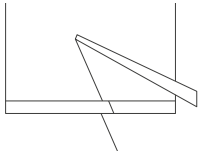
Bring ideas to life
VIA University College



State-Machines & State Pattern

SWE1

What is a State-Machine?

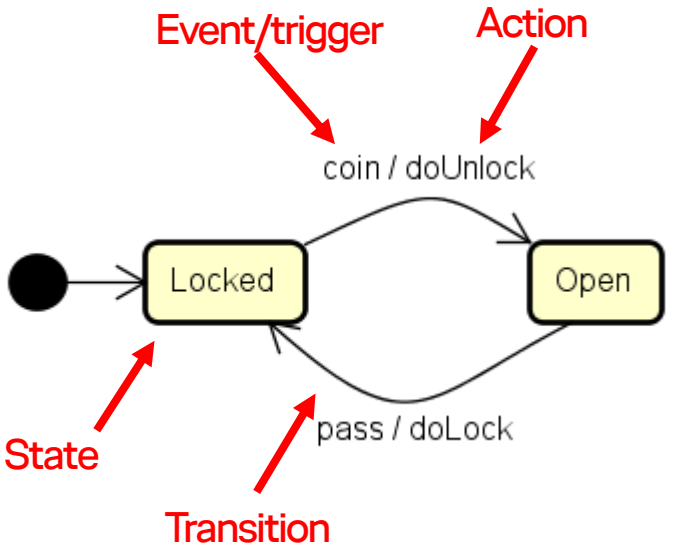
A system/Object that react differently on events depending on its current state

- Elevators
- Protocols
- Alarm clocks
- User Interfaces
- Cars
- Petrol pumps
- Etc.

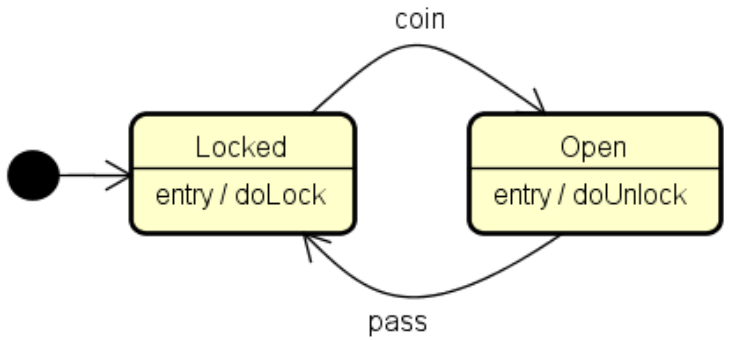
UML State-Machine Diagram

Turnstile example

State Machine Diagram



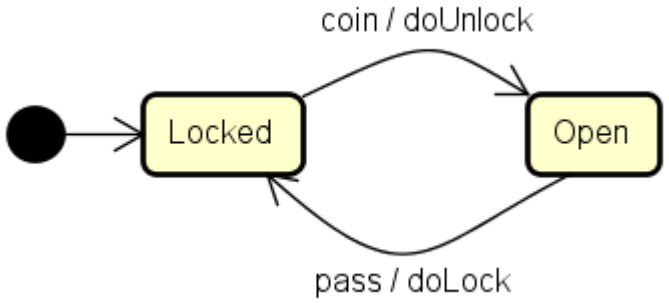
State Machine Diagram - alternative



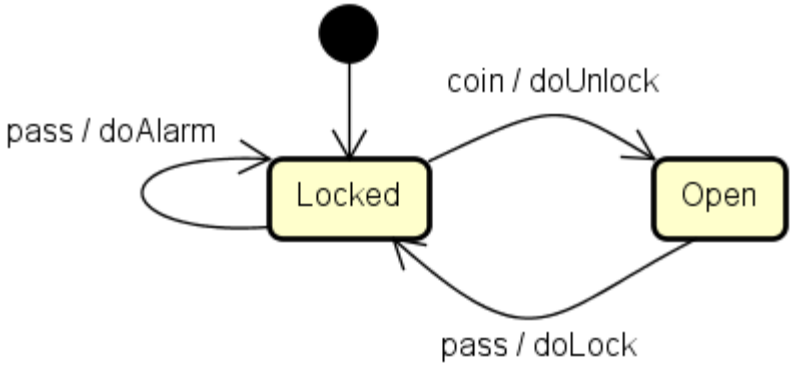
UML State-Machine Diagram

Turnstile example

What should the system do if somebody passes in locked state?



We sound an alarm



Have we covered all possibilities?

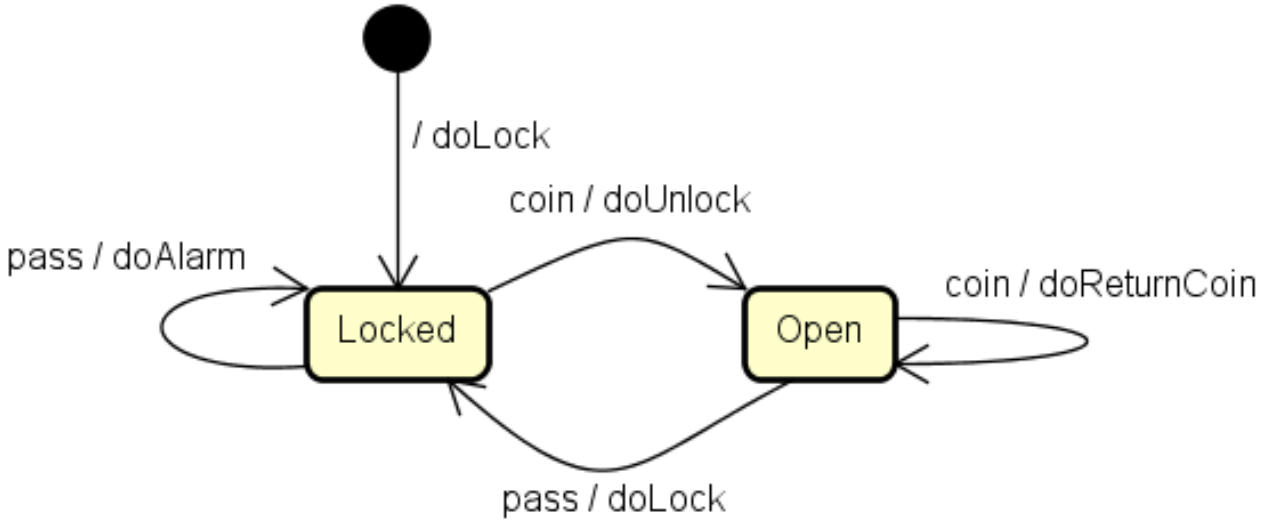
- What happens in *Open* state when somebody puts in an extra coin?

UML State-Machine Diagram

Turnstile example

What should the system do if somebody passes in locked state?

