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
Runtime Errors 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
Exceptions

- 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)
    ```
Exceptions (Cont.)

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.langNumberFormatException (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?
```
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

- When an exception is raised (or thrown), your program terminates!
  - Maybe that's okay (whatevs... it's just students using my program)
  - Maybe that's unacceptable (published software)

- More graceful approaches 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?
How Exceptions Work (Cont.)

- 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
Exception Handling Like Game of "Hot Potato"
How Exceptions Work (Cont.)

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

- **After exception is thrown,**
  - Java searches each call 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

```
Java Virtual Machine (JVM)
main()
A()
B()
C()
!!!
```

Call Stack

Terminate program!
How Exceptions Work (Cont.)

- Take a second to read this code:

```java
11    public static void main(String[] args) {
12        Scanner keyboard = new Scanner(System.in);
13        int x = keyboard.nextInt(); //assume they enter "0"
14        int y = foo(x);
15    }
16
17    public static int foo(int z) {
18        return 10 / z;
19    }
```

- Here's the exception reported by the JVM:

Call Stack

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

Exception originated here in foo()
Called by main()
But main() didn't "handle" it, so it reaches JVM
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
Exception Handling (Option 1)

- A method has two options when it receives an exception:
  - **Option 1**: Play hot potato. (Throw it up to caller)
    - Syntax: `<method declaration> throws exceptionType0, exceptionType1, ...

- Example:

```java
/**
 * @return A foo number
 * @throws ArithmeticException if z is zero
 */
public static int foo(int z) throws ArithmeticException {
    return 10 / z;
}
```

- Note the following:
  - A new @throws tag for Javadocs commenting
Option 2: Try to survive the exception

- Syntax of *try-catch block* directly within the method body
  – Need to have a plan to recover the program from crashing
- Could have multiple catch statements per try-clause
  – Multiple exceptions could be thrown in a block of code

Syntax:

```java
try {
    //code that could result in exception
}
catch(ExceptionType e0) {
    //what to do if one type of exception was thrown in try-block?
}
catch(ExceptionType e1) {
    //what to do if a different type exception was thrown?
}
//maybe more
```
Example

- Handling the exception in `foo(..)`

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

public static int foo(int z) {
    try {
        // This is the "dangerous" statement
        return 10 / z;
    }
    catch (ArithmeticException ex) {
        System.out.println(ex.getMessage()); // what was the error?
        return -1;
    }
}
```
Could also handle the exception in `main(..)`

```java
public static void main(String[] args) {
    Scanner keyboard = new Scanner(System.in);
    int x = keyboard.nextInt(); // assume they enter "0"
    int y;
    try {
        // This is the "dangerous" statement
        y = foo(x);
    } catch (ArithmeticException ex) {
        System.out.println(ex.getMessage()); // what was the error?
        y = -1;
    }
}

public static int foo(int z) throws ArithmeticException {
    return 10 / z;
}
```
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
Throwing Exceptions

- So far, we've only seen exceptions being generated automatically
  - What if we wanted to throw our own exceptions?

- Syntax (in the body of the method):
  - Note: Not to be confused with the `throws` keyword we just saw previously
  - Note: the `new` keyword indicates that Exceptions are objects in Java

```java
public void someMethod() {
    ...

    if (somethingWentWrong) {
        throw new ExceptionType("optional diagnostic message");
    }
}
```
Example: Throwing Our Own Exceptions

Example: What *could* go wrong with the following code?

```java
/**
 * Doubles the size of a rectangle
 * @param r A reference to a Rectangle
 */
public void enlarge(Rectangle r) {
    r.setWidth(r.getWidth() * 2);
    r.setLength(r.getLength() * 2);
}
```
Example (Cont.)

- What could go wrong? Could potentially follow a null reference!
  - Let's inform callers of this potential problem:

```java
/**
 * Doubles the size of a rectangle
 * @param r A reference to a Rectangle
 * @throws IllegalArgumentException when null is given
 */
public void enlarge(Rectangle r) {
    if (r == null) {
        throw new IllegalArgumentException("null given in enlarge()");
    }
    r.setWidth(r.getWidth() * 2);
    r.setLength(r.getLength() * 2);
}
```

**Important:** Wouldn't these still run after the Exception was thrown?

No, throw has an added effect of method termination (like return)
Exceptional Execution

```java
public void foo() {
    Rectangle r;
    try {
        enlarge(r);
    } catch (IllegalArgumentException ex) {
        System.out.println(ex.getMessage());
    }

    //more code blah blah
}

public void enlarge(Rectangle r) {
    if (r == null) {
        throw new IllegalArgumentException("null reference given in enlarge()");
    }
    r.setWidth(r.getWidth() * 2);
    r.setLength(r.getLength() * 2);
}
```

Diagram:
- Call foo()
  - In foo()
    - rectangle r;
    - enlarge(r);
  - catch statement(s)
- In enlarge()
  - r == null?
  - false:
    - r.setWidth(..)
    - r.setLength(..)
  - true:
    - throw new IllegalArgumentException.. (exit immediately)
  - S.o.pln(ex.getMessage())
  - more code (blah blah)
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
Exceptions have an inheritance hierarchy in Java

