**Seminar 2**

**Object-Oriented Design, IV1350**

**Adrian Kastrati,** [**kastrati@kth.se**](mailto:kastrati@kth.se)

**2022-06-05**

**Contents**

**1 Introduction 3**

**2 Method 4**

**3 Result 5**

**4 Discussion 8**

1. **Introduction**

The seminar task is to design a program in an object-oriented programming language that can handle a purchase of items at a store with internal and external systems. The seminar task was discussed with Deni Persson, Jesper Munkeby and William Eriksson.

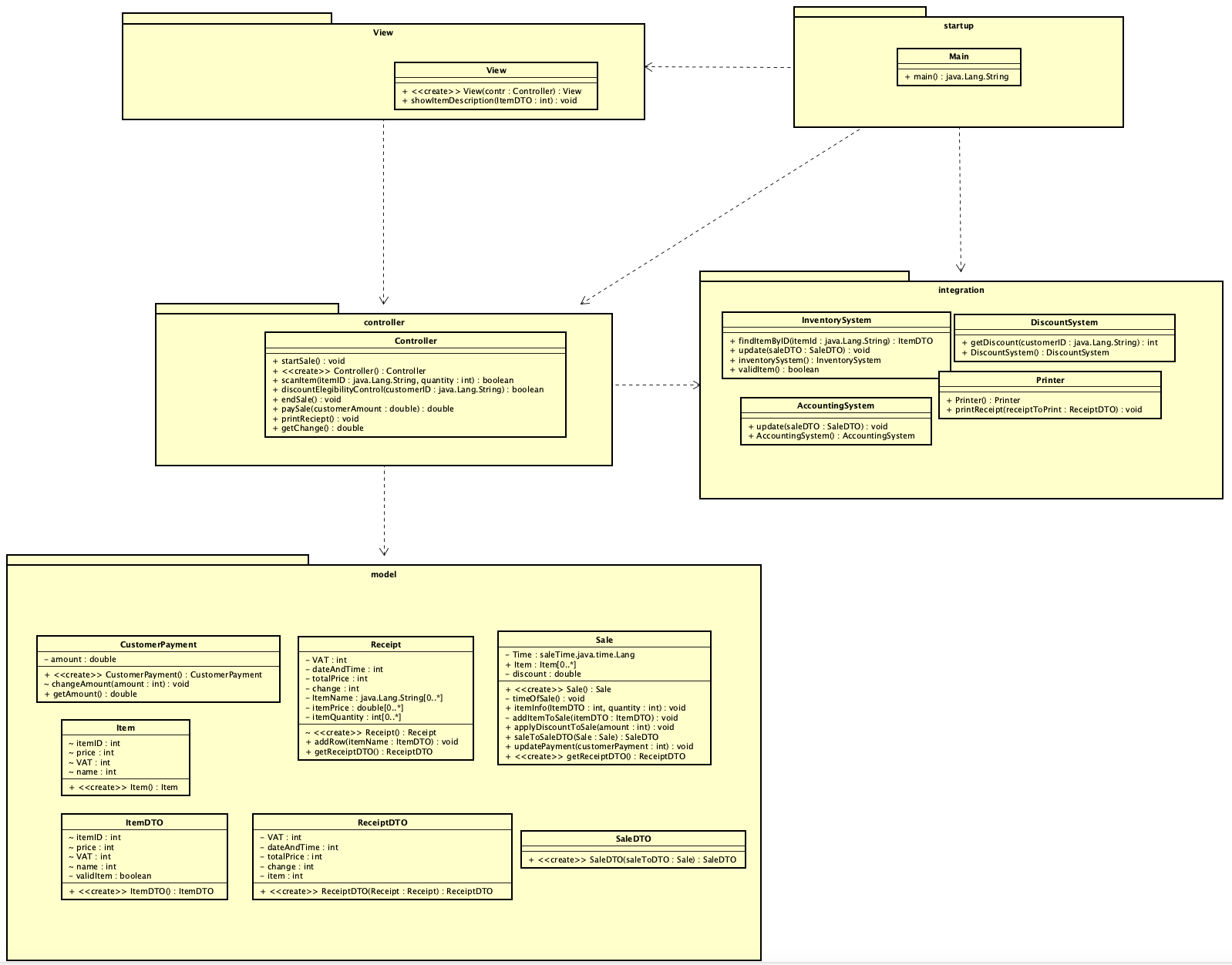
1. **Method**

The program design progress started with adding the interactions from the SSD in order. When an action in the SSD resembled a potential method between a class in the domain model it was taken I to consideration for how it could be handled to minimize the connection to other classes. all while following the design process from lectures where the principles of “high coherency” and “low coupling”. After this, the design was revised to see where possible changes for the better could be made.

