Error handling with Exceptions

Error handling options so far

- Return an error code
- Check the state of an object
- Cross your fingers

Crossed fingers (AKA Cowboy Coding)

```
static int getArraySizeCowboyStyle(int[] intArray){
  return intArray.length;
  // What if intArray is null?
}
```

```
static void withdrawFromAcctCowboyStyle(int amount, Account acct){
  acct.setBalance(acct.getBalance() - amount);
}
```

Returning an error code

```
static int getArraySizeWithErrorCode(int[] intArray){
  if( intArray == null ){ return -1; } // Error code -1. Meaning...?
  return intArray.length;
}
```

```
static boolean withdrawFromAcctWErrorCode(int amount, Account acct){
  if(acct.getBalance < amount) { return false; }
  acct.setBalance(acct.getBalance() - amount);
  return true;
}
```

Client code must check the error code.

Checking object state

```
public static void main(String[] args){
  int withdrawalAmount = 100;
  Account myAccount = new Account();
  withdrawFromAcctCowboyStyle(withdrawalAmount, myAccount);
  if(myAccount.getBalance() < 0){ //uh oh! What should we do?
    System.out.println("Account overdrawn, better put that money back!");
  }
}
```

Again, client programmer must know to look for the exceptional situation.

Another Way

What if we create a special object for conveying erroneous states?

```
class ErrorHolder{
  public boolean errorOccurred = false;
  public String errorType = "";
  public boolean operationCompleted;
}
```

```
static int getArraySizeWithErrorObj(int[] intArray, ErrorHolder err){
  if( intArray == null ){
    err.errorOccurred = true;
    err.errorType = "Null array received";
    err.operationCompleted = false;
    return 0;  //Technically true
  }
  return intArray.length;
}
```

Using ErrorHolder

```
public static void main(String[] args){
  int[] nullArray = null;
  ErrorHolder err = new ErrorHolder();
  int result = getArraySizeWithErrorObj(nullArray, err);
  if(err.errorOccurred){
    System.out.println("ERROR:\t" + err.errorType);
  }
  else { System.out.println("Elements:\t" + result); }
}
```

Introducing Exceptions

Exceptions are better ErrorHolders

- Baked into Java
- Special syntax for handling problems
- Compiler requires client code to acknowledge them

What are some exceptions?

Most programmers encounter Exceptions long before learning about them formally. What are three that you have encountered?

Commonly Seen Exceptions

- `NullPointerException`
- `ArrayIndexOutOfBoundsException`
- `ArithmeticException`

Try-catch blocks

- Code that may throw an exception is enclosed in a block preceded by the keyword `try`

- The code that handles exceptions is in a block preceded by the keyword `catch`

```
try{ int x = 5/0; }
catch(Exception e){
  e.printStackTrace();
  System.out.println("And now back to our regularly scheduled program.");
}
```

Exception Handling Pitfall

Exception Handlers (`catch` blocks) run in the order they are declared until all exceptions have been handled. If there are no exceptions left other handlers will not run. EG:

```
try{ int x = 5/0; }
catch(Exception e){
  System.out.println("Caught an Exception");
}
catch (ArithmeticException e){ // Compilation error
  System.out.println("Caught an Arithmetic Exception.");
}
```

```
try{ int x = 5/0; }
catch (ArithmeticException e){
  System.out.println("Caught an Arithmetic Exception.");
}
catch(Exception e){
  System.out.println("Caught an Exception");
}
```

Throwing Exceptions

- Throwing exceptions allows the current method to abruptly abort when problems occur
- Throw exceptions with the `throw` keyword
- Declare methods that throw exceptions with the `throws` keyword (`throw` != `throws` , but they are closely related)

Counting list elements with exceptions

```
static int getArraySizeWithException
           throws NullPointerException (int[] intArray) {
  if(intArray == null) {
    throw new NullPointerException("Int array was null");
  }
  return intArray.length;
}
```

Creating your own exceptions

Exceptions are a class of objects just like any other. The `Exception` class can be extended to create custom exceptions

```
class NullArrayException extends Exception{}
```

```
static int getArraySizeWithException(int[] intArray)
           throws NullArrayException{
  if(intArray == null) { throw new NullArrayException();}
    return intArray.length;
}
```

Finally blocks

- `try` blocks can also be followed by a `finally` block (with or without a `catch`)
- `finally` blocks run regardless of exceptions

```
try{ int x = 5/0; }
catch(Exception e){
  e.printStackTrace();
}finally {
  System.out.println("And now back to our regularly scheduled program.");
}
```

Checked and Unchecked Exceptions

- The compiler requires methods that throw `Exception`s to be enclosed in try blocks, or contained in a method that throws an `Exception` as well.

- One exception (*ahem*) to this rule: `RuntimeException` and its subclasses are not checked by the compiler
- This is why you can perform division or access arrays without using a `try` block (`ArithmeticException` and `ArrayIndexOutOfBoundsException` are subclasses of `RuntimeException`).

