Object-Oriented Design

Lecture 2

Contents

Classes and Objects

What is a class ?

- A class is a template that defines the attributes and methods of an object, which can be used to create many objects.
- A class has the following characteristics:
    - The name of the class is a noun (e.g., Car, Door, Ship, Dog, Session, Payment, etc.)
    - Functions (behaviors) are defined in the class that contains the function data (attributes)
    - Many objects can be created from the same class (e.g., BMW, Audi, Fiat are all cars with different properties)
    - A class can be inherited by many sub-classes (children)
    - A child class (sub-class) can have one or more parent classes (super-classes)
    - Data in a class is encapsulated (hidden) from other classes and interactions are achieved through publicly exposed interfaces
    - Behaviors of a class are exposed while the implementation details (of functions) are hidden

Classes and Objects

What is an object ?

- An object is an instance of a class, created from the class template during system execution.
- Object-oriented software is composed of many objects created from system classes at runtime.
- An object is a composite data type identified by its attributes (fields) and behaviors (functions), which are defined in the class.
- It is important in an object-oriented design analysis to properly identify the classes and the relationships between the classes to model how the system would work.

Object-Oriented Paradigm Principles

What are the object-oriented paradigm principles ?

- Object-oriented paradigm is the method used to transform software requirements into classes.
- The object-oriented paradigm contains four main principles which are:
    - Abstraction
    - Encapsulation
    - Inheritance
    - Polymorphism

Object-Oriented Paradigm Principles

What is Abstraction ?

- Abstraction hides the internal implementation of (functions) and expose to other classes the behavior of those functions.

  “If you want to drink a cup of coffee, simply drink the coffee, you won’t ask the barista how they made it!!!”

- There are two types of abstractions:
  - Data abstraction → Hide the variables that store the data, and only expose the values
  - Function abstraction → Hide the function code and only expose the function behavior

Object-Oriented Paradigm Principles

What is Encapsulation ?

- Encapsulation is a fundamental principle of object-oriented paradigm that increases data and code security, enhances performance by reducing object interactions at runtime.

  "Too many object interactions affect the system performance negatively, so what do we do ?? .... "

- Encapsulation bundles the data attributes (fields) and the functions using those attributes in the same class. This process tends to minimize the number of interactions from outside to use the data (attributes are used within the class). Also, the attributes are hidden (private) within the class, which increases data security.

Object-Oriented Paradigm Principles

What is Inheritance ?

- Inheritance allows classes to have sub-classes (or children)

  "If we are defining a Person class, a Cashier is a Person, a Manager is a Person, a Waiter is a Person, ... "

- Inheritance principle has the following rules:
  - A parent class (super-class) can have many children
  - A child class (sub-class) can have more than one parent
  - A child class inherits their parent class properties (attributes and functions) and can also override them.

Object-Oriented Paradigm Principles

What is Polymorphism ?

- Polymorphism means more of the same! Polymorphism is the main strength of object-oriented paradigm.

  "A banking application has an abstract operation calculateInterest, but the way the interest is calculated for different accounts can be different."

- Polymorphism allows:
  - Object of different types to be treated as objects of a common parent.
  - Multiple object types with the same parent can implement the same function in different ways.
  - Multiple object types with the same parent can use the same attributes with different values.

Object-Oriented Design

What is Object-Oriented Design (OOD) ?

- Object-oriented design is the process of designing a software using object-oriented paradigm.
- Object-oriented design stages are the following:
    - Identify the classes
    - Identify the classes' attributes (i.e., fields) and behaviors (i.e., functions)
    - Identify the how classes interact with each other (i.e., relationships) and create a visual class model (i.e., class diagram)

Object-Oriented Design

Identify classes:

- Classes are derived from use cases and actors described in the requirement analysis process. Identifying classes in an object-oriented system is the key to success.
- Classes should be identified exactly as needed, to reduce coupling and enable extensibility. Identifying classes is a skill developers improve by practice and experience.
- A class name is always a “noun” starting with uppercase letter (e.g., Bank). If the class is two words, both words should be starting with uppercase letters (e.g., CreditCard).
- Identifying a class always requires to identify the attributes (fields) and functions.
- A class can use the functions of another class. We say that one class (the client) uses or depends on another class (the supplier). E.g., Customer makes a deposit, but the Account class contains the deposit function, and the Customer uses it.

Object-Oriented Design

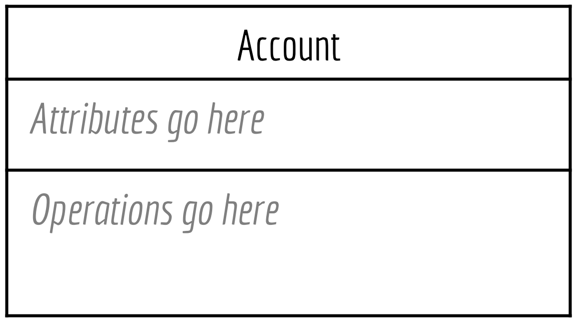
Identify attributes and operations:

- Attributes (or fields) in a class are the variables defined at the top of the class that hold data specific to instances of that class. Attribute names are “nouns” starting with a lowercase letter, e.g., account.
- Attributes are encapsulated (bundled) in the same class where they are used.
- Attributes are abstracted, as the variables that are hidden, and only the values are exposed to other classes.
- Methods of a class are used to perform certain tasks. There are two types of methods:
    - A function → returns a value to the caller. Function names are nouns (e.g., area(), size())
    - A procedure → returns no value to the caller. Procedure names are verbs (e.g., move(), jump())
- Methods are encapsulated (bundled) with the data they use in the same class.
- Operations are abstracted, as the implementations are hidden and only the behaviors are exposed to other classes.

