Lecture 4
Intermediate Array Operations

More practical, advanced operations.

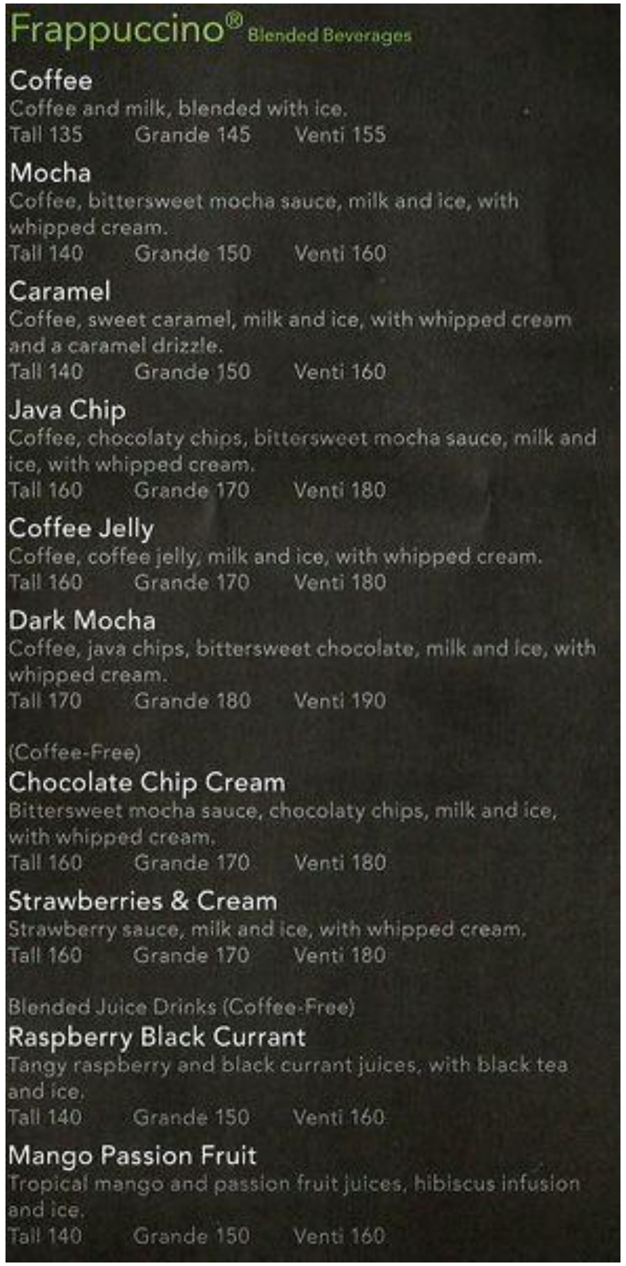
Lecture by Juan Lorenzo Hagad • College of Computer Studies • De La Salle University - Manila

Recap

Last meeting's Seatwork

With **Files**

- Frappuccino Menu
  - String[] name;
  - String[] description;
  - String[][] size;
  - double[][] price;



Frappuccino® Blended Beverages

Coffee
Coffee and milk, blended with ice.
Tall 135 Grande 145 Venti 155

Mocha
Coffee, bittersweet mocha sauce, milk and ice, with whipped cream.
Tall 140 Grande 150 Venti 160

Caramel
Coffee, sweet caramel, milk and ice, with whipped cream and a caramel drizzle.
Tall 140 Grande 150 Venti 160

Java Chip
Coffee, chocolaty chips, bittersweet mocha sauce, milk and ice, with whipped cream.
Tall 160 Grande 170 Venti 180

Coffee Jelly
Coffee, coffee jelly, milk and ice, with whipped cream.
Tall 160 Grande 170 Venti 180

Dark Mocha
Coffee, java chips, bittersweet chocolate, milk and ice, with whipped cream.
Tall 170 Grande 180 Venti 190

(Coffee-Free)
Chocolate Chip Cream
Bittersweet mocha sauce, chocolaty chips, milk and ice, with whipped cream.
Tall 160 Grande 170 Venti 180

Strawberries & Cream
Strawberry sauce, milk and ice, with whipped cream.
Tall 160 Grande 170 Venti 180

