Stacks

Oct 13

Regular Polygon Class

```
public class RegularPolygon extends GeometricObject {
    int numSides;
    int sideLength;
    public RegularPolygon(int n, int l) {
        sideLength=l;
        numSides=n;
    }
    int getPerimiter() {
        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";
    }
}
```

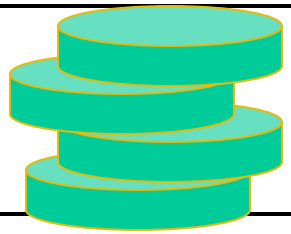
RP & Square Adjustments

A static “Builder” method

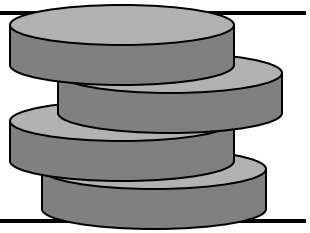
A protected constructor

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



- 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)

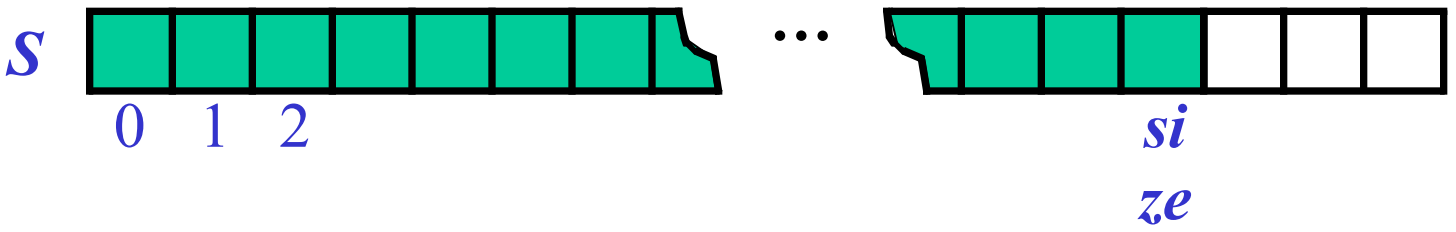
```
public interface StackInft<E> {
    public boolean empty();
    public E push(E e);
    public E peek();
    public E pop();
    public int size();
}
```

Example

Method	Return Value	Stack Comtents
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

- Palindromes

  file: palindromer.java

  - 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.