Object-Oriented Design

Class diagram:

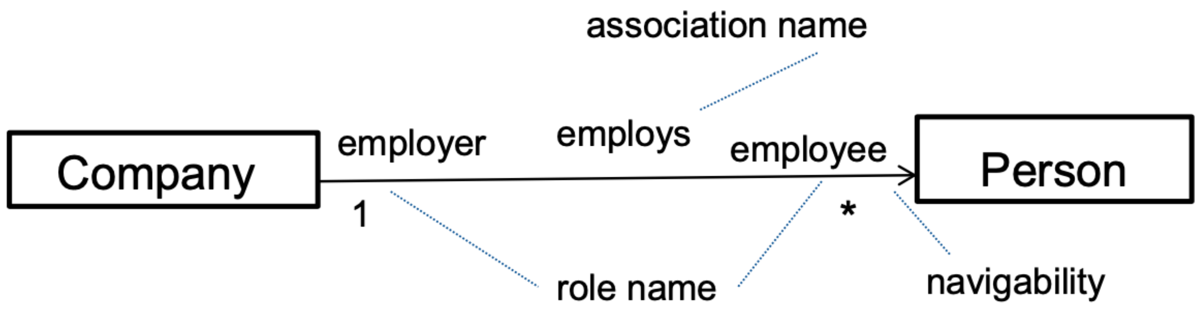
- A class diagram is a UML visual representation of the object-oriented software system.
- A class diagram describes the attributes and operations of a class and any constraint imposed on the system due to dependencies and object relationships.
- A class diagram is used as a plan to construct executable code and define the way in which objects may interact.
- A class diagram consists of classes (with name, attributes, operations specified) and their relationships



UML Class Diagram

Associations:

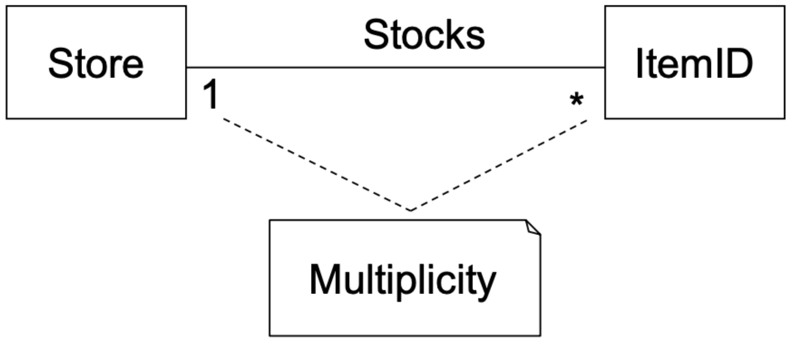
- An association is a relationship between classes of objects that indicates some meaningful and interesting connection.
- An association is labelled by
    - Association name
    - Role name
    - Multiplicity
    - Navigability

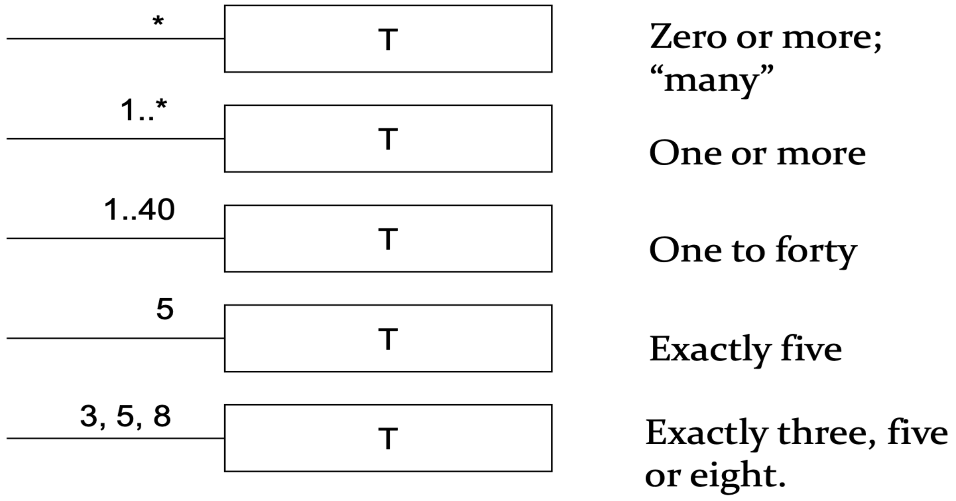


UML Class Diagram

Multiplicity:

- Multiplicity defines how many instances of type A can be associated with one instance of type B, at a particular moment in time. E.g., a single instance of a Store can be associated with “many” (zero or more) Item instances.

