traversing Two-Dimensional Arrays is very similar

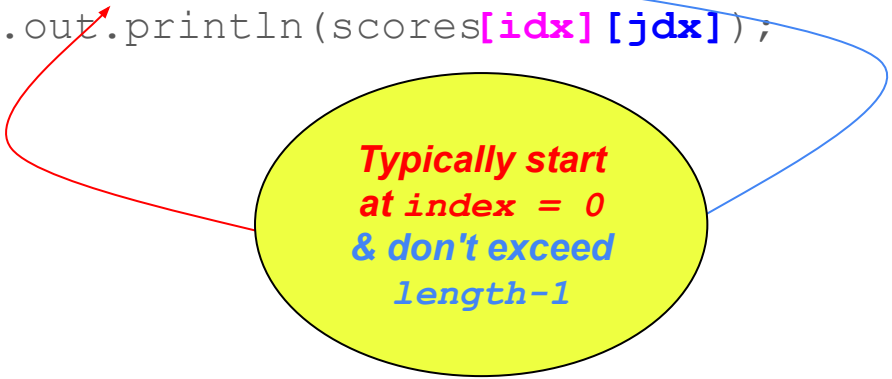
```
int scores[ ][ ] = { {10,20,30},{40,50,60} };
for (int idx = 0; idx < scores.length; idx++) {
  for (int jdx = 0; jdx < scores[idx].length; jdx++) {
    System.out.println(scores[idx][jdx]);
  }
}
```



Traversing Two-Dimensional Arrays with `for` loops

- Traversing Two-Dimensional Arrays is very similar

```
int scores[][] = {{10,20,30},{40,50,60}};
for (int idx = 0; idx < scores.length; idx++) {
  for (int jdx = 0; jdx < scores[idx].length; jdx++) {
    System.out.println(scores[idx][jdx]);
  }
}
```



Note: *Passing an out of range index will cause a* `ArrayIndexOutOfBoundsException`*!*

for-each loops

Review: Traversing Arrays with `for-each` loops

```
for (type arrayItemVariable : arrayVariable) {
    arrayItemVariable resolves to arrayVariable[...]
}
```

```
String[] colors = {"red", "orange", "purple"};
```

```
System.out.println("begin");
for(String color: colors){
  System.out.println(" " + color);
}
System.out.println("end");
```

Review: Traversing Arrays with `for-each` loops

- The type of the `for-each` variable MUST match the type of the values stored in the Array

```
String colors[] = {"red", "orange", "purple"};

for(int color: colors){
  System.out.println(" " + color);
}
```

Review: Traversing Arrays with `for-each` loops

- The type of the `for-each` variable MUST match the type of the values stored in the Array

```
String colors[] = {"red", "orange", "purple"};

for(int color: colors){
  System.out.println(" " + color);
}
```

Note: *color must be of type String since colors is an Array that contains Strings*

Traversing Two-Dimensional Arrays with for-each loops

- Remember during the introduction of Two-Dimensional Arrays - We said: "***Arrays are a type*** *- Which means you can easily create an* ***Array that contains Arrays*** *- often called Two-Dimensional Arrays"*
- That means we can use `for-each` to traverse a Two-Dimensional Array almost exactly like a One-Dimensional Array (**we just have to be careful how we declare the types**)

Traversing Two-Dimensional Arrays with for-each loops

- Remember during the introduction of Two-Dimensional Arrays - We said: "***Arrays are a type*** *- Which means you can easily create an* ***Array that contains Arrays*** *- often called Two-Dimensional Arrays"*
- That means we can use `for-each` to traverse a Two-Dimensional Array almost exactly like a One-Dimensional Array (**we just have to be careful how we declare the types**)

Reminder: `for-each` loops can be super-useful; but you are unable to make use of an index or change the underlying Array while looping

Traversing Two-Dimensional Arrays with for-each loops

Given this Two-Dimensional Array

```
int scores[][] = {{10,20,30},{40,50,60}};
```

And this general description of for-each

```
for (type arrayItemVariable : arrayVariable) {
    arrayItemVariable resolves to Array[...]
}
```

Q: What is the type of the outer array in scores?

