Exercise

Class diagrams

Prof. Dr. Bernhard Rumpe
Lehrstuhl für Software Engineering
RWTH Aachen

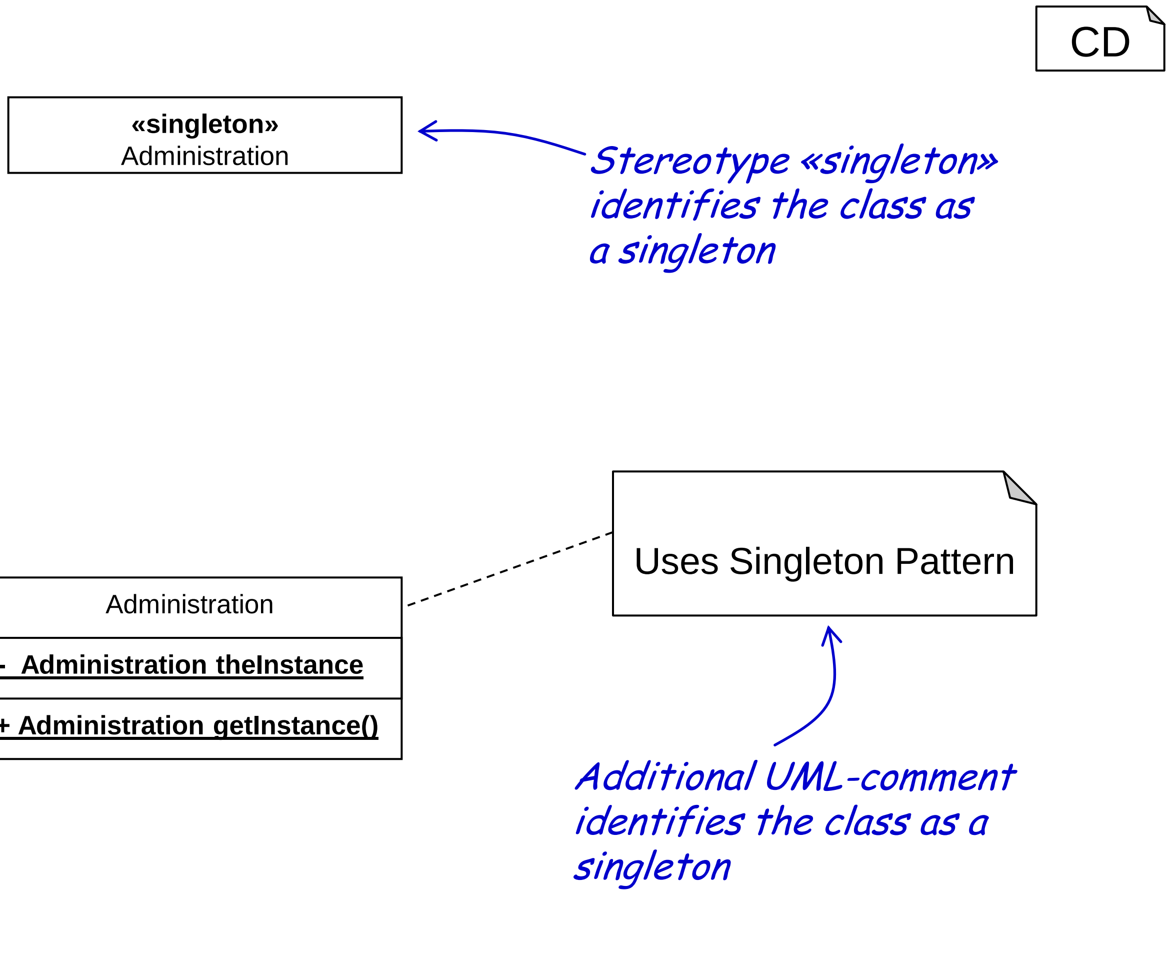
http://www.se-rwth.de/

Exercise 2.1

- Where could you use the design pattern “Singleton“?

- Extend accordingly the class diagram.

