# CSCI 232:
# Data Structures and Algorithms

Java Review

Reese Pearsall

Spring 2024

# Announcements

Lab 1 due this Friday @ 11:59 PM
- Should have it posted within the next 24 hours

**Teaching Assistants:**

Section 003- **Sultan Yarylgassimov**
•Email: sultanyaril@gmail.com
•Office Hours: Mondays 10am - 12pm Barnard Hall 259

Section 004- **Muzhou (Peter) Chen**
•Email: muzhouchen@outlook.com
•Office Hours: Thursdays 9am - 11am Barnard Hall 259

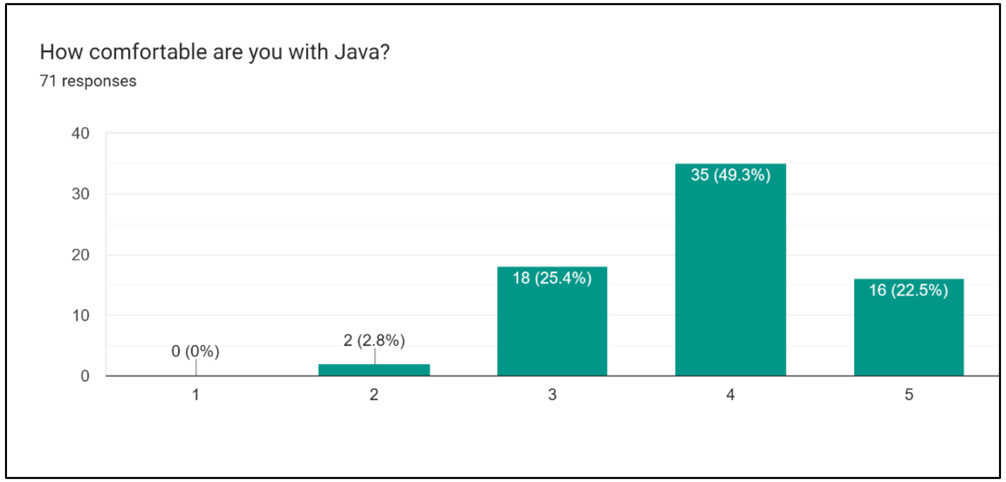
Section 005- **Sultan Yarylgassimov**
•Email: sultanyaril@gmail.com
•Office Hours: Mondays 10am - 12pm Barnard Hall 259

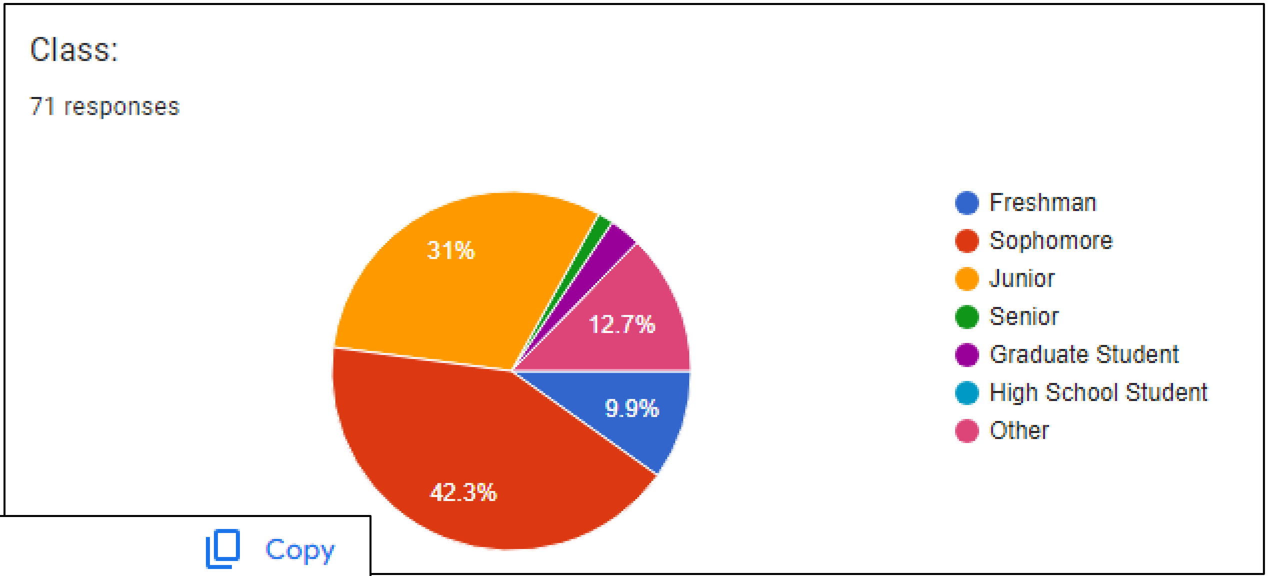
## Student Success Center - Spring 2024

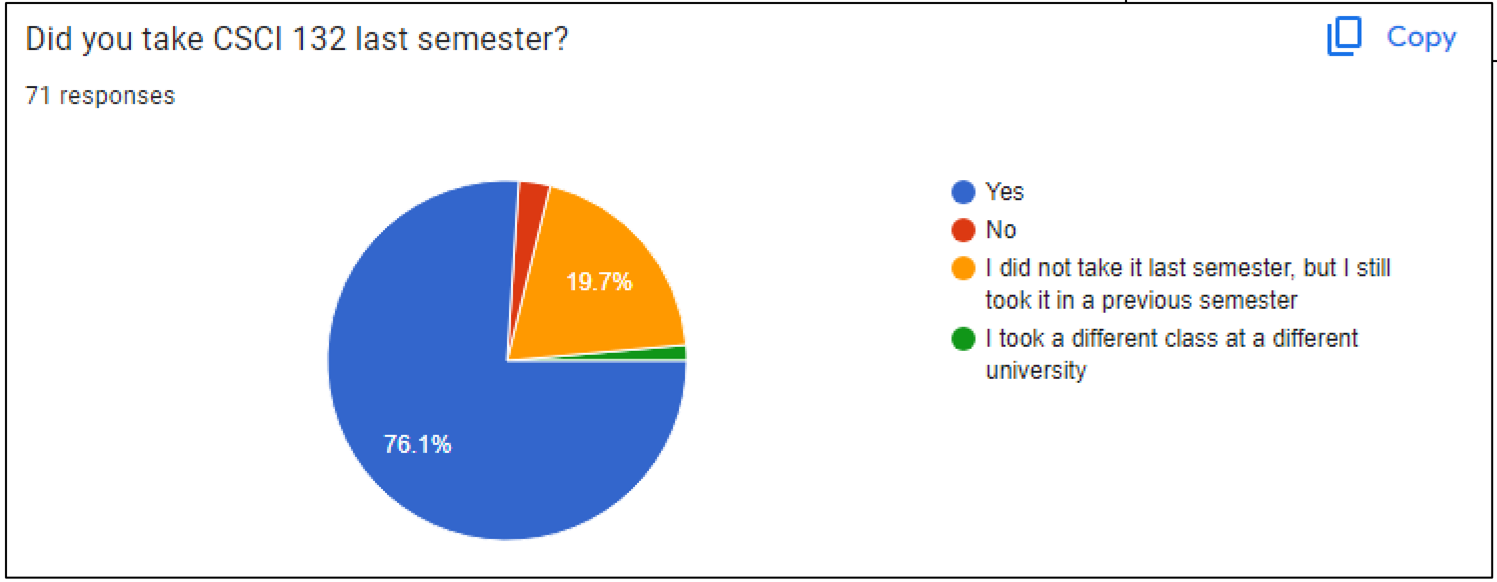
**Tutoring Schedule - Barnard Hall 259: Monday, January 22nd - Friday, May 3rd**

