CS 61B   Discussion 6   Spring 2016

1 Immutable Rocks

A class is immutable if nothing about its instances can change after they are constructed. Which of the following classes are immutable?

```
1  public class Pebble {
2      public int weight;
3      public Pebble() { weight = 1; }
4  }
5  public class Rock {
6      public final int weight;
7      public Rock (int w) { weight = w; }
8  }
9  public class Rocks {
10     public final Rock[] rocks;
11     public Rocks (Rock[] rox) { rocks = rox; }
12 }
13 public class SecretRocks {
14     private Rock[] rocks;
15     public SecretRocks(Rock[] rox) { rocks = rox; }
16 }
17 public class PunkRock {
18     private final Rock[] band;
19     public PunkRock (Rock yRoad) { band = {yRoad}; }
20     public Rock[] myBand() {
21         return band;
22     }
23 }
24 public class MommaRock {
25     public static final Pebble baby = new Pebble();
26 }
```

2 Breaking the System

Below is the SNode class, along with a flawed implementation of the Stack ADT, BadIntStack.

```
1  public class SNode() {
2      public Integer val;
3      public SNode prev;
4
5      public SNode(Integer v, SNode p) {
6          val = v;
7          prev = p;
8      }
9      public SNode(Integer v) {
10         this(v, null);
11     }
12 }
```

```java
1  public class BadIntStack {
2      public SNode top;
3
4      public boolean isEmpty() {
5          return top == null;
6      }
7      public void push(Integer num) {
8          top = new SNode(num, top));
9      }
10     public Integer pop() {
11         Integer ans = top.val;
12         top = top.prev;
13         return ans;
14     }
15     public Integer peek() {
16         return top.val;
17     }
18 }
```

Fill in the Exploiter1 class so that it prints *"Success!"* by causing BadIntStack to produce a NullPointerException.

```java
class Exploiter1 {
    public static void main(String[] args) {
        try {
            // Your exploit here!



        } catch (NullPointerException e) {
            System.out.println("Success!");
        }
    }
}
```

Now, fill in the client class Exploiter2 so that it creates an "infinitely long" stack.

```java
class Exploiter2 {
    public static void main(String[] args) {
        BadIntStack trap = new BadIntStack();
        // Your exploit here!



        while(!trap.isEmpty()) {
            trap.pop();
        }
        System.out.println("This print statement is unreachable!");
    }
}
```

How can we change the BadIntStack class so that it won't throw NullPointerExceptions or allow ne'er-do-wells to produce endless stacks?

3 Design a Parking Lot!

Design a `ParkingLot` package that allocates specific parking spaces to cars in a smart way. Decide what classes you'll need, and design the API for each. Time permitting, select data structures to implement the API for each class. Try to deal with annoying cases (like disobedient humans).

- Parking spaces can be either regular, compact, or handicapped-only.

- When a new car arrives, the system should assign a specific space based on the type of car.

- All cars are allowed to park in regular spots. Thus, compact cars can park in both compact spaces and regular spaces.

- When a car leaves, the system should record that the space is free.

- Your package should be designed in a manner that allows different parking lots to have different numbers of spaces for each of the 3 types.

- Parking spots should have a sense of closeness to the entrance. When parking a car, place it as close to the entrance as possible. Assume these distances are distinct.