Exceptions

- Exceptions are a mechanism for dealing with inappropriate behavior or errors such as attempting to access a null reference, indexing an array out of bounds, or trying to read past the end of a file.
- Java code can explicitly raise an exception by using the ***throw*** expression.
- Exceptions can be handled in ***try/catch/finally*** blocks.

CAL POLY

Exceptions

- The JVM can throw exceptions which can be caught in try/catch blocks.

```
int x = Integer.parseInt(JOptionPane.showInputDialog(null,"Enter an int"));
int y = Integer.parseInt(JOptionPane.showInputDialog(null,"Enter another"));
int [] z = new int[5];
try {
    System.out.println("y/x gives " + (y/x));
    System.out.println("y is " + y + " z[y] is " + z[y]);
}
catch (ArithmeticException e) {
    System.out.println("Arithmetic problem " + e);
}
catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Subscript problem " + e);
}
```

CAL POLY

Exceptions

- Exceptions can be explicitly thrown and caught in try/catch blocks.

```
public class ThrowTest {
    public static void main(String[] args) {     //pardon the poor indentation
        String s = "";
        try {
            s = "http://www.whatzup";            doSomeIO(s);
        }
        catch (MalformedURLException e) {
            System.out.println("URL problem " + s + " " + e);
        }
        try {
            s = "http://www.whatzup.com";            doSomeIO(s);
            s = "http://www.whatzup.org";            doSomeIO(s);
        }
        catch (MalformedURLException e) {
            System.out.println("URL problem " + s + " " + e);        }    }
    public static void doSomeIO(String url) throws MalformedURLException {
        URL tempURL = new URL(url);        //could throw Malformed URLException
        if (-1 == url.indexOf(".com"))          //restrict URLs to only .com's
        {            throw new MalformedURLException();        }   } }
```

Exceptions

- All exceptions are objects in Java.
- All exceptions are subclasses of java.lang.Throwable.
- There are two categories of exceptions.
  - Checked exceptions (java.lang.Exception)
  - Unchecked exceptions
    - Runtime exceptions (java.lang.RuntimeException)
    - Errors (java.lang.Error)
- Many subclasses of the above three are already defined, but you can also create your own classes of exceptions by subclassing one of the above classes.

Runtime Exceptions

- Runtime exceptions are generally problems that could be prevented by the programmer such as:
  - Bad casts
  - Out-of-bounds array access
  - Null pointer access
- Because runtime exceptions should not occur in correct programs, your code is not required to catch them so they are also called unchecked exceptions.

Checked Exceptions

- Other exceptions can be harder to prevent because they rely on user input or external events.
- Some examples of checked exceptions are:
  - Trying to read past the end of a file
  - Trying to open a malformed URL
  - Trying to find a Class object for a string that does not correspond to an existing class.
- Code that may throw a checked exception must provide a try/catch block to handle the exception or the compiler will complain.

Checked Exceptions Example

- Methods which throw checked exceptions must explicitly state what exceptions they throw and be called within a try block.

```
public static void main(String[] args) {
    try {
       doSomeIO("http://www.whatzup");
    }
    catch (MalformedURLException e) {
       System.out.println("URL problem " + e);
    }
 }
 public static void doSomeIO(String url) throws MalformedURLException {
          …
          throw new MalformedURLException();   //create instance in throw
    }
 }
```

Throwing/Catching Multiple Exceptions

```
public static void main(String[] args) {
  try {
     doSomeIO("http://www.whatzup");
  }
  catch (MalformedURLException e) {
     System.out.println("URL problem " + e);
  }
  catch (SomeOtherException e) {
     System.out.println(“Some Other problem “ + e);
  }
}
public static void doSomeIO(String url)
       throws MalformedURLException, SomeOtherException {
       if (…)
           throw new MalformedURLException();   //create instance in throw
       else
           throw new SomeOtherException();   //create instance in throw
  }
}
```

CAL POLY

Finally

- Sometimes you want some code executed at the end of a method regardless of whether an exception was thrown or not.
- The statements in a finally block get executed after the try block if no exceptions are thrown, or after the catch block if an exception is thrown and caught.

```
try {
    doSomeIO("http://www.whatzup");
}
catch (MalformedURLException e) {
    System.out.println("URL problem " + e);
}
finally {
    System.out.println(“The try is done”);
}
```

Re-throwing exceptions

- Sometimes a catch handler may only do part of the job of handling an exception.
- The handler can then re-throw the exception so that a caller of the method can continue to handle the exception.

```
try {
   doSomeIO("http://www.whatzup");
}
catch (MalformedURLException e) {
   System.out.println("URL problem " + e);
   throw e;
}
```

Extending Exception Classes

- Exception classes can be subclasses of other exception classes.
- Catch handlers will catch all exceptions of the specified class or any subclass.
- Separate catch handlers can be defined to catch super and sub-classes.
- Subclass handlers must come before super-class handlers.

Extending Exception Classes

```
class BadUserInputException extends Exception {
 … }
class ReallyBadUserInputException extends BadUserInputException {
 … }

public static void main(String[] args) {
      try {
          getInput();
      }
      catch (ReallyBadUserInputException e) {     //don’t switch the order
          System.out.println(“You really messed up " + e);
      }
      catch (BadUserInputException e) {
          System.out.println(“You messed up “ + e);
      }
  }
```

CAL POLY

Exceptions and Inheritance

- Subclass methods that override a superclass method cannot throw exceptions not defined in the superclass method.
- Subclass methods are not required to throw all exceptions of their corresponding superclass methods.

Exceptions and Inheritance

```
class BaseClass {
  public void doSomething() throws BadUserInputException { … }
  public void doAnotherThing()
                         throws MalformedURLException, EOFException { … }
}
class SubClass1 extends BaseClass {
  public void doSomething() { … }              //okay to not throw anything
}
class SubClass2 extends BaseClass {
  public void doAnotherThing() throws EOFException { … }//okay to throw just one
}
class SubClass3 extends BaseClass {
  public void doSomething() throws ReallyBadUserInputException { … }
            //okay to throw a subclass of the original method's exception
}
```