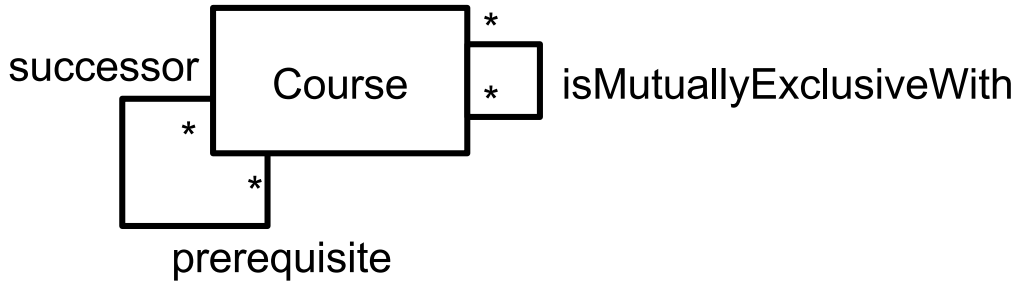




UML Class Diagram

Recursive or reflexive associations:

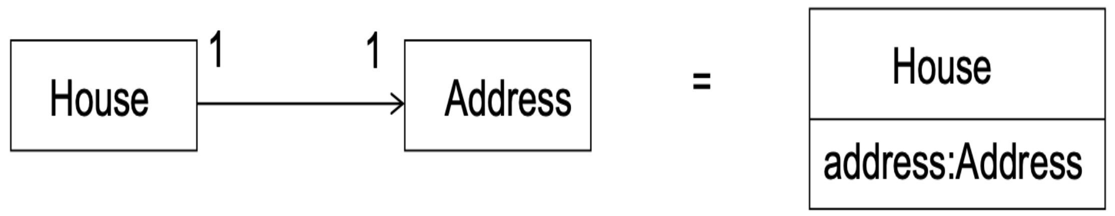
- A class may have an association with itself; this is known as a recursive association or reflexive association.



UML Class Diagram

Associations and attributes (1):

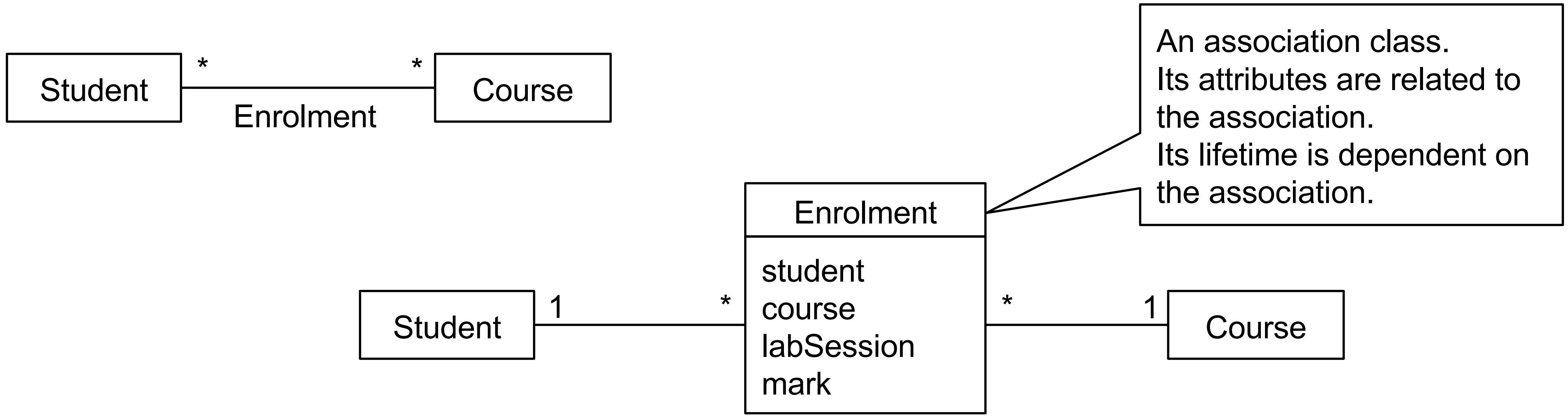
- There is a close link between class associations and class attributes. An association between a source class and a target class can hold an object reference to objects of the target class.
- If the target multiplicity is exactly 1, it is not worth showing it association.



