MODEL-BASED SOFTWARE DEVELOPMENT

PRACTICE I.

UML CLASS DIAGRAMS

NORBERT SOMOGYI

TODAY'S AGENDA

I. UML Class Diagram Overview

II. Practical Examples

WHAT IS UML?

- The modeling standard for software
  - Unified Modeling Language
  - Defined by the Object Management Group (OMG)
  - Many tools support editing UML models
  - Current version: 2.5.1
- UML definition
  - Official: https://www.omg.org/spec/UML/2.5.1
  - Practical summary:
    - https://www.smartdraw.com/uml-diagram/
    - https://www.uml-diagrams.org
    - https://creately.com/blog/diagrams/uml-diagram-types-examples/

UML DIAGRAMS

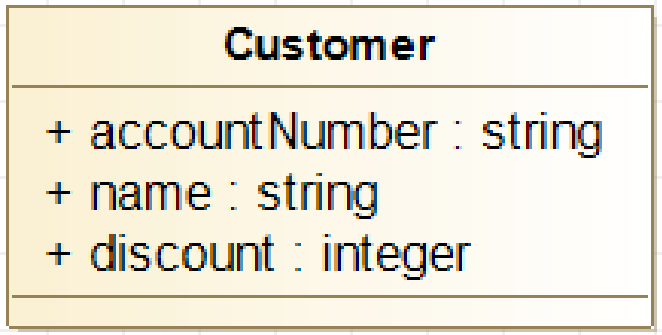
- Diagram = model
- Different aspects of software
  - Actors and interaction
  - Components and hierarchy
  - Structure and dynamic behavior
  - 13 types of diagrams
- Class diagram
  - We are only focusing on this in this from now on

CLASSES AND ATTRIBUTES

Bank account number	Customer name	Customer discount (%)
11742232-15393276	Test Alice	10
11742232-75343245	Test Bob	20
11742232-35393127	Test Li	50

- Let's create a **class** and **attributes** from the table!
- UML – built-in primitive attribute types
  - Boolean, byte, char, integer, float, string, date etc.

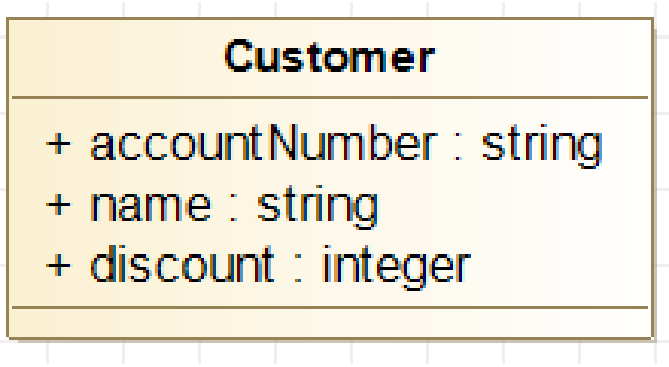
```
public class Customer {
  public String accountNumber;
  public String name;
  public int discount;
}
```



INITIAL VALUE

- Let the discount be 0% initially!
  - Initial value for attributes
  - Not every modeling tool supports it, but it is present in the UML standard

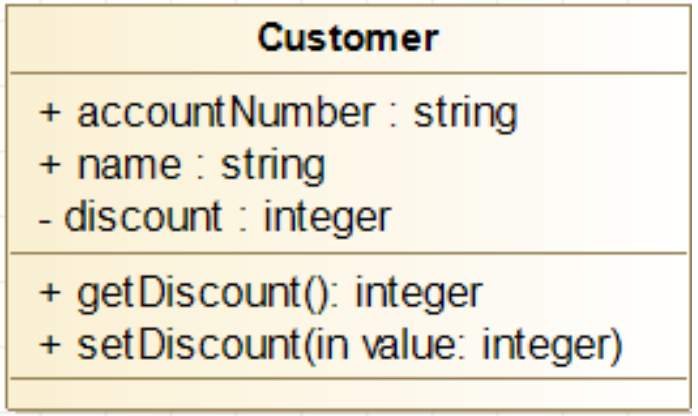
```
public class Customer {
  public String accountNumber;
  public String name;
  public int discount = 0;
}
```



VISIBILITY AND METHODS

- Let's create a getter and setter method for the discount!
  - Method (function), parameters, return type
  - Visibility: public (+), private (-), protected (#)
  - Solution: private attributes and public methods

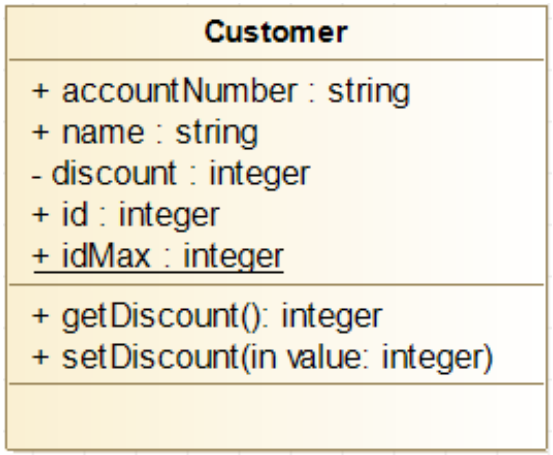
```
public class Customer {
  …
  private int discount = 0;
  public int getDiscount() {
    return discount;
  }
  public void setDiscount(int value) {
    discount = value;
  }
}
```



STATIC ATTRIBUTES

- Every customer should have an identifier. When creating a new customer, the identifier should be the current max value + 1.
  - The customer ID is an instance level attribute
  - The maximum ID is a class-level attribute
  - The idMax attribute is **static** (underlined notion)
  - Solution: private attributes and public methods
  - We do not model the creation logic (behavior)!

