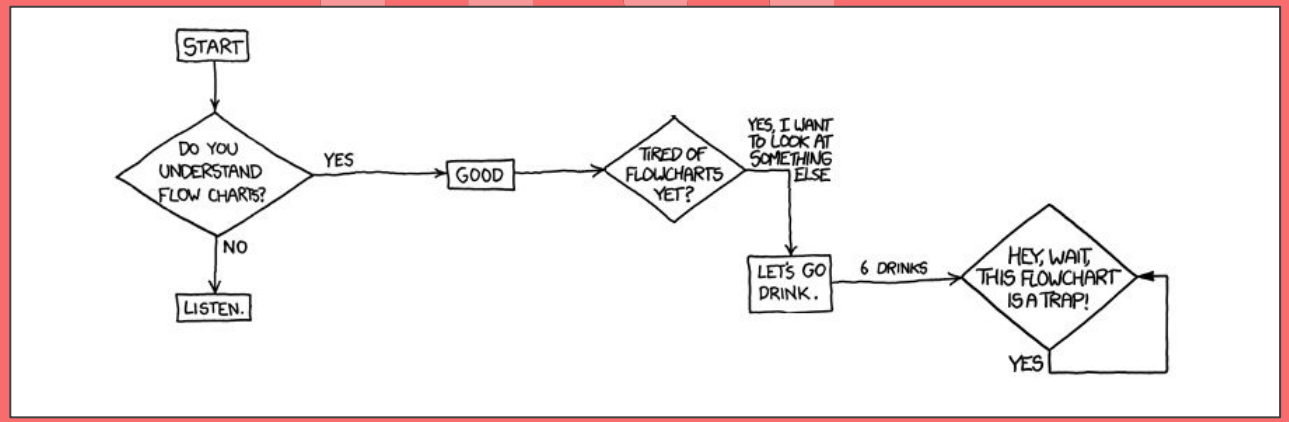
WILD CODE SCHOOL

Getting into the Flow with Business Process Management



WILD CODE SCHOOL

- What is State and why do we need it?
- Modelling State in Java - 1st try
- Modelling State with Design Patterns
- Modelling State with State Machines
- Modelling State with … Rule Engines?
- Modelling State with … Business Process Engines?

Use Cases for “State”

- **Ordering from online (E-Commerce) Sites**

  When you order some item, the order goes through different states like — *Ordered, Packed, Shipped, Cancelled, Delivered, Payed, Refunded* etc.

- **Managing Events and Actions**

  Trip booking also may go through states like — *Trip Requested, Driver Assigned, Customer Cancelled, Driver Cancelled, Customer Waiting, Driver Arrived, Trip Started, Payment Done, Trip Done, Rating Done* etc.

- **Onboarding Customers, Employees, Members**

  Onboarding People typically involves identification, authorization, roles and privilege assignments, etc.

- **ETL jobs, Batchjobs and automated tasks are configured around states**

  Batch jobs have different stages of processing. These stages can succeed or fail.

- **CI/CD Pipelines are modelled around workflow of states of a deployment job**

https://medium.datadriveninvestor.com/state-machine-design-pattern-why-how-example-through-spring-state-machine-part-1-f13872d68c2d

States in Object Oriented Programming

An object is an entity that has states and behaviors.

Usually an object is defined to have two characteristics: State and behavior

- **State** is defined through variables or properties and described **what** the **object is**.
- **Behavior** is usually defined through functions and describes **what** the **object does**.

Most of the times, the term **state represents both attribute and state** of an object. *But state and attribute are not same*.