UML Class Diagram

Associations and attributes (2):

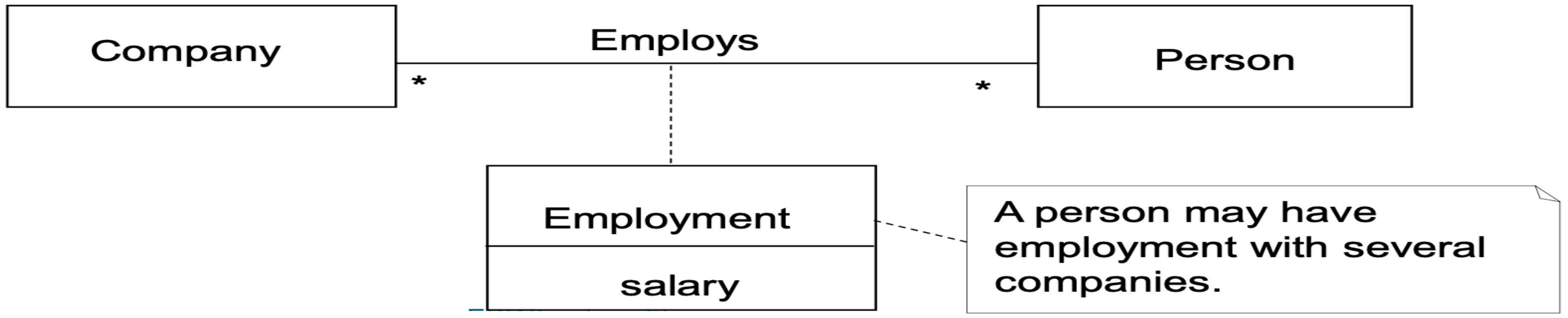
- Sometimes an association has attributes of its own. Such an association must be modelled as a class (because attributes can only be defined in a class).
- Each object of an association class has attribute values and links to the objects of the associated classes.



UML Class Diagram

Association classes:

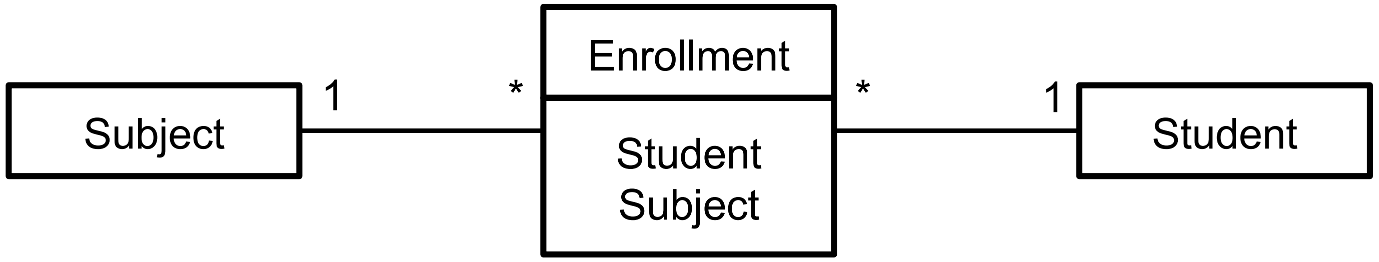
- An attribute is related to an association.
- Instances of the association class have a life-time dependency on the association.
- There is a **many-to-many association** between two concepts. The presence of a many-to-many association between two concepts is a clue that a useful associative type should exist in the background somewhere.



UML Class Diagram

Association classes:

- Instances of the association class are really links that have attributes and operations.
- You can use an association class when there is a single unique link between two objects at any point in time.
- In other words, each Student can enroll in each Subject once at a given time.
- Reify the relationship by expressing it as a normal class.



UML Class Diagram

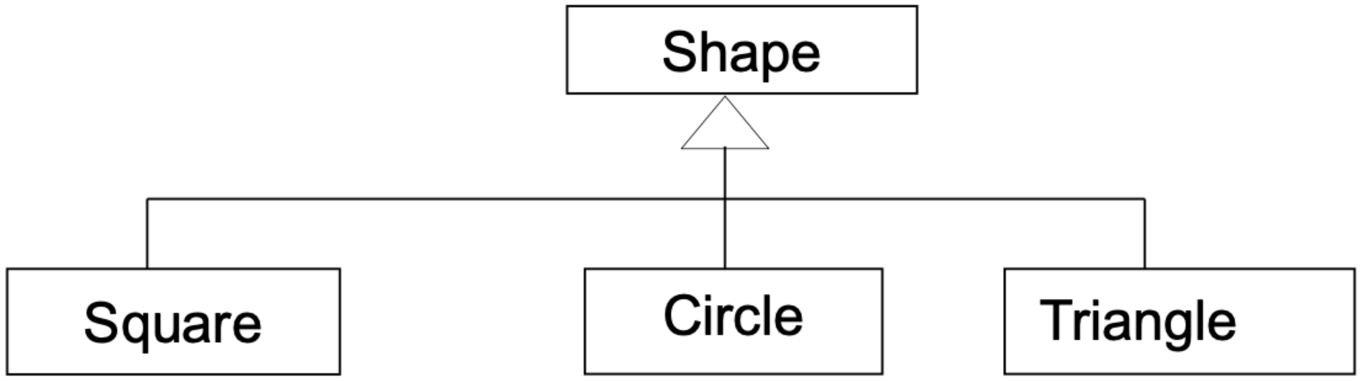
Generalization-Specialization:

- Generalization is the activity of identifying commonalities among concepts and defining Super class (general concept) and subclass (specialized concepts) relationships.
- A conceptual superclass definition is more general than a subclass definition.