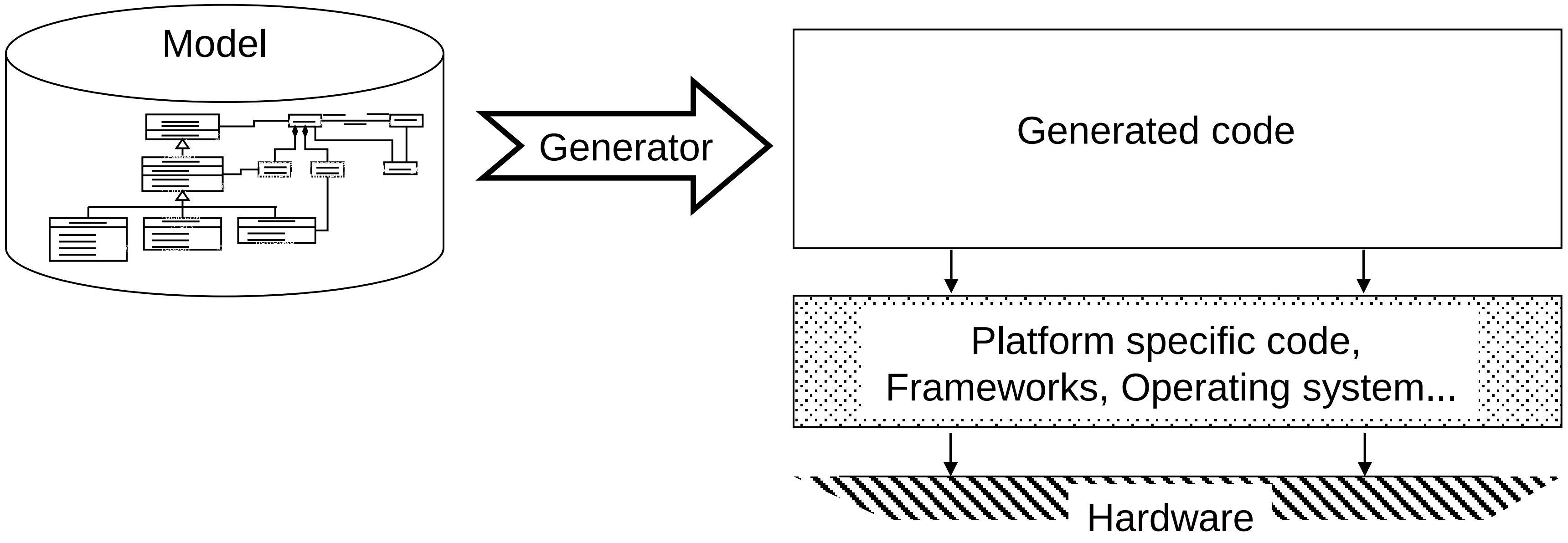
Exercise 2.1


CD
«singleton»
Administration
Stereotype «singleton» identifies the class as a singleton
Administration
- Administration theInstance
+ Administration getInstance()
Uses Singleton Pattern
Additional UML-comment identifies the class as a singleton

