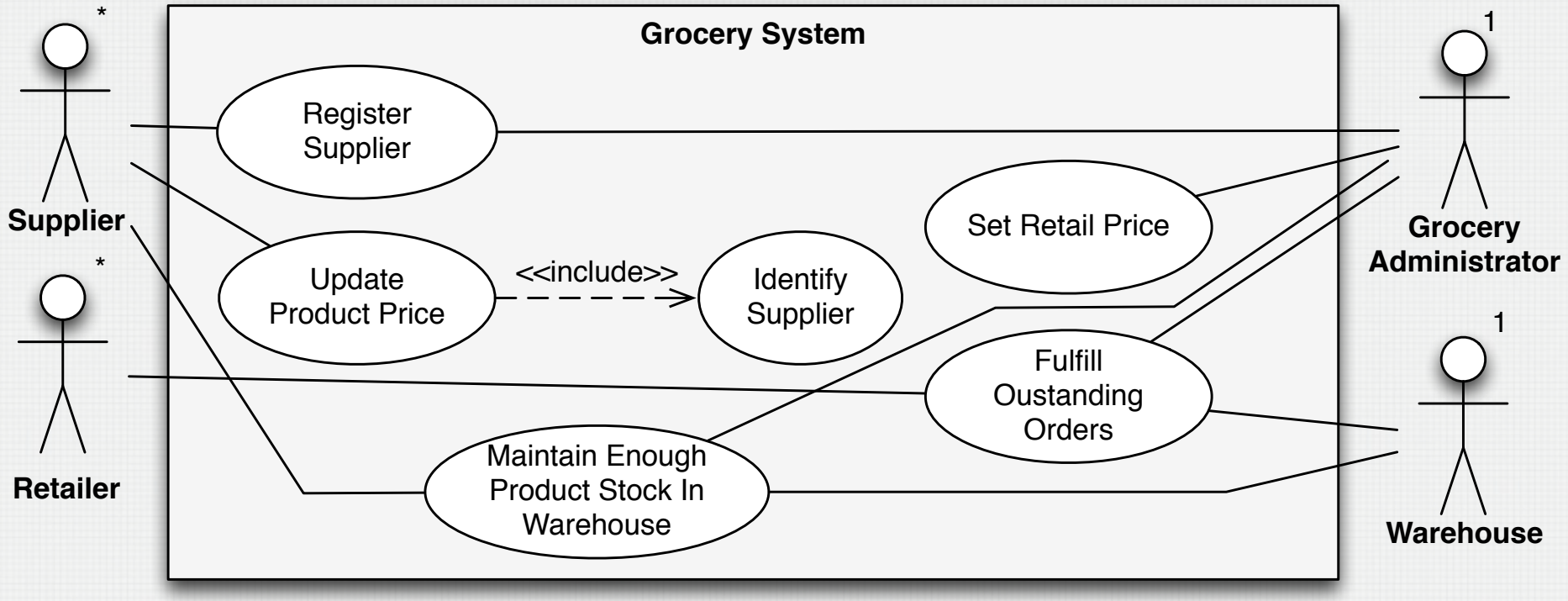
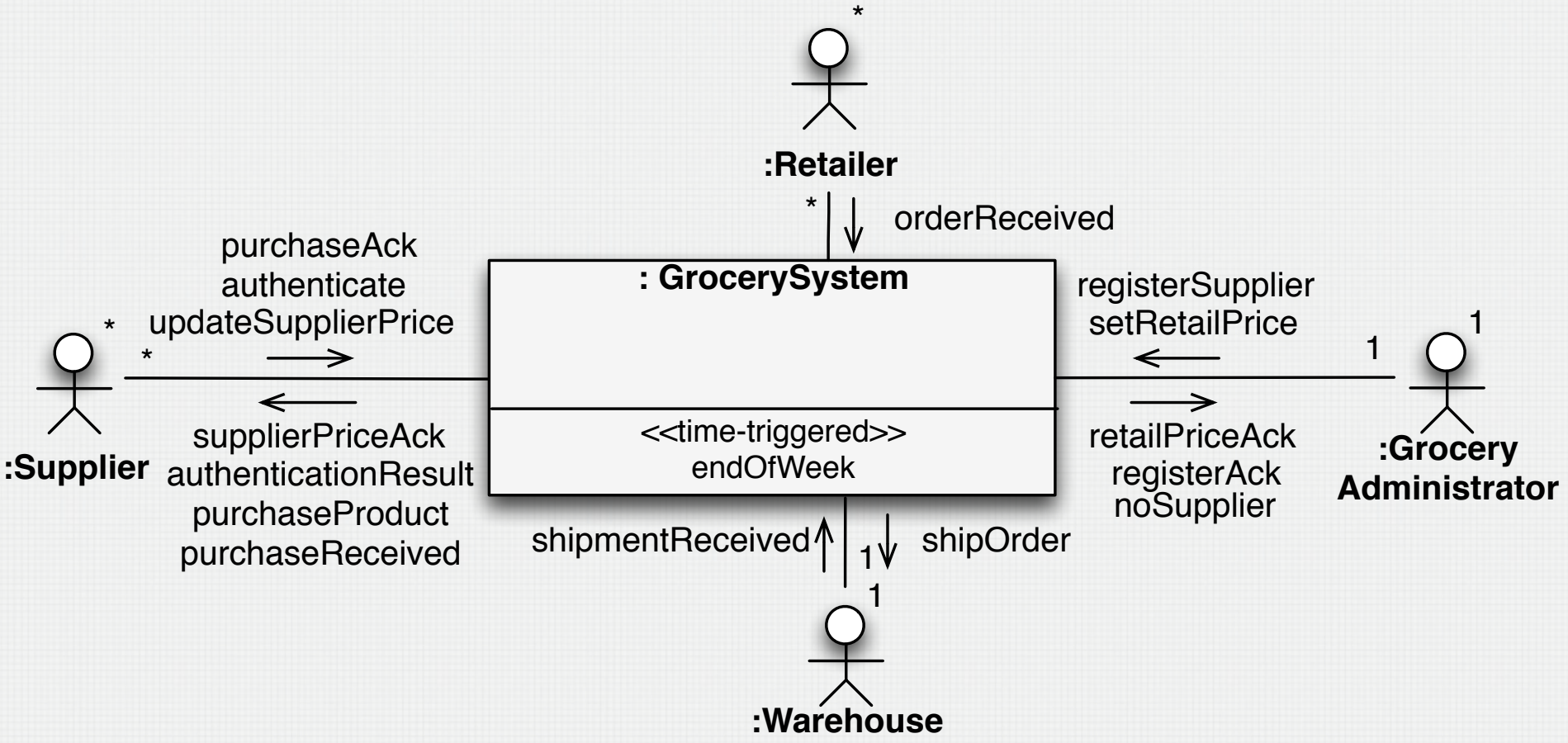
# Grocery Cooperative Design

