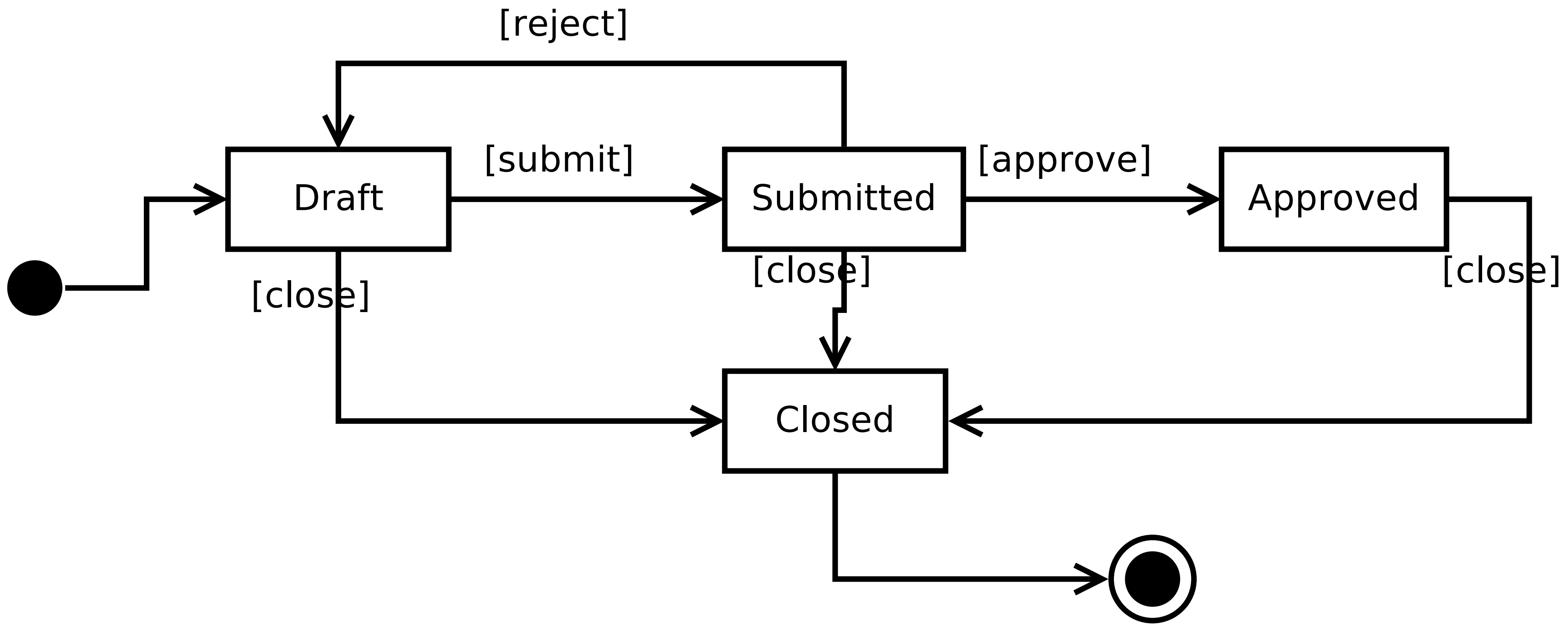
Assignment

→Plan your holiday

Example


[reject]
Draft
[submit]
Submitted
[approve]
Approved
[close]
[close]
[close]
Closed

