Project #4: Review progress
    Use cases II
    Class diagrams II
“How To” Write Use Cases *

1. Identify who will use the system
   - These users are called “actors”
   - Actors are not individuals, but they are “roles” of the users of the system
   - Probably should have been called roles, because actor has too many connotations

2. Pick one of the actors:

3. Figure out what the actor wants to do with the system
   1. Each thing that you figure out is a Use Case
   2. Naturally there may be many Use Cases for each actor
“How To” Write Use Cases - 2

4. For each Use Case decide the most usual interaction course
   ● That is, what **NORMALLY** happens
   ● AKA – Basic course
   ● AKA – Basic flow
   ● AKA – Main success scenario

5. Describe the basic flow for the Use Case
   1. “Actor does something”
   2. “System does something”
   3. “Actor does something”
   4. “System does something” …
   ● Keep it high level and don’t mention GUI
   ● Describe only things for the system that the actor would be aware of
   ● Ditto for the reverse (actor does that the system would be aware of)
   ● Notice “noun verb noun” format above
6. Once you have the main success scenario, consider adding an exception to handle slightly different situations or errors
   • AKA - alternate flow

7. If the Use Case must be extended with additional, optional functionality create a separate use case and use “extends” dependency

8. Examine the use cases and look for common flows, move them into a common use case (and use “includes” dependency)

9. Repeat
Let’s Work on an Example Use Case

- Online store – “Friendly Books and Things”
- Instead of customer browsing and entering orders, FBaT has sales clerks that talk with the customers and do the ordering
  - FBaT believes that they sell more because the clerks provide the personal touch and can talk the customers into buying other things
- Sales clerks talks with customers
- When a customer wants to buy some stuff, the sales clerk interacts with the computer to find the customer’s account, check their credit, and enter the order
- So, let’s write a use case for the computer system to handle this task
  - Go back and review the steps
Example – Steps 1 to 5

Use Case: Enter a Sales Order

Basic Steps
The Sales Order Clerk enters the customer’s surname. The system displays all matching customers. The Sales Order Clerk selects one of those customers. The system displays that customer’s details.
For each item that the customer wishes to order the Sales Order Clerk enters the line details.
When all line items have been entered the system confirms the order.

Use Case: Credit Check a Customer

The Sales Order Clerk enters the customer’s surname. The system displays all matching customers. The Sales Order Clerk selects one of those customers and the system displays that customer’s details.
The system also displays the customer’s payment history for the last six months.
Example – Step 6

Use Case: Enter a Sales Order
The Sales Order Clerk enters the customer’s surname.

Exceptions
If the system can’t locate the customer’s surname, then indicate an error and ask the Sales Order Clerk to enter a different surname.

Use Case: Credit Check a Customer
The Sales Order Clerk enters the customer’s surname.

Exceptions
If the system can’t locate the customer’s surname, then indicate an error and ask the Sales Order Clerk to enter a different surname.
Use Case: Evaluate A Customer’s Credit

This use case ‘extends’ the Credit Check A Customer Use Case. The System evaluates the previous six months of payment data and provides the Sales Order Clerk with a credit worthiness evaluation of the Customer which the Clerk may use to request an up-front deposit on the order.
Example – Step 8

Use Case: Enter a Sales Order
Includes use case “Display Customer Details” to lookup and display a customer’s details.

For each item that the customer wishes to order, the Sales Order Clerk enters the line details. When all line items have been entered the system confirms the order.

Use Case: Credit Check a Customer
Includes use case “Display Customer Details” to lookup and display a customer’s details.
The system also displays the customer’s payment history for the last six months.

Use Case: Display Customer Details
The Sales Order Clerk enters the customer’s surname. The system displays all matching customers. The Sales Order Clerk selects one of those customers and the system displays that customer’s details.

Exception
If the system can’t locate the customer’s surname, then indicate an error and ask the Sales Order Clerk to enter a different surname.
Review of Our Use Cases

● Define:
  - Main success flow (basic steps)
  - Exceptions (alternate flow) in a use case
  - Includes another use case
  - Extends another use case

● Display Customer Details (including exception)

● Enter Sales Order (which includes Display Customer Details)

● Credit Check a Customer (which also includes Display Customer Details)

● Evaluate a Customer’s Credit (this extends Credit Check a Customer)
UML

- UML is silent about Use Cases
- Use Cases are mostly about text documents
- However, you can draw a use case diagram to help you visualize the relationships
  - A picture is worth a thousand words
Sales Order UML Diagram

Sales Order Clerk

Enter A Sales Order

<<includes>>

Display Customer Details

<<includes>>

Credit Check A Customer

<<extends>>

Evaluate a Customer’s Credit
Some Guidelines

- Simple, simple, simple, simple, simple, simple, simple
  - Adding too much is a very bad plan

- It is ok to attach other things to a requirements specification, but they don’t belong inside the Use Cases
  - If you want to have a prototype GUI add a separate image
  - Modeling some object relationships should be in a Class Diagram
  - Model dynamic object relationships in a sequence diagram

- Use Cases need to be sized “just right”
  - Try not to have tiny Use Cases (decomposing into too small of a piece usually means that you are falling into “HOW”)
  - Try not to have giant Use Cases either

- Use “extends” and “includes” sparingly
Problems - 1

● What is wrong with this?
Problems - 2

• ?? Let’s look at two Use Case descriptions to help us understand problem 2 in previous slide

Use Case: Enter A Sales Order
The actor selects a customer from a list displayed by the system. Use Use Case "Lookup Customer". The system then displays the sales order screen and the actor enters the item code and quantity required for each item. The system displays a running total of the value of the order. Once the order is complete the actor confirms the order and the system resets ready for the next order.

