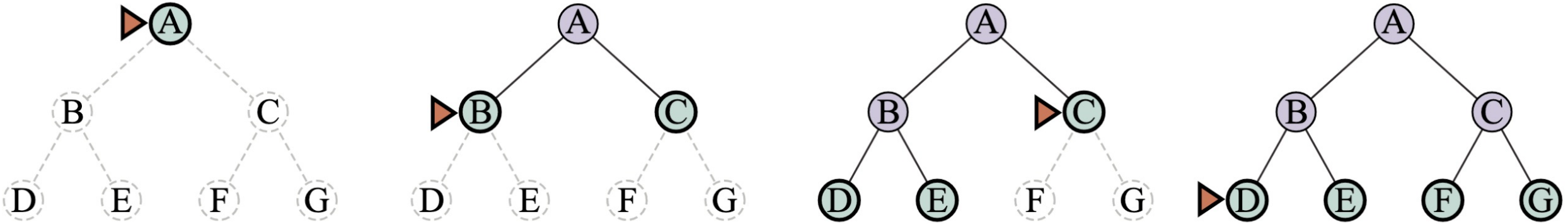
Linked Lists

Forest Agostinelli
University of South Carolina

Motivation: Breadth-First Search

- Prioritize the shallowest nodes
- For breadth-first search, we do not have to wait until the goal node is selected for expansion, we can terminate when the goal state is generated



Node to be expanded next

Not yet generated

In OPEN

Expanded

Motivation: Breadth First Search

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
    node ← NODE(problem.INITIAL)
    if problem.IS-GOAL(node.STATE) then return node
    frontier ← a FIFO queue, with node as an element
    reached ← {problem.INITIAL}
     while not IS-EMPTY(frontier) do
        node ← POP(frontier)
        for each child in EXPAND(problem, node) do
            s ← child.STATE
            if problem.IS-GOAL(s) then return child
            if s is not in reached then
                add s to reached
                add child to frontier
    return failure
```

Breadth-first search is a special case where we can do the goal test when nodes are generated instead of when they are selected for expansion

Arrays

- Fixed, Contiguous Blocks of Memory of the same type
- Pros
  - Random Access
- Cons
  - Cannot Resize
  - Not great if the size is not known or fixed

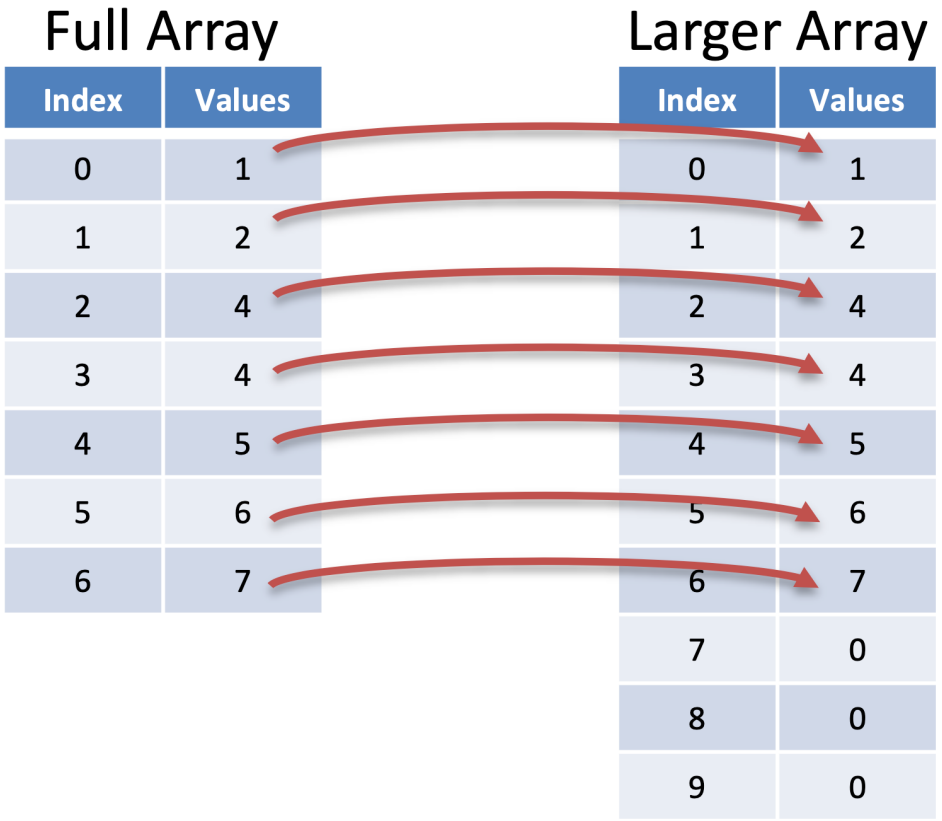
Array of Size 10 In Memory

Identifier	Contents	Byte Address
…	…	…
a[]	36	28
…	….	…
a[0]	256	36
a[1]		42
a[2]		48
…		
a[9]	NULL	90

Arrays

- Array Lists
  - Strings
- Growing
  - Create a new array with a larger size
  - Transfer all the data from the original array to the new array
  - Remove the original Array
- Pros
  - Semi-Random Access
  - Growable Structure
- Cons
  - Lots of Overhead
  - Does not Shrink
  - Not Great for Large Amounts of Data
  - Not Great Performance

Growing Concept



Linked Lists

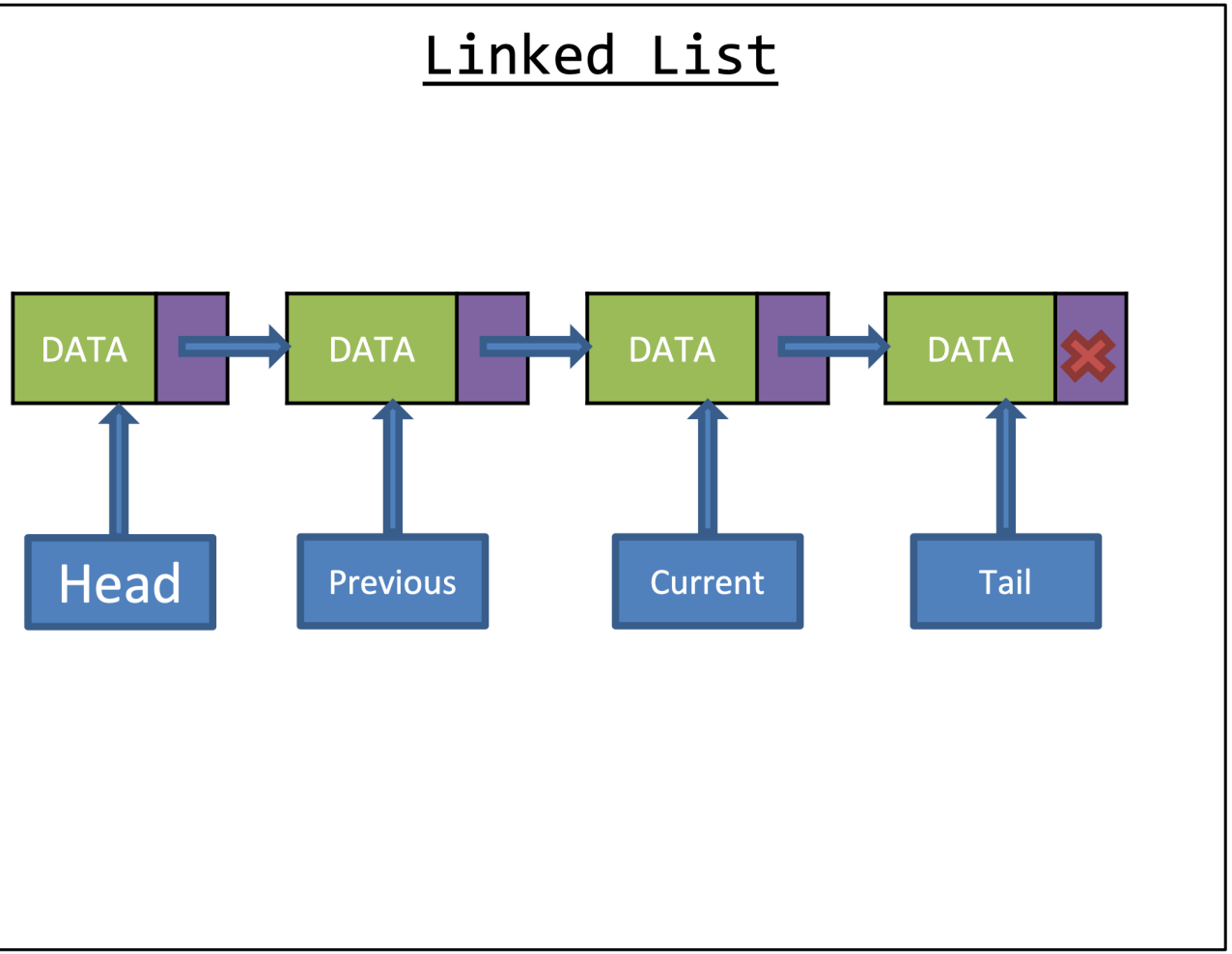
- Groups Together
  - Data
  - Link(s) / Reference(s) / Pointer(s)
  - "Node"
- Pros
  - Growable
  - Shrinkable
- Cons
  - No Random Access

Node

List of Nodes

Linked Lists

- Nodes Contain
  - Data
  - Link
- Special Nodes
  - Head: Always points to the first element of the list
  - Tail: Always points to the last element of the list
  - Current: Movable pointer used to Access and Modify Data in the List
  - Previous: Always stays on node behind Current
- Certain Linked Lists may omit some of these Nodes



Linked List Class

- Class within Classes
- Aids in grouping together like-information that is only used within a class
- Other Programmers do not need access to these classes

Syntax

