
Keeping an ArrayList sorted

Oct 13

To keep in sorted order

- Figure out where something should be put
 - $O(n)$
- put it there
 - $O(n)$
- Overall Complexity for 1 add
 - $O(n) + O(n) = O(n)$
- Complexity for N add
 - $O(n) * O(n) = O(n^2)$

Code it Out

Sal.java

SAL looks a lot like ArrayList

- Should / can “extend ArrayList”
- Can — yes
- Should — almost certainly
- Why NOT?
 - I already have something working
 - lame
 - ArrayList provides lots of functions I do not want to give my users
 - reasonable (possibly)

More code

SALextending.java

Objects and Inheritance

Classes

Abstract Classes

Interfaces

Class

- `public class ABCD`
- by default extends `Object`
 - instance variables
 - methods
- Create instances using `new`
- Extending
 - `public class ABCDE extends ABDC`

Interfaces

- `public interface ABDC`
- Method names but no implementation
- NO instance variables
- cannot be instantiated
 - no "new"
- `public class ABCDE implements ABCD`
 - implementation must have method bodies

Abstract Classes

- `public abstract class ABDC`
- instance variables
- Methods
 - either full or just signature
 - if just signature must say "abstract"
 - `public abstract int getVal();`
- no "new"
- `public class ABCDE extends ABCD`
 - must provide bodies for abstract methods

Why Interfaces and Abstract Classes

- Abstract class
 - can provide method bodies
 - can provide important variables
 - limited to single inheritance
- Interface
 - no method bodies
 - no instance variables
 - multiple inheritance

Multiple Inheritance Problems

```
public class AA {  
    private int x;  
    public String whoAmI() {  
        return "I am me!";  
    }  
    public int doV() {  
        return 2;  
    }  
}
```

```
public class BB {  
    private String x;  
    public int whoAmI() {  
        return 42;  
    }  
    public int doV() {  
        return 1;  
    }  
}
```

- public class CC extends AA, BB
 - What is the type of x?
 - What type does whoAmI return?
 - What does doV return? (RUNTIME)

Multiple Interface Problems

```
public interface A1 {  
    String whoAmI();  
}  
  
public interface A2 {  
    int whoAmI();  
}
```

- `public class A12 implements A1, A2 { ...`
 - what is return type of `whoAmI`?
- So why does Java allow implementation of multiple interfaces?
 - It is really useful
 - Issues are detectable at compile time

Geometric Object

```
public abstract class GeometricObject {
    private String color;
    private String name;
    private boolean filled;
    abstract int getPerimeter();
    abstract int getArea();
    public String toString() {
        return "This is a " +
(filled?"filled ":"") + color + " " + name
+ " with perimeter " + getPerimeter() + "
and area " + getArea();
    }
}
```

Given that an abstract class cannot be instantiated, can/should it have a constructor?

Circle

```
public class Circle extends GeometricObject{
    private int radius;
    public Circle(int radius) {
        this.radius = radius;
    }
    public int getPerimeter() {
        return (int)(radius*2*Math.PI);
    }
    public int getArea() {
        return (int)(radius*radius*Math.PI);
    }
    public static void main(String[] args) {
        Circle c = new Circle(5);
        System.out.println(c);
    }
}
```

Square

```
public class Square extends GeometricObject{
    private int side;
    public Square(int side) {
        this.side=side;
    }
    public int getPerimeter() {
        return 4*side;
    }
    public int getArea() {
        return side*side;
    }
    public static void main(String[] args) {
        Square sq = new Square(5);
        System.out.println(sq);
    }
}
```

Problems?

Solutions?

Regular Polygon Class

```
public class RegularPolygon extends GeometricObject {
    int numSides;
    int sideLength;
    public RegularPolygon(int n, int l) {
        sideLength=l;
        numSides=n;
    }
    int getPerimeter() {
        return numSides*sideLength;
    }
    int getArea() {
        return (int)((sideLength*sideLength*numSides) /
(4*Math.tan(Math.PI/numSides)));
    }
}
```

Should this class be abstract?
Should the Square class even still exist?
if yes, adjustments?
Circle?

Square/RP Adjustments

- Almost everything in Square goes away
- Problem: what to do at RP so can have a single method that constructs / returns a RP or a Square?

```
public class Square extends RegularPolygon {  
    public Square(int l) {  
        super(4, l);  
        this.name="square";  
    }  
}
```

RP & Square Adjustments

A static “Builder” method

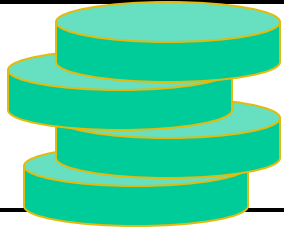
A protected constructor

No One can call the constructor directly

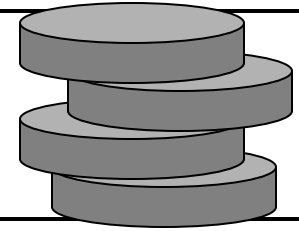
```
public static RegularPolygon regularPolygonBuilder(int n, int l) {  
    if (n==4)  
        return new Square(l);  
    else  
        return new RegularPolygon(n, l);  
}  
protected RegularPolygon(int n, int l) {  
    sideLength=l;  
    numSides=n;  
}
```

Stacks

Oct 13



Stacks



- Insertion and deletions are First In Last Out
 - FILO
 - or LIFO
- Physical stacks are everywhere!
- Function names (in the following slides) follow `java.util.Stack` rather than Goodrich.