| Schedule | Monday | Tuesday | Wednesday | Thursday | Friday |
|---|---|---|---|---|---|
| 8:00 a.m. | | | | | |
| 9:00 a.m. | | | | Muzhou Chen | Kaden Bach |
| 10:00 a.m. | Sultan Yarylgassimov | Ruby Martin<br>Katie Harmon | | Muzhou Chen | Gerard Shu Fuhnwi |
| 11:00 a.m. | Sultan Yarylgassimov | Riley Slater | Jack Ruder | Nicholas Addotey<br>Ryan Johnson | Gerard Shu Fuhnwi |
| Noon | Asibul Islam<br>Shahnaj Mou | Riley Slater | Jack Ruder<br>Muhammad Bhatti | Nicholas Addotey | Jared Matury<br>Matthew Phillips |
| 1:10 p.m. | | Joshua Bowen | Muhammad Bhatti | | |
| 2:10 p.m. | Angelo Porcello<br>Gideon Popoola | Racquel Bowen<br>Muhammad Arju | Gideon Popoola | Nishu Nath | |
| 3:10 p.m. | Angelo Porcello<br>Brayden Miller | Muhammad Arju<br>Justin Mau | Shama Maganur<br>Fatima Ododo | Nishu Nath | |
| 4:10 p.m. | | Justin Mau | Shama Maganur<br>Fatima Ododo | | |
| 5:10 p.m. | Asibul Islam<br>Shahnaj Mou | | | | |

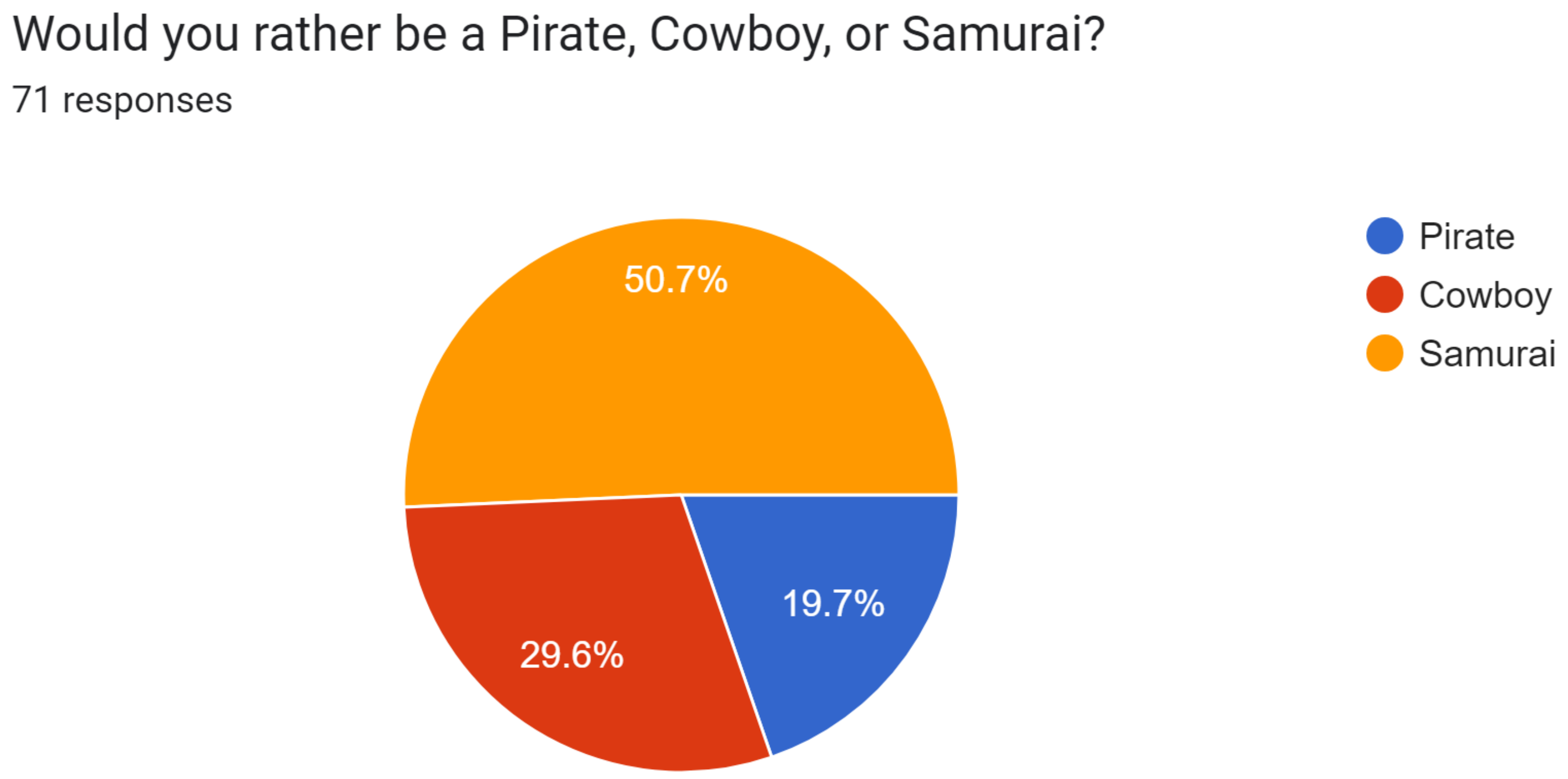
# Course Questionnaire Results



# Course Questionnaire Results



We are going to write a program where a user can keep track of their online shopping cart.

