Class Diagrams

Dr. Jason Barron

South East Technological University

October 17, 2022

What is a Class Diagram?

- A Class Diagram is a diagram describing the structure of a system
- Class diagrams show the classes of the system, their inter-relationships (including inheritance, aggregation, and association), and the operations and attributes of the classes
- Class diagrams are used for a wide variety of purposes, including both conceptual/domain modelling and detailed design modelling

Elements of a UML Class Diagram

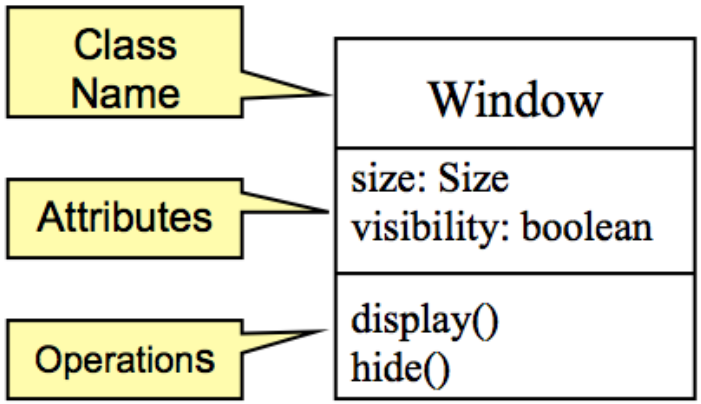
- Class
- Attributes
- Operations
- Relationships
  - Associations
  - Generalisation
  - Realisation
  - Dependency
- Constraint Rules and Notes

Classes

- An object is any person, place, thing, concept, event, screen, or report applicable to your system
- Objects both know things (they have attributes) and they do things (they have methods)
- A class is a representation of an object and is simply a template from which objects are created
- Classes form the main building blocks of an object-oriented application

Class

- Describes a set of objects having similar:
  - Attributes (status)
  - Operations (behaviour)
  - Relationships with other classes
- Attributes and operations may
  - have their visibility marked:
  - "+" for public
  - "#" for protected
  - "-" for private
  - "∼" for package



Classes

- Classes are typically modelled as rectangles with three sections: the top section for the name of the class, the middle section for the attributes of the class, and the bottom section for the methods of the class
- The initial classes of your model can be identified in the same manner as they are when you are CRC modeling, as will the initial responsibilities (its attributes and methods)
- Attributes are the information stored about an object (or at least information temporarily maintained about an object), while methods are the things an object or class do
- When drawing a class element on a class diagram, you must use the top compartment, and the bottom two compartments are optional

Class Example

Create Conceptual Class Diagrams

- To create and evolve a conceptual class diagram, you need to iteratively model:
  - Classes
  - Responsibilities
  - Inheritance relationships
  - Composition associations
  - Vocabularies

Create Design Class Diagrams

- To create and evolve a design class diagram, you need to iteratively model:
  - Classes
  - Responsibilities
  - Inheritance relationships
  - Composition associations
  - Interfaces

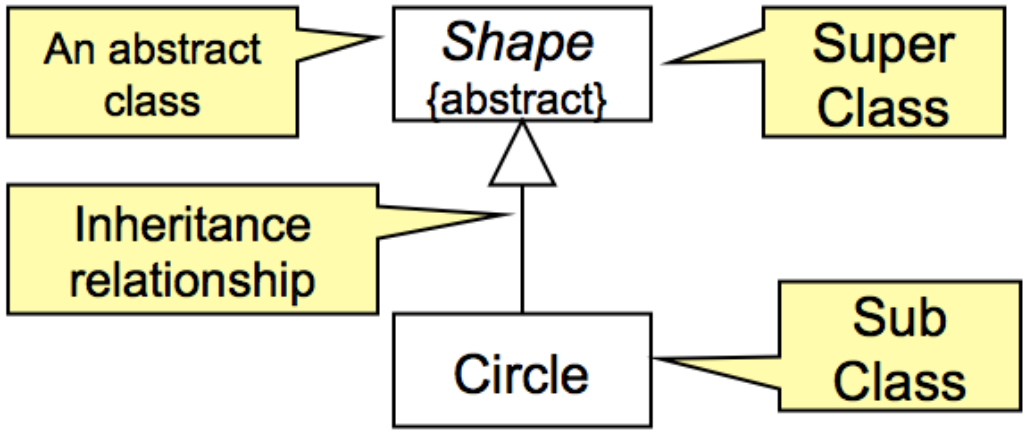
Inheritance

- Inheritance, refers to the ability of one class (child class) to inherit the identical functionality of another class (super class), and then add new functionality of its own
- To model inheritance on a class diagram, a solid line is drawn from the child class (the class inheriting the behaviour) with a closed, unfilled arrowhead (or triangle) pointing to the super class

