SpamSeeker; Improving on StringStack

February 23–24, 2012
CS 60: Principles of Computer Science

Assignment 5 due Monday February 20: Spamseeker
**Review: Stacks and Queues**

*Which terms go together?*

- dequeue
- enqueue
- pop
- push

Stack

Queue

FIFO
FILO
LIFO
LILO
SpamSeeker!

class Maze
{
    private MazeCell[][] maze;
class MazeCell {
    private int row;
    private int col;
    private char contents;
    private boolean visited;
    private MazeCell parent;

    MazeCell(int row, int col, char c) {
        this.row = row;
        this.col = col;
        this.contents = c;
        this.visited = false;
        this.parent = null;
    }

    public String toString() {
        return "[" + row + "," + col + "," + contents + "]";
    }
}

Here are the available characters:
'S' = Start Spam Seeking       '*' = Wall!
'D' = Delectible Dinner Destination   ' ' = Open Space
**Pseudocode for DFS in a Maze**

Create an empty Stack
Mark starting MazeCell as visited
push starting MazeCell onto our Stack

while (the stack's not empty)
{
    current = pop the stack
    for each of current's neighbors
    {
        if (it's not visited or a wall)
        {
            mark it (neighbor) as visited
            set its parent to current MazeCell
            push it onto the stack
        }
    }
}
## Pseudocode for BFS in a Maze?

Create an empty Queue
Mark starting MazeCell as visited
enqueue starting MazeCell in our Queue

while (the queue's not empty)
{
    current = dequeue the queue
    for each of current's neighbors
    {
        if (it's not visited or a wall)
        {
            // mark it (neighbor) as visited
            set its parent to current MazeCell
            enqueue it in the queue
        }
    }
}
**Pseudocode for DFS in a Maze, Revisited**

DFS(MazeCell start)
{
    if (start isn't visited or a wall)
    {
        mark it as visited
        DFS(southNeighbor)
        DFS(eastNeighbor)
        DFS(westNeighbor)
        DFS(northNeighbor)
    }
}

Where'd the stack go?

'S' = Start Spam Seeking
'D' = Delectible Dinner Destination
**Review: StringStack**

```java
public class StringStack extends Object {

    private class StackCell {
        private String data;
        private StackCell next; ...
    }

    private StackCell top;

    public StringStack() { ... }
    public void push(String data) { ... }
    public String pop() { if (this.isEmpty()) return null;
                             String topItem = this.top.data;
                             this.top = this.top.next;
                             return topItem; }
    public String peek() { ... }
    public boolean isEmpty() { ... }
}
```
Stacks as Linked Lists

Here is memory changing over time. What code is executing?

1. \[ \text{st} \]

2. \[ \text{st} \rightarrow \text{top} \]
   
   \[ \text{StringStack} \]

3. \[ \text{st} \rightarrow \text{top} \]
   
   \[ \text{StringStack} \rightarrow \text{StackCell} \]

4. \[ \text{st} \rightarrow \text{top} \]
   
   \[ \text{StringStack} \rightarrow \text{StackCell} \]

5. \[ \text{st} \rightarrow \text{top} \]
   
   \[ \text{StringStack} \rightarrow \text{StackCell} \]

6. \[ \text{st} \rightarrow \text{top} \]
   
   \[ \text{StringStack} \]

---

**SpamSeeker**

**Linked Data**

**Generics**

**Interfaces**

**StringBuffer**
Queues via Linked Lists

(Simplified Diagram)
NAME:

Draw memory after each line. How many references change at each step?

1. `StringQueue bert;`
2. `bert = new StringQueue();`
3. `bert.enqueue("U");`
4. `bert.enqueue("C");`
5. `bert.dequeue();`
6. `bert.dequeue();`

3.

4.

5.

6.
“Industrial Strength” Stacks and Queues

February 23–24, 2012
CS 60: Principles of Computer Science
Problems with StringBuilder

The **StringBuilder** class you’ve seen works. But it’s not perfect:

1. What if we want a stack of **Points**, or a stack of **MazeCells**?

2. It blurs interface and implementation.

3. The `toString()` method is really slow.
How Many Changes are Required to Make a PointStack?

```java
public class PointStack extends Object { // 1
{
    private class StackCell {
        private Point data; // 2
        private StackCell next; ... }

    private StackCell top;

    public PointStack() { ... } // 3
    public void push(Point data) { ... } // 4
    public Point pop() { if (this.isEmpty()) return null; // 5
    Point topItem = this.top.data; // 6
    this.top = this.top.next;
    return topItem; }

    public boolean isEmpty() { ... }
    public String toString() { ... } // careful!
...}
```
**ALTERNATIVE 1: ObjectStack**

```java
public class ObjectStack extends Object {
    private class StackCell {
        private Object data;
        private StackCell next; ...
    }

    private StackCell top;

    public ObjectStack() { ... }

    public void push(Object data) { ... }
    public Object pop() { if (this.isEmpty()) return null;
        Object topItem = this.top.data;
        this.top = this.top.next;
        return topItem; }

    public Object peek() { ... }
    public boolean isEmpty() { ... }
}
```
**ADVANTAGES!**