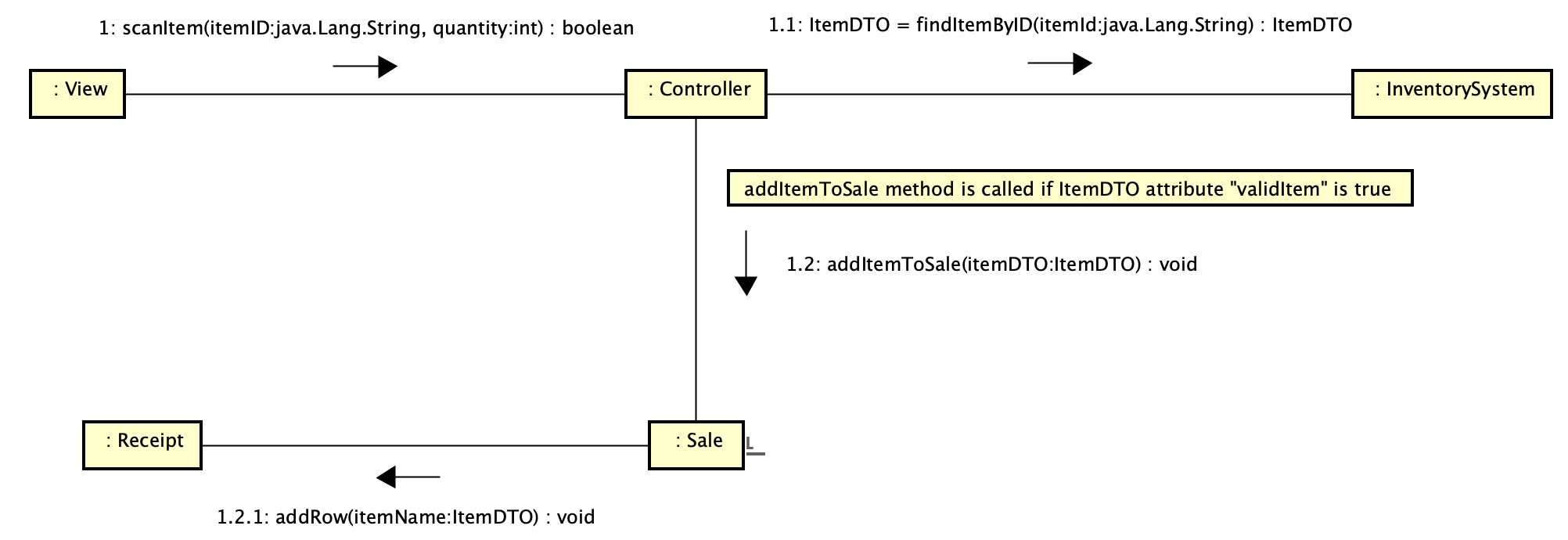
1. **Result**

The design has 6 layers were with one data transfer object (DTO) for the item class.

