UC2: Update Discount Policy Group 1

**Deliverables** -

1. Produce a description of the business process for the use case.

A **Discount Policy Editor** is a tool that allows users to easily create, modify, and manage discount rules for products or services. It provides the ability to set various types of discounts, such as percentages, fixed amounts, or buy-one-get-one offers, and apply them based on different conditions like product categories, customer groups, or purchase amounts. Users can control whether discounts can be combined or if only one should apply and set start and end dates to manage when discounts are active. The editor also includes options for testing and previewing discount rules to ensure they work as intended. With access controls, only authorized users can make changes, making it a secure and flexible solution for managing discount strategies.

2. Do domain model brainstorming on the description produced above, classify the brainstorming results into classes, attributes of classes and relationships between classes, and visualize the classification results in a UML class diagram (which must not show any operations).

The team members get together and identify & list

(*domain specific, domain specific, domain specific*)

**1**. **Nouns / noun phrases**

* Discount rule - Class
* Product Category - Class
* Customer Groups - Class
* Purchase Amounts - attribute
* Preview Discount
* Discount Condition - Class

**2.** **"X of Y" expressions (e.g., color of car)**

* Value of discount – attribute of discount class
* Condition of discount – attribute of discount class
* Start Date of discount – attribute of discount class
* End Date of discount – attribute of discount class

**3.** **Transitive verbs (none domain specific)**

* Create
* Modify
* Manage
* Set
* Apply
* Control
* Combined
* Testing
* Previewing
* Track
* Make

**4.** **Adjectives, enumeration**

* Fixed discount (as in "fixed amounts") - attribute of discount rule
* Buy-one-get-one - attribute of discount rule
* Percentage discount - attribute of discount rule
* Active & Inactive (referring to when discounts are in effect)
* Authorized (referring to users with permission to make changes)
* Secure
* Flexible

**5.** **Numeric, quantity**

* Percentages – attribute of discount rule
* Fixed amounts – attribute of discount rule
* Buy-one-get-one offer - attribute of discount rule
* Purchase amount – multiplicity
* Start and end dates – attribute

**6.** **Possession expressions (has/have, possess, etc.)**

* Users have the ability to set various types of discounts – association
* The Editor includes options for testing and previewing discount rules – aggregation
* Discount Rules have conditions that determine their applicability – composition
* Customer Groups possess specific discount eligibilities – association
* Access Controls have permission levels for different user roles – composition