Blended Juice Drinks (Coffee-Free)
Raspberry Black Currant
Tangy raspberry and black currant juices, with black tea and ice.
Tall 140 Grande 150 Venti 160

Mango Passion Fruit
Tropical mango and passion fruit juices, hibiscus infusion and ice.
Tall 140 Grande 150 Venti 160

Exercise Problems

- After being able to **read data from a text file** into your menu arrays, implement the following methods:

1. Create a method **`listAlphabetical`** that prints out all the menu items (with complete details) in alphabetical order.
2. Create a method **`listCheaperThan`** that accepts a int **`val`** and prints out only the menu items (with details), and only the sizes with prices that are lower than **`val`**.
3. Create a method **`listOrderedByPrice`** that prints out all the menu items (with complete details) from the cheapest to the most expensive based on the smallest/cheapest drink size. Assume the first size in the list is always the cheapest.

Revised Exercise Problems

Finish in <u>30 mins.</u>

- After being able to **read data from a text file** into your menu arrays, implement the following methods:

1. Create a method `printFirst` that prints out only the menu item (complete details) that comes **first alphabetically** (based on name).

2. Create a method `listCheaperThan` that accepts a int `val` and lists the menu items (**name and description only**), but only if they have **at least one size** that is priced lower than `val`.

3. Create a method `listOrderedByPrice` that prints out all the menu items (with complete details) from **the most expensive to the cheapest** based on **the largest drink size**. Assume the last size in the list is always the most expensive.

Checking

- Prepare a 1/8th piece of Yellow Pad
- Write your name and section on the upper-left
- List down the item numbers that you have completely answered
  - E.g.,

    Juan de la Cruz S01

    1

    3

    ^means he was only able to answer items 1 and 3
- During checking, first present the console output for each item, then show the source code to the instructor.

Looping through 1D Array Elements

	name
0	Coffee
1	Mocha
2	Caramel
…	…

	description
0	Coffee and milk…
1	Coffee, bittersweet…
2	Coffee, sweet caramel…
…	…

```
for(int i = 0; i < name.length; i++){
  System.out.println(name[i]);
  System.out.println(description[i]);
}
```

Processing Array Elements: String Example

	name
0	Coffee
1	Mocha
2	Caramel
…	…

	description
0	Coffee and milk…
1	Coffee, bittersweet…
2	Coffee, sweet caramel…
…	…

```
public static void filterByName(String filter){
  for(int i = 0; i < name.length; i++){
    if(name.indexOf(filter) >= 0){
      System.out.println(name[i]);
      System.out.println(description[i]);
    }
  }
}
```

Read up on the **indexOf** method from the Java String API (Oracle Reference).

Searching for Specific Array Elements

	name
0	Coffee
1	Mocha
2	Caramel
…	…

	description
0	Coffee and milk…
1	Coffee, bittersweet…
2	Coffee, sweet caramel…
…	…

```
public static void printExact(String search){
  for(int i = 0; i < name.length; i++){
    if(name[i].equals(search)){
      System.out.println(name[i]);
      System.out.println(description[i]);
    }
  }
}
```

Searching is just filtering by **exact match**!

