EXAM 19/04/2007 (UML PART)

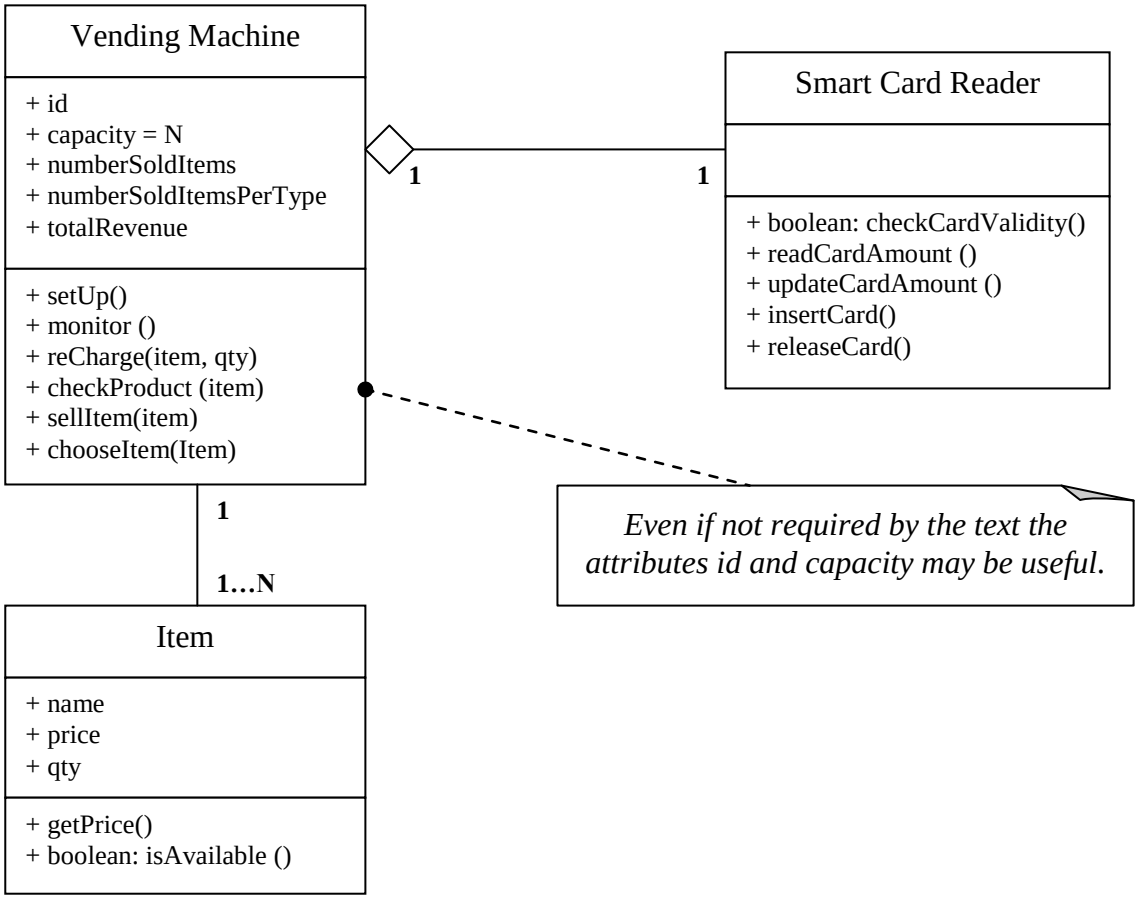
1 a

Model with a **class diagram** the following System: Vending Machine.
A vending machine sells small, packaged, ready to eat items (chocolate bars, cookies, candies, etc.). Each item has a price and a name. A customer can buy an item, using a smart card (issued by the vending machine company) to pay for it. No other payment forms (i.e. cash, credit card) are allowed. The smart card records on it the amount of money available.
The functions supported by the system are:

- Sell an item (choose from a list of items, pay item, distribute item)
- Recharge the machine
- Set up the machine (define items sold and price of items)
- Monitor the machine (number of items sold, number of items sold per type, total revenue)

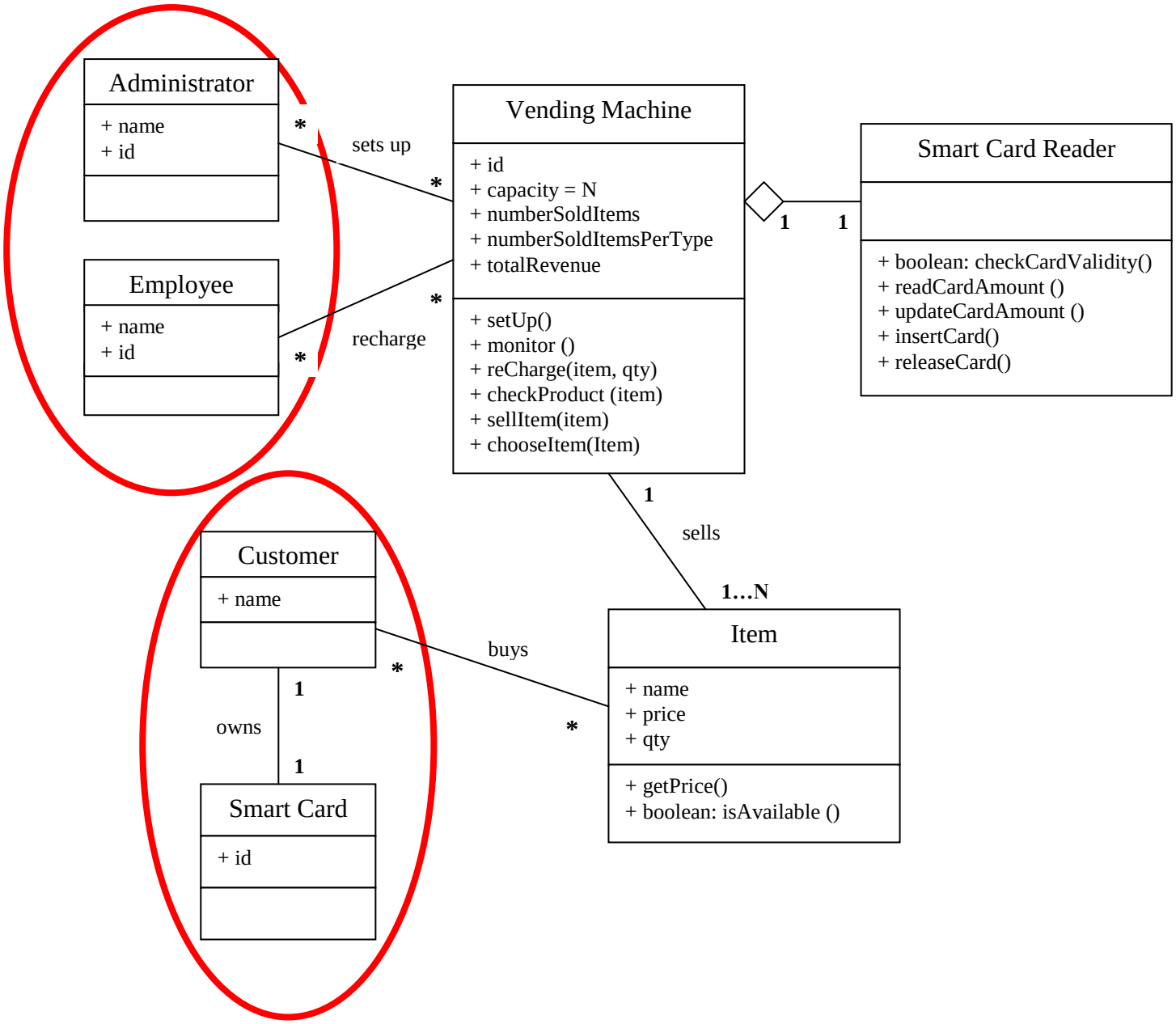
The system can be used by a customer, a maintenance employee (who recharges items in the machines), an administrator (who sets up the machine).