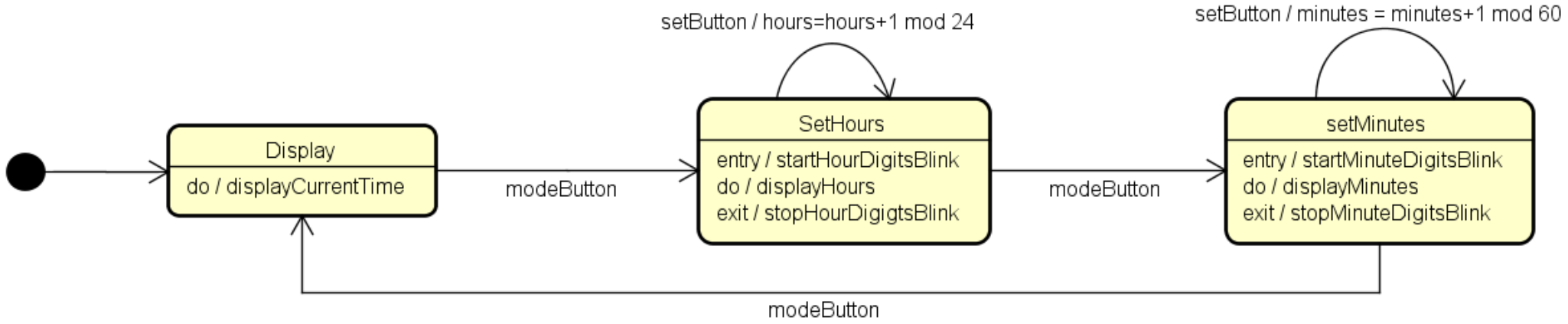
We return the coin



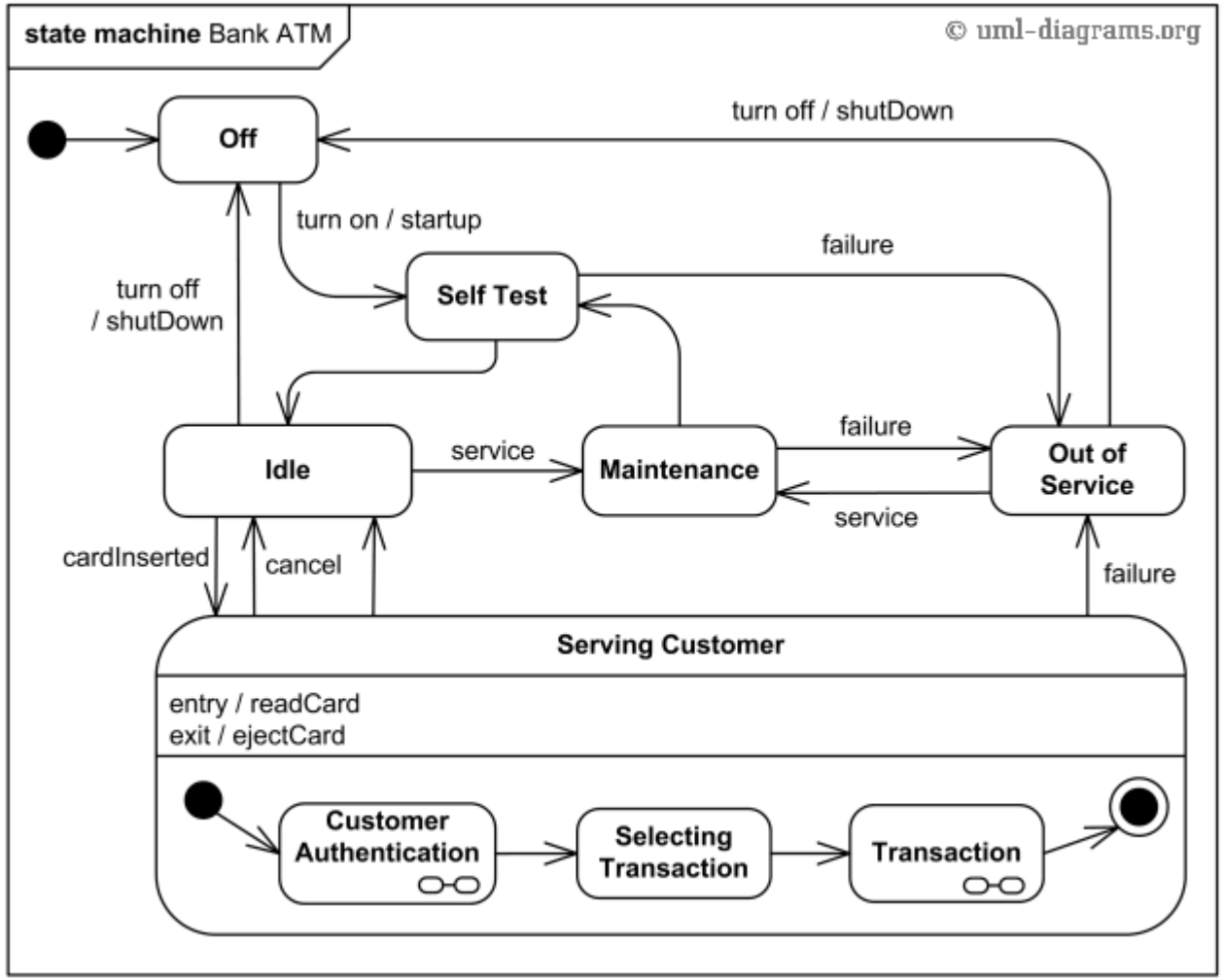
UML State-Machine Diagram

Digital Watch

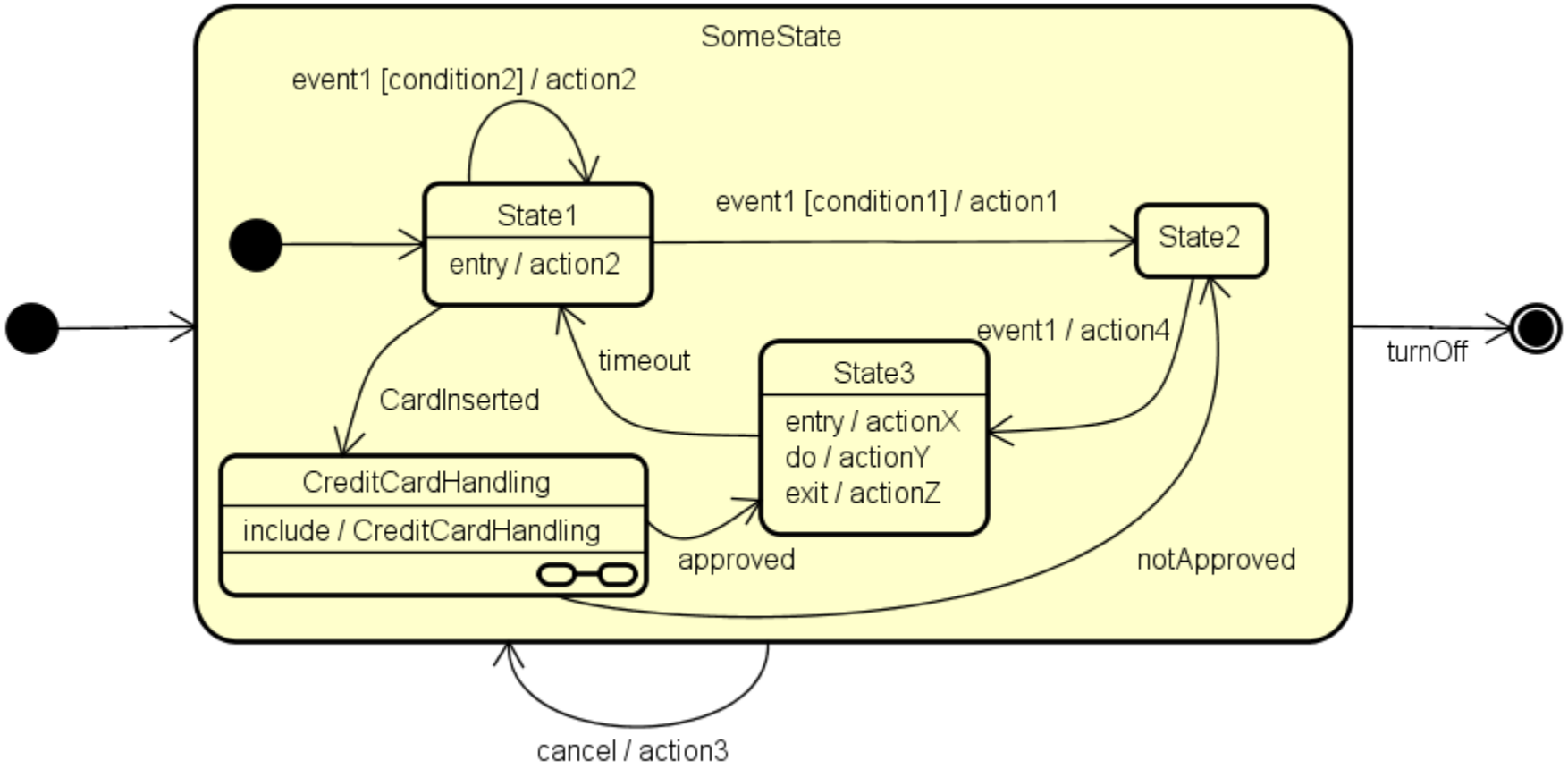
Notice: The function of the two buttons depends on the state of the Watch