Jörg Kienzle

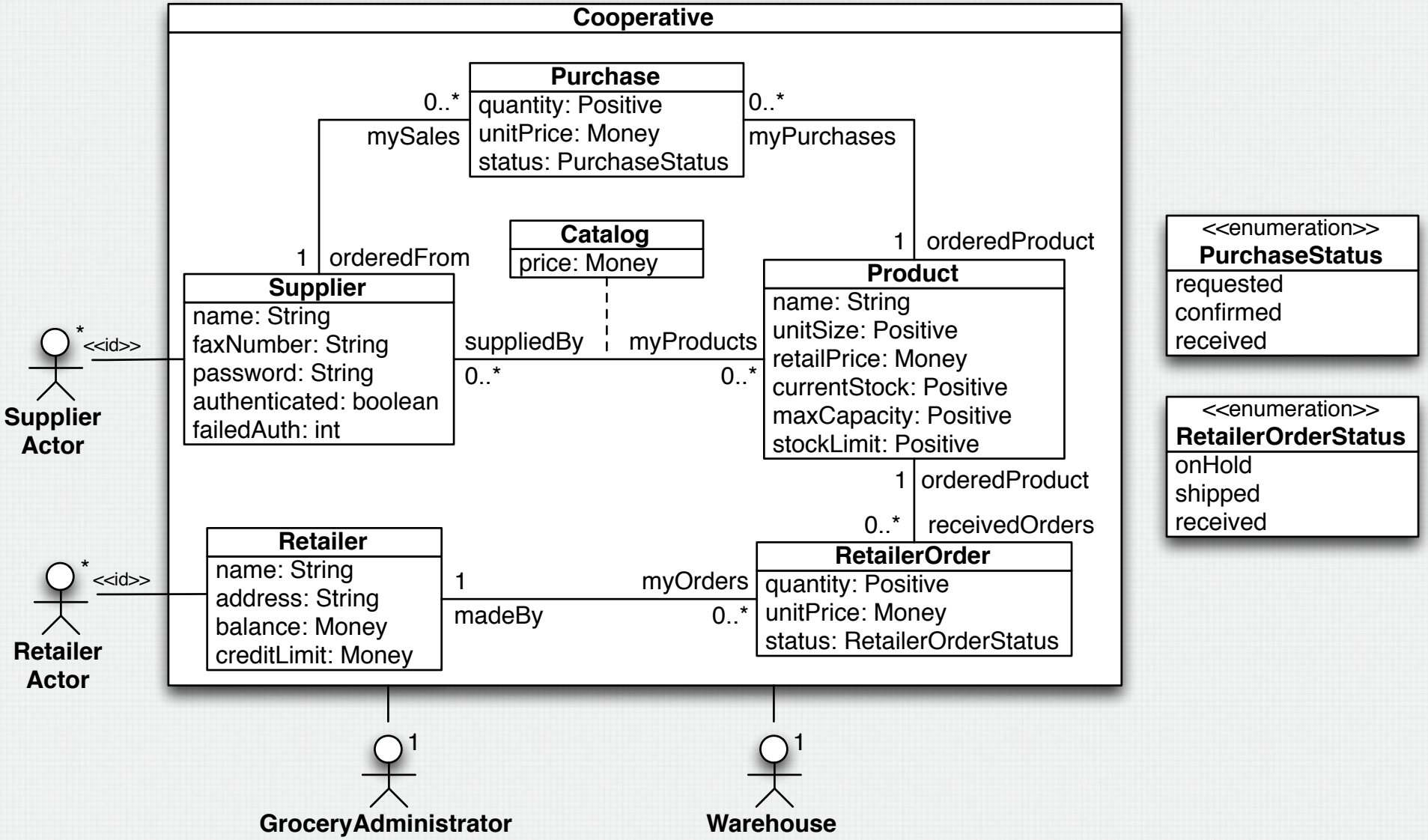
# Grocery Use Case Model



# Grocery Environment Model

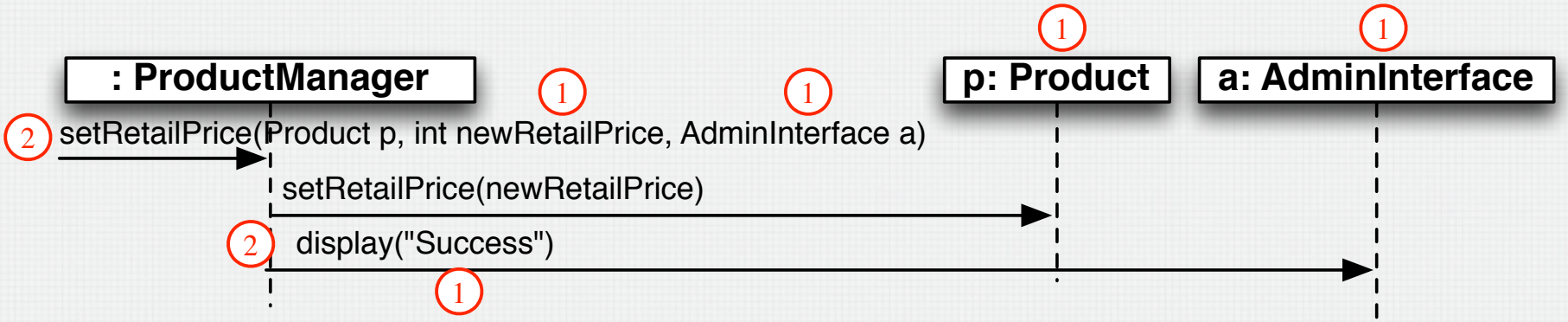


# Concept Model



# SetRetailPrice Design

**Operation**: GrocerySystem::setRetailPrice(p: Product, newPrice: Money)
**Scope**: Product
**Messages**: GroceryAdministrator::{RetailPriceAck}
**Description**: This system operation changes the current price of product p to *newPrice*, and then sends a positive acknowledgment to the administrator.



