File Edit View Help

File Edit View Help

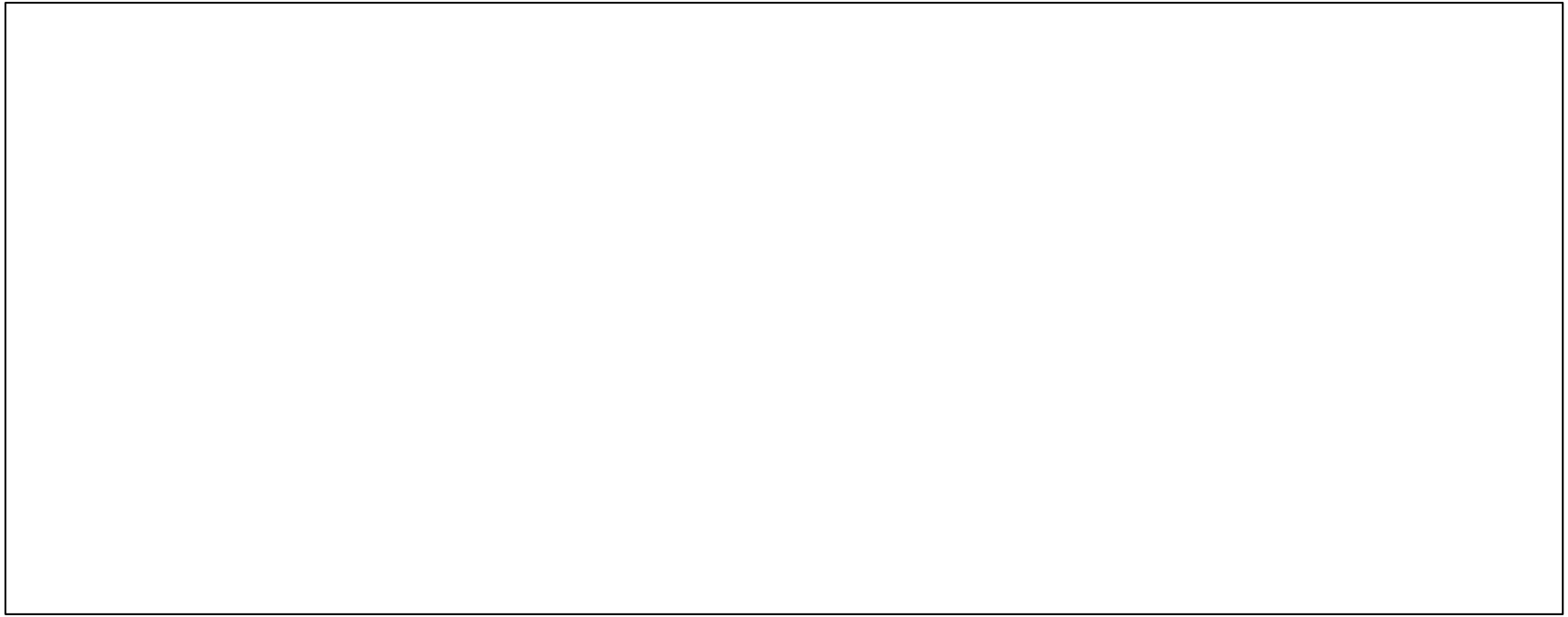
Even More Programming Review

File Edit View Help

Big Example

File Edit View Help

Problem



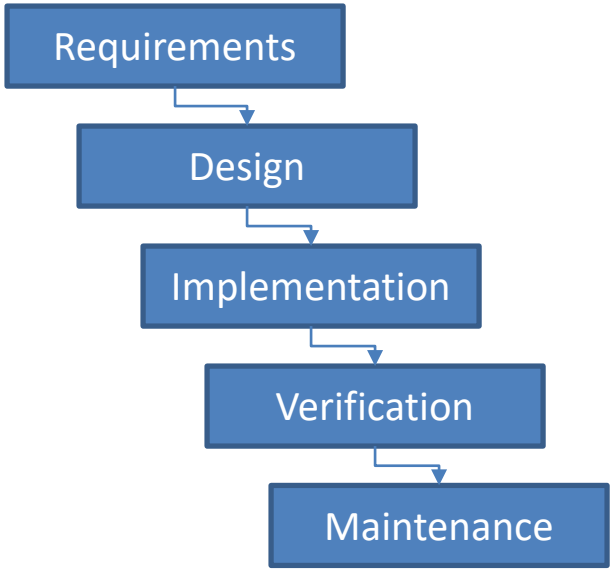
File Edit View Help

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



File Edit View Help

Requirements

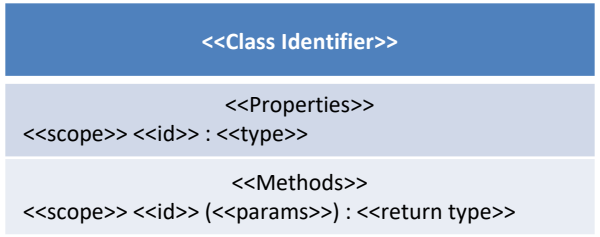
- Keep Track of important Taco Information
- Taco's Information
  - Name
  - Location
  - Price