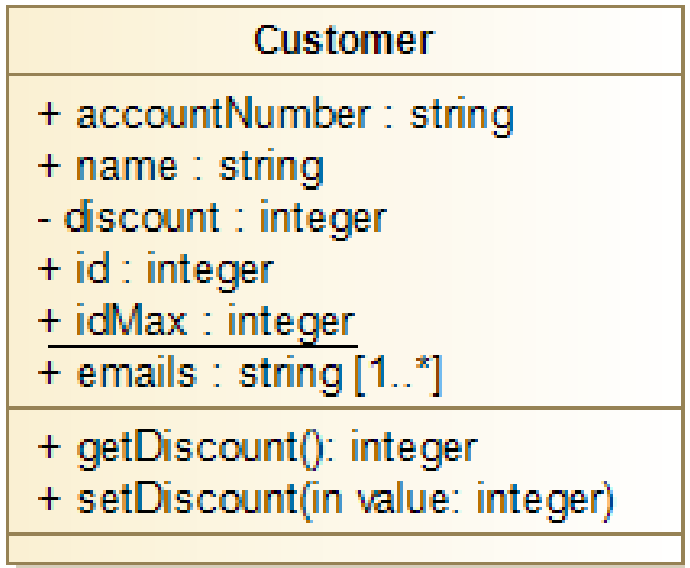
```
public class Customer {
  …
  public int id;
  public static int idMax;
  …
}
```



MULTIPLICITY

- A customer can have one or more email addresses.
  - **Multiplicity**
  - Attribute with multiplicity
    - 0..1
    - 1..1 (default)
    - 0..* (list or array)
    - N .. M

```
public class Customer {
  …
  public String[] emails;
  …
}
```

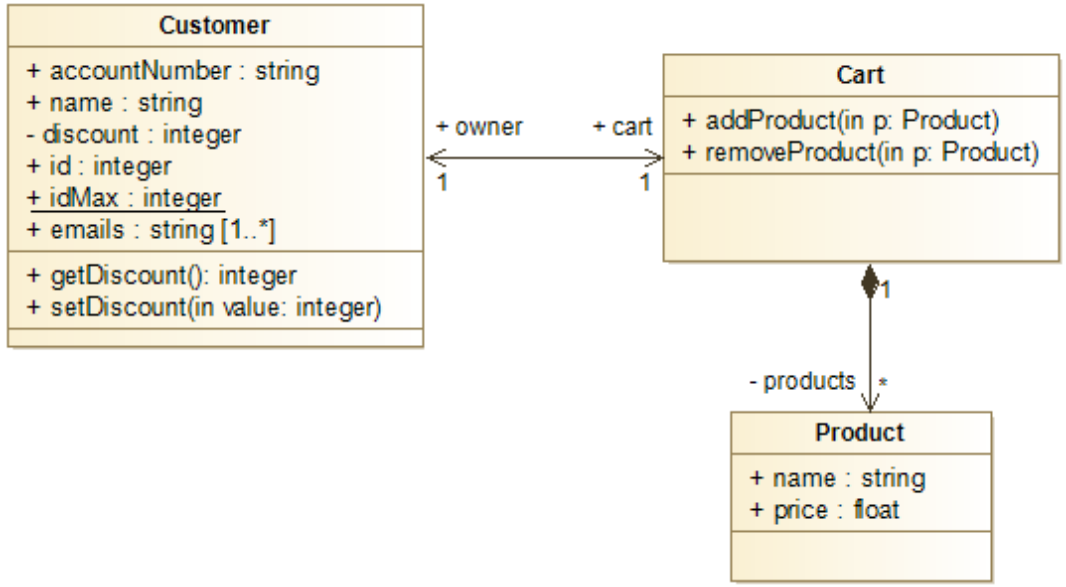


ASSOCIATION AND COMPOSITION

- A customer accesses a cart. A cart can contain products.
  - **Association** and **composition**
  - **Role name** and **multiplicity**
  - Navigability: one or two-way

```
public class Customer {
  …
  public String[] emails;
  public Cart cart;
  …
}
```

```
public class Cart {
  …
  public Customer owner;
  private List<Product> products;
  …
}
```



