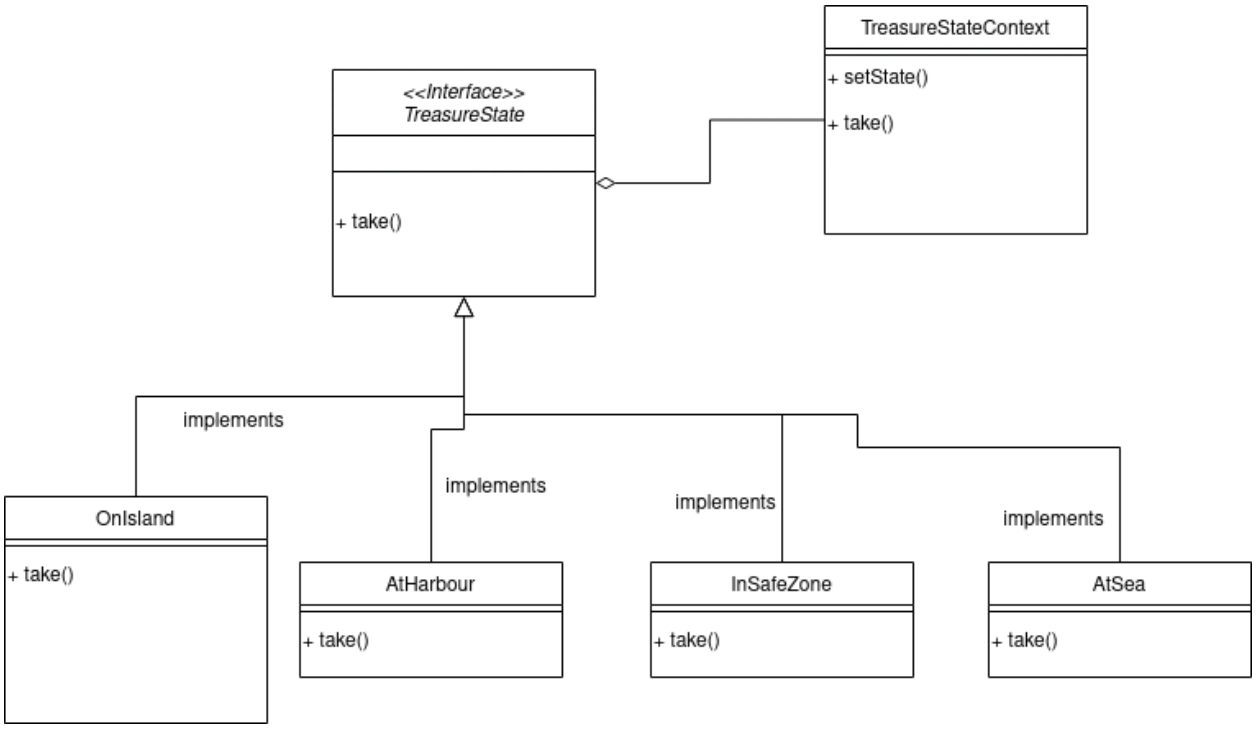
State Design Pattern

A way we can represent states in the program is using the State Design Pattern. This is a better way of changing behaviour based on states instead of using many if statements. Doing it this way means we can add additional states very easily, and have cleaner code which is easier to read and understand.

We have a context class which defines the state of the object, a state interface which defines methods which the different states will change, and a class for each state which implements the state interface.

An example of how this could be used in the game is with the state of a piece of treasure. It could be in a safe zone, on treasure island, at a harbour or in the safe zone and when trying to take that treasure we would want different actions depending on where it is. Below is a UML class diagram for this example:



Prototype

First of all here we have the TreasureState interface. This defines a placeholder take method which the concrete state classes implement:

```
public class TreasureStateContext {
    private TreasureState currentState;
    // By default the treasure is at sea.
    public TreasureStateContext() {
        currentState = new AtSea();
    }

    public void setCurrentState(TreasureState state){
        currentState = state;
    }

    public void take() {
        currentState.take( ctx: this);

    }
}
```

Next we will look at the concrete state classes. These implement the take method and change it to have different behaviour depending on the state:

```
public class OnIsland implements TreasureState {
    @Override
    public void take(TreasureStateContext ctx){
        System.out.println("You take the treasure from the treasure island.");
    }
}
```

```
public class AtHarbour implements TreasureState {
    @Override
    public void take(TreasureStateContext ctx){
        System.out.println("You trade your crew cards for the treasure.");
    }
}
```

```
public class InSafeZone implements TreasureState {
    @Override
    public void take(TreasureStateContext ctx){
        System.out.println("You cannot take this treasure it's in the safe zone!");
    }
}
```

```java
public class AtSea implements TreasureState {
    @Override
    public void take(TreasureStateContext ctx){
        System.out.println("You pick up the treasure at sea.");
    }

}
```

Next we have the context class. This is the class we interact with which defines the current state of the treasure:

```java
public class TreasureStateContext {
    private TreasureState currentState;
    // By default the treasure is at sea.
    public TreasureStateContext() {
        currentState = new AtSea();
    }

    public void setCurrentState(TreasureState state){
        currentState = state;
    }

    public void take() {
        currentState.take( ctx: this);

    }
}
```

And finally here is an application class I made to be able to test out this prototype:

```java
/**
 * Prototype showing an example of state design pattern and how it can be used to represent states in our game.
 *
 * Author: Xander Davies (xad1)
 */
public class Application {
    public static void main(String[] args) {
        TreasureStateContext stateContext = new TreasureStateContext();
        // Default state is at sea
        // Trying to take the treasure at sea
        stateContext.take();
        // Trying to take the treasure in the safe zone
        stateContext.setCurrentState(new InSafeZone());
        stateContext.take();
        // Trying to take the treasure at the harbour
        stateContext.setCurrentState(new AtHarbour());
        stateContext.take();
        // Trying to take treasure from the treasure island
        stateContext.setCurrentState(new OnIsland());
        stateContext.take();
    }

}
```

References:

https://www.geeksforgeeks.org/state-design-pattern/