CSCI 261
Computer Science II
New Topic: Exceptions in Java

- You should now be familiar with:
  - Advanced object-oriented design
    - Class hierarchy, class diagrams (UML)
    - Inheritance and polymorphism
    - Abstract classes and interface

- Now
  - Change in pace: Handling Java runtime errors (exceptions)
    - Why wait until now?
      - Needed to understand inheritance first
      - Exceptions also follow a class hierarchy
  - Keywords in this lecture: throw, throws, try, catch
How Do We Currently Handle Errors?

- (We don't really)

- We try to **prevent** errors.
  - Defensive programming
  - (e.g., check if input is null)

- But we don't handle one **after** it happens.
  - There's no feedback back to the caller method

```java
import DavidsLib;
public class MyClass {
    public void someMethod() {
        int a = getNum();
        int b = getNum();
        DavidsLib.saveWorld(a,b);
        this.celebrate();
    }
}
```

```java
/**
 * Top secret code I've zipped up and no one gets to see.
 */
public class DavidsLib {
    public static void saveWorld(int x, int y) {
        // being super defensive
        if (y != 0) {
            double z = x / y;
            // (lots of lines omitted that uses z)
        }
    }
}
```
Java's Exception Handling System

- Luckily, Java has a very sophisticated error-handling scheme.
  - Called *exception handling*

- Key features:
  - Makes programmers aware of runtime errors that may occur if they call your method.
    - You can even force programmers to do something about it
  - Give programmers a chance to recover after it happens

- New keywords from this lecture:
  - `throw`, `throws`, `try`, `catch`
Outline

- What Are Exceptions?
- How Exceptions Work
- Exception Handling
  - Receiving end: throws clause and try-catch block
  - Producing end: throwing exceptions
- Exception Types: Unchecked vs. Checked
- Defining Custom Exceptions
- Conclusion
Runtime Errors (AKA Exceptions) Happen

- When running a program, it sometimes result in *exceptions*
  - Maybe people didn't bother reading your comments
    - Pro tip: Read the API before diving into code!
  - Maybe it was a simple logic mistake
    - Dreaded 1-off errors, divide by zero, etc.
  - Maybe it was an external factor
    - Opening a file that doesn't exist, or you don't have permission
Common Exceptions in Java

- You've seen exceptions at some point

  - Divide by zero:

    ```java
    int x = 4;
    x /= 0;
    Exception: java.lang.ArithmeticException (/ by zero)
    ```

  - Out of bounds access:

    ```java
    int[] list = {10, 20, 30};
    System.out.println(list[3]);
    Exception: java.lang.ArrayIndexOutOfBoundsException (3)
    ```
More you've probably seen:

- Following a **null** reference:

```java
Triangle t1, t2;
double sum = t1.getArea() + t2.getArea();
Exception: NullPointerException: t1
```

- Illegal input to a method/constructor:

```java
int x = Integer.parseInt("david");
Exception: java.lang.NumberFormatException (For input string: "david")
```

- What about... making our own? (Later)

```java
Shape s = new Rectangle(0, 0, -4, 5); // a -4 x 5 rectangle located at the origin?
```
Graceful Recovery from Exceptions?

- When these exceptions are **thrown**, your program (so-far) terminates
  - Maybe that's **okay** (*whatevs...* it's just students using my program)
  - Maybe that's **unacceptable** (published software)

- Is there a more graceful approach to handling exceptions?
  - Report the error and its likely cause
  - Try to recover (not always possible):
    - Maybe we could just keep asking for a better input?
Outline

- What Are Exceptions?
- How Exceptions Work
- Exception Handling
  - Receiving end: throws clause and try-catch block
  - Producing end: throwing exceptions
- Exception Types: Unchecked vs. Checked
- Defining Custom Exceptions
- Conclusion
How Exceptions Work

Operative terms

- **Throwing** (or **raising**) an exception
  - Complain about an error

- **Catching** an exception elsewhere in code
  - Checks for existence of an exception and "handling" it somehow
  - try-catch clause (more on this later)

In Java, exceptions are **Objects** being thrown and caught

- These objects carry information about the error
- (e.g., what caused it, was there a msg?)
How Exceptions Work (Cont.)

- Mechanism for exception handling:
  - An exception gets "thrown up" the call stack

- After exception is thrown in some method,
  - Java searches each caller for willingness to handle it
    - If the method "catches" and handles it:
      - Great, exception goes away!
    - If the method is not willing to handle it
      - It gets thrown up to its caller method
      - If no one handles it, the exception reaches the JVM, which terminates the program
How Exceptions Work (Cont.)

- Take a second to read this code:

```java
public static void main(String[] args) {
    Scanner keyboard = new Scanner(System.in);
    int x = keyboard.nextInt(); // assume they enter "0"
    int y = method(x);
}

public static int method(int z) {
    return 10 / z;
}
```

- Here's the exception reported by the JVM if 0 is entered for x:

```
java.lang.ArithmeticException: / by zero
    at Testing.method(Testing.java:18)
    at Testing.main(Testing.java:14)
```

**Trace of the Call Stack**

Exception originated here in method()
Called by main()
But main() didn't "handle" it, so it reaches JVM and crashes