Lecture 4:
Abstract Data Types
(Chapter 2)

Sep 22, 2016

CS 102

Today’s Lecture

I. Prelimin, Review, etc
II. ADT and Java Interface
III. Linked Lists

Next...

I. Prelim, Review, etc

Programmers have more fun

Two bytes meet.

The first byte asks, “Are you ill?”

Programmers have more fun

Two bytes meet.

The first byte asks, “Are you ill?”

The second byte replies, “No, just feeling a bit off.”

Next...

II. Abstract Data Types and Java Interface

Data Abstraction

Abstraction: A view of a system that includes only the details essential to the perspective of the user of the system.

Data Abstraction

Abstraction: A view of a system that includes only the details essential to the perspective of the user of the system.

Data abstraction: The separation of a data type's logical properties from its implementation

Data Abstraction

Abstraction: A view of a system that includes only the details essential to the perspective of the user of the system.

Data abstraction: The separation of a data type's logical properties from its implementation

Abstract data type (ADT): A data type whose properties (domain and operations) are specified independently of particular implementations

Java Abstract Method

- Only the “header” of the method is given (no body)
- E.g., class for Geometric figures (circles, squares, etc):

```
public abstract class GeometricObject {
 private boolean filled;
 public boolean isFilled() { return filled; }
 public abstract double getArea(); //bodyless!
 ...
}
```

- No instantiations allowed !!
  - Q: So what can we do with it?
  - A: Extend it!

Java Abstract Method

- Only the “header” of the method is given (no body)
- E.g., class for Geometric figures (circles, squares, etc):

```
public abstract class GeometricObject {
 private boolean filled;
 public boolean isFilled() { return filled; }
 public abstract double getArea(); //bodyless!
 ...
}
```

- No instantiations allowed !!
  - Q: So what can we do with it?
  - A: Extend it!

Java Abstract Method

- Only the “header” of the method is given (no body)
- E.g., class for Geometric figures (circles, squares, etc):

```
public abstract class GeometricObject {
 private boolean filled;
 public boolean isFilled() { return filled; }
 public abstract double getArea(); //bodyless!
 ...
}
```

- No instantiations allowed !!
  - Q: So what can we do with it?
  - A: Extend it!

Java Abstract Method

- Only the “header” of the method is given (no body)
- E.g., class for Geometric figures (circles, squares, etc):

```
public abstract class GeometricObject {
 private boolean filled;
 public boolean isFilled() { return filled; }
 public abstract double getArea(); //bodyless!
 ...
}
```

- No instantiations allowed !!
  - Q: So what can we do with it?
  - A: Extend it!

Java Abstract Method

- Only the “header” of the method is given (no body)
- E.g., class for Geometric figures (circles, squares, etc):

```
public abstract class GeometricObject {
 private boolean filled;
 public boolean isFilled() { return filled; }
 public abstract double getArea(); //bodyless!
 ...
}
```

- No instantiations allowed !!
  - Q: So what can we do with it?
  - A: Extend it!

Java Abstract Method (contd)

- E.g.

```
public class Circle extends GeometricObject {
  private double rad;
  public Circle(double r) { rad= r; }
  public double getArea() { return rad*rad*Math.PI;}
  ...
}
```

Java Interface

What is it?

- More abstract than abstract classes!

- List of headers of methods (no bodies)

- Similar to an (abstract) Java class, but less detail

- Can include variable declarations (but only static and final variables)

Java Interface

A Java interface cannot be instantiated.

Q: So how can we use it?

Java Interface

A Java interface cannot be instantiated.

Q: So how can we use it?

A: we can ”IMPLEMENT” it!

Java Interface (defining one)

- E.g., interface for geometric figures (circles, squares, etc) from Textbook:

```
public interface FigureGeometry {
 final float PI = 3.14f;
 public float area();
 ...
}
```

- Note: it looks just like an abstract class!
- Aside: why final?

Java Interface (implementing one)

- Implementing FigureGeometry:

```
public Circle implements FigureGeometry {
 protected float rad;
 public Circle(float r){ rad = r; }
 public float area(){ return(PI*rad*rad); };
 ...
}
```

- Now, we CAN instantiate the Circle.
- We can imagine another implementation of a Square, with a side length (float) as member.

Java Interface (benefits)

We can check the syntax of our specification.

When we compile the interface, the compiler uncovers any syntactical errors in the method interface definitions.

Java Interface (benefits)

We can verify that the interface contract is met by implementation.