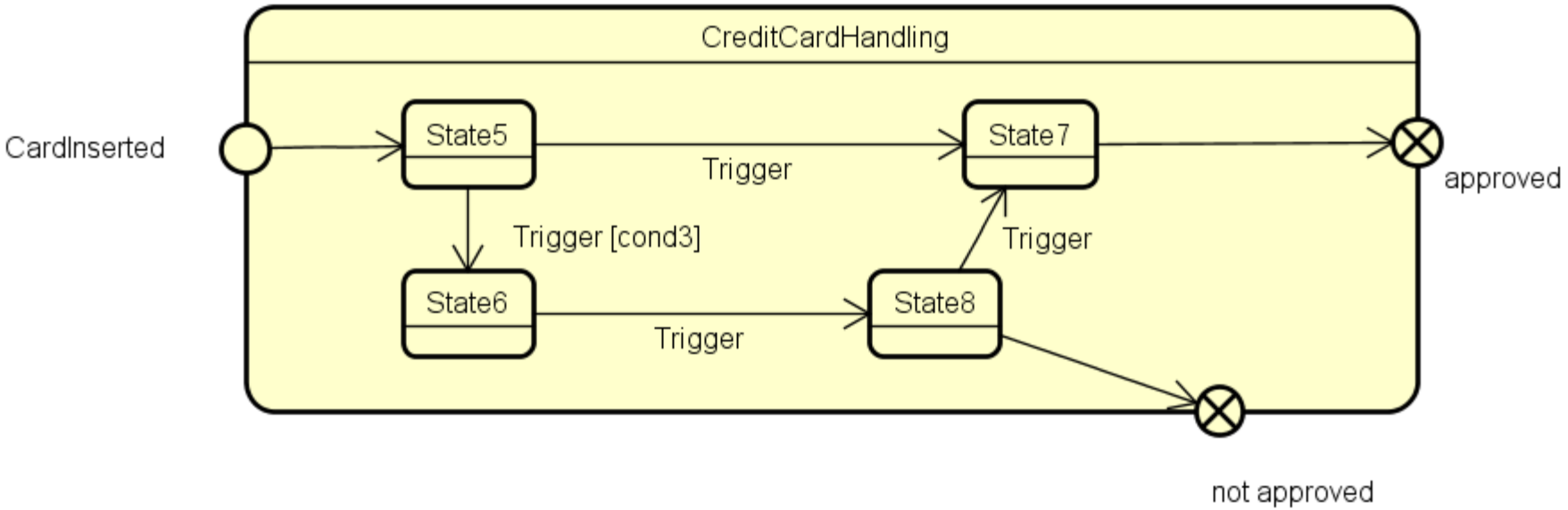
UML State-Machine



UML State-Machine Diagrams



UML State-Machine Diagrams



Calculator Exercise Part I - Mandatory

Design a state machine for this very simple calculator with auto turn-off after 5 minutes without user inputs

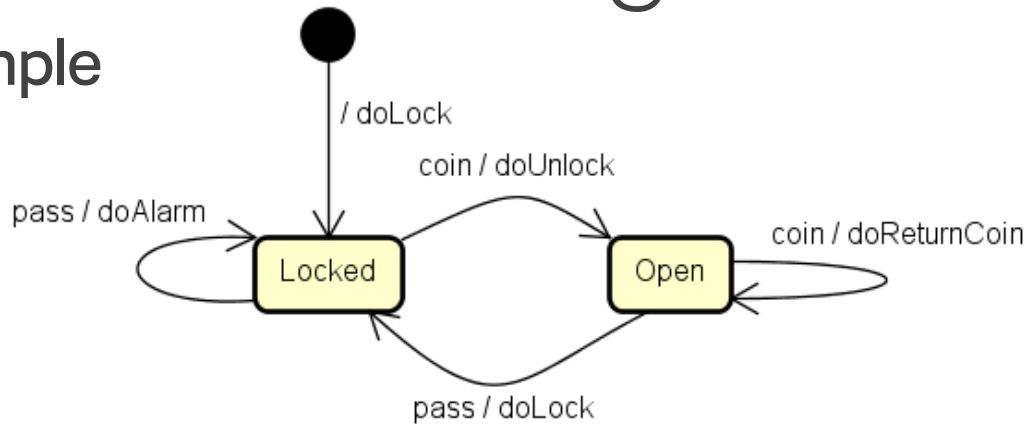
- What kind of events can it be given?
- What kind of actions are needed?
- What states can the calculator be in?

Document it with a UML State-Machine-diagram in Astah

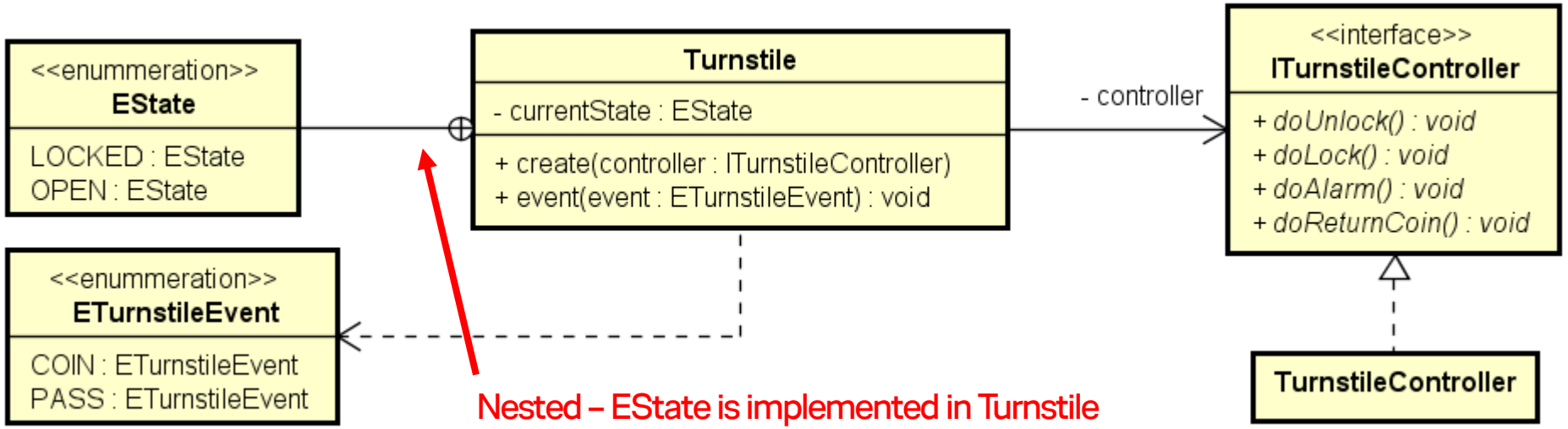
Remember division by zero, overflow etc.!

UML State-Machine Diagram

Back to Turnstile example



Implementation with nested switch-cases:



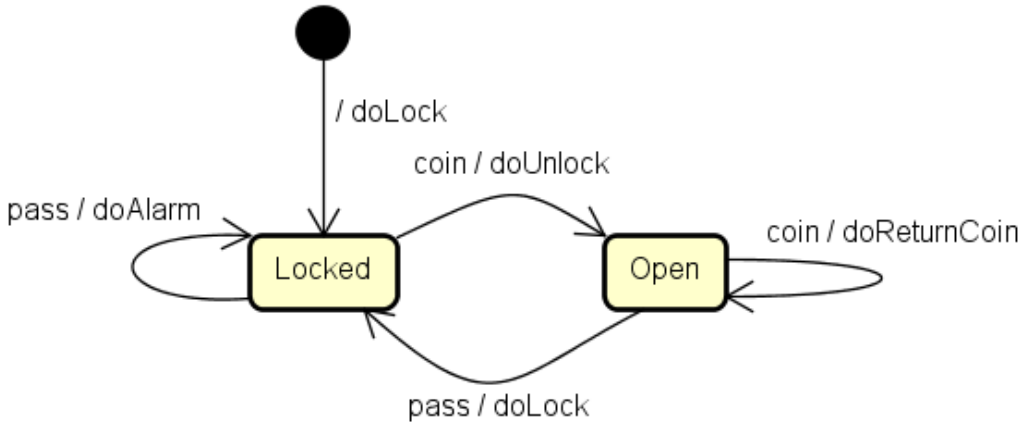
UML State-Machine Diagram

Turnstile example – Nested switch-case

```
public class Turnstile {
    private EState currentState = EState.LOCKED;
    private ITurnstilecontroller controller;

    public Turnstile(ITurnstilecontroller controller) {
        this.controller = controller;
        controller.doLock();
    }

    private enum EState {
        OPEN,
        LOCKED
    }
```

UML State-Machine Dia

Turnstile example – Nested switch-case

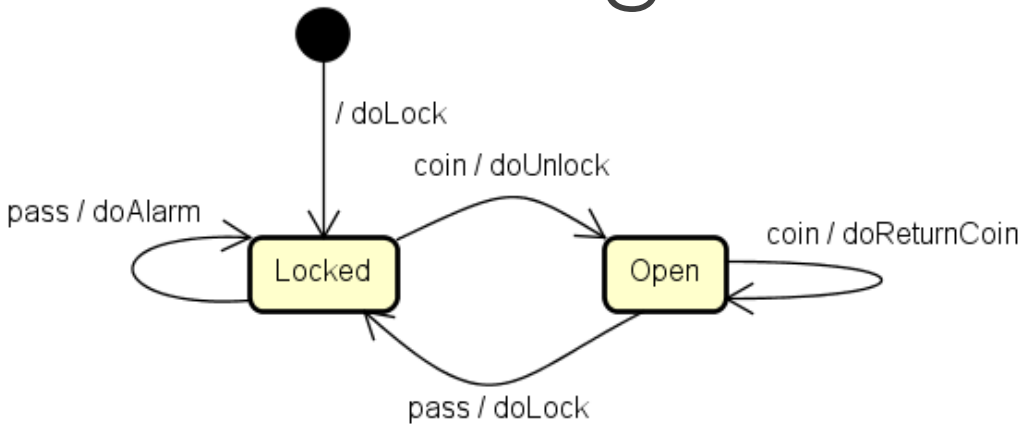


Can you imagine to implement a state machine with 10 states and 5 events as nested switch-cases?

```
public void event(ETurnstileEvent event) {
    switch (currentState) {
    case LOCKED:
        switch (event) {
        case COIN:
        controller.doUnlock();
        currentState = EState.OPEN;
        break;

        case PASS:
        controller.doAlarm();
        break;
        }
    break;

    case OPEN:
        switch (event) {
        case COIN:
        controller.doReturnCoin();
        break;

        case PASS:
        controller.doLock();
        currentState = EState.LOCKED;
        break;
        }
    break;
    }
}
```