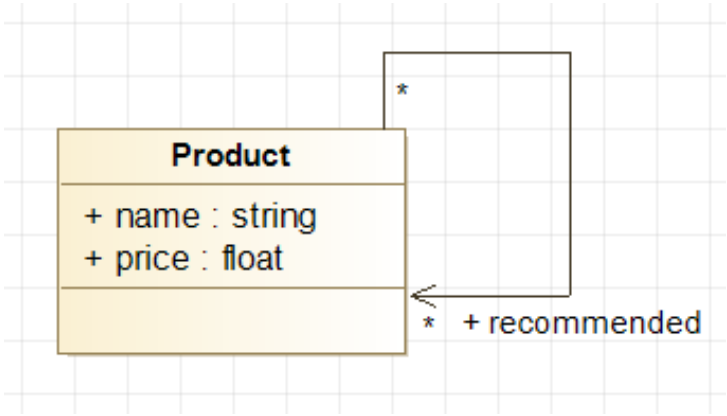
ASSOCIATION AND COMPOSITION

- We use association if…
  - … the relationship is more "loose"
  - … we need navigation, but there is no containment
- We use composition if…
  - … the relationship is more "strict"
  - … we need navigation and there is also containment
- The difference is philosophical
  - In practice, attributes are generated from both
- **Aggregation** relationship also exists, but we do not discuss it

SELF ASSOCIATION

- A product can have other products associated with it that are recommended to the user.
  - **Self association** between two products
  - We do not model the recommendation logic!

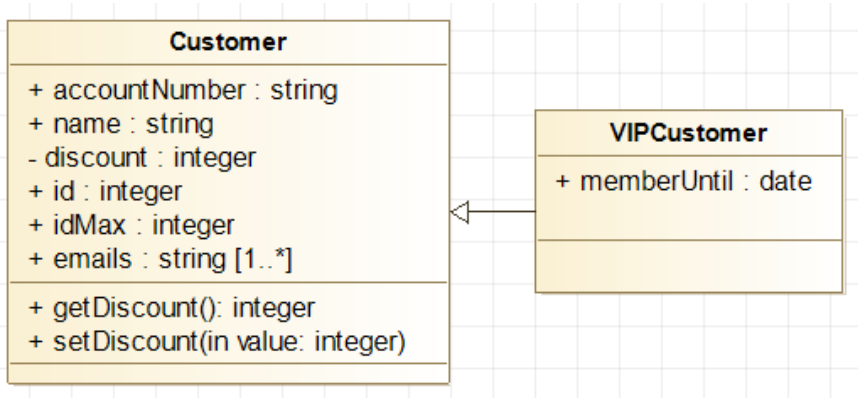
```
public class Product {
  …
  public List<Product> recommended;
}
```



GENERALIZATION / INHERITANCE

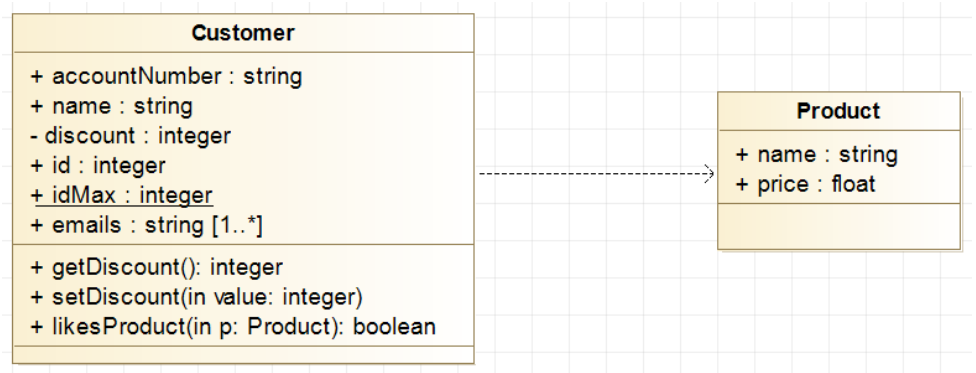
- Let us introduce VIP customers who have a date until which their VIP membership is active. They also behave exactly like normal customers; except they must always have at least 10% discount.
  - **Generalization / inheritance**
  - Inheritance is generated in the source code as well
  - We do not model the discount logic!

```
public class VIPCustomer
 extends Customer {
  public LocalDate memberUntil;
}
```