UML Class Diagram

Class inheritance:

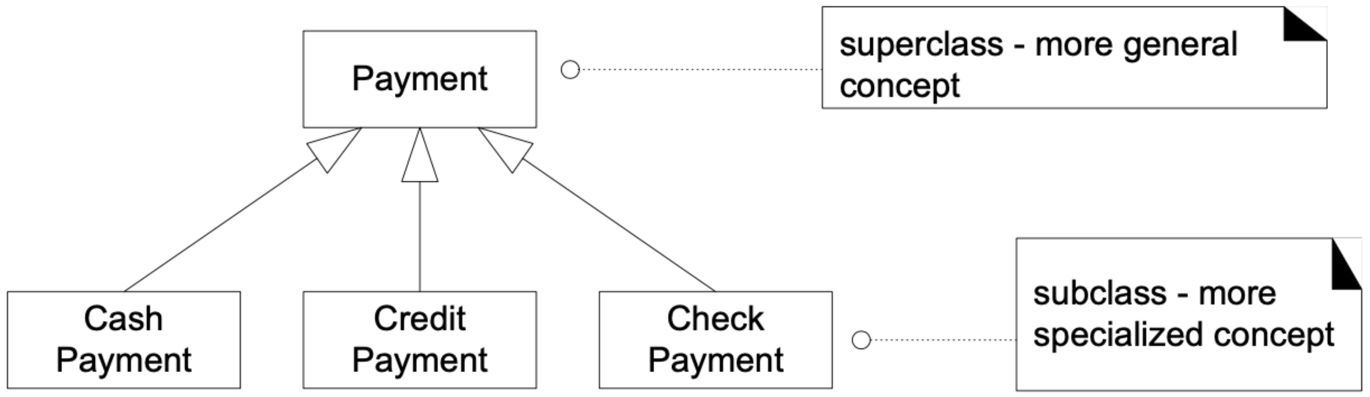
- When classes are arranged into generalization hierarchy, it means that there is inheritance between classes whereby the subclasses inherit all the features (attributes, operations, relationships) of its Super class.
- Generalization-Specialization is also called Class Inheritance.



UML Class Diagram

Class inheritance:

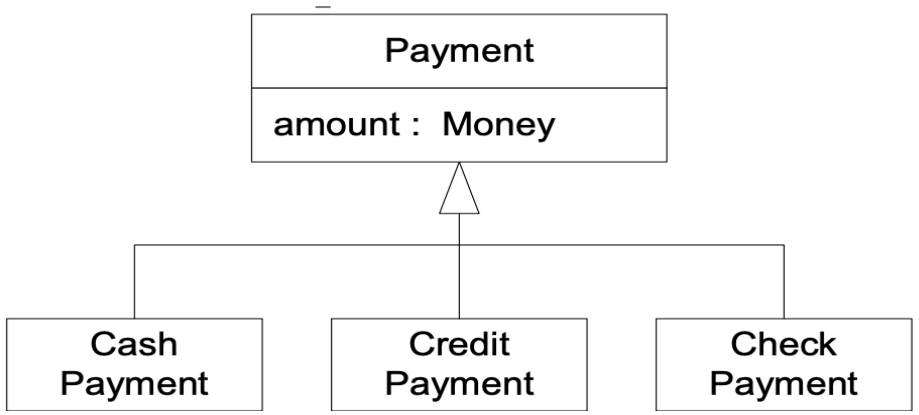
- The concepts CashPayment, CreditPayment and CheckPayment are all very similar.
- We can organize them into a generalization-specialization class hierarchy in which the superclass Payment represents a more general concept, and the subclasses more specialized ones.


Payment
superclass - more general concept
Cash Payment
Credit Payment
Check Payment
subclass - more specialized concept

UML Class Diagram

Conceptual class definition:

- When a hierarchy is created, statements about the superclass apply to subclasses. E.g., All Payments have an amount and are associated with a Sale; a Credit Payment is a transfer of money via a credit institution which needs to be authorized. The definition of Payment encompasses and is more general than the definition of Credit Payment.
- The subclass must fully conform to the attributes and associations of its superclass. ("100% rule").
- All attributes and operations of the superclass are applicable to the subclasses.



UML Class Diagram

Conceptual subclass definition conformance:

- A conceptual subclass should be a member of the set of the superclass
  - CreditPayment should be a member of the set Payments.
- The conceptual subclass is a kind of superclass
  - CreditPayment is a kind of Payment.
- This is called an “is-a relationship”
  - The statement “‹subclass› is a ‹superclass›” should be true.
- Every instance of the ‹subclass› can be viewed as an instance of the ‹superclass›.

UML Class Diagram

Defining subclasses:

- The subclass has additional attributes of interest.
    - Book (subclass of LoanableResource) has an ISBN attribute.
- The subclass has additional associations of interest.
    - CreditPayment, subclass of Payment, is associated with a CreditCard.
- The subclass concept is handled differently than the superclass or other subclasses (in ways that are of interest).
    - Software, subclass of LoanableResource, requires a deposit before it may be loaned.
- The subclass concept represents an object that behaves differently than the superclass or other subclasses (in ways that are of interest).
    - Male, subclass of Human, behaves differently than Female with respect to shopping habits.