UML State-Machine Diagram

Turnstile example

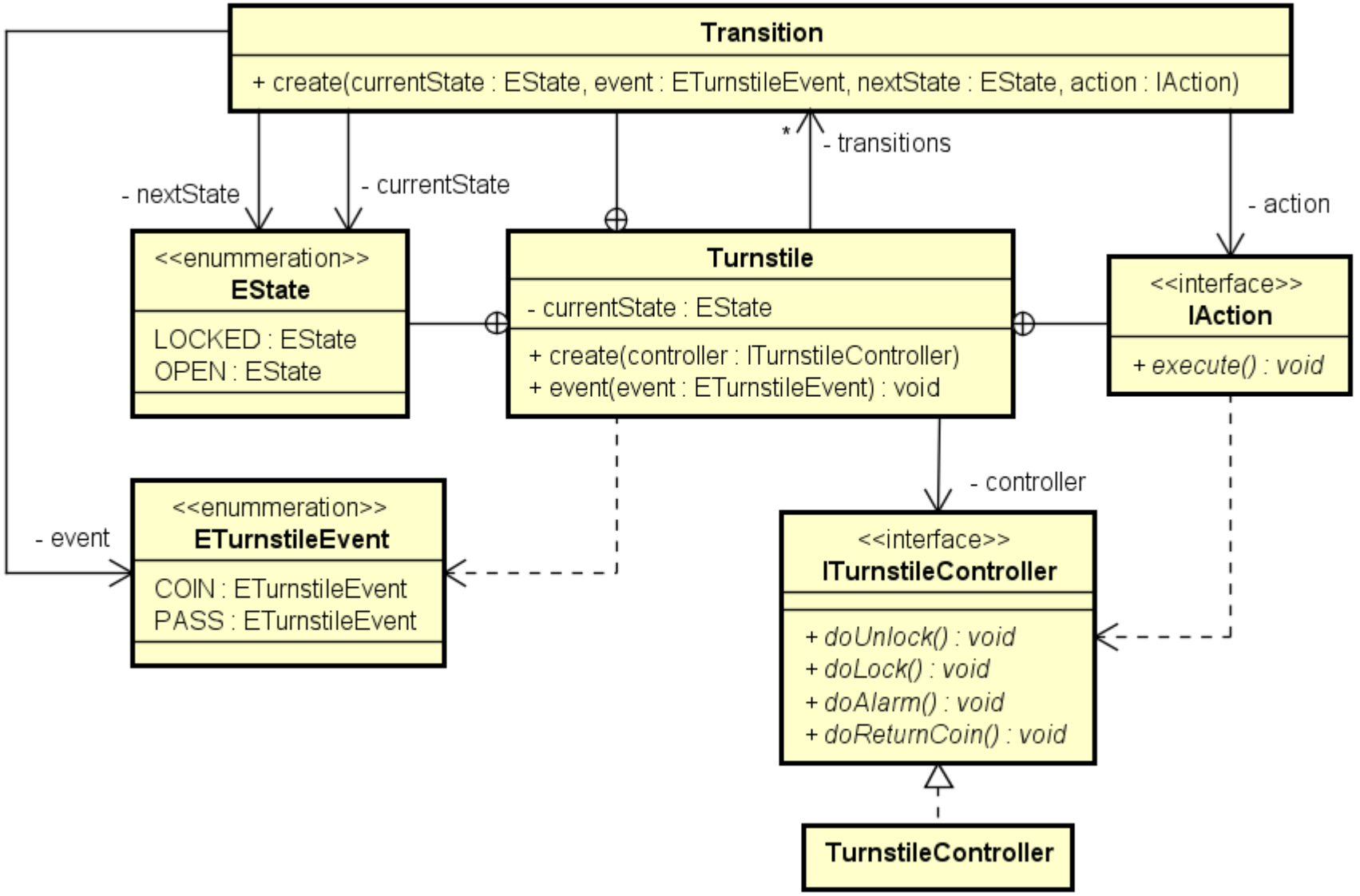


Implementation with state transition table:

Current State	Event	New State	Action
LOCKED	COIN	OPEN	unlock()
LOCKED	PASS	LOCKED	alarm()
OPEN	COIN	OPEN	returnCoin()
OPEN	PASS	LOCKED	lock()

UML State-Machine Diagram

Turnstile example – State transition table



UML State-Machine Diagram

Turnstile example – State transition table


```
public class Turnstile {
    private EState currentState = EState.LOCKED;
    private ITurnstilecontroller controller;

    private Vector<Transition> transitions = new Vector<Transition>();

    public Turnstile(ITurnstilecontroller controller) {
        this.controller = controller;
        controller.doLock();

        addTransition(EState.LOCKED, ETurnstileEvent.COIN, EState.OPEN, doUnlock());
        addTransition(EState.LOCKED, ETurnstileEvent.PASS, EState.LOCKED, doAlarm());
        addTransition(EState.OPEN, ETurnstileEvent.COIN, EState.OPEN, doReturnCoin());
        addTransition(EState.OPEN, ETurnstileEvent.PASS, EState.LOCKED, doLock());
}
```

UML State-Machine Diagram

Turnstile example – State transition table

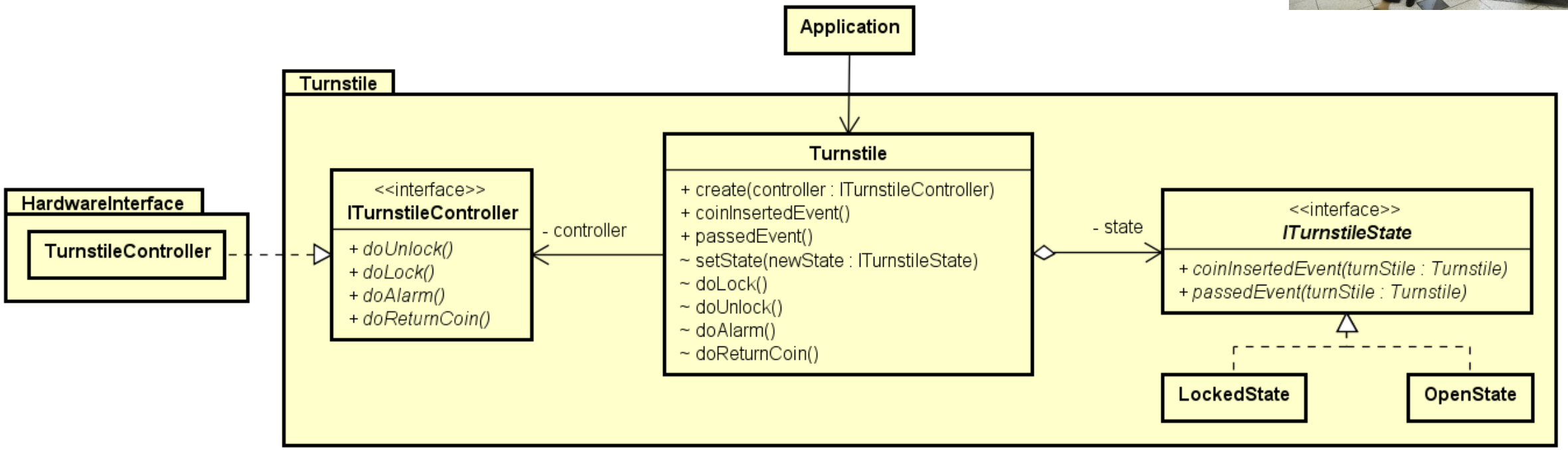
The Engine

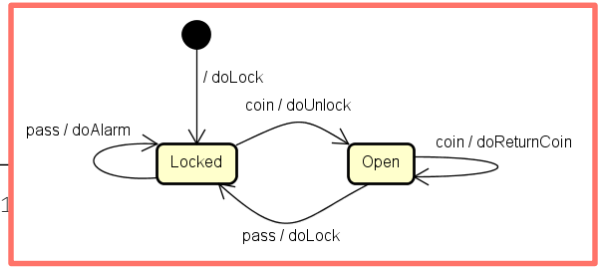
```
    public void event(ETurnstileEvent event) {
        for (Transition transiton : transitions) {
            if ((currentState == transiton.currentState) && (event == transiton.event)) {
                currentState = transiton.nextState;
                transiton.action.execute();
            }
        }
    }
```