- 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

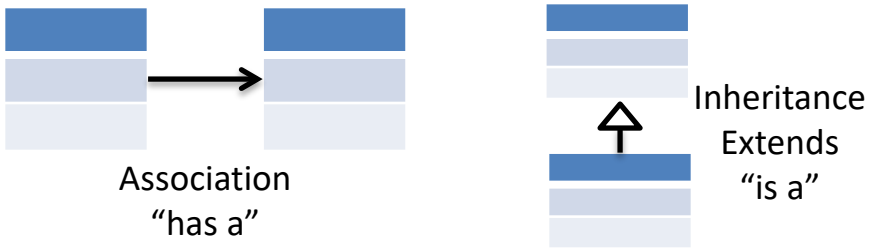
Design

- 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

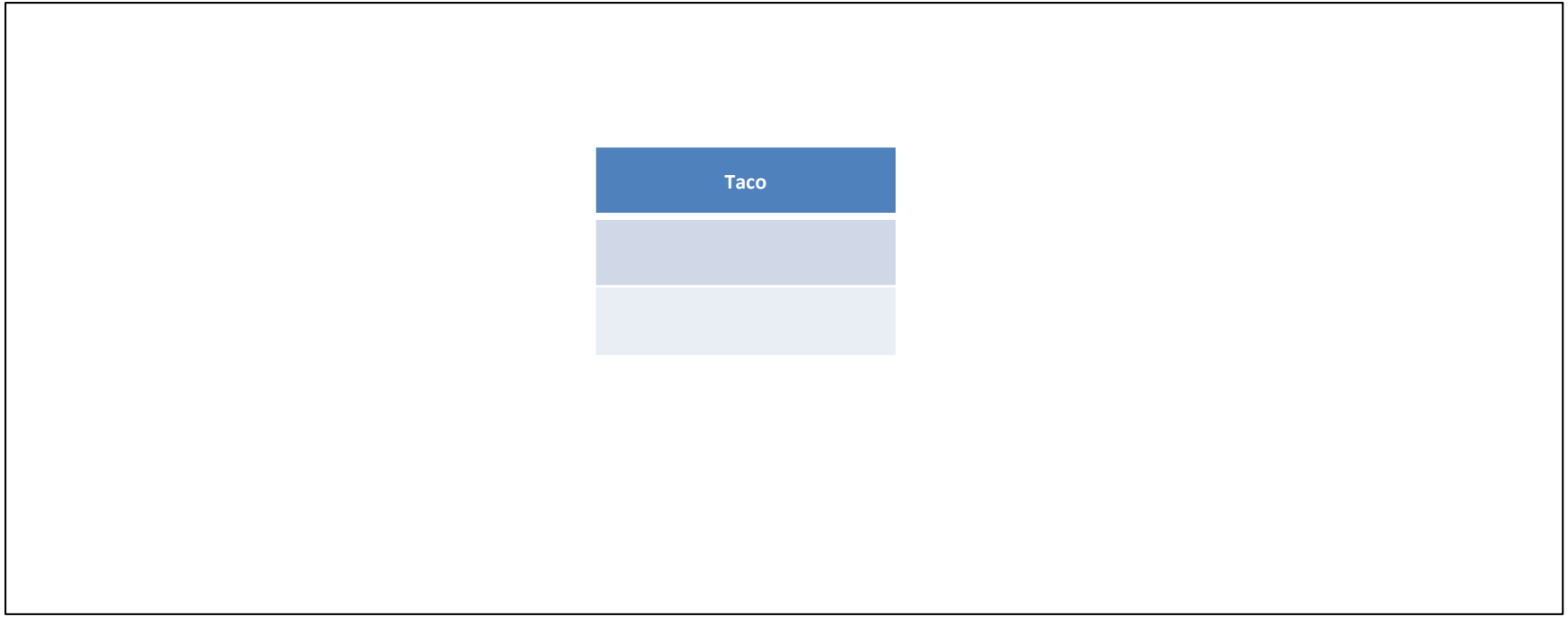


UML Class Diagram Arrows



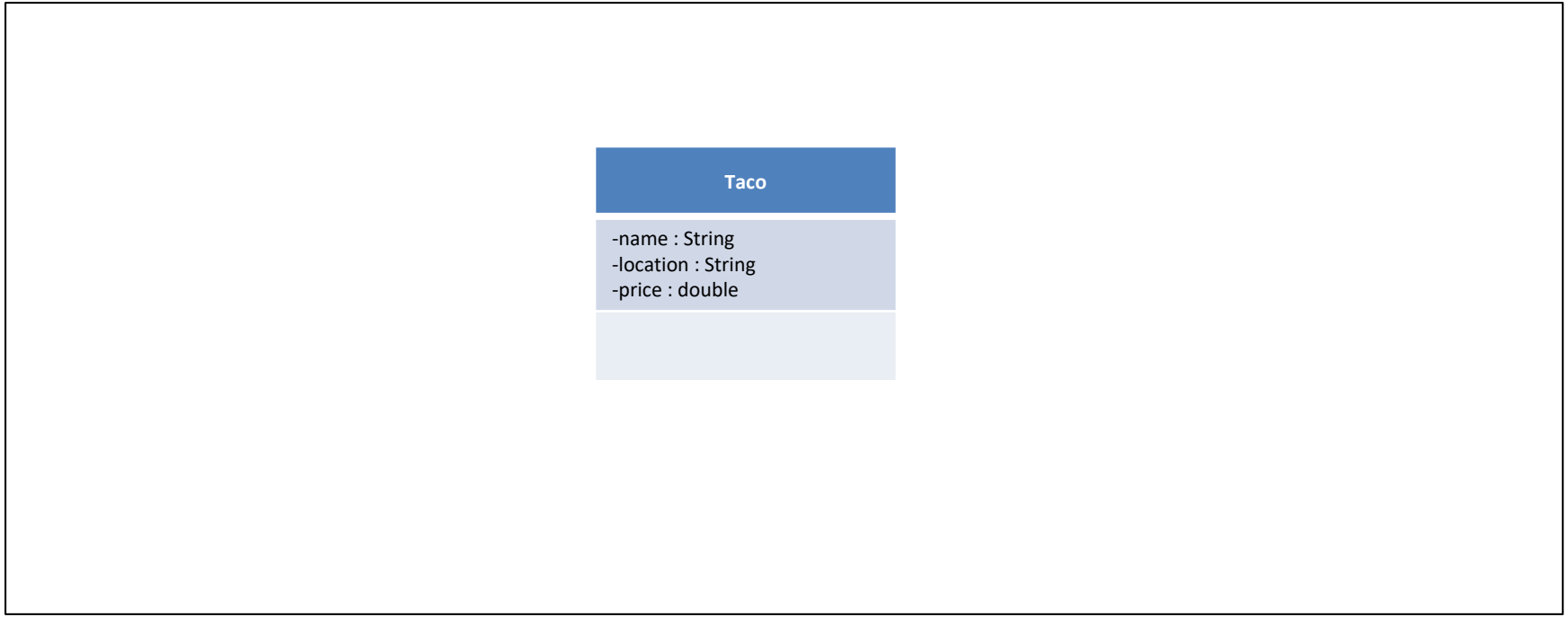
File Edit View Help

Design



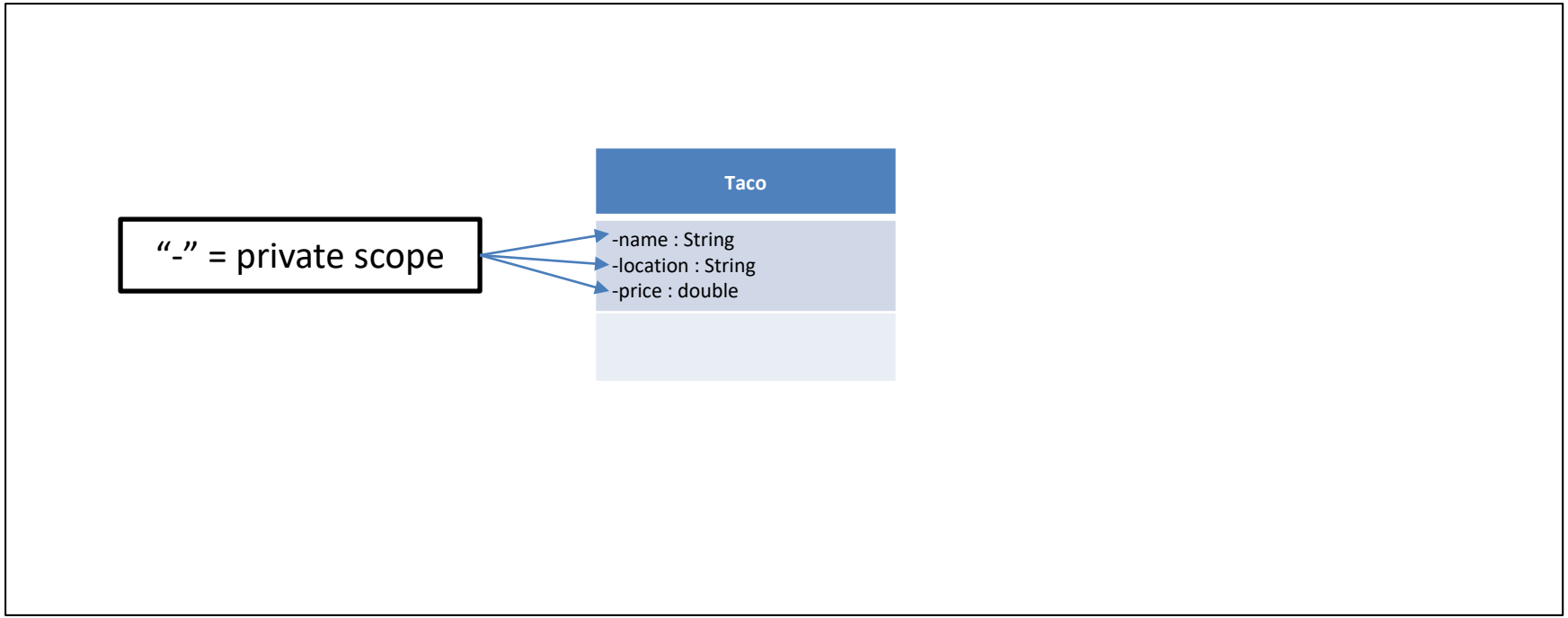