Elements of the Singleton pattern specified directly in the class

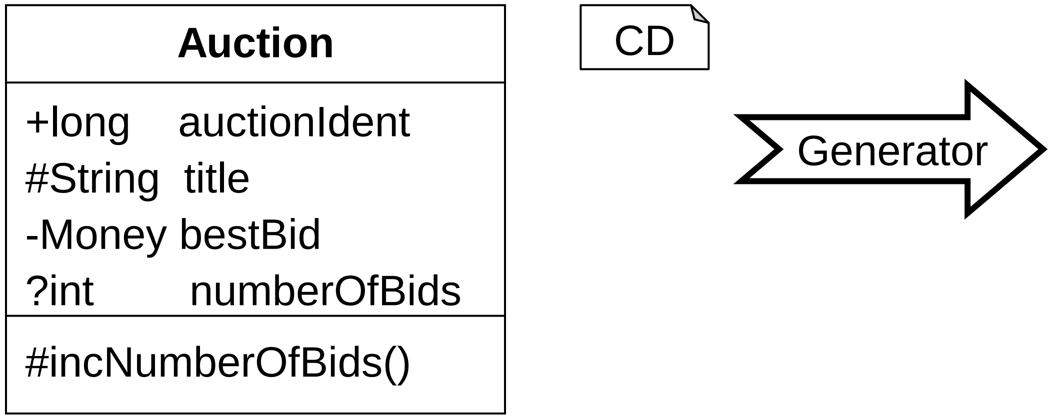
Code Generation

- Principle: Mapping of the model into a programming language



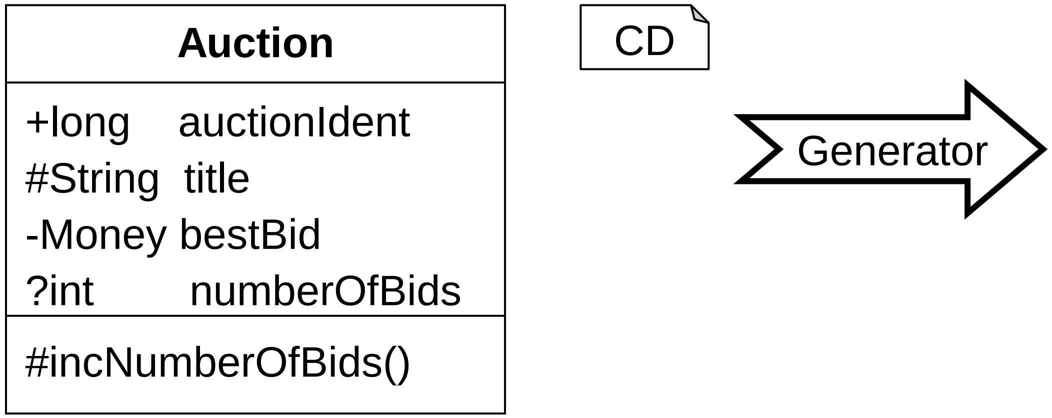
⇨ Even if there is no “automatic” generator available, it is possible to put the transformations of UML into code.

Code Generation from a Class



Suggestions?

Code Generation from a Class



```
class Auction {
  public long       auctionIdent;
  protected String title;
  private Money     bestBid;
  public int        numberOfBids;

  protected void incNumberOfBids() { ... }
}
```

JAVA

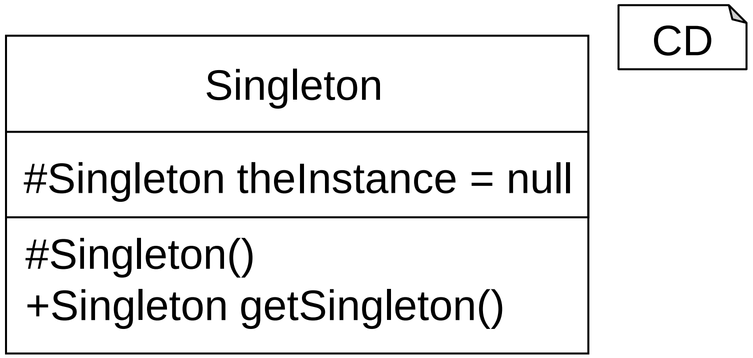
Exercise 2.2

- Suppose that the use of the design pattern “Singleton“ in a class is only marked with the use of the stereotype “Singleton“.

- Implement the code for this design pattern in a way it could have been automatically generated.

Solution 2.2

- Singleton-Pattern:
  1. Visibility of the constructors: private
  2. Static attribute as exclusive instance of the class
  3. Static method for the access to the static instance



- => Generation: Modification of the code and code generation

- If necessary class is marked as final
  (prohibits creation of instances through inheritance)

Solution 2.2

JAVA

```
public class Administration {
  // CodeGeneration: Singleton Start
  private static Administration theInstance = null;

  public  static Administration getInstance() {
    if (theInstance ==  null) {
      theInstance = new Administration();
    }
    return theInstance ;
  }
  // CodeGeneration: Singleton End

  // CodeGeneration: Visibility changed to private
  private Administration() {
    …
  }
  public void return(RentalContract contract) {
    …
  }
  …
}
```

Solution 2.2

- Thread save variant:

JAVA
```
public class Administration {
  // CodeGeneration: Singleton Start
  private static Administration theInstance = new Administration();

  public  static Administration getInstance() {
    return theInstance ;
  }
  // CodeGeneration: Singleton End

  // CodeGeneration: Visibility changed to private
  private Administration() {
    …
  }
  public void return(RentalContract contract) {
    …
  }
  …
}
```

