



Classes and Objects

Part 04

Big Example

Problem



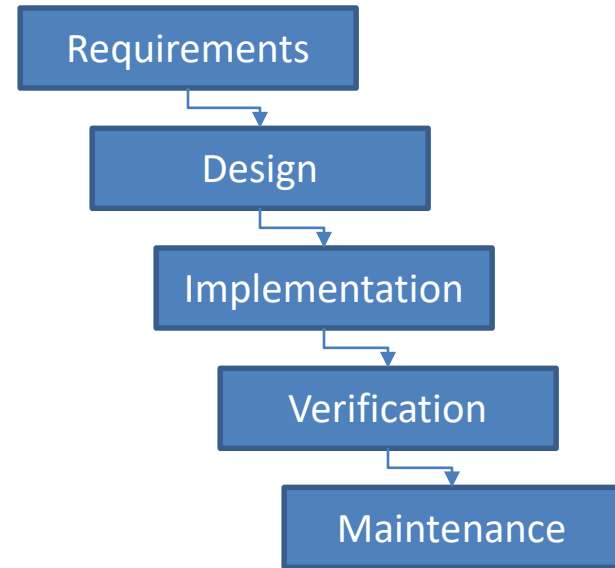
Problem



Problem

- Problem to Solve
 - Keep track of Tacos that I like
- Create Solutions to Problems
- Waterfall Model
 - Requirements
 - Design
 - Implement
 - Verification
 - Maintenance

Waterfall Model



Requirements

- Keep Track of important Taco Information
- Taco's Information
 - Name
 - Location
 - Price

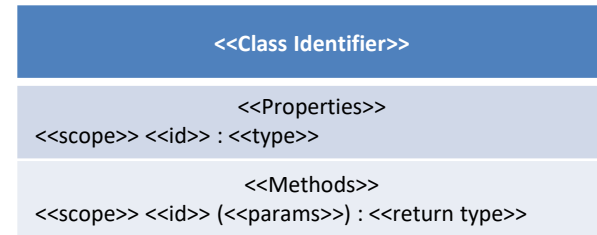


- Should be able to
 - Add a Taco
 - Remove a Taco by Name
 - Sort by Price
 - Display all Taco information
- Clear and Simple Front End

Design

- Separate Front End from Back End
- UML Class Diagram
 - Boxes are Structures (like Classes)
 - Arrows are relationships between structures
- Classes
 - Name of the class
 - Properties
 - Methods
 - “+” / “-” means scope is public or private
- Arrows
 - Stick arrow is the Association or “has a”
 - Numeric values indicate the number of instances
- Static variables and method are underlined
- Constants are all UPPER CASE

UML Class Diagram

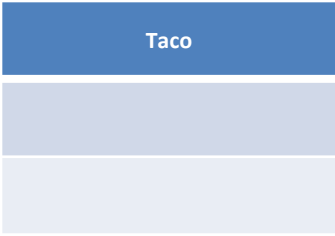
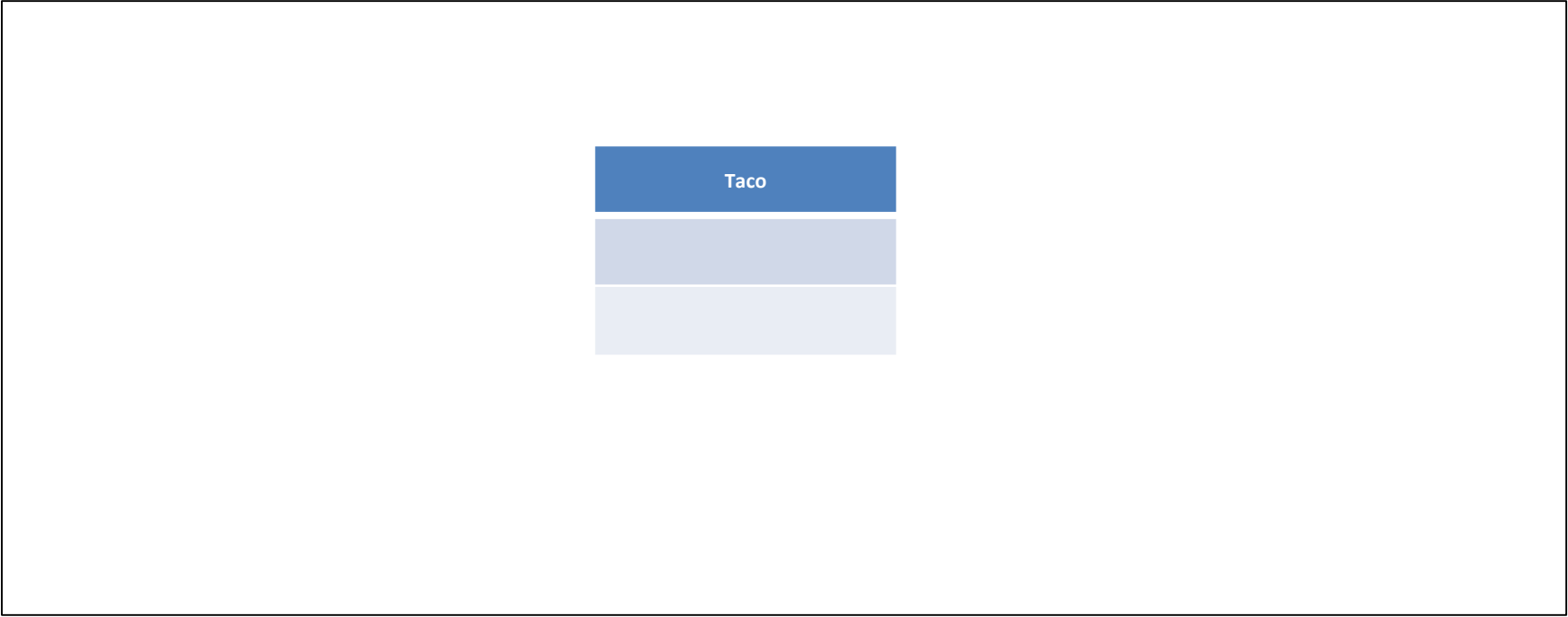


UML Class Diagram Arrows

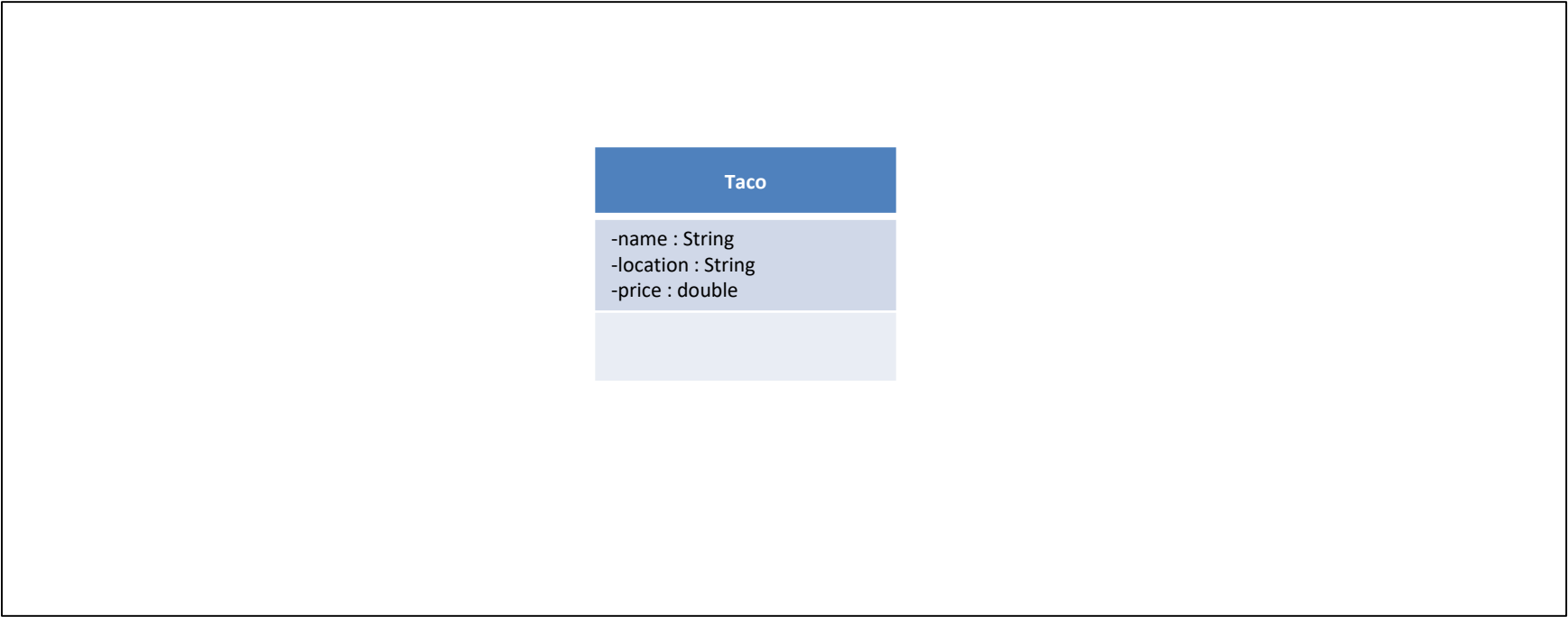


Association
“has a”