**7.** **Consist of / part of" expressions**

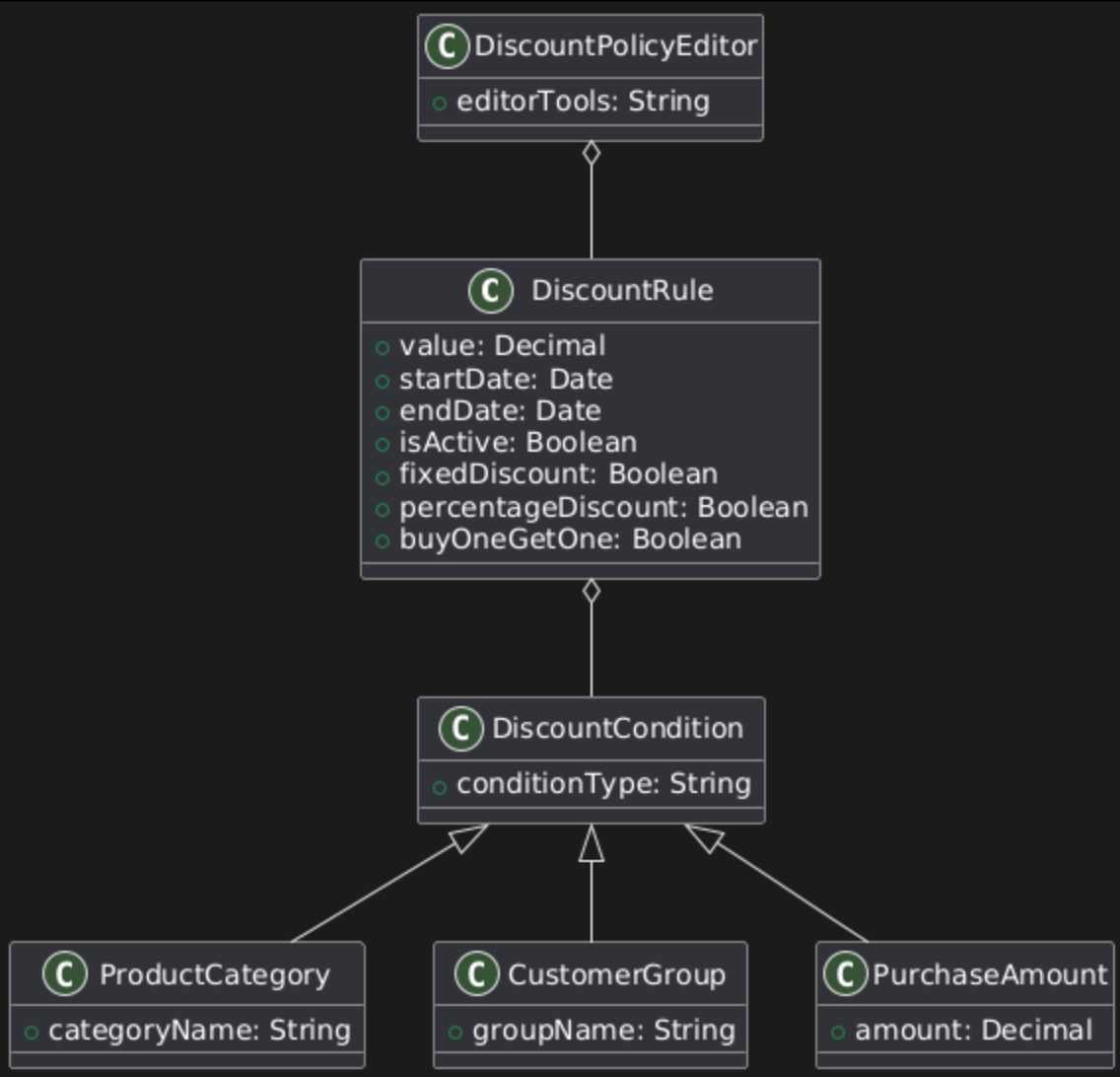
* Discount types include percentages, fixed amounts, or buy-one-get-one offers – aggregation
* Conditions include product categories, customer groups, or purchase amounts – aggregation
* Discount rules have start and end dates to manage when discounts are active – composition
* The Discount Policy Editor consists of tools for creating, modifying, and managing discount rules – aggregation
* Access controls consist of permissions to restrict changes to authorized users – composition

**8.** **Containment / containing expressions**

* DPE consists of tools for creating, modifying, and managing discount rules. - aggregation
* Access controls consist of permissions to restrict changes to authorized users. - association

**9.** **"X is a Y" expressions (generalization/specialization)**  
 \*\* X, Y are nouns/noun phrases

* Product Category is a Discount Condition – inheritance
* Customer Group is a Discount Condition – inheritance
* Purchase Amount is a Discount Condition – inheritance



3. Specify the high-level use case (that is, when and where the use case begins and when the use case ends).

1. TUCBW user clicks on edit rule button
2. TUCEW user sees a confirmation message of edits saved

4. Specify the expanded use case (that is, how an actor will interact with the system to carry out the foreground processing of the use case). Identify the nontrivial step(s) of the expanded use case.

|  |  |
| --- | --- |
|  | |
| Actor: User | System: Discount Policy Editor |
|  | 1. DPE displays discount policies to edit |
| 1. TUCBW user clicks on edit rule button | 1. DPE displays available discount rules |
| 1. User chooses rule(s) to modify | 1. DPE applies changes to rules |
| 1. User clicks save edits | 1. DPE saves edits to document and displays confirmation message (has been successfully updated: |
| 1. TUCEW user sees a confirmation message of edits saved |  |

