*Essential*

Hoping This Works

# Errors Handling

## Axioms

- Software always contains errors
- TDD & Testing couldn’t find all errors
- External environment changes couldn’t be controlled
- Software evolves

## Error codes

- Use POD (plain old data) to define status of operation (integer, boolean, etc.)
- Sometimes mix an error status and result, 0 – error, non-zero result - success
- Specific variable as last operation status (allocated inside TLS – Thread Local Storage)
- RESTful API operation results

# Error codes

```
std::string userName('');
if (getUser(userName)) {
 // process user
 } else {
 // error, user not found
}
```

```
HANDLE hFile;
hFile = CreateFile(argv[1],                 // name of the write
           GENERIC_WRITE,                   // open for writing
           0,                               // do not share
           NULL,                            // default security
           CREATE_NEW,                      // create new file only
           FILE_ATTRIBUTE_NORMAL,           // normal file
           NULL);                           // no attr. Template

if (hFile == INVALID_HANDLE_VALUE) {
           DisplayError(TEXT("CreateFile"));
           _tprintf(TEXT("Terminal failure: Unable to open file \"%s\" for write.\n"), argv[1]);
           return;
}
```

## Exceptions

- Hierarchy of classes defines a set of possible errors in the system
- Stack unwinding
- Additional code, generated by build system, to support exception handling, generating & catching

# Exceptions

```
class Division {
        public static void main(String[] args) {
                int a, b, result;
                Scanner input = new Scanner(System.in);
                System.out.println("Input two integers");
                a = input.nextInt();
                b = input.nextInt();

                // try block
                try { result = a / b;
                        System.out.println("Result = " + result);
                }

                // catch block
                catch (ArithmeticException e) {
                        System.out.println("Exception caught: Division by zero.");
                }
        }
}
```

## Pros & Cons, suggestions

- **Don't** mix **error code** and **exception** within single “**module**”
- Exception **shouldn't leave** “module”
- **Log** all information about **error**
- **Don’t** use **top** level exception **handler** for all **unhandled** exceptions
- Use **exception**, if additional **hidden management** is not an issue
- **Develop hierarchy** of **exceptions** during design time
- **Define clear** rules how to **transform** error **codes** into **exceptions** and vise versa

# Unit tests

# Static Code Analysis