File Edit View Help

Design



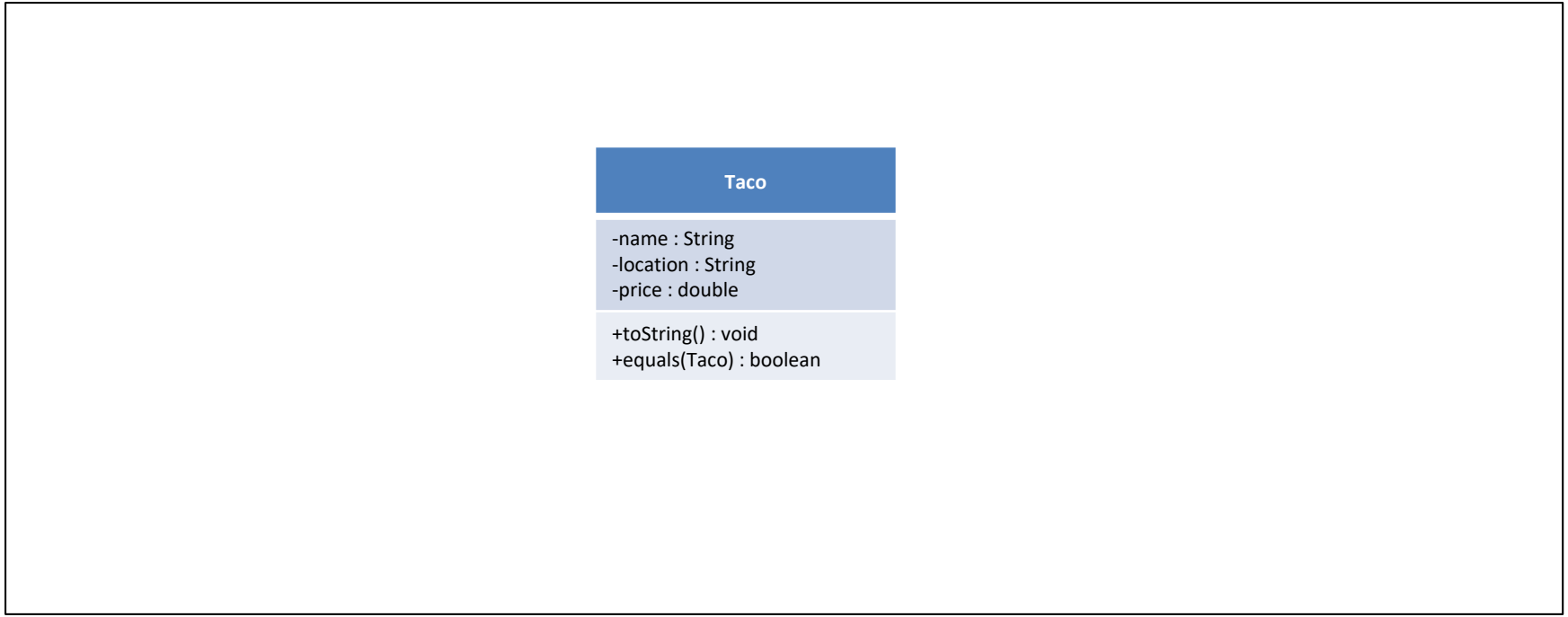
File Edit View Help

Design



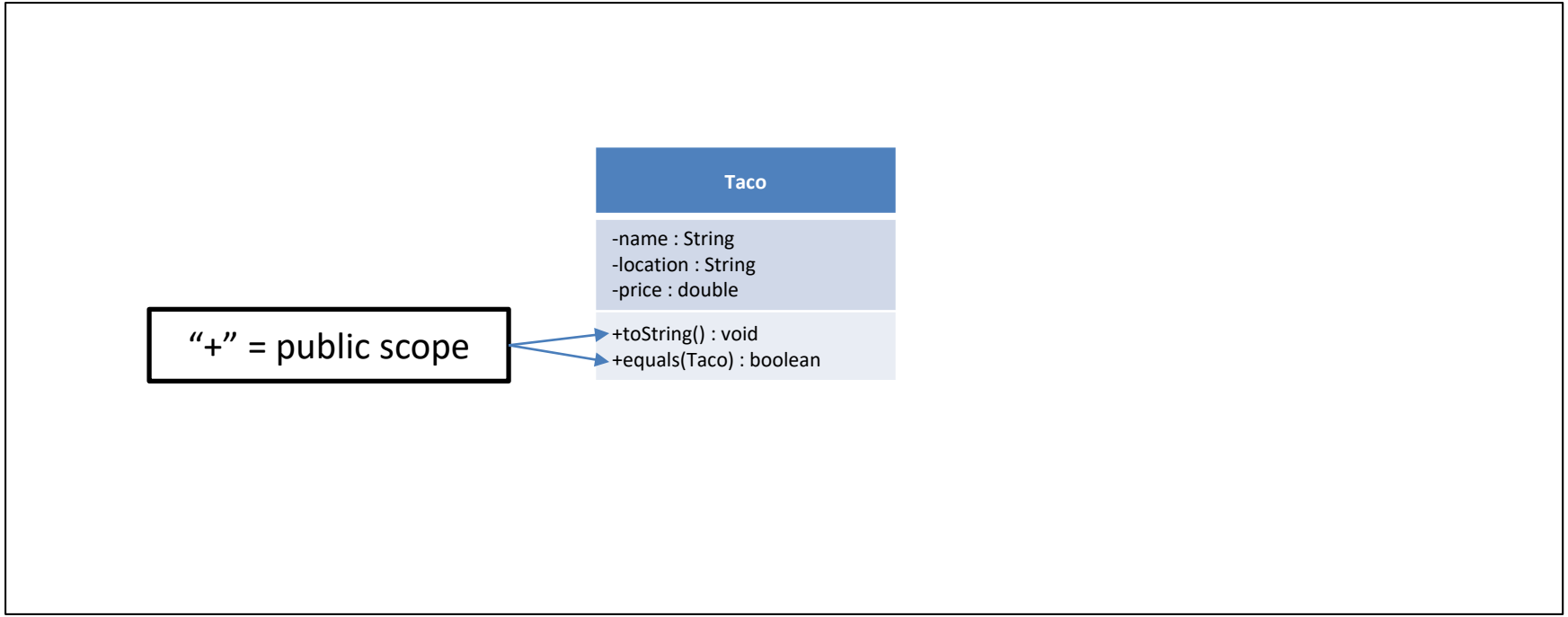
File Edit View Help

Design



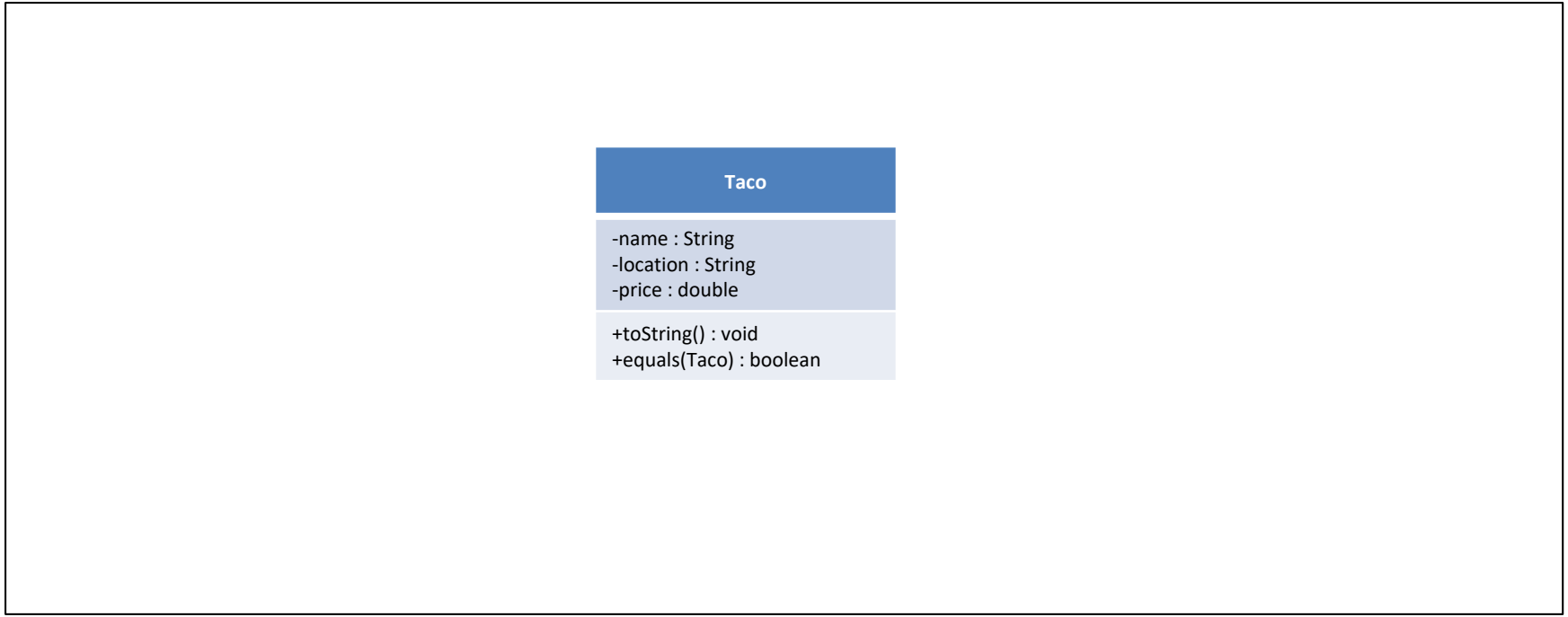
File Edit View Help

Design



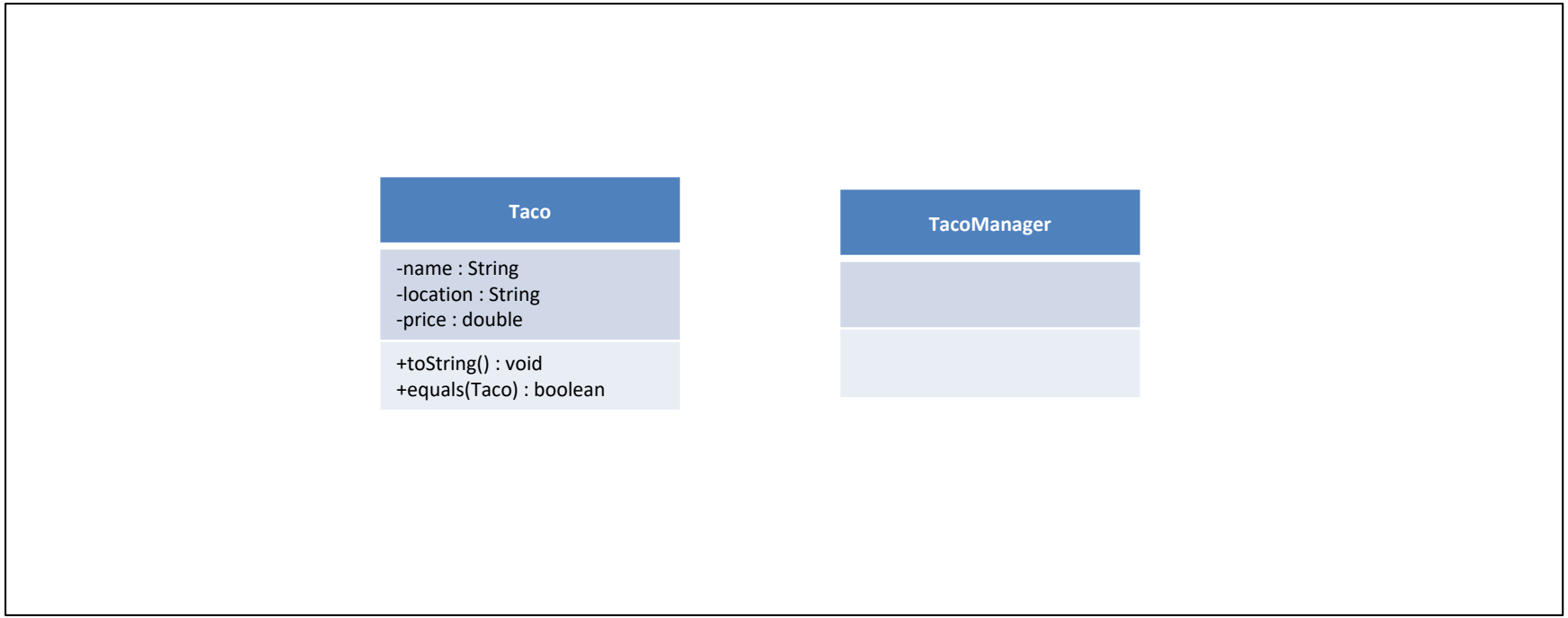
File Edit View Help

Design



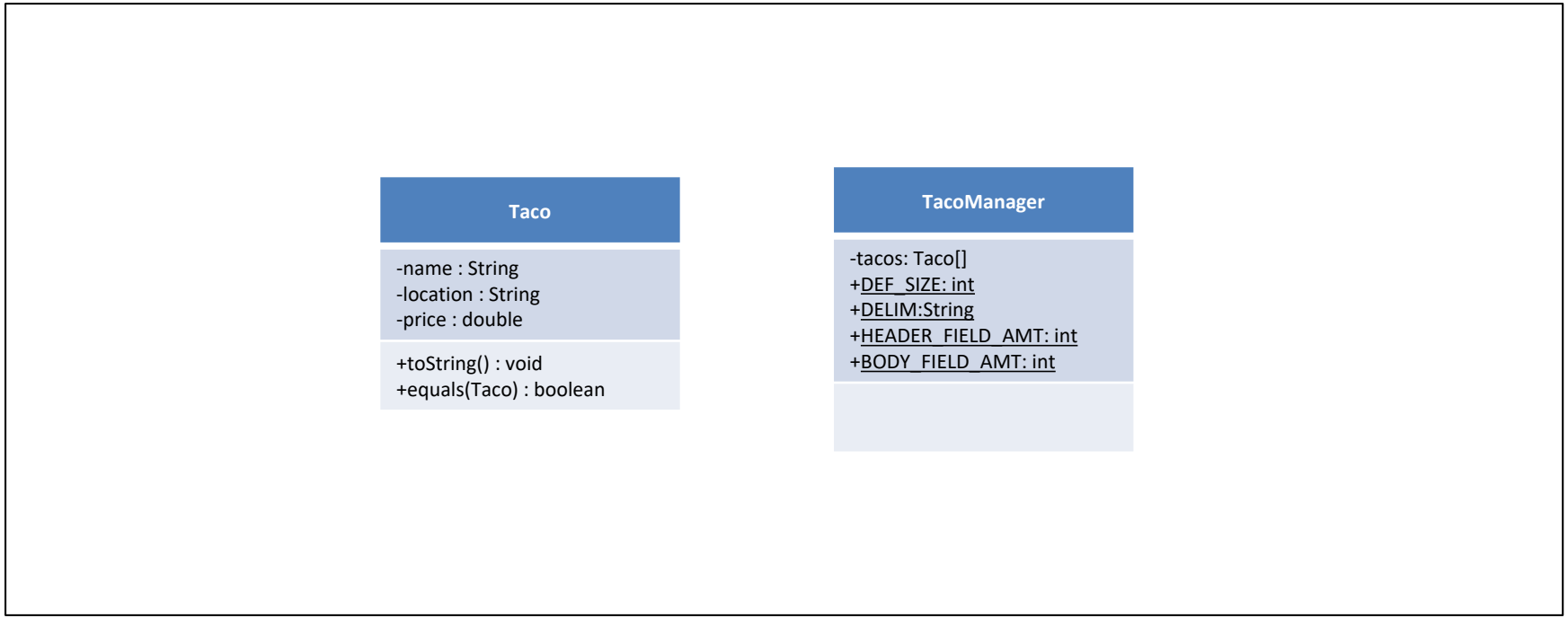