Stack Interface

- How do you inform user of stack that it is empty peek and pop?
 - throw exception?
 - return null?
 - Something else?
- REQUIREMENT
 - every method $O(1)$

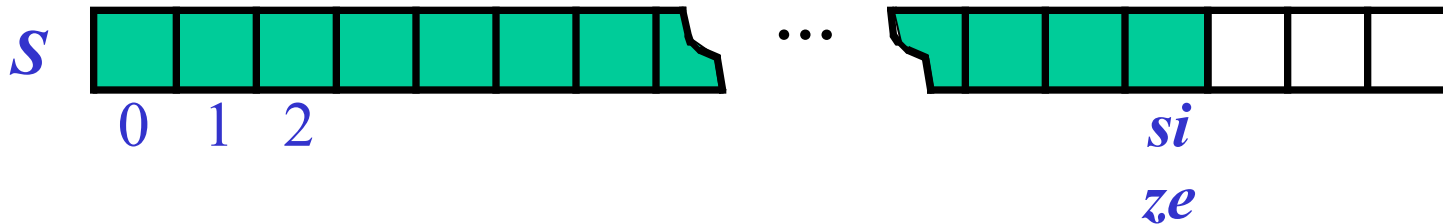
```
public interface StackInft<E> {  
    public boolean empty();  
    public E push(E e);  
    public E peek();  
    public E pop();  
    public int size();  
}
```

Example

Method	Return Value	Stack Contents
push(5)	5	{5}
push(3)	3	{5, 3}
size()	2	{5, 3}
pop()	3	{5}
empty()	FALSE	{5}
pop()	5	{}
empty()	TRUE	{}
pop()	null	{}
push(7)	7	{7}
push(9)	9	{7,9}
peek()	9	{7,9}

Array-based Stack

- Implement the stack with an array
- Add elements onto the end of the array
- Use an int `size` to keep track of the top



Performance and Limitations of Array Stack

- Performance

- let n be the number of objects in the stack
- The space used is $O(n)$
- Each operation runs in time $O(1)$

- Limitations

- Max size is limited and can not be changed
- Pushing onto a full stack will fail
 - need to handle that

Why not ArrayList?

- Every operation in Array stack is $O(1)$
- NOT true for ArrayList
 - Consider grow
- So if want $O(1)$ guarantee for Stack cannot use ArrayList.
- For now, bound to array which means
 - fixed size
 - wasted space

Push

- Array has set size and may become full
- A `push` will fail if the array becomes full
 - Limitation of the array-based implementation
 - Alternatives?
 - Make the array grow (use `ArrayList`)?
 - why not?
 - What do to on fail?
 - return null
 - throw exception

Implementing an Array-based stack

```
public class ArrayStack<K> implements StackIntf<K> {
    private static final int DEFAULT_CAPACITY = 40;
    private int size;
    private K[] underlyingArray;

    public ArrayStack() {
        this(DEFAULT_CAPACITY);
    }

    public ArrayStack(int capacity) {
        size=0;
        underlyingArray = (K[]) new Object[capacity];
    }
}
```

Method Stack in the JVM

- The JVM keeps track of the chain of active methods with a stack
 - `printStackTrace()` — only within catch block of exception
 - `Thread.dumpStack()` — anywhere
- On a method call, the JVM pushes onto the stack a frame containing:
 - parameters
 - local variables
 - return address
- When a method ends, control passes onto the method on top of the stack
- Using VSC to view the stack — `MethodStack.java`

Stack Applications

- Reversing
- Prefix/postfix algebraic interpreter
- Palindromes
 - Madam Im adam
 - A man a plan a canal panama!
 - Dennis, Nell, Edna, Leon, Nedra, Anita, Rolf, Nora, Alice, Carol, Leo, Jane, Reed, Dena, Dale, Basil, Rae, Penny, Lana, Dave, Denny, Lena, Ida, Bernadette, Ben, Ray, Lila, Nina, Jo, Ira, Mara, Sara, Mario, Jan, Ina, Lily, Arne, Bette, Dan, Reba, Diane, Lynn, Ed, Eva, Dana, Lynne, Pearl, Isabel, Ada, Ned, Dee, Rena, Joel, Lora, Cecil, Aaron, Flora, Tina, Arden, Noel, and Ellen sinned.
- Program runtime
- OS Tasks