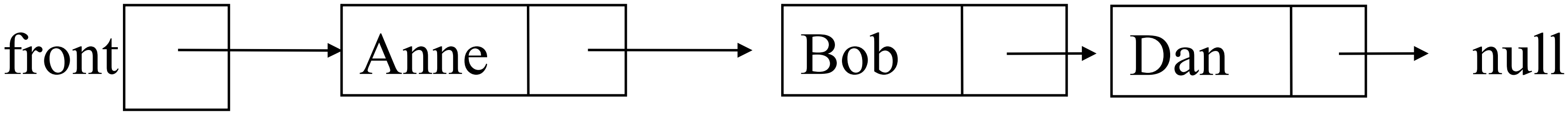
Computer Science 112
Data Structures

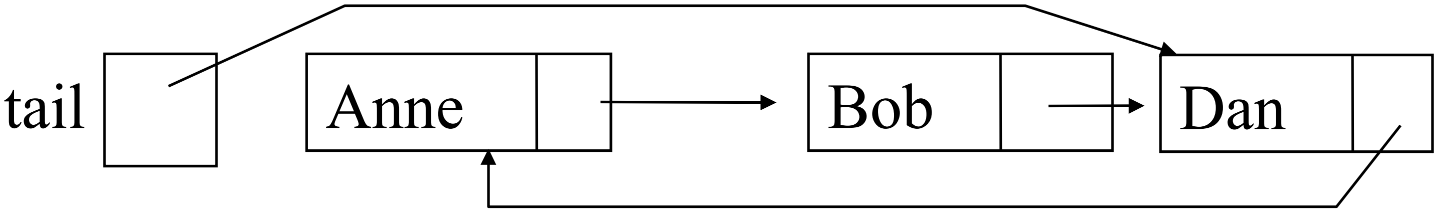
Lecture 06:
Exceptions
Generics

Review:
Varieties of Linked Lists

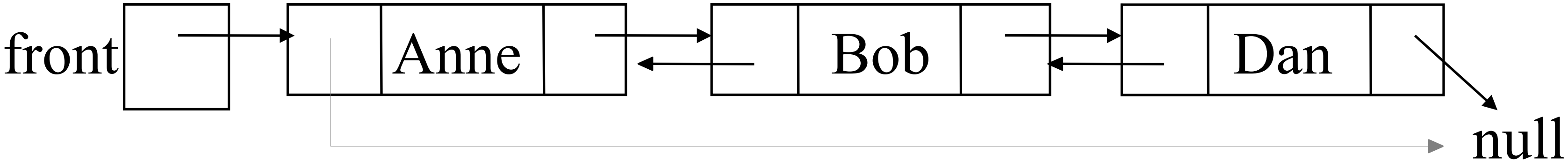
- **Singly Linked List**



- **Circular Linked List**

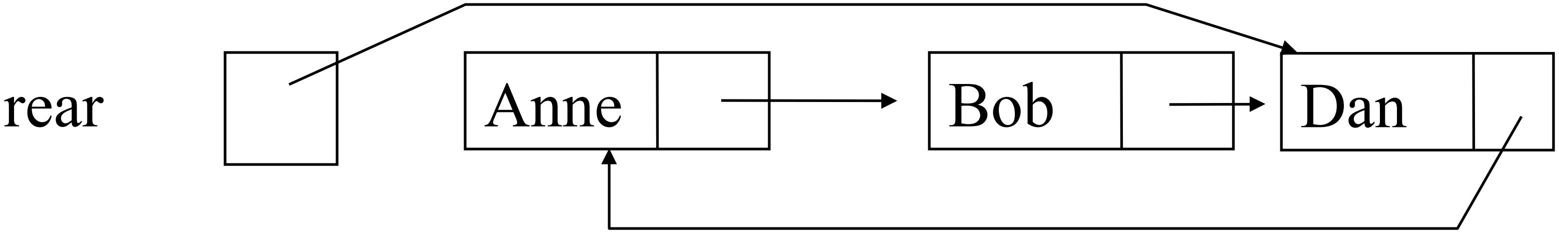


- **Doubly Linked List**

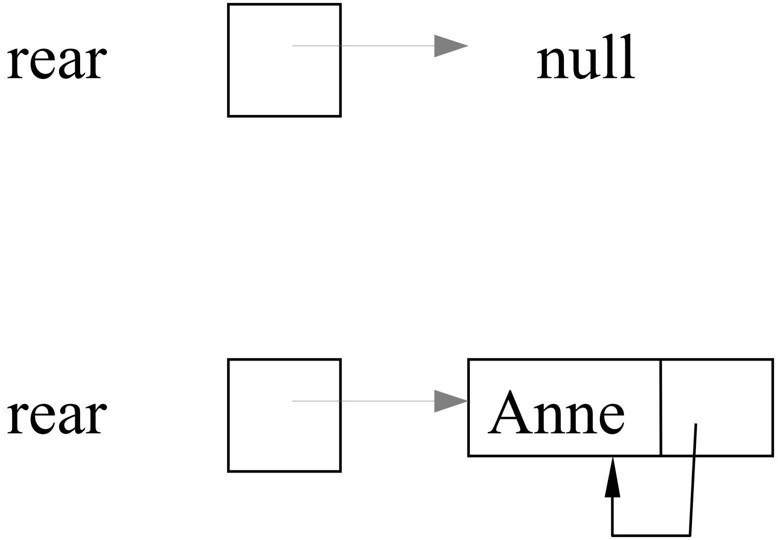


Circular List

- rear takes the place of front, but:
  - rear holds a reference to the last node
  - rear.next holds a reference to the first node