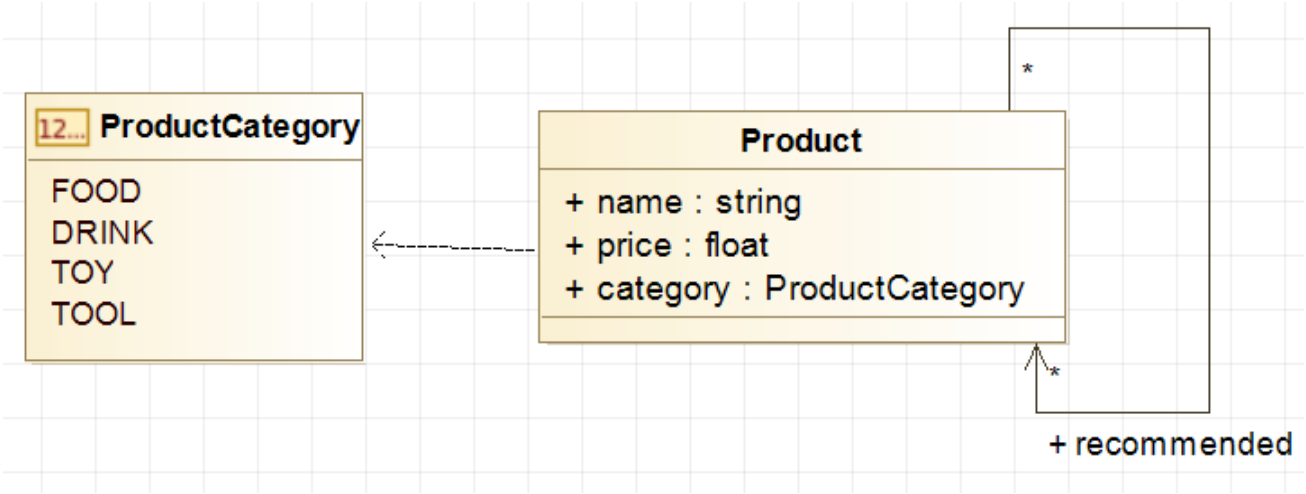
DEPENDENCY

- A customer can like or dislike a product.
  - **Dependency** relationship
  - Typically used when relying upon other classes in function parameters or return types
  - Import / using in code
  - Also used when using enums (see later)
  - No code changes



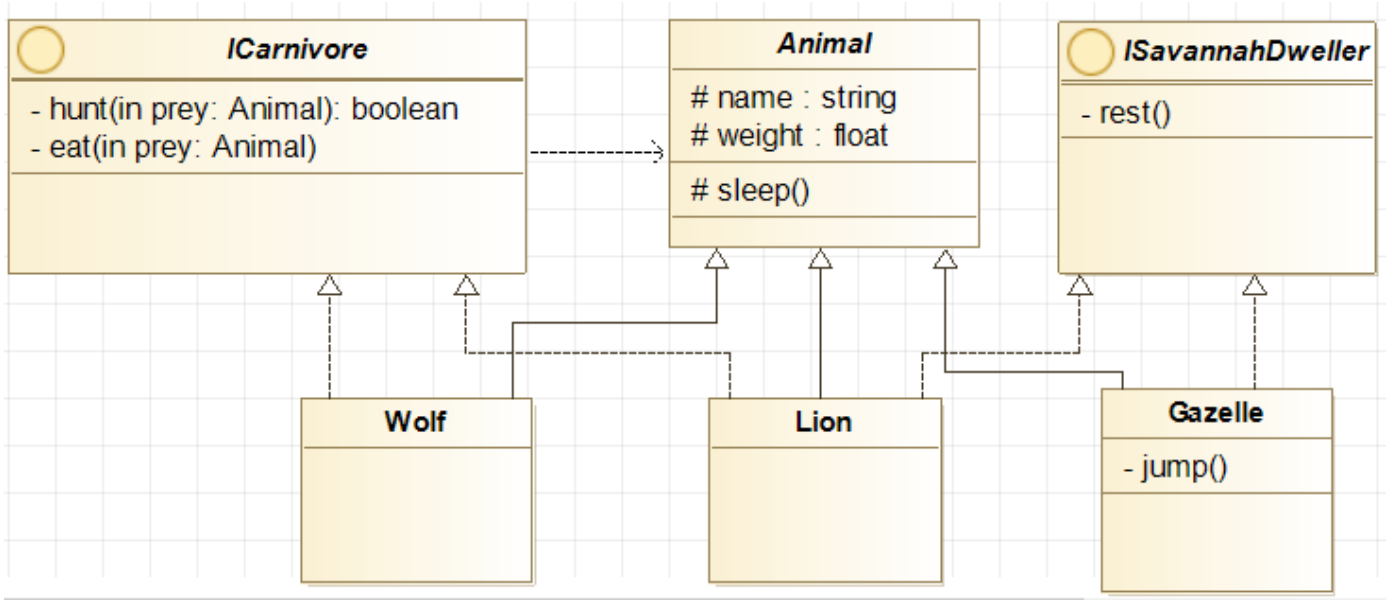
ENUMS

- Products can belong to one of four categories: food, drink, toy or tool.
  - **Enumeration** and **enumeration literals**
  - We use enums when there are a finite number (usually greater than 2) of distinct values



INTERFACES AND ABSTRACT CLASSES

- Let us model animals now! Every animal has a name and a weight, and can sleep. Carnivore animals can hunt other animals (which can either succeed or not) and can eat their prey. Savannah dweller animals can rest. A wolf is a carnivore animal. A lion is a carnivore animal that is also a savannah dweller. A gazelle is a savannah dweller animal that can also jump.