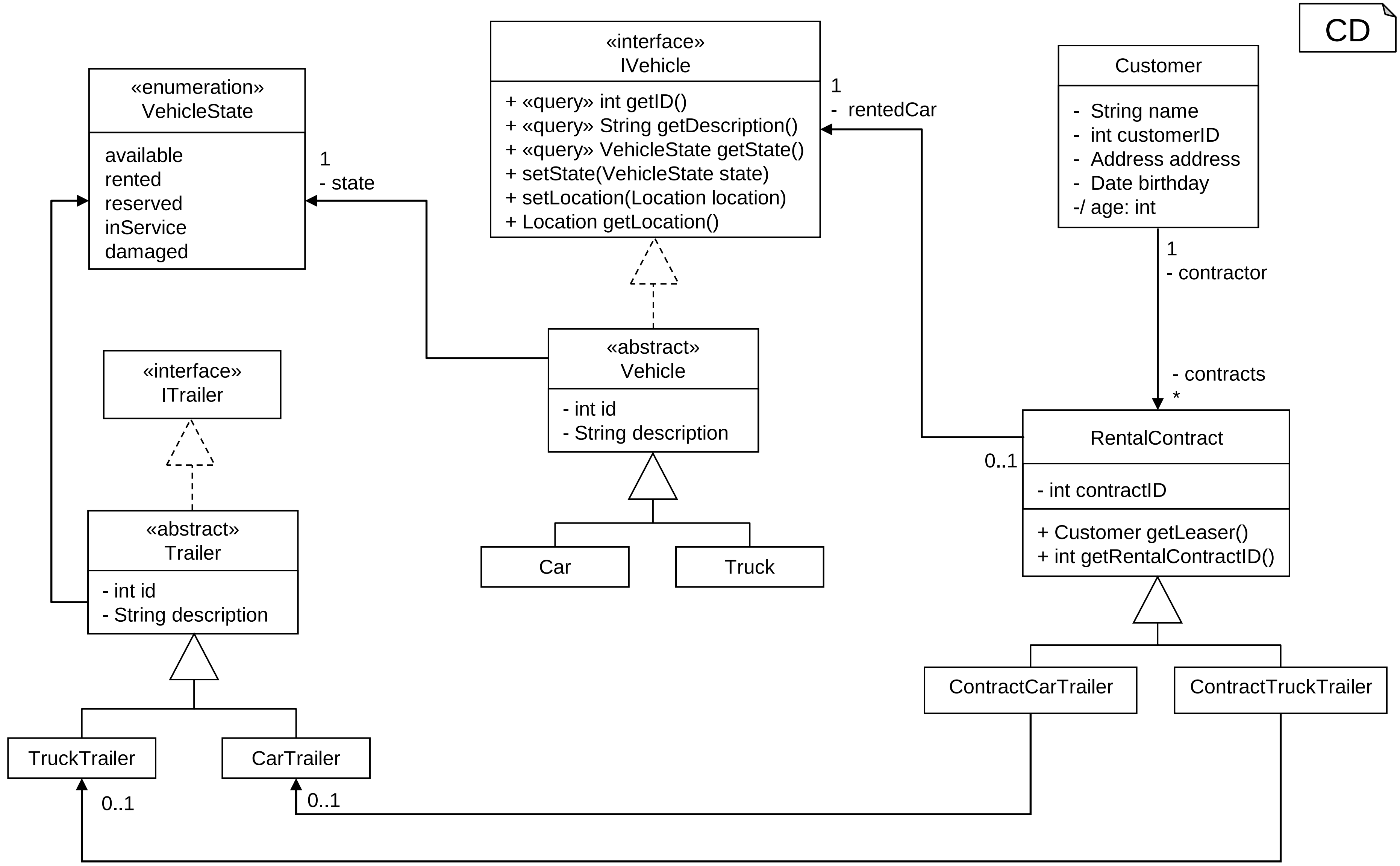
Exercise 2.3

After the car rental has extended its offer by trucks, it is possible to rent trailers for cars and trucks. But there are two different kinds of trailers: one type for every type of vehicle.