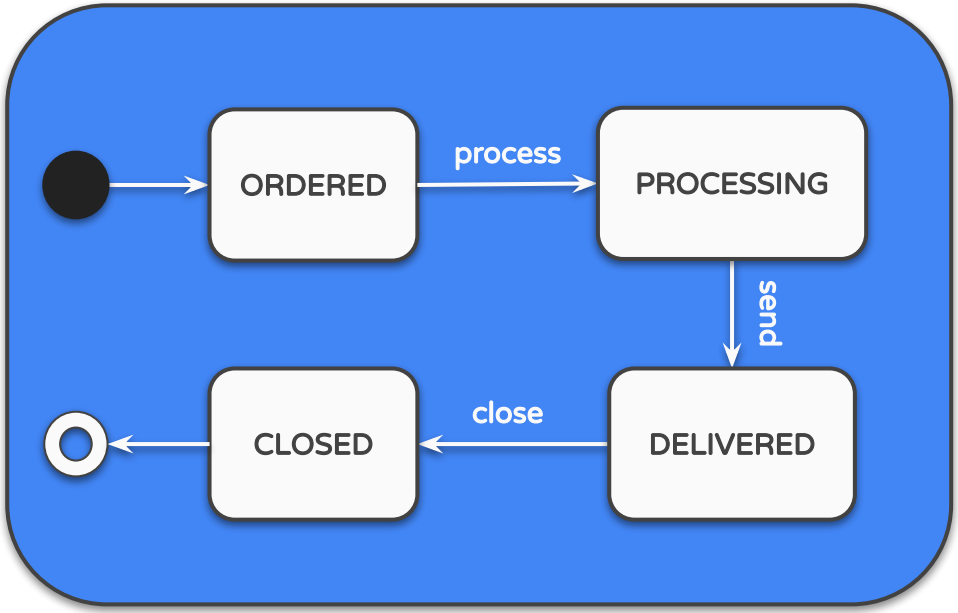
- **State is mutable, Attributes are not.**

```
class Dog {

  // Attributes
  color;
  age;

  // State
  isThirsty;

  // Behavior
  run();
  eat();

}
```

(Remember DDD Value Objects? They have no states and only attributes and are immutable)

https://www.jawahar.tech/blog/oops-attribute-vs-state

Delivering Products modelled with States



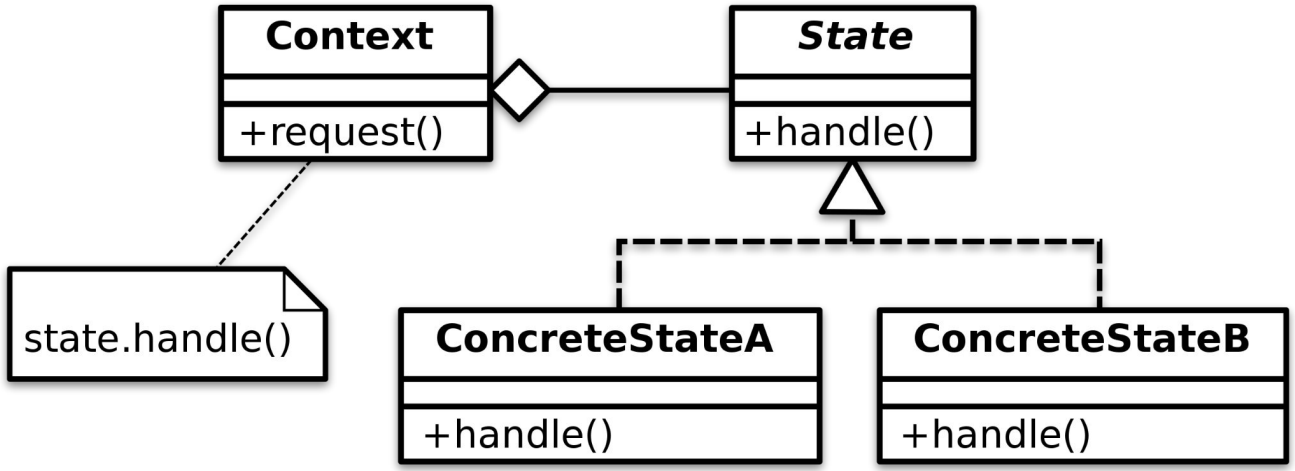
Modelling State - Common Solution

Model state within an object by creating an **instance variable to hold** the state values and writing **conditional code** within a method to handle various states.

```
public void calculateNextState(String action) {
   switch (action.toLowerCase()) {
       case "process": {
           if (state.equals(OrderState.ORDERED)) {
               if (getPrice().intValue() > 0) {
                   if (getQuantity().intValue() > 0) {
                       state = OrderState.PROCESSING;
                   } else {
                       throw new IllegalStateException("Quantity must be positive.");
                   }
               } else {
                   throw new IllegalStateException("Price must be positive.");
               }
           }
       }
       case "send": {
           if (state.equals(OrderState.PROCESSING)) {
               if (available) {
                   state = OrderState.DELIVERED;
                   shippingDate = LocalDate.now();
               } else {
                   throw new IllegalStateException("Cannot be processed, Product not available.");
               }
           }
       }
       case "close": {
           // [...]
       }
   }
}
```

State Design Pattern

The State Pattern **allows** an object to **alter its behaviour** when its internal **state changes**.
The object will **appear to change its class**.



