Iterators and Exceptions

- Resources available:
  kevin-miao.com

Today

- Mini-Review
- Quiz Review
  + What does Java print
  + Iterators
- Worksheet

Exceptions and Error Handling

- Try-Catch blocks

```
Scanner scanner = new Scanner(System.in);
int k;
try {
    k = scanner.nextInt();
} catch (NoSuchElementException e) {
    // Ran out of input
} catch (InputMismatchException e) {
    // Token isn't an integer
} finally {
    // finally will be executed as long as JVM does not exit early
    scanner.close();
}
```

```
int k;
try (Scanner scanner = new Scanner(System.in)) {
    k = scanner.nextInt();
} catch (NoSuchElementException e) {
    // ran out of input
} catch (InputMismatchException e) {
    // token isn't an integer
}
```

- Throwing Exceptions

```
class cookies {
    int num_cookies;
    int price;

    int price(int x) throws Exception {

      if (x<0 || x-num_cookies<0) {
        return new Exception();
      }
      num_cookies -= x;
      return x * price
    }
}
```

Iterators

Analogous to Comparators and Comparables.

Iterables – Objects that can be iterated over

Java Collections → Sets
↓ ↓ ↘
Hashmaps Lists Arrays

```
Iterable<T> {
    Iterator<T> iterator(); }
```

```
public Interface Iterator<T> {
    boolean hasNext();
    T next();
    remove();
    forEachRemaining;
}
```

Quiz Review

Q1 Java Visualizer

Q2

```
private class BadAListIterator implements Iterator<Item> {
    private int bookmark = 0;
    private boolean done = false;

    public boolean hasNext() {
        if (done) {
            return false;
        }
        if (bookmark == size - 1) {
            done = true;
        }
        return true;
    }

    public Item next() {
        Item rtn = values[bookmark];
        bookmark += 1;
        return rtn;
    }
}

"""Q1"""
public static main void (String[] args){
Iterator<Integer> iter = list.iterator();
boolean b;
b = iter.hasNext();
b = iter.hasNext();}
"""Q2"""
public static main void (String[] args){
  Iterator<Integer> iter = list.iterator();
  System.out.println(iter.next());
  System.out.println(iter.next());}
"""Q3"""
public static main void (String[] args){
  Iterator<Integer> iter = list.iterator();
  boolean b;
  int n;
  n = iter.next();
  b = iter.hasNext();}
```

CS 61B Summer 2020

Small Group Tutoring
Section 5 Exceptions, Iterators, and Iterables

Worksheet 5

1 Pusheen Exceptions

Below is a class that represents a Pusheen. Pusheen cares about two things: happiness and food. Her happiness is directly proportional to how much she is fed.

```
public class Pusheen {
    public int happiness;

    public Pusheen() {
        happiness = 0;
    }

    public void feed(int amount) {
        happiness = 14 * amount;
    }
}
```

Unfortunately, some Pusheen haters have decided to try and feed Pusheen a negative amount! Obviously, we must prevent this from happening.

Modify the `feed` method to throw an `InvalidPusheenException` if Pusheen is fed with a negative amount. Being fed a negative amount should **NOT** change Pusheen's happiness.

```
public void feed(int amount) throws InvalidPusheenException {









}
```

2 Exceptions

What does Java display when the main method of Test is run?

```
class Test
{
    String str = "a";

    public void A()
    {
        try
        {
            str +="b";
            B();
        }
        catch (Exception e)
        {
            str += "c";
        }
    }
    public void B() throws Exception
    {
        try
        {
            str += "d";
            C();
        }
        catch(Exception e)
        {
            throw new Exception();
        }
        finally
        {
            str += "e";
        }

        str += "f";

    }
    public void C() throws Exception
    {
        throw new Exception();
    }
    public void display()
    {
        System.out.println(str);
    }
    public static void main(String[] args)
    {
        Test object = new Test();
        object.A();
        object.display();
    }
}
```

3 Pizza Iterator

Artichoke's is overwhelmed by the number of hungry students in line at 12 AM. To make things more efficient, the owner has asked you to build a custom iterator that will aggregate all orders and print out the number of slices that should made for each kind of pizza.

The static menu array declared inside `PizzaIterator` contains the three types of pizza offered that night.

```
static String[] menu = {"Artichoke", "Margherita", "Meatball"};
```

The input array passed into the constructor contains the list of orders.

```
int [] orders = { 0 , 2 , 1 , 0 , 1 , 0 };
```

Each order is represented by an integer that corresponds to the pizza's index in the menu array. For example, 0 represents an order of Artichoke pizza.

Fill in the code for `MenuIterator`, an iterator that takes in an `int[]` array representing orders at the restaurant and iterates over the aggregated results.

Given the input above, calls to `next()` would eventually return "Artichoke 3","Margherita 2", "Meatball 1". Make sure your iterator adheres to standard iterator rules.

```
public class MenuIterator implements Iterator {
    private static String[] menu = {"Artichoke", "Margherita", "Meatball"};
    private int[] order_counts = new int[3];
    private int index;

    public MenuIterator(Integer[] orders){




    }

    public boolean hasNext() {


    }

    public String next() {
        //Should return a string in the format "Artichoke 3".




    }
}
```