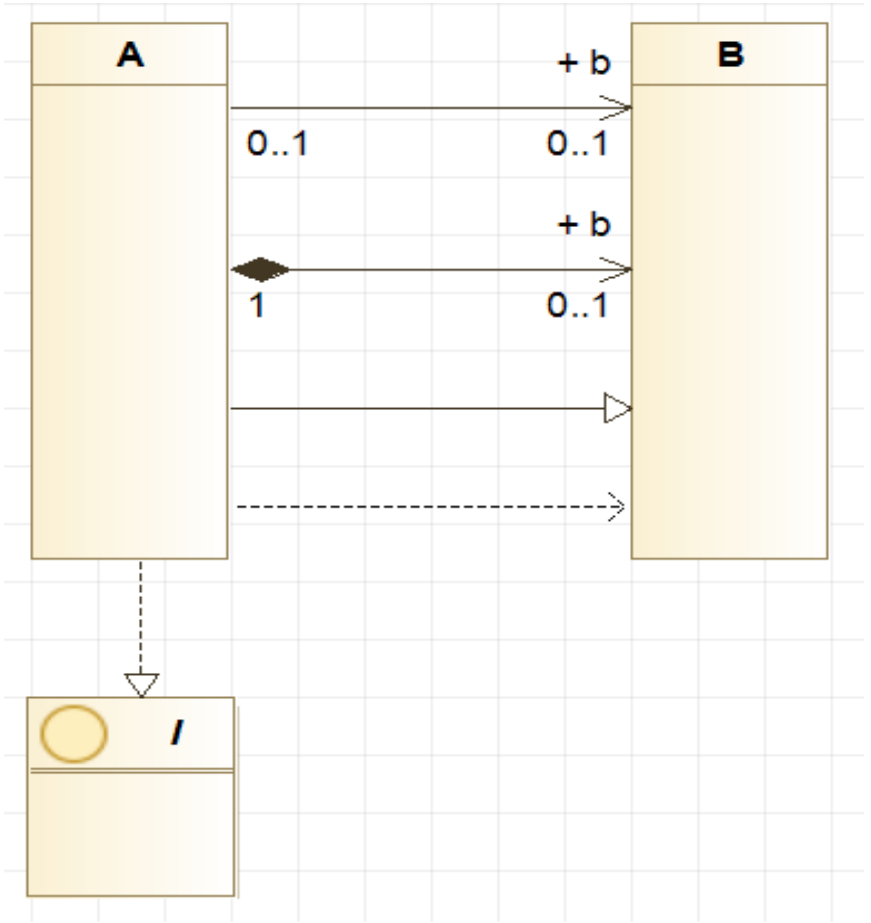


INTERFACES AND ABSTRACT CLASSES

- Most languages do not allow multiple inheritance
- We use interfaces when…
  - … we want to group classes by their behavior and
  - … we cannot inherit due to multiple inheritance
- We use abstract classes when…
  - … we need to move common attributes from subclasses (to avoid code duplication)
  - … we can still inherit (no multiple inheritance)
- Notation
  - <<abstract>> and <<interface>> stereotypes
  - Italic names are used on the previous slide

RELATIONSHIPS – NOTATION SUMMARY

- Relationship types
  - Association
  - Composition
  - Generalization / Inheritance
  - Dependency
  - Interface realization
- Multiplicity
- Role name
- Navigation
  - One-way or two-way