When we compile the implementation, the compiler ensures that the method names, parameters, and return types matches!

Java Interface (benefits)

We can provide a consistent interface to applications from among alternate implementations of the ADT.

Name recognition is very important for humans (e.g., hw)

Signatures

Signatures are more abstract that “headers”

- Its definition is a technical one
- It is used for implementation of Java overloading

Signatures

The signature of a method has two parts:

(1) the method's name

(2) the list of parameter types

Signatures

The signature of a method has two parts:

(1) the method's name

(2) the list of parameter types

E.g., consider the method f here:

```
public String f(double a, int n, double b) {
      ... body of method
}
```

Its signature is just

`f(double, int, double)`.

Signatures

So we discard

- the modifiers (`public`)
- the return type (`String`)
- and parameter names (`a, n, b`)

For more information, see
`http://docs.oracle.com/javase/tutorial/java/javaOO/methods`

Signatures

Q: How is Signatures used?

A: all methods in a class must have unique signatures!

Two important consequences:

(1) You cannot define two methods with the same signature, even if they have different return types or modifiers.

(2) You can define many methods with the same name, as long as their list of parameter types are different or appear in different order.

Signatures

E.g., the following set of signatures are compatible with each other:

```
f(double, int, double)
f(int, double, double)
f(int, double)
f(int)
f()
f(String, double)
```

All these methods are said to be overloading of the method name `f`.

Next...

III. Linked List

Arrays Versus Linked Lists

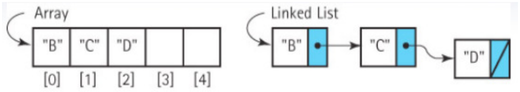


Figure: Array vs Linked List

Arrays Versus Linked Lists

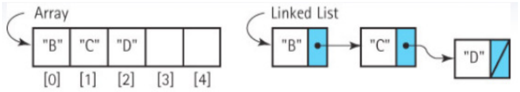


Figure: Array vs Linked List

```
String[] sArray = new String[5];
sArray[0]= "B"; sArray[1]= "C"; sArray[2]= "D";

Node n = new Node("B");
Node n1 = new Node("C");
Node n2 = new Node("D");
n.next = n1;
n1.next = n2;
```

Arrays Versus Linked Lists

(1) What is really in a Node?

```
class Node {
  String val;
  Node next; // Self-referential!
}// Minimalist!
```

No methods in this class!

Plus and minuses of this design?

(2) How to build up a list of arbitrary length?

- Develop code (insert at end or front)
- See demo

Arrays Versus Linked Lists

Key Operations:

- Traversal
- Insertion
- Deletion

Arrays Versus Linked Lists

Arrays Versus Linked Lists

Case Study from Text Book

Logs: nautical log laboratory logbook
inspection records etc

Case Study from Text Book

`StringLogInterface` (see Eclipse)

- void insert(String)
- boolean isFull()
- int size()
- boolean contains(String)
- void clear()
- String getName()
- String toString()

Case Study from Text Book

Implemented by

`ArrayStringLog`

`LinkedStringLog`

Case Study from Text Book

- Application: Trivia Game
  - Game parameters (class `TrivialGame`):
    - Number of Questions
    - Total number of Chances
    - Number of Correct/Wrong Answers
  - Question parameters (class `TrivialQuestion`:
    - StringLog implementation
    - methods to get questions, store answers, check answers
  - User Interface (class `TriviaConsole`):
    - Opens the file with questions
    - Loops and interact with user

Remarks on Software Design

```
    Repeat (Spiral)
      1.Brainstorm ideas
        (use nouns in problem statement to identify
          potential object classes.
      2.  Identify duties of each class
          and their interactions
      3.  Filter the classes into a set that appears
        to solve the problem.
        Identify members and methods in each class.
      4.  Consider problem scenarios using the classes
    Until the set of classes provides an elegant design
```

Next...

IV. Continuing Lecture

Programmers have more fun

Q: Why is a Java Interface like a joke?

Programmers have more fun

Q: Why is a Java Interface like a joke?

A: If you have to explain it, it is not that good.

Minute Quiz

Write a class called `Mystery` with a `main` method.

- The `main` method simply prints out all the

  command line arguments in reverse order on the console.

- Your program must compile and run.

Minute Quiz

DEMO – develop the answer

Minute Quiz

```
public class Mystery{
   public static void main (String[] args) {
      for (int i=args.length; i>0; i--)
         System.out.println(args[i-1]);
   }//main
}//class
```

– What are alternative ways to reverse order?

