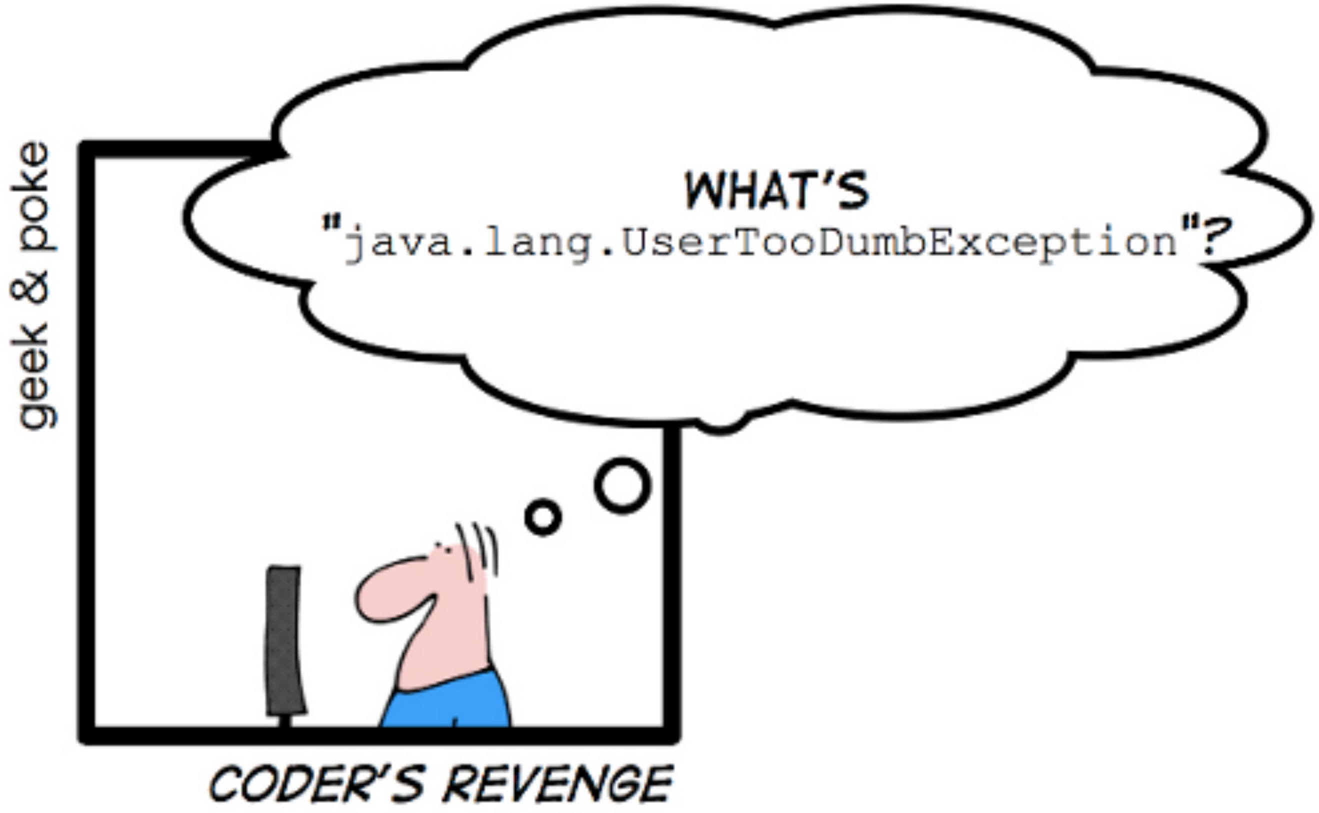
CSE 143 Java

Exceptions



Verifying Validity of Input Parameters

- A non-private method should always perform parameter validation as its caller is out of scope of its implementation

http://docs.oracle.com/javase/7/docs/technotes/guides/language/assert.html

```
/** @param rate refresh rate, in frames per second.
  * @throws IllegalArgumentException if rate <= 0 or
  * rate > MAX_REFRESH_RATE. */
   public void setRefreshRate(int rate) {
       // Enforce specified precondition in public method
       if (rate <= 0 || rate > MAX_REFRESH_RATE) throw new
          IllegalArgumentException("Illegal rate: " + rate);
       setRefreshInterval(1000/rate);
 }
```