Traversing Two-Dimensional Arrays with for-each loops

Given this Two-Dimensional Array

```
int scores[][] = {{10,20,30},{40,50,60}};
```

And this general description of for-each

```
for (type arrayItemVariable : arrayVariable) {
    arrayItemVariable resolves to Array[...]
}
```

Q: What is the type of the outer array in scores?

scores[] -> an array of int[]

Traversing Two-Dimensional Arrays with for-each loops

Given this Two-Dimensional Array

```
int scores[][] = {{10,20,30},{40,50,60}};
```

And this general description of for-each

```
for (type arrayItemVariable : arrayVariable) {
    arrayItemVariable resolves to Array[...]
}
```

Q: What is the type of the outer array in scores?

scores[] -> an array of int[]

Q: What is the type of the inner array in scores?

Traversing Two-Dimensional Arrays with for-each loops

Given this Two-Dimensional Array

```
int scores[][] = {{10,20,30},{40,50,60}};
```

And this general description of for-each

```
for (type arrayItemVariable : arrayVariable) {
    arrayItemVariable resolves to Array[...]
}
```

Q: What is the type of the outer array in scores?

scores[] -> an array of int[]

Q: What is the type of the inner array in scores?

scores[][] -> an array of int

Traversing Two-Dimensional Arrays with for-each loops

```
int scores[][] = {{10,20,30},{40,50,60}};
```

scores[] -> an array of int[]
scores[][] -> an array of int

So we can use `for-each` to traverse the Two-Dimensional Array like this

```
for (int[] outer : scores) {
  for (int inner : outer) {
    System.out.println(inner);
  }
}
```

Traversing Two-Dimensional Arrays with for-each loops

```
int scores[][] = {{10,20,30},{40,50,60}};
```

scores[] -> an array of int[]
scores[][] -> an array of int

So we can use `for-each` to traverse the Two-Dimensional Array like this

```
for (int[] outer : scores) {
  for (int inner : outer) {
    System.out.println(inner);
  }
}
```

Note: The inner array is a One-Dimensional Array - So we use the same `for-each` that we used previously for One-Dimensional Arrays

```
for (type arrayItemVariable : arrayVariable) {
  arrayItemVariable resolves to arrayVariable[...]
}
```

Traversing Two-Dimensional Arrays with for-each loops

```
int scores[][] = {{10,20,30},{40,50,60}};
```

scores[] -> an array of int[]
scores[][] -> an array of int

So we can use `for-each` to traverse the Two-Dimensional Array like this

```
for (int[] outer : scores) {
  for (int inner : outer) {
    System.out.println(inner);
  }
}
```

No indices available for use!

Note: The inner array is a One-Dimensional Array - So we use the same `for-each` that we used previously for One-Dimensional Arrays

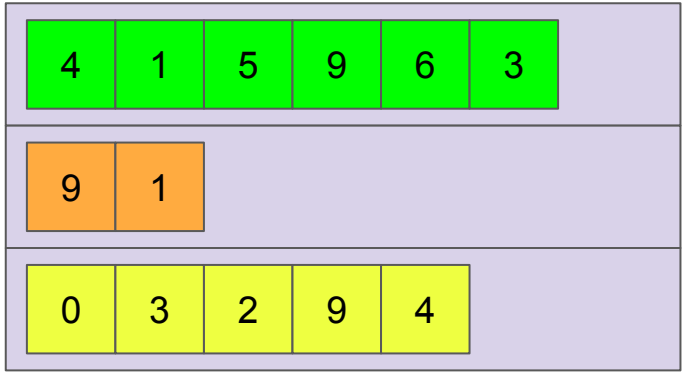
```
for (type arrayItemVariable : arrayVariable) {
  arrayItemVariable resolves to arrayVariable[...]
}
```

Traversal of Jagged Arrays

- Traversal of Two-Dimensional Jagged Arrays works the same!

```
int jaggedTable[][] = new int[3][];

jaggedTable[0] = new int[]{4,1,5,9,6,3};
jaggedTable[1] = new int[]{9,1};
jaggedTable[2] = new int[]{0,3,2,9,4};
```