File Edit View Help

Design



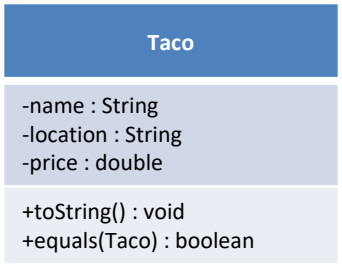
File Edit View Help

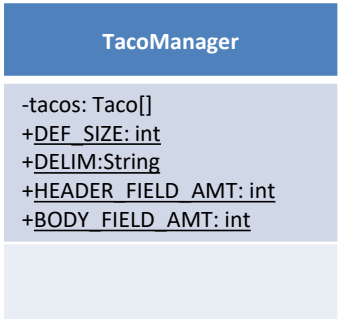
Design



File Edit View Help

Design





File Format

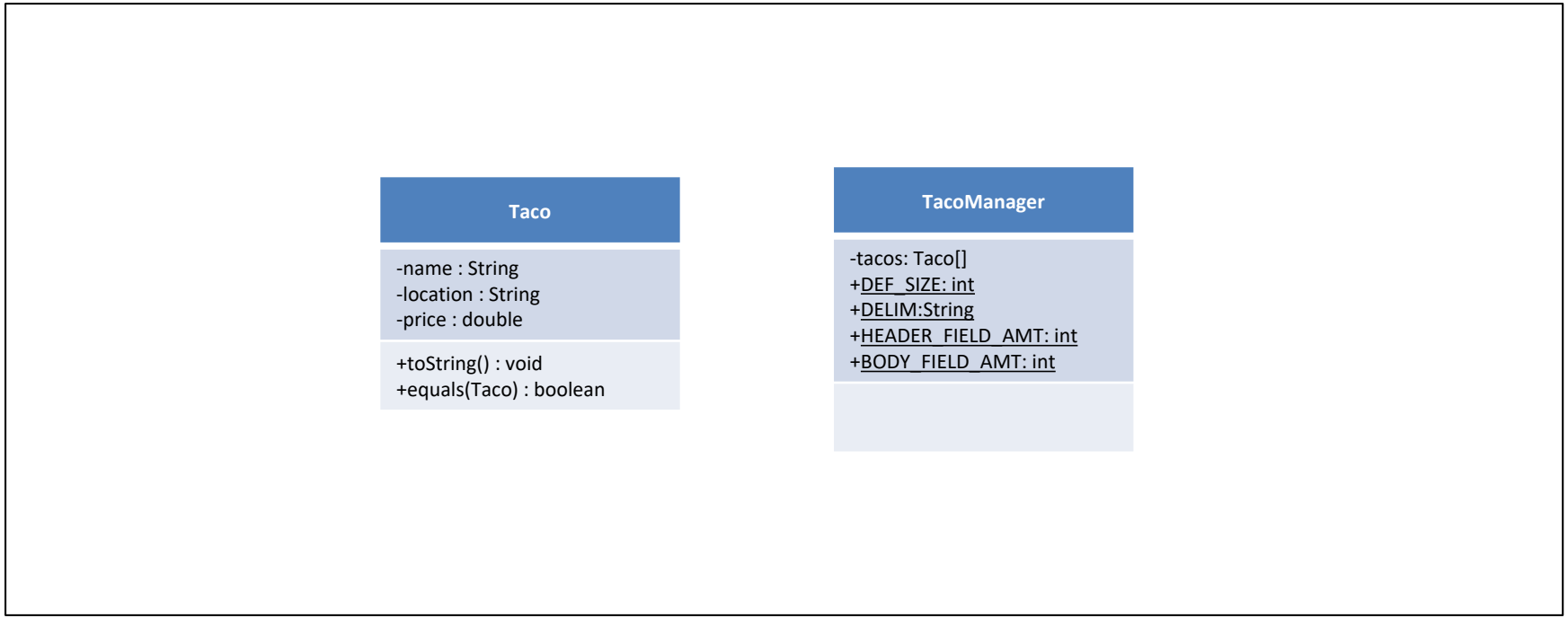
```
//Header
Taco Amt:\t<<number of tacos>>\n
//Body
<<name>>\t<<location>>\t<<price>>\n
…
```