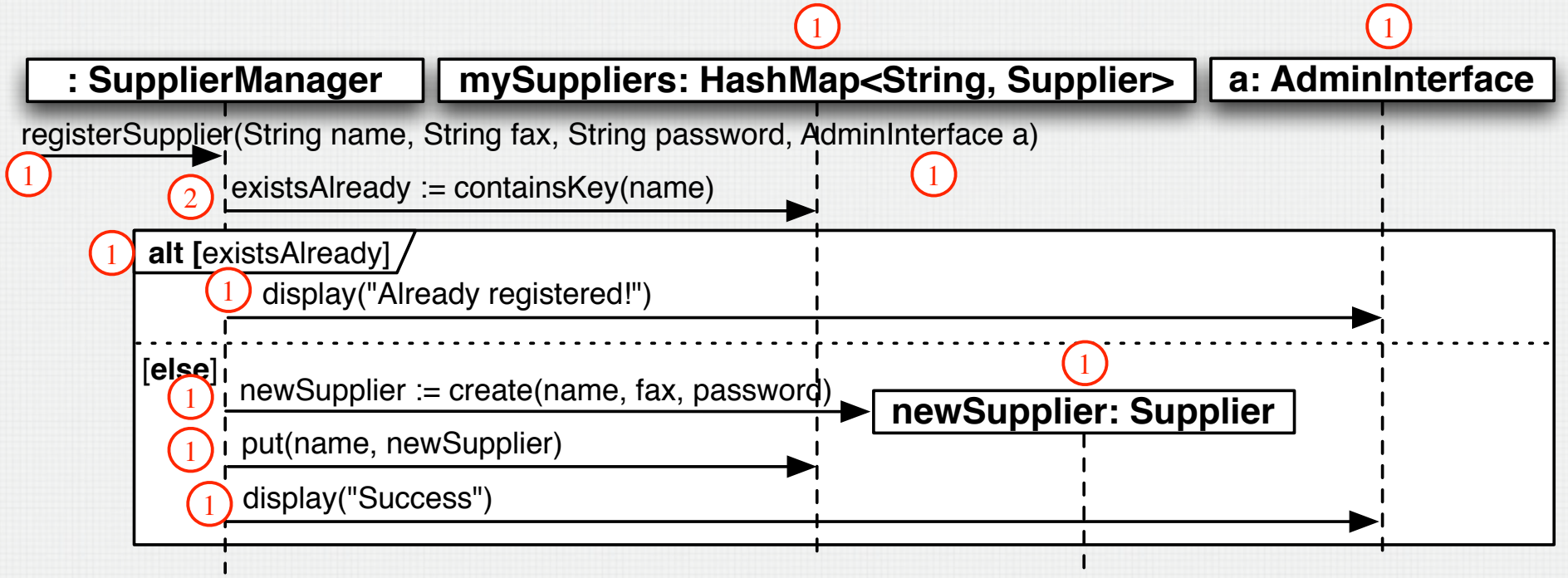

1

*ProductManager* is introduced as a controller. If *Product* would be used, then *Product* would have a parameter and call dependency on *AdminInterface*.

# RegisterSupplier Design

**Operation:** GrocerySystem::registerSupplier(name: String, faxNumber: String, password: String)
**Scope**: Supplier
**New**: newSupplier: Supplier
**Messages**: GroceryAdministrator::{RegisterAck}
**Description**: The effect of this operation is to check whether the system already has a supplier registered under the name name. If yes, a negative *registerAck* message is sent to the *GroceryAdministrator*. If not, a new *supplier* instance is created, its attributes are initialized and a positive *registerAck* message is sent to the *GroceryAdministrator*.