```
public <<class identifier>>
{
        private <<internal class identifier>>
        {
                //Body of Internal Class
        }
}
```

Example

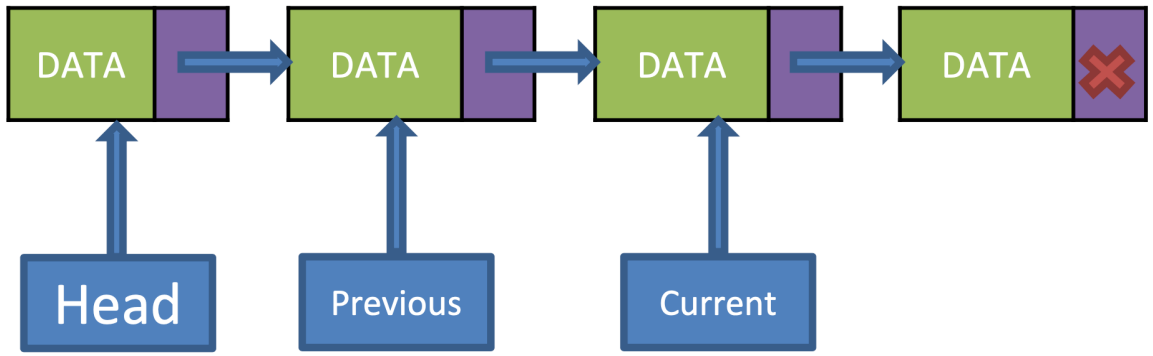
```
public class IntLL
{
      private class ListNode
      {