POSSIBLE SOLUTION



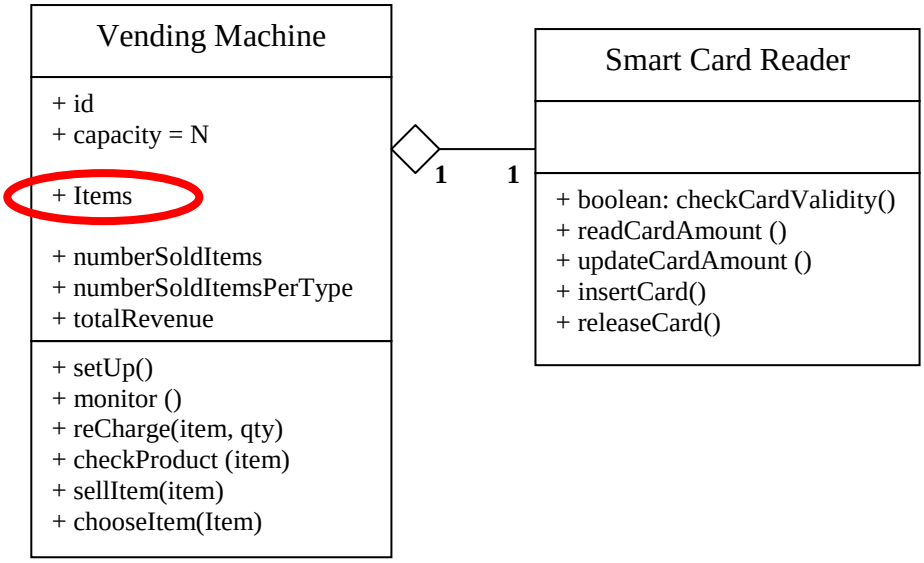
COMMON ERRORS AND SUGGESTIONS

1)



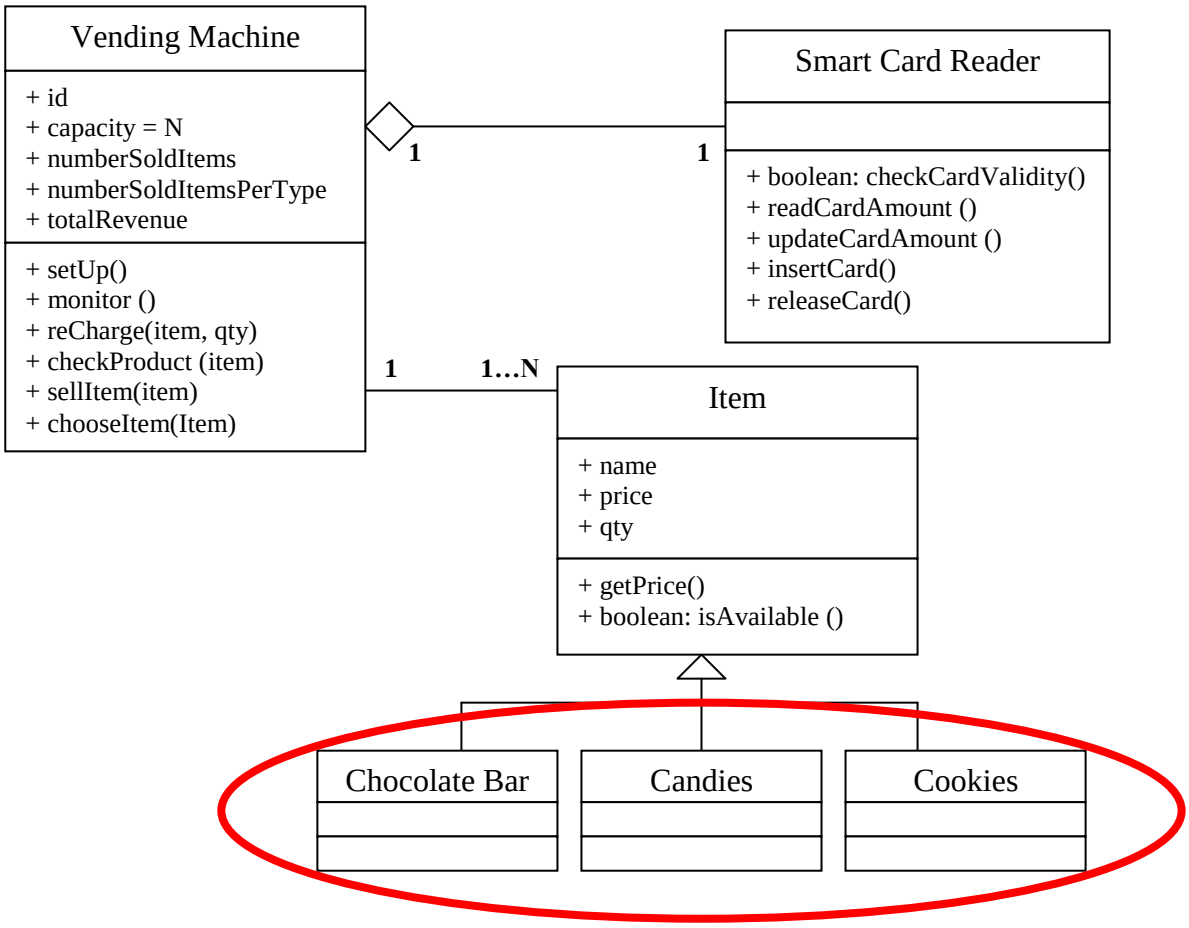
This solution isn't totally wrong, but the above one is better. In fact, the circled classes are **useless**. They are for sure **actors in the use case diagram,** but they are not very meaningful here. In fact, according to the specifications, we don't need to keep track of which user bought something or to keep track of the id of his smart card (we're not interested in saying that the customer Mario Rossi bought a chocolate bar with the smart card # 1).
We can have a hint of the fact that those classes are not needed by analyzing that no special attributes are defined for them in the specifications.
So it would be preferable to avoid inserting these classes in the class diagram, whereas it is very important that Customer, Administrator and Employee are actors in the use case diagram.

