22-0: 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

22-1: Error Checking

- We could check for any conceivable error

\[ A[i] = x / y; \]

22-2: Error Checking

- We could check for any conceivable error

```java
if (i >= 0 && i < A.length)
    if (y != 0)
        A[i] = x / y
    else
        // Handle division by zero case
    // Handle outside bounds of the array case
```

- Problems with this method?

22-3: Exceptions

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

\[ A[i] = x / y \]

- Throws an exception if \( i \leq 0, i \geq A.length, y == 0 \). Program ends.
- Problems with this method?

22-4: Exceptions

- We can let the system catch the errors for us
- We can “catch” the errors ourselves

```java
try
    { A[i] = x / y
    }
catch (Exception e)
    { // do some work to clean up after the exception
```

22-5: Try-Catch Block
try {
    // Any Java Code
}
catch (Exception e) {
    // Any Java Code
}

- If an exception is raised inside the try block:
  - Stop immediately and execute the code in the catch block
  - Continue after the catch block as normal

- If no exception is raised inside the try block
  - Ignore the code in the catch block

22-6: Exceptions

```java
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!");
```

22-7: Exceptions

```java
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!");
```

22-8: Exceptions

```java
int A[] = new int[10];
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!");
```

22-9: Exceptions

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

```java
try {
    int A[] = new int[10];
    for (int i = 0; i < 10; i++)
        A[i] = i;
} catch (Exception e) {
    System.out.println("Error!");
}
```
22-10: Exceptions

- What if you want to know more about what went wrong

```
try
{
    A[i] = x / y;
} catch (Exception e)
{
    System.out.println("Error");
}
```

- There are actually two different kinds of errors that could occur – division by 0, and index out of range

```
try
{
    A[i] = x / y;
} catch (ArithmeticException e)
{
    // Handle division by zero
} catch (IndexOutOfBoundsException e)
{
    // Handle Index out of range
} catch (Exception e)
{
    // Handle anything else
}
```

22-11: Try-Catch Block

- If an exception occurs:
  - Find the first catch block that matches the expression, execute that code
  - If more than one catch block matches, use the first one
  - If no catch block matches, the exception is uncaught

22-13: Exception Hiearchies

![Exception Hiearchies Diagram]

- An exception is an object
  - Contains data members and methods
- All exceptions are subclasses of the base class Exception
  - Some data members and methods that all exceptions have
22-15: Exceptions are Objects

- Some methods defined in exceptions:
  - getMessage()
    - return a string that describes what went wrong
  - toString()
    - All exceptions are objects, so must have a toString method. Name of exception, concatenated to the message
  - getStackTrace()
    - What the stack looks like at time of exception. Not commonly used

22-16: Exceptions

```java
try {
    A[i] = x / y;
} catch (Exception e) {
    // Handles ALL exceptions
} catch (ArithmeticException e) {
    // CANNOT BE REACHED
} catch (IndexOutOfBoundsException e) {
    // CANNOT BE REACHED
}
```

22-17: Uncaught Exceptions

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

void foo()
{
    int x = 5;
    x = divide(x, 2);
    x = divide(x, 1);
    x = divide(x, 0);
    x = divide(x, 2);
}
```

22-18: Uncaught Exceptions

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

void foo()
{
    try {
        int x = 5;
        x = divide(x, 2);
        x = divide(x, 1);
        x = divide(x, 0);
        x = divide(x, 1);
    } catch (ArithmeticException e) {
        System.out.println("Error!");
    }
}
```

22-19: Uncaught Exceptions
int divide(int x, int y)
{
    int result = x / y;
    System.out.println(result);
    return result;
}
void divideBySelf(int A[])
{
    try
    {
        for (int i = 0; i < A.length; i++)
        {
            A[i] = divide(A[i], A[i]);
        }
    }
    catch (ArithmeticException e)
    {
        System.out.println("Excp. caught!");
    }
}

void foo()
{
    int A[] = new int[3];
    A[0] = 4;
    A[1] = 0;
}

void bar()
{
    int y = x / 0; foo();
    x++;
}

22-20: Uncaught Exceptions

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

22-21: Exception Messages

- Exceptions can have a “Message” string
  - Tells you a little bit more about the exception

String msg;
try
{
    a = y / 0;
} catch (ArithmeticException e) { msg = e.getMessage(); }

- msg = “/ by 0”

22-22: Exception Messages

- Exceptions can have a “Message” string
  - Tells you a little bit more about the exception

String msg;
try
{
    int A[] = new int[10];
    A[32] = 6;
} catch (ArithmeticException e) { msg = e.getMessage(); }

- msg = “32”

22-23: Checked vs. Unchecked

- Two kinds of exceptions:
  - Unchecked Exceptions
• Typically caused by programming errors (divide by zero, array index out of bounds, etc)
• Often no graceful way to recover
• Program does not need to handle them explicitly

• Checked Exceptions
• Typically caused by bad data from the user
• Program does need to handle them explicitly

22-24: Checked vs. Unchecked

• If a method could throw a checked exception, it needs to declare that in its function header
• If a method calls another method that could throw a checked exception, then it needs to either:
  • Catch the exception and deal with it
  • Declare that the checked exception may be thrown

22-25: Checked vs. Unchecked

```java
class ExceptionTest{
    static void readFile(String filename) // ERROR!
    {
        scanner scan = new Scanner(new File(filename));
        System.out.println(scan.nextString());
    }
    public static void main(String args[])
    {
        readFile(args[0]);
    }
}
```

22-26: Checked vs. Unchecked

```java
class ExceptionTest{
    static void readFile(String filename)
    {
        try
        {
            scanner scan = new Scanner(new File(filename));
            System.out.println(scan.nextString());
        }
        catch (Exception e)
        {
        }
        public static void main(String args[]) 
        {
            readFile(args[0]);
        }
    }
}
```