Wikimedia, State Design Pattern UML Class Diagram

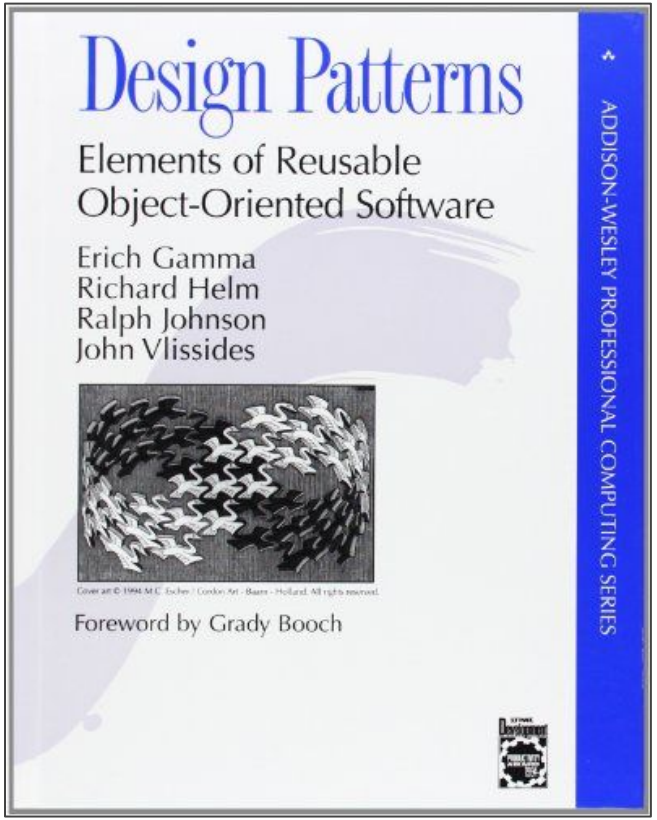
Modelling State - Enum State Pattern

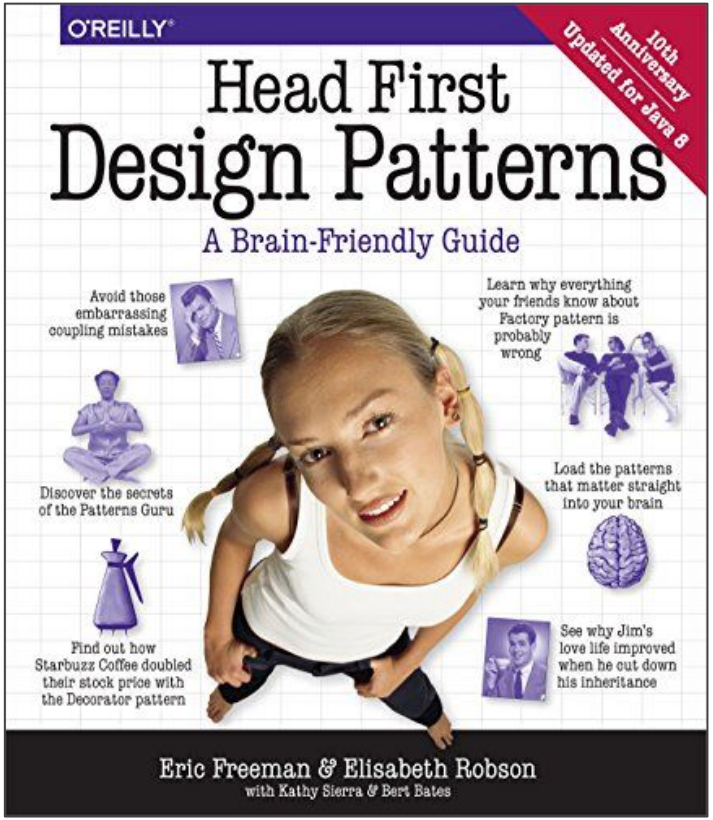
```
private OrderState state = OrderState.Ordered;

public void calculateNextState(String action) {
   state = state.handle(this, action);
}
```

