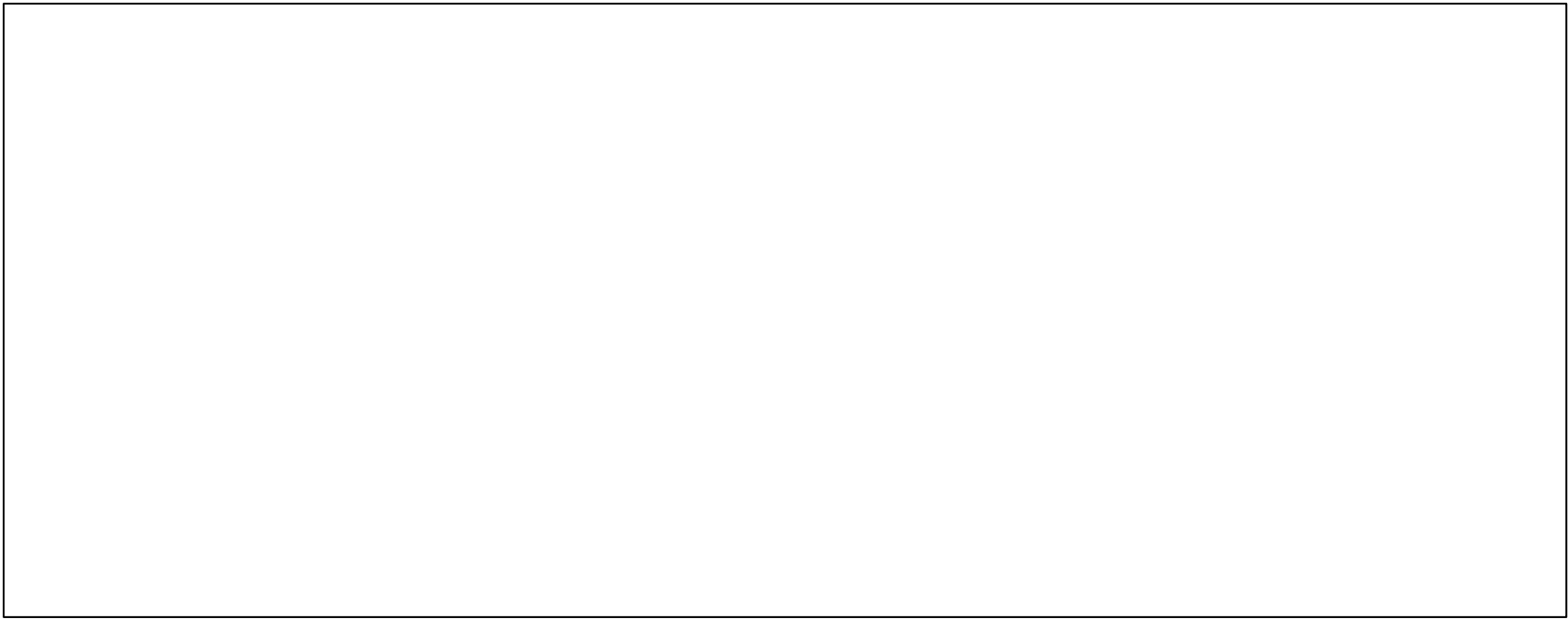




Even More Programming Review

Big Example

Problem



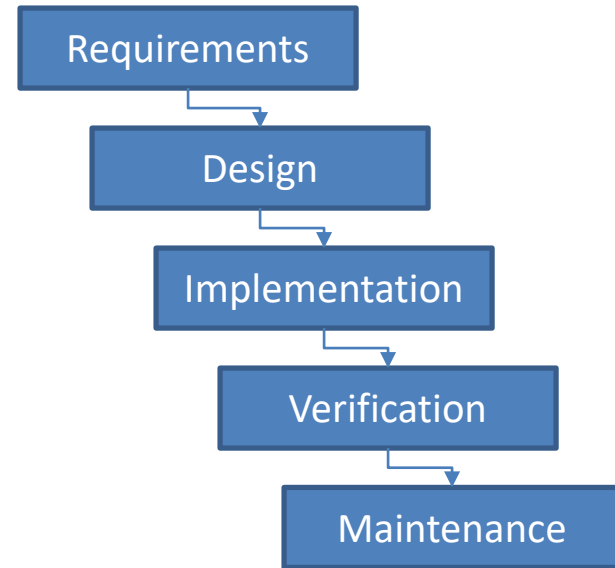
Problem



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

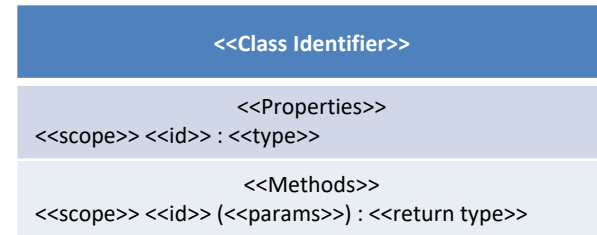


- Should be able to
 - Add a Taco
 - Remove a Taco by Name
 - Sort by Price
 - Display all Taco information
 - Store in a Taco File
 - Read from Stored Taco Files
- Clear and Simple Front End

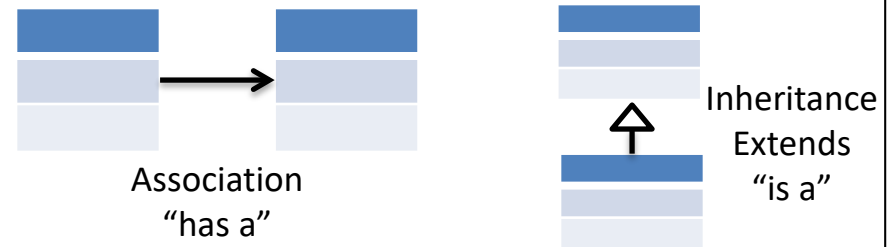
Design

- 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
 - Block Arrows is the Inheritance or “is a”
- Static variables and method are underlined
 - Constants are all UPPER CASE

