Stacks

- Assignment 5:
  - Abstract Classes
  - Hierarchy

- Stacks - an Abstract Data Type
  - Class interface
  - Polymorphism
  - Use of List as representation of Stacks
  - Pop versus Peek
Abstract Classes

• Useful when you want to define only part of an implementation

• Abstract classes
  – Abstract methods are signatures of promised methods to be provided in subclasses of the abstract class
    – Can provide these through definition or inheritance
  – No objects can be created
    – Because abstract method implementations don’t exist
Abstract Classes

• Can define methods (and implementations) in an abstract class which can be inherited by subclasses

• Can also contain instance variables to be inherited by subclasses

• Examples in Assignment 5: `Expr`, `Unary_Expr`, `Binary_Expr`
Assignment 5: Expressions

Object

Expr

abstract eval()
abstract equals()
operator

Const_Expr

operand
c
equals()
eval()

Unary_Expr

operand
equals()
getOperand()

Expr

operator

operand1
operand2
equals()
getOperand1()
getOperand2()

Binary_Expr

Plus_Expr

commute()
eval()

Minus_Expr
eval()

Times_Expr

commute()
eval()

Divide_Expr
eval()
public abstract class Expr extends Object
{
    private String operator;

    Expr(String s)//constructor
    {
        operator = s;
    }

    abstract boolean equals(Expr e);
    abstract int eval();
}
public abstract class Unary_Expr
{
    private Expr operand

    Unary_Expr(Expr e, String s)
    {
        super(s);
        operand = e;
    }

    public Expr getOperand() {.....}
    public String toString() {.....}
    public boolean equals(Expr other) {....}
}
Super

- **Super** acts as a reference to an object as an instance of its superclass

- The reference to **super** in the Unary_Expr class constructor, means call the Expr constructor with argument String s.
  - Implicitly, when a subclass object is created, the constructor of the superclass is called before anything else is done in the subclass constructor
  - If arguments are needed, **super(args)** is used to call this constructor.
public abstract class Binary_Expr extends Expr
{
private Expr operand1, operand2;
Binary_Expr(Expr e1, Expr e2, String s)
{
  super(s);
  operand1 = e1;
  operand2 = e2;
}
public Expr getFirstOperand() {...}
public Expr getSecondOperand() {...}
public String toString() {...}
public boolean equals(Expr other) {...}
}
“Recursive” Objects

• Operands within an expression are themselves expressions
  – $2*3+4$ is a Plus_EXPR constructed from Times_EXPR $e_1$, Const_EXPR $c_1$, and String “+”
    – where $e_1$ is Times_EXPR(2,3,”*”) and $c_1$ is Const_EXPR(4)

• Expr objects with instance variables that are other Expr objects
Expr Objects

- Expression “trees”
  - FirstOperand, SecondOperand

```
  2 * 3 + 4
```

Expression objects:
- Plus_Expr
- Times_Expr
Stacks

• Stacks in real-life
  – Redial button on telephone - calls the last number dialed
  – *history* (his) command in Unix (!! executes your last typed command)
  – Job layoffs of people with least seniority
  – Pile of plates in restaurant
Stack Class Interface

- Instance variables:
  - private List top
  - private int length

- Instance methods:
  - public Stack() //constructor
  - public int getLength() //# of elements
  - public boolean empty()
  - public String toString()
  - public Enumeration getEnumeration()
Stack Methods

public Stack() { //empty stack is top as null List
    top = null;
    length = 0;
}
public int getLength(){ //observer
    return length;
}
public boolean empty(){ //true if length!=0
    return (length == 0);
}
Stack Interface

- **public void push (Object newItem)**
  - adds element newItem to stack
  - polymorphic abstract data type (ADT)

- **public Object pop() throws StackException**
  - removes element from Stack and returns it
  - polymorphic

- **public Object peek() throws StackException**
  - allows examination of top element on Stack without removing it
  - polymorphic
Stack Class: How to build?

• How to represent Stacks?
  – Use List class (first element, rest_of_list) to hold elements in a stack

• Potential special cases
  – Pop off or peek at an empty stack
  – Push onto an empty stack
  – Both can be handled by encoding the empty stack as top == null and length == 0

• Can use length== 0 to check for empty stack
Lists

• A list is a sequence of objects
  – Bad view for thinking about operations on lists
• A list is a pair, a first element and a rest_of_list, which is a sublist
Details here are hidden by List class implementation!
public class List extends Object{
    protected Object info; // field is accessible only
    protected List subList; // by classes in package
        // means field is private to package

    public List{
        info = null;
        subList = null;
    }

    public List (Object element, List oldList){
        info = element;
        subList = oldList;
    }

    (Note: design in cs111.util.* differs slightly from this)
List Construction

element: A

oldList:

result:

new list
Push onto empty stack

Initially,

- top is null
- length is 0

Perform push(E)

List nl = new List(newItem, top)
        top = nl;
Push Method

//create new List with old List as subList and
//newItem as first element
public void push(Object newItem){
    List nl = new List(newItem, top);
    top = nl;
    length++;
}

Push onto non-empty Stack

Initially, `E` then, push( `F`)

length is 1

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>E</td>
</tr>
<tr>
<td></td>
<td>λ</td>
</tr>
</tbody>
</table>
```

length is 2
public Object pop() throws StackException{
    if (empty()) throw new StackException
         ("Attempt to pop from empty Stack");
    Object ret = top.info;
    top = top.subList;
    length--;
    return ret;
}
**Pop off empty stack**

Initially,
- top is null
- length is 0
- empty() yields true

**Pop off non-empty stack**

Initially,
- top
- length is 1

pop()
- return E
- top is null
- length is 0

Object ret = top.info;
- top = top.subList;
- length--;
- return ret;
User-defined Exception

public class StackException extends Exception{
    String msg;
    StackException (String str){
        msg = str;
    }
}

• Define as extension of built-in class Exception
• Pass StackException object with private String instance variable to exception handler for possible printing
• No handler in Stack class means user of Stack class can handle or pass along to default handler in class Object
Pop() versus Peek()

public Object pop() throws StackException{
    if (empty()) throw new StackException
        (“Attempt to pop from empty Stack”);
    Object ret = top.info;
    top = top.subList;
    length--;
    return ret;
}

public Object peek() throws StackException{
    if (empty()) throw new StackException
        (“Attempt to peek at an empty Stack”);
    return top.info;
}
toString Method

//uses toString() method in Lists to return contents
//of Stack
public String toString(){
    String ret = "Stack length is " + length + "\n";
    return ret + "stack is: " + top.toString;
}