# RegisterSupplier Design

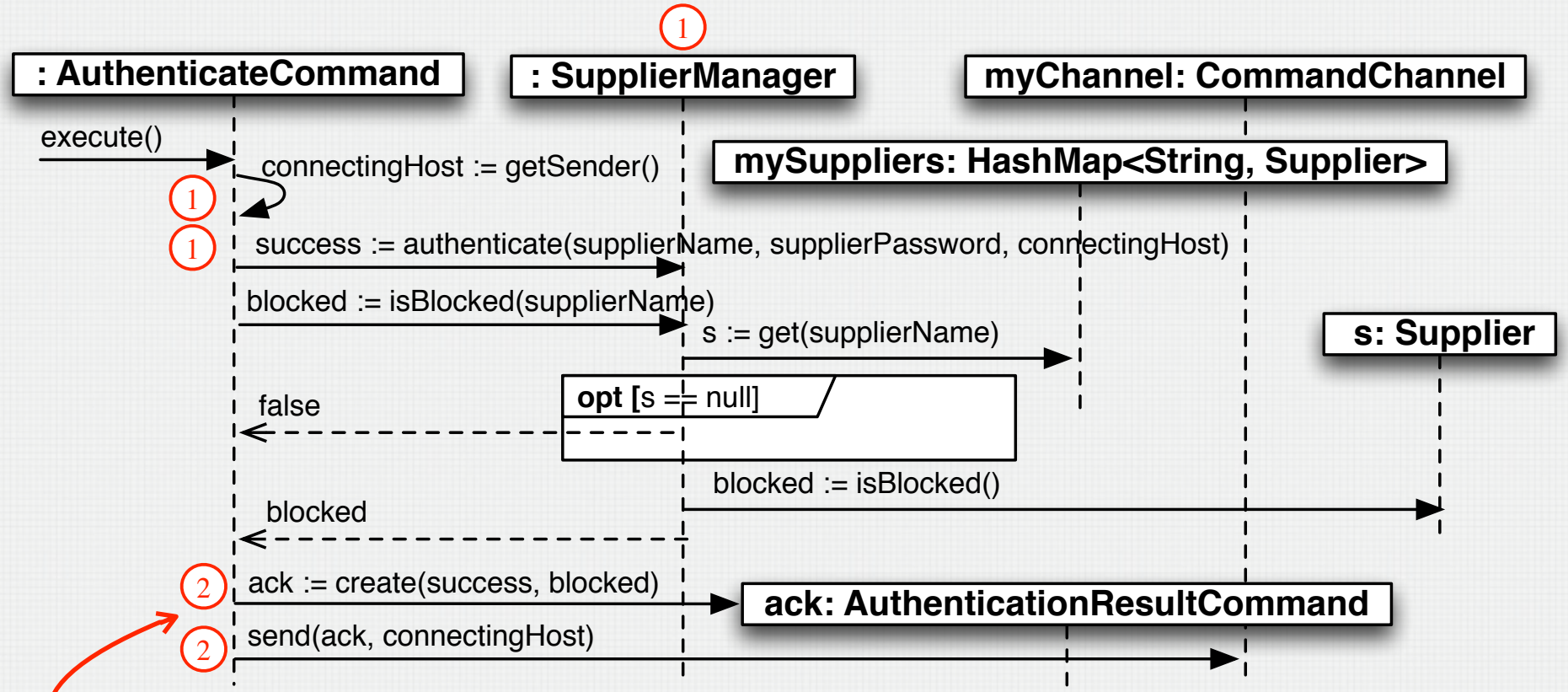



- The controller *SupplierManager* manages/creates *Suppliers*.
- No adaptation of *SetRetailerPrice* design is necessary.

# Authenticate Design

**Operation:** GrocerySystem::authenticate(name: String, password: String)
**Scope**: Supplier
**Messages**: SupplierActor::{AuthenticationResult}
**Description**: If the *name* and *password* match the data stored for a supplier in the system, then authentication is successful. In this case, the system remembers that the supplier has authenticated and sends a positive *AuthenticationResult* message back to the supplier. If authentication is unsuccessful, then a negative *AuthenticationResult* message is sent. Furthermore, if 3 consecutive authentication attempts for the same supplier were unsuccessful, then the supplier's account is blocked.

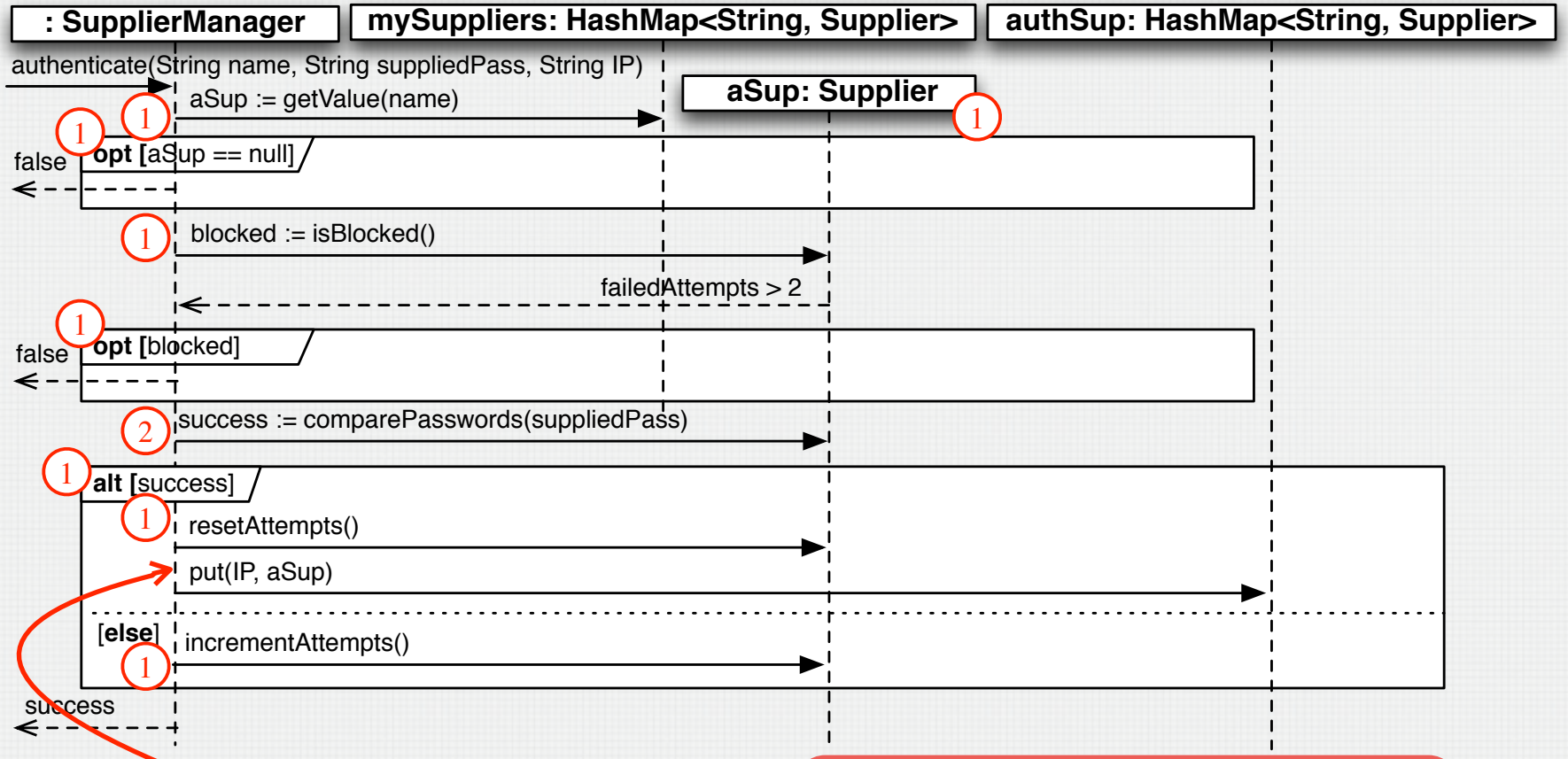
# Authenticate Communication Design




Points counted for 2.1.2

+ 2 points for AuthenticateCommand class with username and password String attributes
+ 1 point for AuthenticationResultCommand with success and blocked boolean attributes