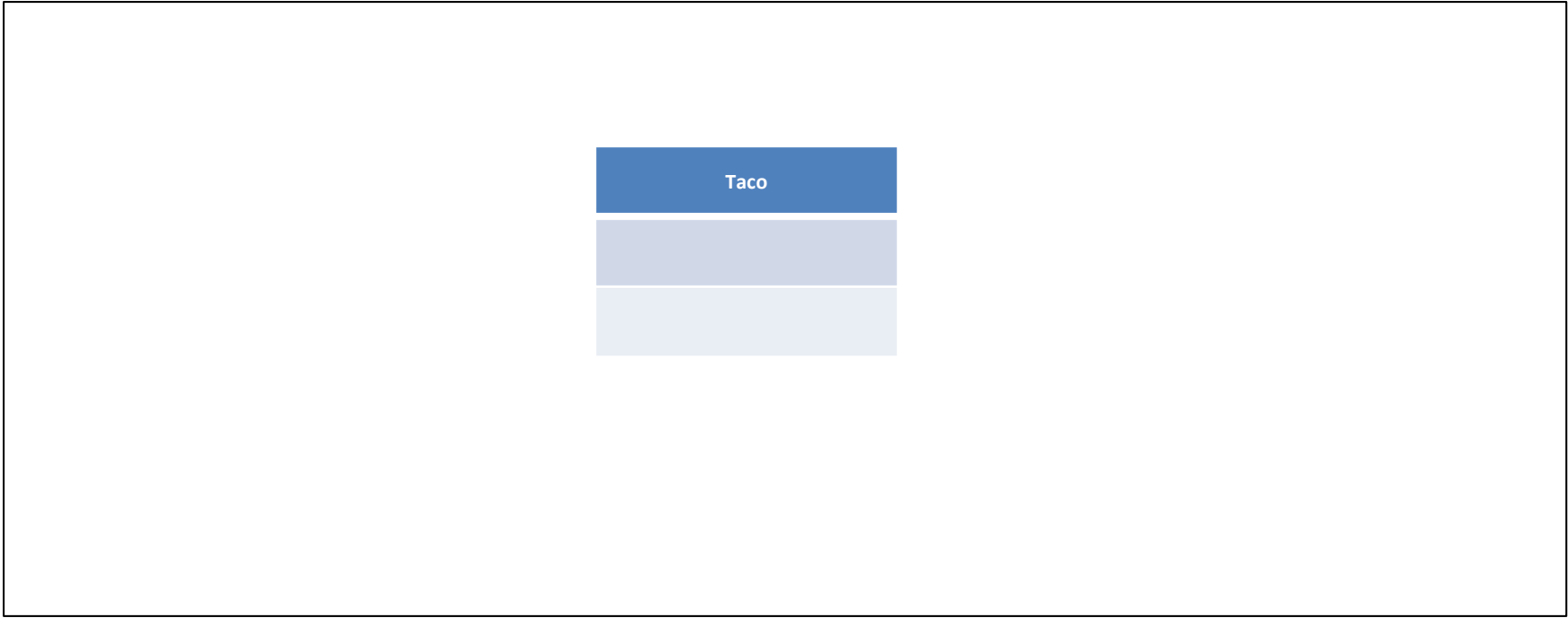
UML Class Diagram



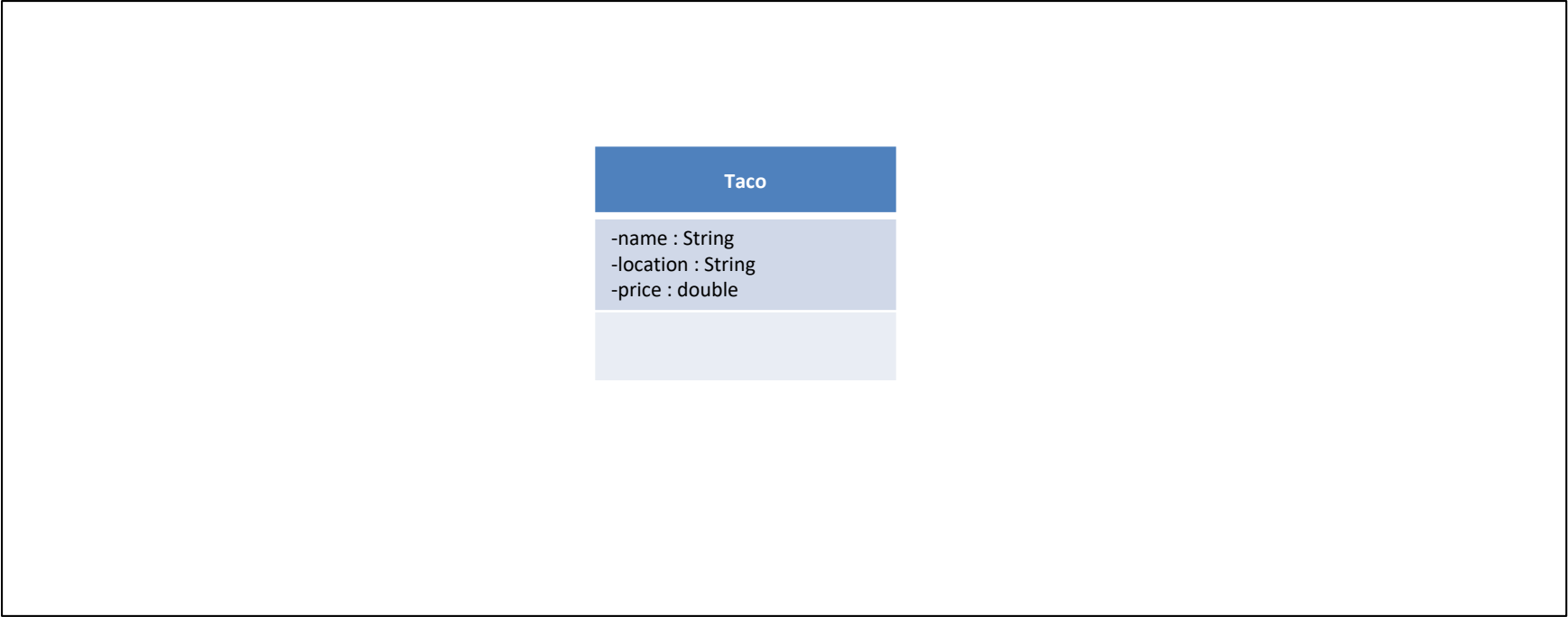
UML Class Diagram Arrows



Design

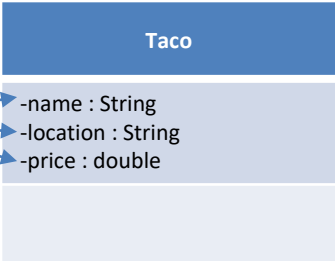


Design

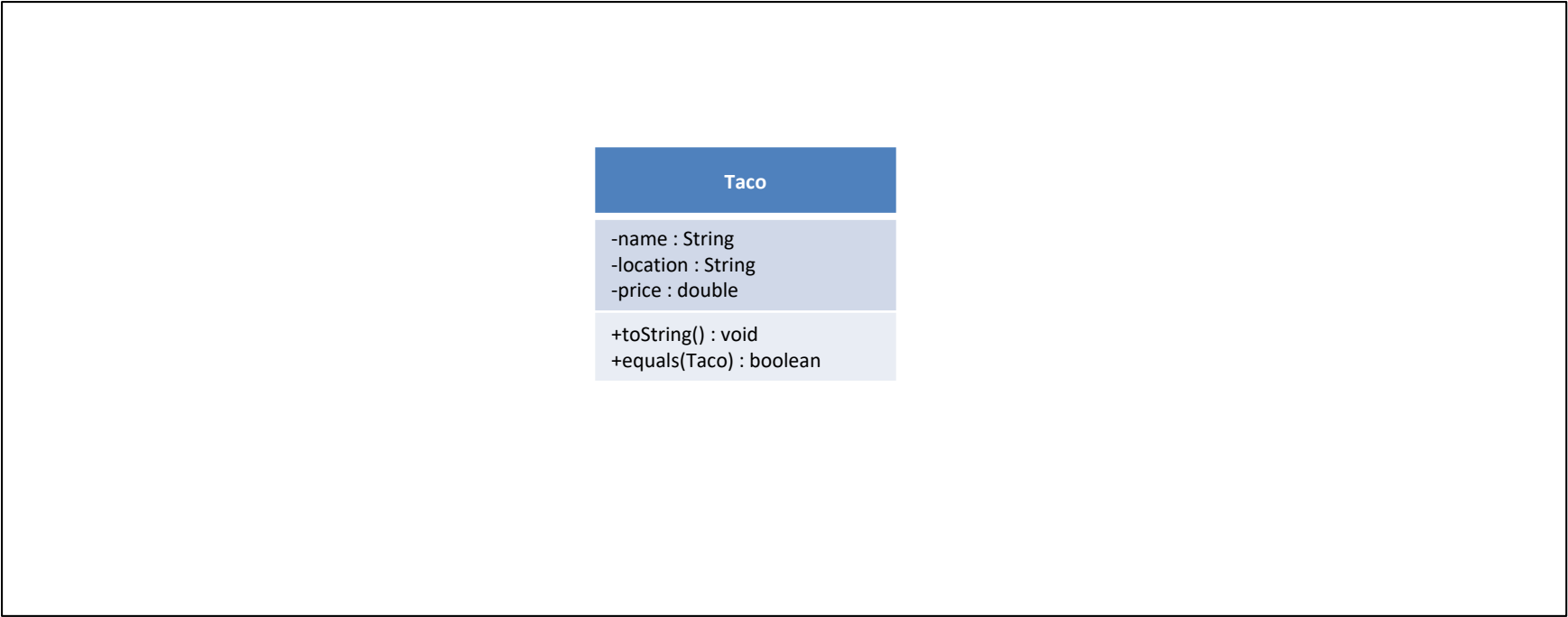


Design

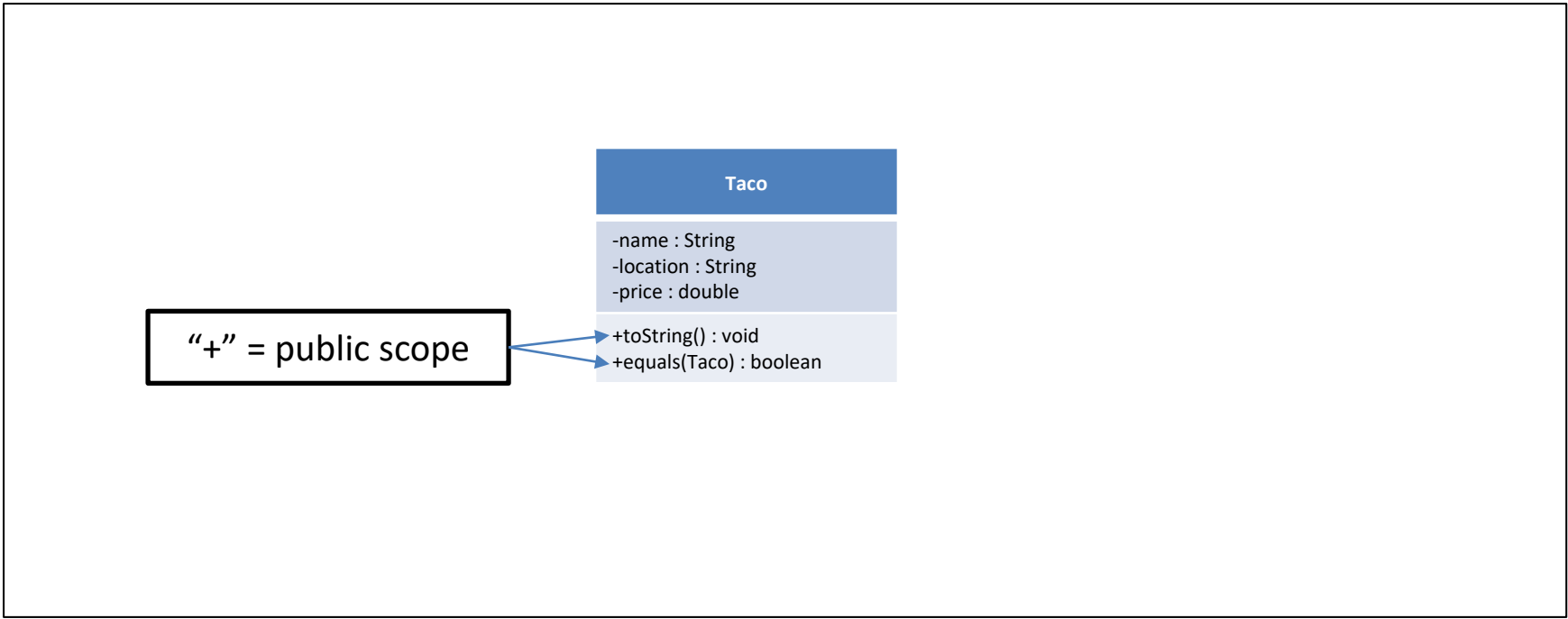
“-” = private scope



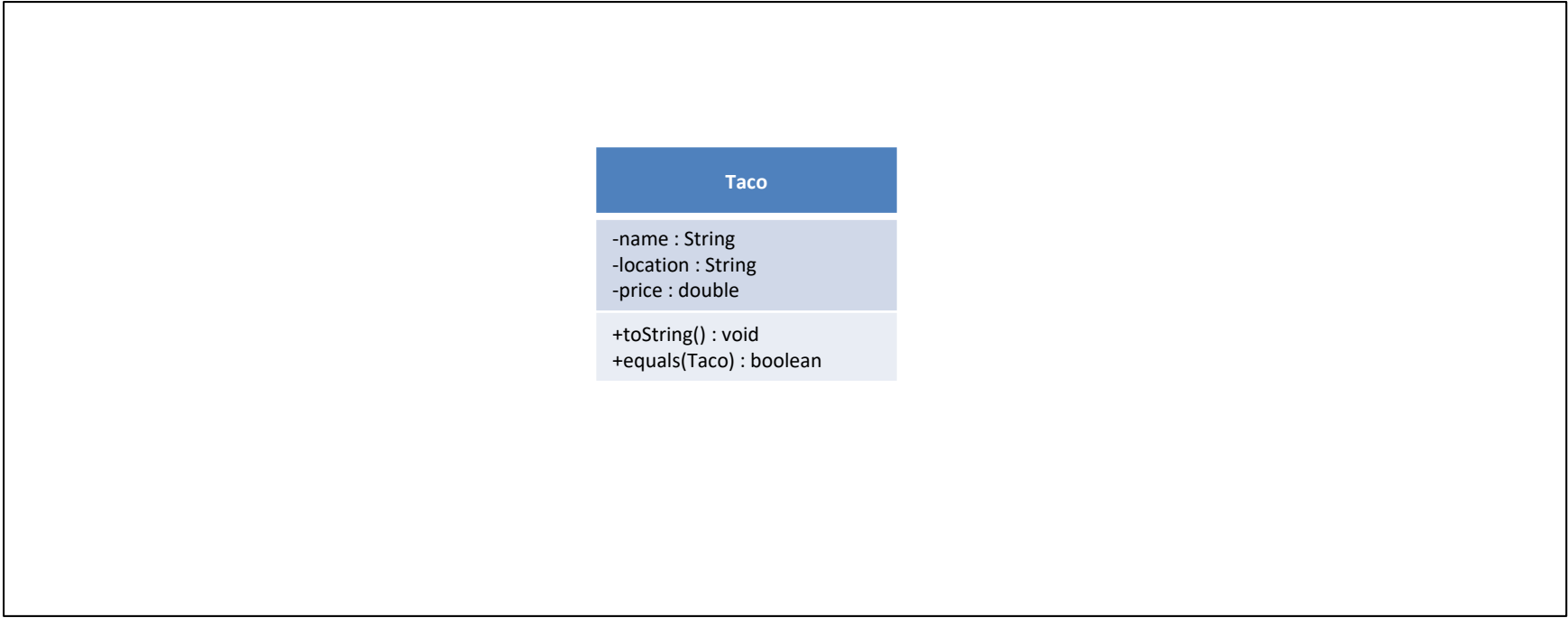
Design



Design



Design



Taco

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

+toString() : void
+equals(Taco) : boolean

Design

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

TacoManager

Design

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

TacoManager
-tacos: Taco[] <u>+DEF_SIZE: int</u> <u>+DELIM:String</u> <u>+HEADER_FIELD_AMT: int</u> <u>+BODY_FIELD_AMT: int</u>