Example

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

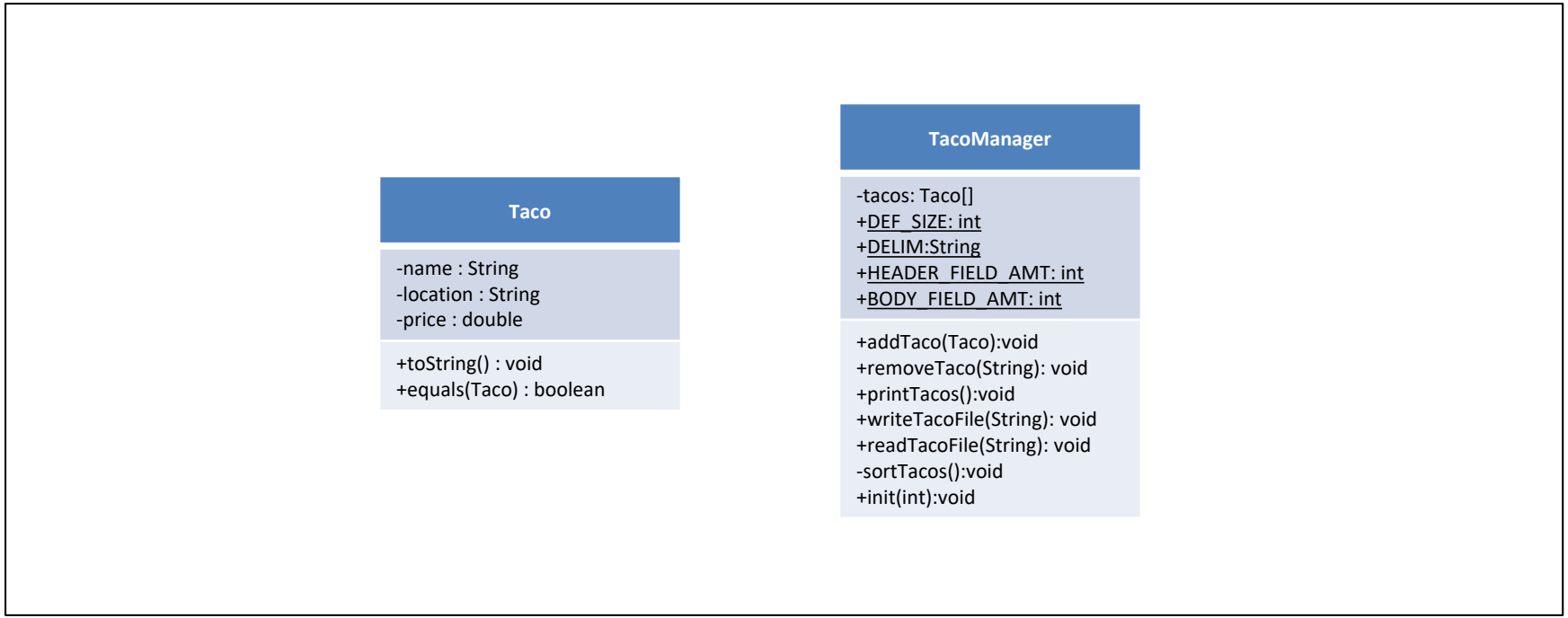
File Edit View Help

Design



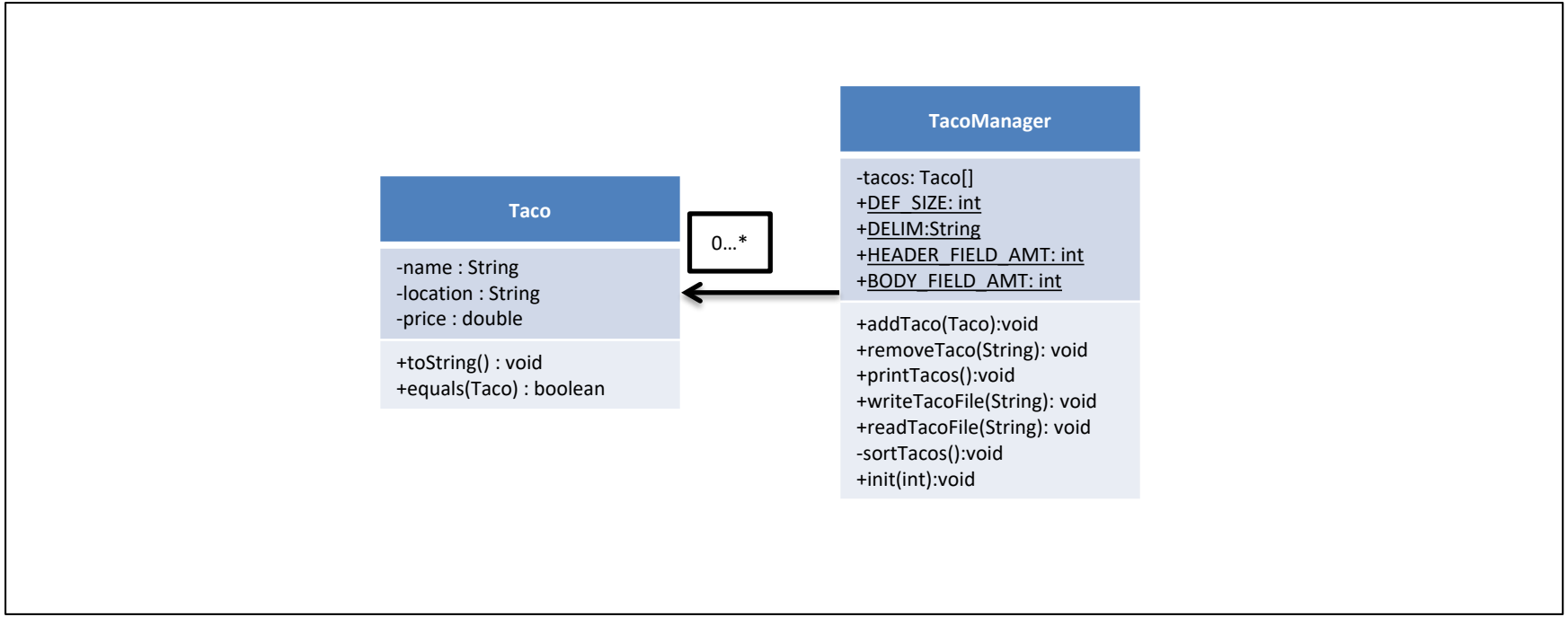
File Edit View Help

Design



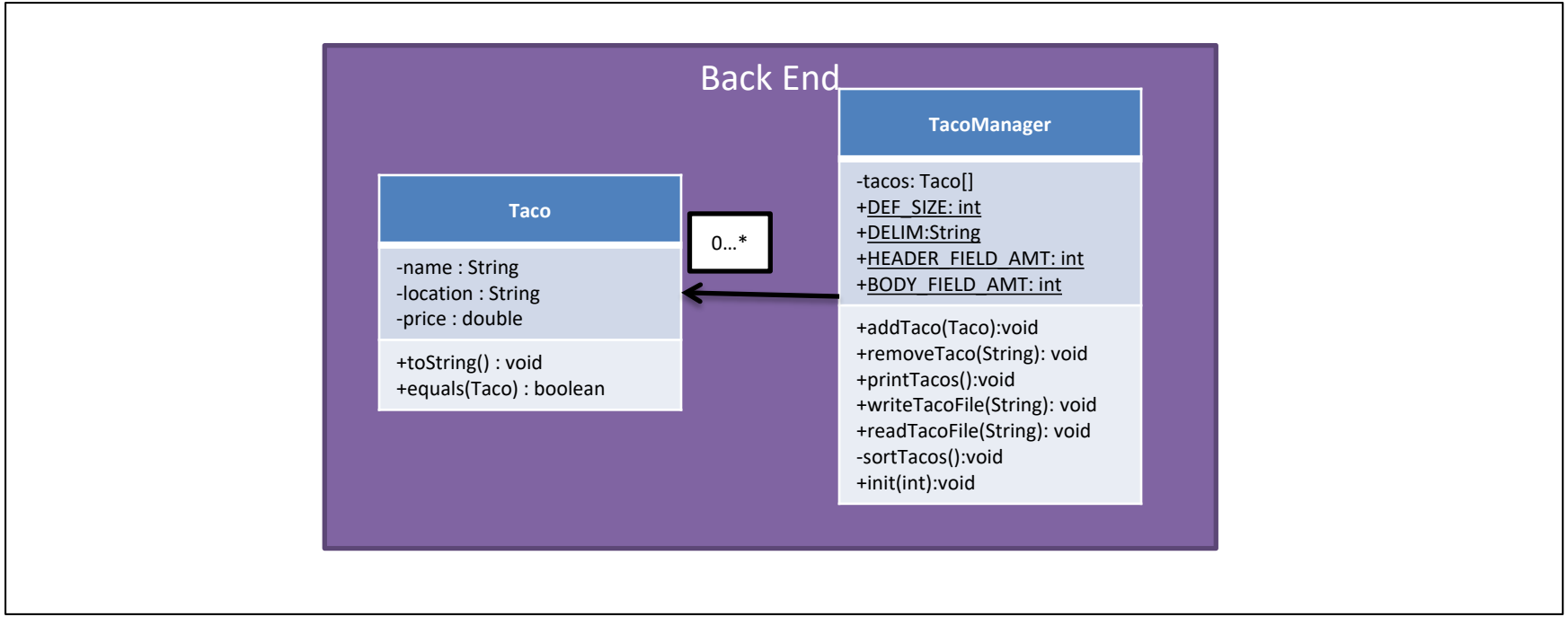
File Edit View Help

Design



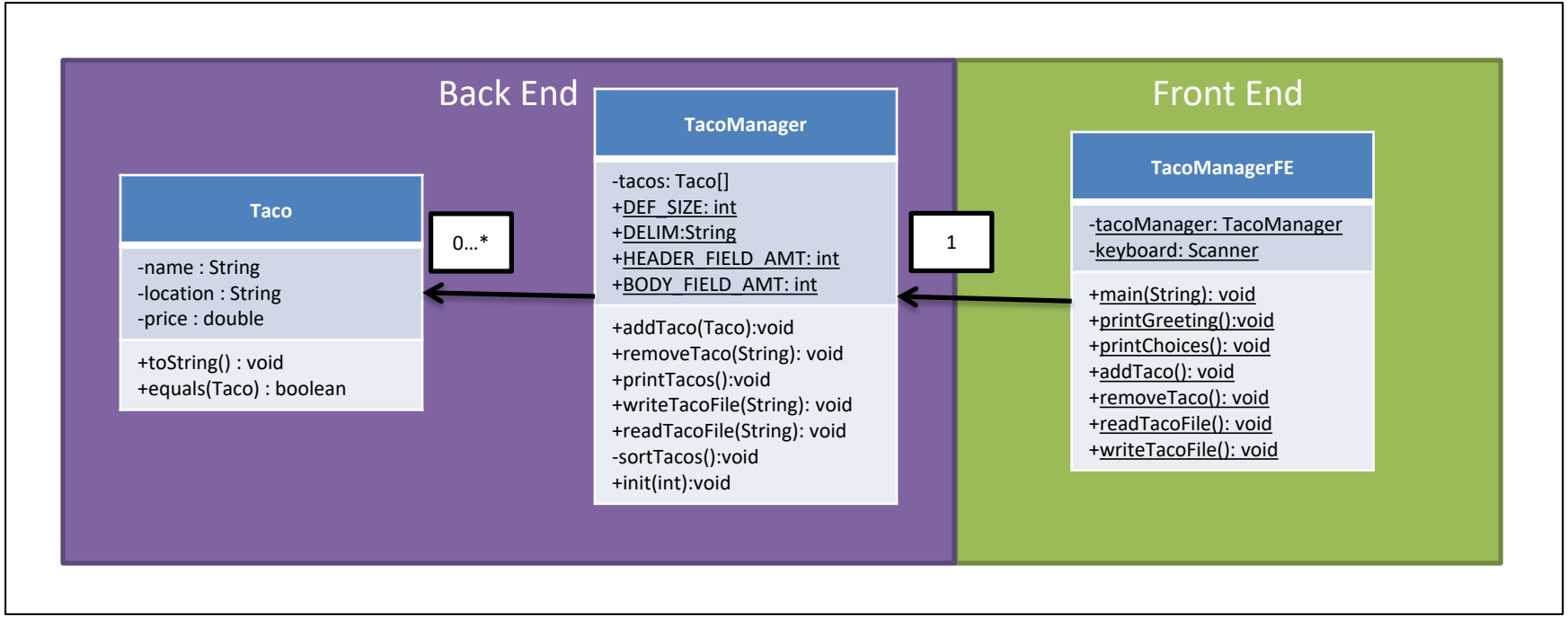
File Edit View Help

Design



File Edit View Help

Design

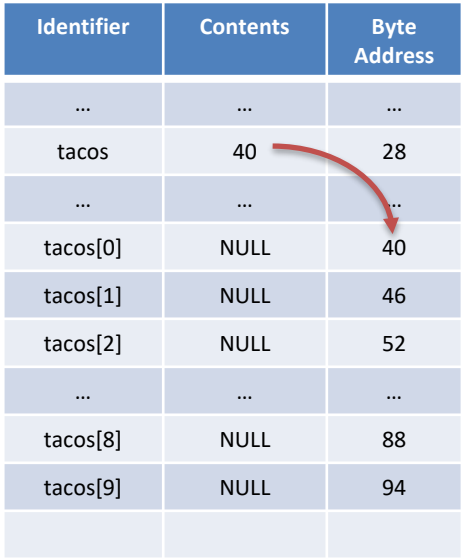


File Edit View Help

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
…	…	…

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

File Edit View Help

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

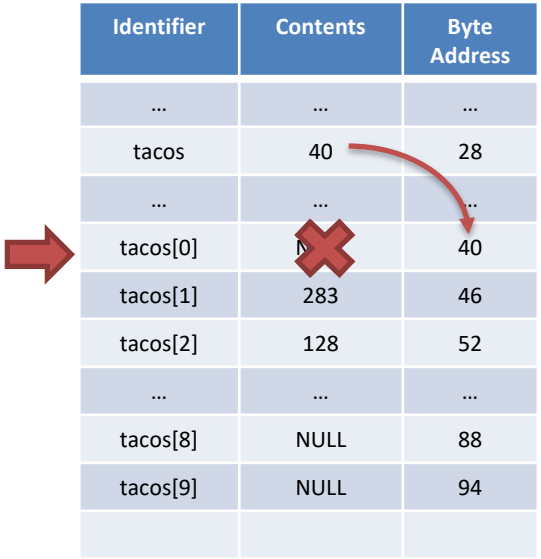
Taco Array Structure

- 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



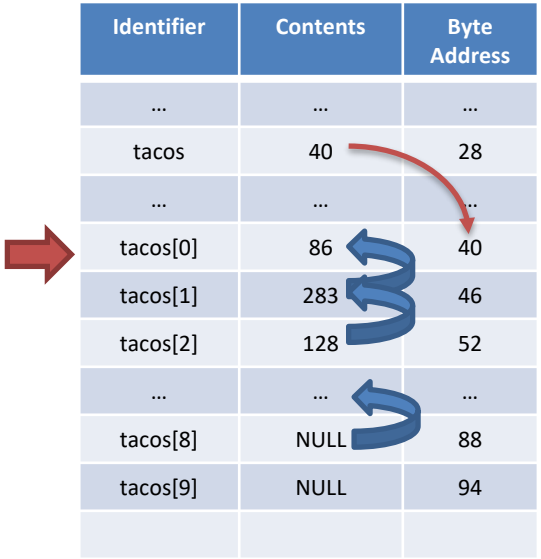
Taco Array Structure

- 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

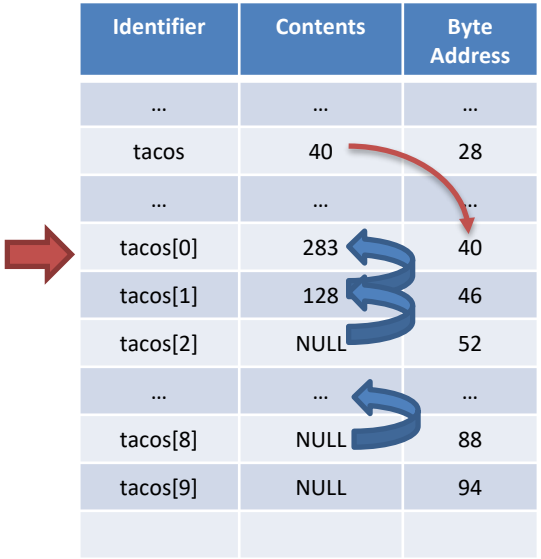
Taco Array Structure

- 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



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



File Edit View Help

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
```

Memory

Identifier	Contents	Byte Address
...	...	...
tacos	NULL	28

More Memory

Identifier	Contents	Byte Address
...	...	...

File Edit View Help

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	NULL	28

More Memory

Identifier	Contents	Byte Address
...	...	...

File Edit View Help

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	NULL	28

More Memory

Identifier	Contents	Byte Address
...	...	...

File Edit View Help

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	NULL	28

More Memory

Identifier	Contents	Byte Address
...	...	...

File Edit View Help

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	NULL	28

More Memory

Identifier	Contents	Byte Address
...	...	...

File Edit View Help

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	NULL	28
...	...	...
tacos[0]	NULL	64
tacos[1]	NULL	70
tacos[2]	NULL	76
tacos[3]	NULL	82
...	...	...

More Memory

Identifier	Contents	Byte Address
...	...	...

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
...	...	...

File Edit View Help

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
...	...	...

File Edit View Help

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
...	...	...

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

```
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

```
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

```
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

```
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
…	…	…
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

```
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
…	…	…
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

```
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
…	…	…
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

```
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
…	…	…
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

```
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]	**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

```
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]	**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

```
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]	**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

```
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]	**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

```
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]	**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

```
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]	**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

```
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]	**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

```
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]	**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

```
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]	**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
```