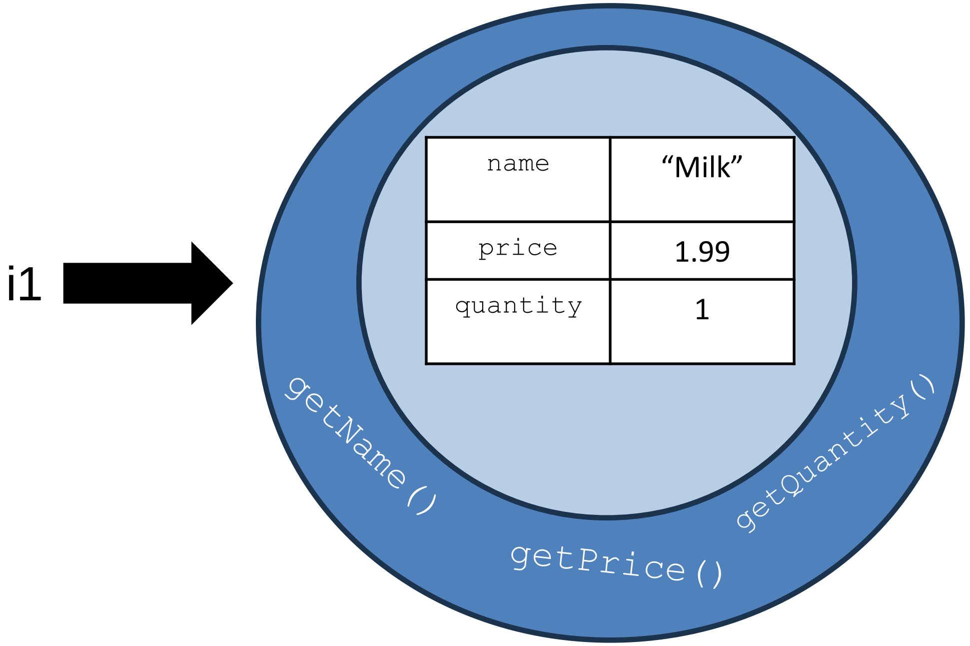
Users can add items, remove items, search for items, get the total price of cart, and apply coupons to items

```java
public class Item {


    private String name;
    private double price;
    private int quantity;

    public Item(String n, double p, int q) {
        this.name = n;
        this.price = p;
        this.quantity = q;
    }

    public String getName() {
        return this.name;
    }

    public double getPrice() {
        return this.price;
    }

    public int getQuantity() {
        return this.quantity;
    }
}
```

Java Class: Blueprint for an object (i.e. a “thing”)

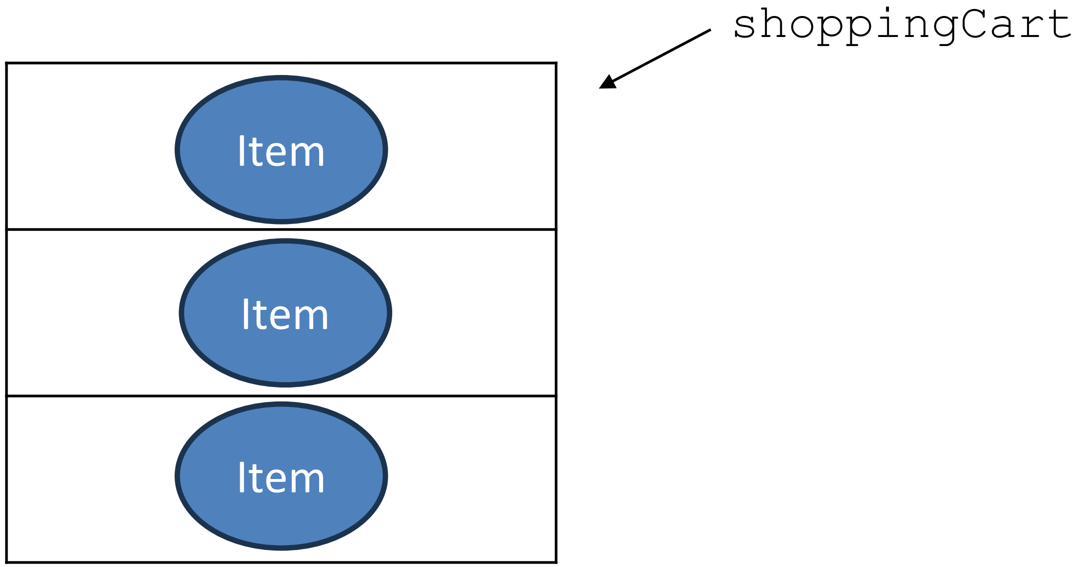
- Instance Field/Attributes
- Methods

```java
Item i1 = new Item("Milk", 1.99, 1);
Item i2 = new Item("Eggs", 3.99, 2);

System.out.println(i1.getName());
System.out.println(i2.getQuantity());
```