More on Linked List

PROBLEM:

Create a linked list with 100 random number

Go through the list, deleting those that are even

More on Linked List

- PROBLEM:
  - Create a linked list with 100 random number
  - Go through the list, deleting those that are even
- Code Development
  - We illustrate how to do optional arguments!
  - Use of random number generators

More on Linked List

```
import java.util.Random;

class Node {
     int data;
     Node next;
     Node(int data) { this.data = data;}
}
```

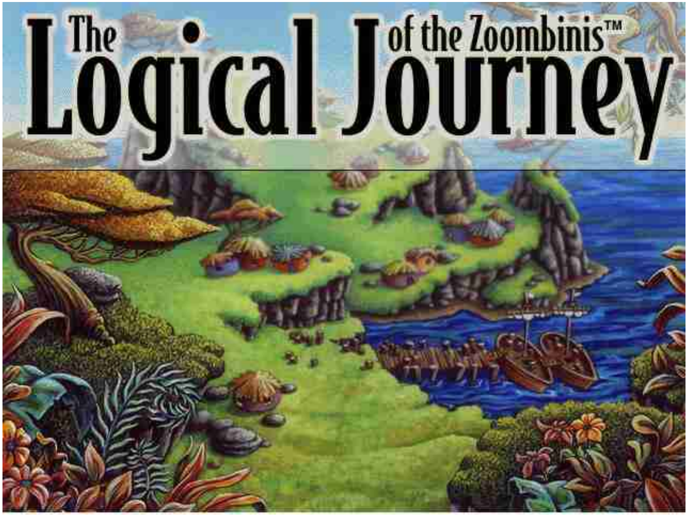
More on Linked List

```
public class RandomList {
      public static void main (String[] args) {
         int n = 10;
         if (args.length > 0)
            n = Integer.parseInt(args[0]);
         int s = 111;
         if (args.length > 1)
            s = Integer.parseInt(args[1]);

         Random gen = new Random(s);

         Node myList = new Node(gen.nextInt(100);
         Node lastNode = myList;
```

More on Linked List

```
          for (int i =1; i<n; i++){
             lastNode.next = new Node(gen.nextInt(100));
             lastNode = lastNode.next;
          }
          lastNode = myList;
          for (int i = 0; i<n; i++) {
             System.out.printf("
             lastNode = lastNode.next;
          }
       }// main
}//class RandomList
```

Next Homework: Zoombinis!!

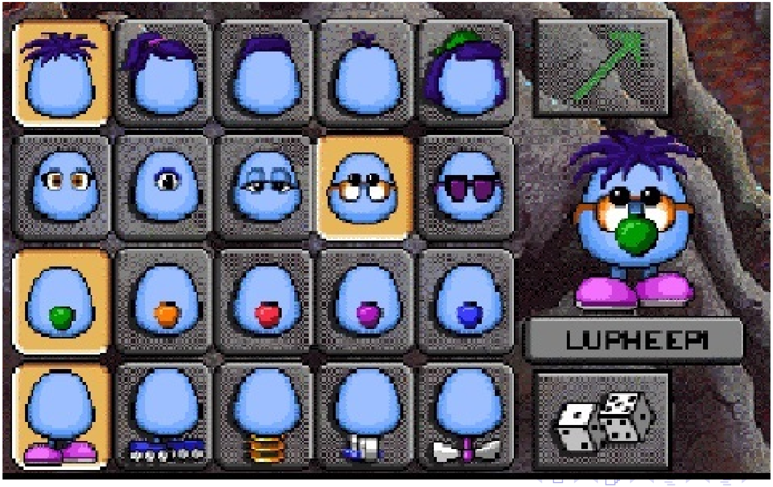


Introduction to Zoombinis

Next Homework: Zoombinis!!

Parental Approval

Wikipedia



Next Homework: Zoombinis!!

The Zoombini name problem

– to generate random names for the Zoombinis. E.g.,

Kejuka, He, Mojejiwijehefa, Mibojafudufe, Felule, Zasefacaga, Rezuragobi, Cepi, Nidi, Ci, Borewi, Puxecaco, Julenoraziniciтuzaxa, Voco, Zi, Laqezetaki, Habape, Fa.

Note that each name is a sequence of

– syllables (i.e., pronounceable)

It is not just fun and games! Ulterior motive?

Next Homework: Zoombinis!!

Let us spice this up!