*We can use ObjectStack with any sort of objects!*

ObjectStack st1 =
   new ObjectStack();

st1.push("1");
st1.push("2");
st1.push("3");
st1.push("4");

ObjectStack st2 =
   new ObjectStack();

st2.push(new Point(1,2));
st2.push(new Point(2,4));
st2.push(new Point());
st2.push(new Point(4,8));
Disadvantages

We can use `ObjectStack` with any sort of objects!

```
ObjectStack st1 = new ObjectStack();

st1.push("1");
st1.push("2");
st1.push("3");
st1.push("4");

String four = st1.pop(); // Doesn't compile! Why not?
// String four = st1.pop(); <--- Doesn't compile! Why not?
Object four = st1.pop();
String three = (String) st1.pop();

st1.push(new Point());
```
**Alternative 2: Write a Generic Stack Class**

```java
Stack<String> st1 = new Stack<String>();
st1.push("1");
st1.push("2");
st1.push("3");
st1.push("4");
String four = st1.pop();
```

```java
Stack<Point> st2 = new Stack<Point>();
st2.push(new Point(1,2));
st2.push(new Point(2,4));
st2.push(new Point());
st2.push(new Point(4,8));
Point p = st2.pop();
```
A Generic Stack

```java
public class Stack<T extends Object> extends Object {
    private class StackCell {
        private T data;
        private StackCell next; ...
    }

    private StackCell top;

    public Stack() { ... } // Constructor just called "Stack"
    public void push(T data) { ... }
    public T pop() { if (this.isEmpty()) return null;
        T topItem = this.top.data;
        this.top = this.top.next;
        return topItem; }
    public T peek() { ... }
    public boolean isEmpty() { ... }
}
```
**Warning**

Generics were added to the language in Java 5.

The Java Virtual Machine that runs code still doesn’t know about generics!

✓ Compiler checks that you’re using your generic stack correctly.

✓ But, the generated code only has stacks of `Object`, with the required typecasts inserted automatically.

As a consequence, there are a few things you might expect to be able to do with generics, but can’t. We shouldn’t run into these issues in CS 60.
Interfaces in Java
Suppose we have program that uses `StringStack`

```java
public class StringStack extends Object {
    private class StackCell { ... }
    private StackCell top;
    // ...constructor and methods...
}

public static boolean test(StringStack st) {
    st.push("a");
    st.push("b");
    boolean test1 = (st.pop() == "b");
    boolean test2 = (st.pop() == "a");
    boolean test3 = st.isEmpty();
    return (test1 && test2 && test3);
}
```
What if We Also Have a Different Implementation...  

Can we use both kinds of stack with test? Would generics help? Inheritance?

public class StringStack extends Object { ...

public class StringStack2 extends Object
{
    private ArrayList<String> items;    // growable array of strings

    public void push(String s) { items.add(s); }
    public String pop() { return (items.remove(items.size() - 1)); }
    public boolean isEmpty() { return (items.size() == 0); }
    // ...other methods...
}

public static boolean test(StringStack st)
{
    st.push("a");
    st.push("b");
    boolean test1 = (st.pop() == "b");
    boolean test2 = (st.pop() == "a");
    boolean test3 = st.isEmpty();
    return (test1 && test2 && test3);
}
**Solution: interfaces**

interface StringStackInterface
{
    public boolean isEmpty();
    public String peek();
    public String pop();
    public void push(String data);
    public String toString();
}

class StringStack extends Object implements StringStackInterface
{
    ...
}

class StringStack2 extends Object implements StringStackInterface
{
    ...
}

public static boolean test(StringStackInterface st)
{
    st.push("a");
    ...
}
**INTERFACES CAN BE GENERIC TOO**

```java
interface StackInterface<T extends Object> {
    public boolean isEmpty();
    public T peek();
    public T pop();
    public void push(T data);
    public String toString();
}

class Stack<T extends Object> extends Object implements StackInterface<T> {
    { ... }
}

class Stack2<T extends Object> extends Object implements StackInterface<T> {
    { ... }
}

public static boolean test(StackInterface<String> st) {
    st.push("a");
    ...
}
```
Speeding up `toString`
**Strings and String Concatenation**

Strings (string objects) in Java:

- ✓ contain a sequence of (unicode) characters
- ✓ are immutable once created.

What is the asymptotic running time of string concatenation?

\[ s_1 + s_2 \]

\[ O(n + m), \text{ where } n \text{ is the length of } s_1 \text{ and } m \text{ is the length of } s_2 \]
What is the asymptotic running time of this `toString()` method?

```java
public String toString()
{
    StackCell current = this.top;

    String accum = "<TOP> ";

    while (current != null)
    {
        accum += " " + current.data.toString() + " ";
        current = current.next;
    }

    accum += "<BOT>";

    return accum;
}
```

(You may assume we’re adding $O(1)$ characters to `accum` in every iteration of the loop.)
Better Java Style: Use StringBuffer to Grow a String

StringBuffer objects are mutable/growable strings. Most importantly, adding one character at the end is $O(1)$!