Design

Taco

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

+toString() : void
+equals(Taco) : boolean

TacoManager

-tacos: Taco[]
+DEF_SIZE: int
+DELIM:String
+HEADER_FIELD_AMT: int
+BODY_FIELD_AMT: int

File Format

```
//Header  
Taco Amt:\t<<number of tacos>>\n  
//Body  
<<name>>\t<<location>>\t<<price>>\n  
...
```

Example

```
Taco Amt: 3  
Name1      location1  1.0  
Name2      location2  2.0  
Name3      location3  2.5
```

Design

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

TacoManager
-tacos: Taco[] <u>+DEF_SIZE: int</u> <u>+DELIM:String</u> <u>+HEADER_FIELD_AMT: int</u> <u>+BODY_FIELD_AMT: int</u>

Design

Taco

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

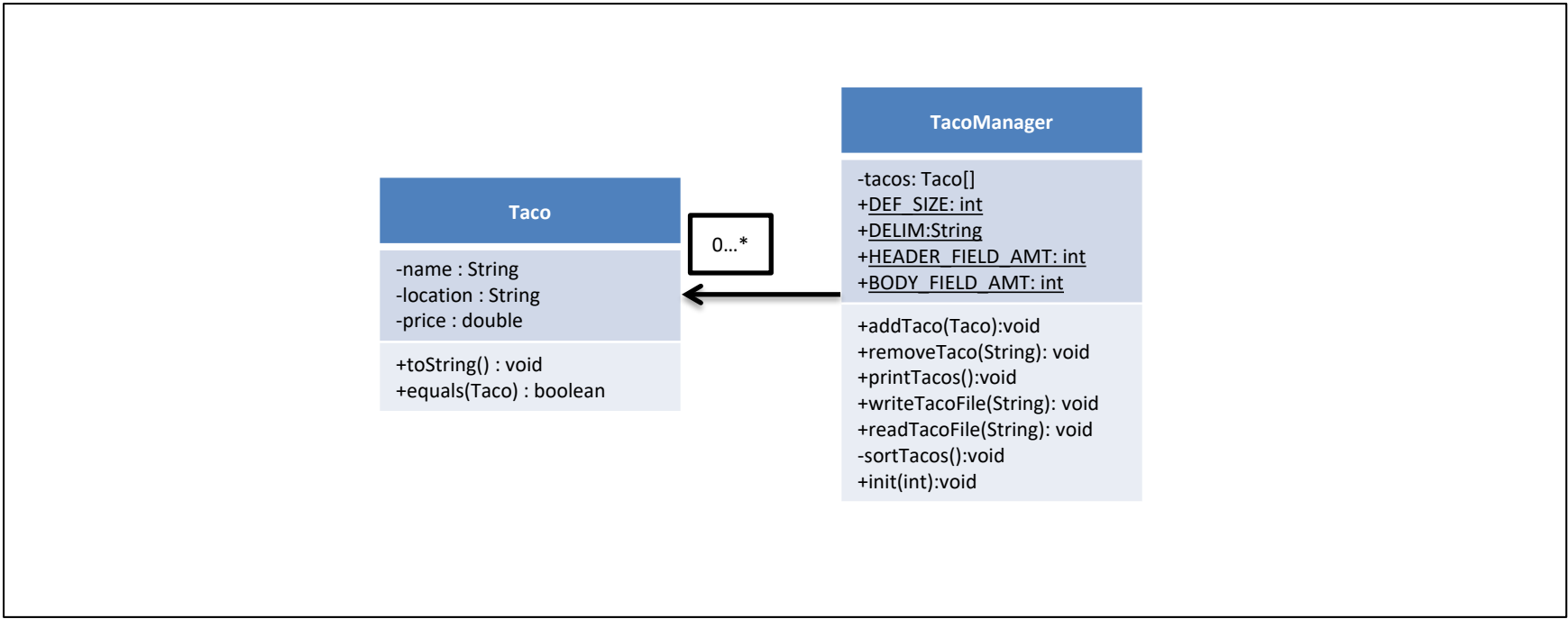
+toString() : void
+equals(Taco) : boolean

TacoManager

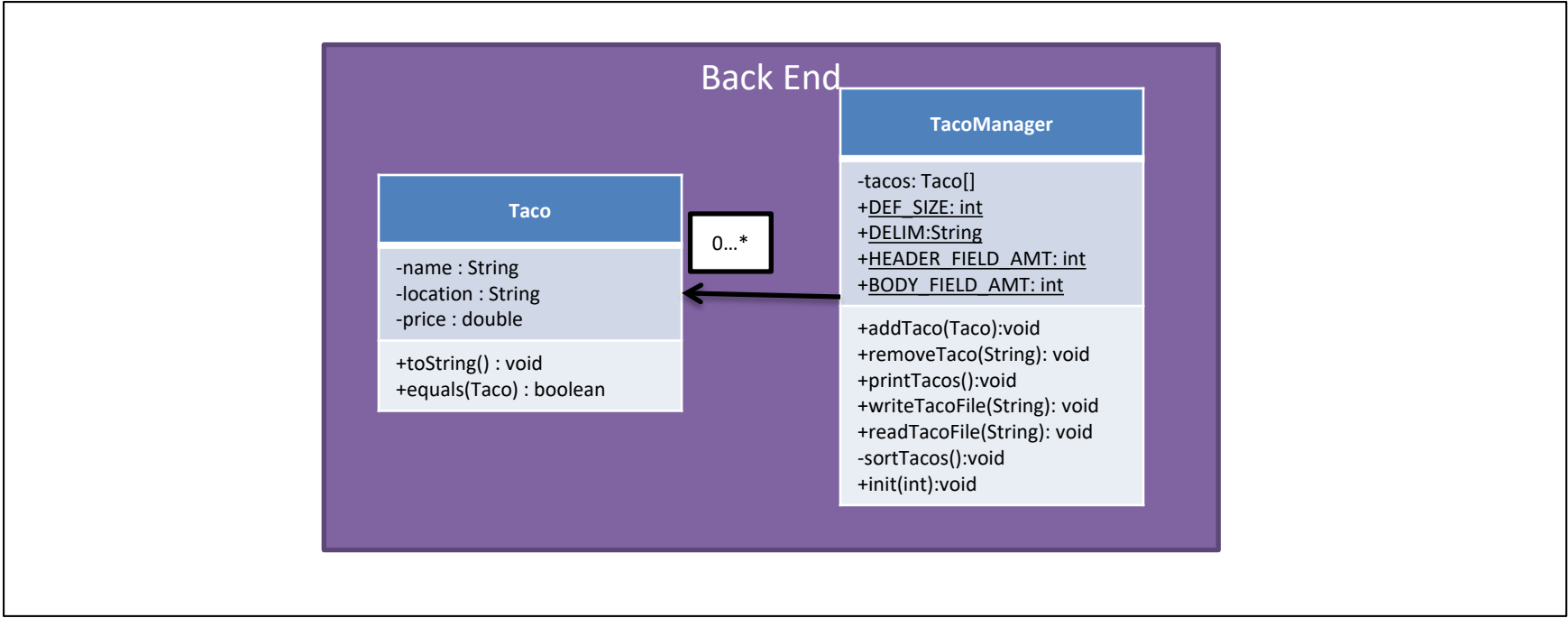
-tacos: Taco[]
+DEF_SIZE: int
+DELIM:String
+HEADER FIELD AMT: int
+BODY FIELD AMT: int

+addTaco(Taco):void
+removeTaco(String): void
+printTacos():void
+writeTacoFile(String): void
+readTacoFile(String): void
-sortTacos():void
+init(int):void