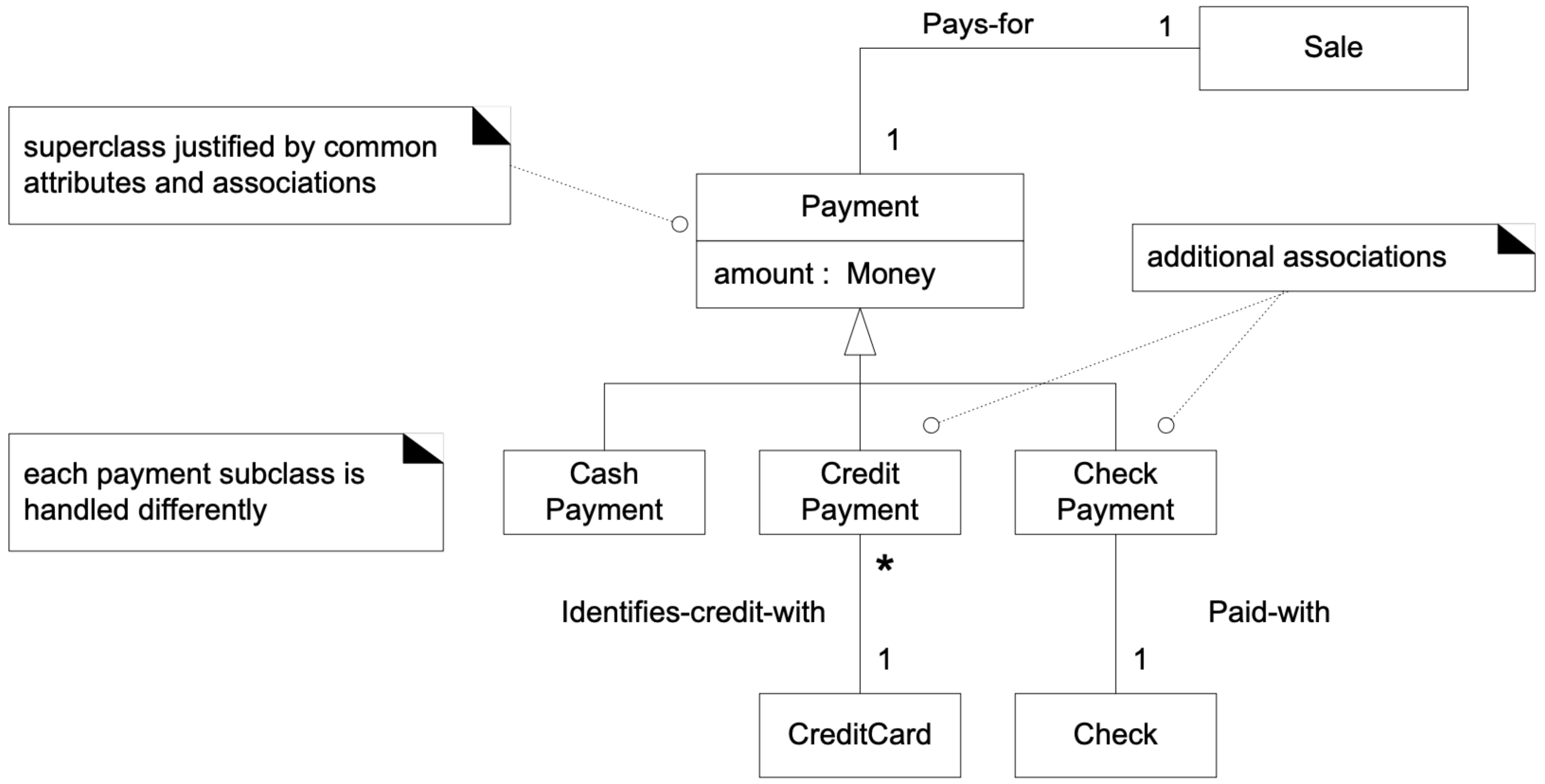
UML Class Diagram

When to define a conceptual superclass?

- Create a conceptual superclass in a generalization relationship to subclasses when:
    - The potential conceptual subclasses represent variations of a similar concept.
    - The subclasses will conform to the 100% and “is-a” rules.
    - All subclasses have the same **attributes** which can be factored out and expressed in the superclass.
    - All subclasses have the same **associations** which can be factored out and related to the superclass.

UML Class Diagram

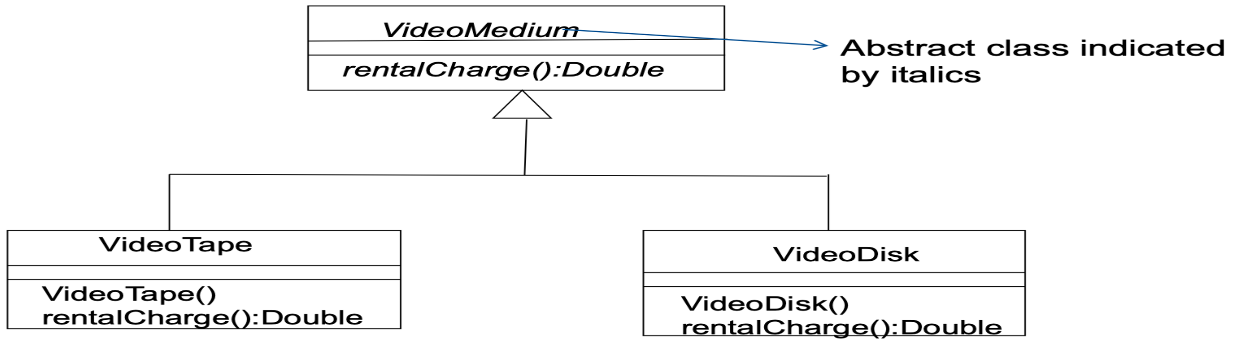
Justifying Payment superclass:



