COMP1531

✅ Correctness - Exceptions

Lecture 5.4

Author(s): Hayden Smith

(Download as PDF)

In This Lecture

- **Why? 🤔**
  - Finding more graceful ways to deal with errors makes your program more robust
- **What? 📰**
  - Exceptions
  - Raising & Catching Exceptions

😡 Dealing With Problems

The simplest way to deal with problems at **run-time**...

Just crash

```typescript
import prompt from 'prompt-sync';
const promptFn = prompt();

function sqrt(x: number) {
  if (x < 0) {
    console.error('Error Input < 0');
    process.exit(1);
  }
  return Math.pow(x, 0.5);
}

const input = promptFn('Please enter a number: ');
console.log(sqrt(parseInt(input)));
```

5.4_just_crash.ts

Not very clean though.

😡 Dealing With Problems

However, if we **throw an exception** 😵‍💫 we start to get into a new territory of programming.

```typescript
import prompt from 'prompt-sync';
const promptFn = prompt();

function sqrt(x: number) {
  if (x < 0) {
    throw new Error('Error Input < 0');
  }
  return Math.pow(x, 0.5);
}

const input = promptFn('Please enter a number: ');
console.log(sqrt(parseInt(input)));
```

5.4_exception1.ts

😡 Dealing With Problems

However, if we **throw an exception** 😵‍💫 we start to get into a new territory of programming.

```typescript
import prompt from 'prompt-sync';
const promptFn = prompt();

function sqrt(x: number) {
  if (x < 0) {
    throw new Error('Error Input < 0');
  }
  return Math.pow(x, 0.5);
}

const input = promptFn('Please enter a number: ');
console.log(sqrt(parseInt(input)));
```

5.4_exception1.ts

😡 Dealing With Problems

However, if we **throw an exception** 😵‍💫 we start to get into a new territory of programming.

```typescript
import prompt from 'prompt-sync';
const promptFn = prompt();

function sqrt(x: number) {
  if (x < 0) {
    throw new Error('Error Input < 0');
  }
  return Math.pow(x, 0.5);
}

const input = promptFn('Please enter a number: ');
console.log(sqrt(parseInt(input)));
```

5.4_exception1.ts

Let's take a step back...

⚡ Exceptions

An **exception** is an action that disrupts the normal flow of a program. This action is often representative of an error being thrown. Exceptions are ways that we can elegantly recover from errors.

⚡ Exceptions

Exceptions are a particular method of ensuring **software safety**. Different languages have different conventions for managing unexpected runtime events.

Javascript relies on Exceptions for the majority of error handling. Unlike C, which has no exceptions

🌻 Easier To Ask Forgiveness Than Permission

- EAFP is the javascript convention for handling errors.
- It encourages you to assume something will work and just have an exception handler to deal with anything that might go wrong
- Pros:
  - Can simplify the core logic
  - Multiple different sorts of errors can be handled with one except block
- Cons:
  - Makes code non-structured
  - Harder to reason what code will be executed.

🌻 Look Before You Leap

- LBYL is a convention for avoiding errors in popular languages like C
- Unlike EAFP it encourages you to check that something can be done before you do it
- Pros:
  - Doesn't require exceptions
  - Code is structured and therefore easier to reason about
- Cons:
  - Core logic can be obscured by error checks

⚡ Exception Examples

This program is good in that it throws an exception, but we aren't handling it.

```typescript
import prompt from 'prompt-sync';
const promptFn = prompt();

function sqrt(x: number) {
  if (x < 0) {
    throw new Error('Error Input < 0');
  }
  return Math.pow(x, 0.5);
}

const input = promptFn('Please enter a number: ');
console.log(sqrt(parseInt(input)));
```

5.4_exception1.ts

⚡ Exception Examples

This program is good in that it throws an exception, but we aren't handling it.

```typescript
import prompt from 'prompt-sync';
const promptFn = prompt();

function sqrt(x: number) {
  if (x < 0) {
    throw new Error('Error Input < 0');
  }
  return Math.pow(x, 0.5);
}

try {
  const input = promptFn('Please enter a number: ');
  console.log(sqrt(parseInt(input)));
} catch (err) {
  console.error(`Error when inputting! ${err}`);
  const input = promptFn('Please enter a number: ');
  console.log(sqrt(parseInt(input)));
}
```

5.4_exception2.ts

⚡ Exception Examples

Or we could make this even more robust

```typescript
import prompt from 'prompt-sync';
const promptFn = prompt();

function sqrt(x: number) {
  if (x < 0) {
    throw new Error('Error Input < 0');
  }
  return Math.pow(x, 0.5);
}

let success = false;
while (!success) {
  try {
    const input = promptFn('Please enter a number: ');
    console.log(sqrt(parseInt(input)));
    success = true;
  } catch (err) {
    console.error(`Error when inputting! ${err}`);
  }
}
```

5.4_exception3.ts

⚡ Exception Examples

- Key points:
  - Exceptions carry data
  - When exceptions are thrown, normal code execution stops

```
function sqrt(x: number) {
  if (x < 0) {
    throw new Error('Error Input < 0');
  }
  return Math.pow(x, 0.5);
}

if (process.argv.length === 3) {
  try {
    console.log(sqrt(parseInt(process.argv[2])));
    console.log('Never called if error!');
  } catch (err) {
    console.error(`Error when inputting! ${err}`);
  }
}
```

5.4_throw_catch.ts

🍅 Testing With Exceptions

We can use `jests` `toThrowError` function to test if functions are appropriately throwing exceptions.

```
function sqrt(x: number) {
  if (x < 0) {
    throw new Error('Error Input < 0');
  }
  return Math.pow(x, 0.5);
}

export { sqrt };
```

5.4_sqrt.ts

```
import { sqrt } from './5.2_sqrt';

describe('sqrt correctness', () => {
  test('deals with valid bases', () => {
    expect(sqrt(4)).toEqual(2);
    expect(sqrt(2)).toBeCloseTo(1.414213, 5);
  });
  test('throws error on negatives', () => {
    // Note that these require a function, not result
    expect(() => sqrt(-2)).toThrow('Error: Input < 0');
    expect(() => sqrt(-5)).toThrowError('Error: Input < 0');
  });
});
```

5.4_catch.test.ts

👂 Feedback

Or go to the form here.