# Authenticate Design




Only needed for 2.2.2

+ 4 points for creating and sending the AuthenticationResultCommand

1

- The controller is the *SupplierManager*, since it already deals with suppliers.

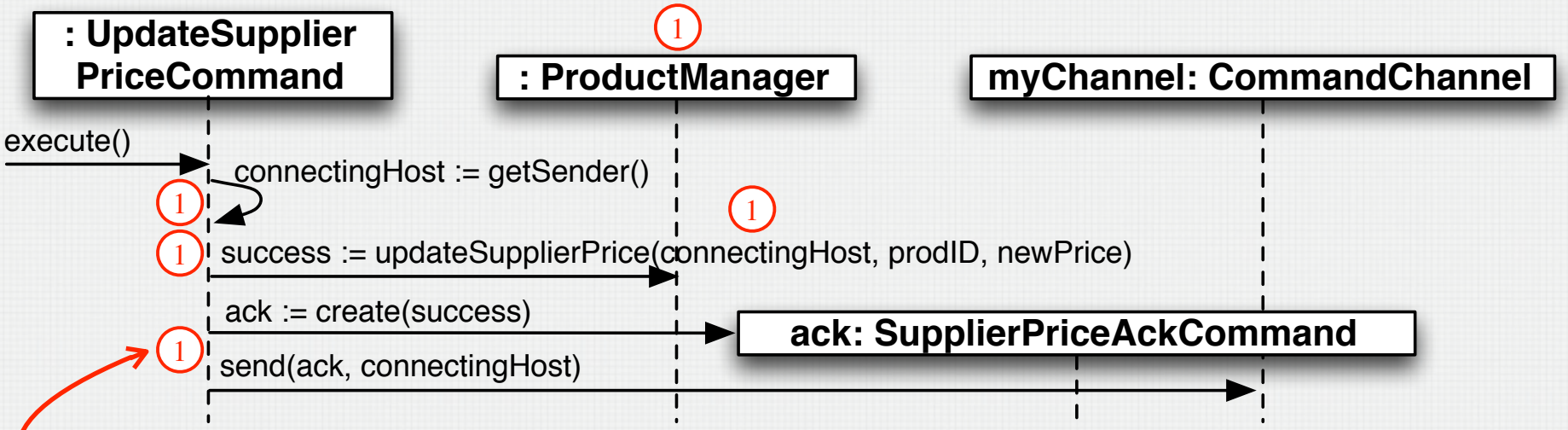
# RegisterSupplier Changes

- No changes needed. *RegisterSupplier* already inserts new suppliers into the *mySuppliers* collection maintained by the *SupplierManager*. ②

# UpdateSupplierPrice Design

**Operation:** GrocerySystem::updateSupplierPrice(p: Product, newSupplierPrice: Money)
**Scope**: Product, Supplier, Catalog
**New**: newCatalogEntry: Catalog
**Messages**: SupplierActor::{SupplierPriceAck}
**Description**: If the calling supplier has not previously successfully authenticated, this operation sends a negative acknowledgement reply to the calling supplier. In case the supplier is authenticated, the effect of this operation is to update the price in the catalog for the supplier invoking the operation and the product *p* to *newSupplierPrice*. In case no such catalog entry exists, a new catalog entry is created. If *newSupplierPrice* is 0, it means that the supplier does not produce product *p* anymore, and therefore the corresponding catalog entry is deleted. Finally, the system sends a positive acknowledgement message to the supplier that invoked the operation.