      }
}
```

Add Node

- Create a new Node with the given Data
- Start from the Head and find the Node with the first Null Link
- Point that Node to the newly Created Node

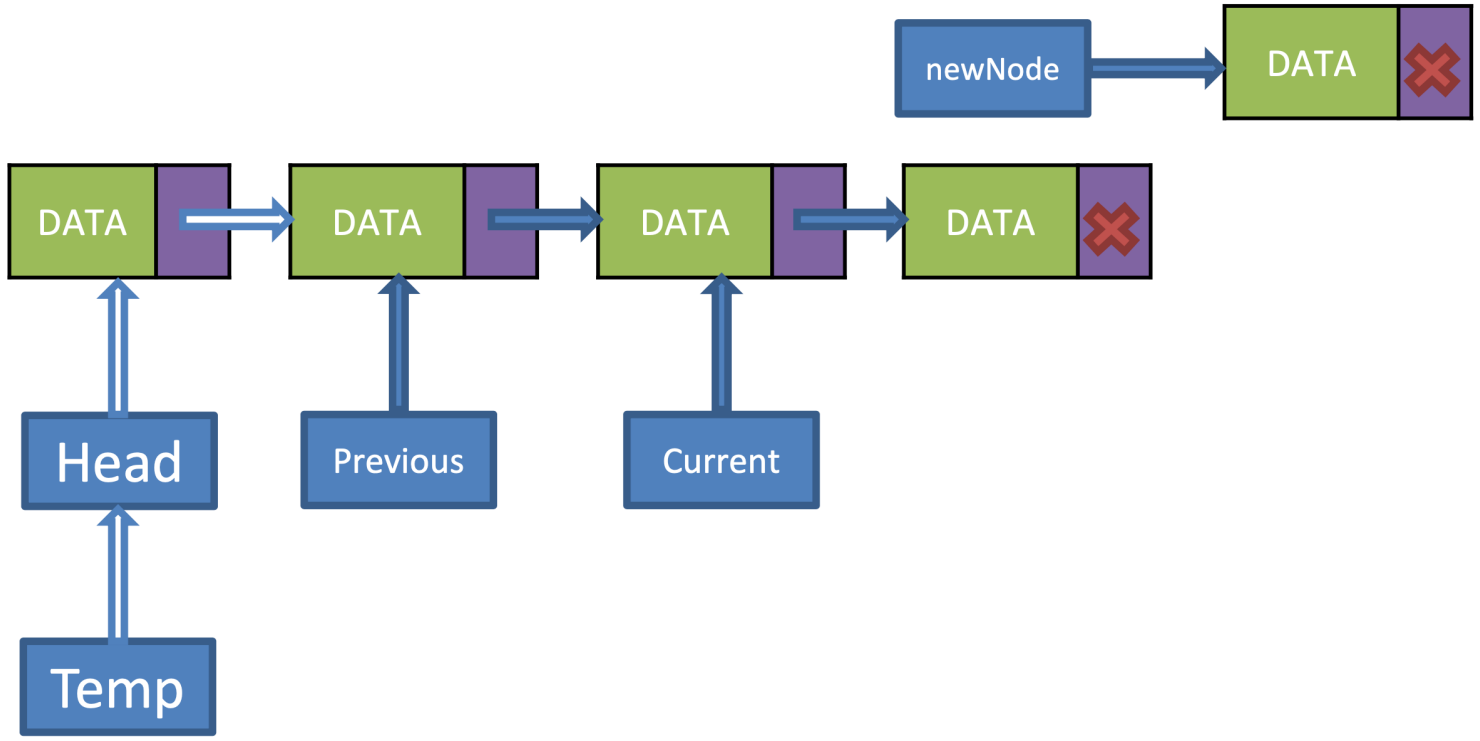
Concept



Add Node

- Create a new Node with the given Data
- Start from the Head and find the Node with the first Null Link
- Point that Node to the newly Created Node

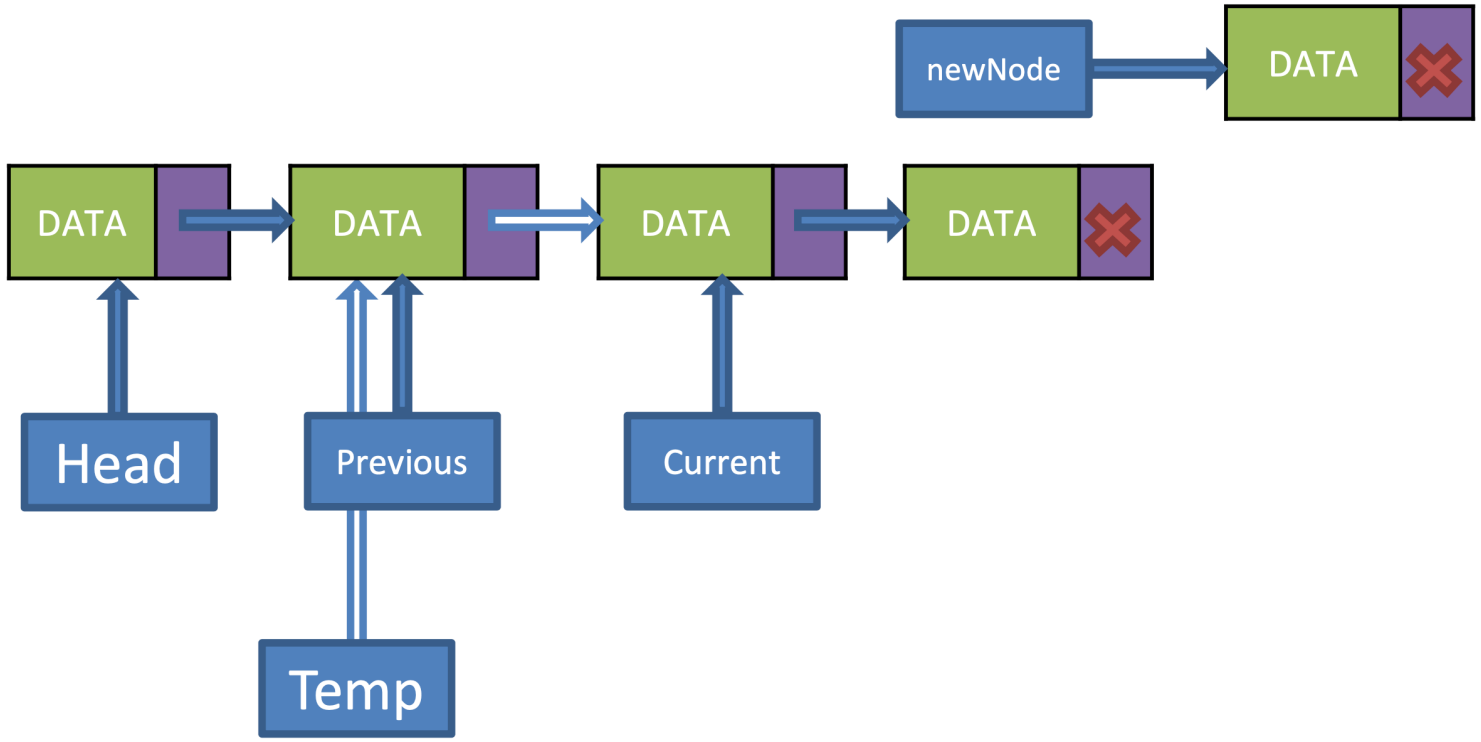
Concept



Add Node

- Create a new Node with the given Data
- Start from the Head and find the Node with the first Null Link
- Point that Node to the newly Created Node

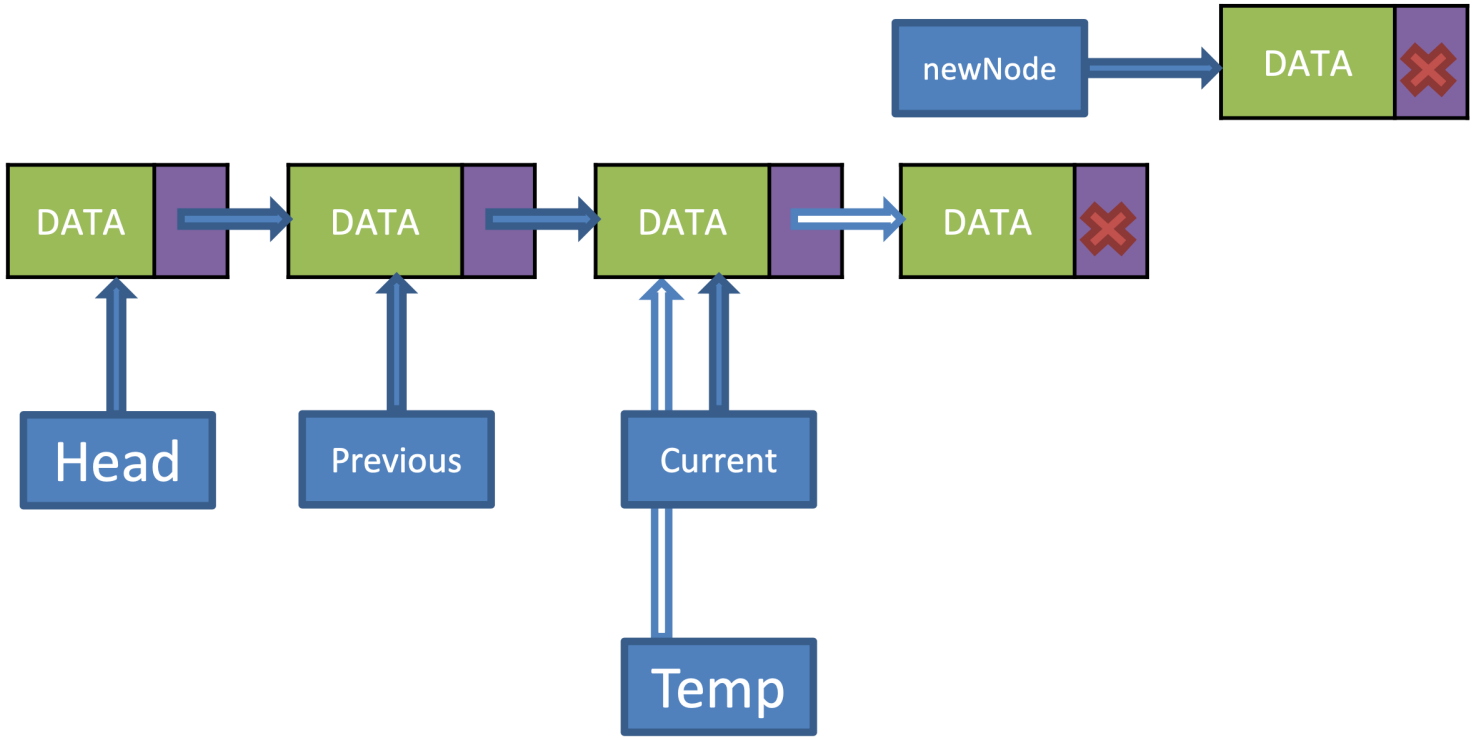
Concept



Add Node

- Create a new Node with the given Data
- Start from the Head and find the Node with the first Null Link
- Point that Node to the newly Created Node

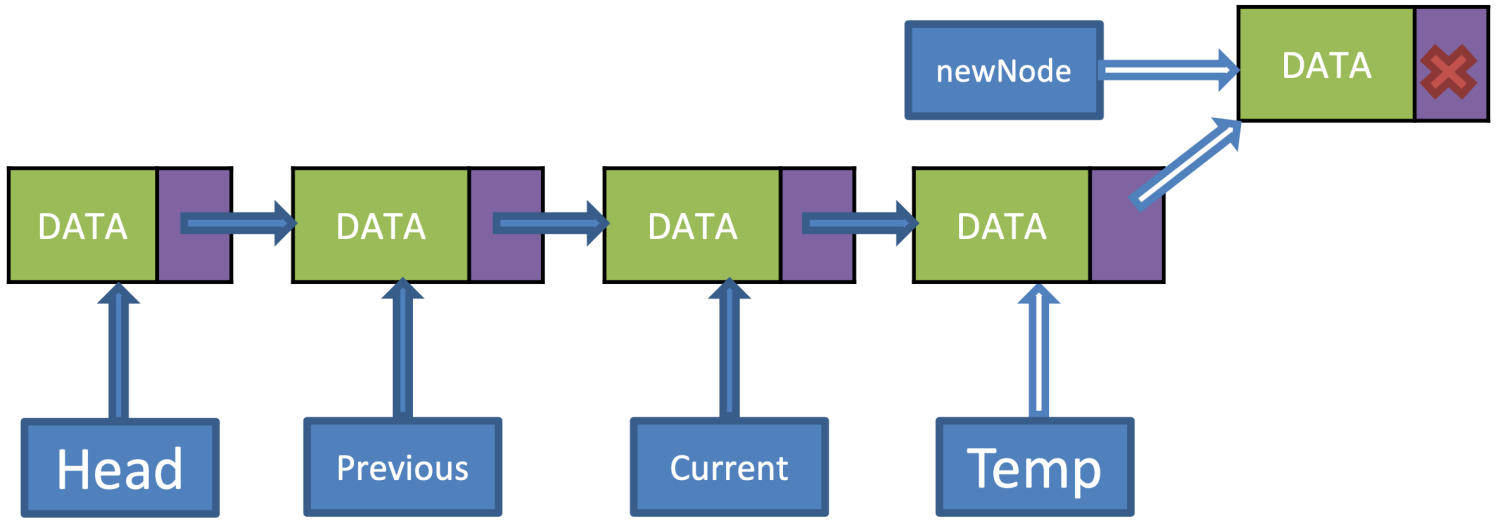
Concept



Add Node

- Create a new Node with the given Data
- Start from the Head and find the Node with the first Null Link
- Point that Node to the newly Created Node

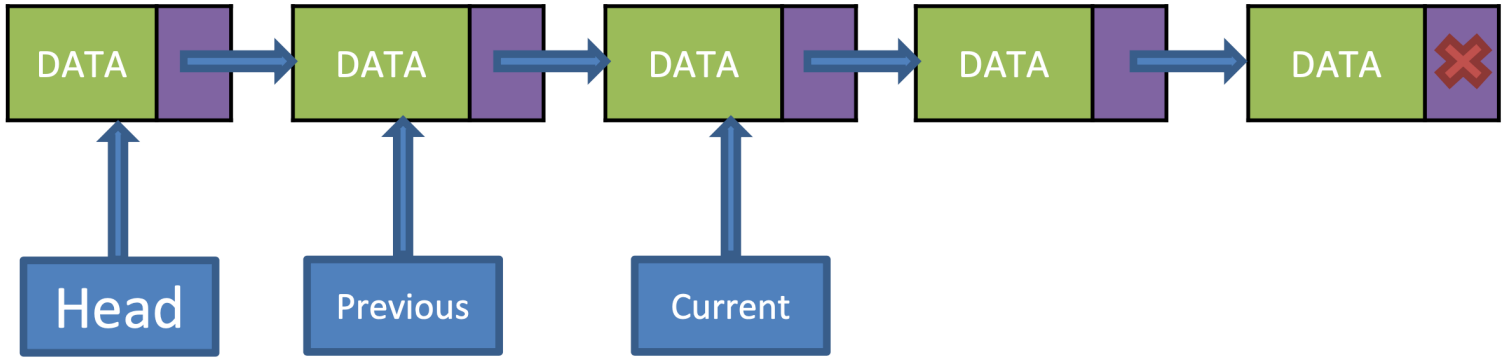
Concept



Add Node

- Create a new Node with the given Data
- Start from the Head and find the Node with the first Null Link
- Point that Node to the newly Created Node

Concept



Add Node

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	NULL	28
…	….	…

More Memory

Identifier	Contents	Byte Address
…	…	…
…	…	…

Add Node

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	50	28
…	….	…
IntLL	-	50
head	NULL	64
current	NULL	70
previous	NULL	76
…	…	…

More Memory

Identifier	Contents	Byte Address
…	…	…
…	…	…

Add Node

4

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	50	28
…	….	…
IntLL	-	50
head	NULL	64
current	NULL	70
previous	NULL	76
…	…	…
aData	4	96
…	…	…

More Memory

Identifier	Contents	Byte Address
…	…	…

Add Node

4

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	50	28
…	….	…
IntLL	-	50
head	NULL	64
current	NULL	70
previous	NULL	76
…	…	…
aData	4	96
…	…	…
newNode	NULL	104
…	…	…

More Memory

Identifier	Contents	Byte Address
…	…	…

Add Node

4

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	50	28
…	….	…
IntLL	-	50
head	NULL	64
current	NULL	70
previous	NULL	76
…	…	…
aData	4	96
…	…	…
newNode	NULL	104
…	…	…

More Memory

Identifier	Contents	Byte Address
…	…	…
ListNode	-	128
data	4	130
link	NULL	134
…	…	…

Add Node

4

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	50	28
…	….	…
IntLL	-	50
head	NULL	64
current	NULL	70
previous	NULL	76
…	…	…
aData	4	96
…	…	…
newNode	128	104
…	…	…

More Memory

Identifier	Contents	Byte Address
…	…	…
ListNode	-	128
data	4	130
link	NULL	134
…	…	…

Add Node

4

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	50	28
…	….	…
IntLL	-	50
head	**NULL**	64
current	NULL	70
previous	NULL	76
…	…	…
aData	4	96
…	…	…
newNode	128	104
…	…	…

More Memory

Identifier	Contents	Byte Address
…	…	…
ListNode	-	128
data	4	130
link	NULL	134
…	…	…

Add Node

4

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	50	28
…	….	…
IntLL	-	50
head	**128**	64
current	**128**	70
previous	NULL	76
…	…	…
aData	4	96
…	…	…
newNode	128	104
…	…	…

More Memory

Identifier	Contents	Byte Address
…	…	…
ListNode	-	128
data	4	130
link	NULL	134
…	…	…