2)



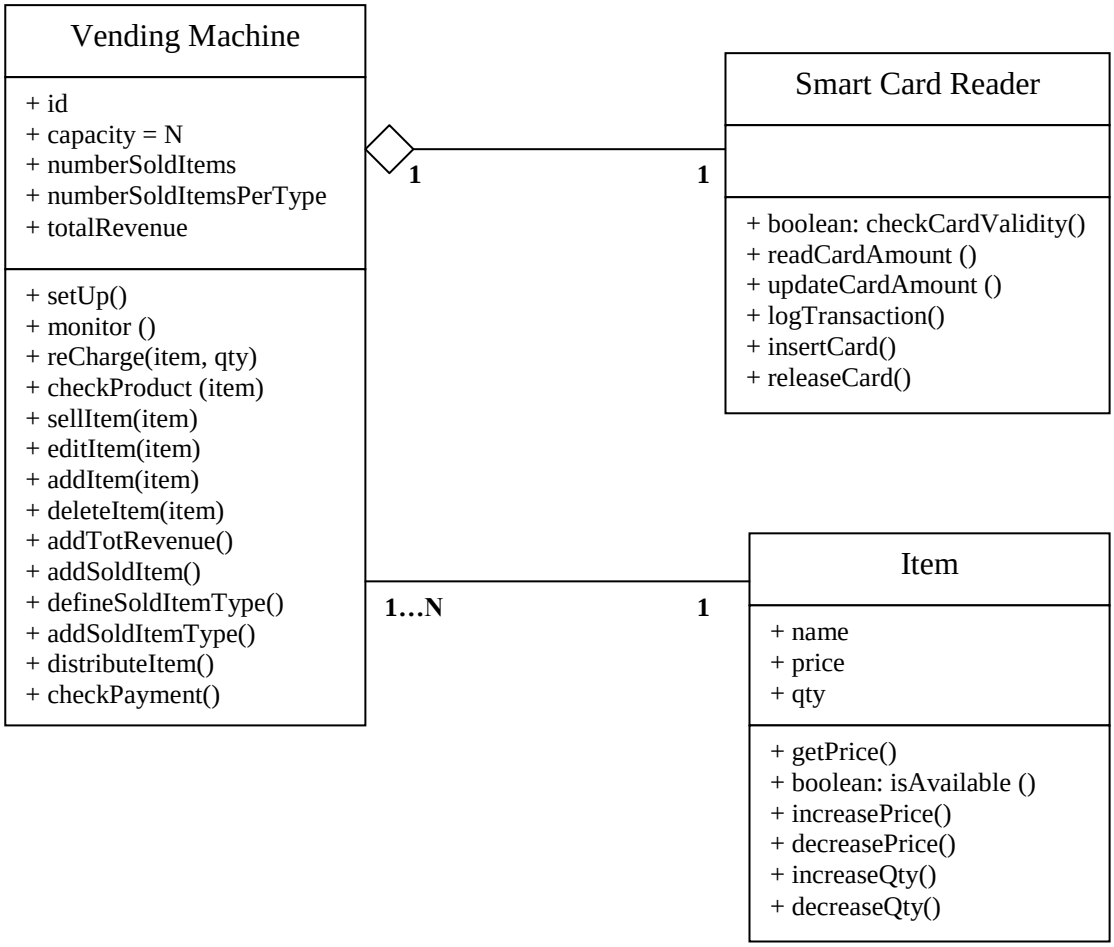
This solution is **not good** because it doesn't foresee the class Item and **replaces it with an attribute** in the class Vending Machine (no matter what kind of attribute). The problem is that there are attributes (the name, the price) related to each Item and, without using a separate class, is difficult to represent them (usually a good way to understand when we must model with a class is, in fact, to look whether there are attributes related to the thing we want to model).

3)



The **classes Chocolate Bar, Candies, Cookies,** that are used in the solution above, **are useless** and so it's better to avoid using them (this is an example of over-specification).
They're useless in the sense that there aren't (at least in the given specifications) special attributes that distinguish them from the class Item, from which they all inherit the same attributes and operations.

4)

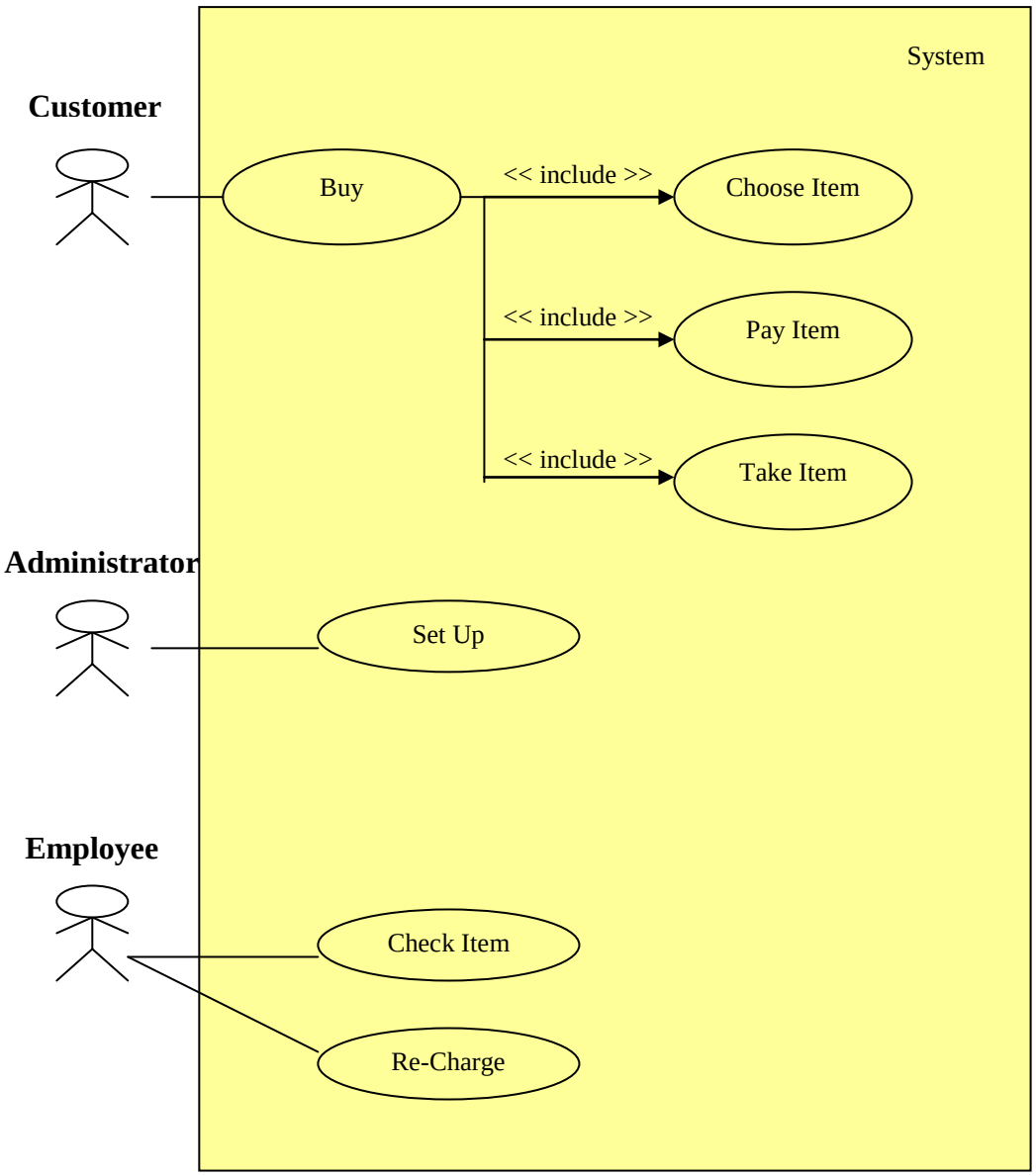


This solution is **correct** but probably too much **detailed** and **complicated**. In each class there are a lot of operations and this is because each single activity is described in detail.
Although this solution is absolutely correct, the first one is probably preferable because it is simpler and more understandable (in that solution the meaning of each operation is straightforward whereas here we have operations such as, for example, `defineSoldItemType()` that requires additional explanations).
Since, usually, the goal of these kinds of diagrams is to be clear for stakeholder, is better to **keep them simple** also because complex diagrams are more error prone (we run the risk of writing wrong diagrams, or to write diagrams that forces or misunderstands the specifications).