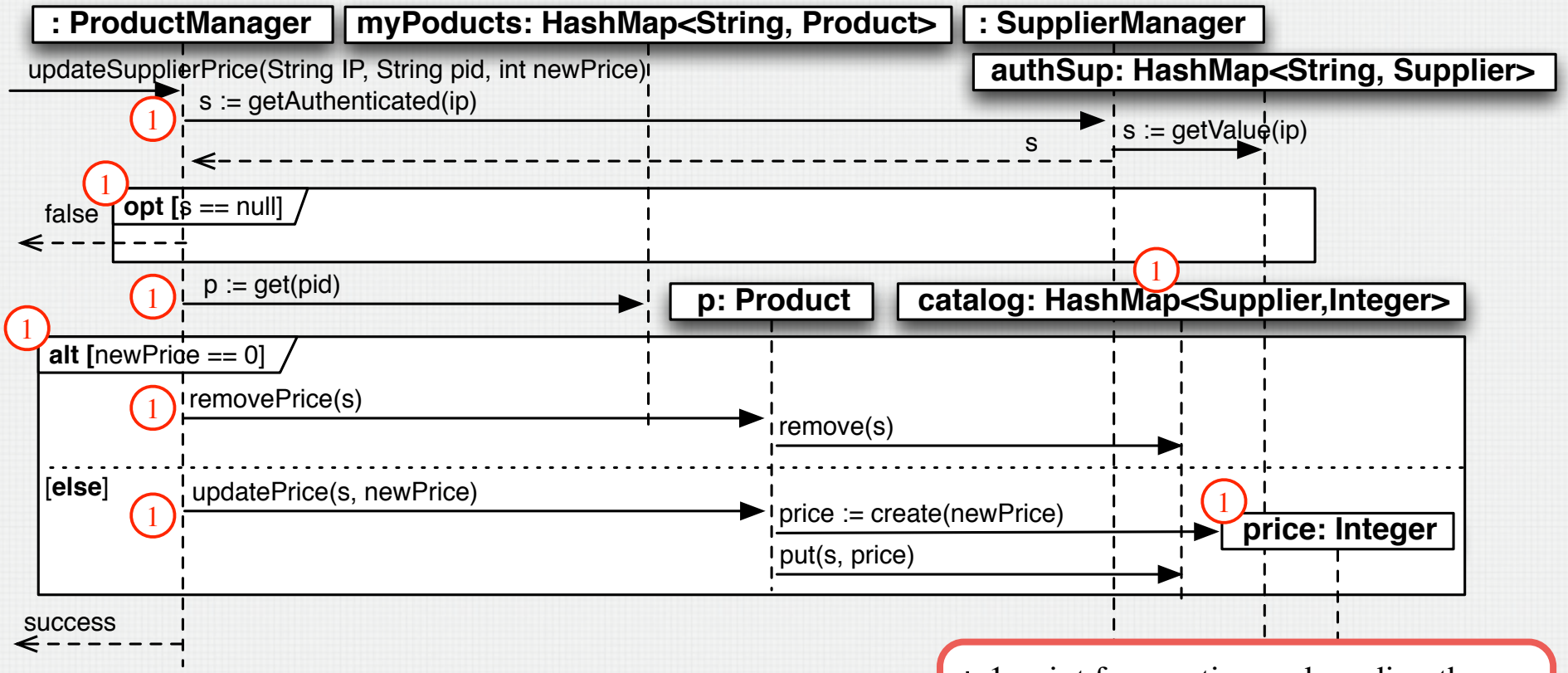
# UpdateSupplierPrice Communication Design




Points counted for 2.2.2

+ 1 point for UpdateSupplierPriceCommand class with product ID and new price attributes
+ 1 point for SupplierPriceAckCommand with success attribute