Add Node

4

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	50	28
…	….	…
IntLL	-	50
head	**128**	64
current	**128**	70
previous	NULL	76
…	…	…

More Memory

Identifier	Contents	Byte Address
…	…	…
ListNode	-	128
data	4	130
link	NULL	134
…	…	…

Add Node

3

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	50	28
…	….	…
IntLL	-	50
head	**128**	64
current	**128**	70
previous	NULL	76
…	…	…

More Memory

Identifier	Contents	Byte Address
…	…	…
ListNode	-	128
data	4	130
link	NULL	134
…	…	…

Add Node

3

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	50	28
…	….	…
IntLL	-	50
head	**128**	64
current	**128**	70
previous	NULL	76
…	…	…
aData	3	96
…	…	…
newNode	265	104
…	…	…

More Memory

Identifier	Contents	Byte Address
…	…	…
ListNode	-	128
data	4	130
link	NULL	134
…	…	…
ListNode	-	265
data	3	270
link	NULL	274

Add Node

3

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	50	28
…	….	…
IntLL	-	50
head	**128**	64
current	**128**	70
previous	NULL	76
…	…	…
aData	3	96
…	…	…
newNode	265	104
…	…	…

More Memory

Identifier	Contents	Byte Address
…	…	…
ListNode	-	128
data	4	130
link	NULL	134
…	…	…
ListNode	-	265
data	3	270
link	NULL	274

Add Node

3

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	50	28
…	….	…
IntLL	-	50
head	**128**	64
current	**128**	70
previous	NULL	76
…	…	…
aData	3	96
…	…	…
newNode	265	104
…	…	…

More Memory

Identifier	Contents	Byte Address
…	…	…
ListNode	-	128
data	4	130
link	NULL	134
…	…	…
ListNode	-	265
data	3	270
link	NULL	274
…	…	…
temp	NULL	355

Add Node

3

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	50	28
…	….	…
IntLL	-	50
head	**128**	64
current	**128**	70
previous	NULL	76
…	…	…
aData	3	96
…	…	…
newNode	265	104
…	…	…

More Memory

Identifier	Contents	Byte Address
…	…	…
ListNode	-	128
data	4	130
link	NULL	134
…	…	…
ListNode	-	265
data	3	270
link	NULL	274
…	…	…
temp	**128**	355

Add Node

3

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	50	28
…	….	…
IntLL	-	50
head	**128**	64
current	**128**	70
previous	NULL	76
…	…	…
aData	3	96
…	…	…
newNode	265	104
…	…	…

More Memory

Identifier	Contents	Byte Address
…	…	…
ListNode	-	128
data	4	130
link	**NULL**	134
…	…	…
ListNode	-	265
data	3	270
link	NULL	274
…	…	…
temp	**128**	355

Add Node

3

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	50	28
…	….	…
IntLL	-	50
head	**128**	64
current	**128**	70
previous	NULL	76
…	…	…
aData	3	96
…	…	…
newNode	265	104
…	…	…

More Memory

Identifier	Contents	Byte Address
…	…	…
ListNode	-	128
data	4	130
link	**265**	134
…	…	…
ListNode	-	265
data	3	270
link	NULL	274
…	…	…
temp	**128**	355

Add Node

3

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	50	28
…	….	…
IntLL	-	50
head	**128**	64
current	**128**	70
previous	NULL	76
…	…	…

More Memory

Identifier	Contents	Byte Address
…	…	…
ListNode	-	128
data	4	130
link	**265**	134
…	…	…
ListNode	-	265
data	3	270
link	NULL	274
…	…	…

Add Node

2

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	50	28
…	….	…
IntLL	-	50
head	**128**	64
current	**128**	70
previous	NULL	76
…	…	…

More Memory

Identifier	Contents	Byte Address
…	…	…
ListNode	-	128
data	4	130
link	**265**	134
…	…	…
ListNode	-	265
data	3	270
link	NULL	274
…	…	…

Add Node

3