State

Design patterns

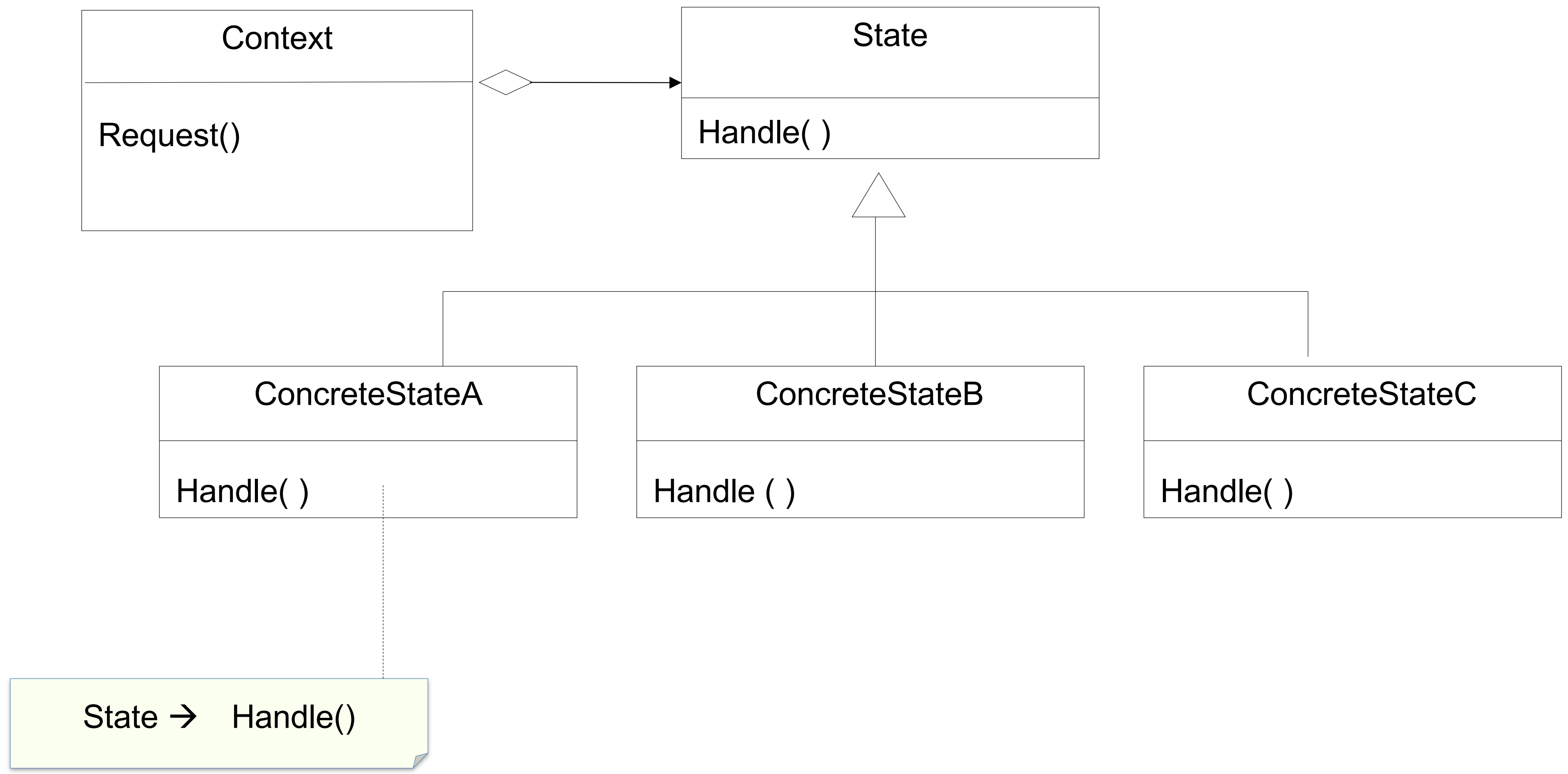
Why?

- Change behavior when state changes
- Avoid code that is hard to maintain (ifs)
- Easy to extend

How

- Create **interface** for state
- Create **implementing classes** for states
- Context class contains all **possible states**
- Context class knows its **current state**
- Method in context class **uses the current state** to handle the request, instead of doing it itself

Class diagram



State vs. Strategy

State

•Client does not choose, depends on context (according to a wel known scheme)