UML Class Diagram

Abstract class and operations (more in Week 8):

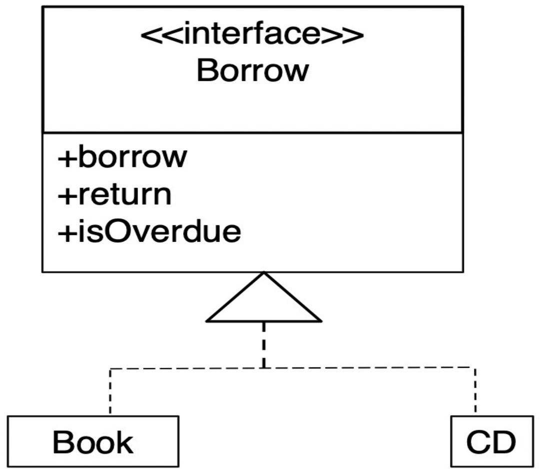
- An abstract class is a parent class that will not have direct instance objects. Only subclasses of the abstract parent class can be instantiated.
- A class is abstract because at least one of its operations is abstract. An abstract operation has its name and signature defined in the abstract parent class, but the implementation of the operation is deferred to concrete child classes.
- Abstract classes are very useful in modelling as they create a high-level modelling vocabulary.



UML Class Diagram

Interface/Realization (more in Week 8):

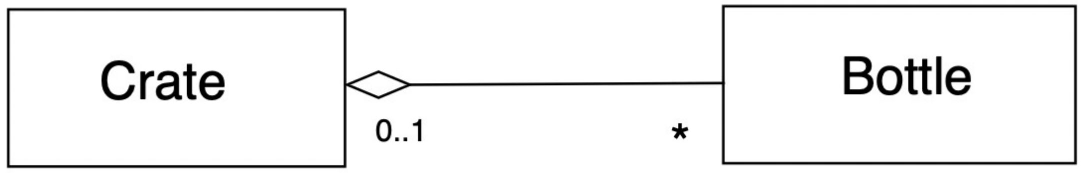
- An interface is a class with no attributes.
- An interface can't be instantiated-it simply declares a contract that may be realized by zero or more classes. The idea is to separate the specification from its implementation.
- Interface only defines a specification for features and it never implies any particular implementation.
- The class implementing an interface has a **realization** relationship with the interface.



UML Class Diagram

Aggregation:

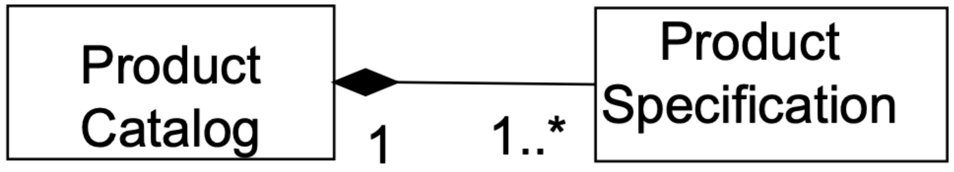
- Aggregation is a type of whole-part relationship in which the aggregate is made up of many parts.
- Signified with a hollow diamond.
- It implies that the part may be in many composite instances.
- For example, at any one point in time, a bottle is part of 0 or 1 crate.



UML Class Diagram

Composition:

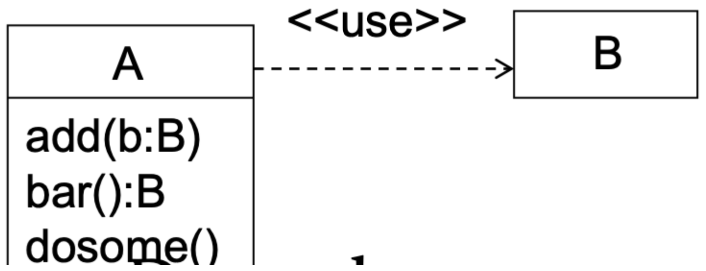
- A stronger form of aggregation is known as composition.
- Composition means that the multiplicity at the composite end may be at most one (signified with a filled diamond).
- A ProductCatalog is composed of various ProductSpecification items.
- There is a create-delete dependency of the ProductSpecification items on ProduceCatelog; their lifetime is bound within the lifetime of the ProduceCatelog.



UML Class Diagram

Dependency:

- A dependency indicates a relationship between two or more model elements whereby a change to one element (supplier) may affect or supply information needed by the other element (client).
- The most common dependency stereotype is ‹‹use››, which simply states that the client makes use of the supplier in some way.

