Linking Structures
• Groups Together
  – Data
  – Link(s) / Reference(s) / Pointer(s)
  – “Node”
• Pros
  – Growable
  – Shrinkable
• Cons
  – No Random Access
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
- How can we make this same structure without having to rewrite the code for every type?

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

<table>
<thead>
<tr>
<th>DATA</th>
<th>DATA</th>
<th>DATA</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Head</td>
<td>Previous</td>
<td>Current</td>
<td></td>
</tr>
</tbody>
</table>

**Concept**
### Adding

- 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
- 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
• 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
• 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
• 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
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
• 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
- 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
• 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
• 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
• 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
• 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
- 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
• 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
• 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
- 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
• 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
<table>
<thead>
<tr>
<th>Concept</th>
<th>Adding After Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Create a new Node with the given Data</td>
<td></td>
</tr>
<tr>
<td>• Set new Node’s Link to Current’s Link</td>
<td></td>
</tr>
<tr>
<td>• Point Current’s Link to the new Node</td>
<td></td>
</tr>
</tbody>
</table>
• 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
• 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
If the Current is referencing the Head
  – Move Head and Current forward one node
Set the Previous’s Link to Current’s Link
Move Current Forward

Concept
• 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
• 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
<table>
<thead>
<tr>
<th>Concept</th>
<th>If the Current is referencing the Head</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>– Move Head and Current forward one node</td>
</tr>
<tr>
<td></td>
<td>• Set the Previous’ Link to Current’s Link</td>
</tr>
<tr>
<td></td>
<td>• Move Current Forward</td>
</tr>
</tbody>
</table>

![Diagram](image)
Removing Current

- If the Current is referencing the Head
  - Move Head and Current forward one node
- Set the Previous’s Link to Current’s Link
- Move Current Forward
• Singly Linked Lists
• Doubly Linked Lists
• Circular Linked Lists
Linking Structures