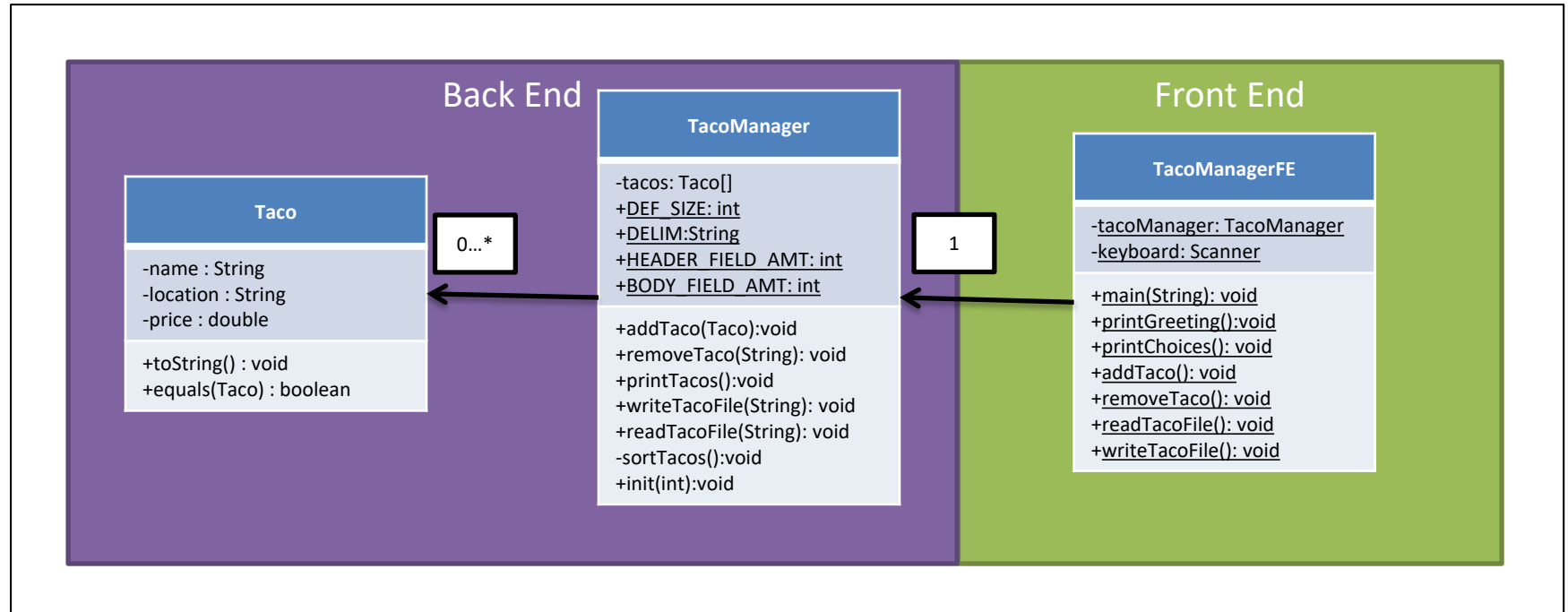
Design



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

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

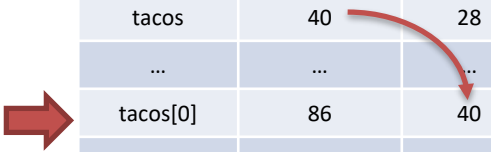
- Keeping constructed Objects to one side
- No NULL elements in between constructed Objects
- First NULL Element means everything after that is also assumed NULL
- Adding
 - Start from the first Index
 - Find first null element
 - Assign value to there
- Removing
 - Start from the first Index
 - Find 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



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



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

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
...

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

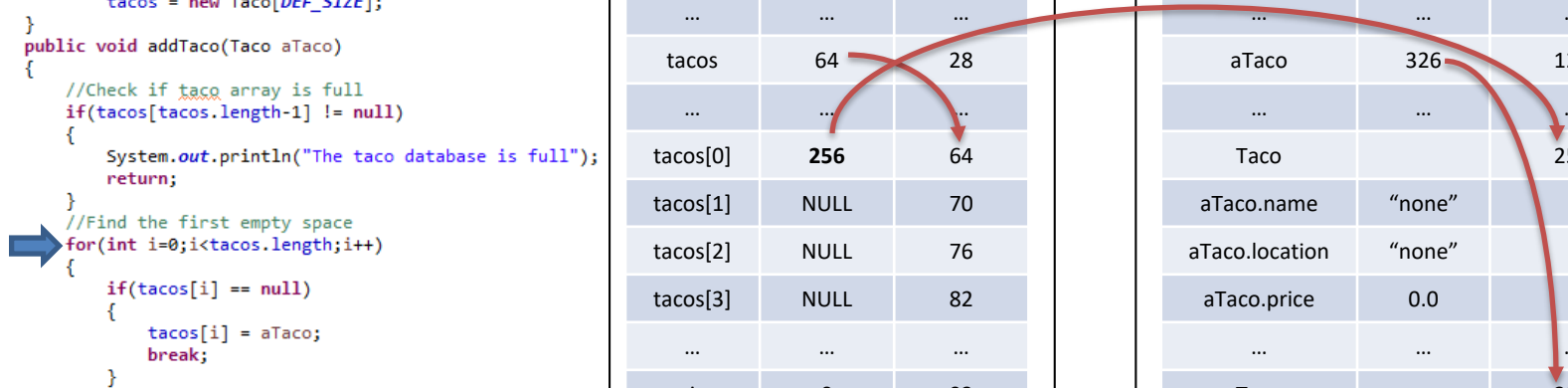
```

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
}

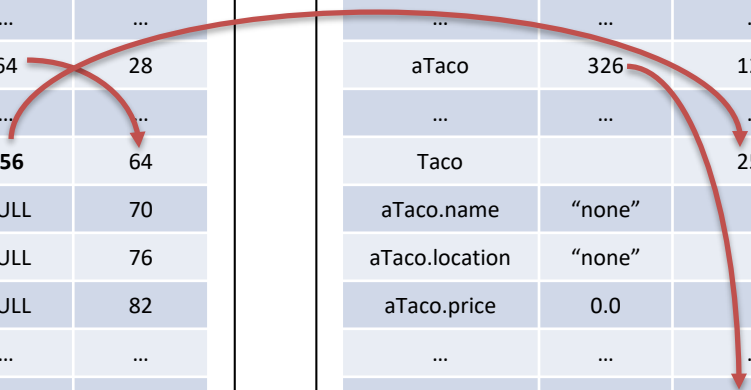
```