d

```
public enum OrderState {

   Ordered {
       @Override
       public OrderState handle(Order order, String action) {
           if (!action.equals("process")) {
               return this;
           }
           if (order.getPrice().intValue() > 0) {
               if (order.getQuantity().intValue() > 0) {
                   return Processing;
               } else {
                   throw new IllegalStateException("Quantity must be positive.");
               }
           } else {
               throw new IllegalStateException("Price must be positive.");
           }
       }
   },

   Processing {
       @Override
       public OrderState handle(Order order, String action) {
           if (!action.equals("send")) {
               return this;
           }
           // [...]
```

GoF Patterns: State Pattern (vs Strategy vs Template Method)





State Machine: Definition and Classification

State Machine modeling is one of the most traditional patterns in Computer Science. It's one of those design patterns which impacts our daily life through different software. *It's not a coding oriented design pattern*, but it's system oriented, mostly used to **model around business use cases**.

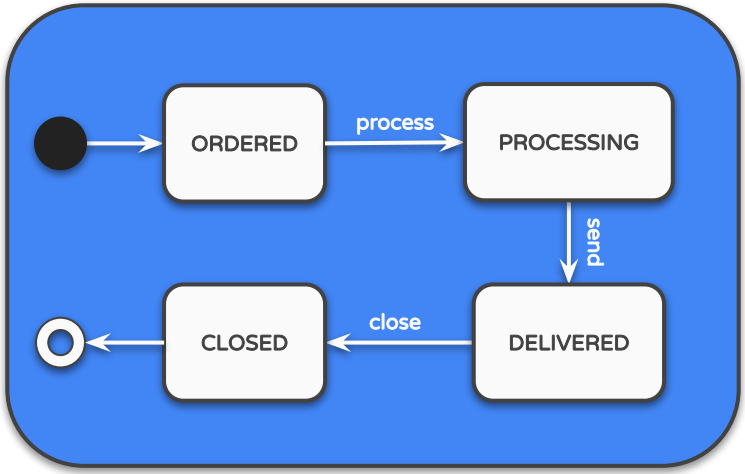
State machine allow to break down **complex tasks** into a **chain of independent smaller tasks**, splitting an activity into **multiple smaller stages**. Smaller tasks are **linked** to each other by **events**, **movement from one state to another** is called **transition**.

While transitioning from one state to another, **some action is performed**: actual booking in the back end, generating an invoice, storing user analytics data, recording booking data in a data store, triggering payment when the trip ends, etc.

State Machine: Current State + Some Action => Another State (+ Output)

https://medium.datadriveninvestor.com/state-machine-design-pattern-why-how-example-through-spring-state-machine-part-1-f13872d68c2d

State Machine: State-Transition Table



Current State	Input	Next State
ORDERED	process	PROCESSING
PROCESSING	send	DELIVERED
DELIVERED	close	CLOSED
CLOSED	-	-

Using State Machines: Pros & Cons

Benefits of State Machines

- **No hard coding** conditions in your code. State machine **abstracts all logic** regarding states & transition.
- State machines usually have a **finite number of states** which makes it **easy to analyze transitions** which caused the current condition of a request.
- Developers can just **concentrate on defining actions and preconditions** after a state machine is configured. With proper validation and precondition, state machines **prevent out of order operations**.
- State machines can be **highly maintainable**. Logically action performed during each transition is independent of each other. So corresponding **piece of code can be isolated**.
- Usually, state machines are **stable and less prone to change**. So if current and future use cases are very clear, it becomes **very easy to maintain such systems**.

https://medium.datadriveninvestor.com/state-machine-design-pattern-why-how-example-through-spring-state-machine-part-1-f13872d68c2d

Using State Machines: Pros & Cons

Disadvantages of State Machines

- State machines are **typically synchronous** in nature and **dependant on external triggers** (events).
- The code can become very messy easily. State machines are data driven, depending on different data/input parameter, it is **difficult to execute different transitions** from the same state. So this kind of requirements can cause having **several transitions with messy precondition check**.
- State machine instances **are not thread-safe per default**, if persistence and multi-threading is necessary this **requires a complex setup** (with many aspects to be considered: persistence, immutability, isolation, error handling).
- **Only few resources or communities are available** as state machines are rarely used (see common solutions for state implementation).