Rest of the implementation can be found in StudyNet

State-Pattern

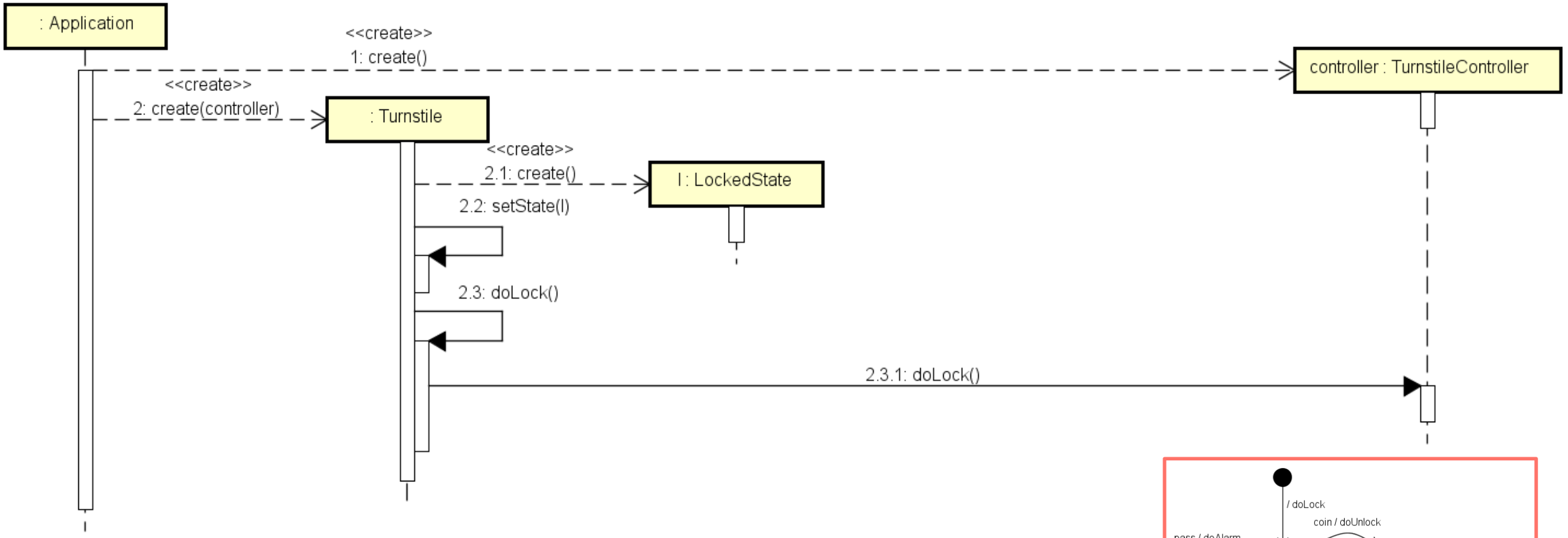
Turnstile example – State-Pattern

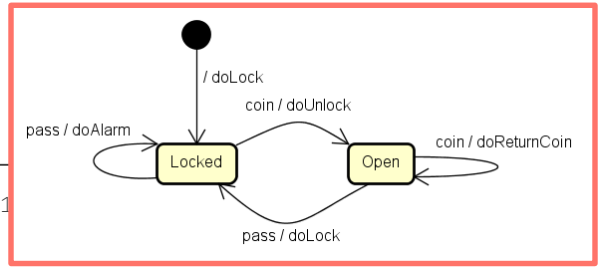




State-Pattern

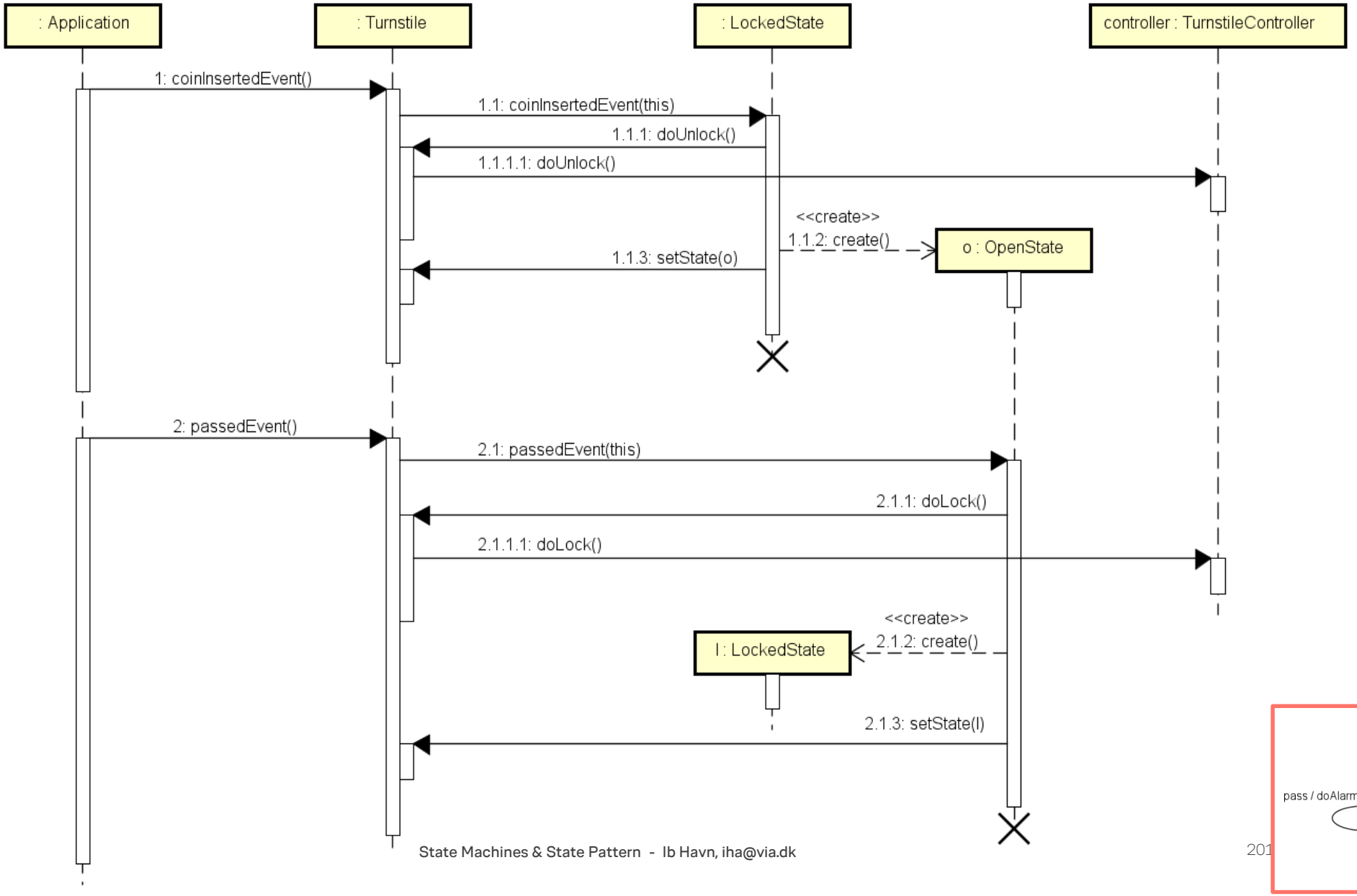
Turnstile example Init – State-Pattern

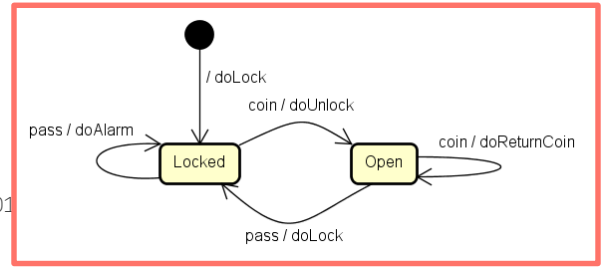




State-Pattern

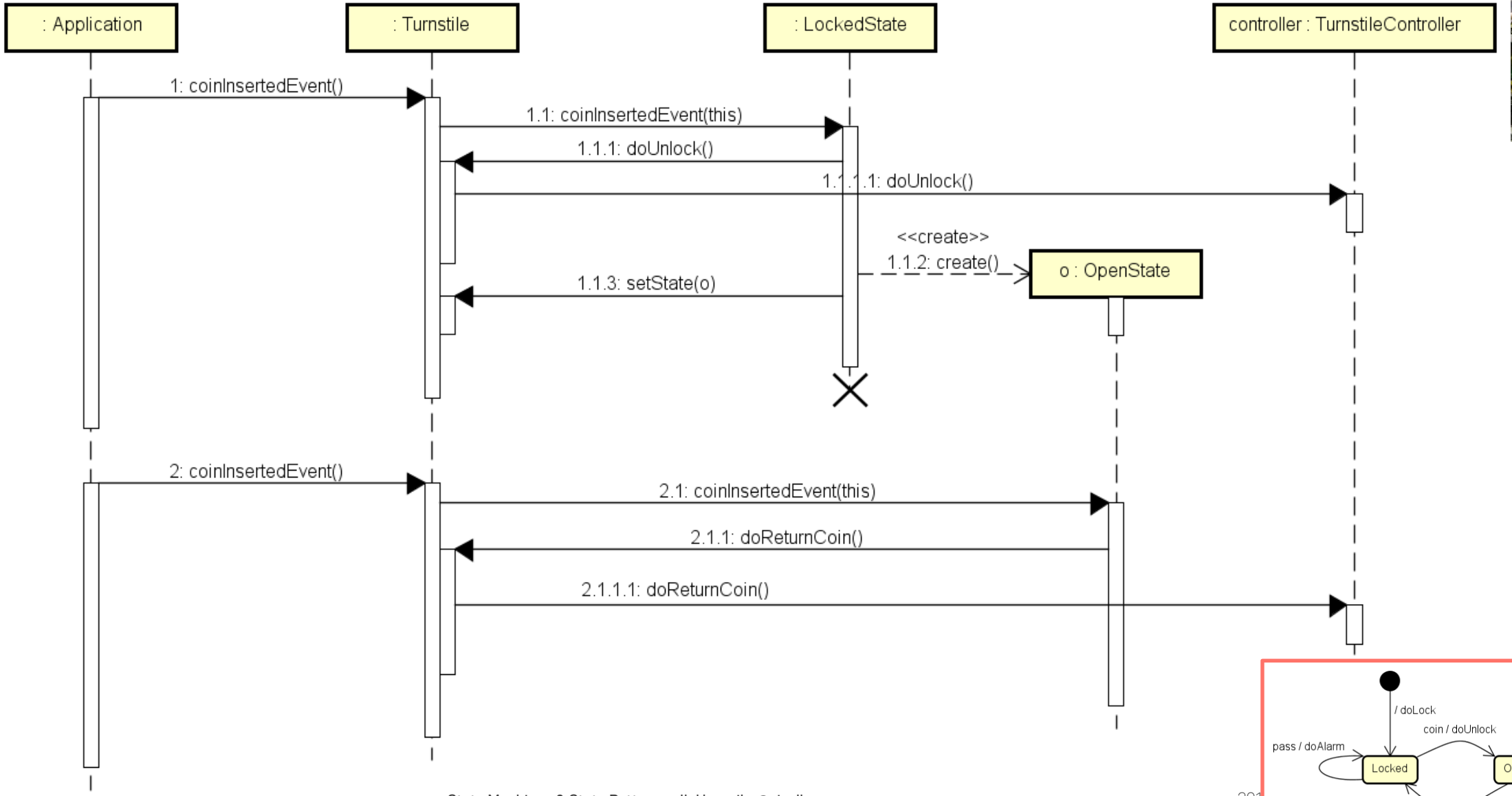
Turnstile example Use – State-Pattern

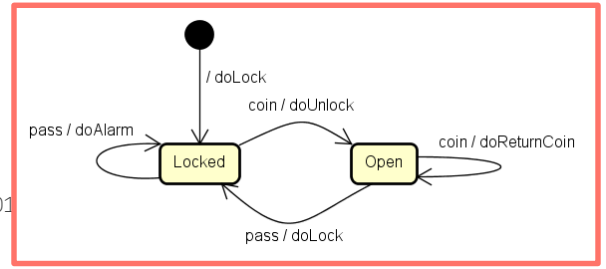




State-Pattern

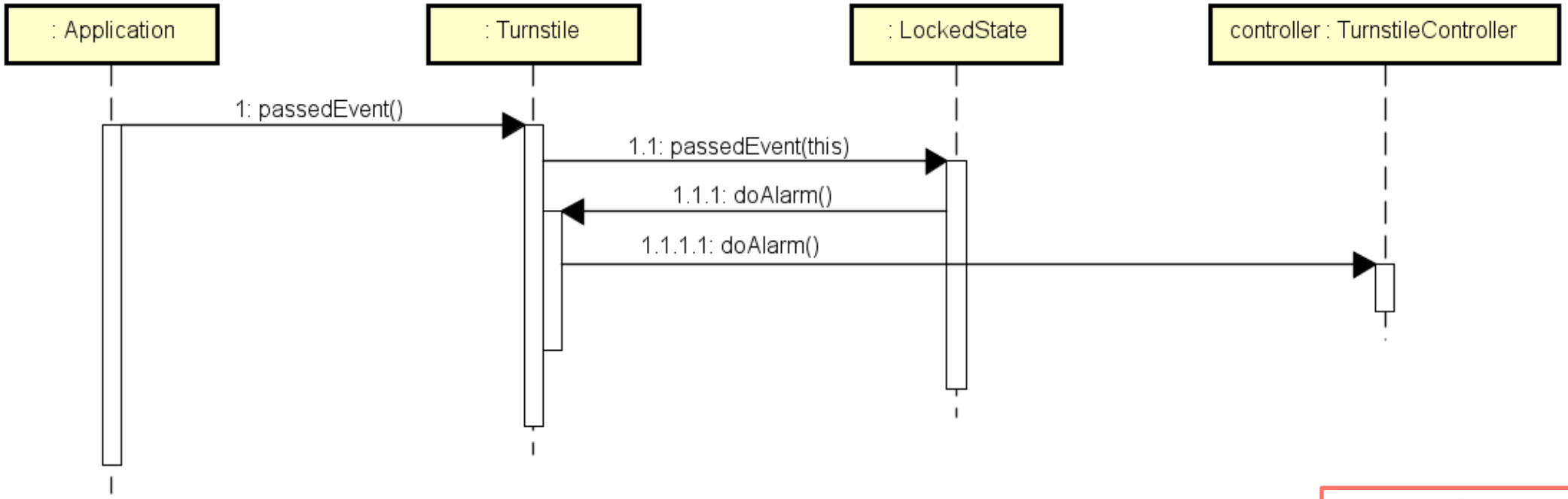
Turnstile example Use – State-Pattern