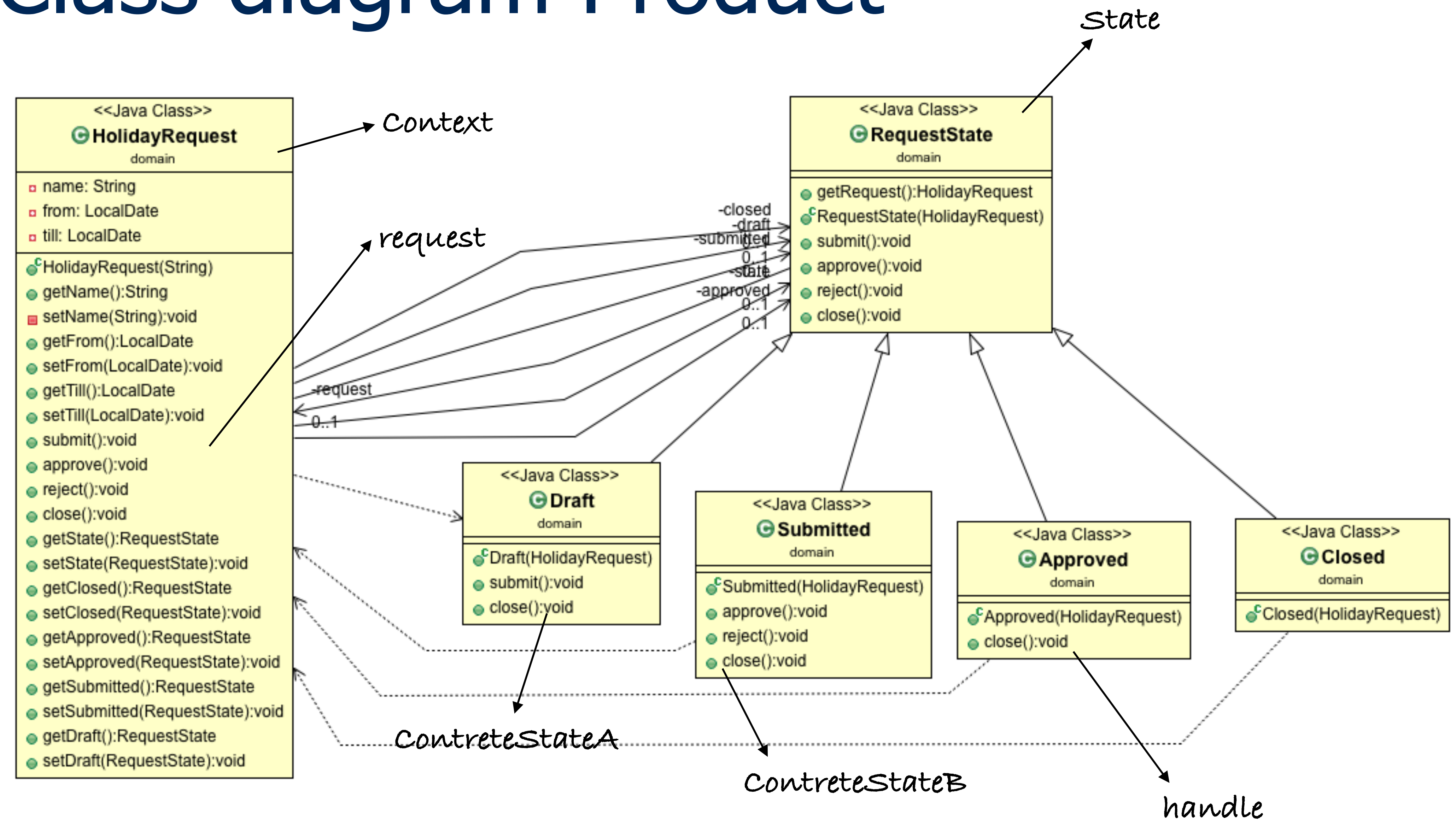
•Alternative for ifs

Strategy

•Client chooses strategy

•Alternative for subclassing

Class diagram Product



State

```
public interface RequestState {

   public void submit();
   public void approve();
   public void reject();
   public void close();

}
```

Assignment

- Implement state-classes

State vs. design principles

How SOLID?

State vs. design principles

- SRP?
    - OK. Each state is encapsulated in its own class
- Open Closed?
    - NOK: introducing a new state means modifying
        - the context class
        - (at least some of) the other states
- Dependency inversion?
    - OK: State interface
- Liskov?
    - Not applicable

?

ON
OFF