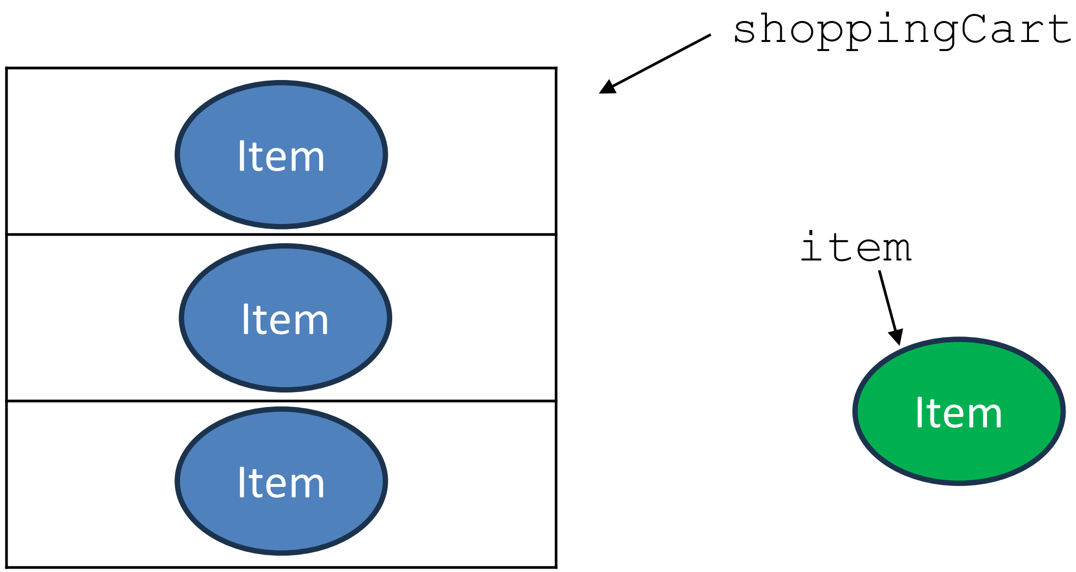



Java Objects: **Instances** of classes.
Program entities

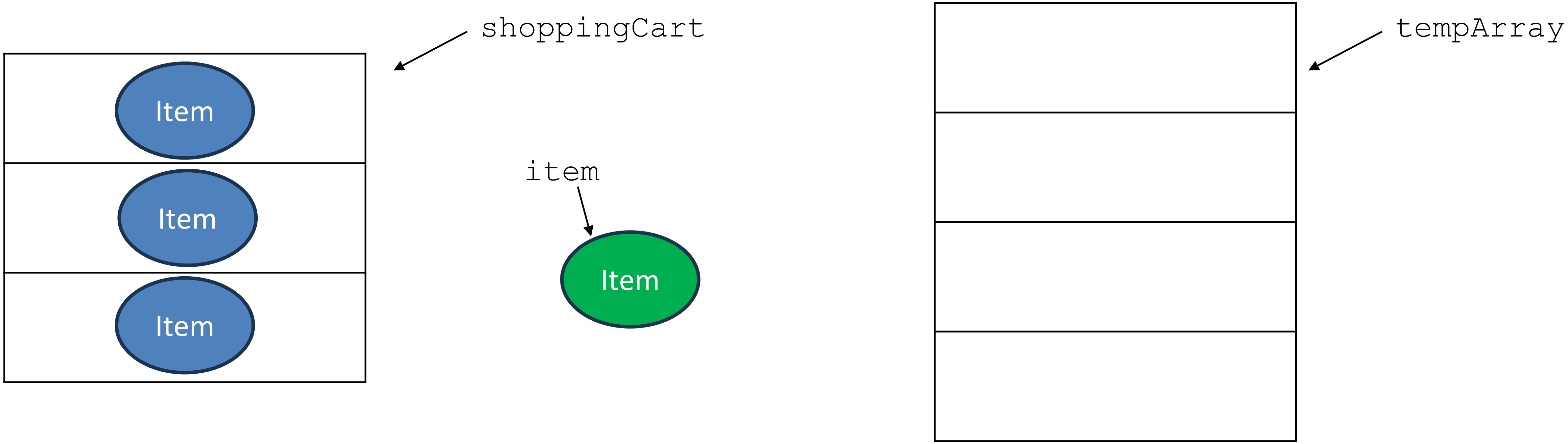
```java
public void addItem(String name, double price, int quantity) {
        Item item = new Item(name, price, quantity);
        Item[] tempArray = new Item[this.shoppingCart.length + 1];
        for(int i = 0; i < this.shoppingCart.length; i++) {
                tempArray[i] = shoppingCart[i];
        }
        tempArray[shoppingCart.length] = item;
        shoppingCart = tempArray;
        this.num_of_items++;
}
```



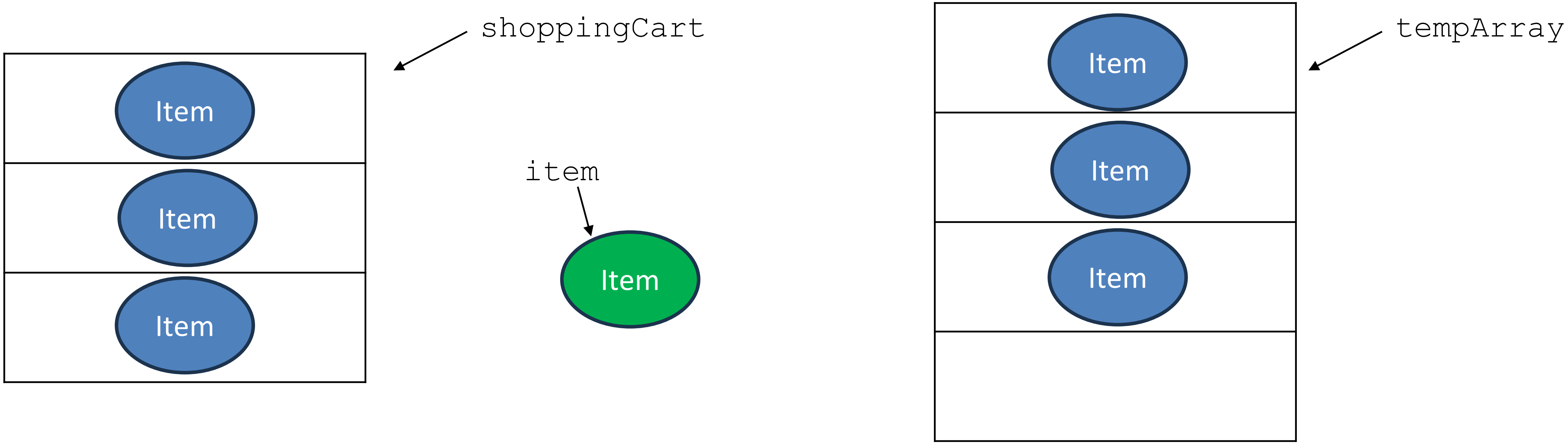
```java
public void addItem(String name, double price, int quantity) {
        Item item = new Item(name, price, quantity);
        Item[] tempArray = new Item[this.shoppingCart.length + 1];
        for(int i = 0; i < this.shoppingCart.length; i++) {
                tempArray[i] = shoppingCart[i];
        }
        tempArray[shoppingCart.length] = item;
        shoppingCart = tempArray;
        this.num_of_items++;
}
```



