Lab session 6

Targets

- Implement in Java a Domain model expressed as a UML class diagram
- Add functionality to the entity classes
- Verify functionality with unit tests

Tasks

Implement the domain model

Following the guidelines given in the related theory session (slides available on the virtual campus), implement in Java the Domain model of the CarWorkshop application. Please, follow this workflow:

- Implement all the entities, for each one add just:
  - natural attributes,
  - constructor(s)
  - all the getters, but just the needed setters
  - *hasCode()* and *equals()* over the attributes that define the identity
  - *toString()*
- Implement the value types
- Implement the associations and its maintenance methods. Do it one by one, testing with the specific JUnit test
  - Add to the entities the attributes needed to keep the references (accidental attributes)
  - Add the *getters*, *_getters* and *_setters* as needed (see the slides)
  - Complete the inner classes of Associations.* with the methods *link()* and *unlink()*

Add functionality

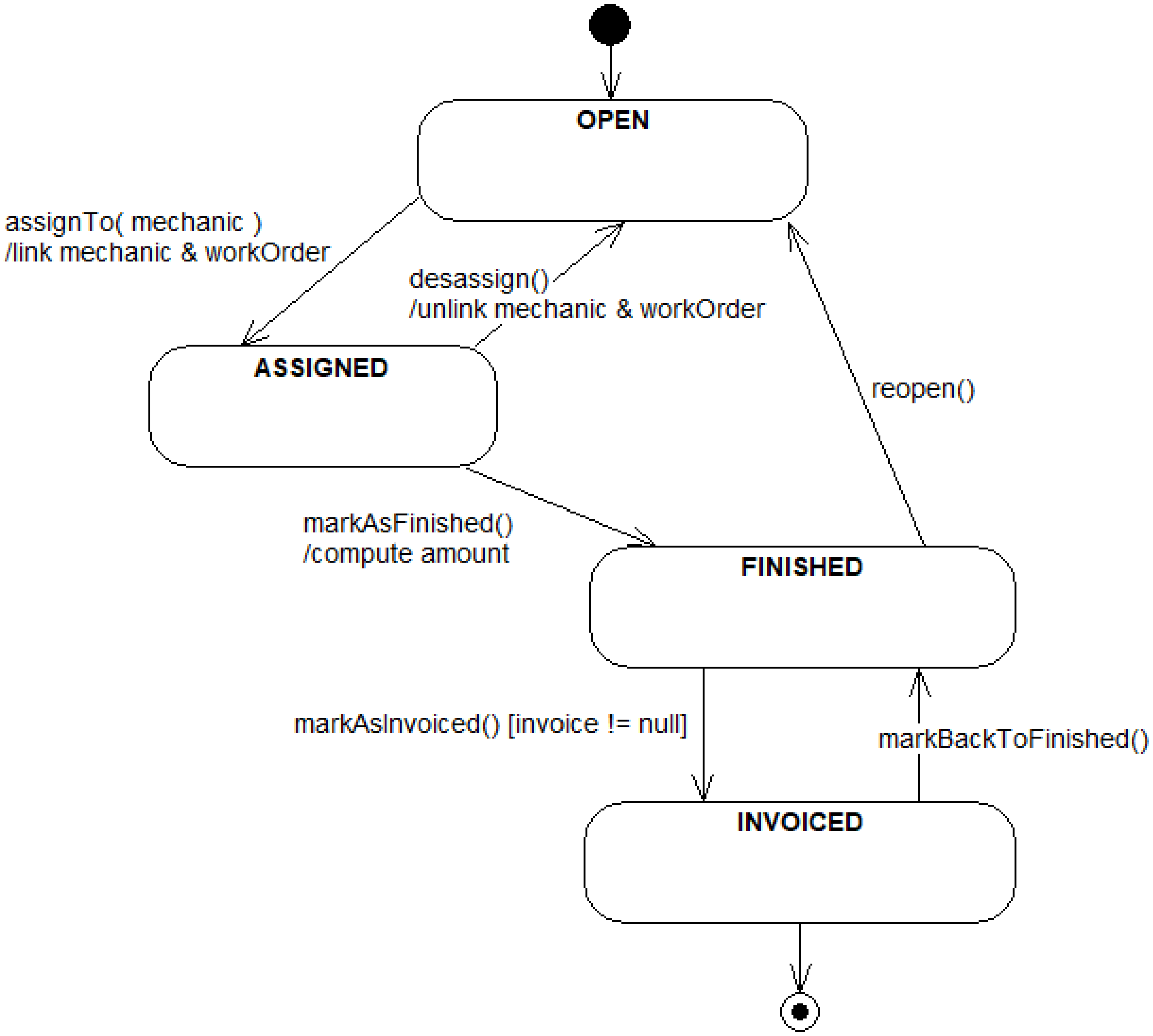
Write Java code to implement this functionality:

- Compute the amount (cost) of a:
  - Substitution
  - All the substitutions of an Intervention
  - The labor cost of an Intervention
  - The total amount of an Intervention (substitutions + labor)
  - Work order
  - Invoice
- Create an invoice for several work orders
- Allow the transition among states of an invoice and a work order. Use for this the states diagram shown below. Follow the Junit tests.