TODAY'S AGENDA

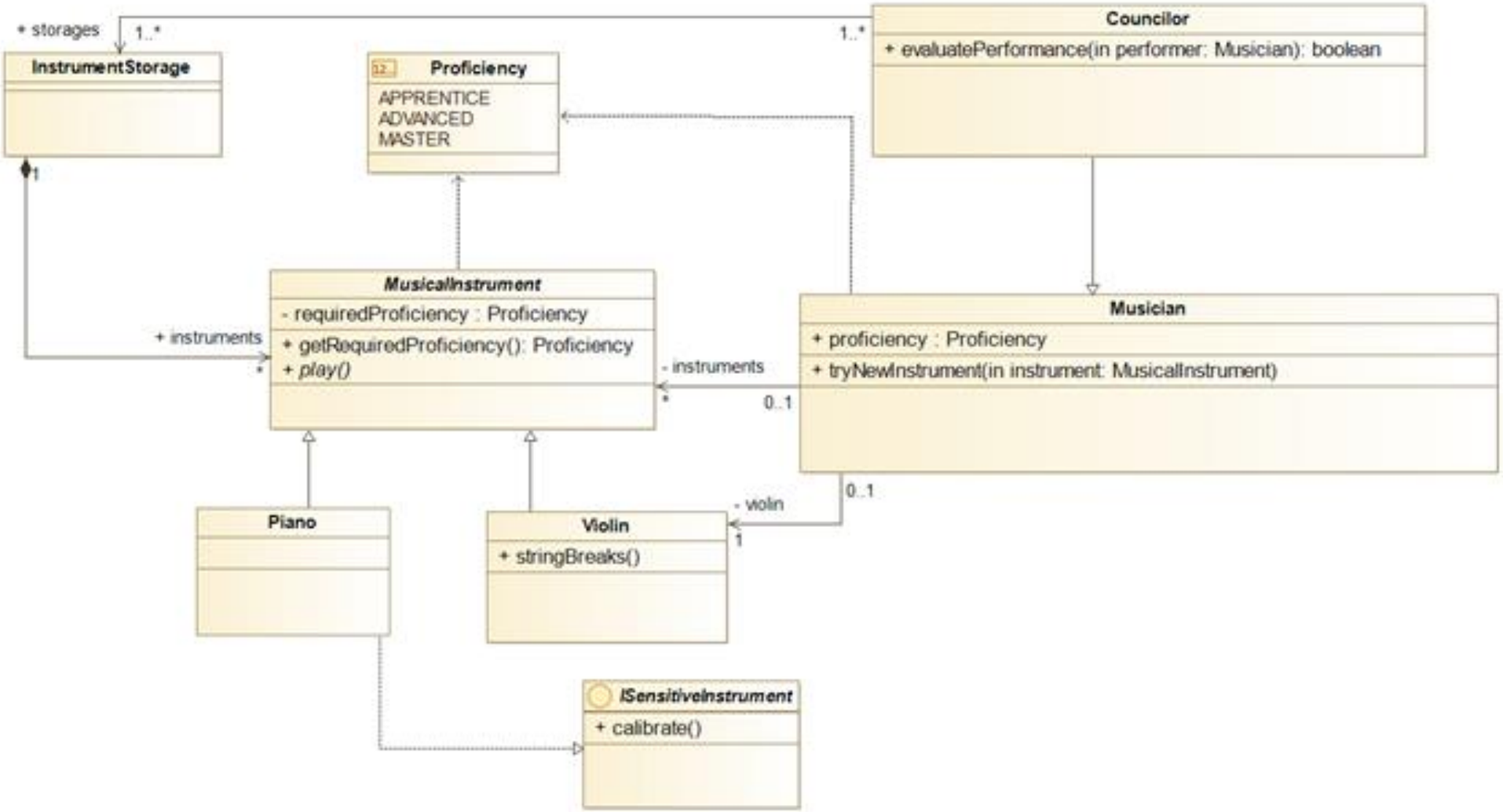
I. UML Class Diagram Overview

II. Practical Examples

TASK #1

- At MusicDot academy, musicians can learn the higher arts of music. There are three proficiencies of musicians: apprentice, advanced, master. At the academy, musical instruments are stored in storages. Every musical instrument has a proficiency requirement, under which the instrument cannot be used by a student. This requirement is set when the instrument arrives at the academy and cannot be changed after. Every instrument can be played, but each one in a different way. Currently there are only two instruments: pianos and violins. When using a violin, its strings can sometimes break. There exist sensitive instruments, which must be calibrated from time-to-time. Every sensitive instrument is calibrated differently. Currently only the piano is a sensitive instrument, but later there can be more. A musician has a proficiency and owns musical instruments. Every owned instrument belongs to said musician, other musicians cannot access it. Besides other instruments, every musician must always have one violin. Musicians can try out a new instrument at any time. Councilors also exist at the academy. Councilors behave as musicians, but they can also rate the performance of another musician: they can decide whether they liked the performance or not. Every councilor has a key to one or more storages of the academy, so that they can access the instruments within.