```
public void add(int aData)
{
    ListNode newNode = new ListNode(aData,null);
    if(head == null)
    {
        head = current = newNode;
        return;
    }
    ListNode temp = head;
    while(temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newNode;
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	50	28
…	….	…
IntLL	-	50
head	**128**	64
current	**128**	70
previous	NULL	76
…	…	…

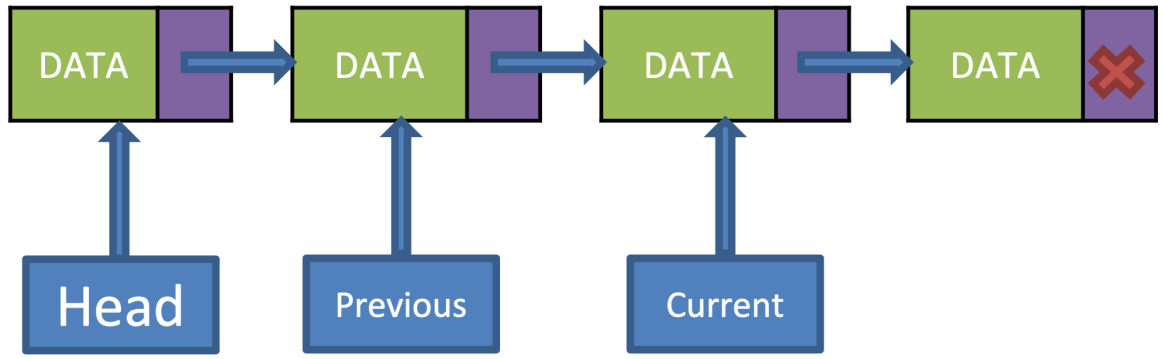
More Memory

Identifier	Contents	Byte Address
…	…	…
ListNode	-	128
data	4	130
link	**265**	134
…	…	…
ListNode	-	265
data	3	270
link	**374**	274
…	…	…
ListNode	-	374
data	2	380
link	NULL	384

Insert Node After Current

- Create a new Node with the given Data
- Set new Node's Link to Current's Link
- Point Current's Link to the new Node

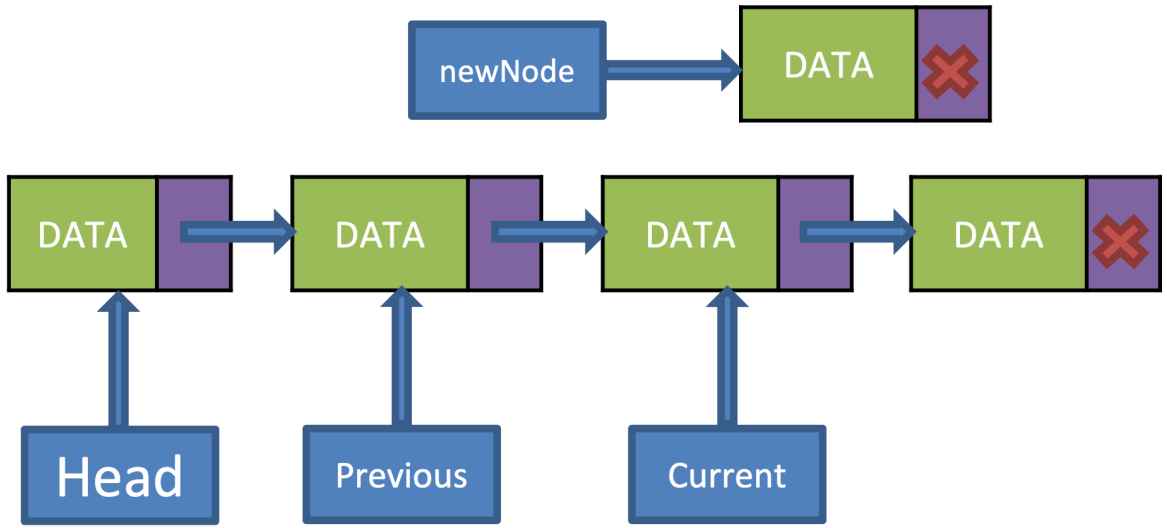
Concept



Insert Node After Current

- Create a new Node with the given Data
- Set new Node's Link to Current's Link
- Point Current's Link to the new Node

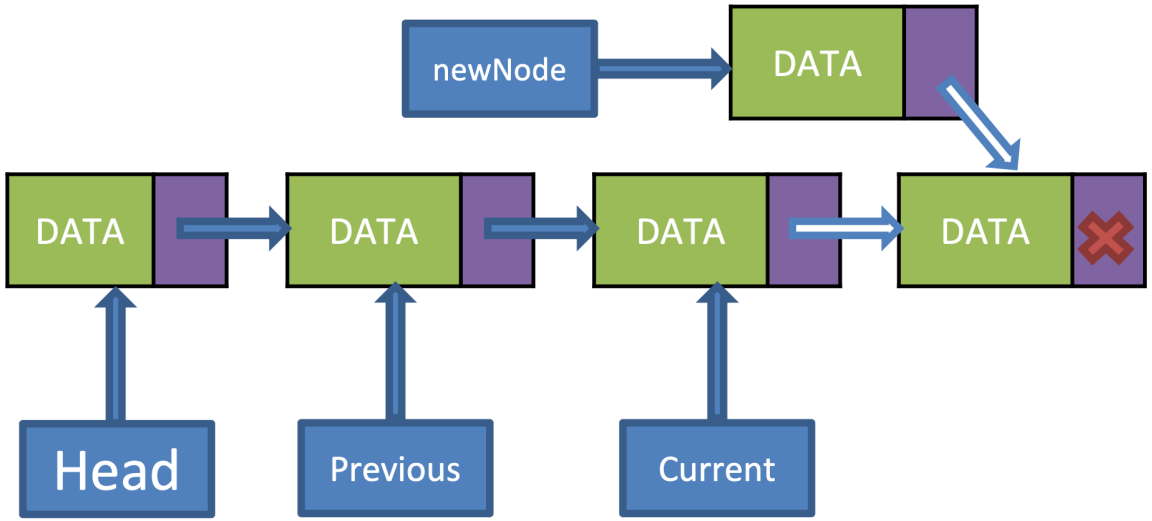
Concept



Insert Node After Current

- Create a new Node with the given Data
- Set new Node's Link to Current's Link
- Point Current's Link to the new Node