Design



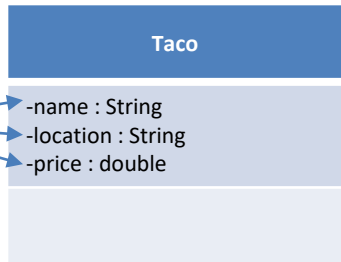
Design



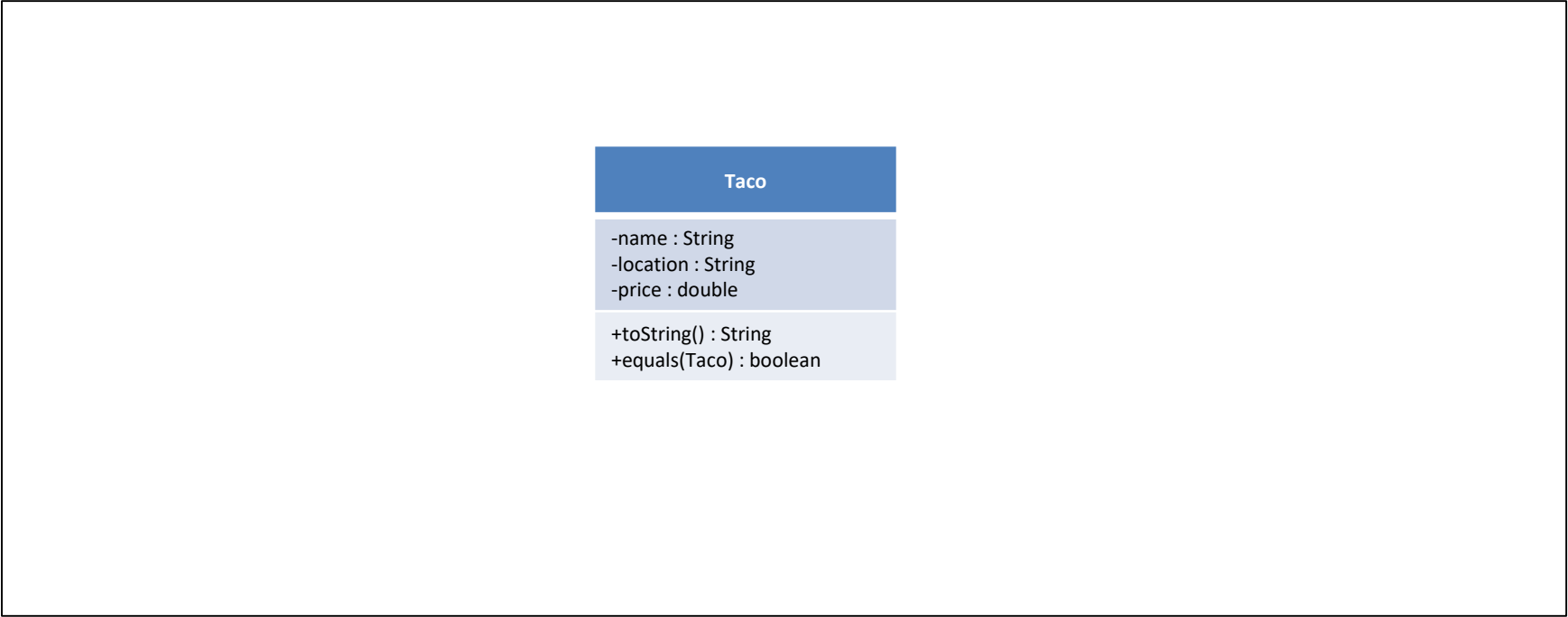
Taco
-name : String
-location : String
-price : double

Design

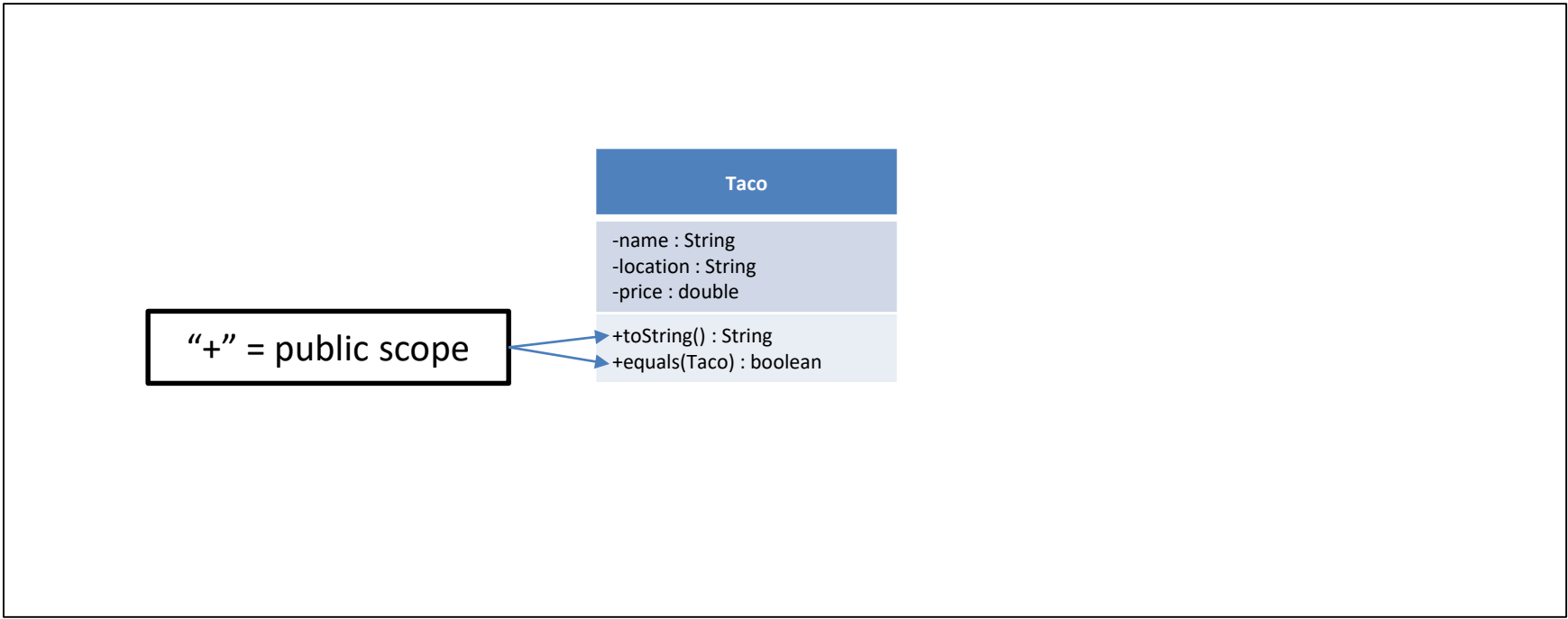
“-” = private scope



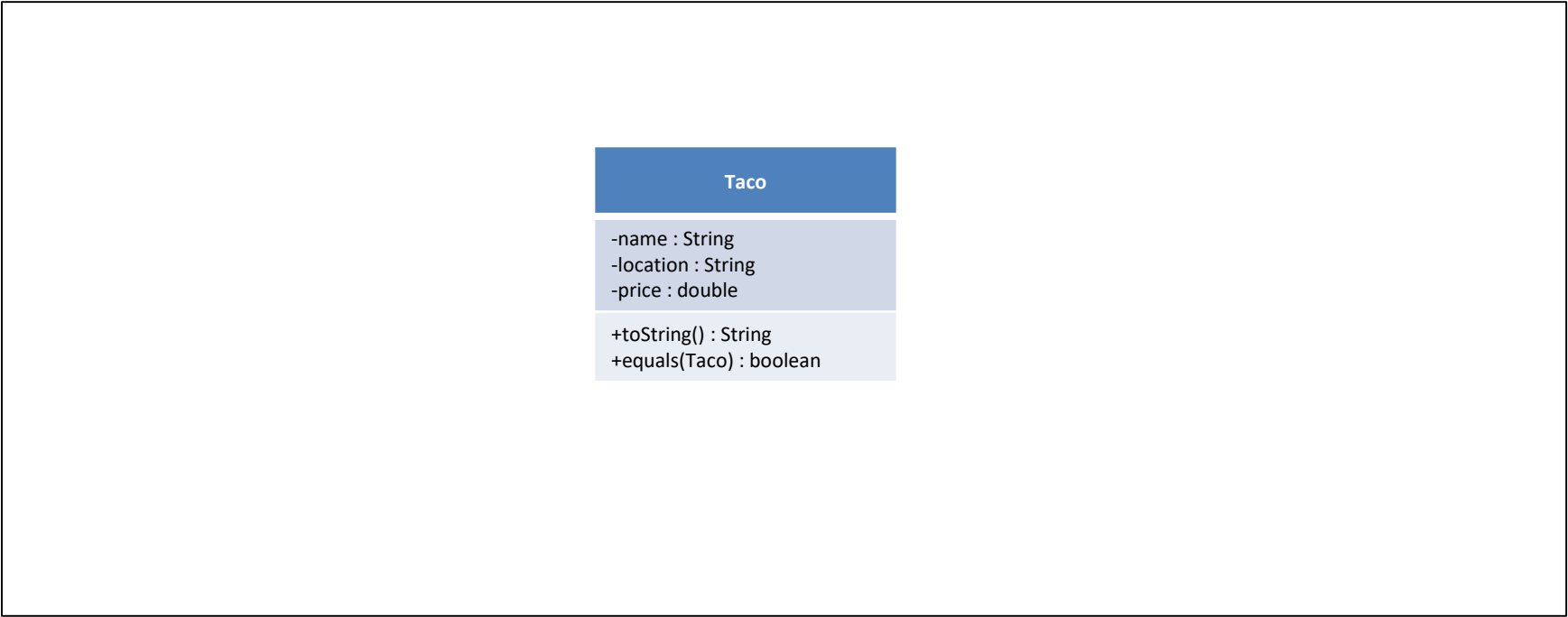
Design



Design



Design



Design

Taco
-name : String -location : String -price : double
+toString() : String +equals(Taco) : boolean

TacoManager

Design

Taco
-name : String -location : String -price : double
+toString() : String +equals(Taco) : boolean

TacoManager
-tacos: Taco[] <u>+DEF_SIZE: int</u>

Design

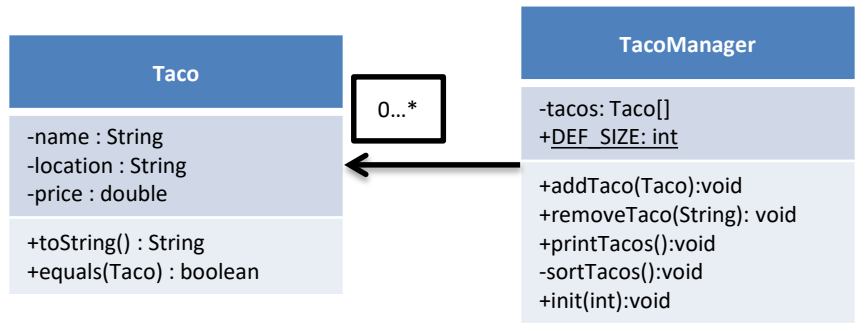
Taco

```
-name : String  
-location : String  
-price : double  
  
+toString() : String  
+equals(Taco) : boolean
```