https://medium.datadriveninvestor.com/state-machine-design-pattern-why-how-example-through-spring-state-machine-part-1-f13872d68c2d

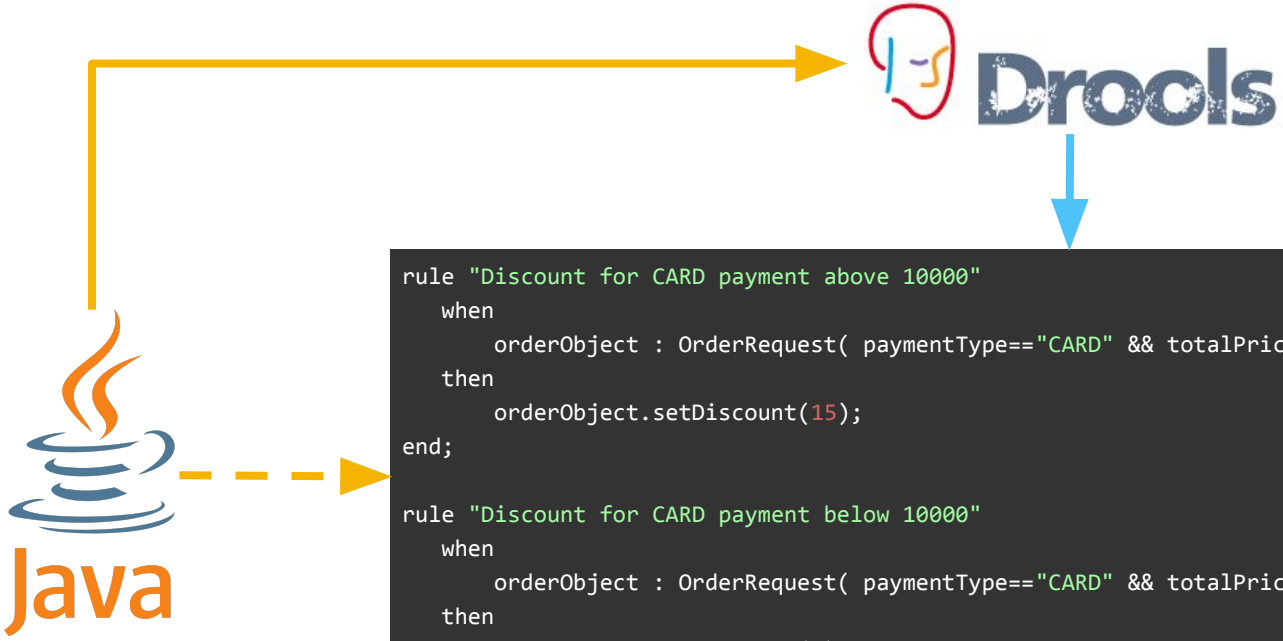
What are Rule Engines?

A **Business Rule Management System** is a software system used to define, deploy, execute, monitor and maintain the **variety and complexity** of **decision logic** that is used by **operational systems within an organization** or enterprise.

The business logic, also referred to as **business rules**, includes **policies, requirements, and conditional statements** that are used to **determine the tactical actions** that take place in applications and systems.

https://en.wikipedia.org/wiki/Business_rule_management_system

What are Rule Engines?

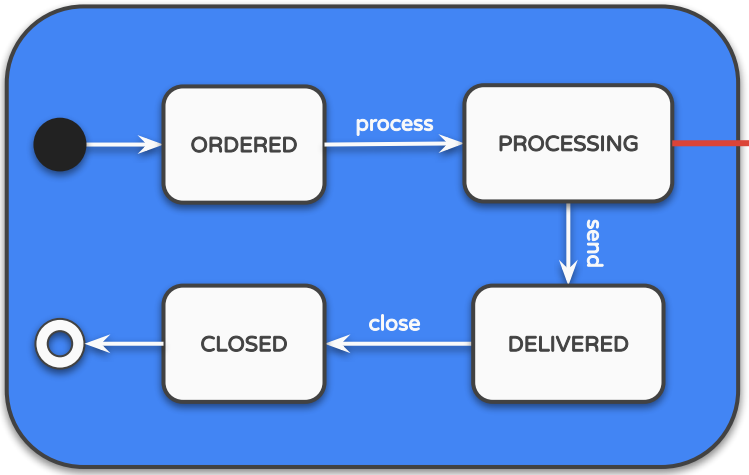



```
rule "Discount for CARD payment above 10000"
   when
       orderObject : OrderRequest( paymentType=="CARD" && totalPrice > 10000 );
   then
       orderObject.setDiscount(15);
end;

rule "Discount for CARD payment below 10000"
   when
       orderObject : OrderRequest( paymentType=="CARD" && totalPrice > 5000 && totalPrice < 10000 );
   then
       orderObject.setDiscount(5);
end;
```

