UCL DEPARTMENT OF ELECTRONIC & ELECTRICAL ENGINEERING

ELEC0021 - PROGRAMMING II
OBJECT-ORIENTED PROGRAMMING

JAVA EXCEPTIONS

Prof. George Pavlou
Communication and Information Systems Group
Dept. of Electronic and Electrical Engineering
http://www.ee.ucl.ac.uk/~gpavlou/
g.pavlou@ucl.ac.uk

Exceptions

- An exception is an indication of a problem that occurs during program execution
  - In many cases, handling the exception allows the program to continue as if no problem had occurred
  - In more severe cases, exception handling allows the program to terminate in a controlled manner
- Exception handling enables programmers to write robust and fault-tolerant programs
- C++ was the first language to introduce exceptions and Java exception features are very similar to those of C++

Exceptions (cont'd)

- Exception handling without relevant language support is also possible, with the programmer including various checks in the "line" of program execution
  - In this case, correct execution and error handling code is mixed, reducing both program clarity and extensibility
  - In some cases, error processing code is often introduced late or even omitted, resulting in less robust programs
- Language-based exception handling, as in C++ and Java, forces programmers to deal with checked exceptions that are identified at *compile time*, e.g. IOException
- Catching also *run time* or unchecked exceptions results in programs which separate basic program functionality from error handling and are more clear, extensible and robust

Exception Examples

- Example of a checked exception:
  - IOException
- Examples of unchecked exceptions:
  - ArrayIndexOutOfBoundsException when trying to access an array element with a too large index
  - ArithmeticException for integer division by zero
  - InputMismatchException when an inappropriate input to what expected is passed, e.g. an integer is expected from the standard input and a double or a string is passed
  - NumberFormatException e.g. when trying to convert a string to an integer but the string does not have the appropriate format, “12” is ok but “1b” isn’t
  - NullPointerException when trying to access an object through a null object reference

Exception Handling

- An exception that occurs in program execution can be thrown by a method using the throw clause in the method's declaration
  - An exception thrown by a method will need to be caught by code in the method calling it or thrown again by the calling method until it is eventually caught
- An exception can be caught by "enveloping" the code that can throw it with a try block followed by one or more catch blocks, one for each exception thrown
  - A catch block effectively catches and handles an exception

Division by Zero Without Exception Handling

```
public class DivideByZero {

    public static int quotient (int numerator, int denominator) {
        return numerator / denominator;
    }

    public static void main (String[] args) {
        if (args.length != 2) {
            System.err.printf("need 2 arguments, try again\n");
            System.exit(1);
        }
        int num   = Integer.parseInt(args[0]);
        int denom = Integer.parseInt(args[1]);
        System.out.printf("result is %d\n", quotient(num, denom));
    }

}  // end class DivideByZero
```

Division by Zero With Exception Handling

```
public class DivideByZeroWithExceptions {

    public static int quotient (int numerator, int denominator)
                                                    throws ArithmeticException {
        return numerator / denominator;
    }

    public static void main (String[] args) {
        if (args.length != 2) {
            System.err.printf("need 2 arguments, try again\n");
            System.exit(1);
        }

 // continues on the next page
```

Division by Zero With Exception Handling (cont'd)

```
// continues from the previous page

        try {
            int num   = Integer.parseInt(args[0]);
            int denom = Integer.parseInt(args[1]);
            System.out.printf("result is %d\n", quotient(num, denom));
        }
        catch (NumberFormatException e) {
            System.err.printf("Exception: %s\n", e);
            System.err.printf("arguments must be integers, try again\n");
        }
        catch (ArithmeticException e) {
            System.err.printf("Exception: %s\n", e);
            System.err.printf("2nd argument should not be zero, try again\n");
        }
    }

} // end class DivideByZeroWithExceptions
```

Some Important Details

- When an exception occurs within the try block, program control is transferred to the catch block that deals with the exception and omits any other try block code further down
  - In the example, when the parseInt(argv[1]) throws a NumberFormatException, the subsequent printf statement that invokes the quotient method is omitted
- A method can declare a number of exceptions it throws or the methods it calls throw, separated by commas
  - In the example, the quotient method throws ArithmeticException
- An exception is an object, so its toString() method can be called in the *catch* block
  - The toString method was implicitly called in the first printf in the two try blocks

Summary

- Implement the divide-by-zero program in the lab to see how exceptions can help with program errors
  - This program does not use O-O Java features but benefits from its exception handling facilities – which C hasn't
- Exceptions allow programs to separate basic program functionality from error handling and are more clear, extensible and robust
- Exceptions are objects themselves and the relevant inheritance hierarchy (see later lecture) is:
  - Throwable <- Exception <- IOException (checked exception)
  - Throwable <- Exception <- RuntimeException <- *various* (unchecked exceptions)