1b

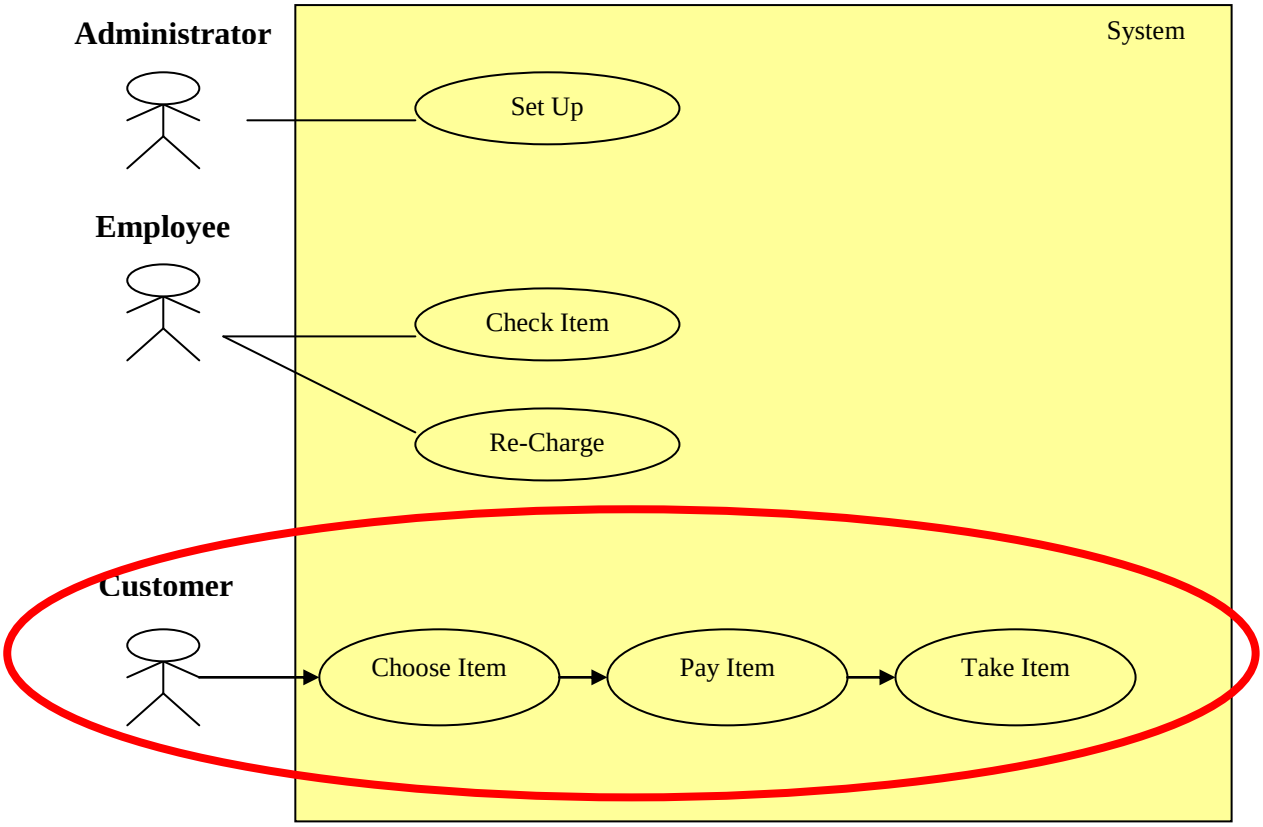
Enrich the model of the vending machine with **use case diagram**.

POSSIBLE SOLUTION

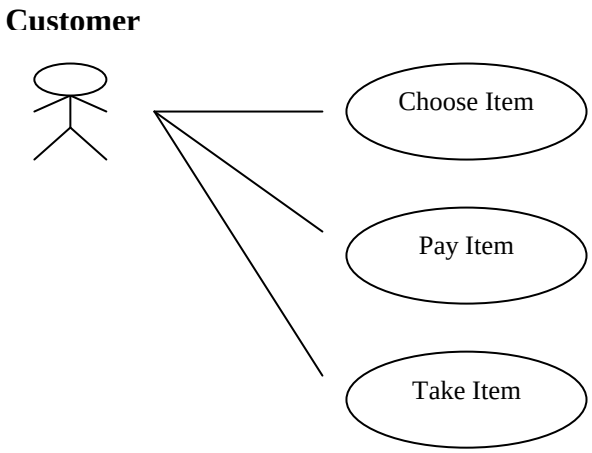


COMMON ERRORS AND SUGGESTIONS

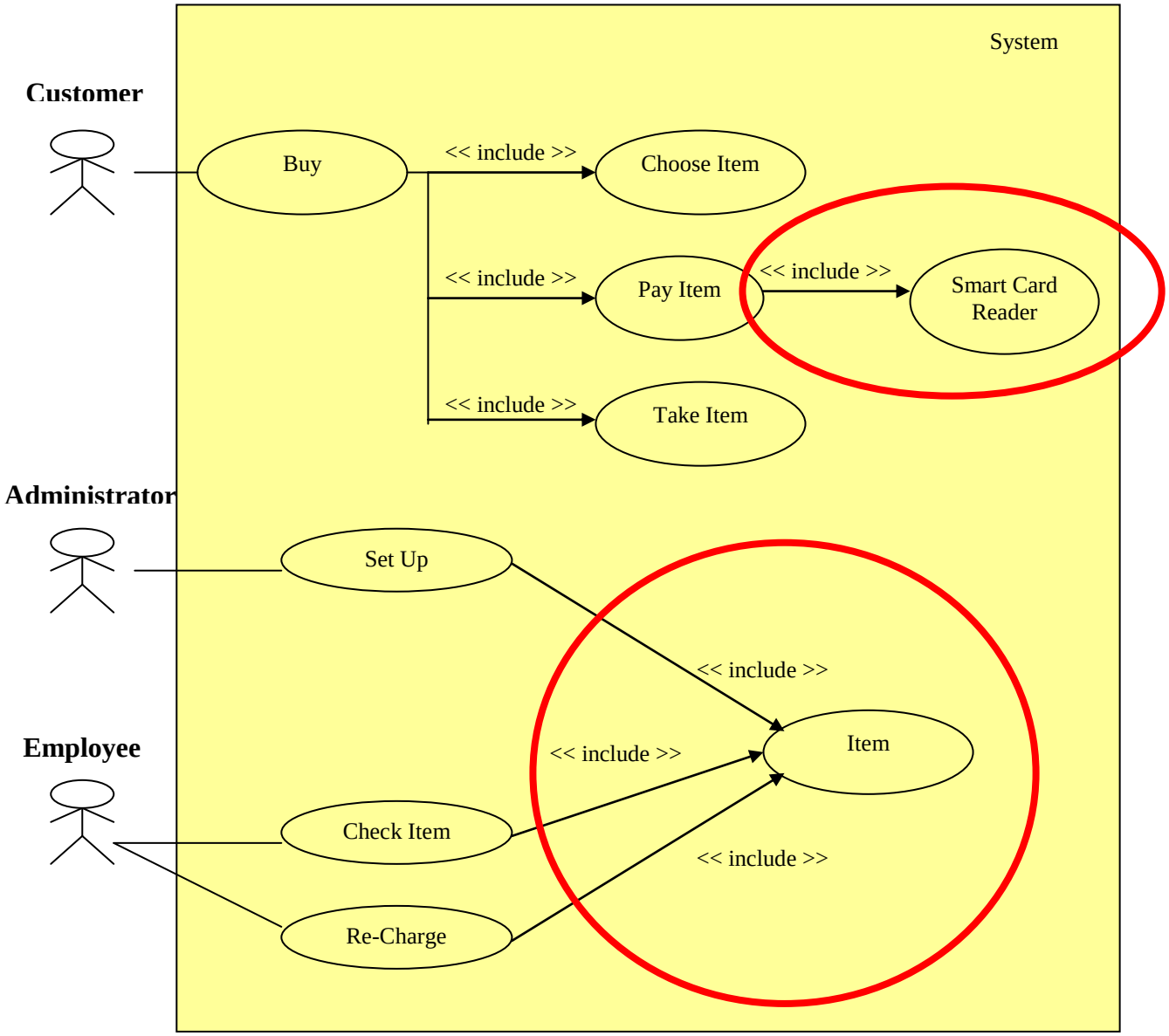
1)