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

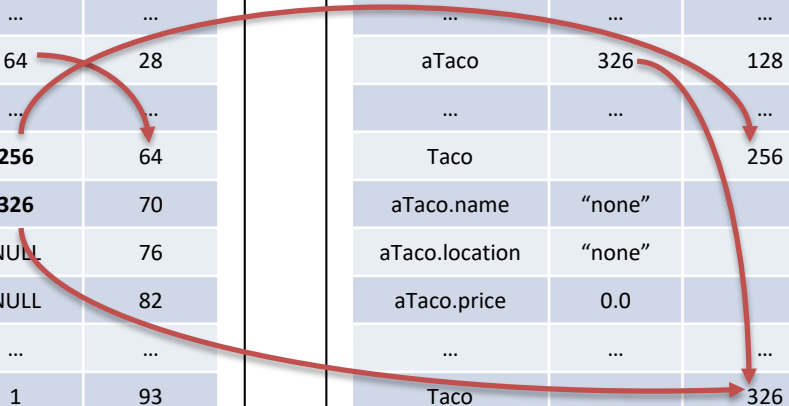
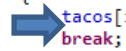
```

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

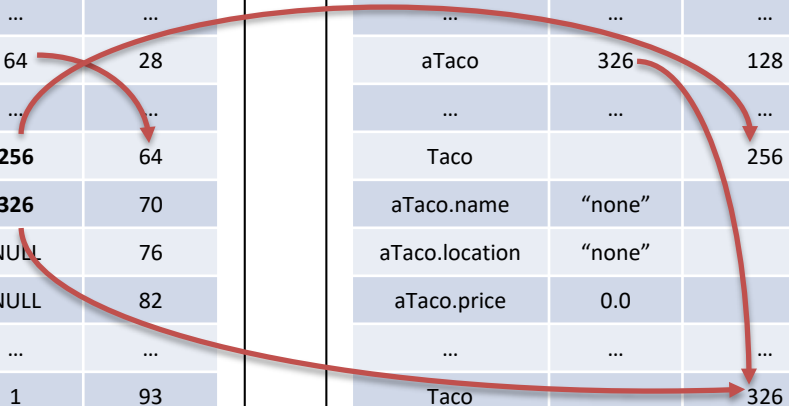
```

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

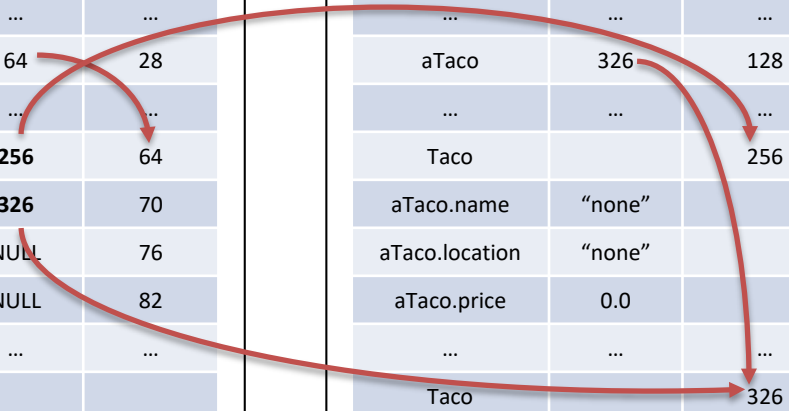
```

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
}

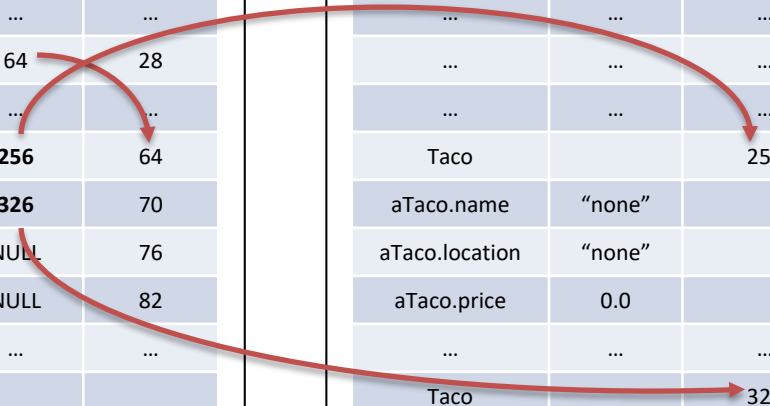
```

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

"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	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

```

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;
    }
}
    
```

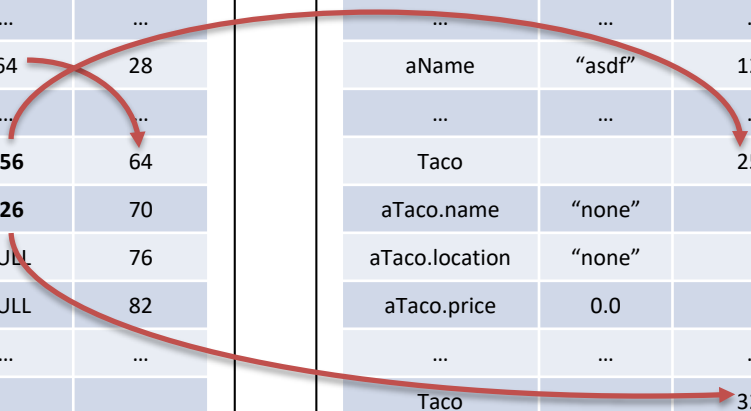
Diagram: A box containing "asdf" has an arrow pointing to the parameter `aName` in the function signature. A blue arrow points to the start of the `removeTaco` function.

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

```

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;
    }
}

```

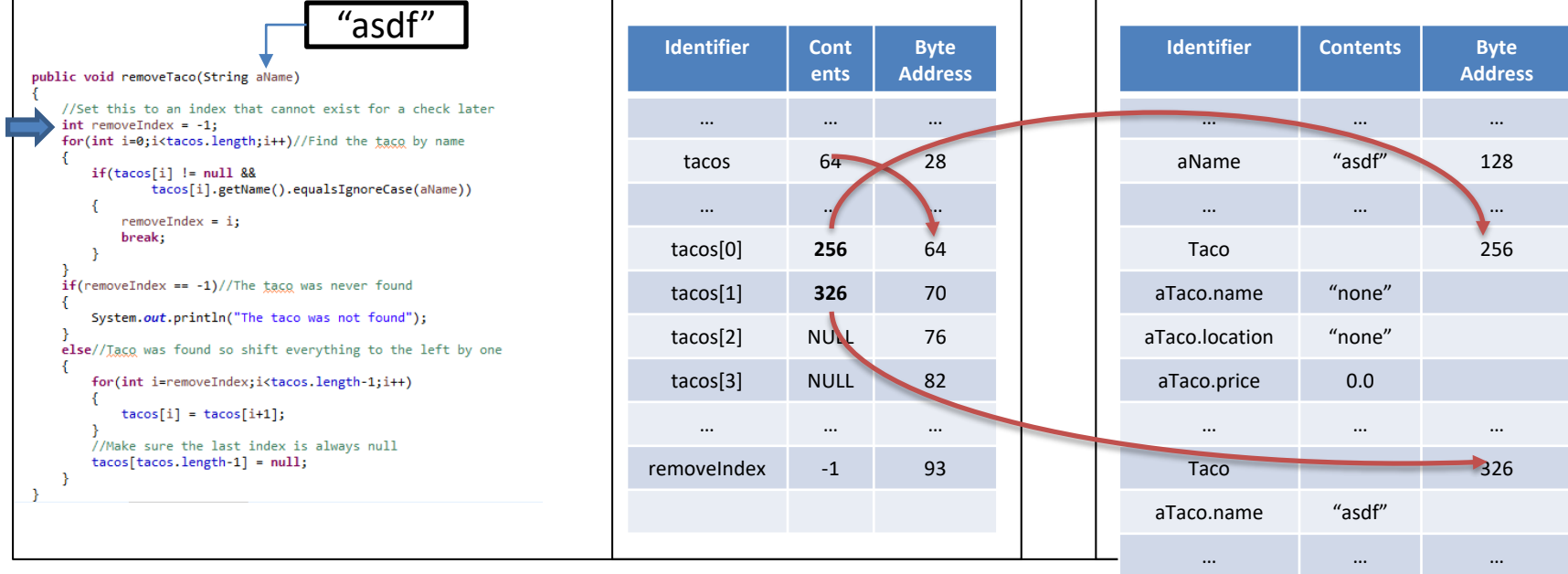
“asdf”