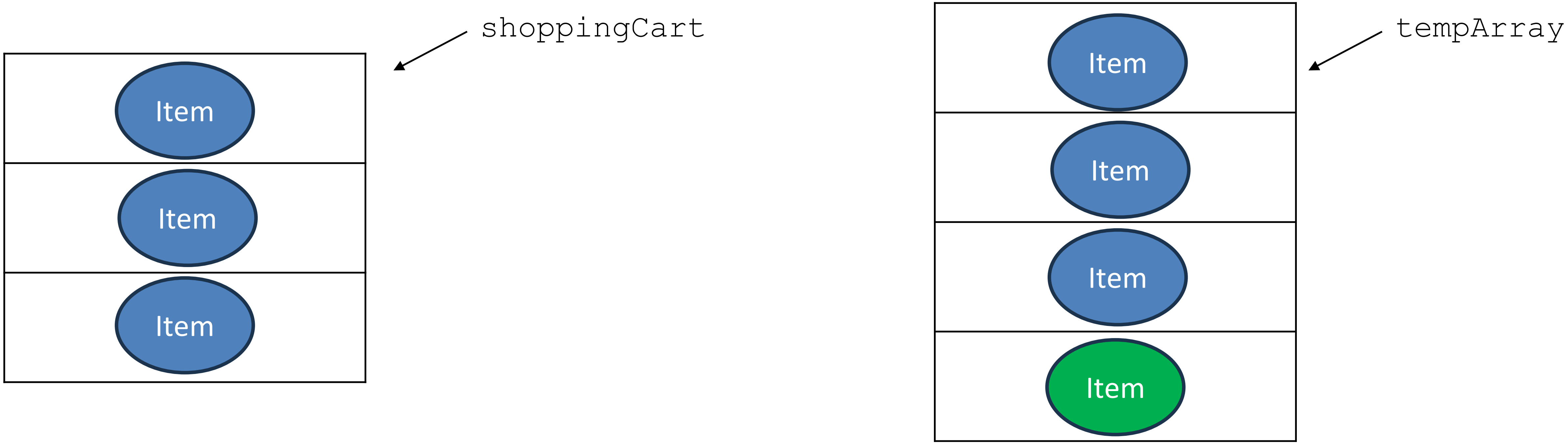
```java
public void addItem(String name, double price, int quantity) {
        Item item = new Item(name, price, quantity);
        Item[] tempArray = new Item[this.shoppingCart.length + 1];
        for(int i = 0; i < this.shoppingCart.length; i++) {
                tempArray[i] = shoppingCart[i];
        }
        tempArray[shoppingCart.length] = item;
        shoppingCart = tempArray;
        this.num_of_items++;
}
```



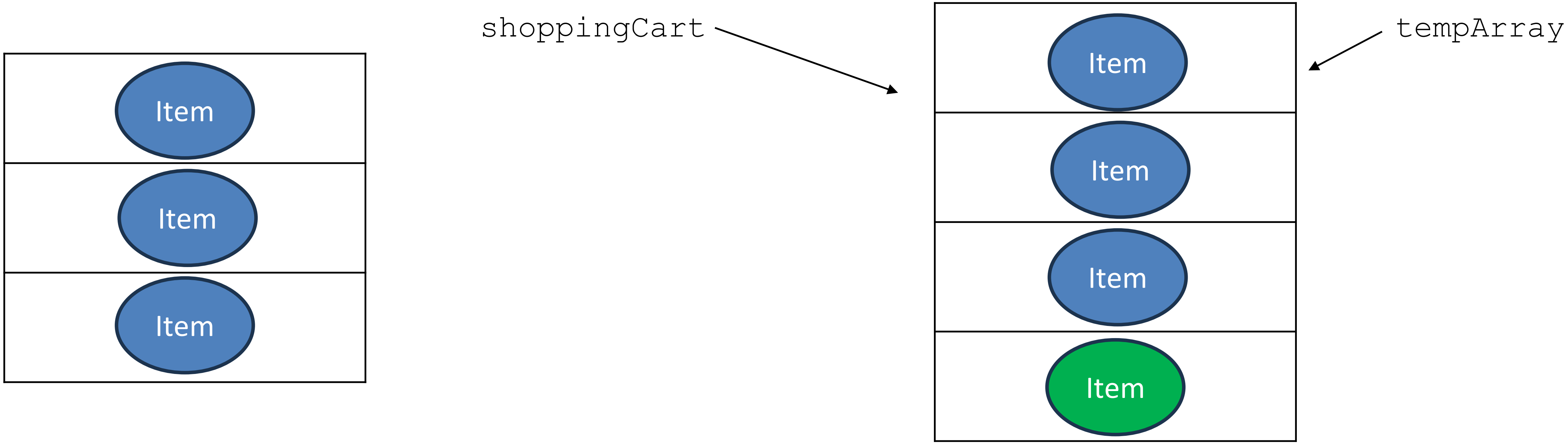
```
public void addItem(String name, double price, int quantity) {
        Item item = new Item(name, price, quantity);
        Item[] tempArray = new Item[this.shoppingCart.length + 1];
        for(int i = 0; i < this.shoppingCart.length; i++) {
                tempArray[i] = shoppingCart[i];
        }
        tempArray[shoppingCart.length] = item;
        shoppingCart = tempArray;
        this.num_of_items++;
}
```



```java
public void addItem(String name, double price, int quantity) {
        Item item = new Item(name, price, quantity);
        Item[] tempArray = new Item[this.shoppingCart.length + 1];
        for(int i = 0; i < this.shoppingCart.length; i++) {
                tempArray[i] = shoppingCart[i];
        }
        tempArray[shoppingCart.length] = item;
        shoppingCart = tempArray;
        this.num_of_items++;
}
```



```java
public void addItem(String name, double price, int quantity) {
       Item item = new Item(name, price, quantity);
       Item[] tempArray = new Item[this.shoppingCart.length + 1];
       for(int i = 0; i < this.shoppingCart.length; i++) {
              tempArray[i] = shoppingCart[i];
       }
       tempArray[shoppingCart.length] = item;
       shoppingCart = tempArray;
       this.num_of_items++;
}
```



```java
public void addItem(String name, double price, int quantity) {
        Item item = new Item(name, price, quantity);
        Item[] tempArray = new Item[this.shoppingCart.length + 1];
        for(int i = 0; i < this.shoppingCart.length; i++) {
                tempArray[i] = shoppingCart[i];
        }
        tempArray[shoppingCart.length] = item;
        shoppingCart = tempArray;
        this.num_of_items++;
}
```

Running time?

```java
public void addItem(String name, double price, int quantity) {
        Item item = new Item(name, price, quantity);
        Item[] tempArray = new Item[this.shoppingCart.length + 1];
        for(int i = 0; i < this.shoppingCart.length; i++) {
                tempArray[i] = shoppingCart[i];
        }
        tempArray[shoppingCart.length] = item;
        shoppingCart = tempArray;
        this.num_of_items++;
}
```

Running time: Number of operations required to complete algorithm

```java
public void addItem(String name, double price, int quantity) {
        Item item = new Item(name, price, quantity);
        Item[] tempArray = new Item[this.shoppingCart.length + 1];
        for(int i = 0; i < this.shoppingCart.length; i++) {
                tempArray[i] = shoppingCart[i];
        }
        tempArray[shoppingCart.length] = item;
        shoppingCart = tempArray;
        this.num_of_items++;
}
```

Running time: Number of operations required to complete algorithm

Big O Notation: Upper bound on asymptotic growth. I.e. Worst case upper bound of a function

```java
public void addItem(String name, double price, int quantity) {
        Item item = new Item(name, price, quantity);
        Item[] tempArray = new Item[this.shoppingCart.length + 1];
        for(int i = 0; i < this.shoppingCart.length; i++) {
                tempArray[i] = shoppingCart[i];
        }
        tempArray[shoppingCart.length] = item;
        shoppingCart = tempArray;
        this.num_of_items++;
}
```

