

Introduction to Java IV: Exceptions

CS 1025 Computer Science Fundamentals I

Stephen M. Watt

University of Western Ontario

Things Can Go Wrong

- When developing software we should program defensively.
 - Users might give wrong input.
 - Our program might have bugs that lead to invalid data.
 - Some necessary resource might not be available.
 - Our program might be maintained by someone with an incomplete (i.e. wrong) understanding of it.
 - Versions of software we depend might change.
 - Unexpected situations may arise from combinations of input.
 - ...
- In practice this means we should
 - Make our programs as obvious and as easy to understand as possible.
 - Check for consistency / correctness of data before it is used, especially if it is coming from a user or another, loosely coupled program.
- Myself, I spend about 1/3 the time making a correct program and about 2/3 of the time making it clear, obvious and safe.

When Things Go Worn

- What should we do when we have wrong input or an unexpected situation?
 - Stop?
 - Print a message and ignore it?
 - Have local error checking and recovery?
 - Have non-local error checking and recovery?
- Example of local error checking: C library error codes.
 - Require lots of code to handle properly.
 - Therefore rarely checked by programs.
 - => Core dumps. Buffer overflow security holes.
- Example of non-local error checking: Exceptions.

Exceptions

- Objects that give information about what happened.
- Can be “thrown” by code up the call stack to some higher-level routine that deals with unexpected situations.
- Each function specifies which exceptions it might throw.
- The higher-level routines indicate what they are prepared to handle with “try-catch” statements.
- Can have class hierarchies of exceptions to have common handlers for base classes.

Example: Definition and Throwing

```
public class ExceptionExample {

    public static class ArithmeticException extends Exception {
        public String message;
        public ArithmeticException(String what) {message = what; }
    }

    public static int isqrt(int n) throws ArithmeticException {
        int i;
        if (n < 0)
            throw new ArithmeticException("Negative argument.");
        i = 0;
        while (i*i < n) i++;
        if (i*i != n)
            throw new ArithmeticException("Not a square");
        return i;
    }

    public static void main(String[] args) {
        ...
    }
}
```

Example: Catching and Handling

```
public class ExceptionExample {

    public static class ArithmeticException extends Exception { ... }

    public static int isqrt(int n) throws ArithmeticException { ... }

    public static void main(String[] args) {
        try {
            int n = 100;
            int r = isqrt(n);
            System.out.println("Square root of " + n + " is " + r);
            n = 0;
            r = isqrt(n);
            System.out.println("Square root of " + n + " is " + r);
            n = -4;
            r = isqrt(n);
            System.out.println("Square root of " + n + " is " + r);
            n = 9;
            r = isqrt(n);
            System.out.println("Square root of " + n + " is " + r);
        }
        catch (ArithmeticException ae) {
            System.out.println("Cannot compute square root: " + ae.message);
        }
    }
}
```

Example: Output

Square root of 100 is 10

Square root of 0 is 0

Cannot compute square root: Negative argument.

- In general, the Exception classes can come from anywhere (be local to a class, to a set of classes, or part of a public pkg).
- In general, the throw-er and the-catcher can be in different modules/classes.