Use Case: Lookup Customer
The actor selects a customer by entering their reference number. The system then displays the complete details of that customer, name, address etc along with a complete history of purchases they have made.

• That means that Enter A Sales Order is really:

Use Case: Enter A Sales Order
The actor selects a customer from a list displayed by the system. The actor selects a customer by entering their reference number. The system then displays the complete details of that customer, name, address etc along with a complete history of purchases they have made. The system then ...

• Also, do we really want to show the customer’s history?
Problems – 3 – Revised Model

[Diagram showing relationships between roles and tasks]

- Sales Order Clerk
  - Enter A Sales Order
  - Credit Limit Exceeded
  - Lookup Customer
  - Find Customer
  - Edit Customer
  - Create New Customer
Problems – 4 – Final Version

- Simplify – all use cases apply to the user!!
- Even though we have lost information that information is most likely not relevant or over constraining the possible implementation

![Diagram of use cases](image-url)
Another Problem

- What is wrong with this?

Diagram:

- Sales Order Clerk
  - Create Sales Order
    - <<extends>>
    - <<includes>>
  - Add Customer
    - Customer Not Found
A Possible Solution

Sales Order Clerk

Create Order

Customer Not Found

Create Customer

Add Customer
Use Case: Enter A Sales Order

The actor selects a customer from a list displayed by the system. The system then displays the sales order screen and the actor enters the item code and quantity required for each item. The system displays a running total of the value of the order. Once the order is complete the actor confirms the order and the system resets ready for the next order.

Use Case: Enter A Sales Order

1. Display list of customers from central database.
2. Actor chooses one of those customers.
3. The system displays sales order form.
4. While there are more items for this order
5. actor enters item and quantity
6. system updates running total
7. End While
8. Confirm the Order

Use Case: Enter A Sales Order

The sales person selects a customer. The system prompts the salesperson to enter the details of the order (item code and quantity) and maintains a running total of the value of the order. Once the salesperson is happy with the order they confirm it and the system records the confirmed order.
Back to UML Classes
**UML Class Review**

- **Data members (attributes):**
  - -x: double
  - -y: double
  - -z: double
  - -n: int

- **Instance methods:**
  - +name()
  - +method1(:double):double
  - +method2():bool
  - +classMethod()

- **Class method (static):**

**Important** – if something isn’t specified, then that doesn’t necessarily mean that it isn’t there. For example, does name take an argument or not, does it return a value?

**These compartments are optional. But if you need methods, then you have to have at least empty data member compartment.**
Unary Association Review

A knows about B, but B knows nothing about A

these two diagrams are identical in what they mean.

Arrow shows direction of association in direction of dependency

UML Comment

Void doSomething() {
    myB.service();
}

void doSomething() {
    myB.service();
}
Dependency Review

- **Weak relationships (also transitory):**
  - Class A simply knows of class B
    » E.g., a method in A creates and immediately returns an object of class B

- That is: a change in one may force changes in the other although there is no explicit association between them.
Generalization (Inheritance) Review

Base class or super class

Arrow shows direction of dependency

Derived class or subclass

```
public class B : A {
    ...
}
```
Aggregation Review

Aggregation = Association with "whole-part" relationship

Shown by hollow diamond at the "whole" side
No lifetime control implied
Composition Review

Composition = Aggregation with lifetime control

Lifetime control: construction and destruction controlled by "owner"
→ call constructors and destructors (or have somebody else do it)

Shown by filled diamond at the "owner" side

Lifetime control implied

Lifetime control can be transferred
Multiplicity in Composition

- A cricket team has 11 players. One of them is the captain.
- A player can play only for one Team.
- The captain leads the team members.

- Is this a good representation?
How About This Version?

![Diagram showing relationships between Team, Captain, and Player]
Some Problems: Modeling Whole/Part Associations

- **Composition**
  - E.g., Glider has a component Tail
  - Composite object is the whole
  - Component is the part

- **Composite cannot exist without component**

- **A component is part of only one composite**
  - Stronger: composite should create component
  - Stronger: if delete composite, component goes away too, i.e., cascading delete

- **Typically a composite has many different kinds of components**
Whole/Part Associations

- Aggregation, weaker than composition
  - E.g., City is an aggregate of houses
  - Aggregate is the whole
  - Constituent is the part

- Aggregate may exist without constituents
- Each object may be part of more than one aggregate
- Typically, constituents are of same class
Composition/Aggregation Notations

- **MovieTheater**
  - **whole**
  - 1

- **BoxOffice**
  - **part**
  - 1

- **Movie**
  - **part**
  - 0..*
  - **aggregation**

- **composition**
Composition vs. Aggregation

- In real world there are 7 or 8 varieties of whole/part relationships, but UML has only two constructs – composition and aggregation
- It can be very difficult, and confusing to choose one
- Some say that UML 2.0 has essentially deprecated aggregation
  - So, they recommend “don’t use it (choose composition or association)”
  - But you might see it
  - And some think that it makes sense
Associations Summary

- Can express different kinds of associations between classes/objects with UML
  - Association, aggregation, composition, inheritance, dependency
- Can go from simple sketches to more detailed design by adding adornments
  - Name, roles, multiplicities
  - lifetime control