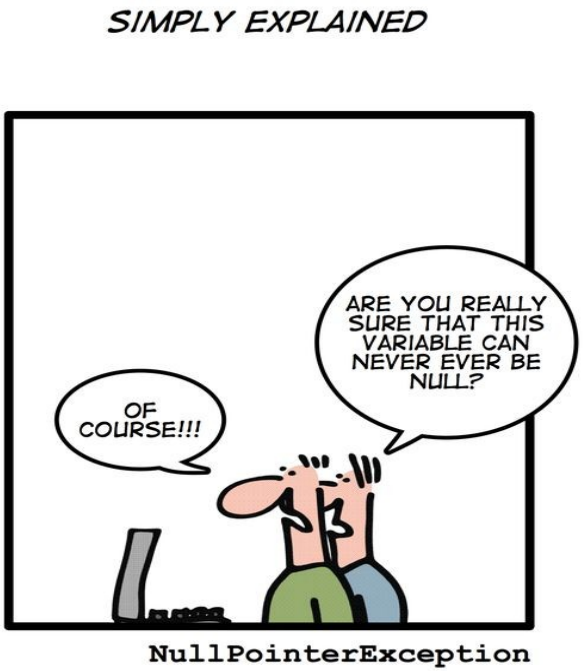
Functors, std::functions



Krishna Kumar

Functors

- A functor is pretty much just a class or a struct which defines the operator(). That lets you create objects which "look like" a function.
- One is that unlike regular functions, they can contain state.

```
class Multiplier { //functor classs
double y;
   public:
    Multiplier(double y): y{y}{};
    double operator()(double x) { return x * y;}
}
Multiplier doubler{2};  // create an instance of the functor class
double x = doubler(5); // call it
```

Functors (cont...)

Multiplier tripler{3}; // create an instance of the functor class

double x = tripler(5); // call it, multiplies given value by 3

std::vector<int> in{1, 2, 3, 4, 5};

// Pass a functor to std::transform, which calls the functor on every element

// in the input sequence, and stores the result to the output sequence

std::transform(in.begin(), in.end(), in.begin(), Multiplier(5));

Pros and Cons

- **instead of plain function**:
  - Pros:
    - Functor may have state
    - Functor fits into OOP
  - Cons:
    - There is more typing, a bit longer compilation time etc.
- **Instead of function pointer:**
  - Pros:
    - Functor often may be inlined
  - Cons:
    - Functor can not be swapped with other functor type during runtime (at least unless it extends some base class, which therefore gives some overhead)
- **Instead of polymorphism:**
  - Pros:
    - Functor (non-virtual) doesn't require vtable and runtime dispatching, thus it is more efficient in most cases
  - Cons:
    - Functor can not be swapped with other functor type during runtime

Better design with Functors

- **get and set accessor functions**

```
class Point {
        double Xcoord_, Ycoord_;
  –   public:
        Point();
        Point( double Xcoord, double Ycoord );
        double getXcoord() const;
        void setXcoord( double newValue );
        double getYcoord() const;
        void setYcoord( double newValue );
};
```

Avoiding accessors?!

- We can eliminate a lot of code by doing:

```
class Point {
    public:
        double Xcoord_, Ycoord_;
        Point();
        Point( double Xcoord, double Ycoord );
};
```

  - **DON't do this. It BREAKS encapsulation!!!**

Simple idiomatic accessor

```
class Point {
        double Xcoord_, Ycoord_;
    public:
        Point();
        Point( double Xcoord, double Ycoord );
        double Xcoord() const;
        void Xcoord( double newValue );
        double Ycoord() const;
        void Ycoord( double newValue );
};
```

Simple idiomatic accessor (cont...)

- **Instead of doing:**
  - Point point;
  - point.setXcoord(point.getXcoord() + 10.);
  - return point.getXcoord();
- **We can do:**
  - Point point;
  - point.Xcoord(point.Xcoord() + 10.);
  - return point.Xcoord();

Objects to the rescue

```
class Coordinate {
        double coord_;
-       public:
        Coordinate();
        Coordinate(double coord) :  coord_ {coord} {};
        double operator()() const {
            return coord_;
         }
        void operator()( double coord ) {
            coord_ = coord; }
};
```

Objects to the rescue

```
class Point {
    public:
        Coordinate Xcoord, Ycoord;
        Point();
        Point( double Xcoord, double Ycoord );
};
```

- **We can do:**
  - Point point;
  - point.Xcoord(point.Xcoord() + 10.);

Advantages

- The class is shorter and much more clearly expresses our intent without any excessive verbosity.
- Instead of writing two sets of accessors which are nearly identical we've written one in a helper class.
- We have two classes that do exactly half the job that one was doing. This means that each class is smaller and easier to understand.
- We're still free to change the underlying implementation if we want to because we haven't changed the syntax used to access the xcoord and ycoord.

Templatised

```
template <class t_coord>
class Coordinate {
        t_coord coord_;
    public:
        Coordinate();
        Coordinate(t_coord coord) :  coord_ {coord} {};
        t_coord operator()() const {
            return coord_;
         }
        void operator()( t_coord coord ) {
            coord_ = coord; }
};
```