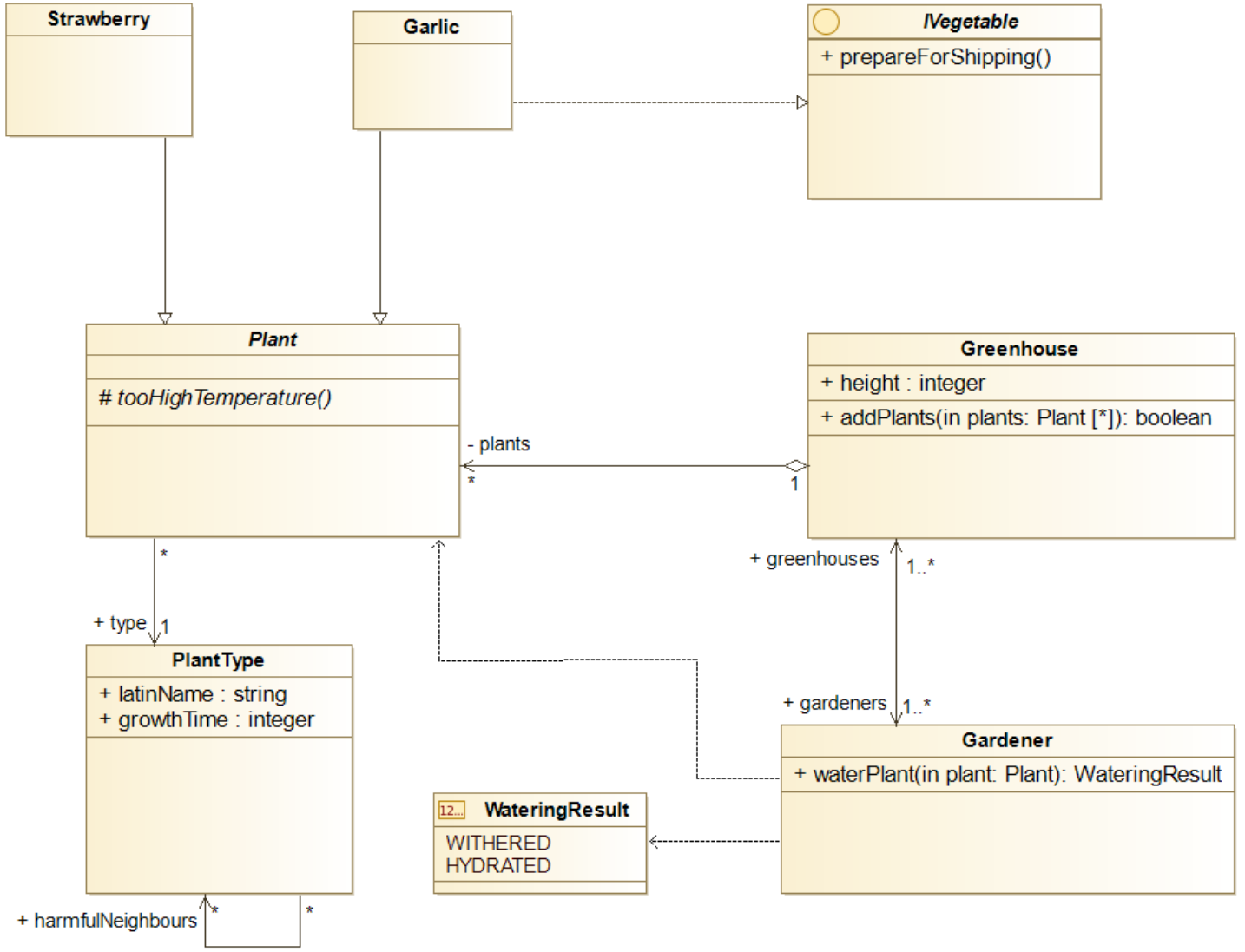
SOLUTION #1


+ storages
1..*
1..*
Councilor
+ evaluatePerformance(in performer: Musician): boolean
InstrumentStorage
Proficiency
APPRENTICE
ADVANCED
MASTER
1
MusicalInstrument
- requiredProficiency : Proficiency
+ getRequiredProficiency(): Proficiency
+ play()
+ instruments
*
- instruments
*
0..1
Musician
+ proficiency : Proficiency
+ tryNewInstrument(in instrument: MusicalInstrument)
0..1
- violin
1
Piano
Violin
+ stringBreaks()
ISensitiveInstrument
+ calibrate()

TASK #2

At GreenShade gardening, plants are cultivated. Every plant has a plant type, which determines its basic properties. These properties include the latin name of the plant and its growing time measured in days. Moreover, every plant type can have harmful neighboring plant types, which means that planting these two types together is not advised. If the temperature of a plant becomes too high, it triggers a reaction, which is an internal process of the plant. Every plant can behave differently during this reaction. Currently, there are two plants cultivated at the gardening: strawberries and garlics. Vegetables are differentiated in the gardening since they must be prepared before transporting to the local market. Every vegetable must be prepared in a different way. Plants are grown in greenhouses. A greenhouse can contain plants, and we also store its height. An entire row of plants can be added to greenhouses at once. An addition can sometimes fail if there is not enough space in the greenhouse. All greenhouses are staffed by gardeners. A gardener knows the greenhouses where they work, and each greenhouse keeps a record of all the gardeners working there. Plants also need watering, which is the gardeners' job. Watering a plant can have one of two outcomes: the plant will either be hydrated, or it will wilt because it has received too much water.

