CS102

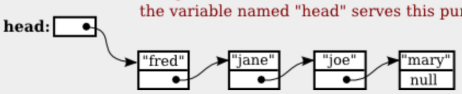
Introduction to
data structures, algorithms,
and object-oriented
programming

February 27, 2016

Singly-Linked Lists w/out Interface

When an object contains a reference to an object of the same type, then several objects can be linked together into a list. Each object refers to the next object.

For a list to be useful, there must be a variable that points to the first node in the list. Here, the variable named "head" serves this purpose.



Traversing Singly-Linked Lists

The common pattern is to start at the head of the list (usually called 'head'), then move, via a pointer, from each node to the next by following the pointer in the node, stopping when null is reached (i.e., when pointer == null, marking the end of the list).

```
Node runner;
// A pointer that will be used to traverse the list.
runner = head;
// Start with runner pointing to the head of the list.
while ( runner != null ) {
// Continue until null is encountered.
     System.out.println( runner.item );
     // Do something with the item in the current node.
     runner = runner.next;
     // Move on to the next node in the list.
}
```


Singly-Linked Lists

A version of the code on the last slide with a for loop that has different syntax than usual.

```
Node runner;
// A pointer that will be used to traverse the list.
runner = head;
// Start with runner pointing to the head of the list.
// Uses the name runner as the loop counter.
for ( Node runner = head;
      runner != null; runner = runner.next ) {
   System.out.println( runner.item );
}
```


A Node in a SinglyLinkedList

```
// Imagine this code is inside a class called
// SinglyLinkedList.
private class IntNode {
   int item;       // One of the integers in list.
   IntNode next;   // Pointer to the next node in list.
}
```

Traversing a list without using recursion to get the sum of all integers in the list

```
public void sumInts() {
  int sum = 0;
  IntNode runner = head;
  while ( runner != null ) {
     sum = sum + runner.item;
     // Add current item to the sum.
     runner = runner.next;
  }
  System.out.println("The sum of the list is " + sum);}
```


Java Exceptions

- Exception
  - Handles an error during execution
- Throw an exception
  - To indicate an error during a method execution
- Catch an exception
  - To deal with the error condition

Catching Exceptions

- Java provides `try-catch` blocks
  - To handle an exception
- Place statement that might throw an exception within the `try` block
  - Must be followed by one or more `catch` blocks
  - When an exception occurs, control is passed to `catch` block
- `Catch` block indicates type of exception you want to handle

Catching Exceptions

- `try-catch` blocks syntax

```
try {
  statement(s);
}
catch (exceptionClass identifier) {
  statement(s);
}
```

- Some exceptions from the Java API cannot be totally ignored
  - You must provide a handler for that exception

Catching Exceptions

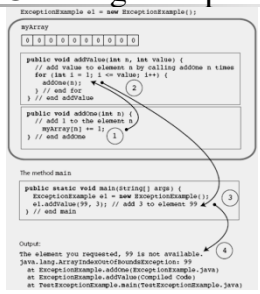
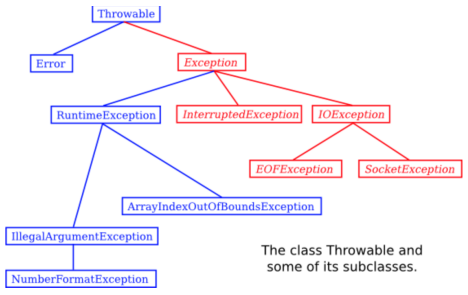


Figure 1-9

Flow of control in a simple Java application

Catching Exceptions

- Types of exception
  - Checked exceptions
    - Instances of classes that are subclasses of `java.lang.Exception`
    - Must be handled locally or thrown by the method
    - Used when method encounters a serious problem
  - Runtime exceptions
    - Occur when the error cannot be handled without exiting program
    - Instances of classes that are subclasses of `java.lang.RuntimeException`



The class Throwable and some of its subclasses.

11

Catching Exceptions

- The `finally` block
  - Executed whether or not an exception is thrown
  - Can be used even if no `catch` block is used
  - Syntax

```
finally {
  statement(s);
}
```

Throwing Exceptions

- `throws` clause
  - Written in a method signature, indicates a method may throw an exception...
    - ...if an error occurs during its execution
  - Syntax

```
public methodName throws ExceptionClassName
```

- `throw` statement
  - Used to throw an exception at any time
  - Syntax

```
throw new exceptionClass(stringArgument);
```

- You can define your own exception class

Creating Exception Classes

```
public class ParseException extends Exception {
  public ParseException(String message) {
      // Create a ParseException object containing
      // the given message as its error message.
    super(message);
  }
}
```