Running time: Number of operations required to complete algorithm

Big O Notation: Upper bound on asymptotic growth. I.e. Worst case upper bound of a function

Big O Notation measures the number of steps needed to complete an algorithm under the worst-case scenario

```java
public int linearSearch(int[] array, int target) {
        for(int i = 0; i < array.length; i++) {
                if(array[i] == target){
                        return i;
                }
        }
        return -1;
}
```

To calculate the running time, we add up the running time of each operation

```
public int linearSearch(int[] array, int target) {
 ??? → for(int i = 0; i < array.length; i++) {
           if(array[i] == target){
                  return i;
           }
      }
      return -1;
}
```

To calculate the running time, we add up the running time of each operation

Worst case scenario, this for loop will need run **n** times

**O(n)**    **Let n = array.length**

```
public int linearSearch(int[] array, int target) {
  O(n) →for(int i = 0; i < array.length; i++) {
        O(???) → if(array[i] == target){
                        return i;
                }
        }
        return -1;
}
```

To calculate the running time, we add up the running time of each operation

```
public int linearSearch(int[] array, int target) {
  O(n) →for(int i = 0; i < array.length; i++) {
       O(???) → if(array[i] == target){
                     return i;
                }
        }
        return -1;
}
```

To calculate the running time, we add up the running time of each operation

Primitive operation – operation that takes constant time (independent of size of the input)

```
public int linearSearch(int[] array, int target) {
 O(n) →for(int i = 0; i < array.length; i++) {
        O(1) → if(array[i] == target){
                    return i;
                }
        }
        return -1;
}
```

To calculate the running time, we add up the running time of each operation

Primitive operation – operation that takes constant time (independent of size of the input)

```
public int linearSearch(int[] array, int target) {
  O(n) →for(int i = 0; i < array.length; i++) {
         O(1) → if(array[i] == target){
                O(1) →  return i;
                }
         }
  O(1) →return -1;
}
```

To calculate the running time, we add up the running time of each operation

Primitive operation – operation that takes constant time (independent of size of the input)

```
public int linearSearch(int[] array, int target) {
  O(n) →for(int i = 0; i < array.length; i++) {
        O(1) → if(array[i] == target){
              O(1) → return i;
              }
        }
  O(1) →return -1;
}
```

To calculate the running time, we add up the running time of each operation

Primitive operation – operation that takes constant time (independent of size of the input)

**Total running time: O(n * 1 + 1)**

```
public int linearSearch(int[] array, int target) {
  O(n) →for(int i = 0; i < array.length; i++) {
        O(1) → if(array[i] == target){
                O(1) → return i;
               }
        }
  O(1) →return -1;
}
```

To calculate the running time, we add up the running time of each operation

Primitive operation – operation that takes constant time (independent of size of the input)

**Total running time: O(n * 1 + 1)**

In Big O notation:

- We can drop non dominant factors
- We can drop multiplicative constants (coefficients)

```
public int linearSearch(int[] array, int target) {
  O(n) →for(int i = 0; i < array.length; i++) {
         O(1) → if(array[i] == target){
                O(1) → return i;
                }
        }
  O(1) →return -1;
}
```

To calculate the running time, we add up the running time of each operation

Primitive operation – operation that takes constant time (independent of size of the input)

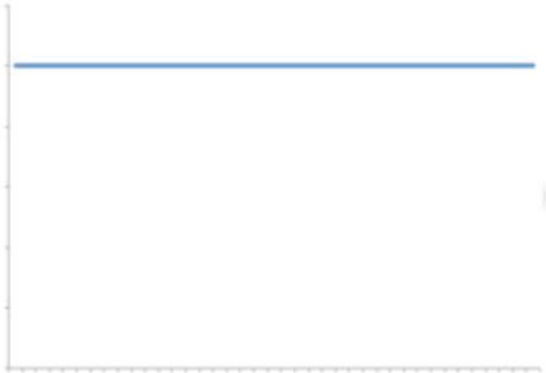
**Total running time: O(n) where n = | array |**

In Big O notation:

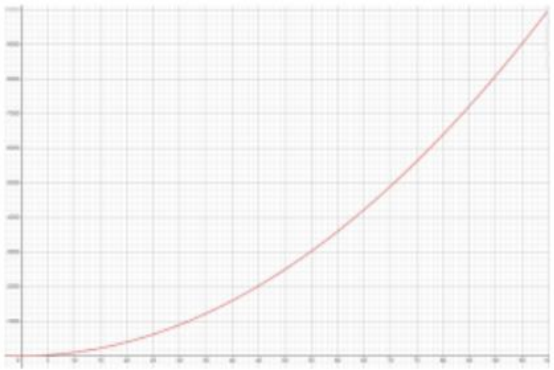
- We can drop non dominant factors
- We can drop multiplicative constants (coefficients)