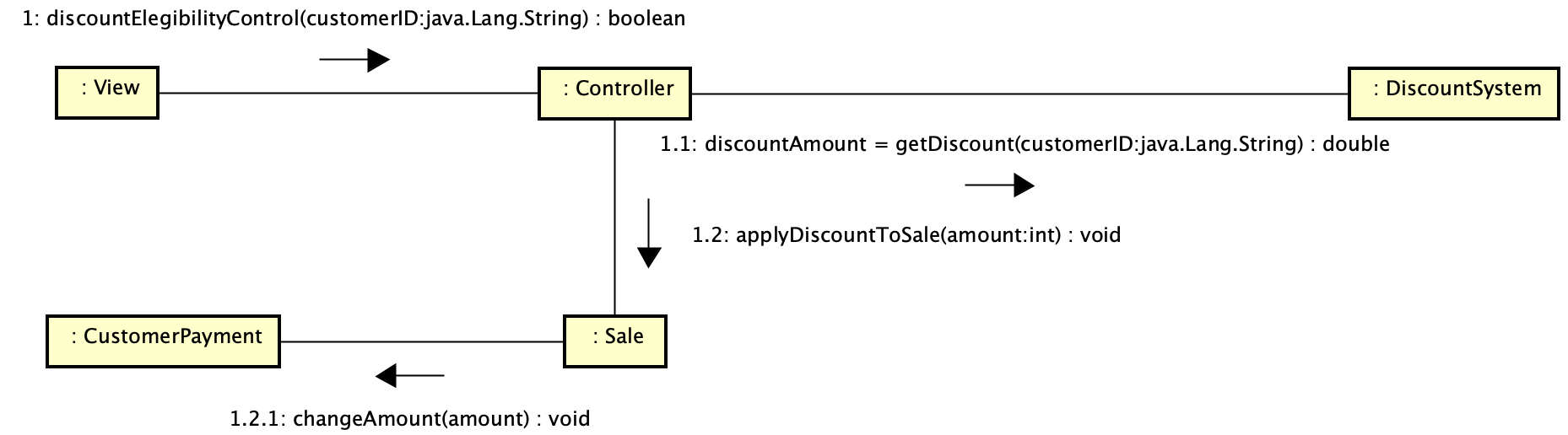
The updated version of the design has 5 layers, View, startup, controller, integration, model. Which follow the MVC patternThe layers consist of total 13 classes with three different DTOs for the Receipt, Item and Sale class. The different layers hold classes that are designed to only interact with other classes in The design ended up in 7 different communication diagrams that fulfill the needed actions, the communication diagrams are sorted into different operations to achieve one goal.



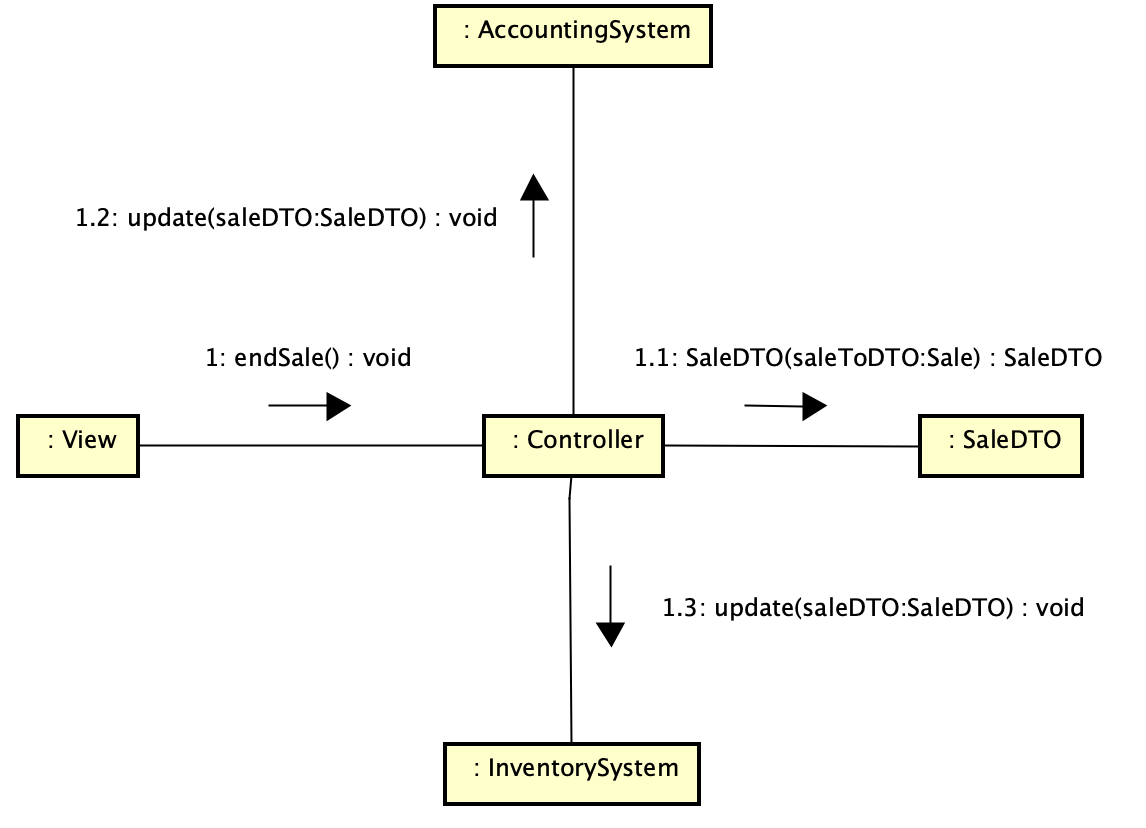
**Figure 3.1** Complete class diagram of the different packages with containing packages