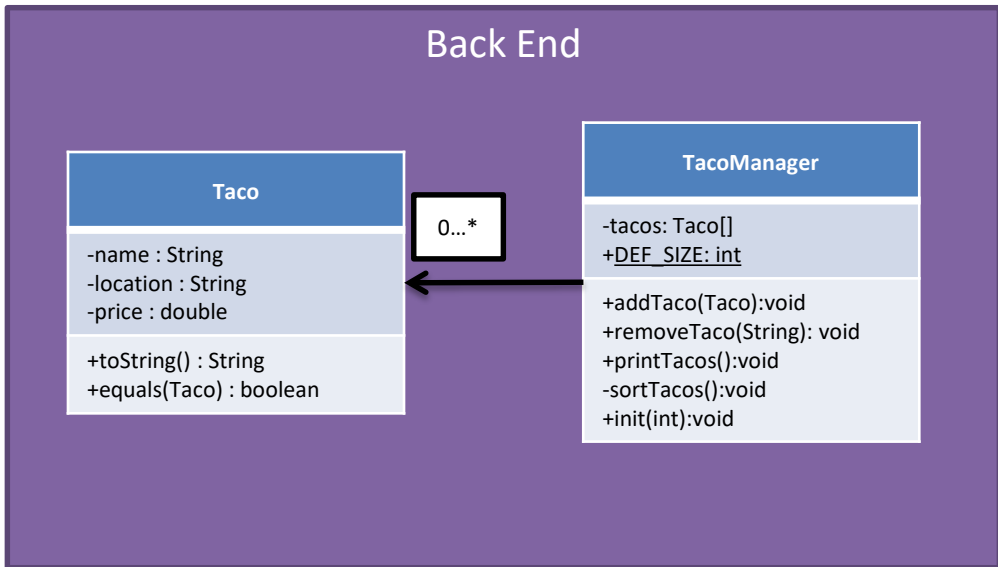
TacoManager

```
-tacos: Taco[]  
+DEF_SIZE: int  
  
+addTaco(Taco):void  
+removeTaco(String): void  
+printTacos():void  
-sortTacos():void  
+init(int):void
```

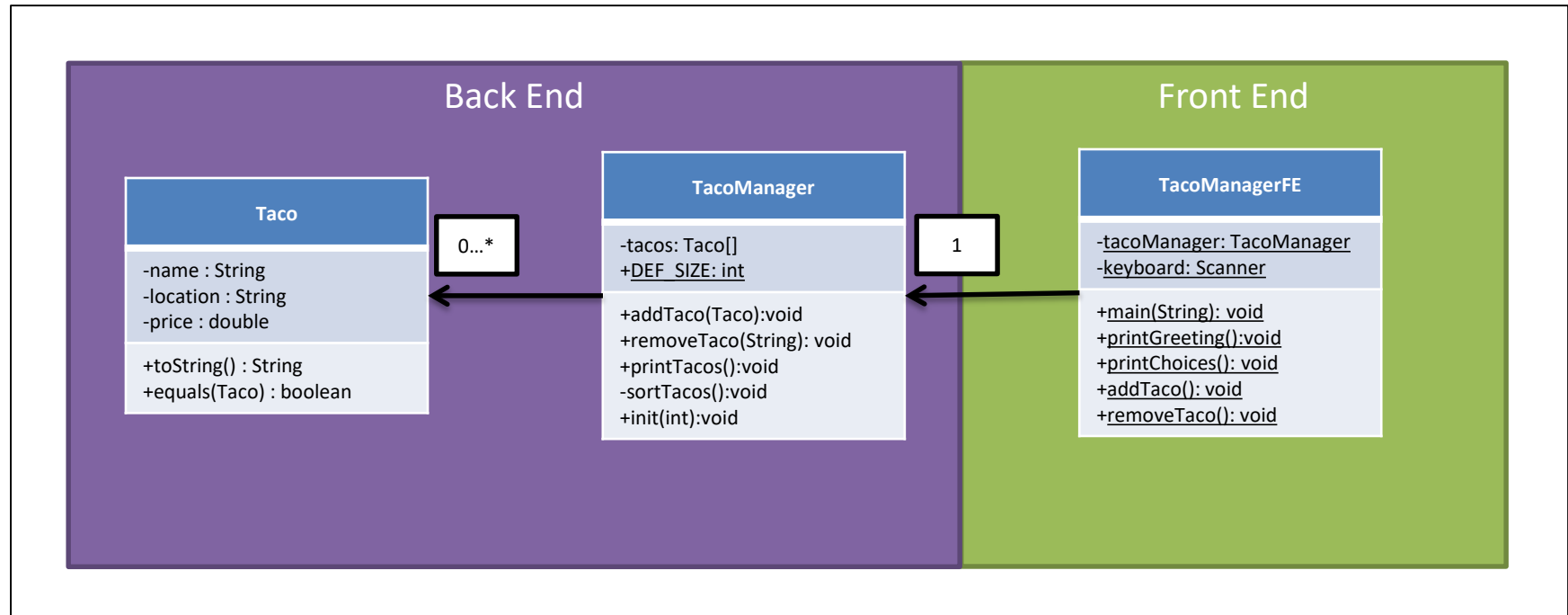
Design



Design



Design




Implementation

Taco Array Structure

- Arrays of Objects are Arrays of Memory Addresses
- Arrays are considered Object Types in Java
- The Array's identifier points to the contents of the array
- The Array's indices point to the contents of the constructed Objects
- Default values for Object Arrays are considered NULL

Identifier	Contents	Byte Address
...
tacos	40	28
...
tacos[0]	NULL	40
tacos[1]	NULL	46
tacos[2]	NULL	52
...
tacos[8]	NULL	88
tacos[9]	NULL	94




Taco Array Structure

Identifier	Contents	Byte Address
...
tacos	NULL	28
...
tacos[0]	NULL	40
tacos[1]	NULL	46
tacos[2]	NULL	52
...
tacos[8]	NULL	88
tacos[9]	NULL	94

Taco Array Structure


Identifier	Contents	Byte Address
...
tacos	40	28
...
tacos[0]	NULL	40
tacos[1]	NULL	46
tacos[2]	NULL	52
...
tacos[8]	NULL	88
tacos[9]	NULL	94



Taco Array Structure

- Keeping constructed Objects to one side
- No NULL elements in between constructed Objects
- First NULL Element means everything after that is also assumed NULL
- Adding
 - Start from the first Index
 - Find first null element
 - Assign value to there
- Removing
 - Start from the first Index
 - Find the element to remove's index
 - If not found then return
 - Then shift over by one ($tacos[i] = tacos[i+1]$)
 - Set last element to NULL

Identifier	Contents	Byte Address
...
tacos	40	28
...
tacos[0]	NULL	40
tacos[1]	NULL	46
tacos[2]	NULL	52
...
tacos[8]	NULL	88
tacos[9]	NULL	94



Taco Array Structure

- Keeping constructed Objects to one side
- No NULL elements in between constructed Objects
- First NULL Element means everything after that is also assumed NULL
- Adding
 - Start from the first Index
 - Find first null element
 - Assign value to there
- Removing
 - Start from the first Index
 - Find the element to remove's index
 - If not found then return
 - Then shift over by one ($tacos[i] = tacos[i+1]$)
 - Set last element to NULL



Identifier	Contents	Byte Address
...
tacos	40	28
...
tacos[0]	NULL	40
tacos[1]	NULL	46
tacos[2]	NULL	52
...
tacos[8]	NULL	88
tacos[9]	NULL	94

Taco Array Structure


- Keeping constructed Objects to one side
- No NULL elements in between constructed Objects
- First NULL Element means everything after that is also assumed NULL
- Adding
 - Start from the first Index
 - Find first null element
 - Assign value to there
- Removing
 - Start from the first Index
 - Find the element to remove's index
 - If not found then return
 - Then shift over by one ($tacos[i] = tacos[i+1]$)
 - Set last element to NULL



Identifier	Contents	Byte Address
...
tacos	40	28
...
tacos[0]	86	40
tacos[1]	NULL	46
tacos[2]	NULL	52
...
tacos[8]	NULL	88
tacos[9]	NULL	94

Taco Array Structure

- Keeping constructed Objects to one side
- No NULL elements in between constructed Objects
- First NULL Element means everything after that is also assumed NULL
- Adding
 - Start from the first Index
 - Find first null element
 - Assign value to there
- Removing
 - Start from the first Index
 - Find the element to remove's index
 - If not found then return
 - Then shift over by one ($tacos[i] = tacos[i+1]$)
 - Set last element to NULL



Identifier	Contents	Byte Address
...
tacos	40	28
...
tacos[0]	86	40
tacos[1]	283	46
tacos[2]	NULL	52
...
tacos[8]	NULL	88
tacos[9]	NULL	94

A red curved arrow points from the 'Contents' cell '40' in the 'tacos' row to the 'Byte Address' cell '40' in the 'tacos[0]' row.