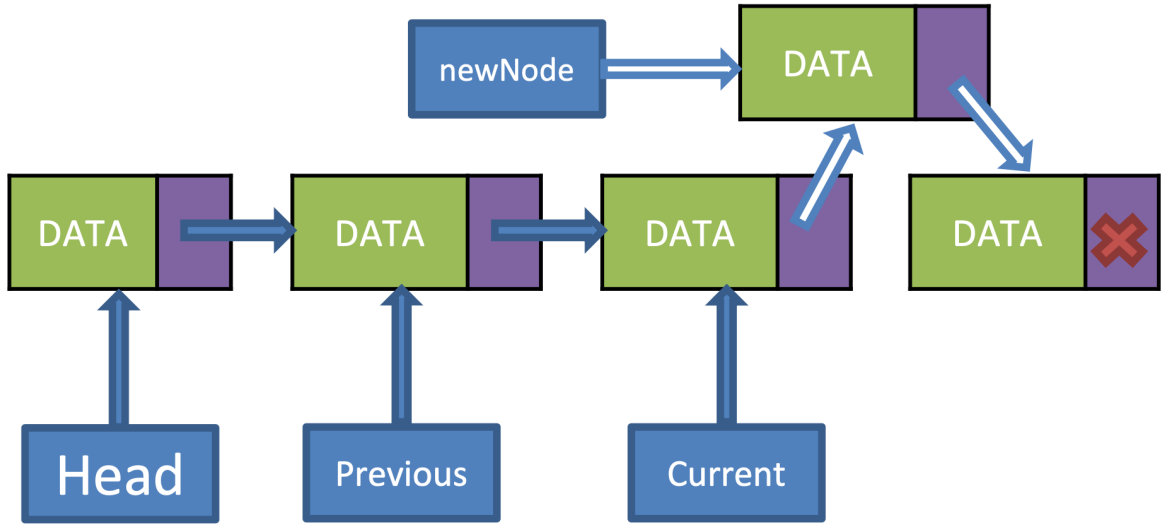
Concept



Insert Node After Current

- Create a new Node with the given Data
- Set new Node's Link to Current's Link
- Point Current's Link to the new Node

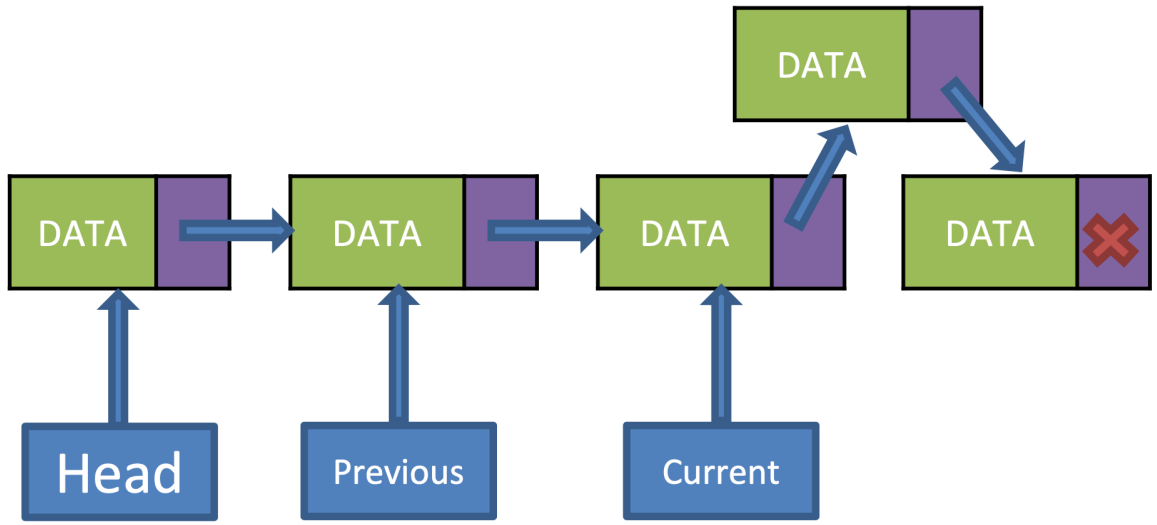
Concept



Insert Node After Current

- Create a new Node with the given Data
- Set new Node's Link to Current's Link
- Point Current's Link to the new Node

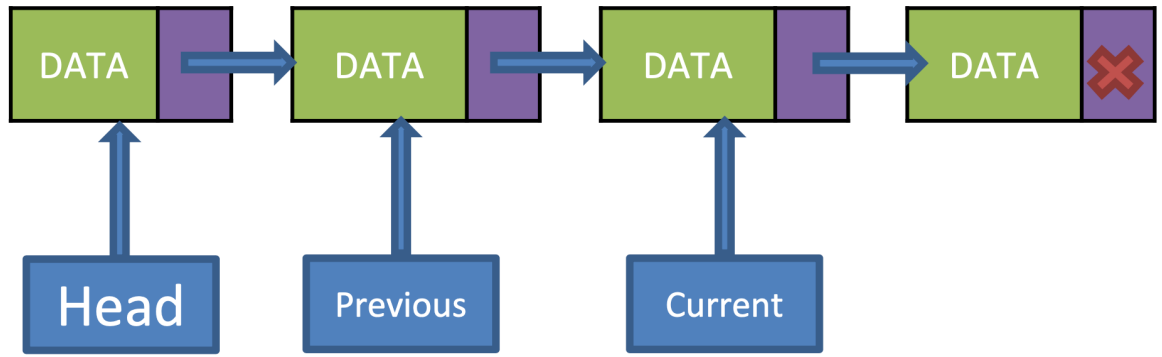
Concept



Remove Current Node

- If the Current is referencing the Head
  - Move Head and Current forward one node
- Set the Previous' Link to Current's Link
- Move Current Forward

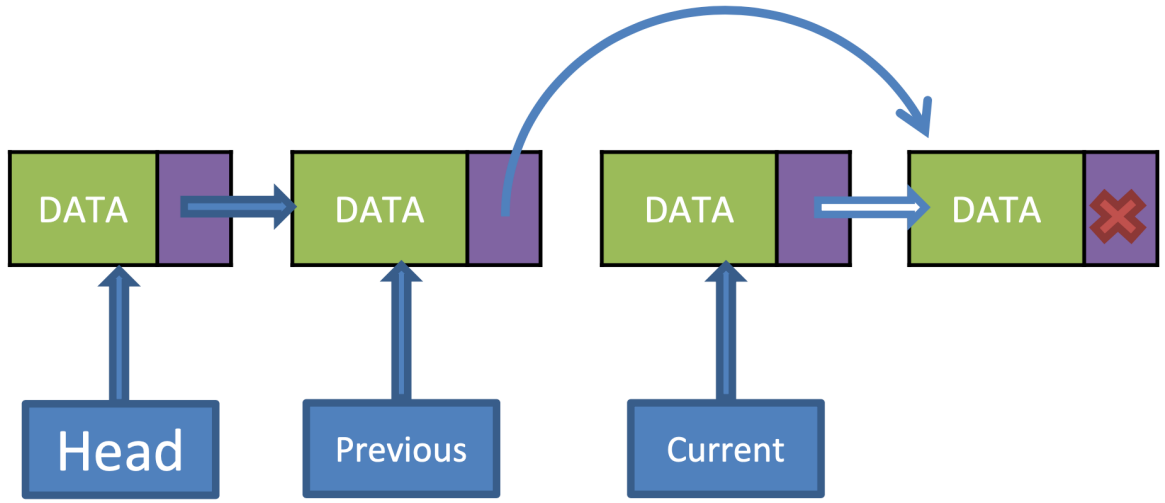
Concept



Remove Current Node

- If the Current is referencing the Head
  - Move Head and Current forward one node
- Set the Previous' Link to Current's Link
- Move Current Forward