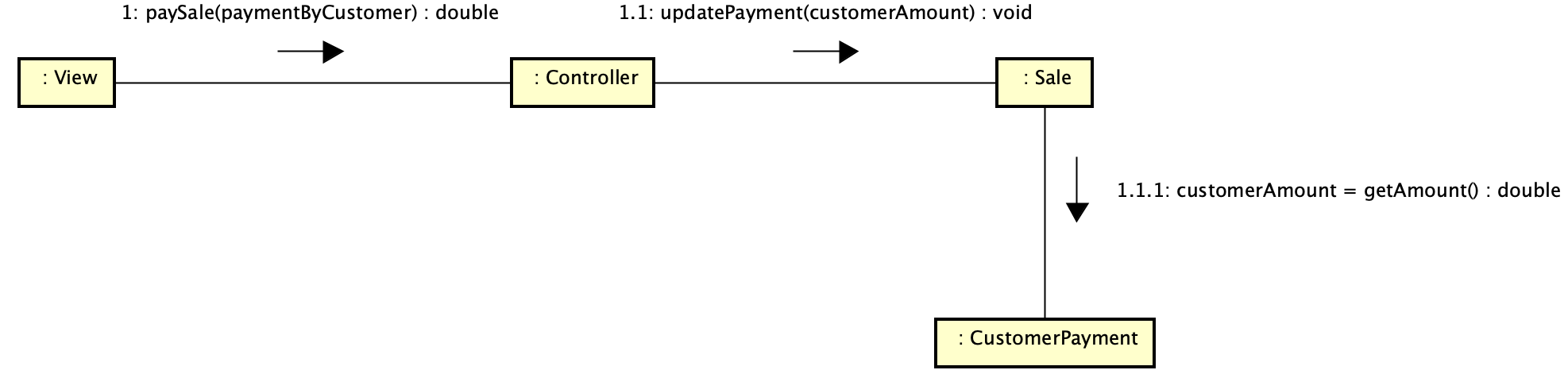
**Figur 3.2** Interaction diagram for adding an item