22-27: Checked vs. Unchecked

```java
class ExceptionTest{
    static void readFile(String filename)
    {
        try
        {
            scanner scan = new Scanner(new File(filename));
            System.out.println(scan.nextString());
        }
        catch (IOException e)
        {
        }
        public static void main(String args[]) 
        {
            readFile(args[0]);
        }
    }
}
```

22-28: Checked vs. Unchecked

```java
class ExceptionTest{
    static void readFile(String filename)
    {
        try
        {
            scanner scan = new Scanner(new File(filename));
            System.out.println(scan.nextString());
        }
        catch (IOException e)
        {
        }
        public static void main(String args[]) 
        {
            readFile(args[0]);
        }
    }
}```
class ExceptionTest{
    static void readFile(String filename) throws IOException
    {
        scanner scan = new Scanner(new File(filename));
        System.out.println(scan.nextString());
    } // TRY-CATCH BLOCK
    public static void main(String args[]) throws IOException
    {readFile(args[0]); // ERROR! (why?)
    }
}

22-30: Checked vs. Unchecked
class ExceptionTest{
    static void readFile(String filename) throws IOException
    {
        scanner scan = new Scanner(new File(filename));
        System.out.println(scan.nextString());
    } // TRY-CATCH BLOCK
    public static void main(String args[]) throws IOException
    {readFile(args[0]); // ERROR! (why?)
    }
}

22-31: Checked vs. Unchecked
class ExceptionTest{
    static void readFile(String filename) throws IOException
    {
        scanner scan = new Scanner(new File(filename));
        System.out.println(scan.nextString());
    } // TRY-CATCH BLOCK
    public static void main(String args[]) throws IOException
    {readFile(args[0]); // ERROR! (why?)
    }
}

22-32: Checked vs. Unchecked
class ExceptionTest{
    static void readFile(String filename) throws IOException
    {
        scanner scan = new Scanner(new File(filename));
        System.out.println(scan.nextString());
    } // TRY-CATCH BLOCK
    public static void main(String args[]) throws IOException
    {readFile(args[0]); // ERROR! (why?)
    }
}
22-34: Creating your own Exception

- We can create our own Exceptions:

```java
public class BadDataException extends Exception {
    public BadDataException(String message) {
        super(message);
    }
}
```

22-35: Throwing Exceptions

```java
int readStudentID(Scanner s) throws BadDataException {
    int ID = s.nextInt();
    if ((ID < SMALLEST_LEGAL_ID) || (ID > LARGEST_LEGAL_ID)) {
        BadDataException e = new BadDataException("Malformed ID Number");
        throw e;
    }
    return ID;
}
```

22-36: Throwing Exceptions

- Custom exceptions that extend “Exception” must be checked exceptions
- If your function could possibly throw a “custom” exception, needs to tell the world about it
  - Throw custom exception through a “throw” statement
  - Call another function that could throw a custom exception, don’t catch it

22-37: Throwing Exceptions

```java
int readStudentIDs(Scanner s, int A[]) throws BadDataException {
    int size = 0;
    while (s.hasNextInt()) {
        A[size] = readStudentID(s);
        size++;
    }
    return size;
}
```

22-38: Throwing Exceptions

```java
int readStudentID(Scanner s) throws BadDataException {
    int ID = s.nextInt();
    if ((ID < SMALLEST_LEGAL_ID) || (ID > LARGEST_LEGAL_ID)) {
        BadDataException e = new BadDataException("Malformed ID Number");
        throw e;
    }
    return ID;
}
```

22-39: Throwing Exceptions
```java
int readStudentIDs(Scanner s, int A[])
{
    int size = 0;
    try
    {
        while (s.hasNextInt())
        {
            A[size] = readStudentID(s);
            size++;
        }
    }
    catch (BadDataException e)
    {
        System.out.println("Bad Data, aborting ... ");
    }
    return size;
}
```

### 22-40: Throwing Exceptions

```java
int readStudentIDs(Scanner s, int A[])
{
    int size = 0;
    while (s.hasNextInt())
    {
        try
        {
            A[size] = readStudentID(s);
            size++;
        }
        catch (BadDataException e)
        {
            System.out.println("Bad Data, skipping ...");
        }
    }
    return size;
}
```

### 22-41: Custom Unchecked

- All Exceptions that extend Exception directly must be checked exceptions
- Custom Exception that subclasses RuntimeException may be unchecked
  - Examples with Eclipse
- Why are custom unchecked exceptions usually a bad idea?

### 22-42: Finally

- Finally clause can optionally appear after all of the catch blocks
- Code in the finally clause will *always* be executed at the end of the try/catch block
  - If no exception, finally is executed after normal completion
  - If exception is caught, finally is executed after appropriate catch block
  - If exception is not caught, finally is executed, then exception is thrown to calling function
- Examples from eclipse

### 22-43: Finally

```java
try
{
    int x = 3 / 0;
    System.out.println("Can't get here");
}
catch (Exception e)
{
    System.out.println("Will get here");
}
finally
{
    System.out.println("Will also get here");
    System.out.println("Will get here.");
}
```
try {
    int x = 3 / 3;
    System.out.println("Will get here");
} catch (Exception e) {
    System.out.println("Will not get here");
} finally {
    System.out.println("Will also get here");
} System.out.println("Will get here.");

22-45: Finally

try {
    int x = 3 / 0;
    System.out.println("Won't get here");
} catch (IndexOutOfBoundsException e) {
    System.out.println("Won't get here either");
} finally {
    System.out.println("Will get here.");
} System.out.println("Won't get here");

22-46: Hands-on

- Minilab, as described on website