Preconditions on *public* methods are enforced by explicit checks that **throw particular, specified exceptions**

Exception Handling

Exceptions: represent unusual events (as well as errors)

- Finite table is full; cannot add new element
- Attempt to open a file failed

Problems:

- the method that detects the error does not know how to handle it (and probably should not)
- the client code could handle the error, but is not in a position to detect it

- **Solution**: method detecting an error **throws** an exception; client code **catches** and handles it

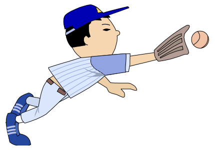
Exceptions as Part of Method Specifications

What should a client code method do with exception?

- Either must handle it

```
void readSomeStuff( ) {
   try {
      readIt( );  // potentially throws an Exception
   }
   catch (Exception e) {
      handle
   }
```

- Or declare that it can potentially throw it



```
void readSomeStuff( ) throws Exception {
   readIt( );
}
```

try-catch

```
try {
    somethingThatMightBlowUp( );
} catch (Exception e) {
    recovery code – e, the exception object, is a “parameter”
}
```

- Execute try block
- If an exception is thrown, catch block can either process the exception, re-throw it, or throw another exception
- Thrown exceptions terminate throwing method and all methods that called it, until reaching a method that catches the exception (has a catch block whose type matches the exception)
- If there is no try/catch → terminate the thread (possibly the program)

try-catch

- Can have **several** catch blocks

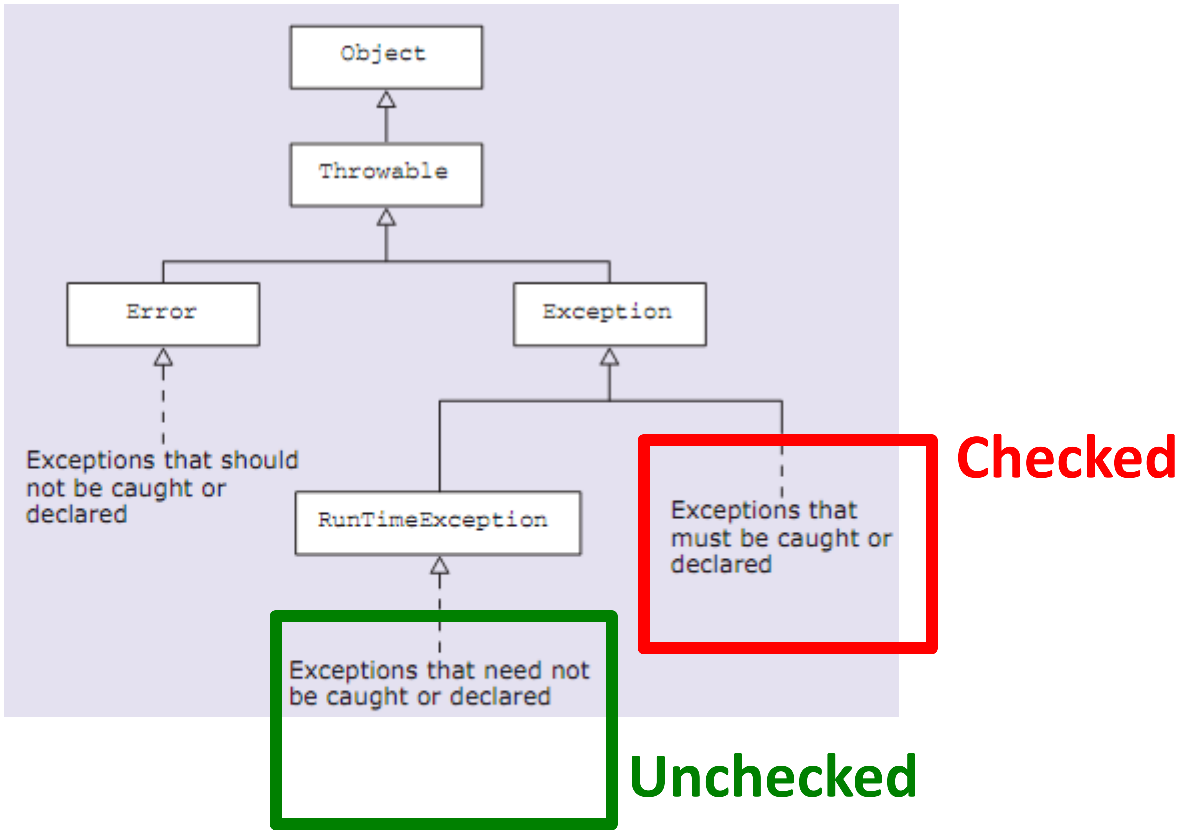
```
try {attemptToReadFile( );}
catch (FileNotFoundException e) {…}
catch (IOException e) {…}
catch (Exception e) { …}
```

