CMSC131

Multi-Dimensional Structures

Seating Chart

Imagine you were going to write a Java class to store seating information about ESJ0202.

What would the data structure look like?

Take a moment to think about this and describe what you have in mind to your neighbor.

Multi-Dimensional Structures

We have seen some data structures that have been used as a single “one-dimensional” list (**`ArrayList`**, arrays, **`PriorityQueue`**).

We have also seen some that held information across two dimensions in a rectangular pattern (**`SquareGrid`**, **`Grid_3x5`**, **`Photograph`**).

We will now see how to build our own multi-dimensional structures…

Multi-Dimensional Arrays

It is sometimes useful to have a custom-made multi-dimensional structure for data storage.

- Consider things such as a chess board or warehouse, very different but unlikely to be stored in a single list.

We will declare them as an "Array of Arrays"

- Allows us to have "ragged edged" arrays.

We could declare them as a rectangular "xD Array"

- Doesn't give any real advantage in Java but might depending on the language.

2D Arrays – Syntax Examples

Rectangular

```
int[][] arr = new int[rows][cols];
```

Ragged

```
int[][] arr = new int[rows][];
arr[0] = new int[colsInRow0];
arr[1] = new int[colsInRow1];
arr[2] = new int[colsInRow2];
arr[3] = new int[colsInRow3];
 :
 :
```

Passing a multi-dimensional array

Passing a multi-dimensional array into a method is similar to passing a one-dimensional array; you just need to indicate how many dimensions there are to the array.

```
public static void takeArr(Float[][] arrParam){
//arrParam will "know" how many rows there are
//and each row will "know" how many cols it has

}
```

Upper Left-Hand Right Triangle

We want to create a structure in which we store the distance from the "origin" to each cell in the structure, but only for the upper left-hand right triangle starting at that origin.

Should it be stored in a Rectangular array or a Ragged array?

How would we make it work for a triangle with sides of length `triangleSize`?

Dealing with Ragged Arrays

```
int[][] reallyRagged;
  reallyRagged = new int[5][];
  reallyRagged[0] = new int[9];
  reallyRagged[1] = new int[3];
  reallyRagged[2] = new int[7];
  reallyRagged[3] = new int[0];   //NOTE
  reallyRagged[4] = new int[12];
```

How could you go in and perform an operation on each element in the entire structure without having to hard-code the different lengths?

Self-awareness

Recall that arrays know their own length!

```
counter = 0;
for (row=0; row<reallyRagged.length; row++) {
  for (col=0; col<reallyRagged[row].length; col++) {
    reallyRagged[row][col] = counter++;
  }
}
```

What would happen if we had used the following in the code on the previous slide?

```
reallyRagged[3] = null;
```

Using arrays with the `ArrayList`

It's also possible to use a hybrid of an array that you manage yourself and the **`ArrayList`** to create a multi-dimensional structure, but the syntax is a bit involved since Java doesn't allow mixing arrays and generics directly.

```
ArrayList<Float>[] twoDhybrid = (ArrayList<Float>[])new ArrayList[4];
…
twoDhybrid[0] = new ArrayList<Float>();
…
twoDhybrid[0].add(3.4F);
```

Using only the `ArrayList`

It is also *possible* to use the **`ArrayList`** rather than arrays you manage to create such a multi-dimensional structure.

```
ArrayList<ArrayList<Float>> twoDarraylist =
                  new ArrayList<ArrayList<Float>>(
twoDarraylist.add(new ArrayList<Float>());
twoDarraylist.add(new ArrayList<Float>());
twoDarraylist.add(new ArrayList<Float>());
twoDarraylist.get(0).add(0.1F);
twoDarraylist.get(1).add(1.1F);
twoDarraylist.get(2).add(2.1F);
twoDarraylist.add(1, new ArrayList<Float>());
```

Design Decisions

These last few options demonstrate the realities of software engineers having to balance the needs of the program (in terms of data structure flexibility) with the complexity of the syntax that might be required for a certain approach…

More than Two Dimensions

These examples expand to more than two dimensions. For example, if you were representing things being stored in a cube layout or even in multiple dimensions beyond the three traditional physical ones, the data structures could reflect this…

Rubik's Cube: How many dimensions to use?

- 20% A. One-dimensional (list)
- 20% B. Two-dimensional (matrix)
- 20% C. Three-dimensions (cube)
- 20% D. Fewer dimensions
- 20% E. More dimensions

Response Counter