We also want some culturally appropriate names

– American Names:

Short (one or 2 syllable preferred)

– Chinese Names:

Usually two syllable, but may be one or three.

Preponderance of ”Ch-”, ”Sh-”, ”-ng”, ”-n”.

So we have different ”modes”:

Mode 0 (default, simplistic)

Mode 1 (American)

Mode 2 (Chinese)

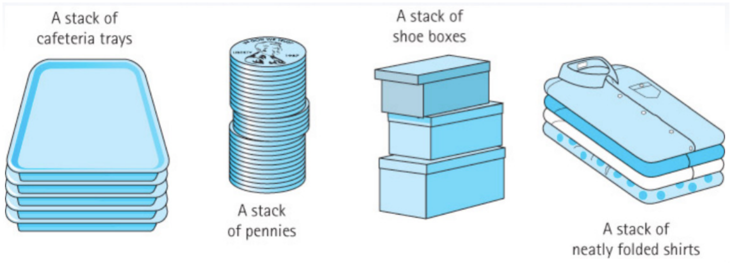
... etc

Next...

[Start] [End]

V. Stacks

What are Stacks?



- LIFO: *last in, first out*
- cf. FIFO or Queue

Stacks as an ADT

- ADT = interface in Java!
- Suppose you want a stack of strings:

```
public interface StringStack {
 public String pop();
 public void push(String);
 public boolean isEmpty();
}
```

- What if you want a stack of integers? or Accounts?

Stacks of Objects

How about a stack of `Objects`?

Stacks of Objects

```
public interface ObjectStack {
 public Object pop();
 public void push(Object);
  ...
}
public class Stack implements ObjectStack {
  ...
}
//==========================================
Stack ss = new Stack();
ss.push("My string"); // implicit casting
String = (String) ss.pop(); // explicit cast!
```

Stacks of Objects

Note:

To cast an object `x` to a subclass `SubType`

you may first check its type using:

```
if (x instance of SubType)
    ...  do cast of x to SubType ....
```

Generic Stack Interface

Alternative to the use of Objects:

use a generic interface

```
public interface Stack<T>{
   public T pop();
   public void push(T x);
   public boolean isEmpty();
   ...
}
```

Generic Stack Interface

Now, we implement the generic interface:

use a generic class

```
public class MyStack<T>
 implements Stack<T>{
  public T pop();
  public void push(T x);
  public boolean isEmpty();
  ...
}
```

Generic Stack Interface

See Eclipse Demo

Based on uploaded files:

MyStack.java, Stack.java, StackEmptyException.java

Exceptions in Stacks

What are the error conditions?

Called `Exceptions`

Exceptions in Stacks

What are the error conditions?

Called `Exceptions`

E.g.,

- pop an `empty` stack
- push to a `full` stack
- push the `wrong` type

Exceptions in Stacks

What are the error conditions?

Called `Exceptions`

- Who handles them?
- What is the right response?

Exceptions in Stacks

Doing this in Java:

(Step 1) Defining the exception (subclass of Java's `Exception`)

(Step 2) Raising the exception

(Step 3) Handling the exception

Exceptions in Stacks

(Step 1): Defining it

```
public class StackEmptyException extends Exception {
 public StackEmptyException() { super() }
 public StackEmptyException(String msg) {
   super(msg);
 }
}
```

You can use this simplistic design for any exception you want to define!

Exceptions in Stacks

(Step 2): Raising it

```
public T pop() {
   if (curr < 0)
      throw new StackEmptyException("Pop of empty stack");
   else
      return( stack[curr--]);
}
```

Exceptions in Stacks

(Step 3) Recall 2 ways to handle exceptions:

Simple way — by “Throwing”

```
public static void main(String[] args)
  throws IOException {
   // Code that throws IOException here...
   // Nothing special to be done!
}//main
```

Local way — by “Catching”