This solution is **wrong**. We must remember that a **use case diagram is not a flow chart** whereas the circled part of the solution represents a flow chart.
The **correct way** of drawing that part is the following:



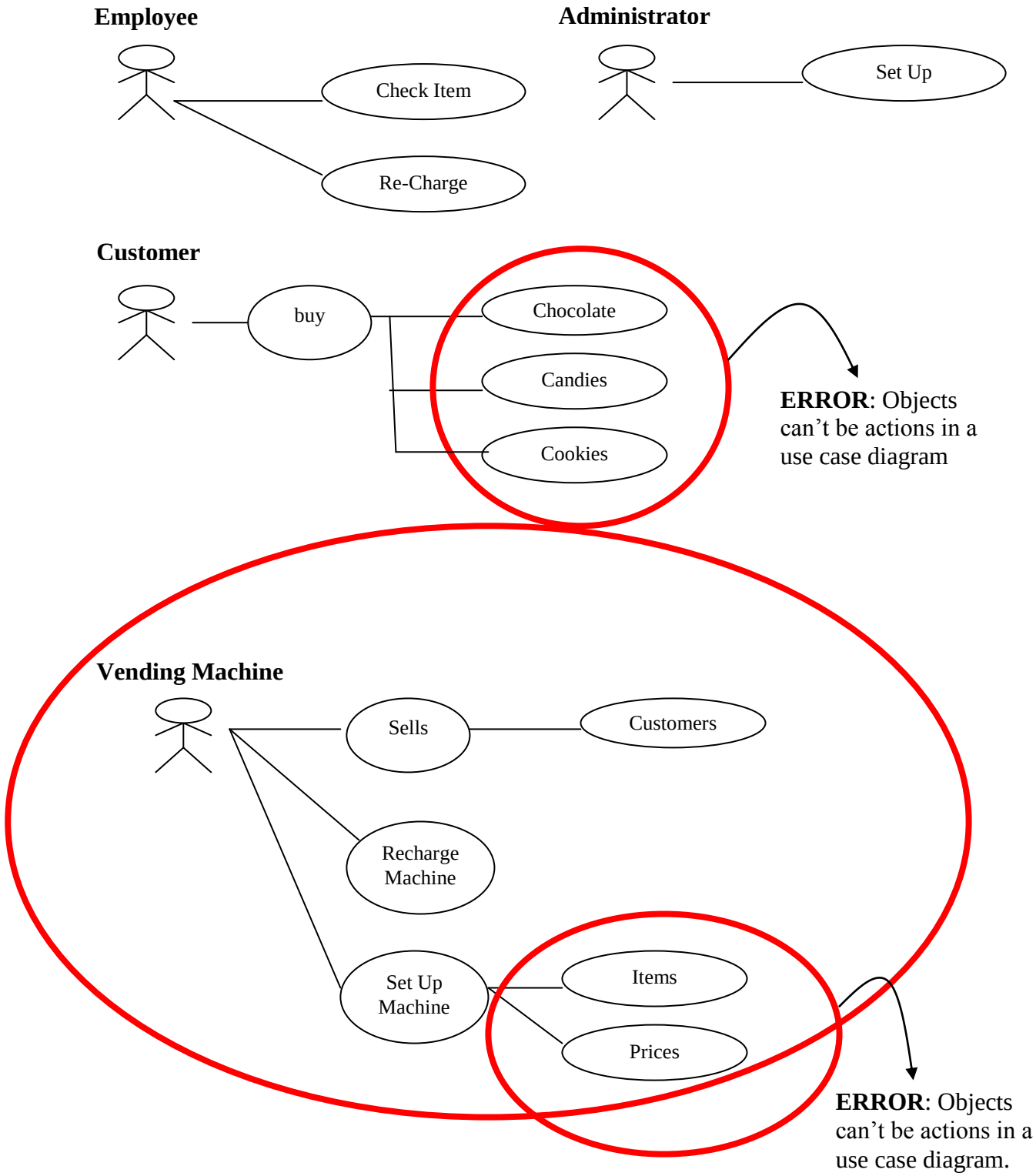
2)



This solution is **wrong** and its wrong parts are circled in red. In fact both ***Item*** and Smart ***Card Reader*** are object (classes) and so **they can only be actors** (if needed) whereas in this solutions they are represented as **actions**.

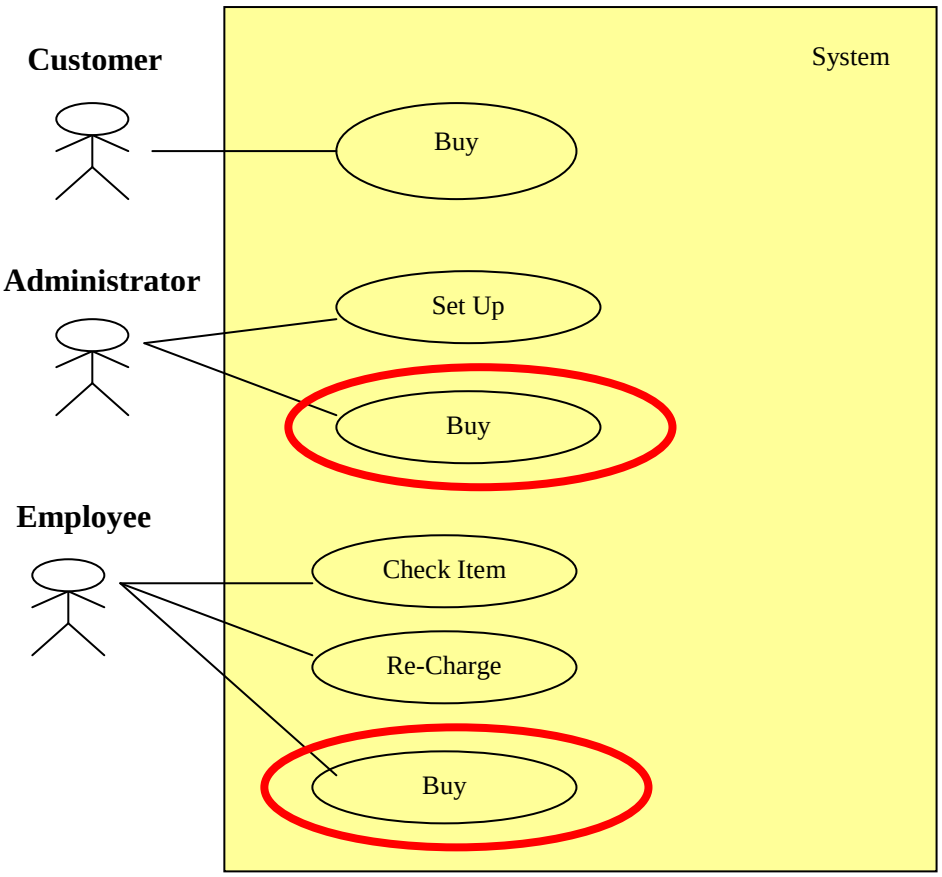
The **correct solution** is to **draw Smart Card Reader as an actor** (or to delete it) and to **delete Item** that isn't really needed in the use case diagram.