- We can create new exceptions by extending the `Exception` class
- Error vs. Exceptions
  - Error: unrecoverable errors. Don't try to catch; let program fail
    - e.g., out of memory, stack overflow
  - Exception: errors the programmer has control over
### Throwable API

- Full API here:
  - [https://docs.oracle.com/javase/8/docs/api/java/lang/Throwable.html](https://docs.oracle.com/javase/8/docs/api/java/lang/Throwable.html)

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Throwable()</td>
<td>Constructs a new throwable with null as its detail message.</td>
</tr>
<tr>
<td>public Throwable(String message)</td>
<td>Constructs a new throwable with the specified detail message.</td>
</tr>
<tr>
<td>public String getMessage()</td>
<td>Returns the detail message string of this throwable.</td>
</tr>
<tr>
<td>public void printStackTrace()</td>
<td>Prints this throwable and its backtrace to the standard error stream.</td>
</tr>
<tr>
<td>public String toString()</td>
<td>Returns a short description of this throwable.</td>
</tr>
</tbody>
</table>
## Exception API

- Full API here:
  - [https://docs.oracle.com/javase/8/docs/api/java/lang/Exception.html](https://docs.oracle.com/javase/8/docs/api/java/lang/Exception.html)

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Exception()</td>
<td>Constructs a new exception with null as its detail message.</td>
</tr>
<tr>
<td>public Exception(String message)</td>
<td>Constructs a new exception with the specified detail message.</td>
</tr>
<tr>
<td><em>(Other inherited Throwable methods)</em></td>
<td></td>
</tr>
</tbody>
</table>
Java Has Two Exception Types

- **Unchecked exception**: Cases where programmer screwed up
  - e.g., Div by 0, null exception, illegal array access
  - Java says, "Don't have to explicitly warn users about these."
    - That is, you don't need to write a try-catch clause!
    - Compiler won't "check" for that!

- **Checked exception**: Cases where program-user screwed up
  - e.g., Bad input from keyboard, write file to a full disk
  - Java says, "You must catch and handle."
    - You must either throw it up, or write a try-catch clause for these!
    - Compiler will "check" for that to keep you honest!
Exception Hierarchy

- Checked exceptions should extend the `Exception` class
- Unchecked exceptions should extend the `RuntimeException` class

![Exception Hierarchy Diagram]

- MyCheckedException
- MyUncheckedException
- Other checked Exceptions
- Other unchecked Exceptions
Example 1

- Check out `IllegalArgumentException` documentation
  - Is it checked or unchecked?

- The compiler lets this go:

```java
public void foo() {
    Rectangle r;
    enlarge(r);
    //more code blah blah
}

gpublic void enlarge(Rectangle r) {
    if (r == null) {
        throw new IllegalArgumentException("null given in enlarge()"y);
    }
    r.setWidth(r.getWidth() * 2);
    r.setLength(r.getLength() * 2);
}
```

Well, `enlarge()` might throw an unchecked exception, but whatevs... "_(ツ)_/"  
Unchecked means unrecoverable, so program's gonna die soon anyway
Exception Handling

Example 1 (Cont.)

- Check out `IllegalArgumentException` documentation
  - Is it checked or unchecked?

- The compiler lets this go:

```java
public void foo() {
    Rectangle r;
    enlarge(r);
    //more code blah blah
}

public void enlarge(Rectangle r) {
    if (r == null) {
        throw new IllegalArgumentException("null given in enlarge()");
    }
    r.setWidth(r.getWidth() * 2);
    r.setLength(r.getLength() * 2);
}
```

Method does not need to indicate that it throws unchecked exceptions!
Example 1 (Cont.)

- Check out `IllegalArgumentException` documentation
  - Is it checked or unchecked?

- The compiler lets this go:

```java
public void foo() {
    Rectangle r;
    enlarge(r);
    //more code blah blah
}

public void enlarge(Rectangle r) throws IllegalArgumentException {
    if (r == null) {
        throw new IllegalArgumentException("null given in enlarge()");  
    }
    r.setWidth(r.getWidth() * 2);
    r.setLength(r.getLength() * 2);
}
```

But there's nothing wrong with indicating it anyway! (Preferred, actually)
Example 2

- Check out `FileNotFoundException` documentation
  - Is it checked or unchecked?

- The compiler won't let this one go:

```java
public void foo() {
    Scanner keyboard = new Scanner(System.in);
    String file = keyboard.nextLine(); // get a filename from the user
    open_file(file); // ...
}

public void open_file(String filename) throws FileNotFoundException {
    if (filename == null || !exists(filename)) {
        throw new FileNotFoundException("null given in enlarge()");
    }

    // code to open file
}
```

Well, `open_file()` might throw a checked exception. This could be recoverable...
Example 2 (Cont.)

- Check out `FileNotFoundException` documentation
  - Is it checked or unchecked?