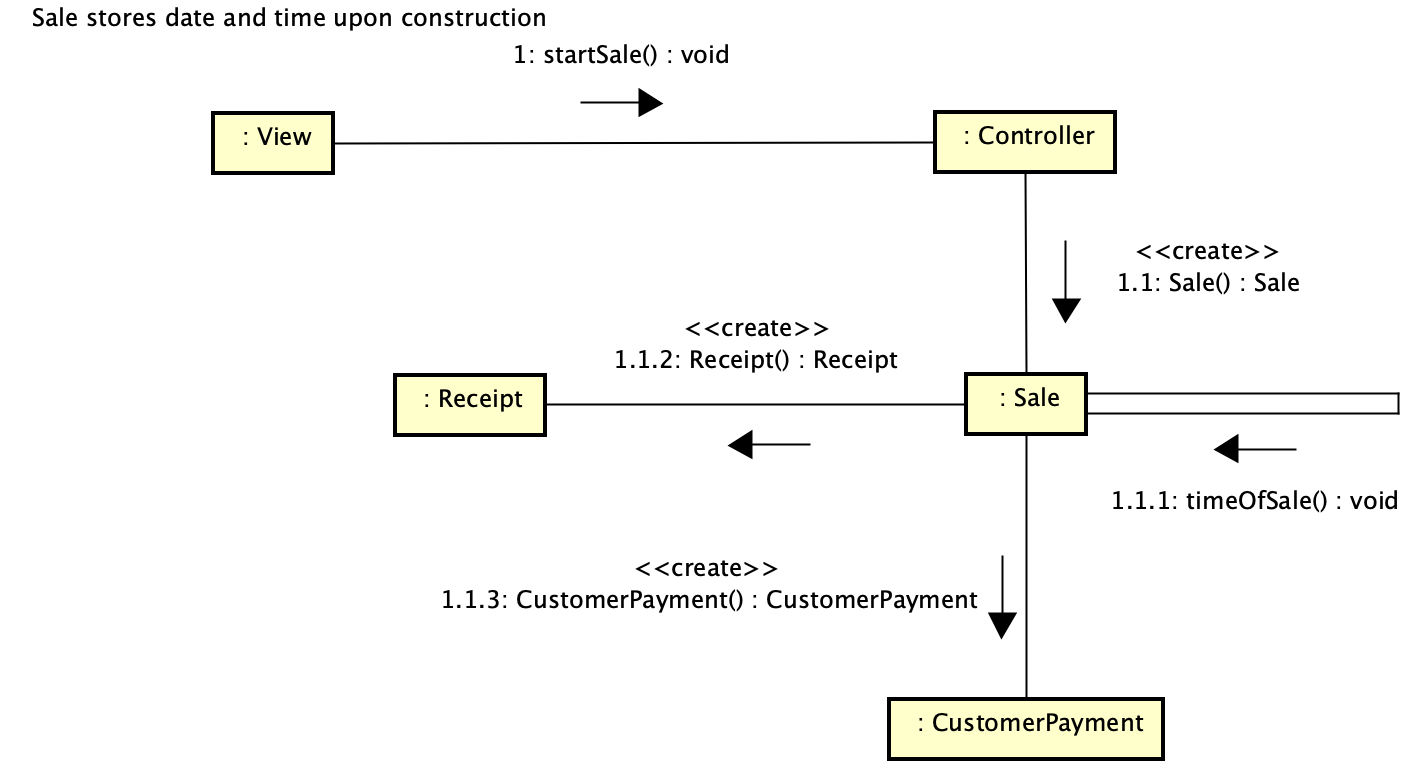
**Figure 3.3** Interaction diagram for adding discount to sale



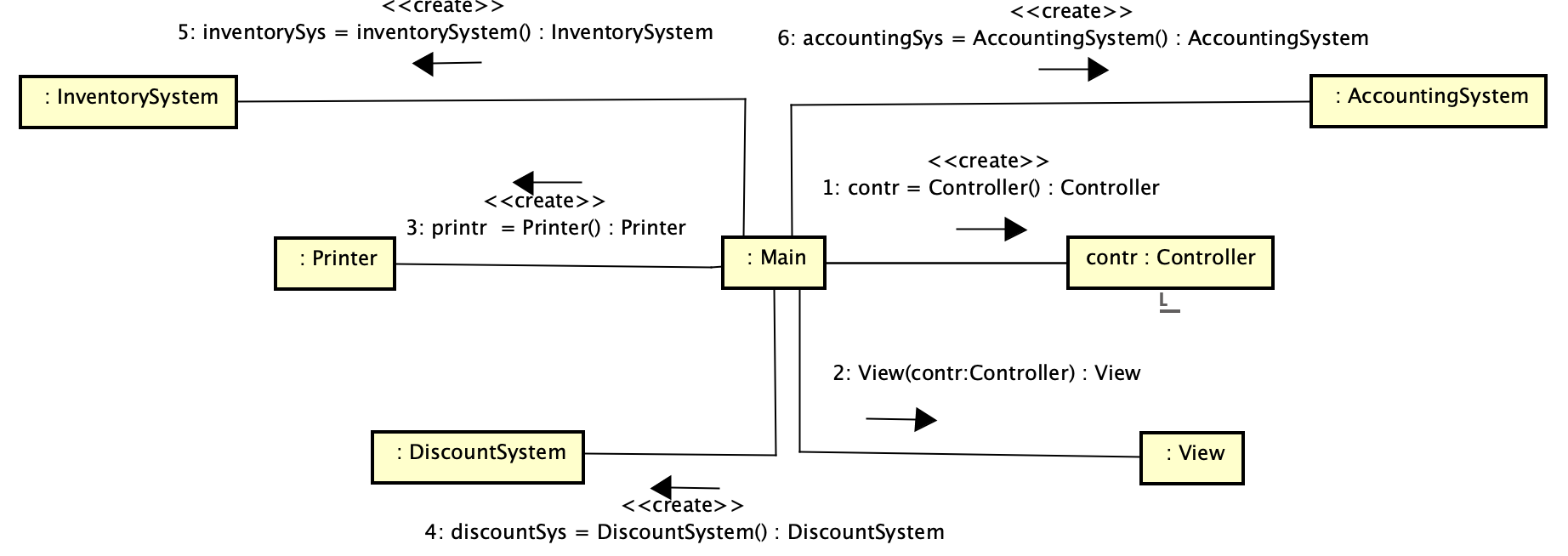
**Figure 3.4** Interaction diagram for ending s sale