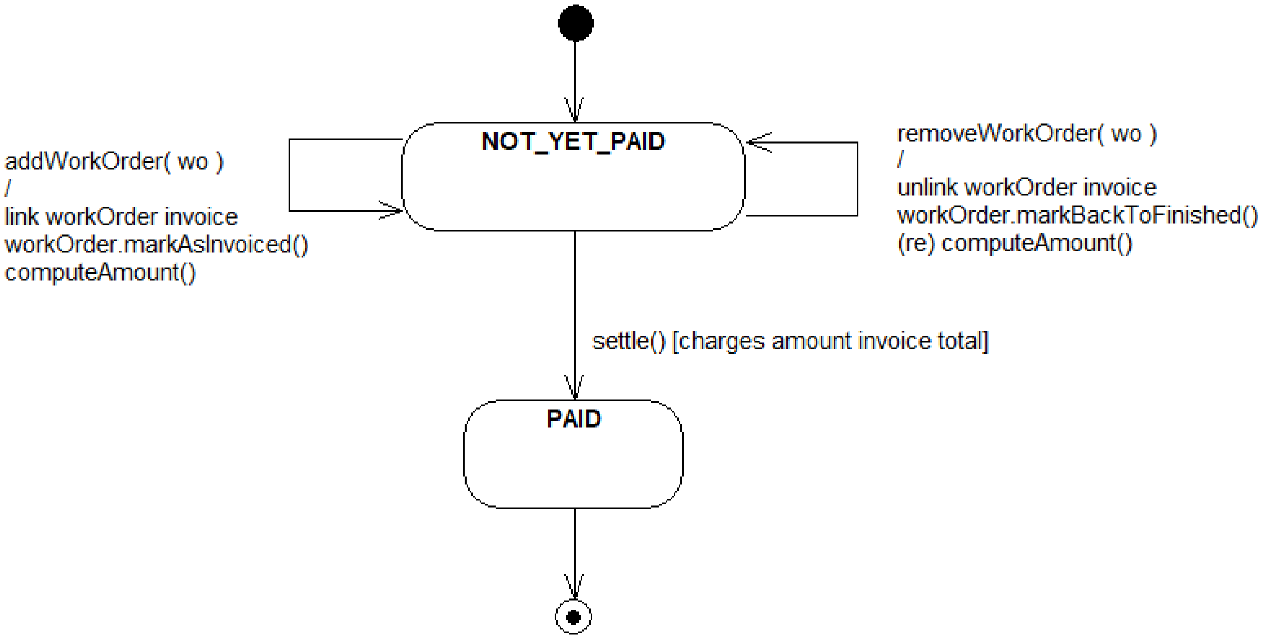
Start from the skeleton provided for this session. Add the domain model classes in the package `uo.ri.cws.domain`.

The work will be finished when all the tests show green.