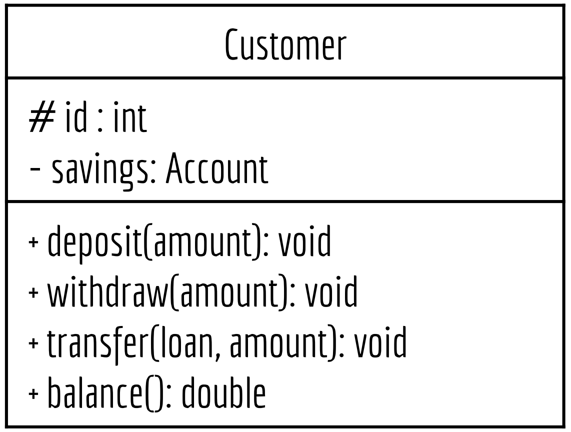


UML Class Diagram

Visibility

Object-oriented design requires that class-diagram denotes visibility of attributes and operations in all classes. There are four types of visibility modifiers:

- The + modifier indicates that an attribute of operation is public
- The - modifier indicates that an attribute of operation is private
- The # modifier indicates that an attribute of operation is protected
- The ~ modifier indicates that an attribute of operation is a package



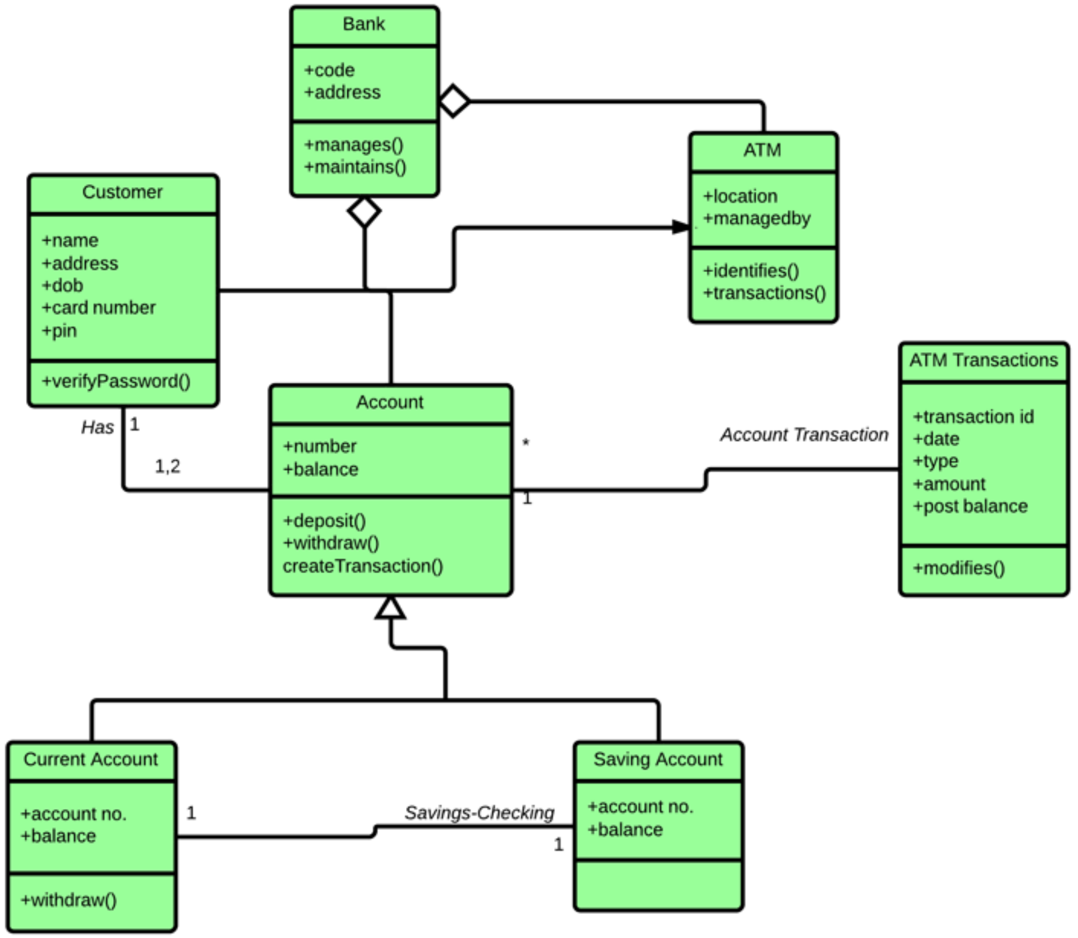
UML Class Diagram Example

Class-Diagram: ATM System Example

Previously, your team used software development process methods to analyze the case-study requirements, test the requirements, map the requirements into user-story table and create a use-case diagram model.

The objective this iteration is to develop complete class diagram for the DeskBankApp application.

This diagram is an example of ATM class-diagram. Use this diagram as a reference to develop the DeskBankApp diagram solution.



References

- Introduction to Software Engineering, By Elvis C. Foster
- Chapter 4,6,7, Object-Oriented Systems Analysis and Design. Noushin Ashrafi and Hessam Ashrafi
- Object-Oriented Design with UML and Java, By Kenneth A. Barclay, and J. Savage.