Rule Engines and State Machines

Rule Engines and State Modelling are **complimentary**: If business rules get too complex, using a BRE enhances **maintainability and extendability** a lot (by applying Seperation-of-Concerns).

State transitions can still be handled by a **State Machine**, **Conditions** for action execution are handled by the **Rules Engine**.




```
rule "Discount for CARD payment above 10000"
   when
       [...]
   then
       [...]
end;

rule "Discount for CARD payment below 10000"
   when
       [...]
   then
       [...]
end;
```

From State Machines to Workflow Engines

- **Workflow**: A workflow consists of an **orchestrated and repeatable pattern of activity**, enabled by the systematic organisation of **resources into processes** that [...] process information.

- **State Machine:** A state machine is a **mathematical model of computation**. It's an abstract concept whereby the **machine can have different states**, but at a given time **fulfils only one of them**.

- **Workflow Engine:** A workflow engine **manages and monitors** the **state of activities in a workflow**, and **determines** which new **activity to transition to** according to **defined processes** (workflows).

https://itnext.io/the-concept-of-workflow-engines-c14e8088283

Workflow Engine vs State Machine

- In a workflow engine, **transition** to the **next step** occurs when a **previous action is completed**, whilst a state machine needs an **external event** that will **cause** branching to the **next activity**. In other words, **state machines are event driven and workflow engines are not**.
- Workflow engine supports **sequential pattern** when tasks are executed one after another. The next step in a workflow will not start till the **previous is finished**. For instance, the document will not be signed by the boss till the lawyer signs it. **As a result, workflow engine seems to be rigid and deterministic in its nature. State machine, on the contrary, works asynchronously.** Since the steps in the machine are triggered by certain events/actions, they shouldn't necessarily be performed in a strict order. From this point of view state machines are more flexible.
- **No change** to a State Machine is possible **without breaking the rest. Instead of adding new functionality, often a rebuilding of a new system is necessary.**

https://workflowengine.io/blog/workflow-engine-vs-state-machine/

Workflow Engine vs State Machine

- **State Machines suit only businesses with one-dimensional problems**

State machines are good solutions if your system **is not very complex** (if you are capable of drawing all the possible states as well as the events that will cause transitions to them), eg. protocol implementations or embedded systems.

- **Workflow engines are a good way of managing business processes**

It is the right solution for **task allocation**, **CRM** and other **complex systems**. Its ultimate goal is to **improve business processes** and company's **efficiency**. That is why it perfectly suits for **business process automation**.

https://workflowengine.io/blog/workflow-engine-vs-state-machine/

Workflow Engine vs Rule Engine

Workflow Engine	Rule Engine
A program designed to run workflow instances based on the process model	A program designed to help with complex decision-making
A component specific to a workflow technology	A component of an enterprise application
Inherent driving force in an automated workflow	Works as a pluggable element that could be separated from the application code
Help with carrying out a business process	Help with creating business knowledge
Is based on process design	Is rules-driven

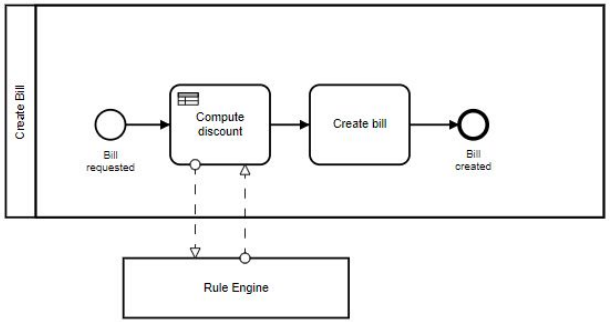
https://kissflow.com/workflow/workflow-engine-business-rule-engine-difference/