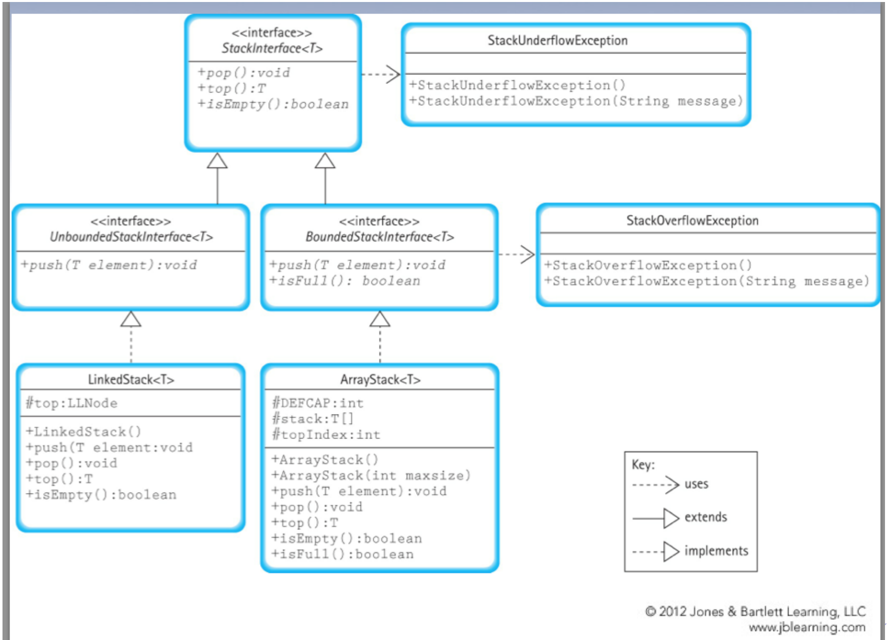
```
public static void main(String[] args) {
  try {
   // Code that throws IOException here...  }
  catch (IOException e) {
   // Do something with e:
   e.printStackTrace(); }
}//main
```

Exceptions in Stacks

Example of handling exception:

```
public static void main(String[] args) {
   MyStack<String> ss = new MyStack<String>();
   String> item;
      ...
   try {
      item = ss.pop();
   }
   catch(StackEmptyException ex){
      item = "";
   }
}
```

Road Map



<<interface>>
StackInterface<T>
+pop():void
+top():T
+isEmpty():boolean
StackUnderflowException
+StackUnderflowException()
+StackUnderflowException(String message)
<<interface>>
UnboundedStackInterface<T>
+push(T element):void
<<interface>>
BoundedStackInterface<T>
+push(T element):void
+isFull(): boolean
StackOverflowException
+StackOverflowException()
+StackOverflowException(String message)
LinkedStack<T>
#top:LLNode
+LinkedStack()
+push(T element:void
+pop():void
+top():T
+isEmpty():boolean
ArrayStack<T>
#DEFCAP:int
#stack:T[]
#topIndex:int
+ArrayStack()
+ArrayStack(int maxsize)
+push(T element):void
+pop():void
+top():T
+isEmpty():boolean
+isFull():boolean
Key:
uses
extends
implements

Road Map

Difference between Array and Linked List Implementation:

Shoppers' Club with 2 kinds of membership

Array Membership: buy in bulk of 100, and pay half member price!

- Use a card to get items at vending machine

List Membership: buy as needed, pay full member price

- Must pay at cashier each time

Road Map

Interfaces can be inherited!

Road Map

Interfaces can be inherited!

```
public interface BoundedStackInterface<T>
   extends StackInterface<T> {
      ...
}
```

Road Map

There is a Java implementation:

`java.util.Stack`

extends `Vectors`

includes other members...

Well-Formed Expression

Matched pairs of parentheses: `( () ( [ ( ) ] ( ( ) ) )`

Several flavors of parentheses:

- `(...)` — regular parentheses
- $\{...\}$ — braces
- `[...]` — square brackets
- `<...>` — angle brackets
- $(_s ...)_s$ — subscripted parentheses, etc

Classifications of parentheses expressions:

- Balanced (0), mismatched (1), open (2)

Well-Formed Expression

Design a class called `Balanced`

Constructor: `Balanced( String leftPar, String rightPar)`

Method: `public int test(String expr)`

Well-Formed Expression

Let us design this!

Must use a stack.

Reverse Polish Notation

Also called: Postfix Notation:

Lukasiewicz (1875-1956)

(a + b) * c

a b + c *

(((a - b) / c) * (d + e))

...

Reverse Polish Notation

GOAL:

design a program to evaluate a a Postfix Expression

Reverse Polish Notation

Relationship between a standard arithmetic expression and postfix notation?

Problem: converting an arithmetic expression into a postfix notation

Final Thoughts

Stacks are central to computer science

Do we really need a stack to check for balanced parenthesis?

Thanks for Listening!

“Algebra is generous,
she often gives more than is asked of her.”

— JEAN LE ROND D’ALEMBERT (1717-83)