Taco Array Structure

- Keeping constructed Objects to one side
- No NULL elements in between constructed Objects
- First NULL Element means everything after that is also assumed NULL
- Adding
 - Start from the first Index
 - Find first null element
 - Assign value to there
- Removing
 - Start from the first Index
 - Find the element to remove's index
 - If not found then return
 - Then shift over by one ($tacos[i] = tacos[i+1]$)
 - Set last element to NULL




Identifier	Contents	Byte Address
...
tacos	40	28
...
tacos[0]	86	40
tacos[1]	283	46
tacos[2]	128	52
...
tacos[8]	NULL	88
tacos[9]	NULL	94

Taco Array Structure

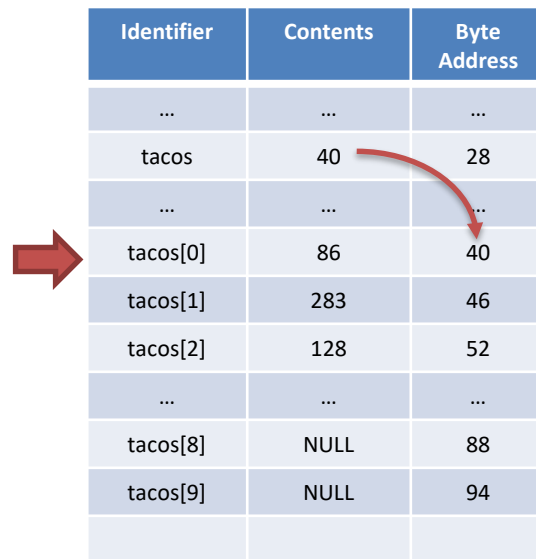
- Keeping constructed Objects to one side
- No NULL elements in between constructed Objects
- First NULL Element means everything after that is also assumed NULL
- Adding
 - Start from the first Index
 - Find first null element
 - Assign value to there
- Removing
 - Start from the first Index
 - Find the element to remove's index
 - If not found then return
 - Then shift over by one ($tacos[i] = tacos[i+1]$)
 - Set last element to NULL

Identifier	Contents	Byte Address
...
tacos	40	28
...
tacos[0]	86	40
tacos[1]	283	46
tacos[2]	128	52
...
tacos[8]	NULL	88
tacos[9]	NULL	94



Taco Array Structure


- Keeping constructed Objects to one side
- No NULL elements in between constructed Objects
- First NULL Element means everything after that is also assumed NULL
- Adding
 - Start from the first Index
 - Find first null element
 - Assign value to there
- Removing
 - Start from the first Index
 - Find the element to remove's index
 - If not found then return
 - Then shift over by one ($tacos[i] = tacos[i+1]$)
 - Set last element to NULL



Identifier	Contents	Byte Address
...
tacos	40	28
...
tacos[0]	86	40
tacos[1]	283	46
tacos[2]	128	52
...
tacos[8]	NULL	88
tacos[9]	NULL	94

Taco Array Structure

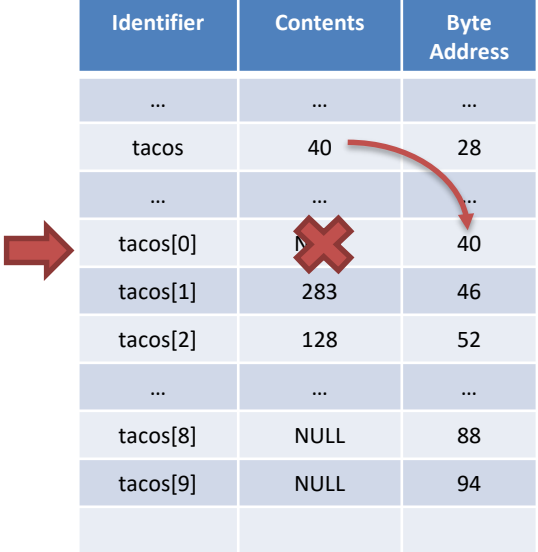
- Keeping constructed Objects to one side
- No NULL elements in between constructed Objects
- First NULL Element means everything after that is also assumed NULL
- Adding
 - Start from the first Index
 - Find first null element
 - Assign value to there
- Removing
 - Start from the first Index
 - Find the element to remove's index
 - If not found then return
 - Then shift over by one ($tacos[i] = tacos[i+1]$)
 - Set last element to NULL



Identifier	Contents	Byte Address
...
tacos	40	28
...
tacos[0]	NULL	40
tacos[1]	283	46
tacos[2]	128	52
...
tacos[8]	NULL	88
tacos[9]	NULL	94

Taco Array Structure

- Keeping constructed Objects to one side
- No NULL elements in between constructed Objects
- First NULL Element means everything after that is also assumed NULL
- Adding
 - Start from the first Index
 - Find first null element
 - Assign value to there
- Removing
 - Start from the first Index
 - Find the element to remove's index
 - If not found then return
 - Then shift over by one ($tacos[i] = tacos[i+1]$)
 - Set last element to NULL



A table representing the Taco Array Structure. The table has three columns: Identifier, Contents, and Byte Address. The rows are as follows:

Identifier	Contents	Byte Address
...
tacos	40	28
...
tacos[0]	...	40
tacos[1]	283	46
tacos[2]	128	52
...
tacos[8]	NULL	88
tacos[9]	NULL	94

A red arrow points from the left towards the table. A red 'X' is drawn over the 'Contents' cell of the 'tacos[0]' row. A red curved arrow points from the 'Contents' cell of the 'tacos' row to the 'Contents' cell of the 'tacos[0]' row.

Taco Array Structure

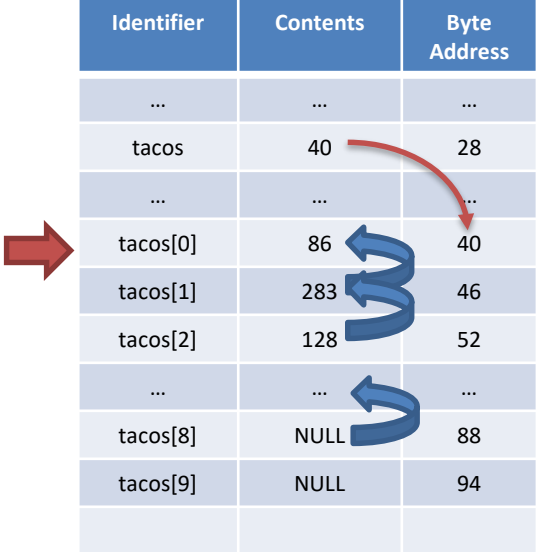
- Keeping constructed Objects to one side
- No NULL elements in between constructed Objects
- First NULL Element means everything after that is also assumed NULL
- Adding
 - Start from the first Index
 - Find first null element
 - Assign value to there
- Removing
 - Start from the first Index
 - Find the element to remove's index
 - If not found then return
 - Then shift over by one ($tacos[i] = tacos[i+1]$)
 - Set last element to NULL



Identifier	Contents	Byte Address
...
tacos	40	28
...
tacos[0]	86	40
tacos[1]	283	46
tacos[2]	128	52
...
tacos[8]	NULL	88
tacos[9]	NULL	94

Taco Array Structure

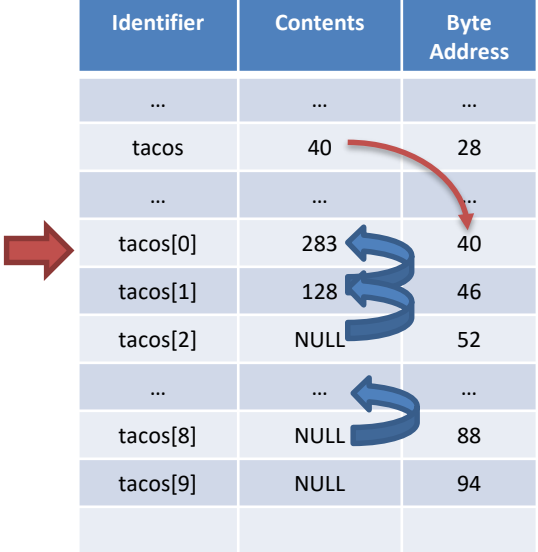
- Keeping constructed Objects to one side
- No NULL elements in between constructed Objects
- First NULL Element means everything after that is also assumed NULL
- Adding
 - Start from the first Index
 - Find first null element
 - Assign value to there
- Removing
 - Start from the first Index
 - Find the element to remove's index
 - If not found then return
 - Then shift over by one ($tacos[i] = tacos[i+1]$)
 - Set last element to NULL



Identifier	Contents	Byte Address
...
tacos	40	28
...
tacos[0]	86	40
tacos[1]	283	46
tacos[2]	128	52
...
tacos[8]	NULL	88
tacos[9]	NULL	94

Taco Array Structure

- Keeping constructed Objects to one side
- No NULL elements in between constructed Objects
- First NULL Element means everything after that is also assumed NULL
- Adding
 - Start from the first Index
 - Find first null element
 - Assign value to there
- Removing
 - Start from the first Index
 - Find the element to remove's index
 - If not found then return
 - Then shift over by one ($tacos[i] = tacos[i+1]$)
 - Set last element to NULL



Identifier	Contents	Byte Address
...
tacos	40	28
...
tacos[0]	283	40
tacos[1]	128	46
tacos[2]	NULL	52
...
tacos[8]	NULL	88
tacos[9]	NULL	94

Add Tacos Demo

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it
}

```

4

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	NULL	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...

More Memory

Identifier	Contents	Byte Address
...
aTaco	256	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	

Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	NULL	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...

More Memory

Identifier	Contents	Byte Address
...
aTaco	256	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	

Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

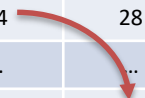
```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	NULL	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...

More Memory

Identifier	Contents	Byte Address
...
aTaco	256	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	



Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

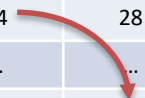
```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	NULL	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...

More Memory

Identifier	Contents	Byte Address
...
aTaco	256	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	



Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	NULL	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
i	0	93

More Memory

Identifier	Contents	Byte Address
...
aTaco	256	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	

Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	NULL	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
i	0	93

More Memory

Identifier	Contents	Byte Address
...
aTaco	256	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	

Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	NULL	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
i	0	93

More Memory

Identifier	Contents	Byte Address
...
aTaco	256	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	

Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	NULL	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
i	0	93

More Memory

Identifier	Contents	Byte Address
...
aTaco	256	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	

Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
i	0	93

More Memory

Identifier	Contents	Byte Address
...
aTaco	256	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	

Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

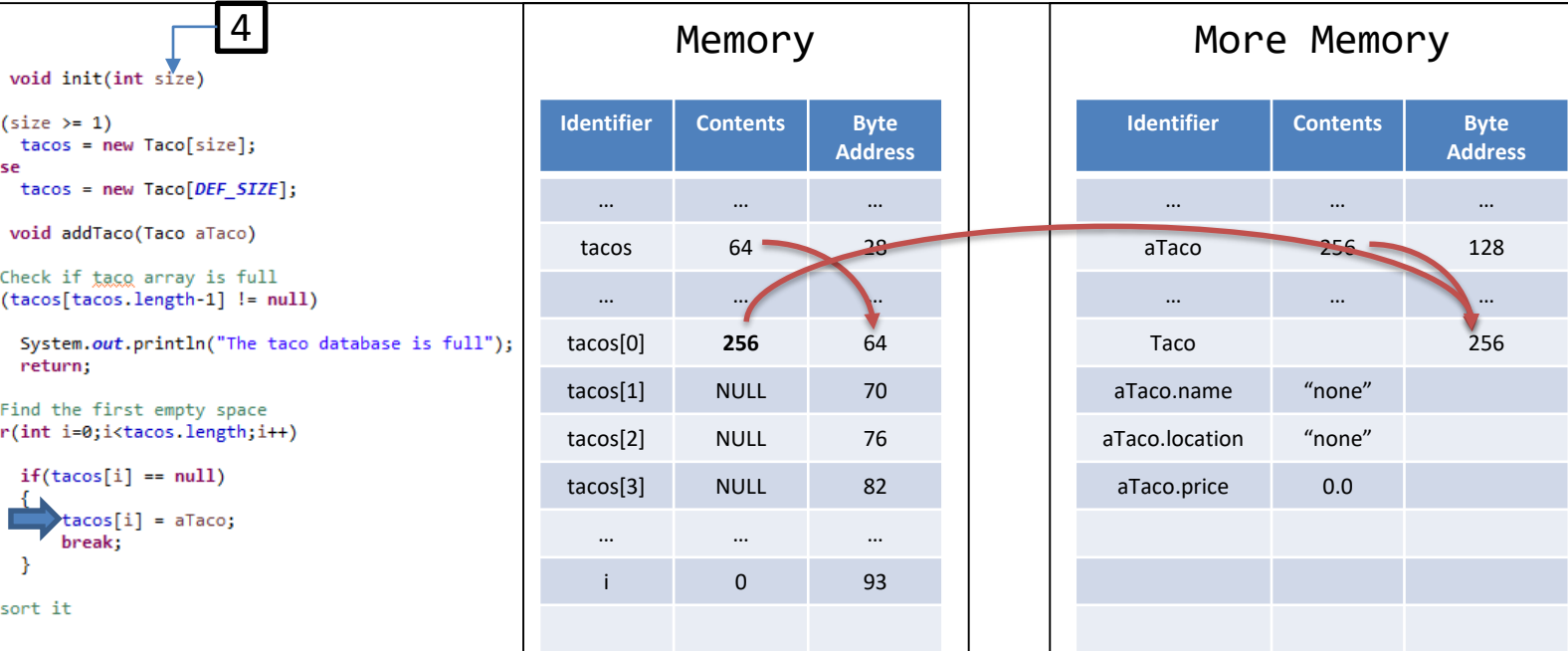
```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
i	0	93

More Memory

Identifier	Contents	Byte Address
...
aTaco	256	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	



Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
i	0	93

More Memory

Identifier	Contents	Byte Address
...
aTaco	256	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	



Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...

More Memory

Identifier	Contents	Byte Address
...
aTaco	256	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	



Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it
}

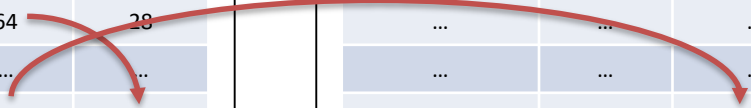
```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...

More Memory

Identifier	Contents	Byte Address
...
...
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	



Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

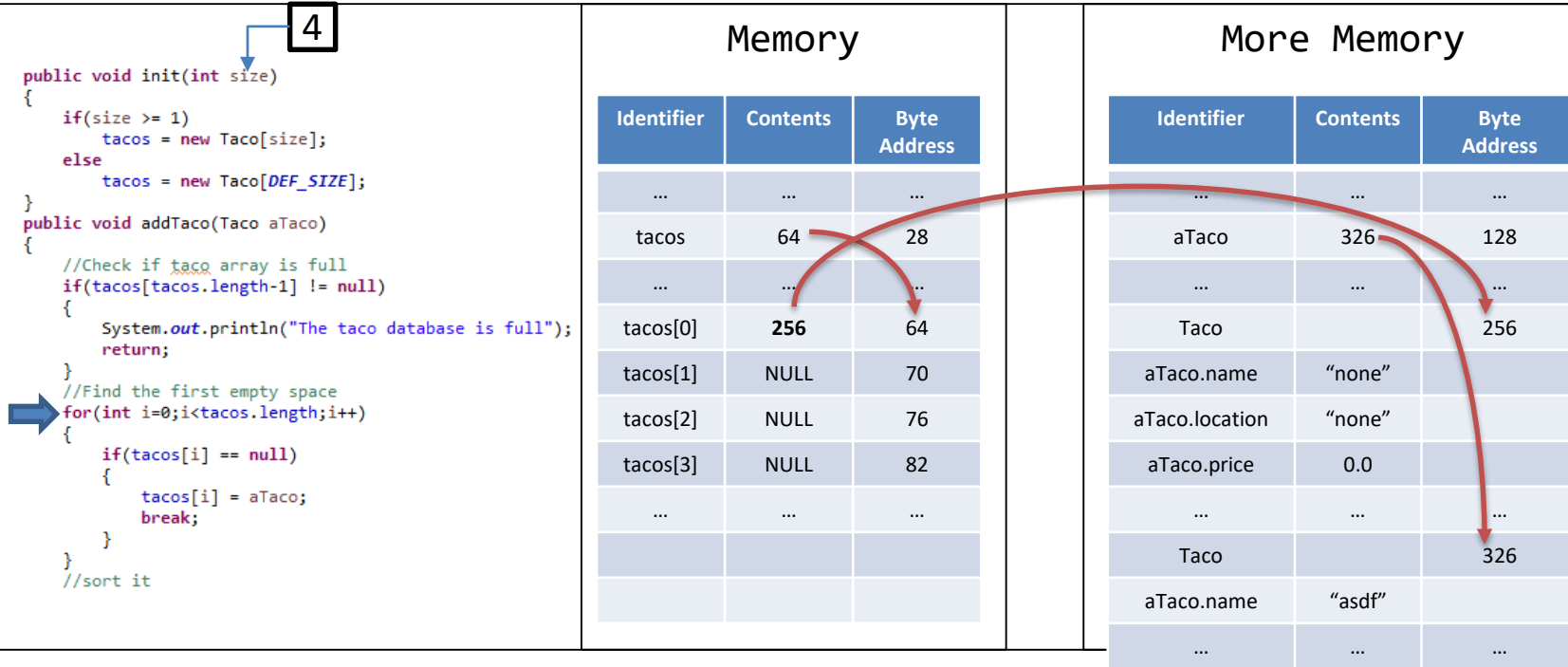
```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...

More Memory

Identifier	Contents	Byte Address
...
aTaco	326	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...



Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
i	0	93

More Memory

Identifier	Contents	Byte Address
...
aTaco	326	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...

Add Tacos Demo

4

```
public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it
}
```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
i	0	93

More Memory

Identifier	Contents	Byte Address
...
aTaco	326	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...

Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

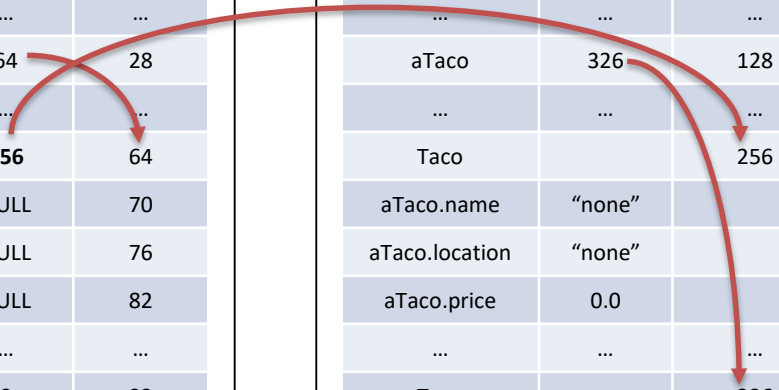
```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
i	0	93

More Memory

Identifier	Contents	Byte Address
...
aTaco	326	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...



Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
i	1	93

More Memory

Identifier	Contents	Byte Address
...
aTaco	326	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...

Add Tacos Demo

4

```
public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it
}
```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
i	1	93

More Memory

Identifier	Contents	Byte Address
...
aTaco	326	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...

Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	326	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
i	1	93

More Memory

Identifier	Contents	Byte Address
...
aTaco	326	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...

Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

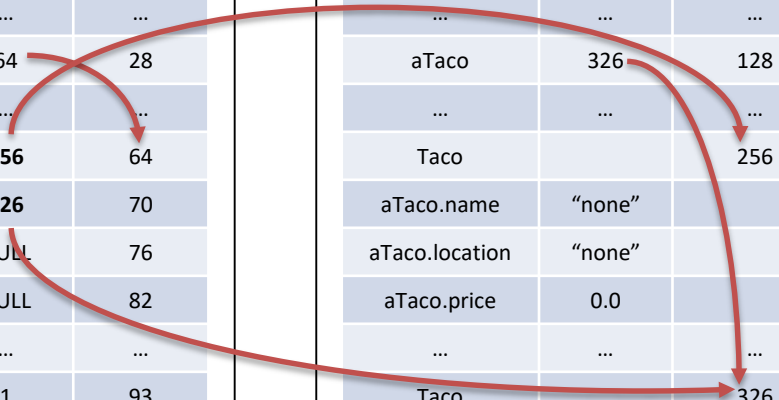
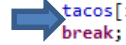
```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	326	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
i	1	93

More Memory

Identifier	Contents	Byte Address
...
aTaco	326	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...



Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

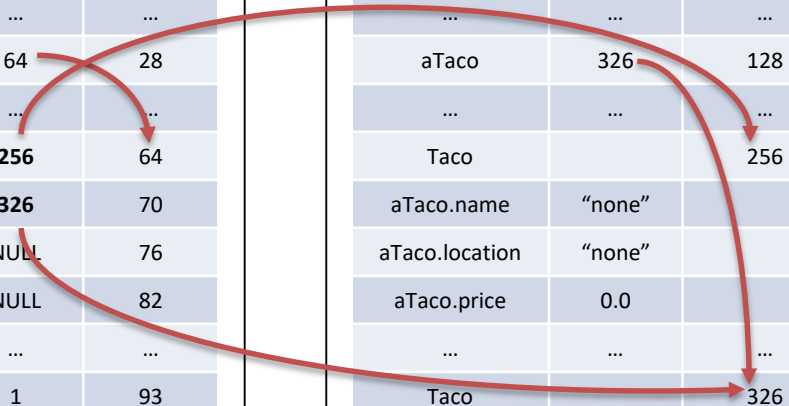
```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	326	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
i	1	93

More Memory

Identifier	Contents	Byte Address
...
aTaco	326	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...



Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

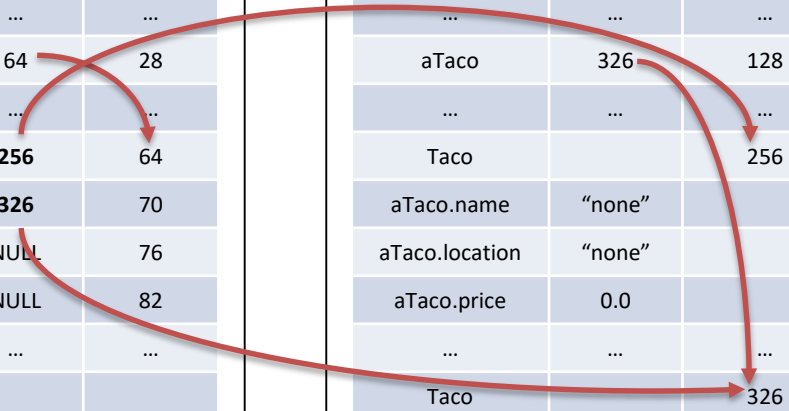
```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	326	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...

More Memory

Identifier	Contents	Byte Address
...
aTaco	326	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...



Add Tacos Demo

4

```

public void init(int size)
{
    if(size >= 1)
        tacos = new Taco[size];
    else
        tacos = new Taco[DEF_SIZE];
}
public void addTaco(Taco aTaco)
{
    //Check if taco array is full
    if(tacos[tacos.length-1] != null)
    {
        System.out.println("The taco database is full");
        return;
    }
    //Find the first empty space
    for(int i=0;i<tacos.length;i++)
    {
        if(tacos[i] == null)
        {
            tacos[i] = aTaco;
            break;
        }
    }
    //sort it

```

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	326	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...

More Memory

Identifier	Contents	Byte Address
...
...
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...

Remove Tacos Demo

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
        }
        //Make sure the last index is always null
        tacos[tacos.length-1] = null;
    }
}

```

“asdf”

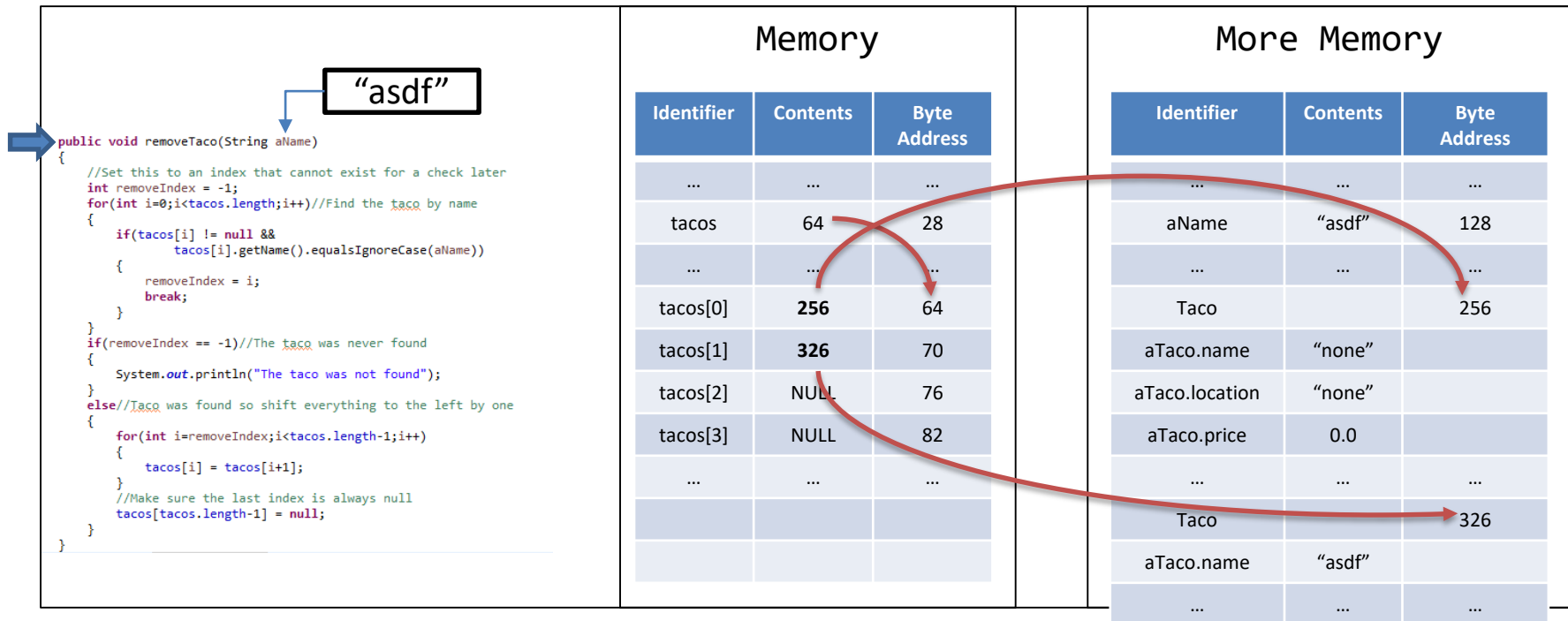
Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	326	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...

More Memory

Identifier	Contents	Byte Address
...
aName	“asdf”	128
...
Taco		256
aTaco.name	“none”	
aTaco.location	“none”	
aTaco.price	0.0	
...
Taco		326
aTaco.name	“asdf”	
...

Remove Tacos Demo



Remove Tacos Demo

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
        }
        //Make sure the last index is always null
        tacos[tacos.length-1] = null;
    }
}

```

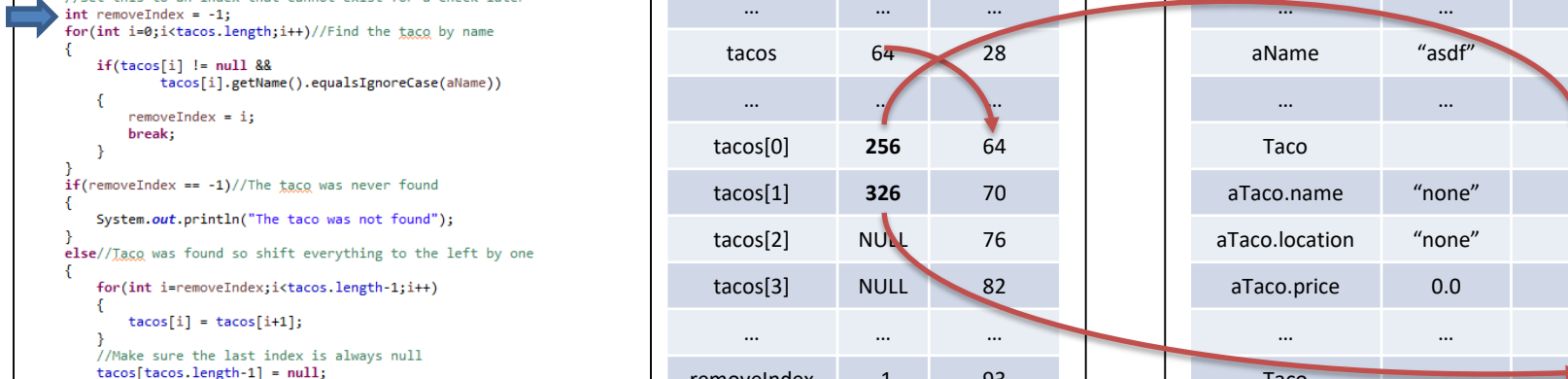
Diagram: A box containing "asdf" has an arrow pointing to the `aName` parameter in the `removeTaco` method. A blue arrow points to the `for` loop.

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	326	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	-1	93

More Memory

Identifier	Contents	Byte Address
...
aName	"asdf"	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...



Remove Tacos Demo

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
        }
        //Make sure the last index is always null
        tacos[tacos.length-1] = null;
    }
}

```

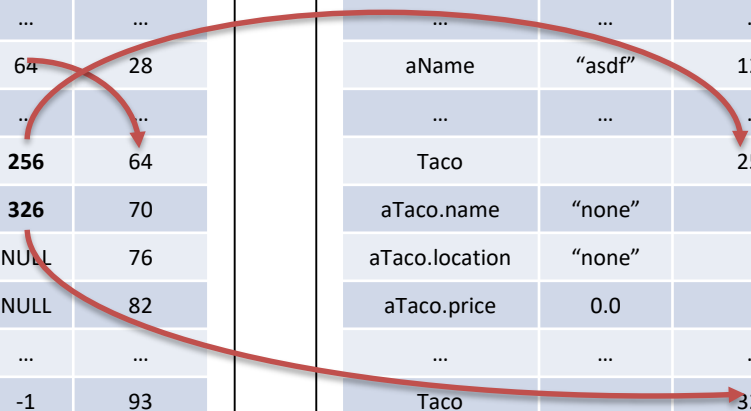
“asdf”

Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	326	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	-1	93
i	0	97

More Memory

Identifier	Contents	Byte Address
...
aName	“asdf”	128
...
Taco		256
aTaco.name	“none”	
aTaco.location	“none”	
aTaco.price	0.0	
...
Taco		326
aTaco.name	“asdf”	
...



Remove Tacos Demo

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
        }
        //Make sure the last index is always null
        tacos[tacos.length-1] = null;
    }
}

