Handling Errors

• What happens if someone attempts to add an item to a queue that is full?
• Many times it is the client that knows what the appropriate thing to do is in the case of an error.
• Exceptions provide a method of informing a client that an error has occurred and provides a mechanism for handling the error cleanly.
• Many languages provide support for exceptions.

Java Exception Handling

• An exception occurs, thrown, when something goes wrong while a program is running.
• If an exception is not handled, caught, the program will terminate.
• When an exception is thrown, the exception is passed to the caller of the method.
• The caller can then handle the exception or allow it to propagate back up the call chain.

Java Exceptions

• An exception object stores information about the cause of an exception and can be passed to an exception handler
• Exceptions must be a subclass of Throwable
  – Throwable and its subclasses have two constructors, one that takes no arguments, and one that takes a string argument that can be used to produce an error message
  – A Throwable class contains a snapshot of the state of the program at the time it was created
  – It can also contain a message string that gives more information about the error
Java Exceptions

- ActivationException
- ApplicationException
- AWTException
- ClassNotFoundException
- DataFormatException
- GeneralSecurityException
- IllegalAccessException
- InstantiationException
- InterruptedException
- IOException
- NoSuchMethodException
- ParseException
- PrinterException
- RuntimeException
- ArithmeticException
- ArrayIndexOutOfBoundsException
- ArrayStoreException
- ClassCastException
- ClassNotFoundException
- IllegalArgumentException
- IndexOutOfBoundsException
- NegativeArraySizeException
- NullPointerException
- NumberFormatException
- RuntimeException
- SecurityException

What Exceptions are Thrown?

- How do you figure out what exception will be thrown that you must handle?
  - read the documentation for the class you are using
  - read the documentation about the various exception classes
- I often simply use the methods I want and let the compiler tell me when I missed something!!
Try, Catch, and Finally

- A **try** block is wrapped around code that may cause an exception.
- catch blocks following the try block can contain code to deal with the exception.
- A finally block contains code that will be executed whether or not an exception occurs.
- In most cases try and catch blocks suffice

---

### Syntax

```
try {
    statementSequence
} catch (parameter) {
    statementSequence
}
finally {
    statementSequence
}
```

---

### parseInt

```java
public static int parseInt(String s, int radix)
    throws NumberFormatException
```

Parses the string argument as a signed integer in the radix specified by the second argument. The characters in the string must all be digits of the specified radix (as determined by whether Character.digit(char, int) returns a non-negative value), except that the first character may be an ASCII minus sign '-' (\u002d) to indicate a negative value. The resulting integer value is returned.

An exception of type NumberFormatException is thrown if any of the following situations occur:
- The first argument is null or is a string of length zero.
- The radix is either smaller than Character.MIN_RADIX or larger than Character.MAX_RADIX.
- Any character of the string is not a digit of the specified radix, except that the first character may be a minus sign '-' (\u002d) provided that the string is longer than length 1.
- The integer value represented by the string is not a value of type int.
public class CmdLine {
    public static void main( String args[] ) {
        for ( int i=0; i<args.length; i++ ) {
            int val;
            try {
                val = Integer.parseInt( args[i] );
                System.out.println( val );
            } catch ( NumberFormatException e ) {
                System.out.println( "??" );
            }
        }
    }
}