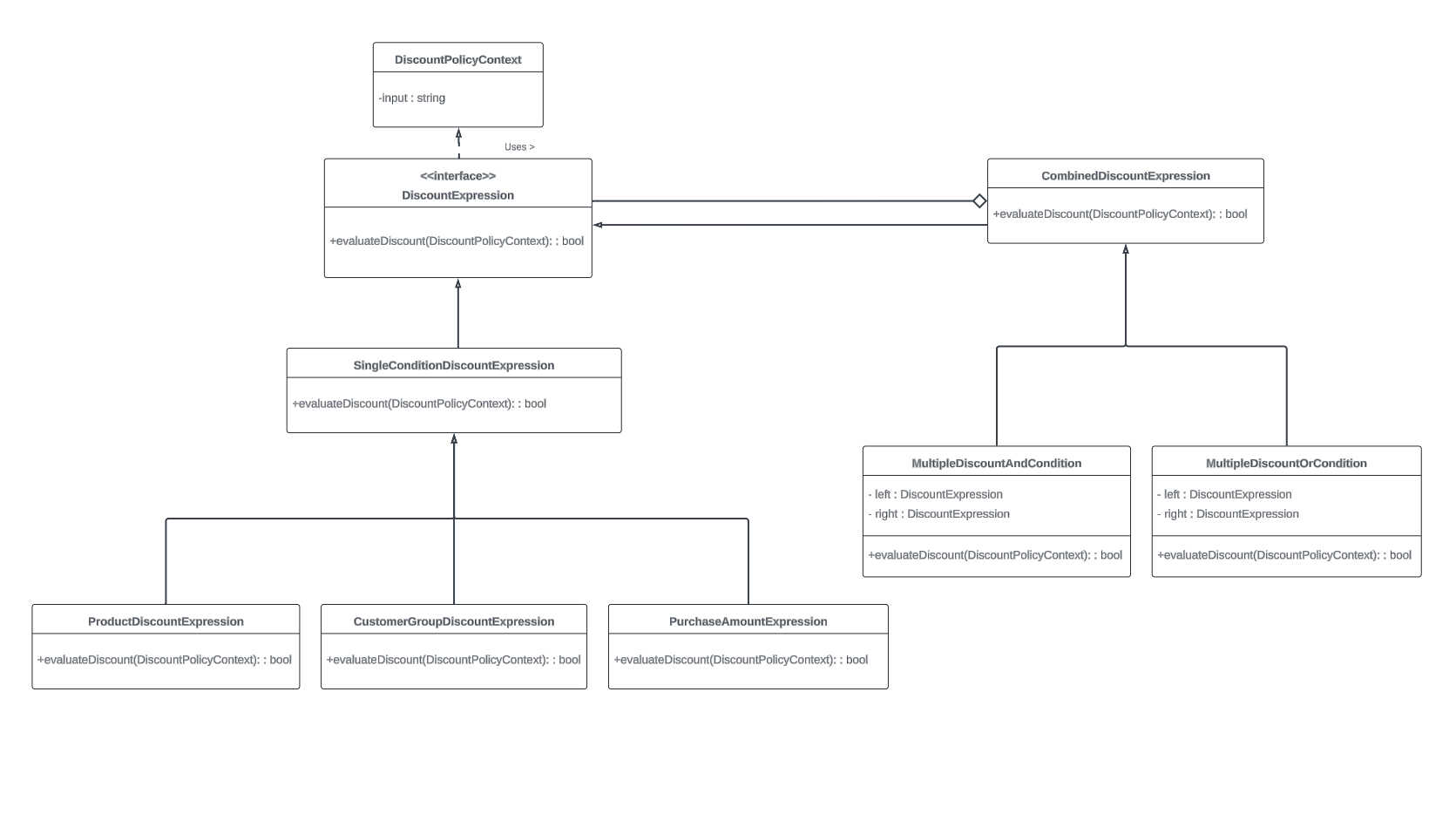
**Non-trivial steps:**

**Step 4:** DCP applies changes to the rules\*

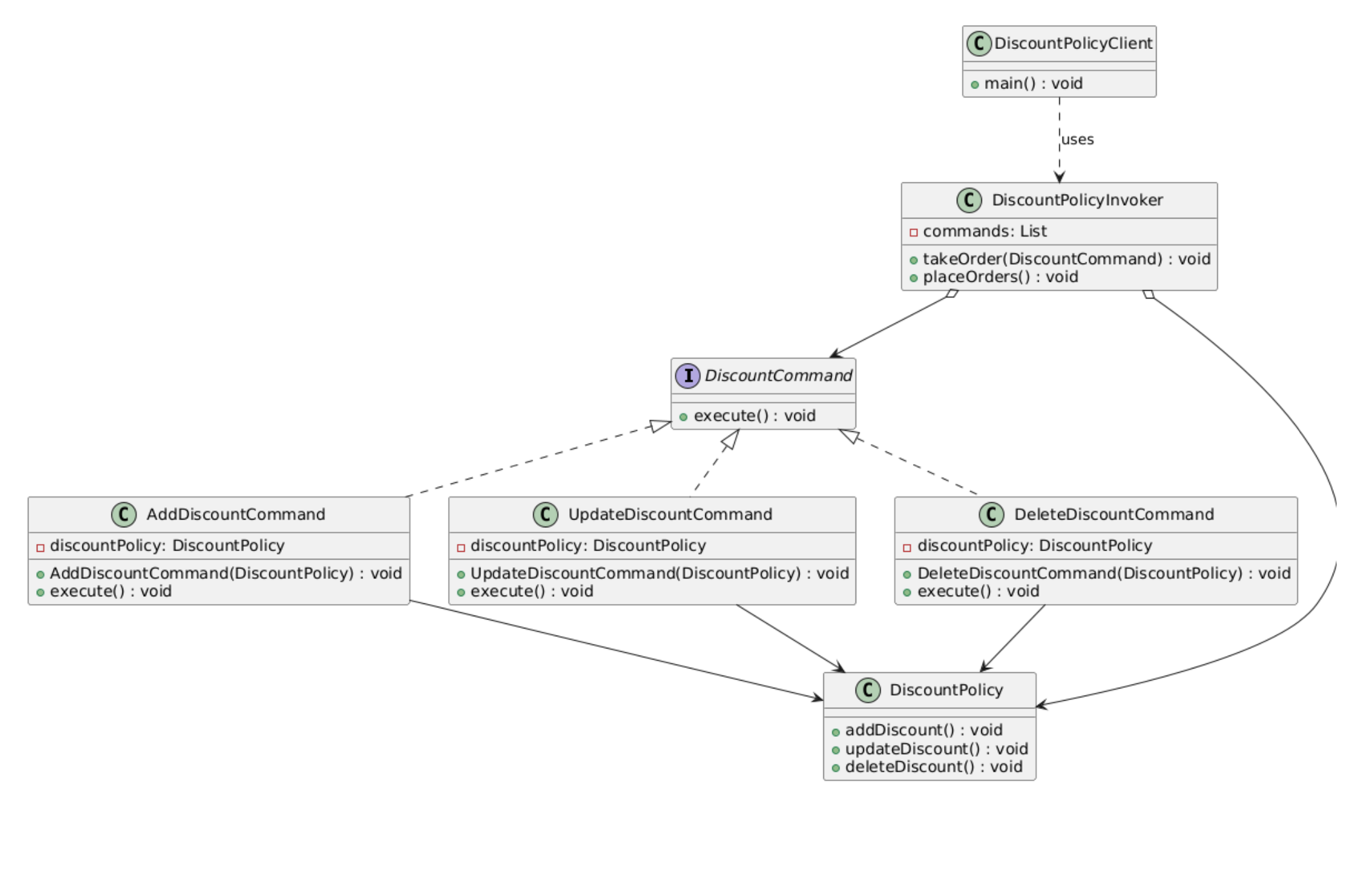
**Step 6:** DCP saves edits to document and displays confirmation message (has been successfully updated) \

5. Apply each of the required Gang of Four (GoF) patterns by copying the structural design of the pattern, and replacing the generic class names, method names and attribute names with application-specific classes, methods and attributes.