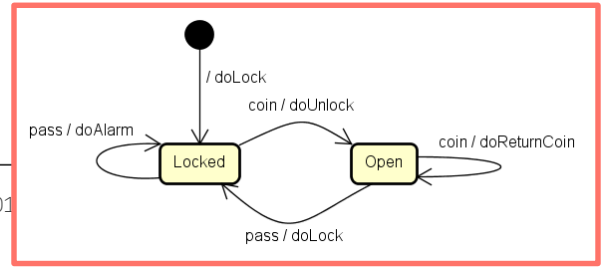




State-Pattern

Turnstile example Use – State-Pattern

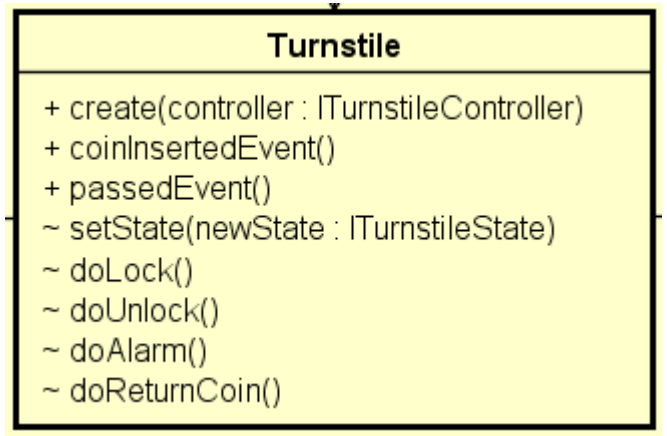


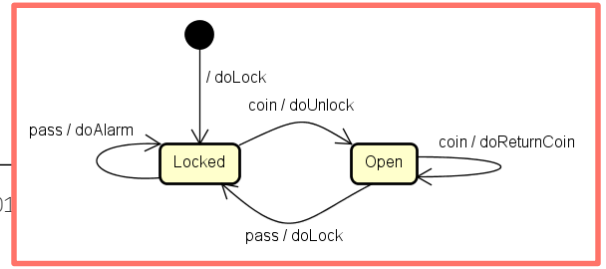


UML State-Machine Diagram

Turnstile example – State-Pattern

```
public class Turnstile {
    private ITurnstileState state = null;

    private ITurnstilecontroller controller = null;

    public Turnstile(ITurnstilecontroller controller) {
        this.controller = controller;
        setState(new LockedState());
        doLock();
    }
```