```

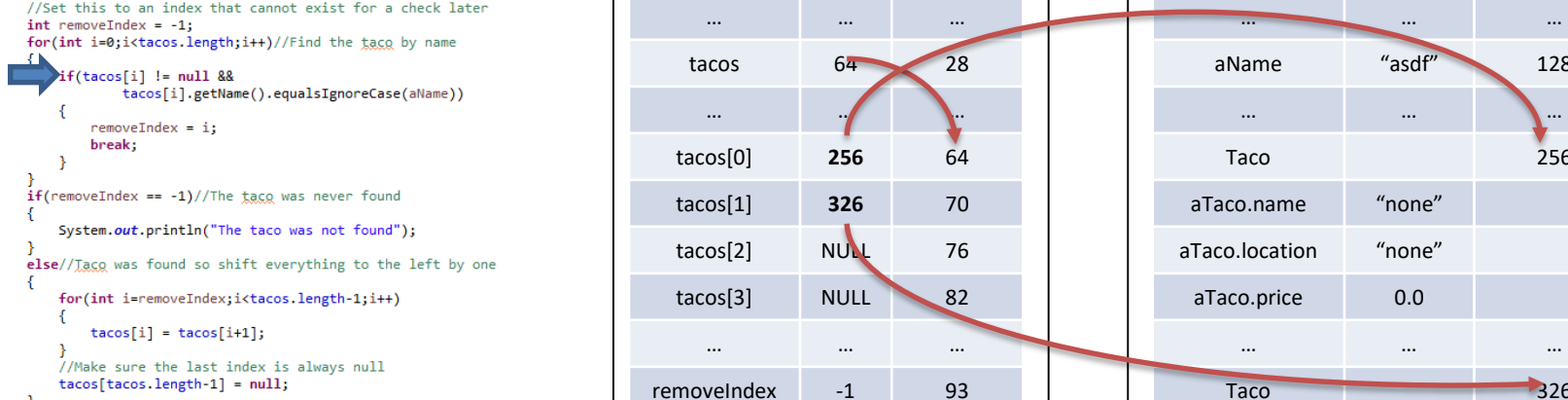
“asdf”

Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	326	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	-1	93
i	0	97

More Memory

Identifier	Contents	Byte Address
...
aName	“asdf”	128
...
Taco		256
aTaco.name	“none”	
aTaco.location	“none”	
aTaco.price	0.0	
...
Taco		326
aTaco.name	“asdf”	
...



Remove Tacos Demo

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
        }
        //Make sure the last index is always null
        tacos[tacos.length-1] = null;
    }
}

```

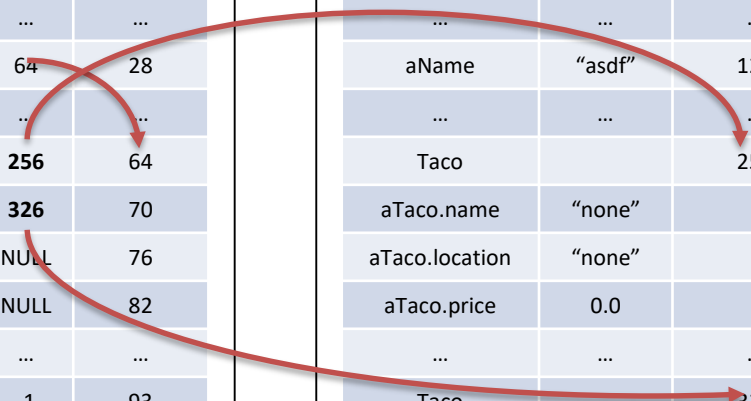
Diagram: A box containing "asdf" has an arrow pointing to the `aName` parameter in the code. A blue arrow points to the `break;` statement in the code.

Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	326	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	-1	93
i	0	97

More Memory

Identifier	Contents	Byte Address
...
aName	"asdf"	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...



Remove Tacos Demo

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
        }
        //Make sure the last index is always null
        tacos[tacos.length-1] = null;
    }
}

```

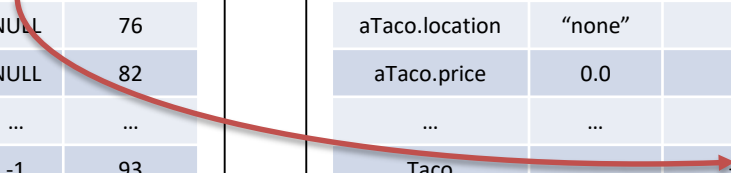
“asdf”

Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	326	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	-1	93
i	1	97

More Memory

Identifier	Contents	Byte Address
...
aName	“asdf”	128
...
Taco		256
aTaco.name	“none”	
aTaco.location	“none”	
aTaco.price	0.0	
...
Taco		326
aTaco.name	“asdf”	
...



Remove Tacos Demo

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
        }
        //Make sure the last index is always null
        tacos[tacos.length-1] = null;
    }
}

```

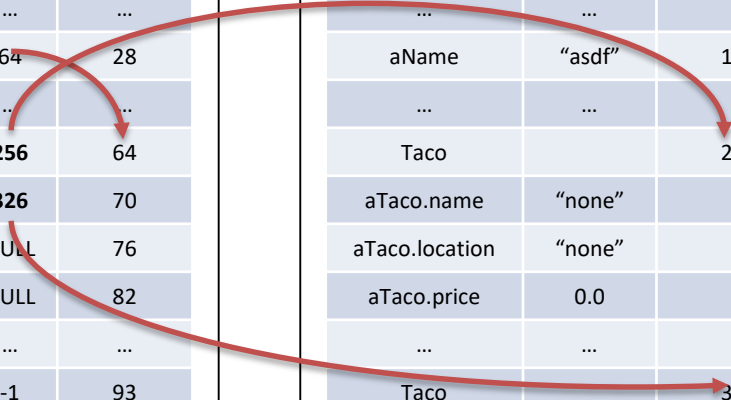
“asdf”

Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	326	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	-1	93
i	1	97

More Memory

Identifier	Contents	Byte Address
...
aName	“asdf”	128
...
Taco		256
aTaco.name	“none”	
aTaco.location	“none”	
aTaco.price	0.0	
...
Taco		326
aTaco.name	“asdf”	
...



Remove Tacos Demo

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
        }
        //Make sure the last index is always null
        tacos[tacos.length-1] = null;
    }
}

```

"asdf"

Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	326	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	1	93
i	1	97

More Memory

Identifier	Contents	Byte Address
...
aName	"asdf"	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...

Remove Tacos Demo

"asdf"

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
            tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
        }
        //Make sure the last index is always null
        tacos[tacos.length-1] = null;
    }
}

```

Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	326	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	1	93
i	1	97

More Memory

Identifier	Contents	Byte Address
...
aName	"asdf"	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...

Remove Tacos Demo

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
        }
        //Make sure the last index is always null
        tacos[tacos.length-1] = null;
    }
}

```

"asdf"

➔

Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	326	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	1	93

More Memory

Identifier	Contents	Byte Address
...
aName	"asdf"	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...

Remove Tacos Demo

"asdf"

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
            tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
        }
        //Make sure the last index is always null
        tacos[tacos.length-1] = null;
    }
}
            
```

Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	326	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	1	93

More Memory

Identifier	Contents	Byte Address
...
aName	"asdf"	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...

Remove Tacos Demo

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
        }
        //Make sure the last index is always null
        tacos[tacos.length-1] = null;
    }
}

```

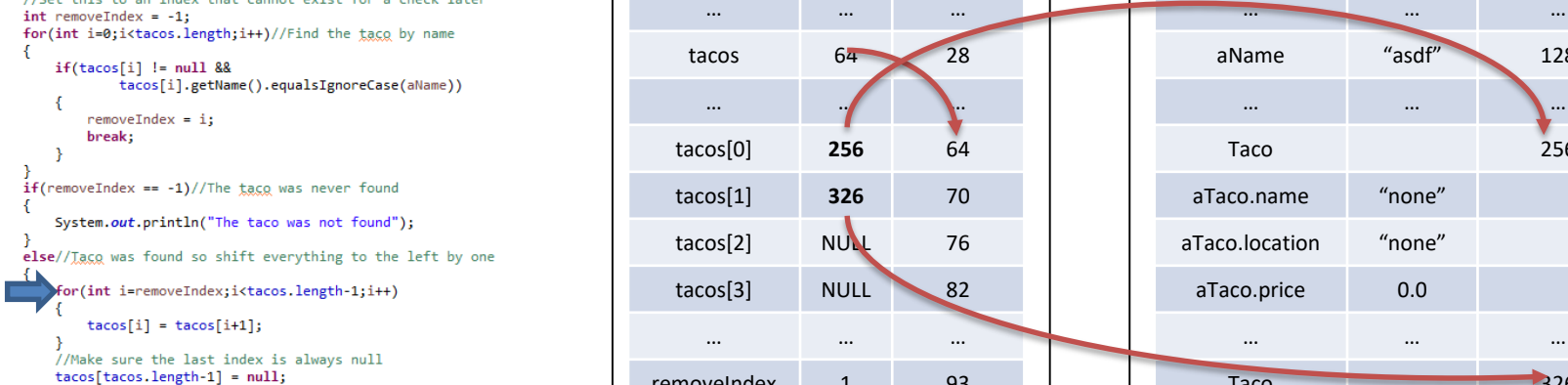
"asdf"

Memory

Identifier	Contents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	326	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	1	93

More Memory

Identifier	Contents	Byte Address
...
aName	"asdf"	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...