3)



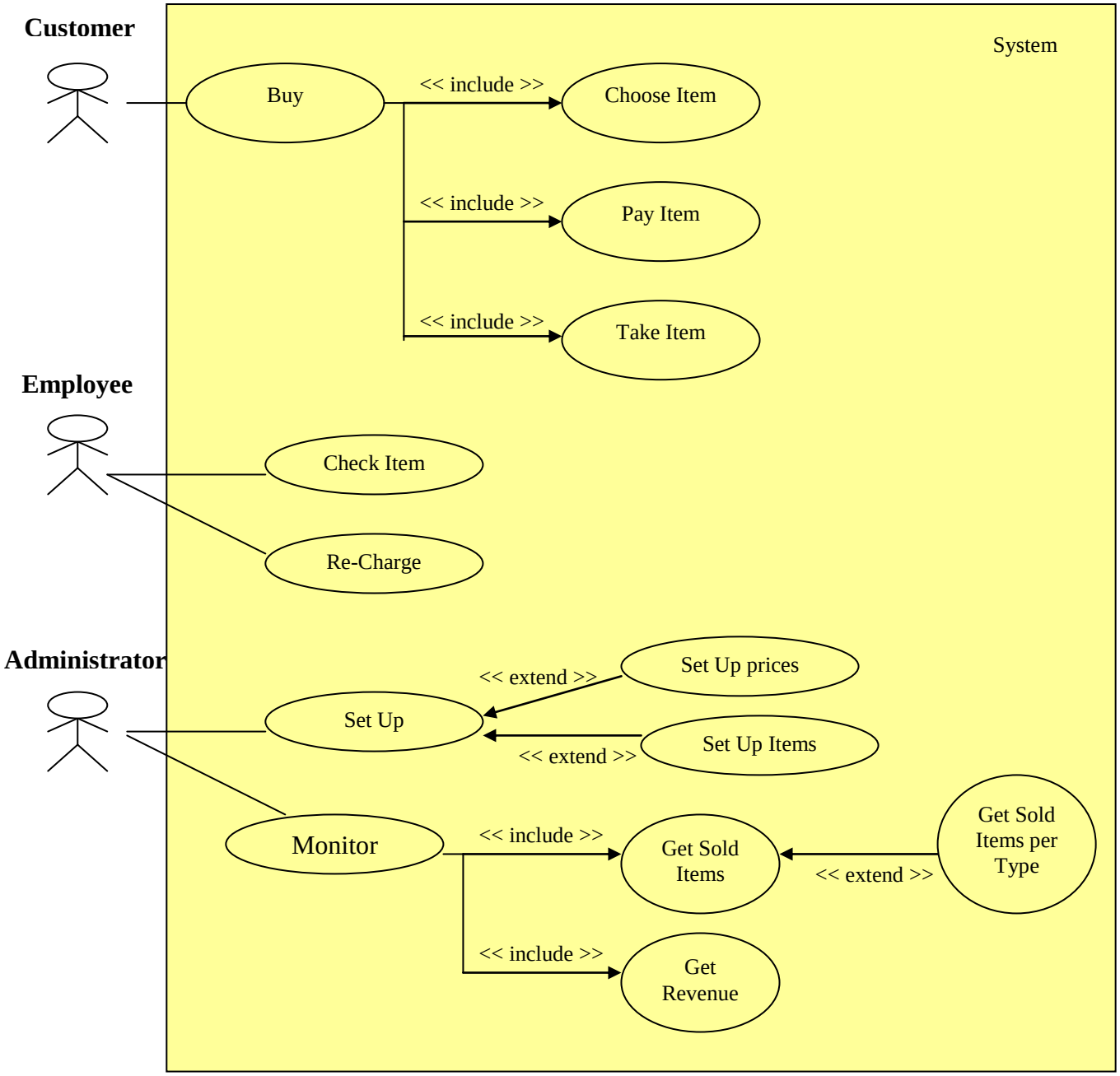
This solution is **totally wrong** for two reasons. The first one is that it contains objects treated as actions (similarly to the previous solution); the second one is that the Vending **Machine is here represented as an actor**. To understand what's wrong consider that it is like saying: "the actor Vending Machine does the actions of recharging (or setting up, or selling) the Machine". This is a non sense as the diagram itself!

4)



The problem of this solution (a part from the fact that it's not detailed enough) is that the action **buy** is defined for the actor Customer (correctly) but also **for the actors Administrator and Employee**. For these two actors **the action is not meaningful**: it's true that an administrator or an employee can buy a snack, but in this case they're acting as normal customer and so this scenario is already represented in the Use Case Diagram also without the action (that is so a **useless redundancy**).

5)



This solution is perfectly **correct**. Its only problem is that it is a bit **too complex and detailed.** Writing correct complex solutions is surely more difficult than writing simpler but equally correct ones (as the first proposed). A suggestion is so **to keep** (at least in a first phase) **the solution simple** in order to avoid problems and misunderstandings.
For example, in this solution there is the action ***Monitor*** (that it the first one has not been drawn) and this action is associated to the actor Administrator: this is surely an acceptable hypothesis, but it is not written in the text. In a real situation it is better to avoid drawing this action and to ask the stakeholder for additional details.

1c

Enrich the model of the Vending Machine with one **scenario** describing a successful sales procedure.

POSSIBLE SOLUTION

Scenario name: Customer successful buying operation
Precondition: the customer has a card (valid) he inserts it, the machine has something to sell and the customer has enough credit on his card.

Step	Description
1	Customer chooses an item
2	The machine find that the item is available
3	Customer pays the item
4	Machine releases the item
5	Machine releases the card
6	Customer picks the product and the card

Postcondition: The sold item is no more in the machine, the user card has been updated, the total Revenue has been updated and the number of sold item for that type is incremented.

SUGGESTIONS

Be as precise as possible: sentences like “there is credit on the card” may lead to misunderstandings and may be considered wrong.