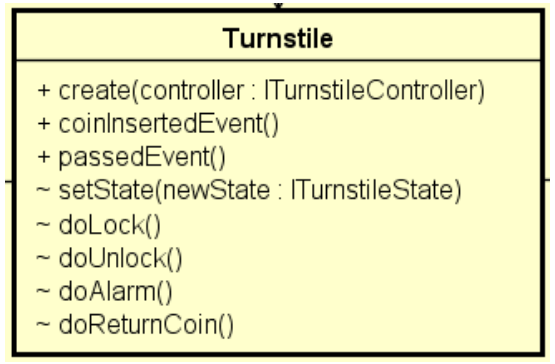




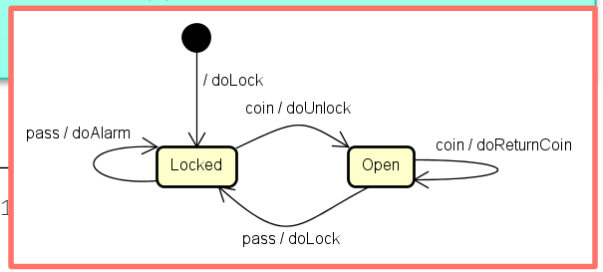
UML State-Machine Diagram

Turnstile example – State-Pattern

```
// Handle external events
public void coinInsertedEvent() {
    state.coinInsertedEvent(this);
}

public void passedEvent() {
    state.passedEvent(this);
}
```



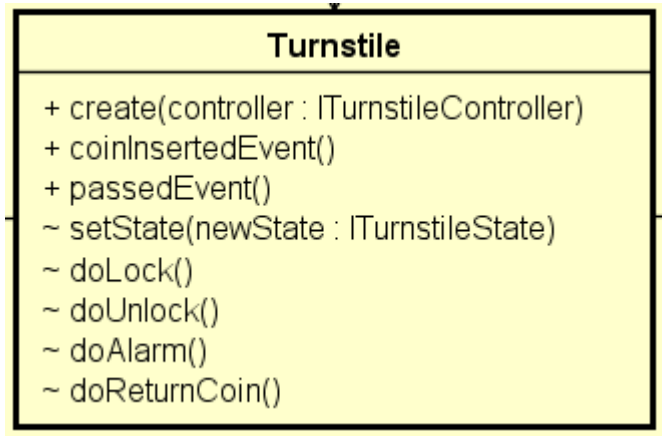

```
// Actions
protected void doLock() {
    controller.doLock();
}

protected void doUnlock() {
    controller.doUnlock();
}

protected void doAlarm() {
    controller.doAlarm();
}

protected void doReturnCoin() {
    controller.doReturnCoin();
}
```

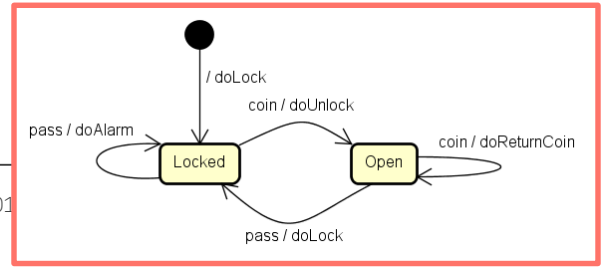


UML State-Machine Diagram

Turnstile example – State-Pattern

```
// Set state
void setState(ITurnstileState newState)
{
    state = newState;
}
```

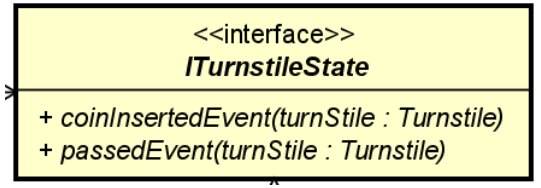


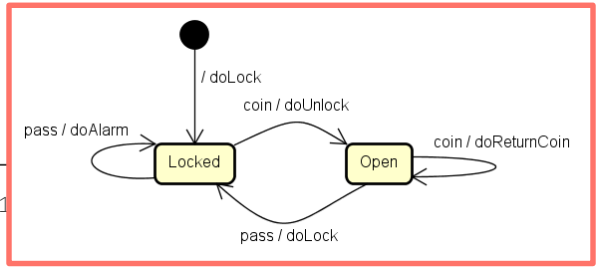


UML State-Machine Diagram

Turnstile example – State-Pattern

```
public interface ITurnstileState {
    public void coinInsertedEvent(Turnstile turnstile);
    public void passedEvent(Turnstile turnstile);
}
```

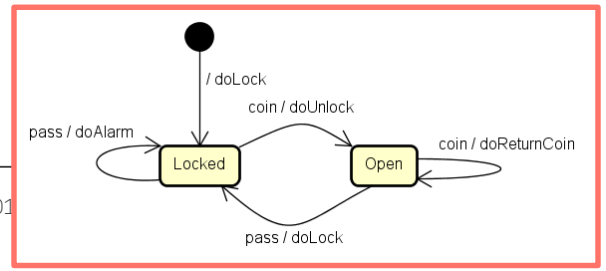




UML State-Machine Diagram

Turnstile example – State-Pattern

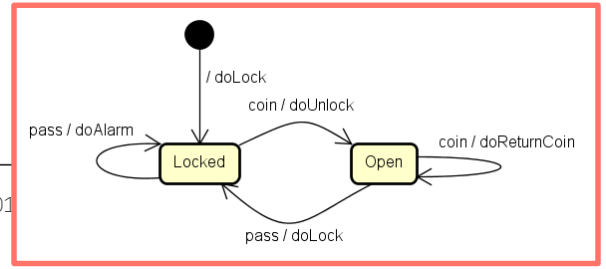
```
public class LockedState implements ITurnstileState {
    @Override
    public void coinInsertedEvent(Turnstile turnstile) {
        turnstile.doUnlock();
        turnstile.setState(new OpenState());
    }

    @Override
        public void passedEvent(Turnstile turnstile) {
        turnstile.doAlarm();
    }
}
```



UML State-Machine Diagram

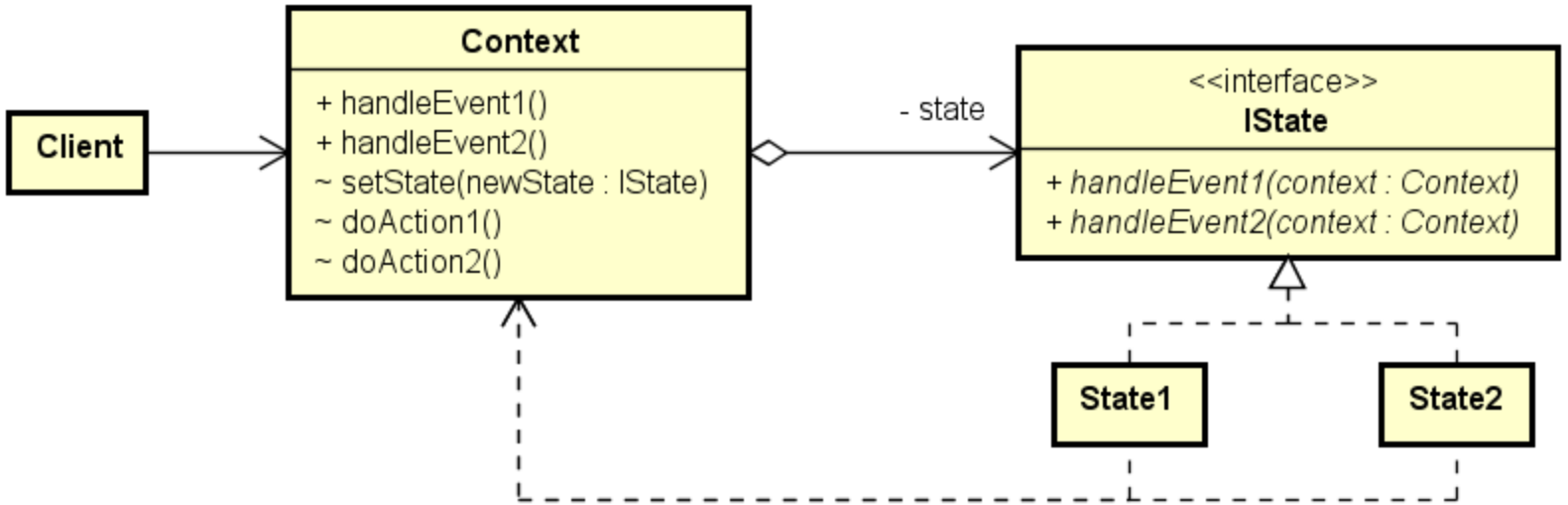
Turnstile example – State-Pattern

```
public class OpenState implements ITurnstileState {
    @Override
    public void coinInsertedEvent(Turnstile turnstile) {
        turnstile.doReturnCoin();
    }

    @Override
    public void passedEvent(Turnstile turnstile) {
        turnstile.doLock();
        turnstile.setState(new LockedState());
    }
}
```