- The compiler won't let this one go:

```java
public void foo() {
    Scanner keyboard = new Scanner(System.in);
    String file = keyboard.nextLine();  // get a filename from the user
    open_file(file);
    // ...
}

public void open_file(String filename) throws FileNotFoundException {
    if (filename == null || !exists(filename)) {
        throw new FileNotFoundException("null given in enlarge()");
    }

    // code to open file
}
```

Compiler insists that you either throw it up to caller or try-catch it here!
Example 2 (Cont.)

- Check out `FileNotFoundException` documentation
  - Is it checked or unchecked?

- The compiler won't let this one go:

```java
public void foo() {
    Scanner keyboard = new Scanner(System.in);
    String file = keyboard.nextLine();  //get a filename from the user
    open_file(file);
    //...
}

public void open_file(String filename) throws FileNotFoundException {
    if (filename == null || !exists(filename)) {
        throw new FileNotFoundException("null given to enlarge()");
    }

    //code to open file
}
```

Compiler wants programmer to know it **can** be handled without resulting in termination.
Take-Home Exercise

- Peruse Java's documentation on the following exceptions, and determine:
  - When they occur
  - Whether they're checked or unchecked

<table>
<thead>
<tr>
<th>Name</th>
<th>When</th>
<th>Checked</th>
<th>Unchecked</th>
</tr>
</thead>
<tbody>
<tr>
<td>IllegalArgumentException</td>
<td>Invalid argument received by method.</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>ArithmeticException</td>
<td>Illegal arithmetic operation detected.</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>FileNotFoundException</td>
<td>Attempting to access a non-existent file.</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>IOException</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ArrayIndexOutOfBoundsException</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EOFException</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NullPointerException</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NumberFormatException</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Take-Home Exercise (Soln)

<table>
<thead>
<tr>
<th>Name</th>
<th>When</th>
<th>Checked</th>
<th>Unchecked</th>
</tr>
</thead>
<tbody>
<tr>
<td>IllegalArgumentException</td>
<td>Invalid argument received by method.</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>ArithmeticException</td>
<td>Illegal arithmetic operation detected.</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>FileNotFoundException</td>
<td>Attempting to access a non-existent file.</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>IOException</td>
<td>An I/O operation has failed. (e.g., writing to a file, reading from scanner, ..)</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ArrayIndexOutOfBoundsException</td>
<td>Self-explanatory</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>EOFException</td>
<td>An &quot;end-of-file&quot; marker has been reached expectedly when reading file or stream.</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>NullPointerException</td>
<td>Self-explanatory</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>NumberFormatException</td>
<td>Attempted to convert a string to one of the numeric types, but that the string does not have the appropriate format</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
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
Defining Our Own Exceptions

- Note how the names of these exceptions are quite descriptive

- Sometimes, not descriptive enough
  - We can optionally create our own `Exception` classes
    - We get to decide whether our exception needs to be "checked"

- Let's say we want to throw an `IllegalShapeException` if a user tries to set its dimensions to invalid values
Defining **IllegalShapeException**

```java
/**
 * An IllegalShapeException is a checked exception that should be thrown whenever
 * a user attempts to change the dimensions of a shape to illegal value(s).
 *
 * @author David
 * @version 6/8/2017
 */
public class IllegalShapeException extends Exception {
    /**
     * Creates an IllegalShapeException object
     */
    public IllegalShapeException() {
        super();
    }

    /**
     * Creates an IllegalShapeException object with a diagnostic note
     * @param message a diagnostic message
     */
    public IllegalShapeException(String message) {
        super(message);
    }
}
```
public class Circle extends Shape {
    protected double radius;

    public Circle(int x, int y, double r) {
        super(x, y);  //call Shape(x,y)
        this.radius = r;
    }

    /**
     * Assigns a new radius.
     * @param new_radius  length of new radius
     * @throws IllegalShapeException if new radius is not positive
     */
    public void setRadius(double new_radius) throws IllegalShapeException {
        if (new_radius <= 0) {
            throw new IllegalShapeException("the radius of a circle must be positive: " + new_radius);
        }
        radius = new_radius;
    }

    //other methods...
}
public class Tester {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        Circle c = new Circle(0, 0, 5); // a circle at the origin with radius=5
        boolean success = false;

        while (!success) {
            System.out.println("Input a new radius: ");
            double new_radius = keyboard.nextDouble();
            try {
                c.setRadius(new_radius);
                success = true; // made it here, that means setRadius didn't cause errors
            } catch (IllegalShapeException e) {
                System.out.println(e.getMessage()); // print out a message warning user
            }
        }
        System.out.println("radius: " + c.getRadius());
    }
}

**Interaction (with exception)**

Input a new radius: 0  
the radius of a circle must be positive: 0.0
Input a new radius: -3  
the radius of a circle must be positive: -3.0
Input a new radius: 9  
radius: 9.0

**Interaction (normal)**

Input a new radius: 20  
radius: 20.0
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
Summary

- Exceptions are objects representing errors in execution
  - Can be checked or unchecked
  - You can create your own

- Mechanism is to throw exceptions when error is detected
  - Thrown to the caller method, which has a choice of what to do with it!

- "I was doing all these defensive programming stuff already!"
  - Yes, but Exceptions can get the compiler involved too
  - If you don't explicitly handle a checked exception, it won't compile