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”

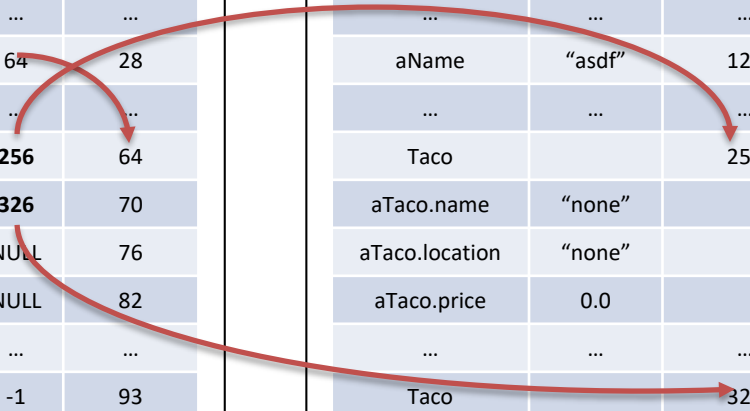


Memory

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	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;
    }
}

```

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"	
...

Diagram illustrating the removal of a taco from a list. The code snippet shows a loop that finds the index of the taco to be removed (aName = "asdf"). The memory table shows the state of the array (tacos) and the index (removeIndex) after the removal. The 'More Memory' table shows the state of the array (tacos) and the index (removeIndex) after the removal, with the removed taco's memory address (326) highlighted.

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;
    }
}
    
```

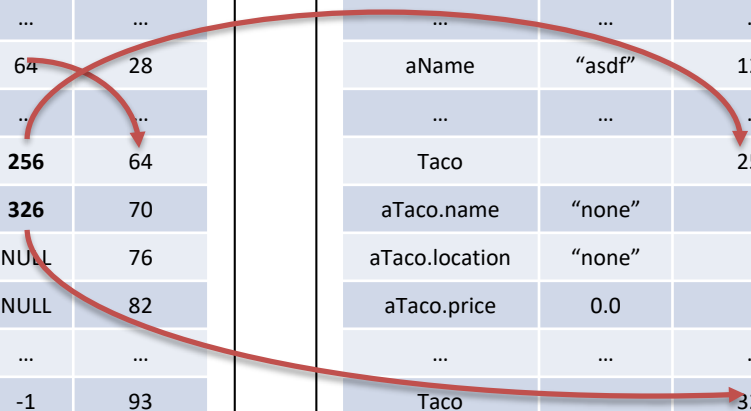
Diagram: A box containing "asdf" has an arrow pointing to the `aName` parameter in the code. A blue arrow points to the `break;` statement in the code.

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;
    }
}

```

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"	
...

Diagram illustrating the removal of a taco from a list. The code on the left shows the logic: finding the index of the taco to be removed (indicated by a blue arrow pointing to the `removeIndex` variable) and then shifting the remaining elements one position to the left. The "asdf" string is highlighted in a box.

The "Memory" table shows the state of memory before removal. The `tacos` array has 4 elements. The `removeIndex` is -1, and `i` is 1. Red arrows indicate the shift of elements from `tacos[1]` to `tacos[0]`, and from `tacos[2]` to `tacos[1]`, and from `tacos[3]` to `tacos[2]`.

The "More Memory" table shows the state of memory after removal. The `tacos` array now has 3 elements. The `removeIndex` is 326, and `aTaco.name` is "asdf". Red arrows indicate the shift of elements from `tacos[2]` to `tacos[1]`, and from `tacos[3]` to `tacos[2]`.

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;
    }
}

```

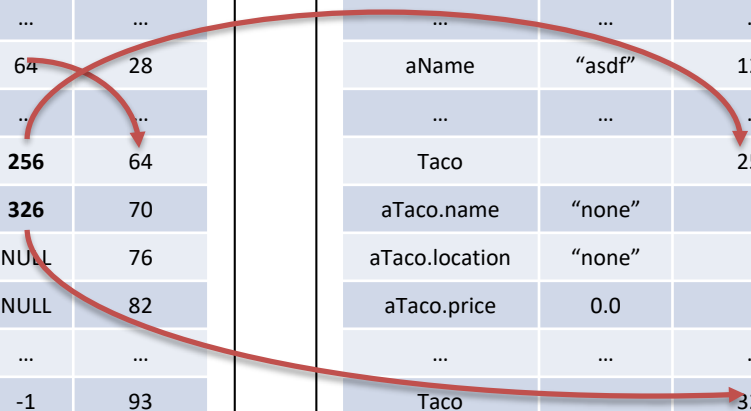
“asdf”

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;
    }
}

```

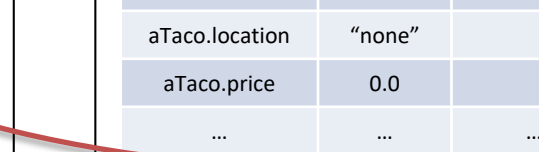
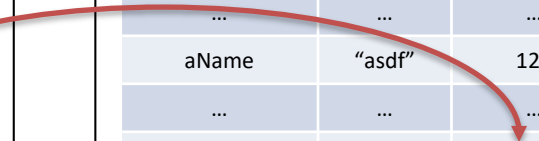
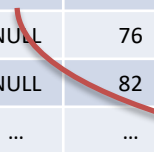
“asdf”

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;
    }
}

```

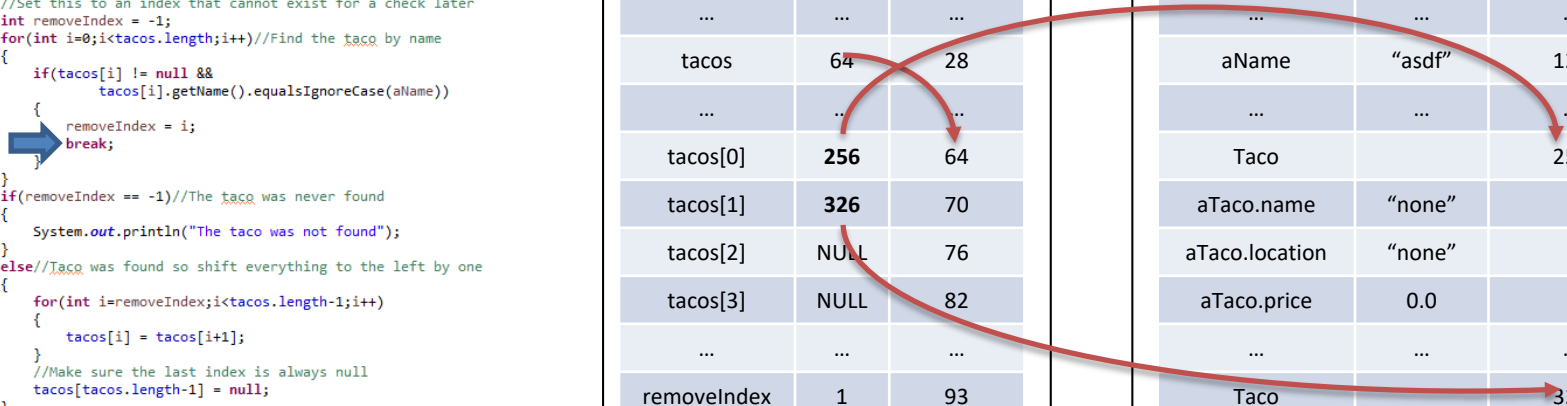
“asdf”