State-Pattern

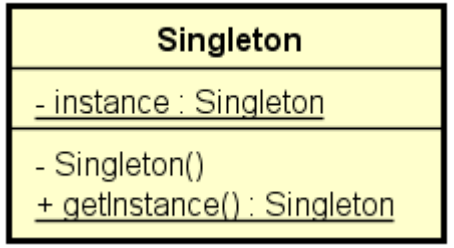
State-Pattern Classic



Singleton Design Pattern

The singleton design pattern solves problems like:

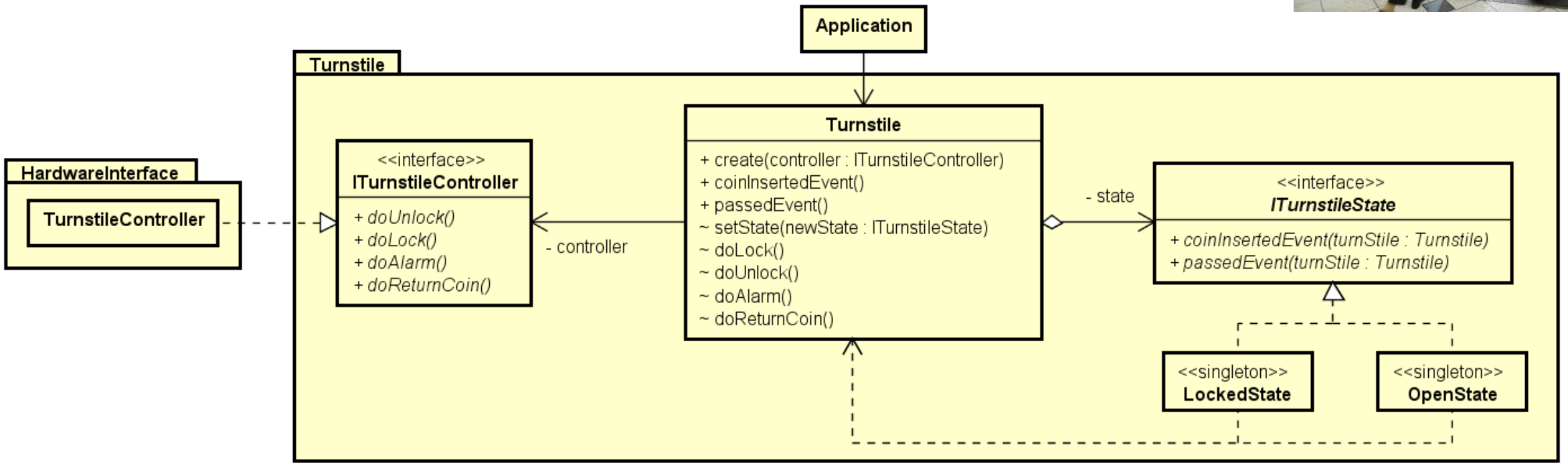
- How can it be ensured that a class has only one instance?
- How can the sole instance of a class be accessed easily?
- How can a class control its instantiation?
- How can the number of instances of a class be restricted?

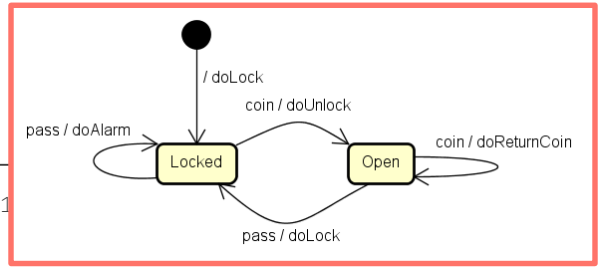



```
public class Singleton {
    private final static Singleton INSTANCE = new Singleton();

    private Singleton() {}

    public static Singleton  getInstance() {
        return INSTANCE;
    }
}
```

State-Pattern

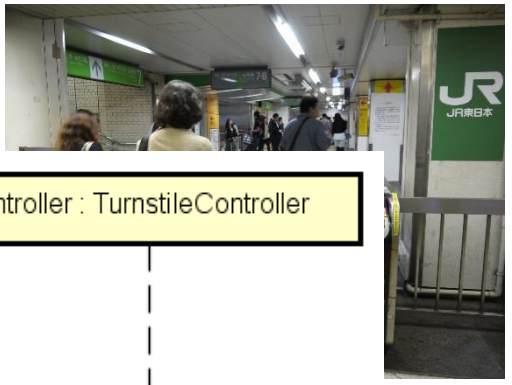
Turnstile example – State-Pattern **Singleton Implementation**





State-Pattern

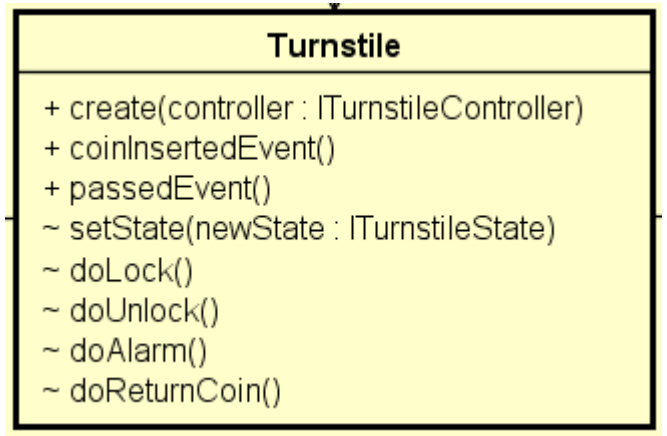
Turnstile example – State-Pattern
Singleton Implementation

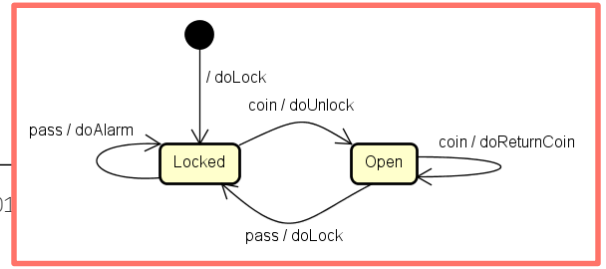


UML State-Machine Diagram

Turnstile example – State-Pattern **Singleton Implementation**

```
public class Turnstile {
    private ITurnstileState state = null;

    private ITurnstilecontroller controller = null;

    public Turnstile(ITurnstilecontroller controller) {
        this.controller = controller;
        setState(LockedState.getInstance());
        doLock();
    }
```

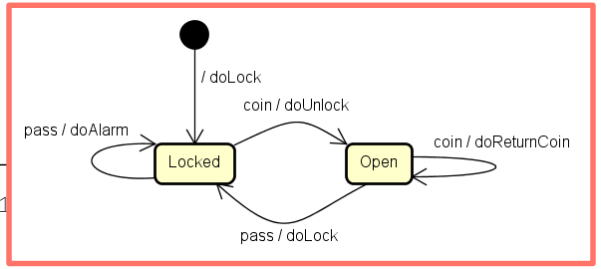




UML State-Machine Diagram

Turnstile example – State-Pattern **Singleton Implementation**

```
public class LockedState implements ITurnstileState {
    private static final LockedState INSTANCE = new LockedState();

    private LockedState() {}

    @Override
    public void coinInsertedEvent(Turnstile turnstile) {
        turnstile.doUnlock();
        turnstile.setState(OpenState.getInstance());
    }

    @Override
    public void passedEvent(Turnstile turnstile) {
        turnstile.doAlarm();
    }

    public static ITurnstileState getInstance() {
        return INSTANCE;
    }
}
```



Calculator Exercise Part II - Mandatory

Design a state machine for this very simple calculator with auto turn-off after 5 minutes without user inputs

- What kind of events can it be given?
- What kind of actions are needed?
- What states can the calculator be in?

Document it with a UML Class-, State machine- and Sequence-diagrams (remember descriptions to all diagrams)

- Use the State-Pattern

Remember division by zero, overflow etc.!

Are the SOLID principles violated?

Implement it in Java