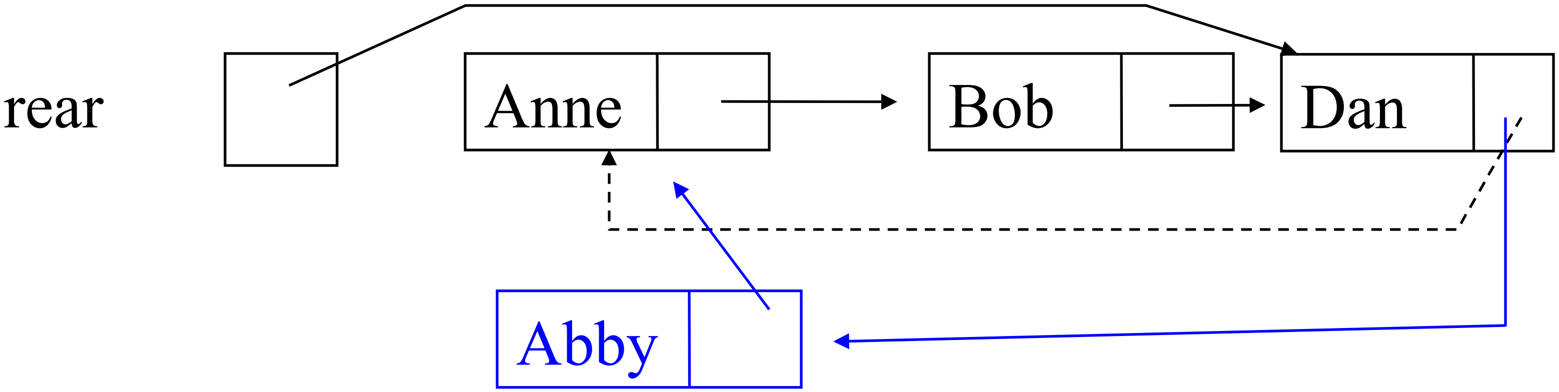


Circular List With No Nodes and With One Node

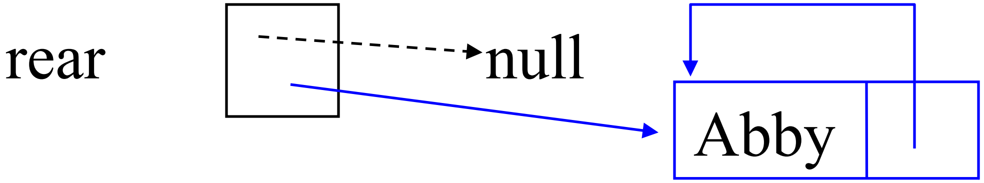


Insert at Head: 2 Cases

- **List not empty**



- **List empty**



Insert at Head

```
if(rear == null){  // insert in empty list
     rear =  new Node(newData, null);
     rear.next = rear;
  } else {           // insert in non-empty list
     Node newNode= new Node(newData,
                                       rear.next);
     rear.next = newNode;
  }
  return rear;
```