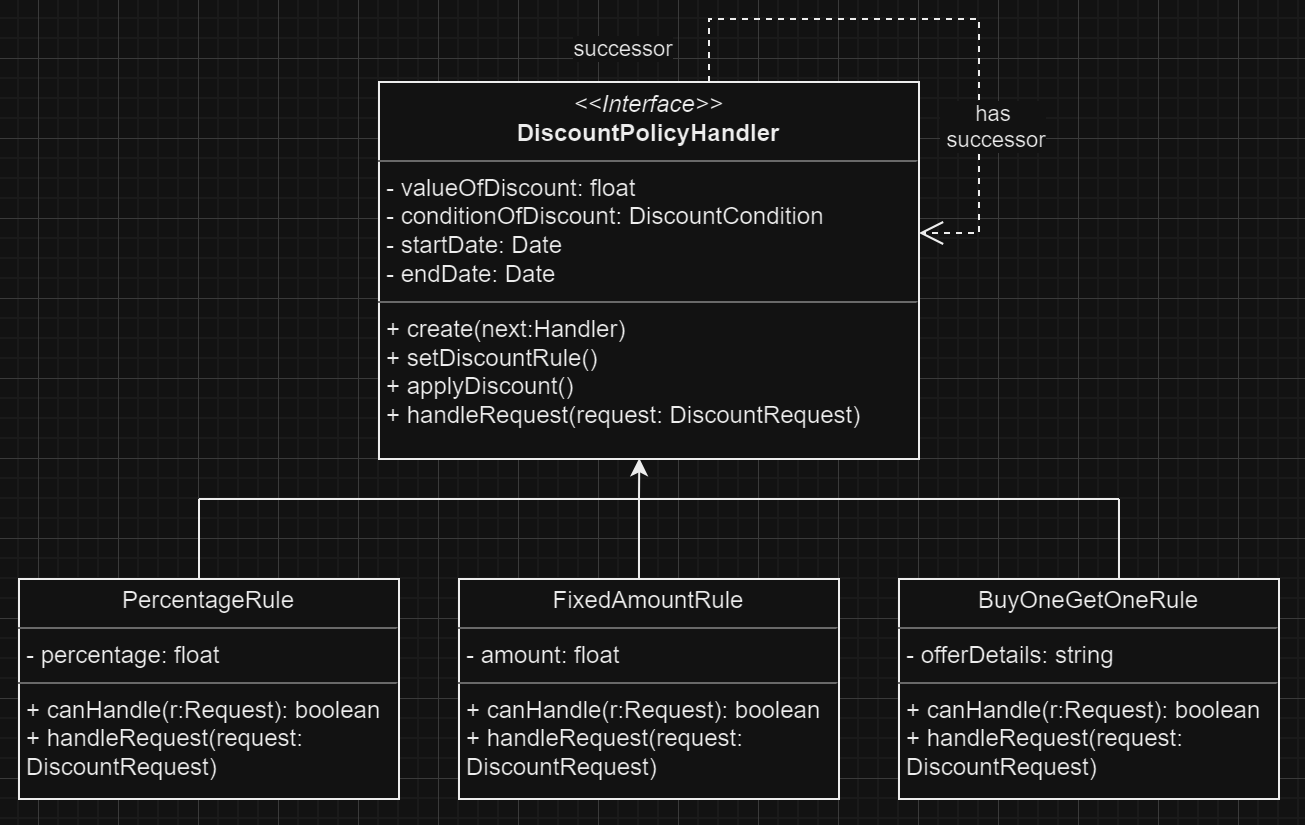
**Interpreter pattern**

Interpreter pattern: [link](https://lucid.app/lucidchart/2abce06b-7782-4ed3-bfa7-e7bb24d7d409/edit?viewport_loc=-1042%2C-858%2C4608%2C2362%2C0_0&invitationId=inv_28e94379-09f7-4956-932d-7da3131db52c)

**Command pattern**



**Chain of Responsibility (COR)**



6. For each of the nontrivial steps of the expanded use case produce a scenario description to describe how software objects interact with each other to process the actor request to produce the system response (as specified in the nontrivial step). The scenario description must also apply the required design patterns including the controller, expert and creator patterns whenever applicable.