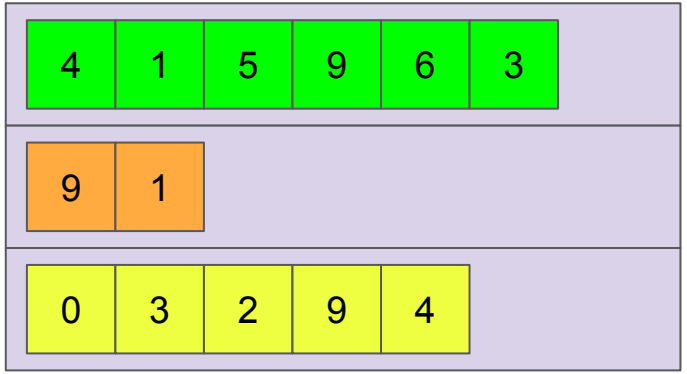
Traversal of Jagged Arrays

- Traversal of Two-Dimensional Jagged Arrays works the same!

```
int jaggedTable[][] = new int[3][];

jaggedTable[0] = new int[]{4,1,5,9,6,3};
jaggedTable[1] = new int[]{9,1};
jaggedTable[2] = new int[]{0,3,2,9,4};

for (int idx = 0; idx < jaggedTable.length; idx++) {
  for (int jdx = 0; jdx < jaggedTable[idx].length; jdx++) {
    System.out.print(jaggedTable[idx][jdx] + " ");
  }
  System.out.println();
}
```



Example: Basketball Scores

```
int NUM_PLAYERS = 5, NUM_GAMES = 3;
int scores[][] = new int[NUM_PLAYERS][NUM_GAMES];
```

	Game 1	Game 2	Game 3
Player 1	14	19	22
Player 2	24	13	5
Player 3	5	26	31
Player 4	0	18	40
Player 5	15	9	46

Example: Basketball Scores

```
int NUM_PLAYERS = 5, NUM_GAMES = 3;
int scores[][] = new int[NUM_PLAYERS][NUM_GAMES];
```

	Game 1	Game 2	Game 3
Player 1	14	19	22
Player 2	24	13	5
Player 3	5	26	31
Player 4	0	18	40
Player 5	15	9	46

`scores[][]`

Example: Basketball Scores

```
int NUM_PLAYERS = 5, NUM_GAMES = 3;
int scores[][] = new int[NUM_PLAYERS][NUM_GAMES];
```

	Game 1	Game 2	Game 3
Player 1	14	19	22
Player 2	24	13	5
Player 3	5	26	31
Player 4	0	18	40
Player 5	15	9	46

```
scores[3][0], scores[3][1], scores[3][2]
```

Example: Basketball Scores

```
int NUM_PLAYERS = 5, NUM_GAMES = 3;
int scores[][] = new int[NUM_PLAYERS][NUM_GAMES];
```

	Game 1	Game 2	Game 3
Player 1	14	19	22
Player 2	24	13	5
Player 3	5	26	31
Player 4	0	18	40
Player 5	15	9	46

```
scores[0][1], scores[1][1], scores[2][1], scores[3][1], scores[4][1]
```

Example: Basketball Scores

```
int NUM_PLAYERS = 5, NUM_GAMES = 3;
int scores[][] = new int[NUM_PLAYERS][NUM_GAMES];
```

	Game 1	Game 2	Game 3
Player 1	14	19	22
Player 2	24	13	5
Player 3	5	26	31
Player 4	0	18	40
Player 5	15	9	46

Q1: *How would you determine the total points scored by all 5 players in all 3 games?*
Q2: *How would you determine which player had the highest number of points in Game 3?*
Q3: *How would you determine the average points scored by Player 3 in all 3 games?*

Practice on your own

- CSAwesome 8.2 - Two-Dimensional Arrays
- Replit - [Chart Render]
  - Provide function bodies for
    - `Chart.getMaxSegmentTotal()`
    - `Chart.getMaxRowLabelWidth()`
    - `Chart.adjustFillBlocks()`
  - Two charts are pre-created into memory; Draw them using the draw command
    - `draw example0`
    - `draw example1`
  - Additional charts can be loaded from the `chart_data_files` directory using the load command
    - `load population-usa.txt`
    - `load population-usa-age.txt`
  - What other capabilities can you add?