Insert at Head

```
if(rear == null){  // insert in empty list
    rear =  new Node(newData, null);
    rear.next = rear;
  } else {           // insert in non-empty list
    rear.next = new Node(newData,
                                  rear.next);
  }
  return rear;
```

Delete Head

- 3 cases:
  - List already empty, i.e. `rear == null`

    ```
    return rear;
    ```
  - list has one node, i.e. `rear.next == rear`

    ```
    rear = null;  return rear;
    ```
  - list had more nodes, i.e. `rear.next != rear`

    ```
    rear.next = rear.next.next;  return rear;
    ```

Add at Rear

- 2 cases
  - add to empty list
  - add to non-empty list
- Like addAtFront but step rear one step on list

Other CLL Methods

- **See resources > Steinberg > Java > CLLApp.java**

- **note finding the rear is O(1) but**
- **removing the rear is still O(n)**

Doubly Linked Lists

- See resources > Steinberg > Java >
  DLLApp.java
- Note that these DLLs are not circular

New: Exceptions

When an error occurs, an exception is thrown.

- **An exception is an object**
  - **Its class is a descendant of Exception**
  - **Its class tells you what error has occurred**
    - **ArrayIndexOutOFBoundsException**
    - **NumberFormatException**

When an error occurs ...

E.g, when code tries to access index -1 of an array

- **An exception that is an instance of the appropriate class is created**
- **This exception (instance) is “thrown”**
  - **The throw is caught by a try-catch statement, or else**
  - **the throw causes the program to crash**

To do your own throw

- Use the throw statement

```
if (r < 0 || r >= n){
  throw new NoSuchElementException(r+” ”);
}
```

Checked vs Runtime Exceptions

- **Runtime Exceptions**
  - **Classes are descendants of RuntimeException**
  - **Do not use throws clauses in method headers**
  - **Represent program errors**
    - **ArrayIndexOutOfBoundsException**

Checked vs Runtime Exceptions

- **Checked Exceptions**
  - **Classes are descendants of Exception but not of RuntimeException**
  - **Require throws clauses in method headers**
  - **Represent user or environmental errors**
    - **FileNotFoundException**

To catch a throw

```
 try{
    <statements>
} catch (<class of exceptions> <variable>){
     <statements>
}
```

To catch a throw

```
String line;
try{
    line = IO.readString( );
    double a = Double.parseDouble(line );
} catch (NumberFormatException e){
      IO.printString(“bad double, try again”);
      …
}
```

Finding a Catch

- A catch is active during the time its try statements are executing
  - Including any methods they call

```
try{ foo( )} catch (FileNotFoundException e)
    {…};
void foo( ){ … fie( ); …}
void fie( ){ … }
```

Finding a Catch

When an exception is thrown, java finds

- **The innermost active try**
  - **innermost = most recently entered**
  - **Active = not exited**
- **Where the exception being thrown is a subclass of the class in the catch**

Once catch is found

- **Skip rest of the try;**
- **Go immediately to the statements in the catch**

Bad uses of try-catch

- Don’t use it where an if, break, return, etc. would be simpler
- Don’t use it to simply ignore an error

Exceptions

- **See StringLLE.java, DriveLLNDie.java, DriveLLE.java**

Generics

- Consider ReadOnlyPairString:
  - Cf: ReadOnlyPairInteger.java
    - Class declarations and methods are almost identical
  - Solution "generics" (Java 5 & later)
    - Class & method definitions parameterized by type
  - See ReadOnlyPairInteger.java, ReadOnlyPairString.java, ReadOnlyPair.java

Generic List

- See LL.java
- Note use of wrapper class Integer