Try-with-resources

Starting with Java 7, `try` blocks can set up resources and clean them up automatically. This is often used for file I/O.

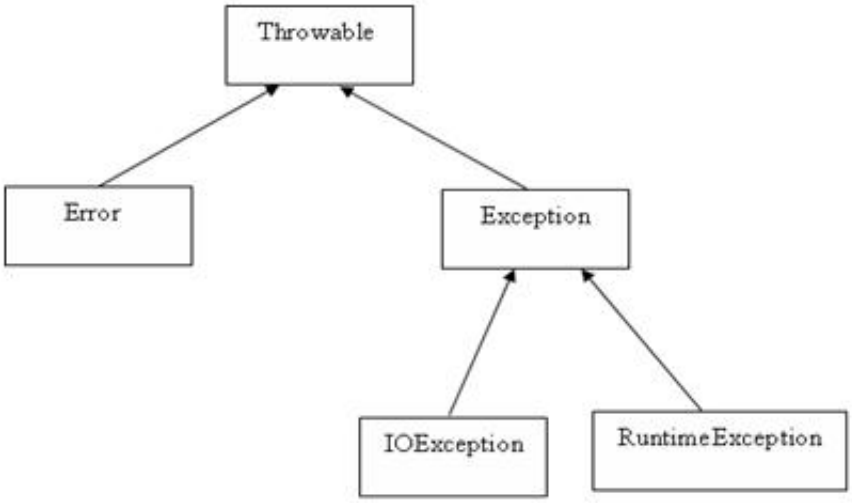
```
static String readFirstLineFromFile(String filepath){
  try(BufferedReader br =
            new BufferedReader(new FileReader(filePath))){
    return br.readLine();
  }
  catch(IOException e){ ... }
}
```

Rethrowing Exceptions

- Sometimes you catch an exception too soon, or can only partially recover from it
- You can throw an exception you caught, to be handled higher up the call stack

```
try{
  exceptionalMethod();
} catch (ExceptionalException e){
  System.out.println("Caught an ExceptionalException. Carry on.");
} catch (Exception e){
  System.out.println("Caught some other type of exception, " +
       "not sure how to proceed.");
  throw e;
}
```

Exceptions are Throwable objects



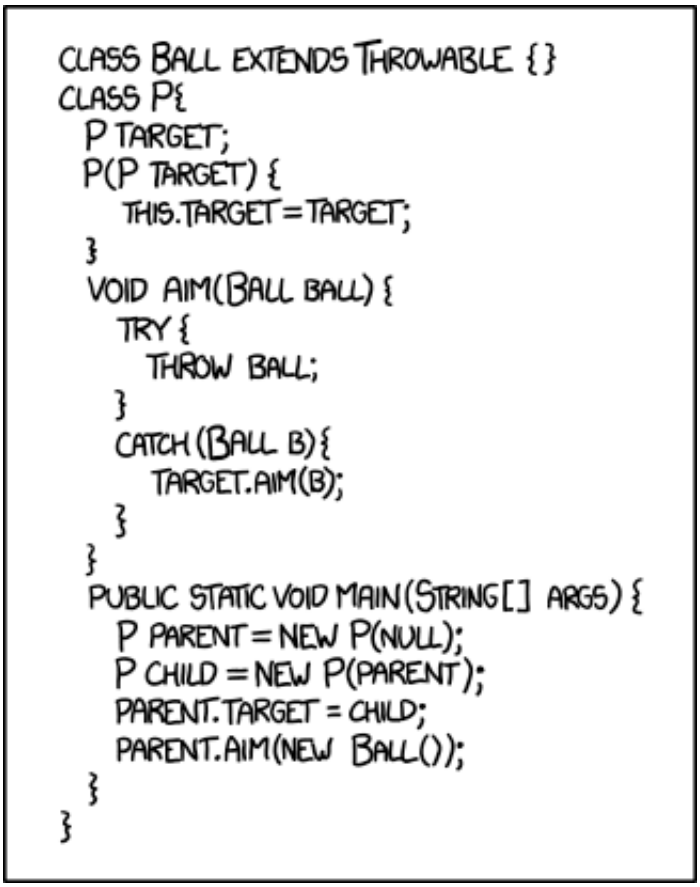
Throwable Methods

- `getMessage()` - get the detail message
- `printStackTrace()` - print the call stack (list of methods called to get to the point where this was thrown)
- `getStackTrace()` - returns an array of `StackTraceElement`s reflecting the call stack
- `fillInStackTrace()` - updates the stack trace to reflect the current call stack
- `initCause(Throwable t)` - Set the cause of this throwable (used in exception chaining)

Exceptions vs Errors

[The difference between Exception and Error]

Bonding



Source: XKCD #1188

More resources

- Oracle Exceptions Tutorial