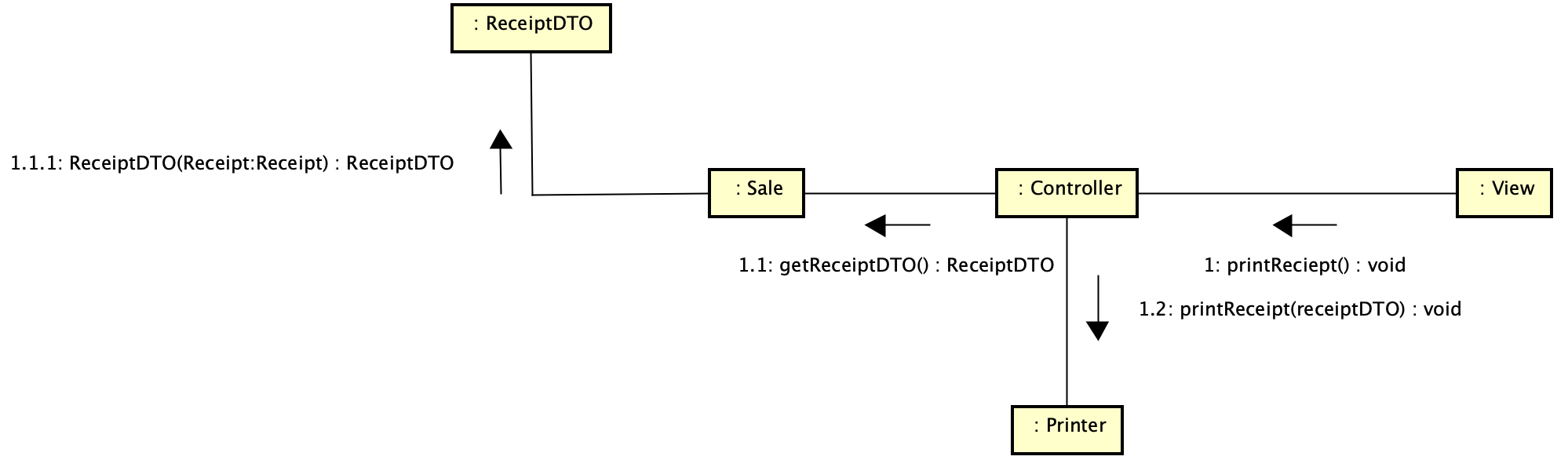
**Figure 3.5** Interaction diagram for payment of sale

****

**Figure 3.6** Interaction diagram for starting a new Sale

****

**Figure 3.7** Interaction diagram for the startup of the program



**Figure 3.8** Interaction diagram for printing a receipt

1. **Discussion**

This is the discussion regarding the updated version of the design. The attributes of the classes are set to private to increase encapsulation and use get/set methods to fetch those values. The DTO classes do not have attributes that are not set to private since they are only designed to get the primitive data values. This could be seen as lowering the cohesion since it is not possible to use getters and setters for the values but they are supposed to be open so that is my reasoning for not using them. The rest of the classes uses getters/setters for the attributes. The cohesion would lower if some of the classes got some added attributes, as currency for items, which would lead to some new classes. Creating the classes I felt that the quantity attributes is appropriate for the current size of the program. Some of the classes as receipt could be rewritten but since it is not dependent on a real printer or even a class that creates text files it is like now.