Workflow Engine vs Rule Engine

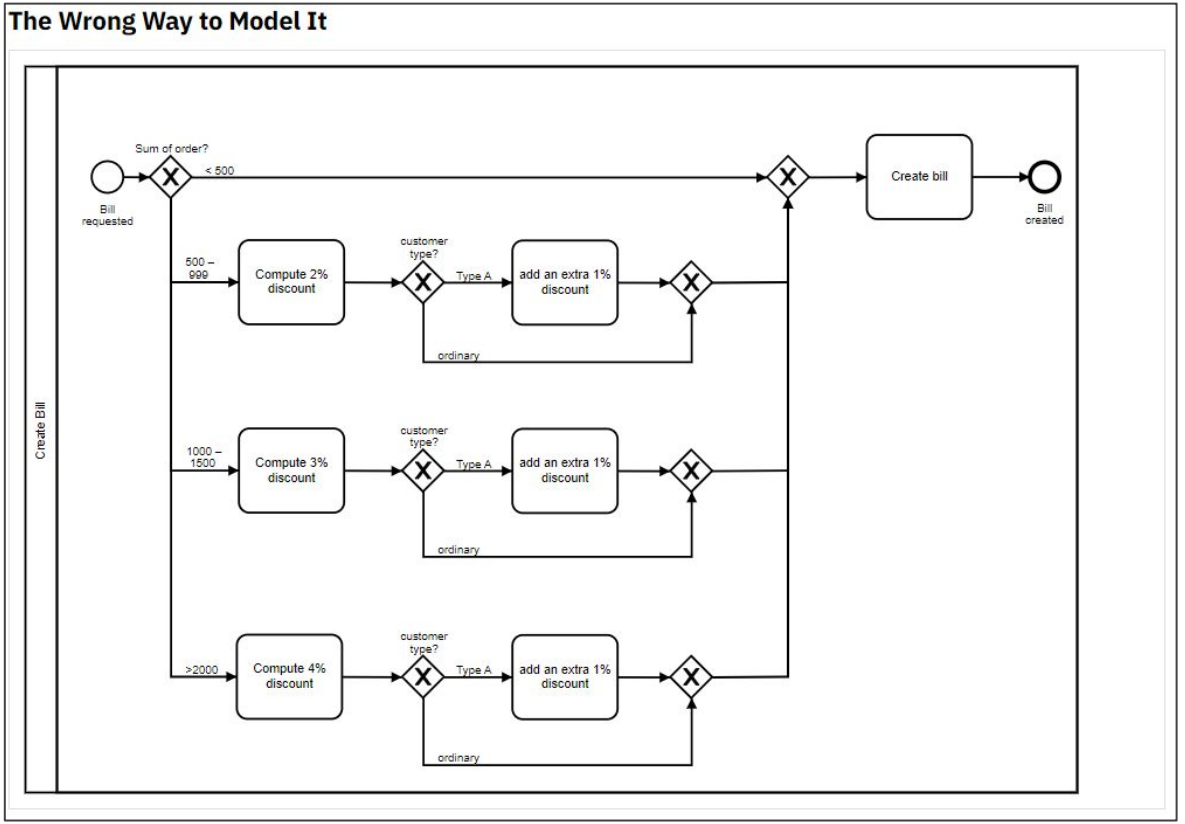
In order to **create the bill**, a **discount needs to be computed**.

The sum of the order and the customer type are the relevant criteria to compute the discount.

The Solution as BPMN 2.0 Diagram



The Wrong Way to Model It



https://camunda.com/bpmn/examples/

Links and other information

State, State Pattern and State Machines

- **Theoretical Background of State and State Machines:** https://en.wikipedia.org/wiki/Automata_theory
- **State Pattern Implementation (Java):** https://howtodoinjava.com/design-patterns/behavioral/state-design-pattern/
- **Spring State Machine Persist Recipe:** https://docs.spring.io/spring-statemachine/docs/3.0.1/reference/#statemachine-examples-persist

Rule Engines

- **Business Rule Management System Drools:** https://www.drools.org/
- **Spring Boot Drools Sample Project:** https://springhow.com/spring-boot-drools/

Workflow Engines

- **Camunda with Spring Boot:** https://docs.camunda.org/get-started/spring-boot/