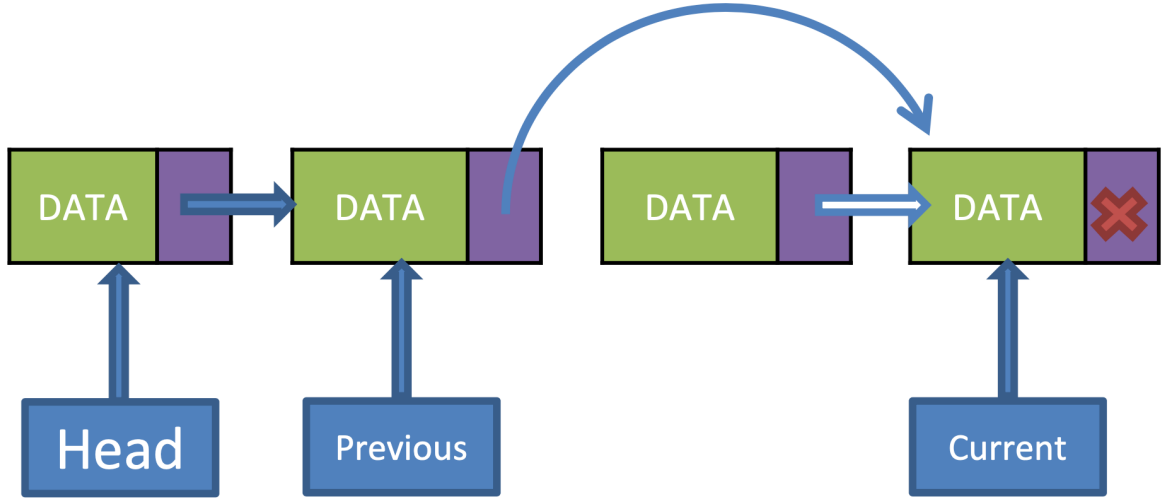
Concept



Remove Current Node

- If the Current is referencing the Head
  - Move Head and Current forward one node
- Set the Previous' Link to Current's Link
- Move Current Forward

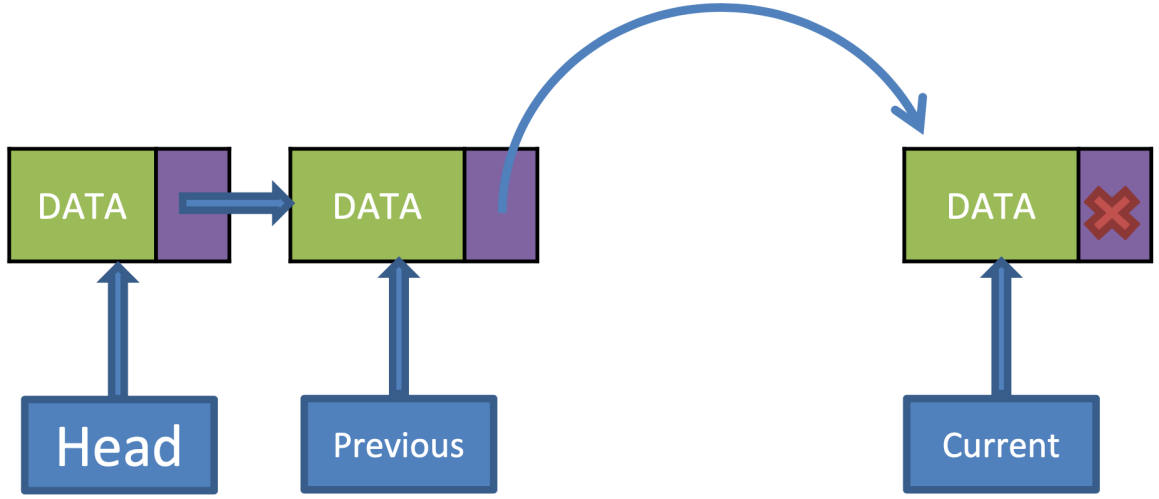
Concept



Remove Current Node

- If the Current is referencing the Head
  - Move Head and Current forward one node
- Set the Previous' Link to Current's Link
- Move Current Forward

Concept



Remove Current Node

```
public void removeCurrent()
{
    if ((current != null) && (previous != null))
    {
        previous.link = current.link;
        current = current.link;
    }
    else if ((current != null) && (previous == null))
    {//At head node
        head = current.link;
        current = head;
    }
}
```

Memory

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	....	...
IntLL	-	50
head	128	64
current	265	70
previous	128	76
...	...	...

More Memory

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	265	134
...	...	...
ListNode		265
data	3	270
link	374	274
...	...	...
ListNode	-	374
data	2	380
link	NULL	384

Remove Current Node

```
public void removeCurrent()
{
    if ((current != null) && (previous != null))
    {
        previous.link = current.link;
        current = current.link;
    }
    else if ((current != null) && (previous == null))
    {//At head node
        head = current.link;
        current = head;
    }
}
```

Memory

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	....	...
IntLL	-	50
head	128	64
current	**265**	70
previous	**128**	76
...	...	...

More Memory

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	265	134
...	...	...
ListNode		265
data	3	270
link	374	274
...	...	...
ListNode	-	374
data	2	380
link	NULL	384

Remove Current Node

```
public void removeCurrent()
{
    if ((current != null) && (previous != null))
    {
        previous.link = current.link;
        current = current.link;
    }
    else if ((current != null) && (previous == null))
    {//At head node
        head = current.link;
        current = head;
    }
}
```

Memory

Identifier	Contents	Byte Address
…	…	…
iLinkedList	50	28
…	….	…
IntLL	-	50
head	128	64
current	**265**	70
previous	**128**	76
…	…	…

More Memory

Identifier	Contents	Byte Address
…	…	…
ListNode	-	128
data	4	130
link	**374**	134
…	…	…
ListNode		265
data	3	270
link	**374**	274
…	…	…
ListNode	-	374
data	2	380
link	NULL	384

Remove Current Node

```
public void removeCurrent()
{
    if ((current != null) && (previous != null))
    {
        previous.link = current.link;
        current = current.link;
    }
    else if ((current != null) && (previous == null))
    {//At head node
        head = current.link;
        current = head;
    }
}
```

Memory

Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	....	...
IntLL	-	50
head	128	64
current	**374**	70
previous	**128**	76
...	...	...