Be detailed (especially in preconditions and postconditions): for example, in this specific situation:

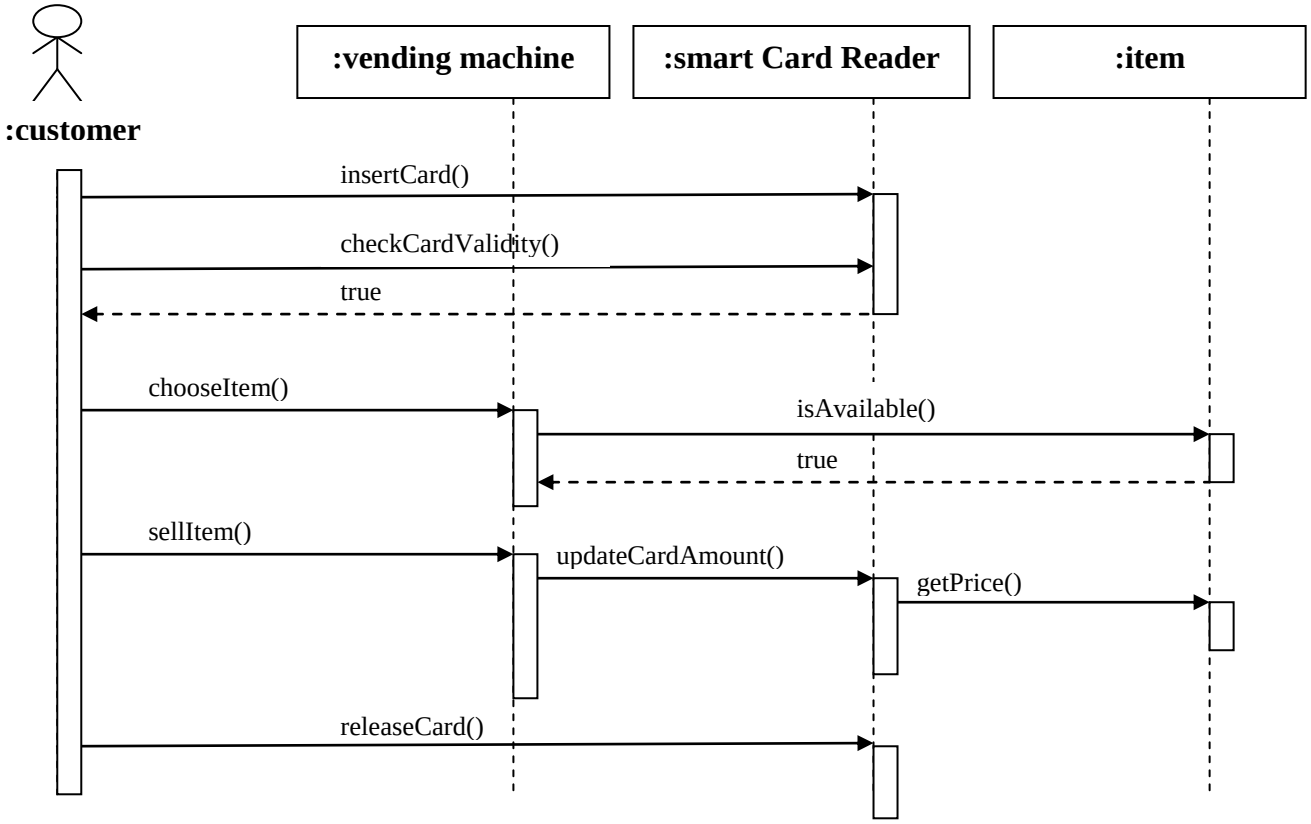
- put the statistics updating (update total revenue…) in the postcondition.
- put the condition “the machine has something to sell” in the precondition.

1d

Enrich the model of the Vending Machine with a **sequence diagram** describing a successful sales procedure.

POSSIBLE SOLUTION

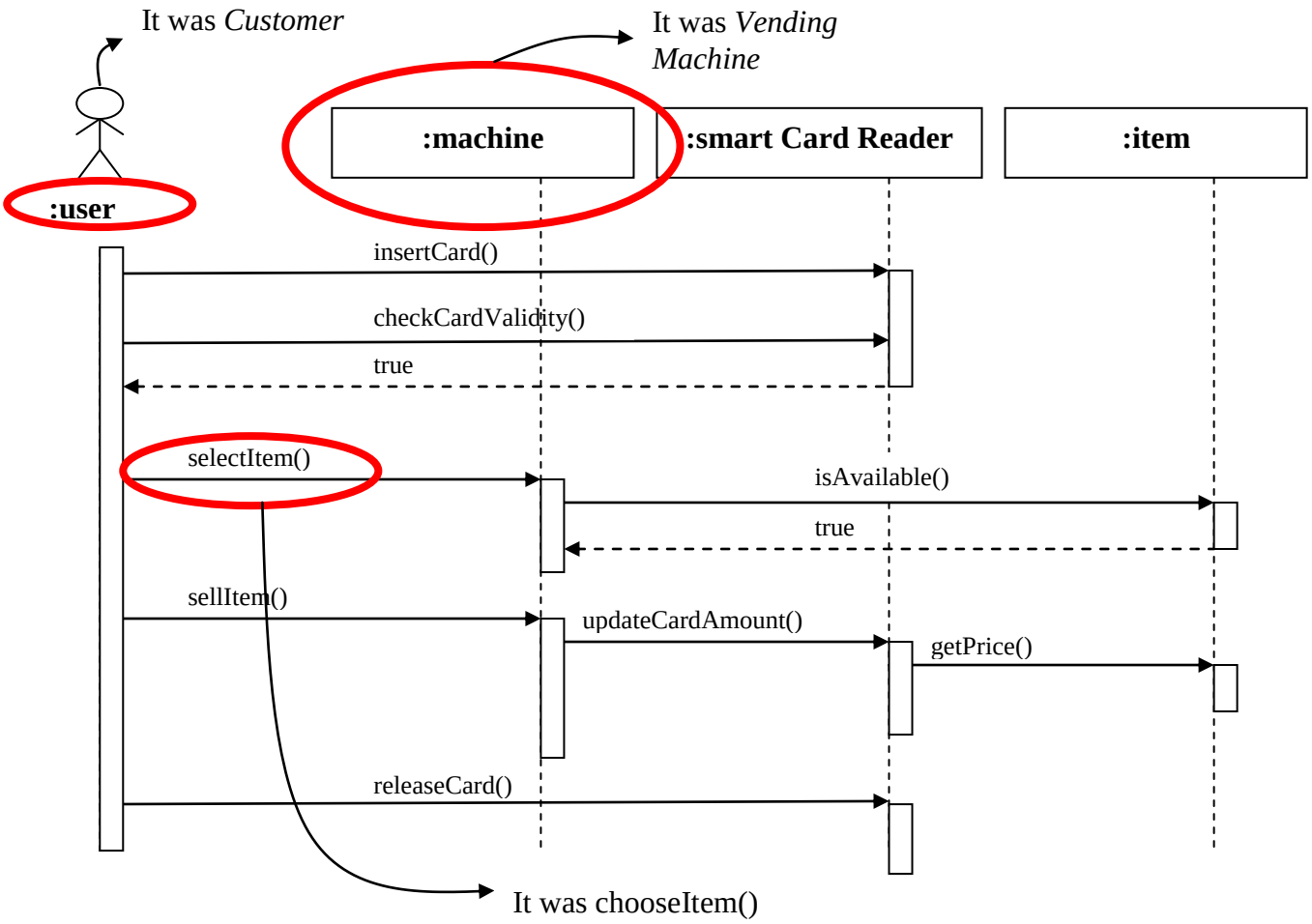
ATTENTION: Don't confuse sequence diagrams with state diagrams (it seems a trivial suggestion but such mismatch happened!)