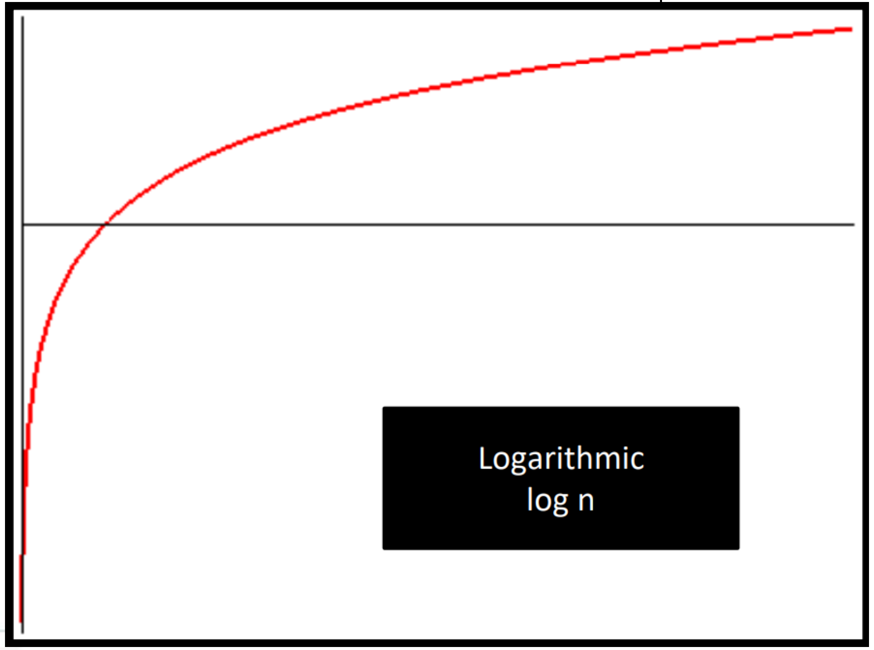
```
{

+ 1];
```

**Constant**

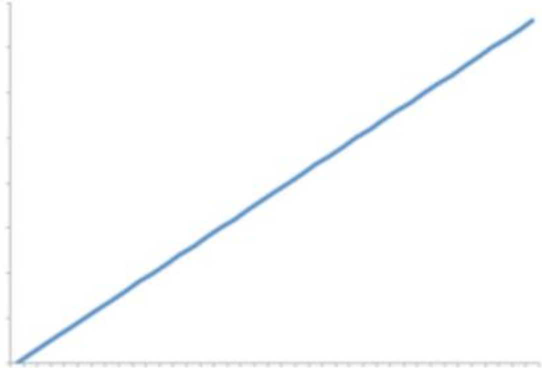


**Quadratic**

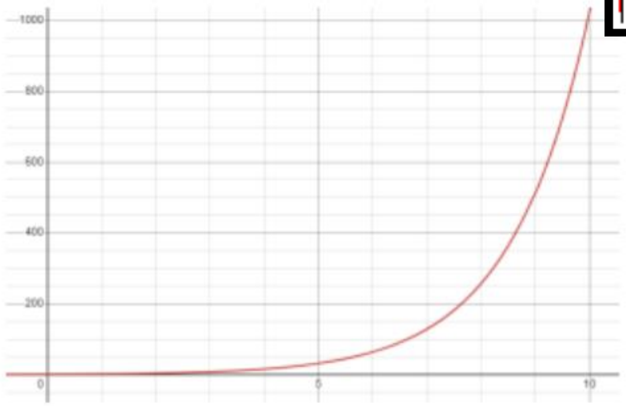





**Linear**



**Exponential**



```
function computeDistanceBetweenHouses():

      for each house in neighborhood i;
            for each house in neighborhood j;
                  compute_distance(i, j)
```

| | **H1** | **H2** | **H3** | **…** | **H9** |
|---|---|---|---|---|---|
| H1 | / | D(1,2) | D(1,3) | … | D(1,9) |
| H2 | D(2,1) | / | D(2,3) | … | D(2,9) |
| H3 | D(3,1) | D(3,2) | / | … | D(3,9) |
| … | … | … | … | … | …. |
| H9 | D(9,1) | D(9,2) | D(9,3) | … | / |

```
function computeDistanceBetweenHouses():

O(n)  for each house in neighborhood i;
      O(n-1) for each house in neighborhood j;
             O(1) compute_distance(i, j)
```

| | H1 | H2 | H3 | … | H9 |
|---|---|---|---|---|---|
| H1 | / | D(1,2) | D(1,3) | … | D(1,9) |
| H2 | D(2,1) | / | D(2,3) | … | D(2,9) |
| H3 | D(3,1) | D(3,2) | / | … | D(3,9) |
| … | … | … | … | … | …. |
| H9 | D(9,1) | D(9,2) | D(9,3) | … | / |

```
function computeDistanceBetweenHouses():

O(n) for each house in neighborhood i;
       O(n) for each house in neighborhood j;
              O(1) compute_distance(i, j)
```

| | H1 | H2 | H3 | … | H9 |
|---|---|---|---|---|---|
| H1 | / | D(1,2) | D(1,3) | … | D(1,9) |
| H2 | D(2,1) | / | D(2,3) | … | D(2,9) |
| H3 | D(3,1) | D(3,2) | / | … | D(3,9) |
| … | … | … | … | … | …. |
| H9 | D(9,1) | D(9,2) | D(9,3) | … | / |

```
function computeDistanceBetweenHouses():

O(n) for each house in neighborhood i;
        O(n) for each house in neighborhood j;
                O(1) compute_distance(i, j)
```

| | H1 | H2 | H3 | ... | H9 |
|---|---|---|---|---|---|
| H1 | / | D(1,2) | D(1,3) | ... | D(1,9) |
| H2 | D(2,1) | / | D(2,3) | ... | D(2,9) |
| H3 | D(3,1) | D(3,2) | / | ... | D(3,9) |
| ... | ... | ... | ... | ... | .... |
| H9 | D(9,1) | D(9,2) | D(9,3) | ... | / |

Total running time = O(n) * ( O(n) * O(1) )

```
function computeDistanceBetweenHouses():

O(n) for each house in neighborhood i;
      O(n) for each house in neighborhood j;
            O(1) compute_distance(i, j)
```

| | H1 | H2 | H3 | ... | H9 |
|---|---|---|---|---|---|
| H1 | / | D(1,2) | D(1,3) | ... | D(1,9) |
| H2 | D(2,1) | / | D(2,3) | ... | D(2,9) |
| H3 | D(3,1) | D(3,2) | / | ... | D(3,9) |
| ... | ... | ... | ... | ... | .... |
| H9 | D(9,1) | D(9,2) | D(9,3) | ... | / |

Total running time = O(n) * ( O(n) * O(1) )