Remove Tacos Demo

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
        }
        //Make sure the last index is always null
        tacos[tacos.length-1] = null;
    }
}

```

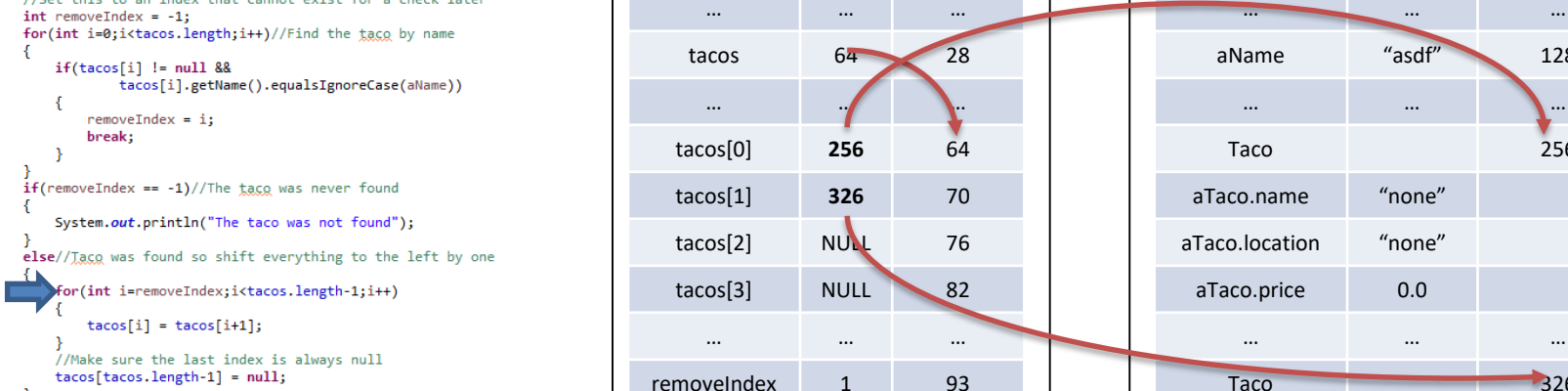
"asdf"

Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	326	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	1	93
i	1	97

More Memory

Identifier	Contents	Byte Address
...
aName	"asdf"	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...



Remove Tacos Demo

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
            //Make sure the last index is always null
            tacos[tacos.length-1] = null;
        }
    }
}

```

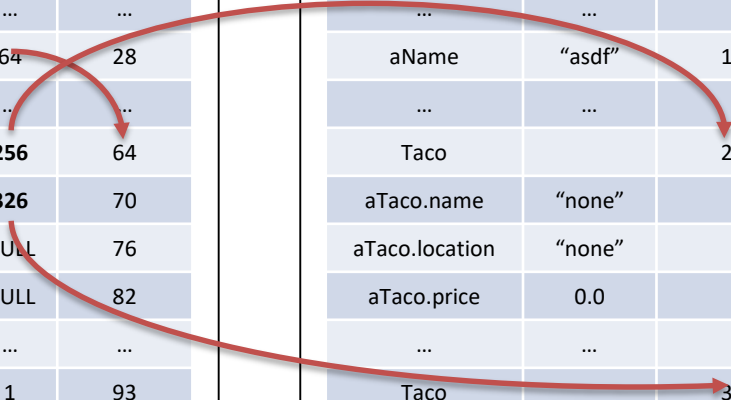
“asdf”

Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	326	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	1	93
i	1	97

More Memory

Identifier	Contents	Byte Address
...
aName	“asdf”	128
...
Taco		256
aTaco.name	“none”	
aTaco.location	“none”	
aTaco.price	0.0	
...
Taco		326
aTaco.name	“asdf”	
...



Remove Tacos Demo

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
            //Make sure the last index is always null
            tacos[tacos.length-1] = null;
        }
    }
}

```

"asdf"

Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	1	93
i	1	97

More Memory

Identifier	Contents	Byte Address
...
aName	"asdf"	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...

Remove Tacos Demo

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
            //Make sure the last index is always null
            tacos[tacos.length-1] = null;
        }
    }
}

```

Diagram: A box containing "asdf" has an arrow pointing to the `aName` parameter in the code above. Another arrow points from the `tacos[i]` line to the `tacos` table in the 'Memory' section.

Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	1	93
i	1	97

More Memory

Identifier	Contents	Byte Address
...
aName	"asdf"	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...

Remove Tacos Demo

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
        }
        //Make sure the last index is always null
        tacos[tacos.length-1] = null;
    }
}

```

“asdf”

Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	1	93
i	2	97

More Memory

Identifier	Contents	Byte Address
...
aName	“asdf”	128
...
Taco		256
aTaco.name	“none”	
aTaco.location	“none”	
aTaco.price	0.0	
...
Taco		326
aTaco.name	“asdf”	
...

Remove Tacos Demo

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
            //Make sure the last index is always null
            tacos[tacos.length-1] = null;
        }
    }
}

```

Diagram: A box containing "asdf" has an arrow pointing to the `aName` parameter in the code. Another arrow points from the `tacos[i]` expression in the loop to the `tacos` array in the memory table.

Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	1	93
i	2	97

More Memory

Identifier	Contents	Byte Address
...
aName	"asdf"	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...

Remove Tacos Demo

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
        }
        //Make sure the last index is always null
        tacos[tacos.length-1] = null;
    }
}

```

Diagram: A box containing "asdf" has an arrow pointing to the `aName` parameter in the `removeTaco` method. Another arrow points from the `for` loop in the code to the `tacos` array in the memory table.

Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	1	93
i	3	97

More Memory

Identifier	Contents	Byte Address
...
aName	"asdf"	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...

Remove Tacos Demo

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
        }
        //Make sure the last index is always null
        tacos[tacos.length-1] = null;
    }
}

```

"asdf"

Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	1	93
i	3	97

More Memory

Identifier	Contents	Byte Address
...
aName	"asdf"	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...

Remove Tacos Demo

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
        }
        //Make sure the last index is always null
        tacos[tacos.length-1] = null;
    }
}

```

"asdf"

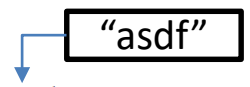
Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	1	93
i	3	97

More Memory

Identifier	Contents	Byte Address
...
aName	"asdf"	128
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...

Remove Tacos Demo



```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
            tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
        }
        //Make sure the last index is always null
        tacos[tacos.length-1] = null;
    }
}

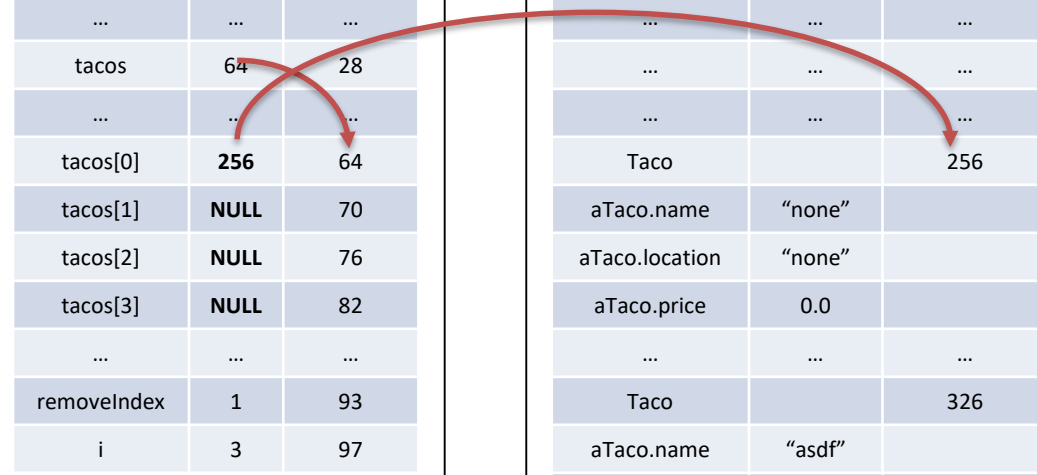
```

Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	1	93
i	3	97

More Memory

Identifier	Contents	Byte Address
...
...
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...
Taco		326
aTaco.name	"asdf"	
...



Remove Tacos Demo

“asdf”

```

public void removeTaco(String aName)
{
    //Set this to an index that cannot exist for a check later
    int removeIndex = -1;
    for(int i=0;i<tacos.length;i++)//Find the taco by name
    {
        if(tacos[i] != null &&
           tacos[i].getName().equalsIgnoreCase(aName))
        {
            removeIndex = i;
            break;
        }
    }
    if(removeIndex == -1)//The taco was never found
    {
        System.out.println("The taco was not found");
    }
    else//Taco was found so shift everything to the left by one
    {
        for(int i=removeIndex;i<tacos.length-1;i++)
        {
            tacos[i] = tacos[i+1];
        }
        //Make sure the last index is always null
        tacos[tacos.length-1] = null;
    }
}

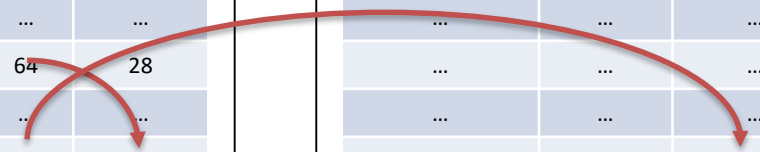
```

Memory

Identifier	Cont ents	Byte Address
...
tacos	64	28
...
tacos[0]	256	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...
removeIndex	1	93
i	3	97

More Memory

Identifier	Contents	Byte Address
...
...
...
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
...



For-Each Loop

- Special version of a for-loop
- Store each element in a temporary variable
- Great of looping through every element in a collection (like an array)
- Should NOT use a for-each-loop when modifying the data structure
 - Should not use when adding new elements
 - Should not use when removing elements

For-Each-Loop Syntax

```
for(<<type>> <<id>> : <<type collection>>)
{
    //Body of the for-each-loop
}
```

Example

```
for(Taco t : tacos)
{
    System.out.println(t);
}
```

For-Each Loop

For-Loop Example

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

For-Each-Loop Example

```
for(Taco t : tacos)  
{  
    System.out.println(t);  
}
```


Switch Statement

- Special version of an if, else-if, and else statement
- The argument is a “Controlling Value” corresponds to “Cases”
- The “Controlling Value” can either be
 - An integer type
 - A character type
 - An Enum
 - A String
- The “break” statement is needed to stop the execution of any following cases
 - Without the “break” the following cases’ statements will run or *fall through*

Switch-Statement Syntax

```
switch(<<Controlling Value>>)  
{  
    case <<value00>>:  
        //Case00 Statements  
        break;  
    case <<value01>>:  
        //Case01 Statements  
        break;  
    default:  
        //Default Case  
}
```

Switch Statement

If, Else-if, Else Example

```
int choice = keyboard.nextInt();
keyboard.nextLine();
if(choice == 1)
{
    addTaco();
}
else if(choice == 2)
{
    removeTaco();
}
else if(choice == 9)
{
    quit = true;
}
else
{
    System.out.println("Invalid Input");
}
```

Switch-Statement Example

```
int choice = keyboard.nextInt();
keyboard.nextLine();
switch(choice)
{
    case 1:
        addTaco();
        break;
    case 2:
        removeTaco();
        break;
    case 9:
        quit = true;
        break;
    default:
        System.out.println("Invalid Input");
}
```