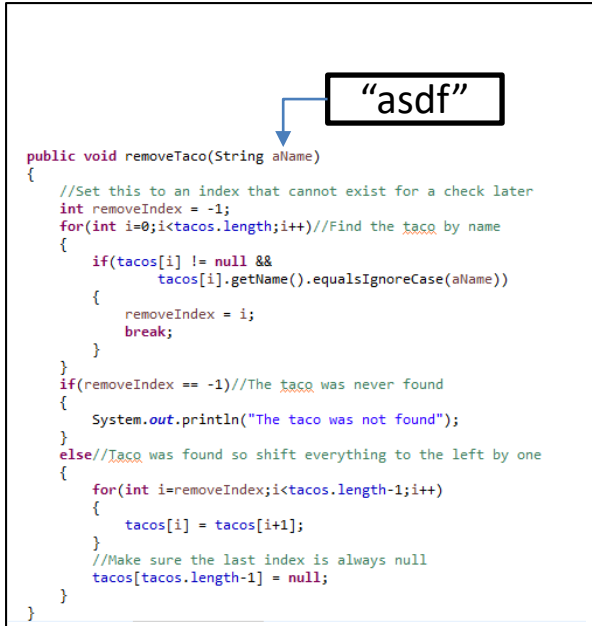
Memory

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



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

"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;
    }
}
```

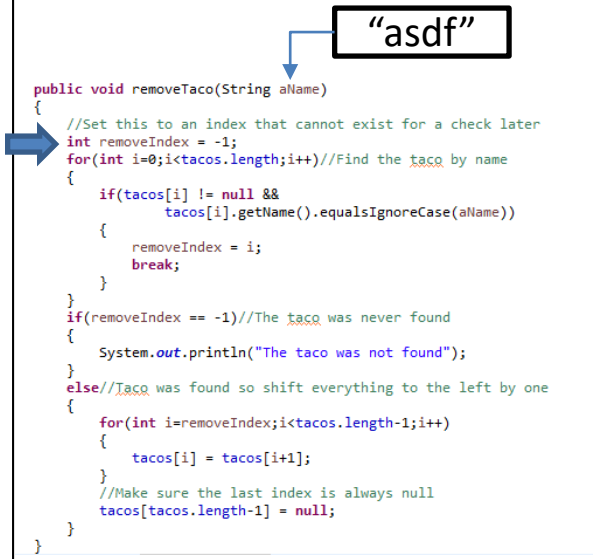
Memory

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



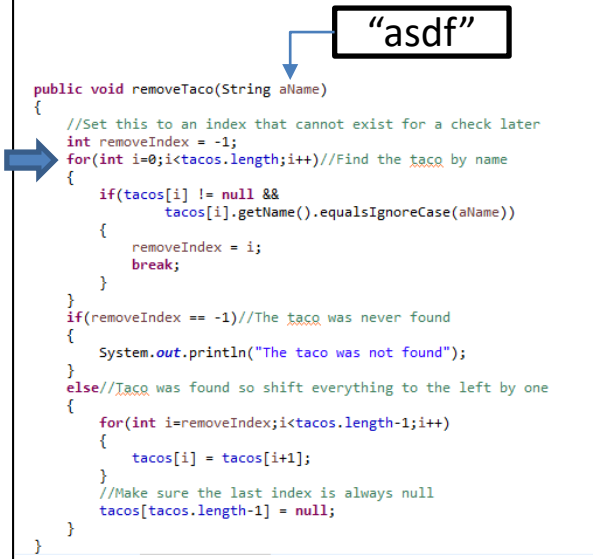
Memory

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



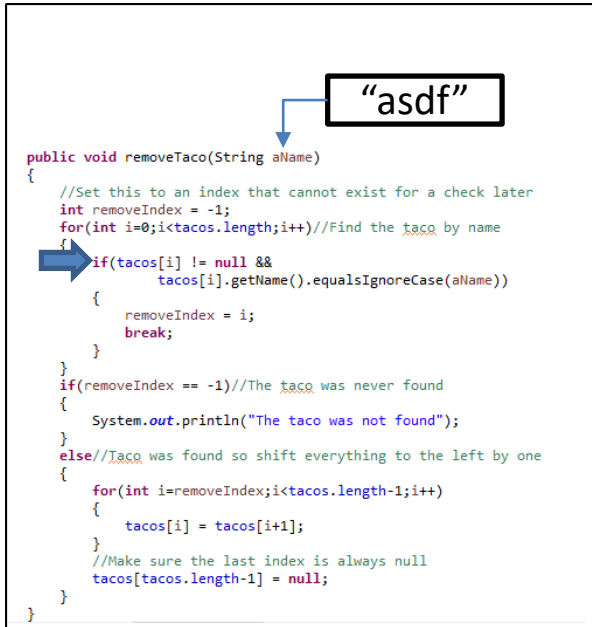
Memory

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
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



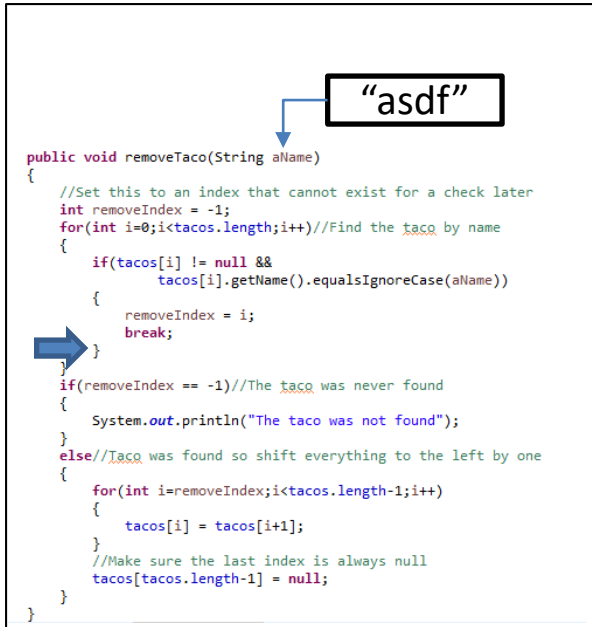
Memory

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
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



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
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

"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;
    }
}
```

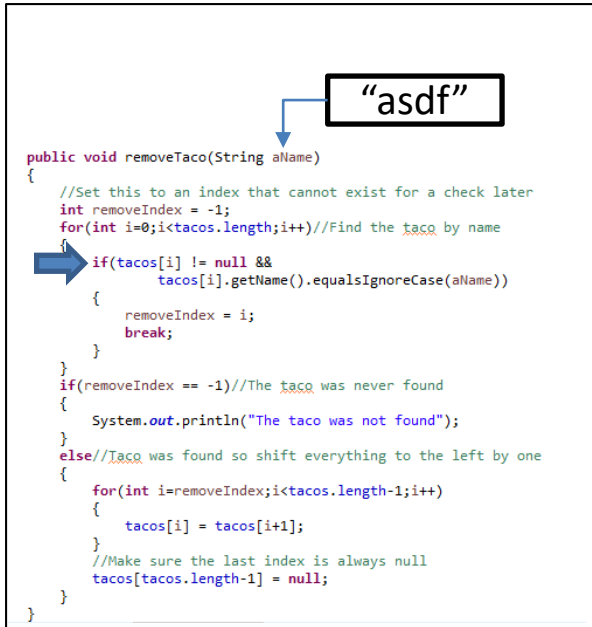
Memory

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



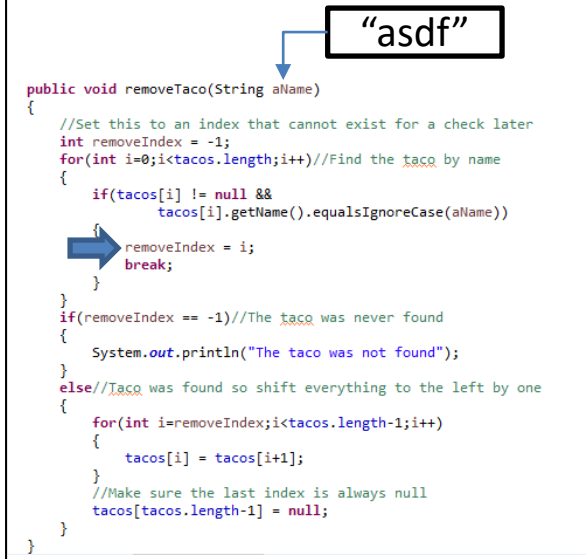
Memory

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
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



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
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;
    }
}
```

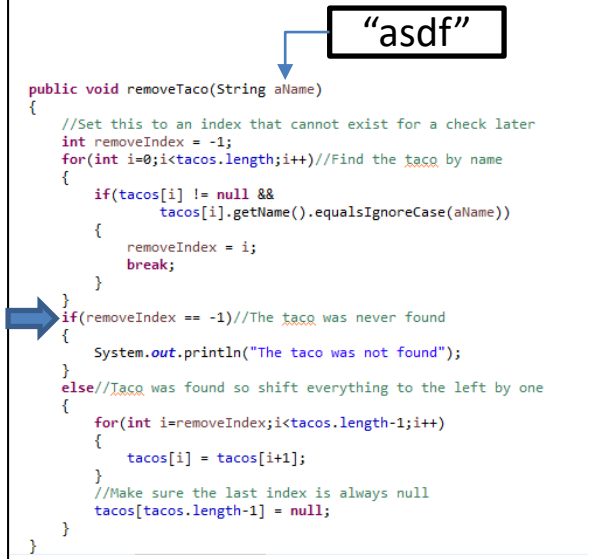
Memory

Identifier	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
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



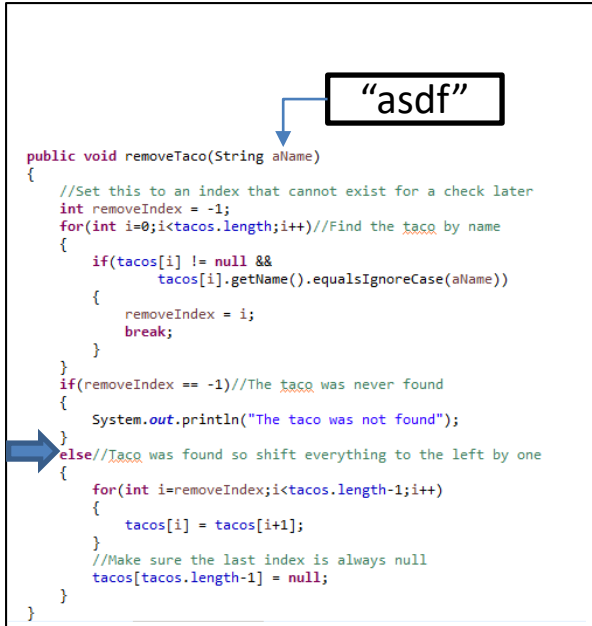
Memory

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



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

"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;
    }
}
```

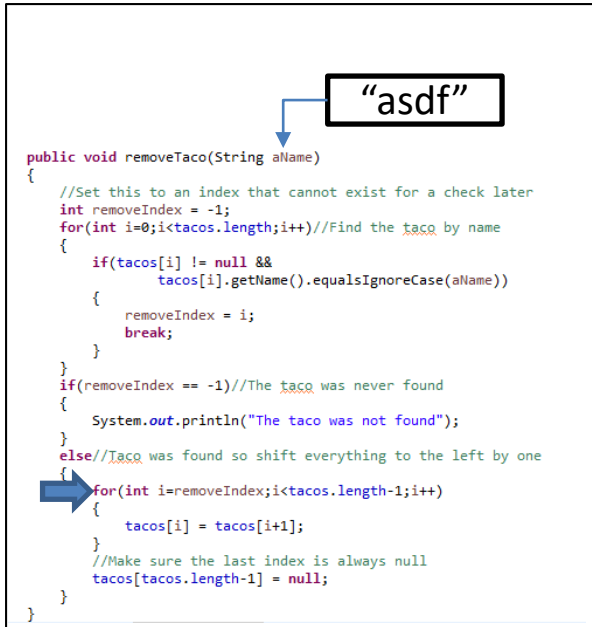
Memory

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



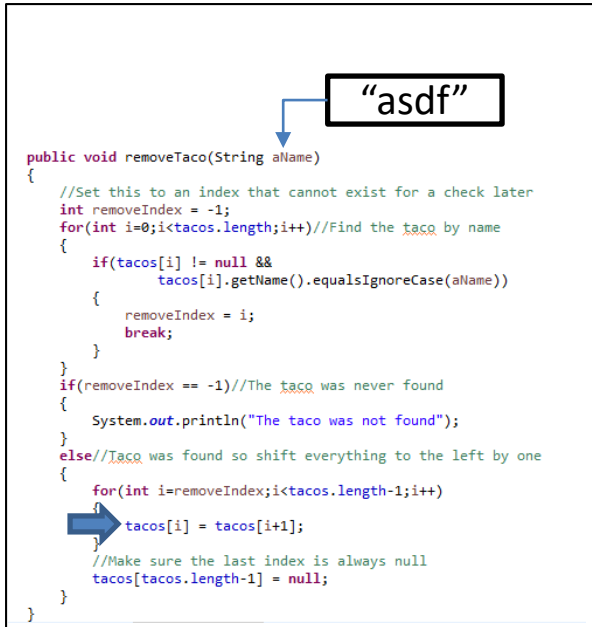
Memory

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
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