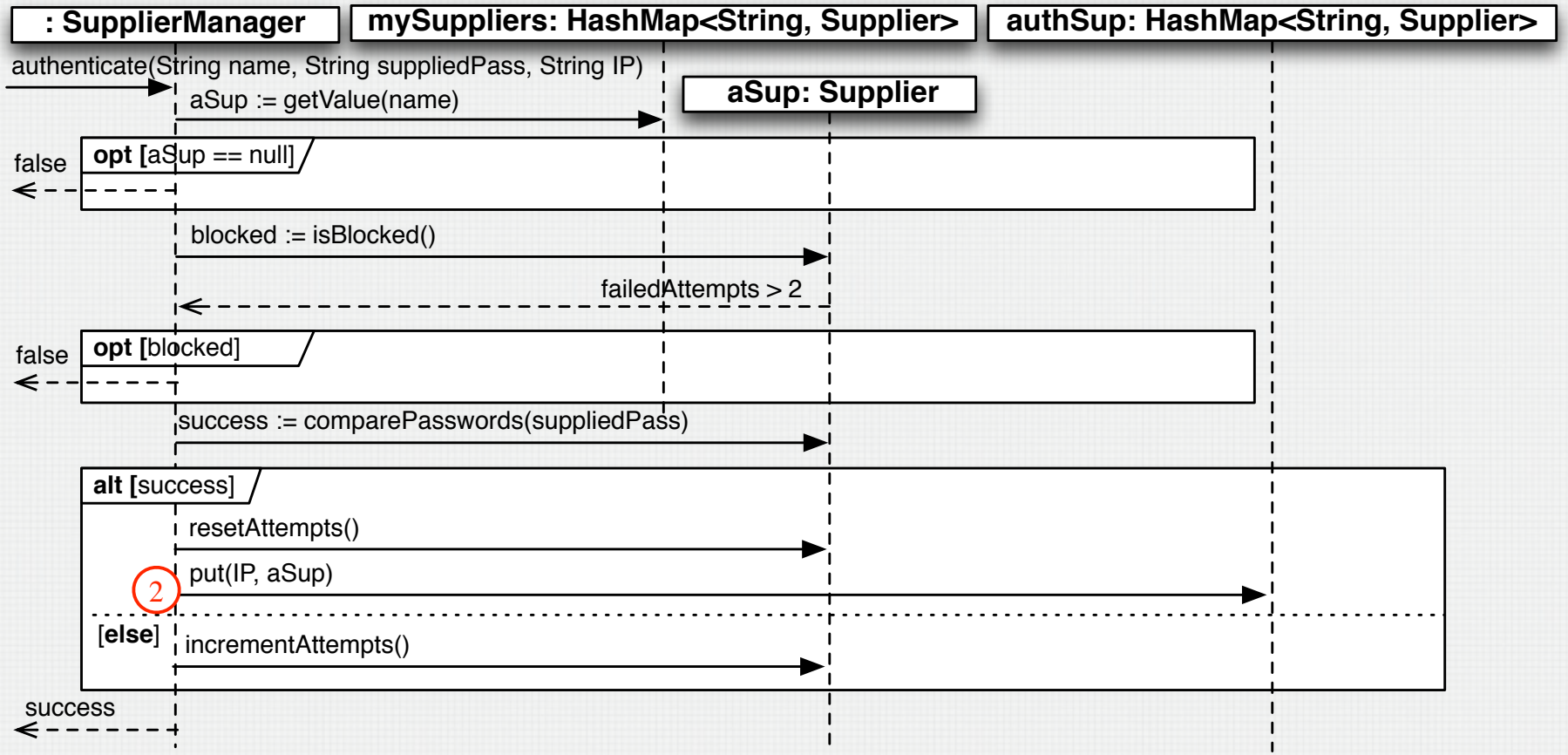
# UpdateSupplierPrice Design




+ 1 point for creating and sending the SupplierPriceAckCommand

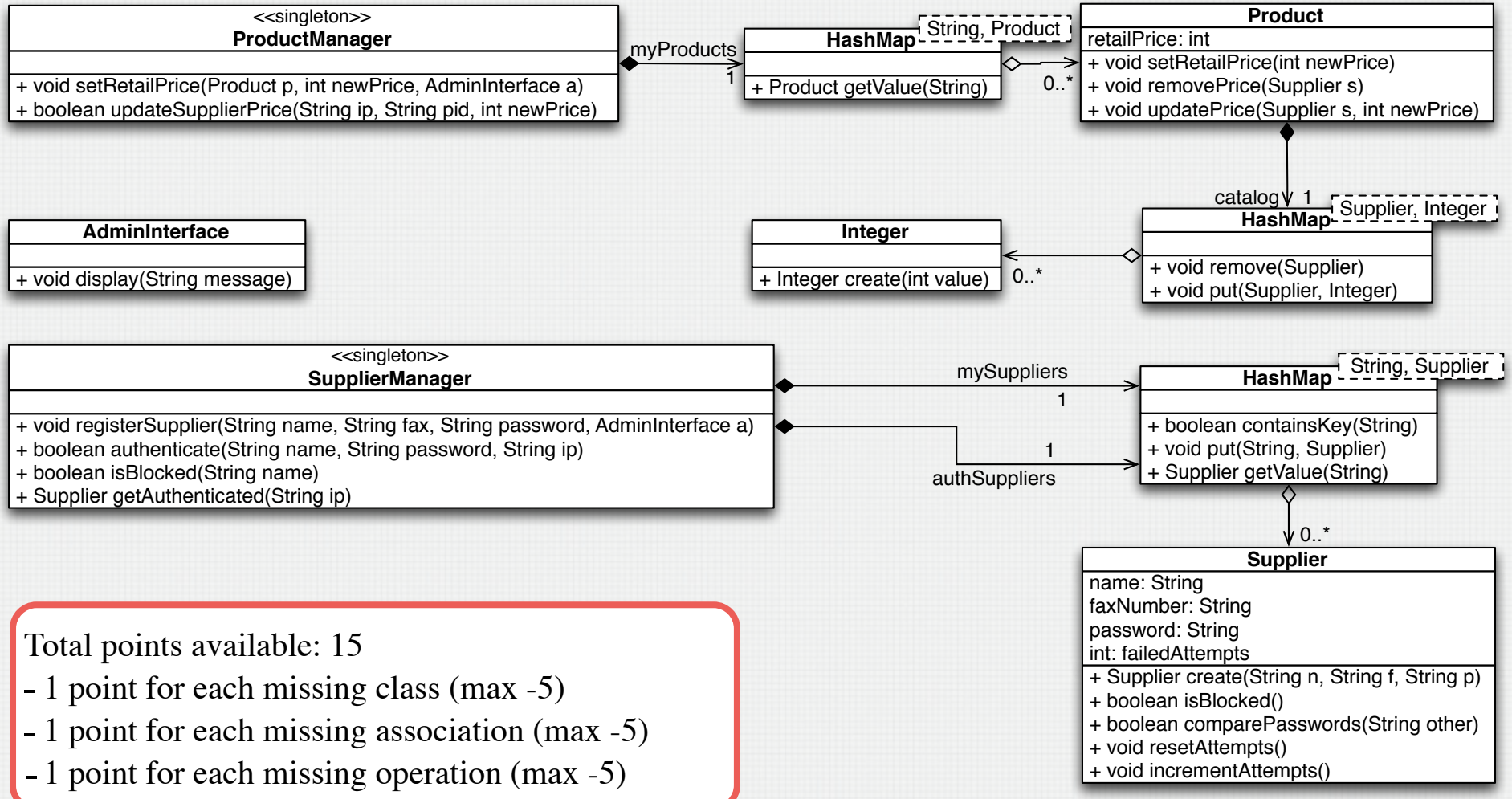
- The controller is the *ProductManager*, since we need to find the product based on its ID, and it is the product that stores the catalog. (1)

# Authenticate Changes




- Authenticate needs to remember the hosts that have successfully authenticated.

