Functional Design

Functional Design

Functional Design

Jan 2021

Takashi Idobe

Exceptional Cases i

```
class Main {
  public static void main(String args[]) {
    div(10, 0);
  }

  public static Integer div(Integer a, Integer b) {
      /* Implement Division here */
  }
}
```

The method signature here is lying: We're saying we have a static function that *always* returns an Integer. But it can throw an exception too.

```
class Main {
  public static void main(String args[]) {
    div(10, 0);
  }

  public static Integer div(Integer a, Integer b) {
      try {
        return a / b;
      } catch (Exception e) {
        throw new IllegalArgumentException("Divide by Zero Error");
      }
  }
}
```

What about this?

```
class Main {
  public static void main(String args[]) {
    div(null, null);
  }

  public static Integer div(Integer a, Integer b) {
    /* Implement Division here */
  }
}
```

So it can be null too:

```
class Main {
  public static void main(String args[]) {
    div(null, null);
  }

  public static Integer div(Integer a, Integer b) {
      if (a == null || b == null) {
          throw new IllegalArgumentException("Divide by Null");
      }
      try {
            return a / b;
      } catch (Exception e) {
          throw new IllegalArgumentException("Divide by Zero Error");
      }
  }
}
```

I just wanted to divide two numbers:

The method writer always needs to be cautious of `Reference` types (which can be null), and throwing exceptions:

There's no way to label functions that can throw Exceptions. The compiler can help with `checked` exceptions, but the function lacks that information in its type signature.

Exceptions are quite unwieldly.

In C++ it's a little better:

```
/* This can throw */
auto div(int a, int b) -> int {
  if (b == 0) throw std::invalid_argument("Divide by Zero Error");

  return a / b;
}
/* If we divide by zero, never throw an exception, return -1 */
auto div(int a, int b) -> int noexcept {
  if (b == 0) return -1;

  return a / b;
}
```

Swift uses `Optional<T>`s

```
func div(a: Int, b: Int) -> Int? {
  if b == 0 {
    return Optional.None
  }
  Optional.Some(a / b)
}
```

Swift also allows for exceptions:

It also inverts the C++ standard, which labels that functions cannot throw: All throwable functions can be labeled:

```
func div(a: Int, b: Int) throws -> Int {
  a / b
}
```

```
func div(_ a: Int, _ b: Int) -> Int {
    a / b
}

print(div(10, 0))
```