**Step 4: Execute Command**

System Action: Execute the command to apply the discount update.

1. Interaction:

* DiscountPolicyController receives the update request and creates an UpdateDiscountCommand.
* CommandInvoker receives the command and passes it to the DiscountRuleInterpreter.
* DiscountRuleInterpreter interprets the discount rule syntax and validates it.
* If valid, the CommandInvoker executes the command, calling DiscountManager.updateDiscount().
* The result passes back through the chain: DiscountManager -> CommandInvoker -> Controller.

1. Design Patterns:

* Command Pattern: Update operation encapsulated in UpdateDiscountCommand.
* Interpreter Pattern: DiscountRuleInterpreter interprets and validates discount rule syntax.
* Chain of Responsibility: Request flows through Controller, CommandInvoker, Interpreter, and DiscountManager.

**Step 6: Finalize the Update**

System Action: Finalize the update, store the discount policy, and return confirmation.

1. Interaction:

* DiscountPolicyController creates a FinalizeUpdateCommand and passes it to the CommandInvoker.
* CommandInvoker executes the command, which triggers the DiscountManager.
* DiscountManager retrieves the policy and applies final validations.
* DiscountManager uses DataStore to save the policy.
* A ConfirmationResponse is created and passed back through the chain.

1. Design Patterns:

* Command Pattern: Finalization encapsulated in FinalizeUpdateCommand.
* Chain of Responsibility: Request flows through Controller, CommandInvoker, DiscountManager, and DataStore.
* Interpreter Pattern: While not directly used in this step, the interpreted rules from Step 4 are part of the policy being finalized.

7. Convert the scenario description to an informal sequence diagram in which the messages between the objects are labeled by English texts.