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

```

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	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"	
...

Diagram illustrating the removal of a taco from a list. The code on the left shows the logic: finding the index of the taco to be removed (indicated by a blue arrow pointing to the `removeIndex` assignment) and then shifting the remaining elements to the left. The `removeIndex` is set to 1, and the taco at index 1 is removed. The `removeIndex` is then updated to 1, and the taco at index 1 is removed. The `removeIndex` is then updated to 1, and the taco at index 1 is removed.

The **Memory** table shows the state of memory after the removal. The `removeIndex` is 1, and the taco at index 1 is removed. The `removeIndex` is then updated to 1, and the taco at index 1 is removed. The `removeIndex` is then updated to 1, and the taco at index 1 is removed.

The **More Memory** table shows the state of memory after the removal. The `removeIndex` is 1, and the taco at index 1 is removed. The `removeIndex` is then updated to 1, and the taco at index 1 is removed. The `removeIndex` is then updated to 1, and the taco at index 1 is removed.

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	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"	
...

Diagram illustrating the memory state during the removal of a taco. The code on the left shows the logic for finding a taco by name and shifting elements to the left. The "Memory" table shows the state before removal, with the taco at index 1 (address 70) and the removal index at 1. The "More Memory" table shows the state after removal, with the taco moved to index 3 (address 326) and the removal index updated to 3. Red arrows indicate the shift of elements and the update of the removal index.

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;
    }
}

```

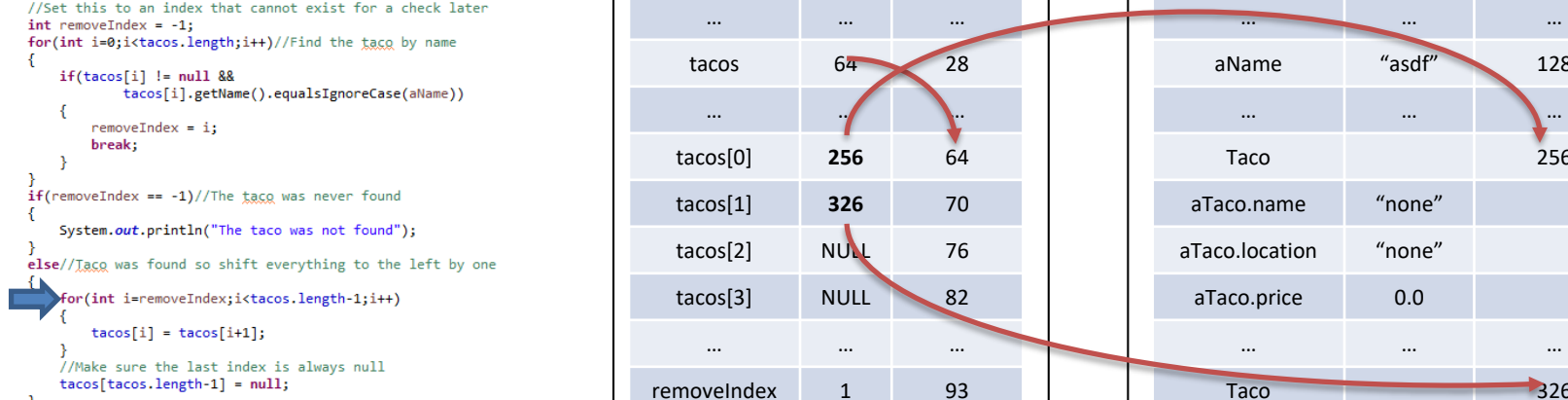
"asdf"

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

"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	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 `removeTaco` method. Another arrow points from the `removeIndex` variable to the `for` loop that shifts elements to the left.

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

Diagram: Red arrows show the shift of memory addresses. An arrow points from the `tacos` entry to `tacos[0]`, and another from `tacos[0]` to `tacos[1]`, illustrating the leftward shift of elements.

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"	
...

Diagram: A red arrow points from the `tacos` entry in the 'Memory' table to the `Taco` entry in the 'More Memory' table, indicating the state of the object after removal.

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	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;
    }
}
    
```

“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;
    }
}

```

"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
...
...
...
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]	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	
...

Split for Strings

- The “split” method separates a String into an Array of Strings given delimiters
 - Start of String to first delimiter = Index 0
 - After the first delimiter to the next one = Index 1
 - After the second delimiter to the next one = Index 2
 - ...
 - After the last delimiter to the end of the String = Last Index
- Uses a Regular Expression (REGEX)

Example

```
//Assume str = “name1\tlocation1\t1.0\n”  
String[] splitStr = str.split(“\t”);
```

Array

splitStr =

Split for Strings

- The “split” method separates a String into an Array of Strings given delimiters
 - Start of String to first delimiter = Index 0
 - After the first delimiter to the next one = Index 1
 - After the second delimiter to the next one = Index 2
 - ...
 - After the last delimiter to the end of the String = Last Index
- Uses a Regular Expression (REGEX)

Example

```
//Assume str = "name1\tlocation1\t1.0\n"  
String[] splitStr = str.split("\t");
```

Array

splitStr =

Index	0	1	2
Value			

Split for Strings

- The “split” method separates a String into an Array of Strings given delimiters
 - Start of String to first delimiter = Index 0
 - After the first delimiter to the next one = Index 1
 - After the second delimiter to the next one = Index 2
 - ...
 - After the last delimiter to the end of the String = Last Index
- Uses a Regular Expression (REGEX)

Example

```
//Assume str = "name1\tlocation1\t1.0\n"  
String[] splitStr = str.split("\t");
```

Array

splitStr =

Index	0	1	2
Value			

Split for Strings

- The “split” method separates a String into an Array of Strings given delimiters
 - Start of String to first delimiter = Index 0
 - After the first delimiter to the next one = Index 1
 - After the second delimiter to the next one = Index 2
 - ...
 - After the last delimiter to the end of the String = Last Index
- Uses a Regular Expression (REGEX)

Example

```
//Assume str = "name1\tlocation1\t1.0\n"  
String[] splitStr = str.split("\t");
```

Array

splitStr =

Index	0	1	2
Value	"name1"		

Split for Strings

- The “split” method separates a String into an Array of Strings given delimiters
 - Start of String to first delimiter = Index 0
 - After the first delimiter to the next one = Index 1
 - After the second delimiter to the next one = Index 2
 - ...
 - After the last delimiter to the end of the String = Last Index
- Uses a Regular Expression (REGEX)

Example

```
//Assume str = "name1\tlocation1\t1.0\n"  
String[] splitStr = str.split("\t");
```

Array

splitStr =

Index	0	1	2
Value	"name1"	"location1"	

Split for Strings

- The “split” method separates a String into an Array of Strings given delimiters
 - Start of String to first delimiter = Index 0
 - After the first delimiter to the next one = Index 1
 - After the second delimiter to the next one = Index 2
 - ...
 - After the last delimiter to the end of the String = Last Index
- Uses a Regular Expression (REGEX)

Example

```
//Assume str = “name1\tlocation1\t1.0\n”  
String[] splitStr = str.split(“\t”);
```

Array

splitStr =

Index	0	1	2
Value	“name1”	“location1”	“1.0”

Split for Strings

- The “split” method separates a String into an Array of Strings given delimiters
 - Start of String to first delimiter = Index 0
 - After the first delimiter to the next one = Index 1
 - After the second delimiter to the next one = Index 2
 - ...
 - After the last delimiter to the end of the String = Last Index
- Uses a Regular Expression (REGEX)

Example

```
//Assume str = "name1\tlocation1\t1.0\n"  
String[] splitStr = str.split("\t");
```

Array

splitStr =

Index	0	1	2
Value	"name1"	"location1"	"1.0"