Recitation 3

2D Arrays, Exceptions
2D arrays

Many applications have multidimensional structures:
- Matrix operations
- Collection of lists
- Board games (Chess, Checkers)
- Images (rows and columns of pixels)
- ...
1D Array Review

```java
Animal[] pets = new Animal[3];

pets.length is 3
pets[0] = new Animal();
pets[0].walk();
```

Why is the following illegal?

```java
pets[1] = new Object();
```
Java arrays vs Python lists

Java arrays do not change size!

```java
String[] b = {"Cornell", "Ithaca"};
String[] bBig = Arrays.copyOf(b, 4);
b = bBig;
```
Java array initialization

Instead of

```java
int[] c = new int[5];
```

Use an array initializer:

```java
int[] c = new int[] {5, 4, 7, 6, 5};
```

Note: The length of `c` is the number of values in the list.
Exercise 1: Looping over an array

/** Return index of occurrence number n of t in b.
 *  Precondition: n >= 1.
 *  Return -1 if not found. */

public static int get(int[] b, int n, int t) {
    ...
}

get(new int[]{2110, 0, 1, 2110, 2110}, 2, 2110);  
would return 3
2D arrays: An array of 1D arrays.

Java only has 1D arrays, whose elements can also be arrays.

```java
int[][] b = new int[2][3];
```

This array has 2 `int[]` arrays of length 3 each.
2D arrays: An array of 1D arrays.

How many rows in \( b \)? \( b.length \)
How many columns in row 0? \( b[0].length \)
How many columns in row 1? \( b[1].length \)
2D arrays: An array of 1D arrays.

```java
int[][] b = new int[2][];
```

The elements of `b` are of type `int[]`. 
2D arrays: An array of 1D arrays.

```java
int[][] b = new int[2][];
b[0] = new int[] {0,4,1,3,9,3};
b[1] = new int[] {1110,2110,3110};
```

b is called a ragged array
Exercise 2: Transpose Matrix

\[
A = \begin{bmatrix}
  a_{11} & a_{12} & a_{13} \\
  a_{21} & a_{22} & a_{23} \\
  a_{31} & a_{32} & a_{33}
\end{bmatrix}
\]

\[
A^T = \begin{bmatrix}
  a_{11} & a_{21} & a_{31} \\
  a_{12} & a_{22} & a_{32} \\
  a_{13} & a_{23} & a_{33}
\end{bmatrix}
\]

\[(A_{[i][j]})^T\text{ is } A_{[j][i]}\]
Exceptions
Exceptions make your code crash

public static void main(String[] args) {
    System.out.println(args[0]);
}

public static void main(String[] args) {
    System.out.println(8 / 0);
}

public static void main(String[] args) {
    System.out.println(null.toString());
}
public static double getAverage(double[] b) {
    double sum = 0;
    for (int i = 0; i < b.length; i++) {
        sum += b[i];
    }
    return sum / b.length;
}

If \texttt{b.length} is 0, what should be returned?
- Infinity
- “special” int: \texttt{Integer.MAX\_VALUE}\?
- 2110\?
- 0?
The superclass of exceptions: Throwable

class Throwable:

- Superclass of Error and Exception
- Does the “crashing”
- Contains the constructors and methods
- Throwable()
- Throwable(String)

class Error:

- A very serious problem and should not be handled
  Example: StackOverflowError

class Exception:

- Reasonable application might want to crash or handle the Exception in some way
A Throwable instance: ArithmeticException

There are so many exceptions we need to **organize** them.

- **ArithmeticException@x2**
  - Throwable
    - detailMessage: 
      - "/ by zero"
  - Exception
  - RuntimeException
  - ArithmeticException

### Diagram:

```
Throwable
  Exception
  RuntimeException
  ArithmeticException
```
Bubbling up exceptions

Exceptions will bubble up the call stack and crash the methods that called it.

Method call: `first();`

Console:
Exception in thread "main"
java.lang.ArithmeticException: 
at Ex.third(Ex.java:11)
at Ex.second(Ex.java:7)
at Ex.first(Ex.java:3)

AE = ArithmeticException
Decoding the output from an exception

```java
public static void main(String[] args) {
    int div = 5/0;
}
```

Exception in thread "main" java.lang.ArithmeticException: / by zero
at Animal.main(Animal.java:2)
# Try-catch blocks

An exception will bubble up the call stack and crash the methods that called it … unless it is caught.

**catch** will handle any exceptions of type *Exception* (and its subclasses) that happened in the **try** block.

```java
class Ex {
    void first() {
        second();
    }
    void second() {
        try {
            System.out.println("in");
            third();
            System.out.println("out");
        } catch (Exception e) {
            System.out.print("error");
        }
    }
    void third() {
        int c = 5/0;
    }
}
```

**Console:**

- in
- error

**Exception Type**

```
Exception Type
ArithmeticException!
```
throw keyword: Forcing a crash

Why might I want to crash the application?

parseInt("42") -> 42
parseInt("Sid") -> ???

class Integer {
    /** Parse the string argument as a
    * signed decimal integer. Throw a
    * NumberFormatException if not possible
    */
    public static int parseInt(String s){
        if (can’t convert to int){
            throw new NumberFormatException();
        }
        ...
    }
}
Exercise 3: Illegal Arguments

Create `class Person` with two fields, `name` and `age`. Throw an `IllegalArgumentException` instead of having preconditions when given a `null` name or a non-positive age.
How to write an exception class

/** An instance is an exception */
public class OurException extends Exception {

    /** Constructor: an instance with message m*/
    public OurException(String m) {
        super(m);
    }

    /** Constructor: an instance with no message */
    public OurException() {
        super();
    }
}
**Class to illustrate exception handling**

```java
public class Ex {
    public static void main() { 
        try { second(); } catch (OurException e) {} 
    }
    public static void second() throws OurException { 
        third();
    }
    public static void third() throws OurException { 
        throw new OurException("mine");
    }
}
```

If you're interested in the "controversy", [http://docs.oracle.com/javase/tutorial/essential/exceptions/runtime.html](http://docs.oracle.com/javase/tutorial/essential/exceptions/runtime.html)
Demo 1: Pythagorean Solver

- Given $a$ and $b$: solve for $c$ in $a^2 + b^2 = c^2$
- Reads in input from keyboard
- Handles any exceptions
Exercise: Prelim Review

Analyze try-catch code to see what values will throw an exception
Key takeaways

1. Java arrays do not extend!
2. A 2D array is just an array of 1D arrays.
3. Thrown exceptions bubble up the call stack until they are handled by a try-catch block. In the system, the call of method main is in a try-catch statement, and its catch block prints out information about the thrown exception.

http://xkcd.com/1188/