More Memory

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	**374**	134
...	...	...
ListNode	-	265
data	3	270
link	**374**	274
...	...	...
ListNode	-	374
data	2	380
link	NULL	384

Remove Current Node

```
public void removeCurrent()
{
    if ((current != null) && (previous != null))
    {
        previous.link = current.link;
        current = current.link;
    }
    else if ((current != null) && (previous == null))
    {//At head node
        head = current.link;
        current = head;
    }
}
```

Memory

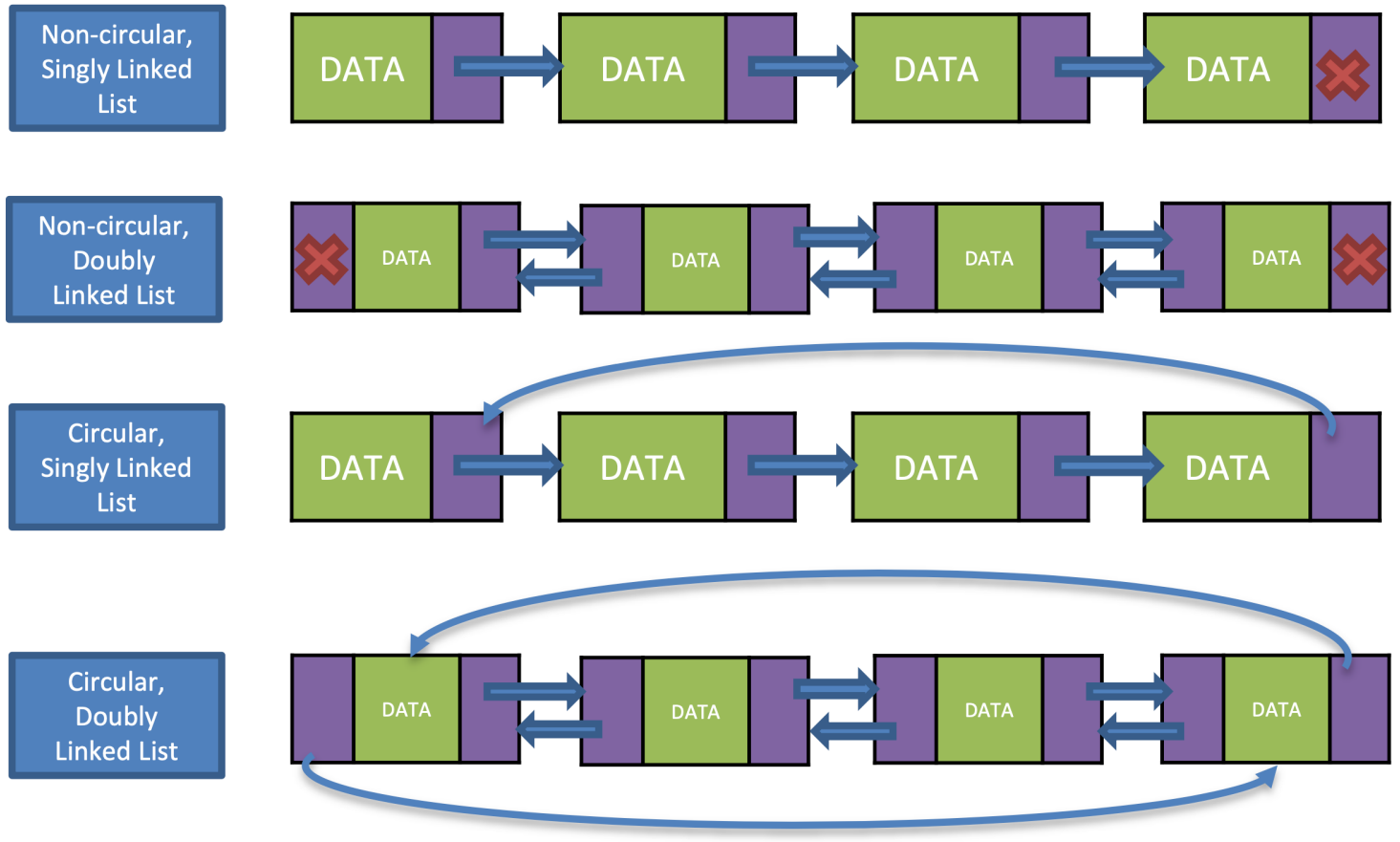
Identifier	Contents	Byte Address
...	...	...
iLinkedList	50	28
...	....	...
IntLL	-	50
head	128	64
current	**374**	70
previous	**128**	76
...	...	...

More Memory

Identifier	Contents	Byte Address
...	...	...
ListNode	-	128
data	4	130
link	**374**	134
...	...	...
...	...	...
ListNode	-	374
data	2	380
link	NULL	384

Types of Linked Lists

- Singly Linked Lists
- Doubly Linked Lists
- Circular Linked Lists



Types of Linked Lists

- This is only a Linked List of Integers
- How can we make this same structure without having to rewrite the code for every type?

Generics

- Generics
  - "Variables for Types"
  - Spoken: "This is a class of <<types>"
- In Java the Generic type must be an Object-Type
  - Everything in Java is assumed to inherit from type "Object"

Syntax

```
public class <<class identifier>> < <<Generic Type>> >
{

}
```

Example

```
public class GenLL <T>
{

}
```

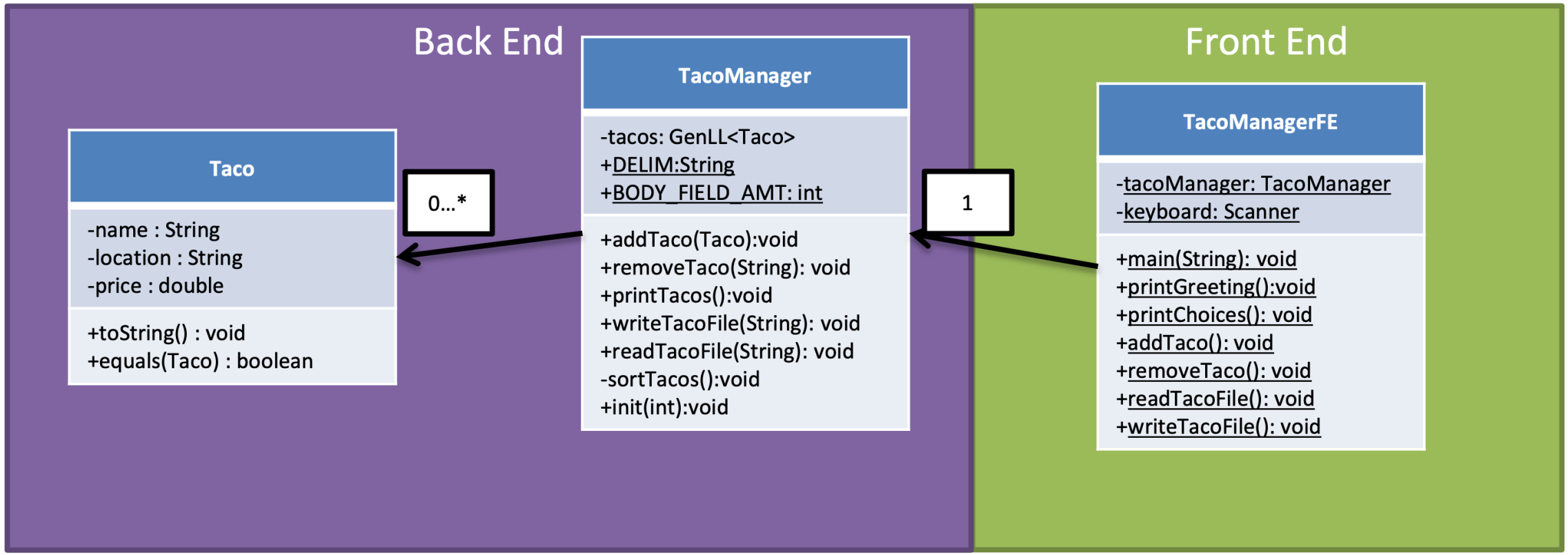
Generics

- Change the type for the data to “T”
  - All functionality previously described works in the exact same way
  - Only difference being the type
- “T” is always an Object-Type in Java
  - The “==“ and “!=“ should only be used to refer to memory addresses
  - All Objects are assumed to have a “.equals(Object)” method in Java
  - All Objects are assumed to have a “.toString()” method

Example

```
public class GenLL <T>
{
	private class ListNode
	{
		T data;
		ListNode link;
		public ListNode(T aData, ListNode aLink)
		{
			data = aData;
			link = aLink;
		}
	}
}
```

Generics: Example



Generics: Example