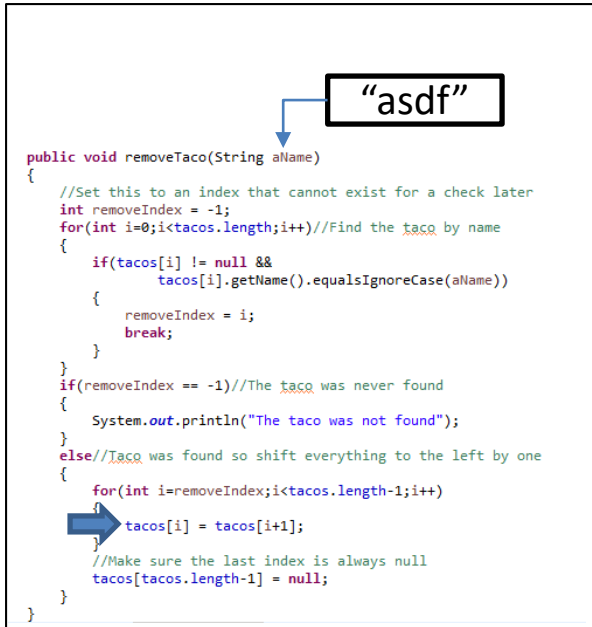
Memory

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
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



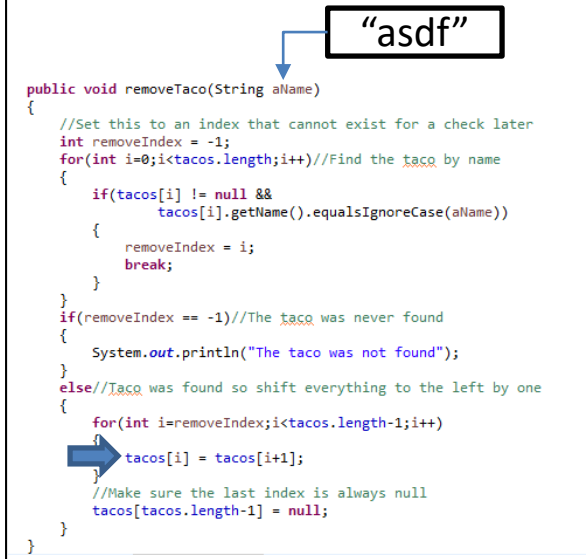
Memory

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	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



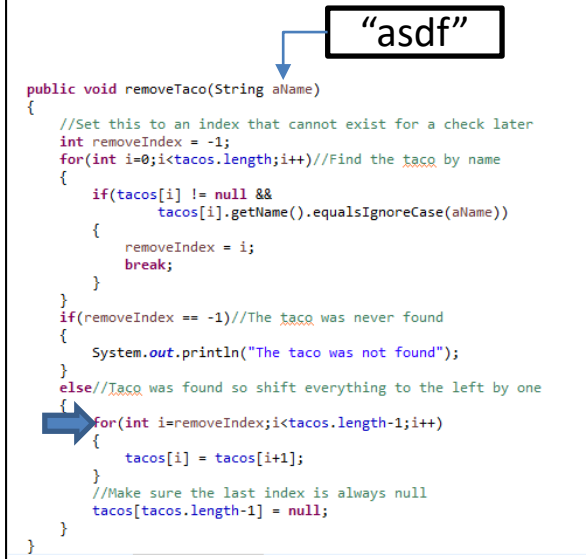
Memory

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	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



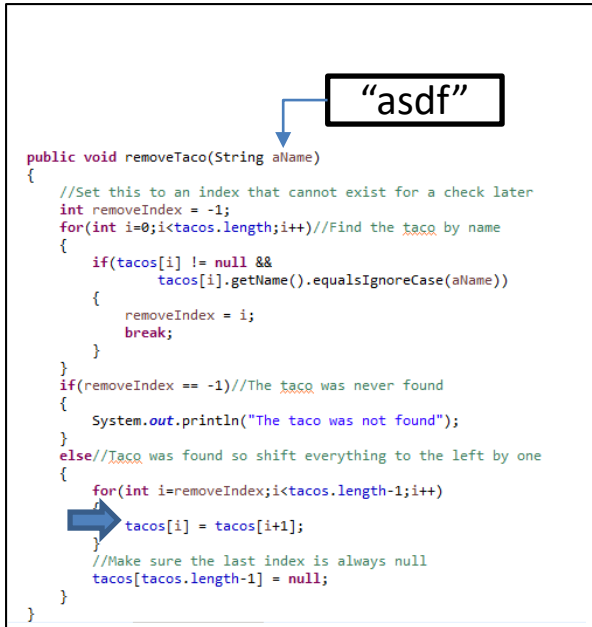
Memory

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	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



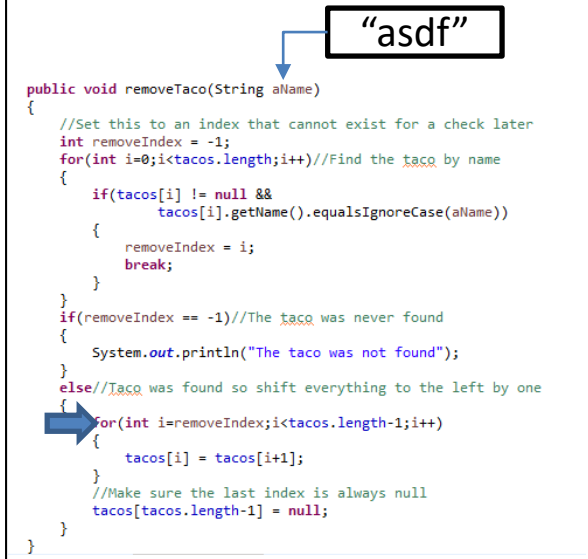
Memory

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	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



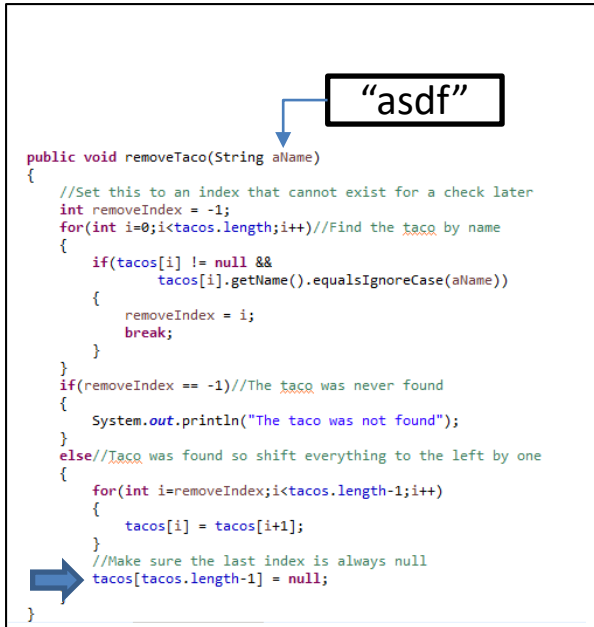
Memory

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
…	…	…
aName	"asdf"	128
…	…	…
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
…	…	…
Taco		326
aTaco.name	"asdf"	
…	…	…

Remove Tacos Demo



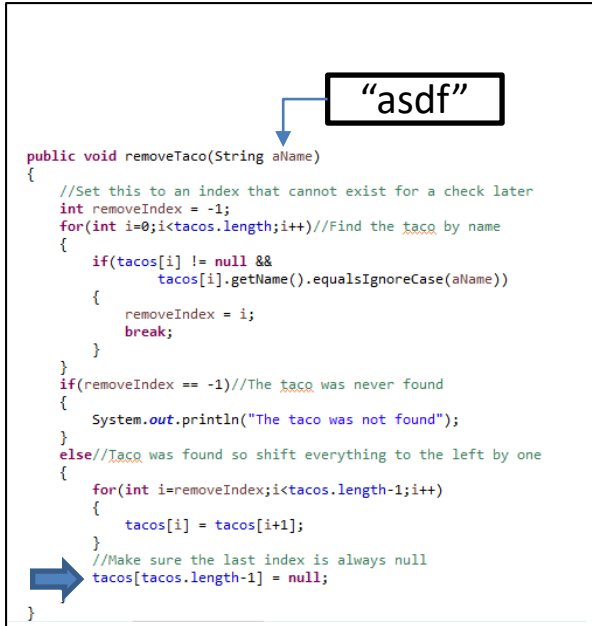
Memory

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
…	…	…
aName	"asdf"	128
…	…	…
Taco		256
aTaco.name	"none"	
aTaco.location	"none"	
aTaco.price	0.0	
…	…	…
Taco		326
aTaco.name	"asdf"	
…	…	…

Remove Tacos Demo



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
…	…	…
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	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	
…	…	…
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]	**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”