# Design Patterns in Other Languages (and Testing)

October 12th, 2022

## Design Patterns

### OOP in `C`

Consider the example in `oop.c`

### Imperiave Programming in `Haskell`

We use monads. Basically we just make structure that holds the value of all of our variables.

## Reflection

Reflection makes classes, methods, and fields into first-class objecgs that exist at runtime.

Consider the example in `reflection.java`

Reflection lets us write cool code that we can't write without it easily. But, it turns compile-time checks into runtime checks, which could introduce bugs easily.

- Reflection does not add any expressive power to the language
- Reflection does not solve any problem

### `class`

We have a few options if we want to get a `Class` object.

```
Class<?> c = String.class;
Class<?> d = "hello".getClass();
Class<?> e = Class.forName("java.lang.String");
```

The first one accesses the static field of a Class. The second one is invoked on an instance of a class (implemented for all objects). The third one is to pass the full string representing the location of that type in the Java Standard Library to the `Class` static method.

The reflective calls nearly all throw some kind of exception. Those need to get caught.

Going further...

```
class A { A(String x, int y) { ... }}
Class<?> c = A.class;
Constructor<?> cons = c.getConstructor(String.class, int.class);
Object o = cons.newInstance("foo", 42);
A a = (A) o;
```

Yay, now no one can ever fire us because our code is so convoluted that no one will ever be able to understand it.

We can also reimplement the factory methods from p2...

```
Hashmap<Character, String> m; // the String is the classname of the Piece
String name = m.get('p');     // assume name.equals("Pawn")
Constructor cons = c.getConstructor();
Piece p = (Piece) cons.newInstance();
```

Or something even more fun:

```
if (str.matches("pwd")) {
    pwd();
} else if (str.matches("cd .*")) {
    cd(...);
} else if .......

String[] split = str.split(" ");
```

***WARNING***

This exposes **any** method of