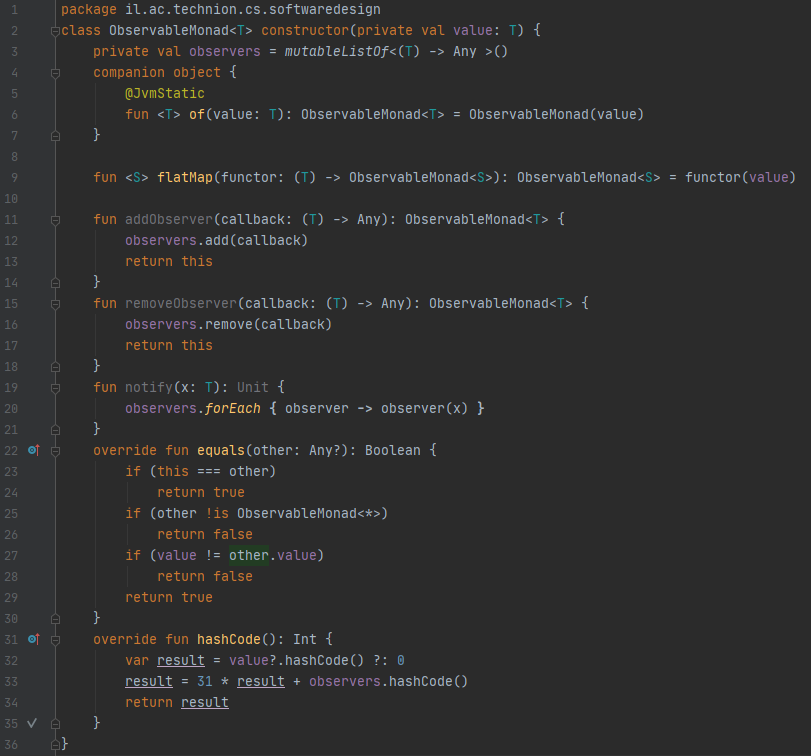
Student Name: Bahjat Kawar Student ID: 206989105

Student Name: Mori Levinzon Student ID: 308328467

**Software Design H.W 2 – Dry Part**

1. Implementing the Observable as Monad:



ObservableMonad is an object that has observers. These observers are represented by callbacks that are saved in a List, and can be added or removed from it. When observableMonad finish calculating the value that was changed by the function it notifies each of the observers.

1. Proving that the monadic laws hold for this definition:

We implemented Test to prove the 3 laws:

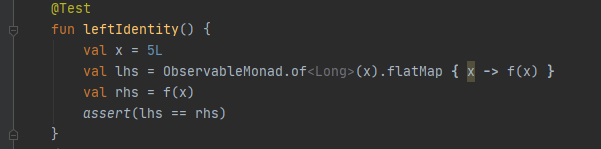
\*We used this functions to prove the identities:



**Left identity**

The first monad law states that if we take a value, put it in a default context with return and then feed it to a function by using flatMap, it's the same as just taking the value and applying the function to it. To put it formally:

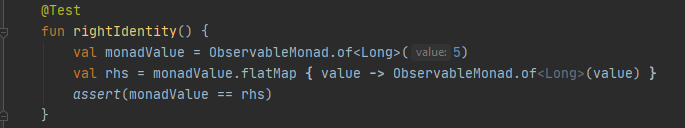
return ObservableMonad.of(x).flatMap(f) is the same thing as f(x)



**Right identity**

The second law states that if we have a monadic value and we use flatMap to feed it to return, the result is our original monadic value. Formally:

m.flatMap{ ObservableMonad.of(it) } is no different than just  m



**Associativity**

The final monad law says that when we have a chain of monadic function applications with flatMap, it shouldn't matter how they're nested. Formally written:

m.flatMap(f).flatMap(g) is just like doing  m.flatMap{ f(it).flatMap(g) }