# Grocery Design Class Model

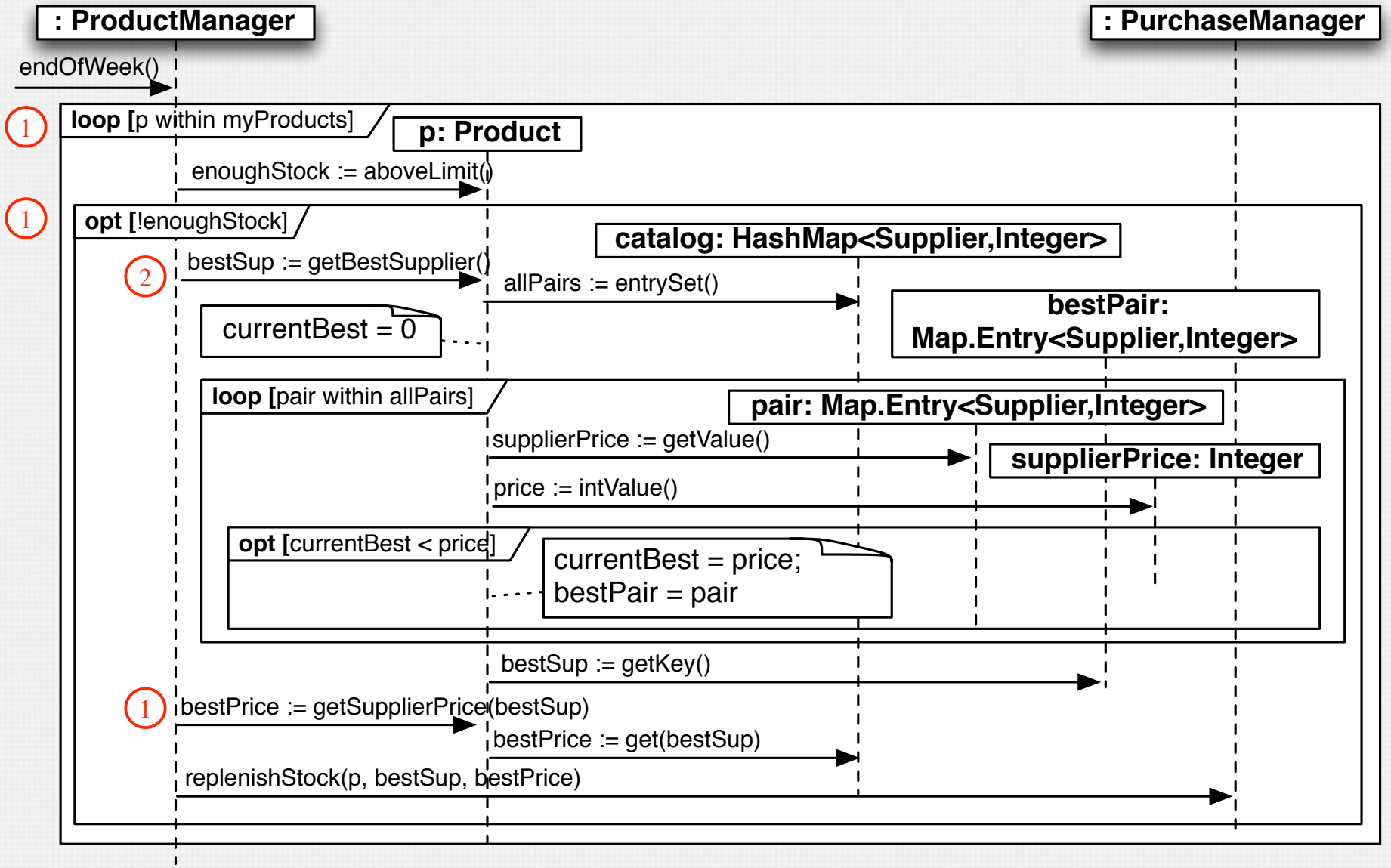



Total points available: 15
- 1 point for each missing class (max -5)
- 1 point for each missing association (max -5)
- 1 point for each missing operation (max -5)

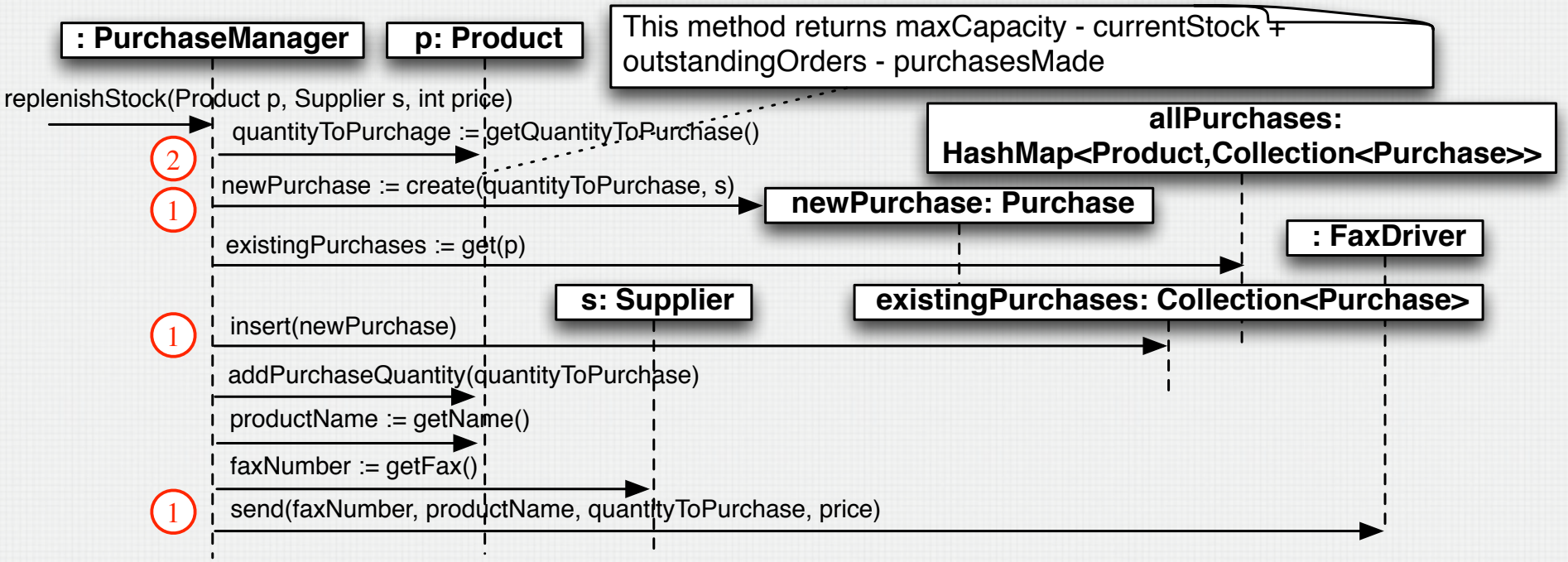
# Design Dependency Analysis

- Which objects in your design are system-wide objects (also known as singletons)?
  - ProductManager
  - SupplierManager
- List all the classes that are responsible for instantiating objects of other classes.
  - SupplierManager instantiates Supplier
  - Product instantiates Integer
  - AuthenticateCommand instantiates AuthenticationResultCommand
  - UpdateSupplierPriceCommand instantiates SupplierPriceAckCommand
- Do any of your business classes (i.e., the classes that represent classes in the concept model) depend on classes that implement communication with the outside world?
  - No, in our design only manager classes communicate with the outside world by calling the *AdminInterface* class, and the *RemoteCommand* subclasses call the communication channel class of the network interface.

# EndOfWeek Communication Design (1)



# EndOfWeek Communication Design (2)



# Complete Design Class Diagram

Total points available: 5
- 1 point for each missing class (max -2)
- 1 point for each missing association (max -1)
- 1 point for each missing operation (max -2)