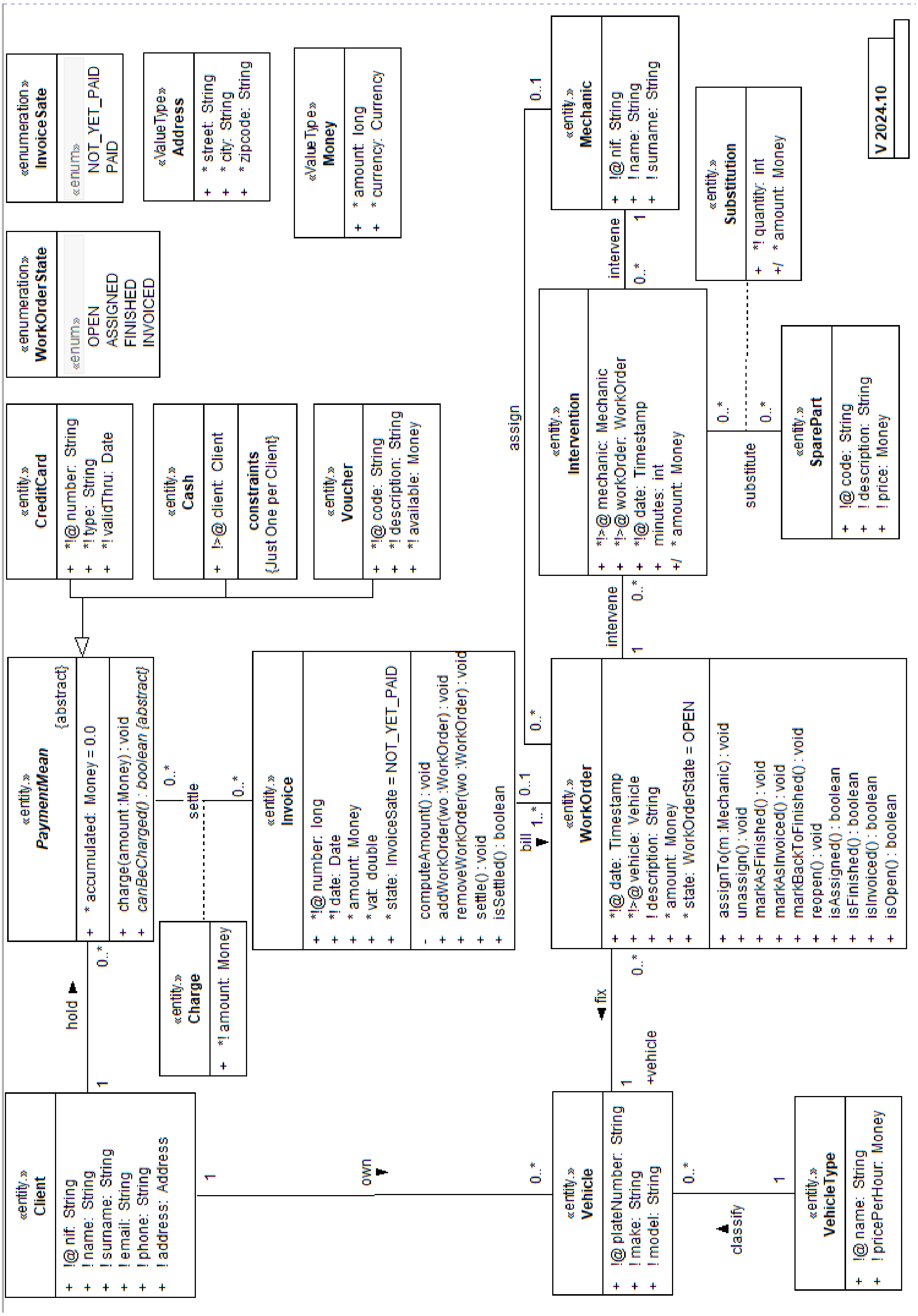
State diagram of a work order



State diagram of an invoice



Domain model diagram


«enumeration»
InvoiceSate
«enum»
NOT_YET_PAID
PAID
«ValueType»
Address
+ * street: String
+ * city: String
+ * zipcode: String
«ValueType»
Money
+ * amount: long
+ * currency: Currency
«enumeration»
WorkOrderState
«enum»
OPEN
ASSIGNED
FINISHED
INVOICED
«entity»
CreditCard
+ *!@ number: String
+ *! type: String
+ *! validThru: Date
«entity»
Cash
+ !>@ client: Client
constraints
{Just One per Client}
«entity»
Voucher
+ *!@ code: String
+ *! description: String
+ *! available: Money
«entity»
PaymentMean
{abstract}
+ * accumulated: Money = 0.0
+ charge(amount :Money) : void
+ canBeCharged() : boolean {abstract}
«entity»
Charge
+ *! amount: Money
«entity»
Invoice
+ *!@ number: long
+ *! date: Date
+ * amount: Money
+ * vat: double
+ * state: InvoiceSate = NOT_YET_PAID
- computeAmount() : void
+ addWorkOrder(wo :WorkOrder) : void
+ removeWorkOrder(wo :WorkOrder) : void
+ settle() : void
+ isSettled() : boolean
«entity»
Client
+ !@ nif: String
+ ! name: String
+ ! surname: String
+ ! email: String
+ ! phone: String
+ ! address: Address
«entity»
Mechanic
+ !@ nif: String
+ ! name: String
+ ! surname: String
«entity»
Intervention
+ *!>@ mechanic: Mechanic
+ *!>@ workOrder: WorkOrder
+ *!@ date: Timestamp
+ minutes: int
+/ * amount: Money
«entity»
Substitution
+ *! quantity: int
+/ * amount: Money
«entity»
SparePart
+ !@ code: String
+ ! description: String
+ ! price: Money
«entity»
WorkOrder
+ *!@ date: Timestamp
+ *!>@ vehicle: Vehicle
+ ! description: String
+ * amount: Money
+ * state: WorkOrderState = OPEN
+ assignTo(m :Mechanic) : void
+ unassign() : void
+ markAsFinished() : void
+ markAsInvoiced() : void
+ markBackToFinished() : void
+ reopen() : void
+ isAssigned() : boolean
+ isFinished() : boolean
+ isInvoiced() : boolean
+ isOpen() : boolean
«entity»
Vehicle
+ !@ plateNumber: String
+ ! make: String
+ ! model: String
«entity»
VehicleType
+ !@ name: String
+ ! pricePerHour: Money
V 2024.10
hold
1
0..*
settle
0..*
0..*
bill
1..*
0..1
assign
0..1
0..*
intervene
0..*
1
intervene
1
0..*
substitute
0..*
0..*
own
1
0..*
fix
1
0..*
+vehicle
classify
1
0..*