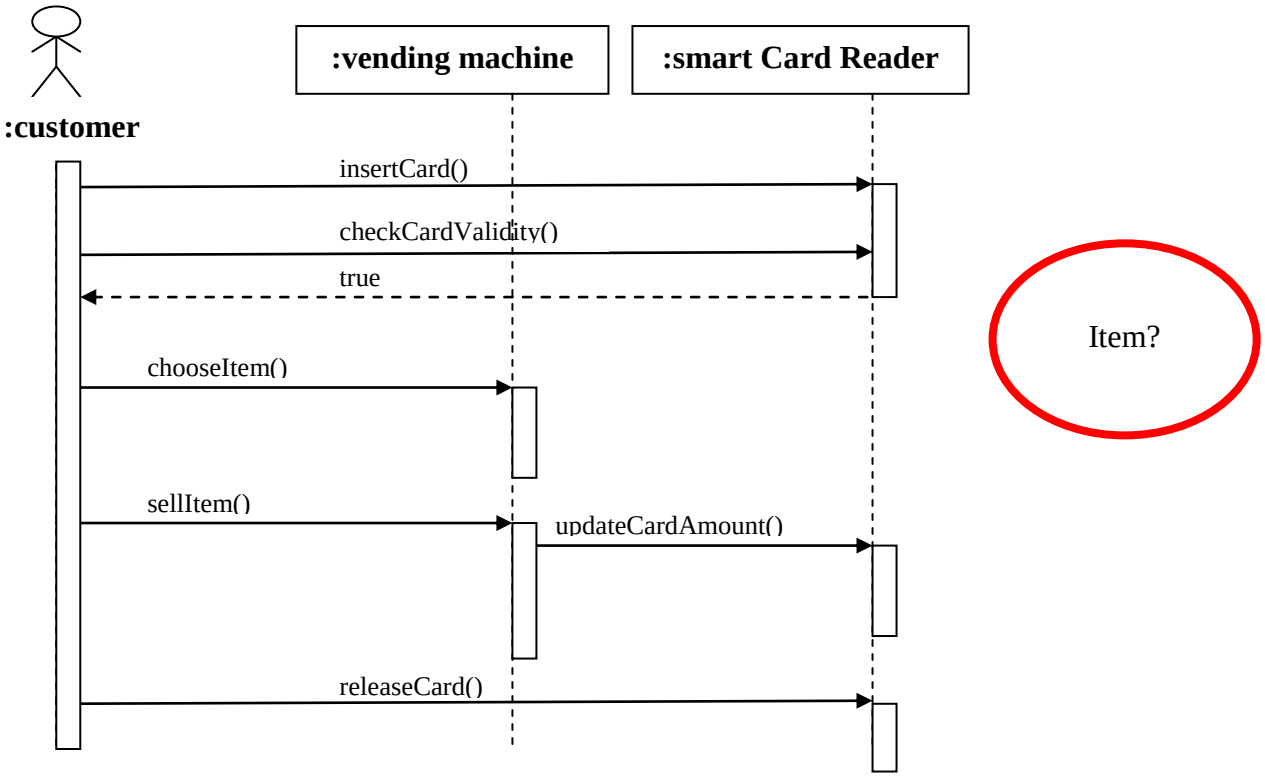
COMMON ERRORS AND SUGGESTIONS

1)

The commonest mistake is to **use different names than the one used in the use case diagram or in the class diagram**.
For example:

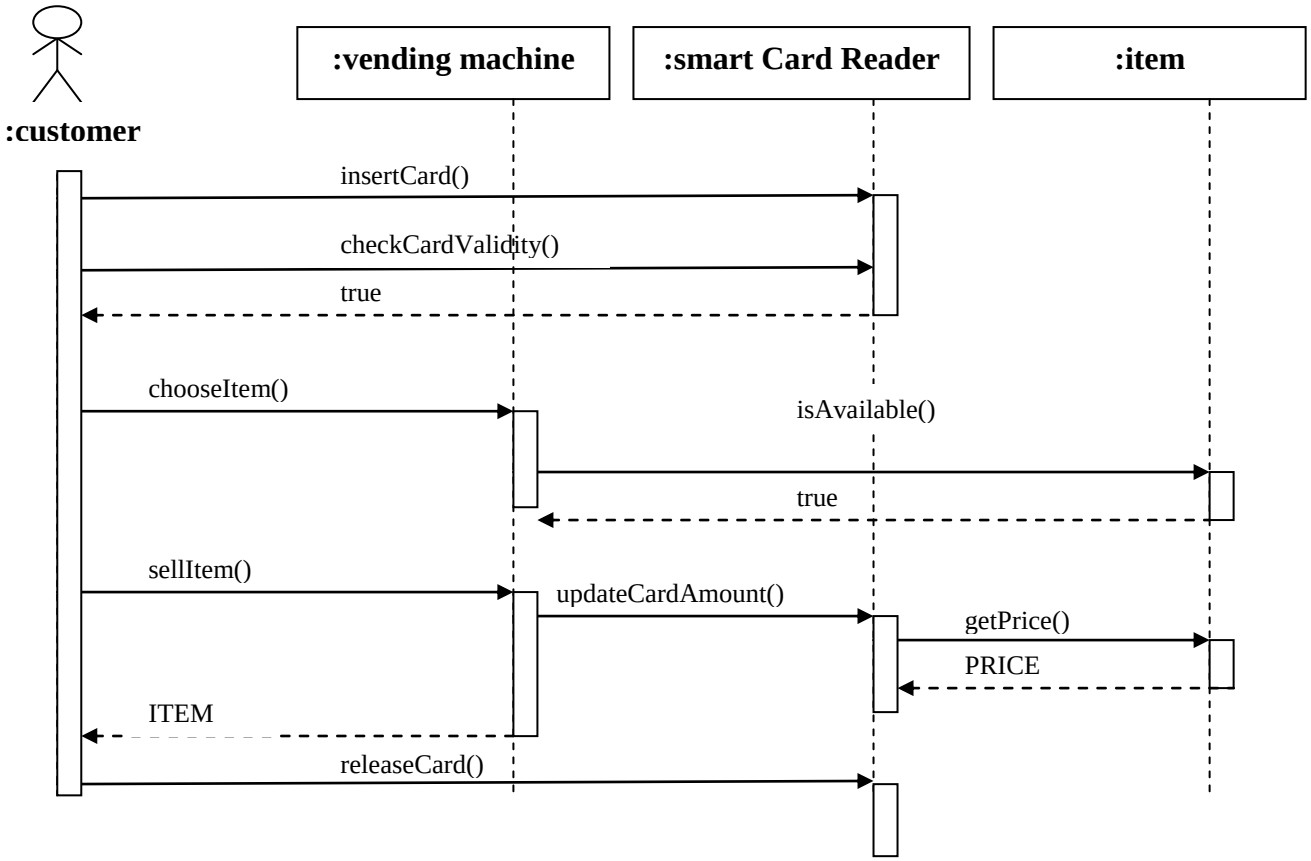


2)



This solution is **wrong** because it doesn't take into account the class Item. This class must be inserted because otherwise we're not able of saying if the Item is available and most of all we have no way of saying which is its price.

3)



This solution is very similar to the first one and of course it is **correct.** It has a negligible problem that is the fact that it **contains some useless return values** such as *PRICE* and *ITEM*.
Since is obvious that the functions returns such values it's better to avoid putting them in order to have a clearer diagram. The suggestion is to put return values only when they're really useful and not guessable (usually the **boolean values**, such as true in the picture, **must be put** because they're fundamental to easily understand the diagram).