**O(n^2)** **Where n = # of houses**

```java
public void addItem(String name, double price, int quantity) {
	Item item = new Item(name, price, quantity);
	Item[] tempArray = new Item[this.shoppingCart.length + 1];
	for(int i = 0; i < this.shoppingCart.length; i++) {
		tempArray[i] = shoppingCart[i];
	}
	tempArray[shoppingCart.length] = item;
	shoppingCart = tempArray;
	this.num_of_items++;
}
```

```java
public void addItem(String name, double price, int quantity) {
  O(1) → Item item = new Item(name, price, quantity);
        Item[] tempArray = new Item[this.shoppingCart.length + 1];
        for(int i = 0; i < this.shoppingCart.length; i++) {
                tempArray[i] = shoppingCart[i];
        }
        tempArray[shoppingCart.length] = item;
        shoppingCart = tempArray;
        this.num_of_items++;
}
```

```
public void addItem(String name, double price, int quantity) {
  O(1) → Item item = new Item(name, price, quantity);
O(n+1) → Item[] tempArray = new Item[this.shoppingCart.length + 1];
         for(int i = 0; i < this.shoppingCart.length; i++) {
                 tempArray[i] = shoppingCart[i];
         }
         tempArray[shoppingCart.length] = item;
         shoppingCart = tempArray;
         this.num_of_items++;
}
```

```
public void addItem(String name, double price, int quantity) {
  O(1) → Item item = new Item(name, price, quantity);
O(n+1) → Item[] tempArray = new Item[this.shoppingCart.length + 1];
  O(n) → for(int i = 0; i < this.shoppingCart.length; i++) {
                 tempArray[i] = shoppingCart[i];
         }
         tempArray[shoppingCart.length] = item;
         shoppingCart = tempArray;
         this.num_of_items++;
}
```

```java
public void addItem(String name, double price, int quantity) {
   O(1) → Item item = new Item(name, price, quantity);
 O(n+1) → Item[] tempArray = new Item[this.shoppingCart.length + 1];
   O(n) → for(int i = 0; i < this.shoppingCart.length; i++) {
              O(1) → tempArray[i] = shoppingCart[i];
          }
          tempArray[shoppingCart.length] = item;
          shoppingCart = tempArray;
          this.num_of_items++;
}
```

```
public void addItem(String name, double price, int quantity) {
  O(1) → Item item = new Item(name, price, quantity);
O(n+1) → Item[] tempArray = new Item[this.shoppingCart.length + 1];
  O(n) → for(int i = 0; i < this.shoppingCart.length; i++) {
           O(1) → tempArray[i] = shoppingCart[i];
         }
  O(1) → tempArray[shoppingCart.length] = item;
  O(1) → shoppingCart = tempArray;
  O(1) → this.num_of_items++;
}
```

```
public void addItem(String name, double price, int quantity) {
  O(1) → Item item = new Item(name, price, quantity);
O(n+1) → Item[] tempArray = new Item[this.shoppingCart.length + 1];
  O(n) → for(int i = 0; i < this.shoppingCart.length; i++) {
            O(1) → tempArray[i] = shoppingCart[i];
         }
  O(1) → tempArray[shoppingCart.length] = item;
  O(1) → shoppingCart = tempArray;
  O(1) → this.num_of_items++;
}
```

```
public void addItem(String name, double price, int quantity) {
  O(1) → Item item = new Item(name, price, quantity);
  O(n) → Item[] tempArray = new Item[this.shoppingCart.length + 1];
  O(n) → for(int i = 0; i < this.shoppingCart.length; i++) {
           O(1) → tempArray[i] = shoppingCart[i];
         }
  O(1) → tempArray[shoppingCart.length] = item;
  O(1) → shoppingCart = tempArray;
  O(1) → this.num_of_items++;
}
```

Total running time: O(n) + O(n)

O(2n)

**O(n) where n = shoppingCart.length**

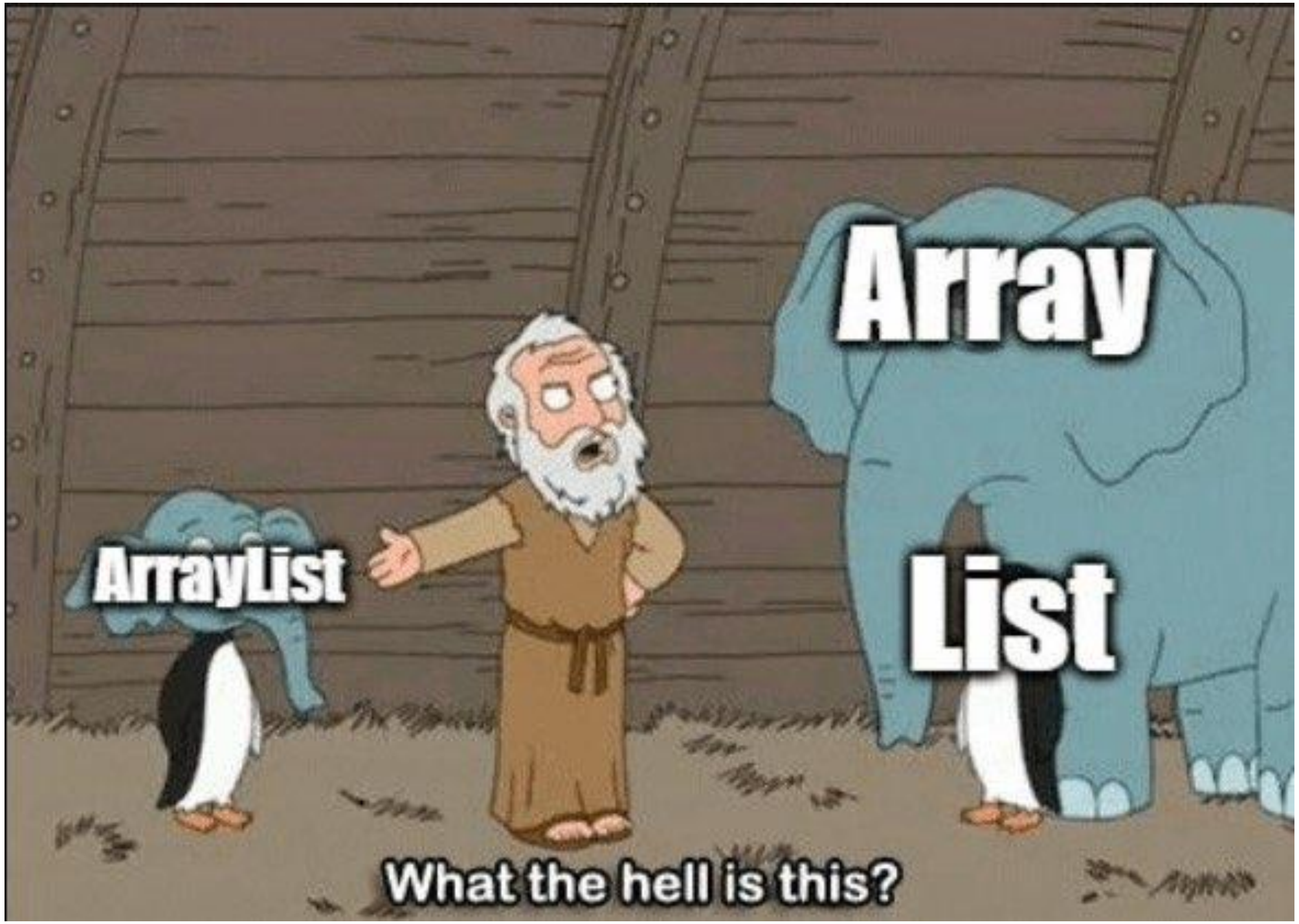
```
public void addItem(String name, double price, int quantity) {
  O(1) → Item item = new Item(name, price, quantity);
 O(n) → Item[] tempArray = new Item[this.shoppingCart.length + 1];
  O(n) → for(int i = 0; i < this.shoppingCart.length; i++) {
           O(1) → tempArray[i] = shoppingCart[i];
         }
  O(1) → tempArray[shoppingCart.length] = item;
  O(1) → shoppingCart = tempArray;
  O(1) → this.num_of_items++;
}
```

Total running time: O(n) + O(n)

O(2n)

**O(n) where n = shoppingCart.length**

Takeaway: Adding to a full array takes O(n) time


Array
ArrayList
List
What the hell is this?