A diagram of a software project

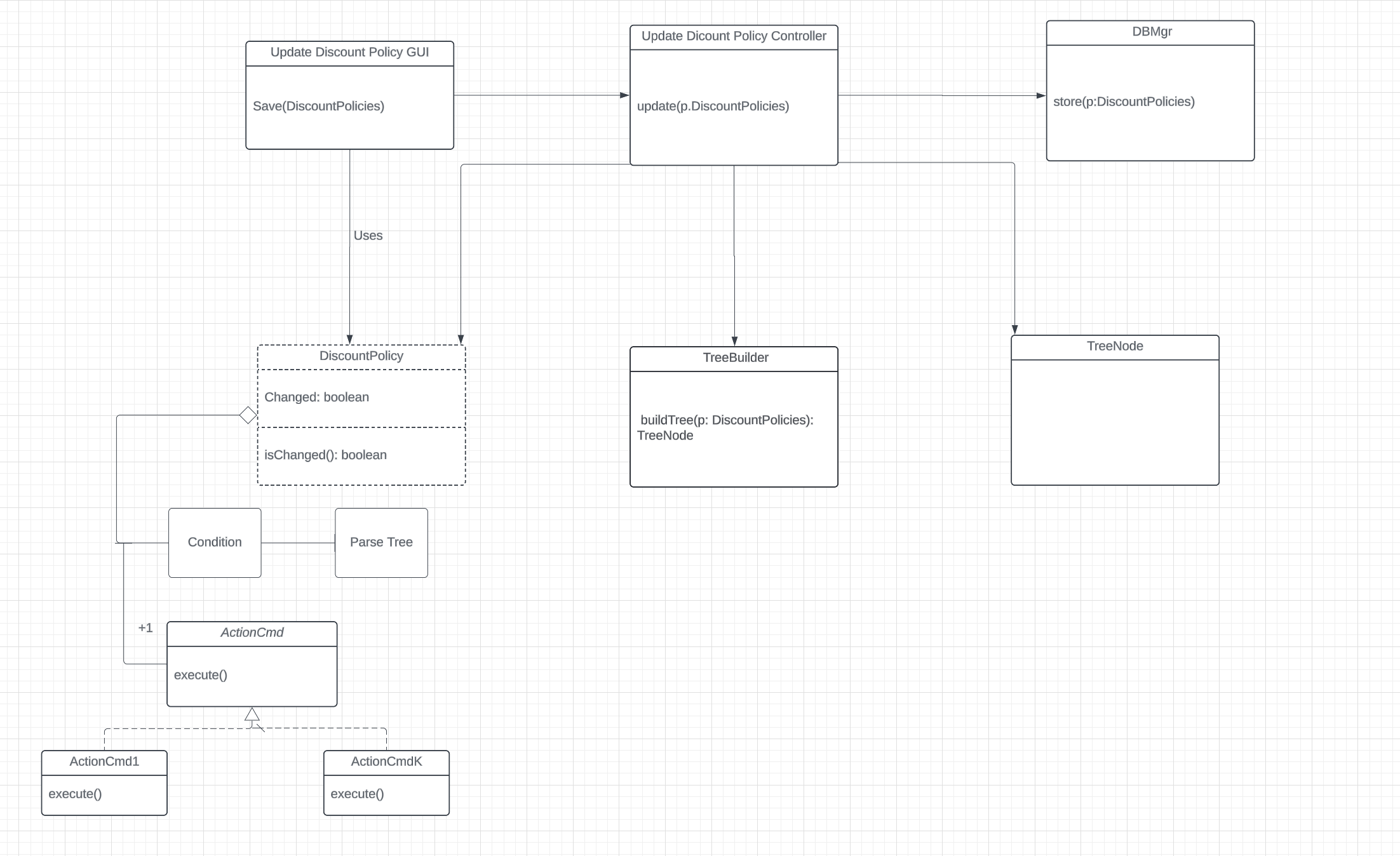
Description automatically generated

8. Convert the informal sequence diagram to a design sequence diagram, which is the same as the informal sequence diagram except that the messages between the objects are converted to function calls with parameters and parameter types, and possibly return values and return types.

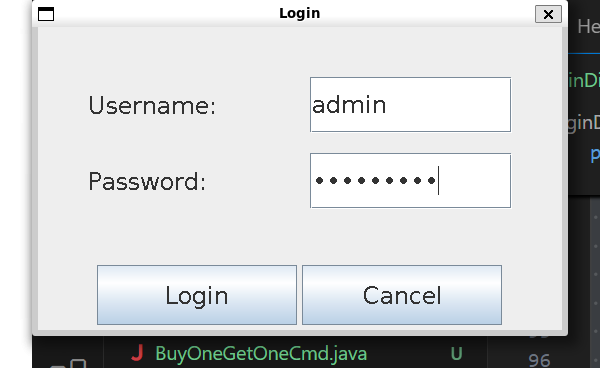
A diagram of a software application

Description automatically generated with medium confidence

9. Derive a design class diagram (DCD) from the design sequence diagram. That is, derive classes, methods and attributes of classes, and relationships between the classes.



10. Implement all classes and all methods in the DCD. Compile, run and debug the software. Produce screen shots showing the working of the software.

Login screen, checks login info against a csv: 

Main screen, showing list of discounts: A screenshot of a computer

Description automatically generated

Edit Rule Screen: A screenshot of a computer

Description automatically generated

After pressing remove rule on the main screen: A screenshot of a computer

Description automatically generated

After pressing save policy (it saves it to a tx that it then reads upon application bootup): A screenshot of a computer error

Description automatically generated

Add rule, same layout as edit rule:  
A screenshot of a computer

Description automatically generated

Rule added: A screenshot of a computer

Description automatically generated

Test Screens:

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Discounts Applied Screen (doesn’t show new price if it is unchanged): A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated