COMP 3550

4.1 — EXCEPTIONAL CASES, EDGE TESTING, AND ASSERTION TOOLS

Week 4: Exceptional Testing & Technical Debt

"HAPPY PATH" ISN’T ENOUGH

Quick Recap: Standard Unit Tests

- Test expected inputs and valid use cases
- Ensure core functionality works
- Fast feedback, focused, isolated

"HAPPY PATH" ISN’T ENOUGH

Quick Recap: Standard Unit Tests

- Test expected inputs and valid use cases
- Ensure core functionality works
- Fast feedback, focused, isolated

But What If Users...

- Enter invalid data?
- Ignore instructions?
- Chain unexpected actions?
- Use your API in “creative” ways?

"HAPPY PATH" ISN’T ENOUGH

Quick Recap: Standard Unit Tests

- Test expected inputs and valid use cases
- Ensure core functionality works
- Fast feedback, focused, isolated

But What If Users...

- Enter invalid data?
- Ignore instructions?
- Chain unexpected actions?
- Use your API in “creative” ways?

Your system passes all tests but breaks in production

- Because the happy path isn’t the only path users take

WHAT IS AN EDGE CASE?

- Classic Examples:
    - 🥇 First element
    - 🏁 Last element
    - 🕳️ Empty list
    - 🫥 Null input
    - ➗ Division by 0
    - ➕ Adding null to a list
    - 🔍 Empty search result

WHAT IS AN EDGE CASE?

- Classic Examples:
  - 🥇 First element
  - 🏁 Last element
  - 🕳️ Empty list
  - 😐 Null input
  - ➗ Division by 0
  - ➕ Adding null to a list
  - 🔍 Empty search result

Why They Matter:

- Can cause crashes, exceptions, or silent failures
- Are often where real-world bugs hide

TESTING EXCEPTIONAL CASES

Don’t just test what should work — test what should fail.

TESTING EXCEPTIONAL CASES

Don’t just test what should work — test what should fail.

What would you test here?

```
public void createUser(String email) {
    if (!email.contains("@")) {
        throw new IllegalArgumentException("Invalid email format");
    }
    // continue...
}
```

TESTING EXCEPTIONAL CASES

Don’t just test what should work — test what should fail.
What would you test here?

```java
public void createUser(String email) {
   if (!email.contains("@")) {
      throw new IllegalArgumentException("Invalid email format");
   }
   // continue...
}
```

```java
@Test
void rejectsBadEmail() {
    assertThrows(IllegalArgumentException.class, () -> {
        createUser("invalid-email");
    });
}
```

DON’T CATCH OR SWALLOW EXCEPTIONS

Why It’s a HUGE NO:

- Catches everything — even bugs you didn’t expect
- Swallowed exceptions disappear silently
- Tests may pass when the code is broken
- Debugging becomes a guessing game

NEVER CATCH OR TEST FOR EXCEPTION

Nowhere in your **code** should there be a swallowed exceptions like:

```
try {
   doSomething();
} catch (Exception e) {
   // nothing happens — swallowed 😬
}
```

NEVER CATCH OR TEST FOR EXCEPTION

Nowhere in your **code** should there be a swallowed exceptions like:

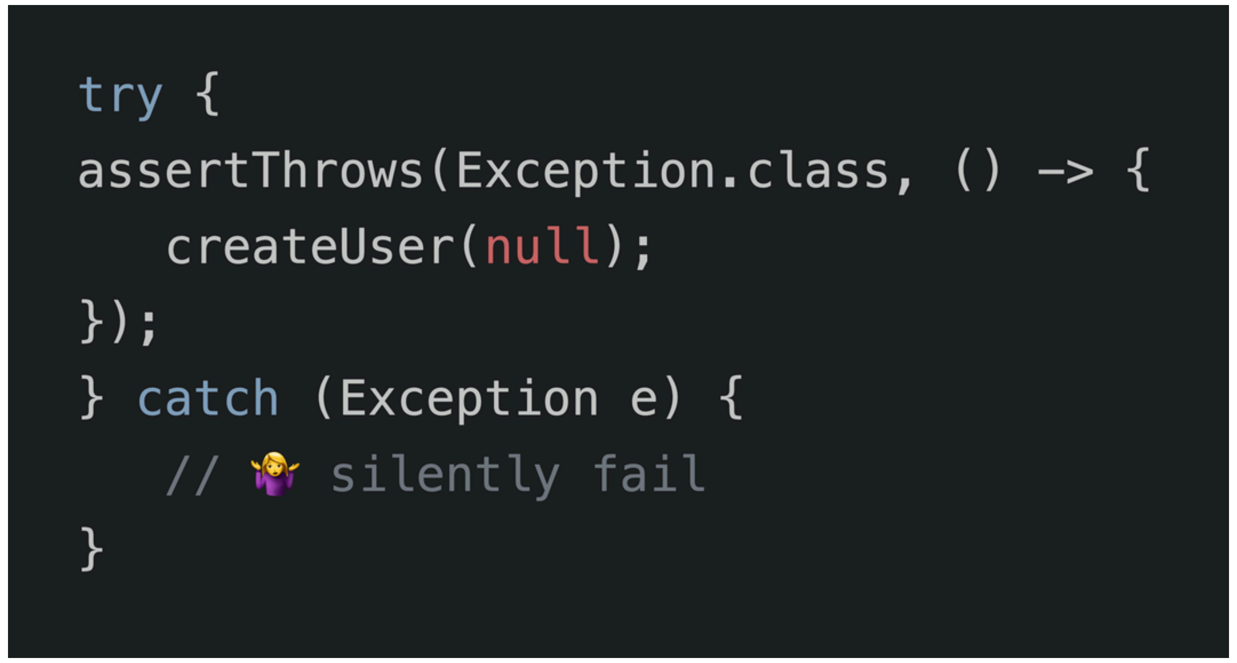
```
try {
   doSomething();
} catch (Exception e) {
   // nothing happens — swallowed 😬
}
```

or even....

```
try {
   doSomething();
} catch (SuperSpecificException e) {
   // STILL swallowed
}
```

NEVER CATCH OR TEST FOR EXCEPTION

Nowhere is your **tests** should be:



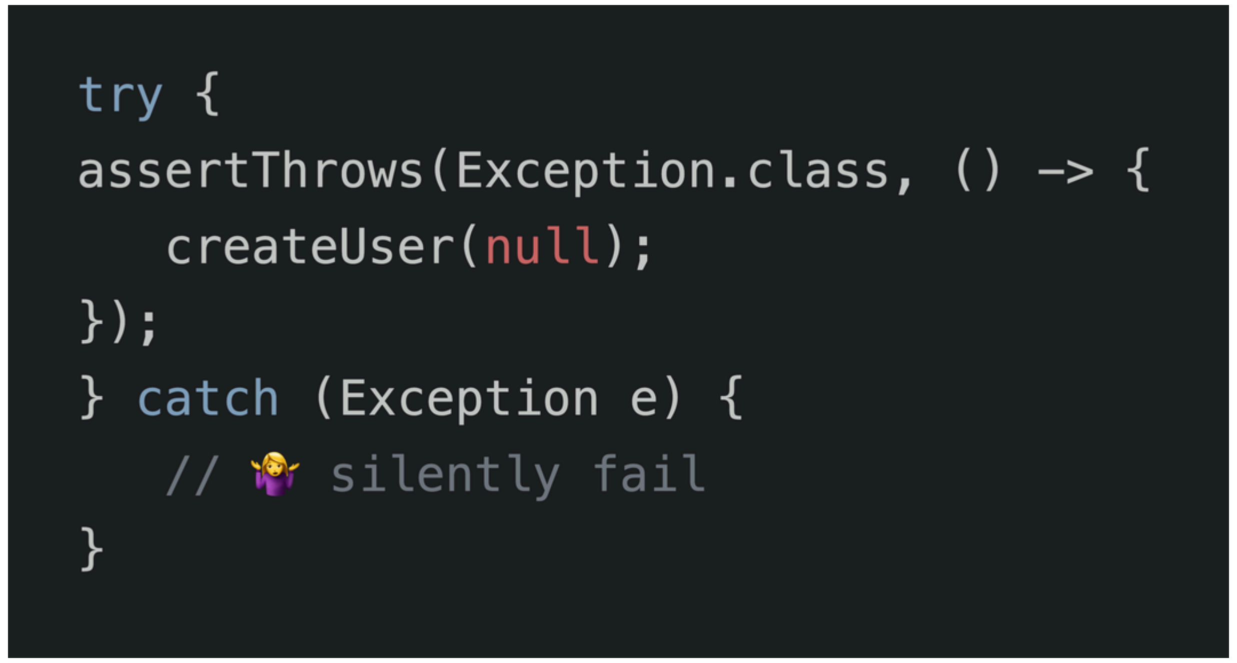
```
try {
assertThrows(Exception.class, () -> {
   createUser(null);
});
} catch (Exception e) {
   // 🤷‍♀️ silently fail
}
```

Throw or catch specific exceptions like:

- IllegalArgumentException
- NullPointerException ***(sparingly!)***
- IOException, etc.

NEVER CATCH OR TEST FOR EXCEPTION

Nowhere is your **tests** should be:



```
try {
assertThrows(Exception.class, () -> {
   createUser(null);
});
} catch (Exception e) {
   // 🤷‍♀️ silently fail
}
```

Throw or catch specific exceptions like:

- IllegalArgumentException
- NullPointerException ***(sparingly!)***
- IOException, etc.

🎯 Be precise in tests
🚨 Never silence failure
🧹 Swallowed exceptions = hidden messes that will blow up later

ASSERTION TECHNIQUES (WITH EXAMPLES)

Common JUnit Assertions:

```
assertEquals(42, result);               // exact value
assertTrue(user.isActive());            // boolean condition
assertNotNull(response);                // non-null check
assertThrows(IllegalArgumentException.class, () -> {
   createUser("no-at");
});
```

ASSERTION TECHNIQUES ADVANCED [OPTIONAL]

More Expressive: **assertThat** (**JUnit** + **Hamcrest**)

```
assertThat(score, is(greaterThan(80)));
assertThat(name, startsWith("Dr."));
```

ASSERTION TECHNIQUES ADVANCED [OPTIONAL]

Even Better: **AssertJ** for Fluent, Readable Assertions

```
assertThat(list)
    .isNotEmpty()
    .contains("apple")
    .doesNotContain("banana");

assertThat(thrown)
    .isInstanceOf(IllegalArgumentException.class)
    .hasMessageContaining("Invalid");
```

EXAMPLE CODE COMPARISON

Basic JUnit Test

```
@Test
void testEmail() {
   User u = createUser("bob@example.com");
   assertNotNull(u);
}
```

EXAMPLE CODE COMPARISON

Basic JUnit Test

```
@Test
void testEmail() {
   User u = createUser("bob@example.com");
   assertNotNull(u);
}
```

Improved Test: Multiple Assertions + Exception

```
@Test
void createsValidUserAndRejectsInvalid() {
   User u = createUser("bob@example.com");

   assertNotNull(u);
   assertEquals("bob@example.com", u.getEmail());
   assertTrue(u.isActive());

   assertThrows(IllegalArgumentException.class, () -> {
      createUser("invalid-email");
   });
}
```

Unit test does not need to be **one line**
BUT RATHER one unit of code

This is better because:

- Verifies multiple behaviors in one logical flow
- Catches edge cases and invalid inputs
- Documents expected behavior clearly

TEST DATA DESIGN

Include “Bad” Data On Purpose:

- Nulls
- Empty strings
- Invalid formats ("abc@com")
- Boundary values (0, -1, Integer.MAX_VALUE)

Edge-Focused Test Plan:

- Think: “What’s plausible but unusual?”

TEST DATA DESIGN

Include “Bad” Data On Purpose:

- Nulls
- Empty strings
- Invalid formats ("abc@com")
- Boundary values (0, -1, Integer.MAX_VALUE)

Edge-Focused Test Plan:

- Think: “What’s plausible but unusual?”

Consider:
“What’s the worst email you could pass to createUser() and expect the test to catch?”

```
void createUser(String email) {
   try {
      validateEmail(email);
  } catch (EmailExceptions ee) { /* ... */ }
}
```

```
void validateEmail(String email) {
   /* logic for email validations
    * throws various exceptions for
    * a variety of invalidate inputs
    */
}
```

PAUSE AND PROJECT REFLECTION

Pick one of your existing test cases/methods that **you did not write (something a team member wrote for the project)**.

Add 2 edge tests and one failure case.