- Semantics: try to match exception parameters in order until one matches
- **Need to go** from more specific to more general (why?)
- If no match – exception propagates (gets thrown) to calling method
- In Java SE 7 and later, a single catch block can handle more than one type of exception :

```
catch (FileNotFoundException | IOException | Exception e) {…}
```

- http://www.oracle.com/technetwork/articles/java/java7exceptions-486908.html

Throwable/Exception Hierarchy


Object
Throwable
Error
Exception
Exceptions that should not be caught or declared
RunTimeException
Exceptions that must be caught or declared
Checked
Exceptions that need not be caught or declared
Unchecked

Checked vs Unchecked Exceptions

Checked: are exceptions that are checked at the compile time

- Represent invalid conditions in areas outside the immediate control of the program (invalid user input, database problems, network outages, absent files)
- Are subclasses of Exception
- Method must establish a policy for all checked exceptions thrown by its implementation
  - ➢ either handle them somehow
    - → catch all checked exceptions it might encounter (`try-catch`)
  - ➢ or pass the checked exceptions further up the stack
    - → declare that it might throw them (using `throws` keyword)

Checked vs Unchecked Exceptions

Unchecked: are not checked at the compile time.

- Represent defects in the program (bugs)
- Reflect errors in program's logic from which it is not possible to recover at a run time
- Often invalid arguments passed to a non-private method.

- Are subclasses of RuntimeException, and are usually implemented using IllegalArgumentException, NullPointerException, or IllegalStateException

- Method is NOT obliged to establish a policy for the unchecked exceptions thrown by its implementation
 (almost always does not do so)

Checked vs Unchecked Exceptions

- No need to declare anything about unchecked exceptions
- Include an @throws in the JavaDocs for ones specifically thrown
- RuntimeException (unchecked) is itself a subclass of Exception (checked).
- Why to have both types?

http://docs.oracle.com/javase/tutorial/essential/exceptions/runtime.html

- http://www.javamadesoeasy.com/2015/05/exceptions-top-60-interview-questions_16.html

Writing your own exception

```
/**
• Represents an exception thrown when an invalid value is
 given for radius
 */
 public class InvalidRadiusException extends RuntimeException {
        /**
        * {@inheritDoc}
         */
        public InvalidRadiusException(String message) {
               super(message);
        }
 }
```

Is this checked or unchecked exception?

What can we do with **`InvalidRadiusException`**?

```
public class Circle extends AbstractShape {
 /**
 * Given a pin and a radius greater than 0, creates a circle
 * @param pin the location of this circle's pin
 * @param radius this circle's radius. The radius must be greater than 0
  * @throws InvalidRadiusException if the radius is negative or zero
      public Circle(Posn pin, Integer radius) {
          super(pin);
          if (radius <= 0) {
             throw new InvalidRadiusException("Radius must be
                                     > 0, given: " + radius);
          }
          this.radius = radius;
          } // elided code
}
```

Do we need to handle it?

- Since **`InvalidRadiusException`** is unchecked (why?), we may or may not handle it
- Example of how to handle:

Somewhere inside VERY important client code:

```
try {
      Circle myCircle = new Circle(new Pin(0,0), -2);
 }
catch (InvalidRadiusException invalidRadius) {
      ShowErrorMEssage errorMessage = new ShowErrorMessage
                    ( "We detected an incorrect value " +
                    "for myCircle. “ + "Please provide a
                    positive number.");
      new Window(errorMessage).exit();
}
```

Always be VERY descriptive in your error message