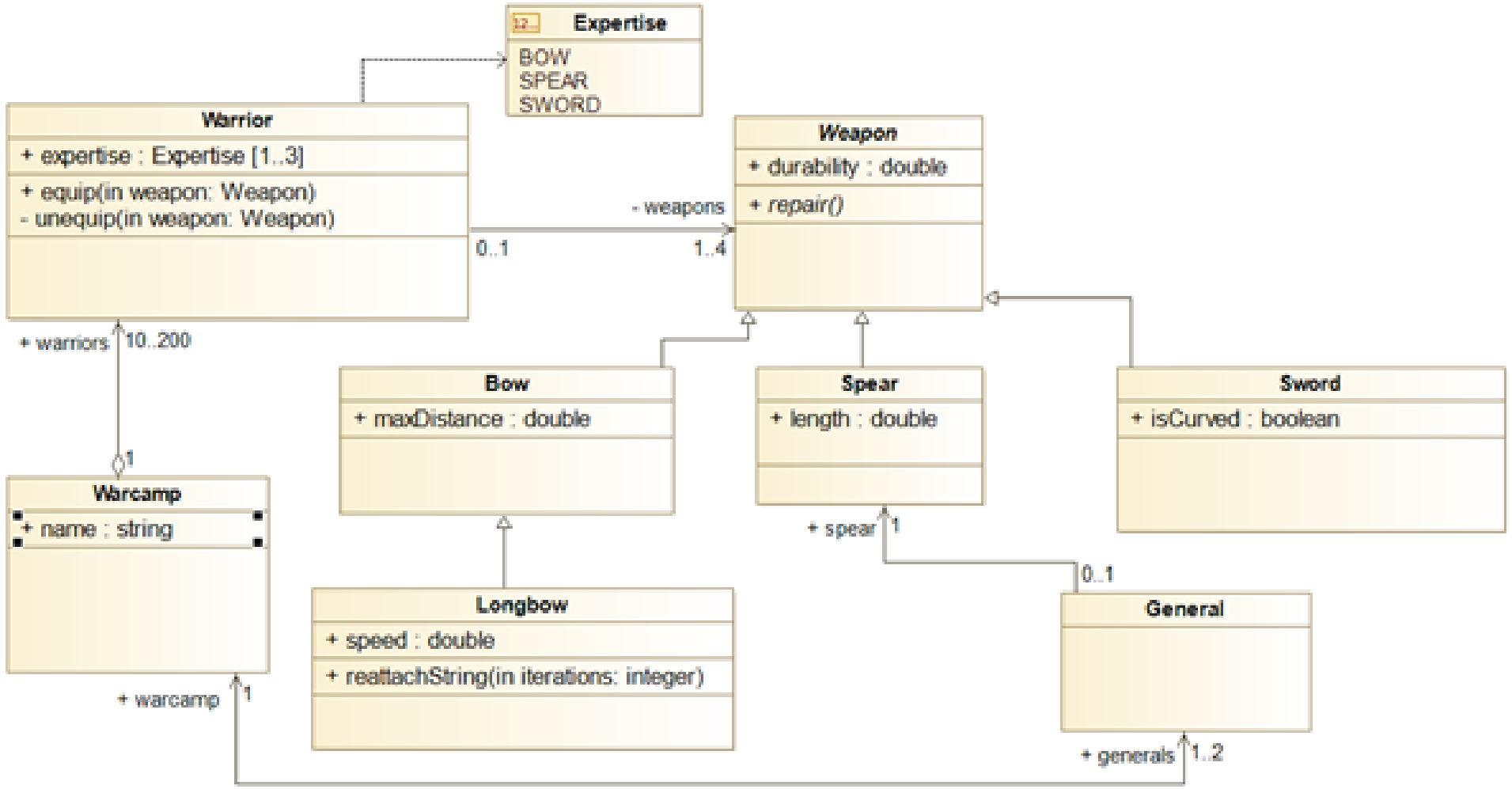
SOLUTION #2


Strawberry
Garlic
IVegetable
+ prepareForShipping()
Plant
tooHighTemperature()
- plants
*
1
Greenhouse
+ height : integer
+ addPlants(in plants: Plant [*]): boolean
*
+ type
1
+ greenhouses
1..*
PlantType
+ latinName : string
+ growthTime : integer
+ gardeners
1..*
Gardener
+ waterPlant(in plant: Plant): WateringResult
12...
WateringResult
WITHERED
HYDRATED
+ harmfulNeighbours
*
*

TASK #3

The country of Jochiussau uses a variety of weapons in its campaigns of conquest. The durability of each weapon is recorded, represented by a rational number between 0 and 1. Each weapon requires occasional repairs, depending on the type of weapon. Three types of weapons are currently used: bows, spears and swords. Bows have the property of maximum range. There exist longbows as well, whose speed must also be stored. To repair longbows, the string must be restrung a given number of times at the beginning, and then the repair continues in the same way as with regular bows. The property of the spear is its length. Among swords there are curved swords, for which the repair process is exactly the same as the repair process of a non-curved sword. Weapons are used by warriors and are not accessible to anyone other than the owner. Each warrior has at least 1 and no more than 4 weapons. A warrior may be proficient with at least one, or even with all three weapons (bow, spear and sword). A warrior can equip a weapon, during which, if he already has 4 weapons, he automatically unequips one of his existing weapons. Warriors live in war camps, each with a unique name. A camp supports a minimum of 10 and a maximum of 200 warriors. Each camp also registers 1 or 2 generals, who oversee the camps. Each general oversees exactly 1 camp. The generals always have exactly 1 spear, they do not use any other weapons.

SOLUTION #3


Expertise
BOW
SPEAR
SWORD
Warrior
+ expertise : Expertise [1..3]
+ equip(in weapon: Weapon)
- unequip(in weapon: Weapon)
- weapons
0..1
1..4
Weapon
+ durability : double
+ repair()
+ warriors
10..200
1
Warcamp
+ name : string
Bow
+ maxDistance : double
Spear
+ length : double
Sword
+ isCurved : boolean
+ spear
1
0..1
Longbow
+ speed : double
+ reattachString(in iterations: integer)
General
+ warcamp
1
+ generals
1..2

THANK YOU FOR YOUR ATTENTION!