Swapping Around Array Elements

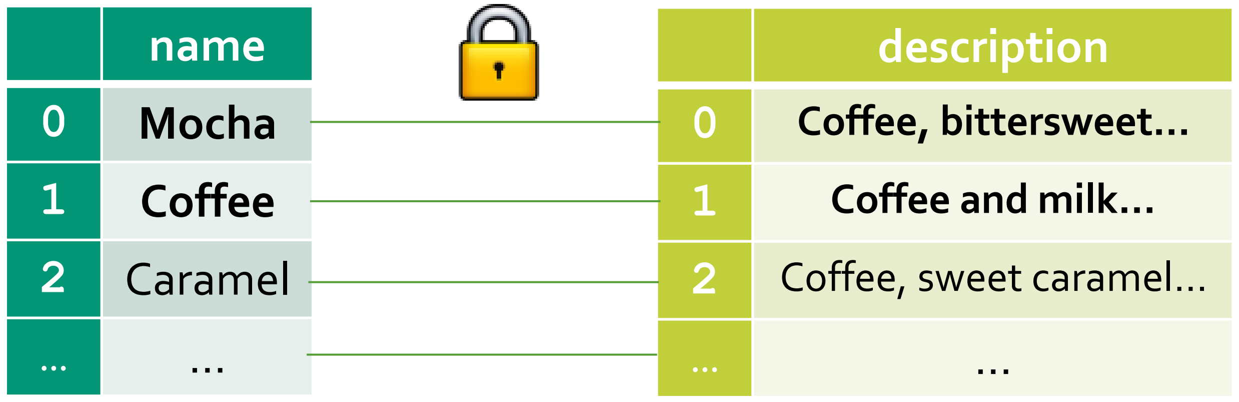
	name
0	Coffee
1	Mocha
2	Caramel
…	…

	name
0	**Mocha**
1	**Coffee**
2	Caramel
…	…

```
String temp = name[0];
name[0] = name[1];
name[1] = temp;
```

Just like swapping regular variables!

The menu: **Don't forget** to swap related data, too



	name
0	**Mocha**
1	**Coffee**
2	Caramel
…	…

	description
0	**Coffee, bittersweet…**
1	**Coffee and milk…**
2	Coffee, sweet caramel…
…	…

```
String temp = name[0];
name[0] = name[1];
name[1] = temp;
temp = description[0];
description[0] = description[1];
description[1] = temp;
```

API methods like Arrays.sort() cannot do this.

Sorting

- Sorting is just repetitive swapping around!
- Each operation transfers (swaps) elements which are in the wrong order, into their proper place.
- Every operation is a small step closer to forming the sorted list.
- Read up on basic sorting algorithms (insertion sort, selection sort, bubble sort, etc.)

Arrays are just a way to **organize related data**

- Arrays are just a way of managing a **set of something**:
  - SetOfSeats[]
  - SetOfRooms[]
  - SetOfGrades[]
  - SetOfFoodItems[]

- An index is like a **key** that we can use to get a **single sample of that something**:
  - [SeatPosition]
  - [RoomNumber]
  - [StudentNumber]
  - [ItemNumber]

Accessing 2D Array Elements

Col Row	0	1	2	3	4
0	43	15	7	101	-7
1	24	43	4	4	10
2	43	8	54	9	23
3	21	-32	23	88	54

`arrVar[0]`

- Think of it as a two-step process
  1. Accessing an array element which is **also a set of something (array)**
  2. Using the retrieved array, and further access its array element, which in this example is **an int**

Index	0	1	2	3	4
Value	43	15	7	101	-7

`arrVar[0] [1]`

A Short Analogy of 2D

- Referencing real word things in 2D
  - Dorm Room – [Floor][RoomNumber]
  - Parking Slot – [Floor][SlotNumber]
  - House – [Street][LotNumber]
  - Item Price – [Item][Variant]

Looping through 2D Arrays

size	0	1	2	...
0	Tall	Grande	Venti	...
1	Tall	Grande	Venti	...
2	Tall	Grande	Venti	...
...	...	...	...	...

price	0	1	2	...
0	135	145	155	...
1	140	150	160	...
2	140	150	160	...
...	...	...	...	...

```
for(int i = 0; i < size.length; i++){
  for(int j = 0; j < size[i].length; j++){
    System.out.print(size[i][j]);
    System.out.println(" - P"+price[i][j]);
  }
}
```

First, the outer loop (`i`) reads every row (menu item).
Then, the inner loop (`j`) reads every column of each row (variant).

Another way of looking at 2D array operations

```
for(int i = 0; i < size.length; i++){
  for(int j = 0; j < size[i].length; j++){
    System.out.print(size[i][j]); }
}
```

```
for(int i = 0; i < size.length; i++){
  String[] aRow = size[i];
  for(int j = 0; j < aRow.length; j++){
    System.out.print(aRow[j]);
  }
}
```

It's just a nested 1D array operation!

Beyond 2D

- It’s the same basic concept:

  Using many levels of organization.

E.g.,

[Building][Floor][Unit][Room][Seat]

[College][Section][Student]

[Menu][Item][Variant]