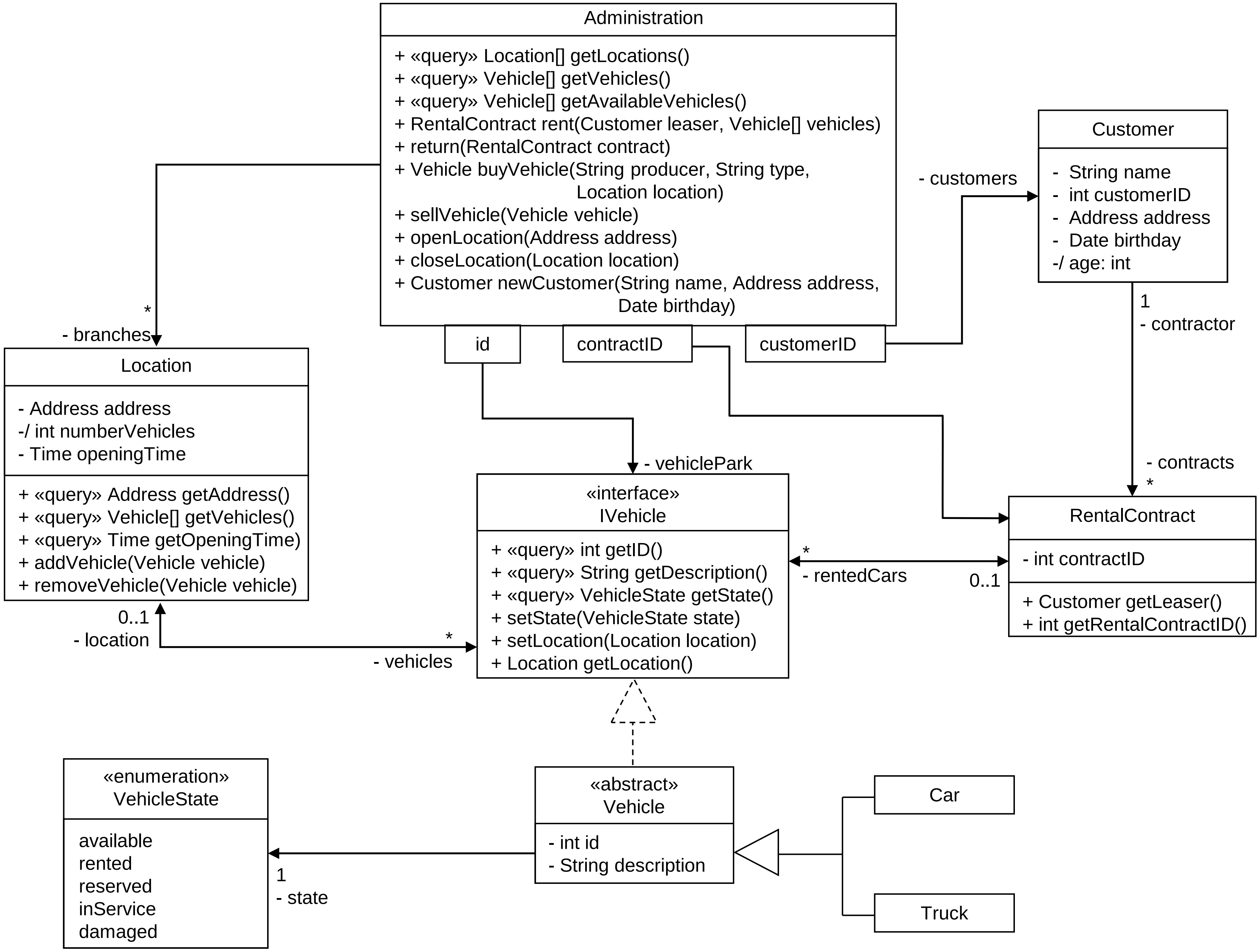
Change your class diagram in such a way that

- a trailer can only be rented with one vehicle and
- a truck trailer can only be rented in combination with a truck and a car trailer can only be rented with a car.

Solution 2.3


CD
«enumeration»
VehicleState
available
rented
reserved
inService
damaged
1
- state
«interface»
IVehicle
+ «query» int getID()
+ «query» String getDescription()
+ «query» VehicleState getState()
+ setState(VehicleState state)
+ setLocation(Location location)
+ Location getLocation()
1
- rentedCar
Customer
- String name
- int customerID
- Address address
- Date birthday
-/ age: int
1
- contractor
- contracts
*
«abstract»
Vehicle
- int id
- String description
Car
Truck
«interface»
ITrailer
«abstract»
Trailer
- int id
- String description
TruckTrailer
CarTrailer
0..1
0..1
RentalContract
0..1
- int contractID
+ Customer getLeaser()
+ int getRentalContractID()
ContractCarTrailer
ContractTruckTrailer

Questions?


Administration
+ «query» Location[] getLocations()
+ «query» Vehicle[] getVehicles()
+ «query» Vehicle[] getAvailableVehicles()
+ RentalContract rent(Customer leaser, Vehicle[] vehicles)
+ return(RentalContract contract)
+ Vehicle buyVehicle(String producer, String type, Location location)
+ sellVehicle(Vehicle vehicle)
+ openLocation(Address address)
+ closeLocation(Location location)
+ Customer newCustomer(String name, Address address, Date birthday)
id
contractID
customerID
- customers
Customer
- String name
- int customerID
- Address address
- Date birthday
-/ age: int
1
- contractor
- contracts
*
*
- branches
Location
- Address address
-/ int numberVehicles
- Time openingTime
+ «query» Address getAddress()
+ «query» Vehicle[] getVehicles()
+ «query» Time getOpeningTime()
+ addVehicle(Vehicle vehicle)
+ removeVehicle(Vehicle vehicle)
0..1
- location
*
- vehicles
- vehiclePark
«interface»
IVehicle
+ «query» int getID()
+ «query» String getDescription()
+ «query» VehicleState getState()
+ setState(VehicleState state)
+ setLocation(Location location)
+ Location getLocation()
*
- rentedCars
0..1
RentalContract
- int contractID
+ Customer getLeaser()
+ int getRentalContractID()
«enumeration»
VehicleState
available
rented
reserved
inService
damaged
1
- state
«abstract»
Vehicle
- int id
- String description
Car
Truck