06-0: **Errors**

- Errors can occur in program
  - Invalid input / bad data
  - Unexpected situation
  - Logic error in code
- Like to handle these errors gracefully, not just halt the program
  - Running a web server, don't want one piece of bad data to bring the whole thing down

06-1: **Error Checking**

- We could check for any concievable error

```
ArrayList<Integer> A;
// Initialize A
A.set(i, new Integer(x/y));
```

06-2: **Error Checking**

- We could check for any concievable error

```
if (i >= 0 && i < A.size())
{
   if (y != 0)
   {
      A.set(i, new Integer(x / y);
   }
   else
   {
      // Handle division by zero case
   }
   // Handle outside bounds of the array case
}
```

- Problems with this method?

06-3: **Exceptions**

- We can let the system catch all the errors for us

```
A.set(i, new Integer(x / y));
```

- Throws an exception if i ¡ 0, i ¿= A.size(), y == 0. Program ends.
- Problems with this method?

06-4: **Exceptions**

- We can let the system catch the errors for us
- We can "catch" the errors ourselves

```
try
{
   A.set(i, new Integer(x,y));
}
catch (Exception e)
{
   // do some work to clean up after the exception
}
```

06-5: **Try-Catch Block**

```
try
{
   // Any Java Code
}
catch (Exception e)
{
   // Any Java Code
}
```

- If an exception is rasied inside the try block:
  - Stop immediately and execute the code in the catch block
  - Coninue after the catch block as normal
- If no exception is raised inside the try block
  - Ignore the code in the catch block

06-6: **Exceptions**

```
int x;
int y;
try
{
   x = 3;
   y = 0;
   x = x / y;
   System.out.println("Can't get here!");
}
catch (Exception e)
{
   System.out.println("Exception caught!");
}
System.out.println("Done with try block!");
```

06-7: **Exceptions**

```
int x;
int y;
try
{
   x = 3;
   y = 5;
   x = x / y;
   System.out.println("We will get here!");
}
catch (Exception e)
{
   System.out.println("We won't get here!");
}
System.out.println("Done with try block!");
```

06-8: **Exceptions**

```
ArrayList<Integer> A = new ArrayList<Integer>();
for (int i = 0; i < 10; i++)
{
   A.add(new Integer(i));
}
try
{
   for (int i = 0; i < 10; i++)
      System.out.println(i);
   for (int i = 10; i > 0; i--)
      System.out.println(i);
}
catch (Exception e)
{
   System.out.println("Error!");
}
System.out.println("Done with try block!");
```

06-9: **Exceptions**

```
ArrayList<Integer> A = new ArrayList<Integer>();
for (int i = 0; i < 10; i++)
{
   A.add(new Integer(i));
}
try
{
   for (int i = 0; i < 10; i++)
      System.out.println(A.get(i));
   for (int i = 10; i > 0; i--)
      System.out.println(A.get(i));
}
catch (Exception e)
{
   System.out.println("Error!");
}
System.out.println("Done with try block!");
```

06-10: **Exceptions**

- Variables declared within a try block are not visible outside
- Actually, variables declated within *any* block are not visible outside the block

```
try
{
   ArrayList<Integer> A = new ArrayList<Integer>();
   for (int i = 0; i < 10; i++)
      A.add(new Integer(i));
}
catch (Exception e)
{
   System.out.println("Error!");
}
A.set(3, new Integer(5)); // ERROR!
```

06-11: **Exceptions**

- What went wrong?

```
try
{
      A.set(new Integer(x/y));
}
catch (Exception e)
{
   System.out.println(e.getMessage());
}
```

- We'll do more with this after Inheritance

06-12: **Uncaught Exceptions**

```
int divide(int x, int y)
{
    int result = x / y;
    System.out.println(result);
    return result;
}

void foo()
{
   int x = 6;
   x = divide(x, 2);
   x = divide(x, 1);
   x = divide(x, 0);
   x = divide(x, 3);
}
```

06-13: **Uncaught Exceptions**

```
int divide(int x, int y)
{
    int result = x / y;
    System.out.println(result);
    return result;
}
void foo()
{
   try
   {
      int x = 6;
      x = divide(x, 3);
      x = divide(x, 2);
      x = divide(x, 0);
      x = divide(x, 2);
   }
   catch (ArithmeticException e)
   {
      System.out.println("Error!");
   }
}
```

06-14: **Uncaught Exceptions**

```
int divide(int x, int y)
{
    int result = x / y;
    System.out.println(result);
    return result;
}
void divideBySelf(ArrayList<Integer> A)
{
   for (int i = 0; i < A.size(); i++
```

```
void foo()
{
   ArrayList<int> A = new ArrayList<int>();
   A.add(new Integer(4));
   A.add(new Integer(0));
   A.add(new Integer(3));
   try
   {
      divideBySelf(A);
```

```
   {                                          }
      int res = divide(A.get(i), A.get(i));
      A.set(res);                             catch (ArithmeticException e)
   }                                          {
}                                                System.out.println("Excp. caught!");
                                              }
                                           }
```

06-15: **Uncaught Exceptions**

```
class Silly {
   public int x;
   int badFunc()                       void bar()
   {                                   {
       x++;                               x++;
       int y = x / 0;                     foo();
       x++;                               x++;
   }                                   }
   void foo()                          void start()
   {                                   {
      x++;                                try
      badFunc();                          {
      x++;                                   x = 0;
   }                                         bar();
                                          }
                                          catch (Exception e) { }
                                          System.out.println(x);
                                       }
```

06-16: **Different Kinds of Exceptions**

- There are many different kinds of exceptions
  - Array Out of bounds
  - Arithmetic (divide by zero)
  - I/O (file doesn't exist, etc)
- We'll come back to different kinds of exceptions after we've covered inheritance

06-17: **Errors Ahead**

- You can warn other programmers using you code that your code might throw an exception
- They will need to either deal with the exception themselves, or throw it on to who called them
- We've already seen this (class File), examples in class
- (Examples)

06-18: **ArrayList Fun**

- Write a function that reverse an ArrayList of strings

```
void reverse(ArrayList<string> list)
{

}
```

06-19: **ArrayList Fun**

```
void reverse(ArrayList<string> list)
{
   String tmp;
   for (int i = 0; i < list.size() / 2; i++)
   {
      tmp = list.get(i);
      list.set(i, list.get(list.size() - 1 - i));
      list.set(list.size() - 1 - i, tmp);
   }
}
```

06-20: **In-Class Assignment**

- Go to lecture note website
- Get ListFun.java
- Fill in body of reverse2, so that it returns a reversed copy of the list passed in (without changing the list passed in)