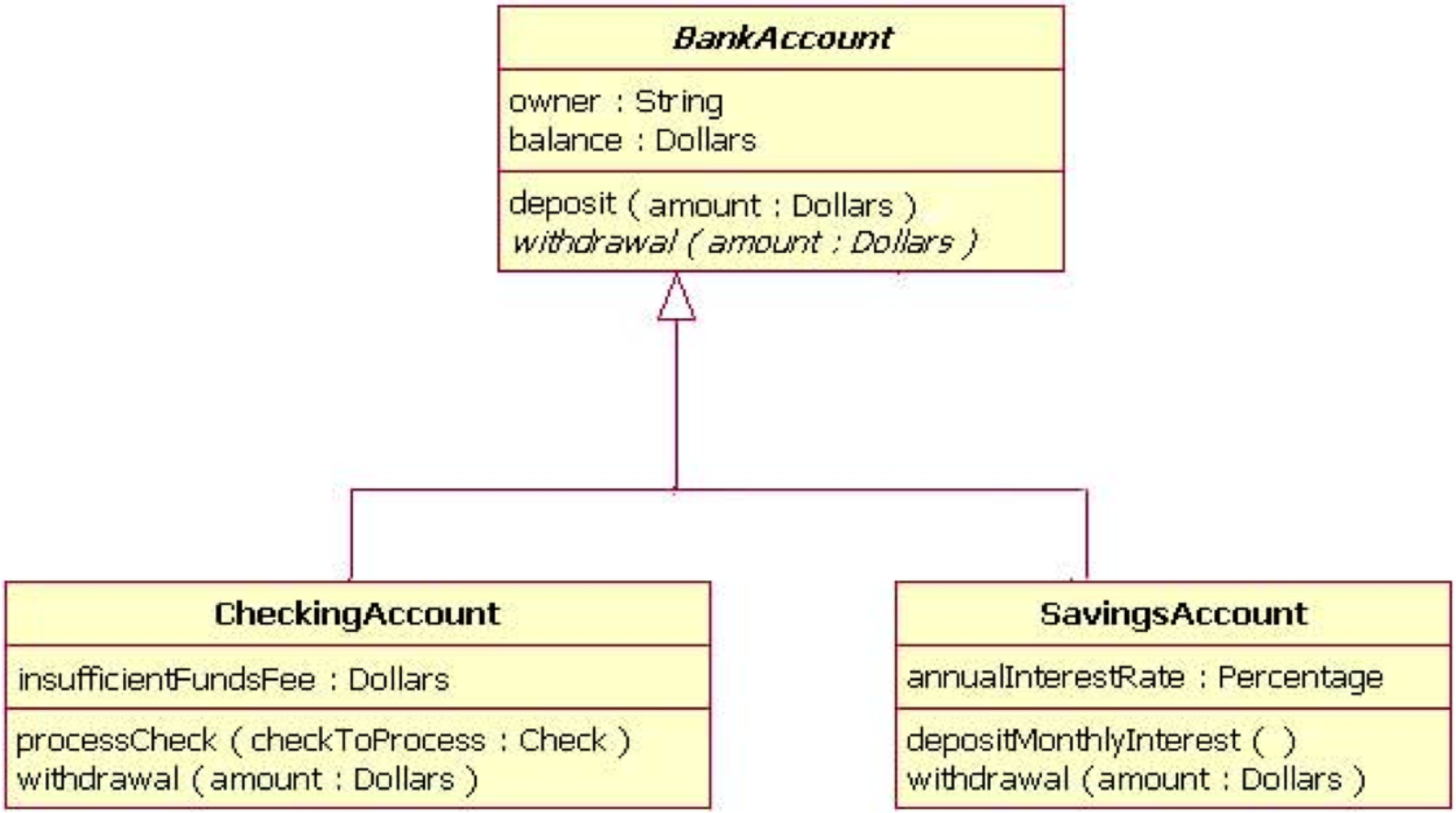
Inheritance Diagram

- Indicates that objects of the specialised class (subclass) are substitutable for objects of the inherited class (super-class)
- **"is kind of"** relationship

{abstract} is a tagged value that indicates that the class is abstract. The name of an abstract class should be italicized



Inheritance Example



Inheritance

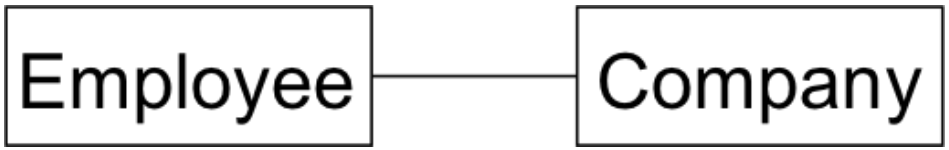
- A sub-class inherits from its super-class
  - Attributes
  - Operations
  - Relationships
- A sub-class may
  - Add attributes and operations
  - Add relationships
  - Refine (override) inherited operations
- An inheritance relationship **may not** be used to model interface implementation

Associations

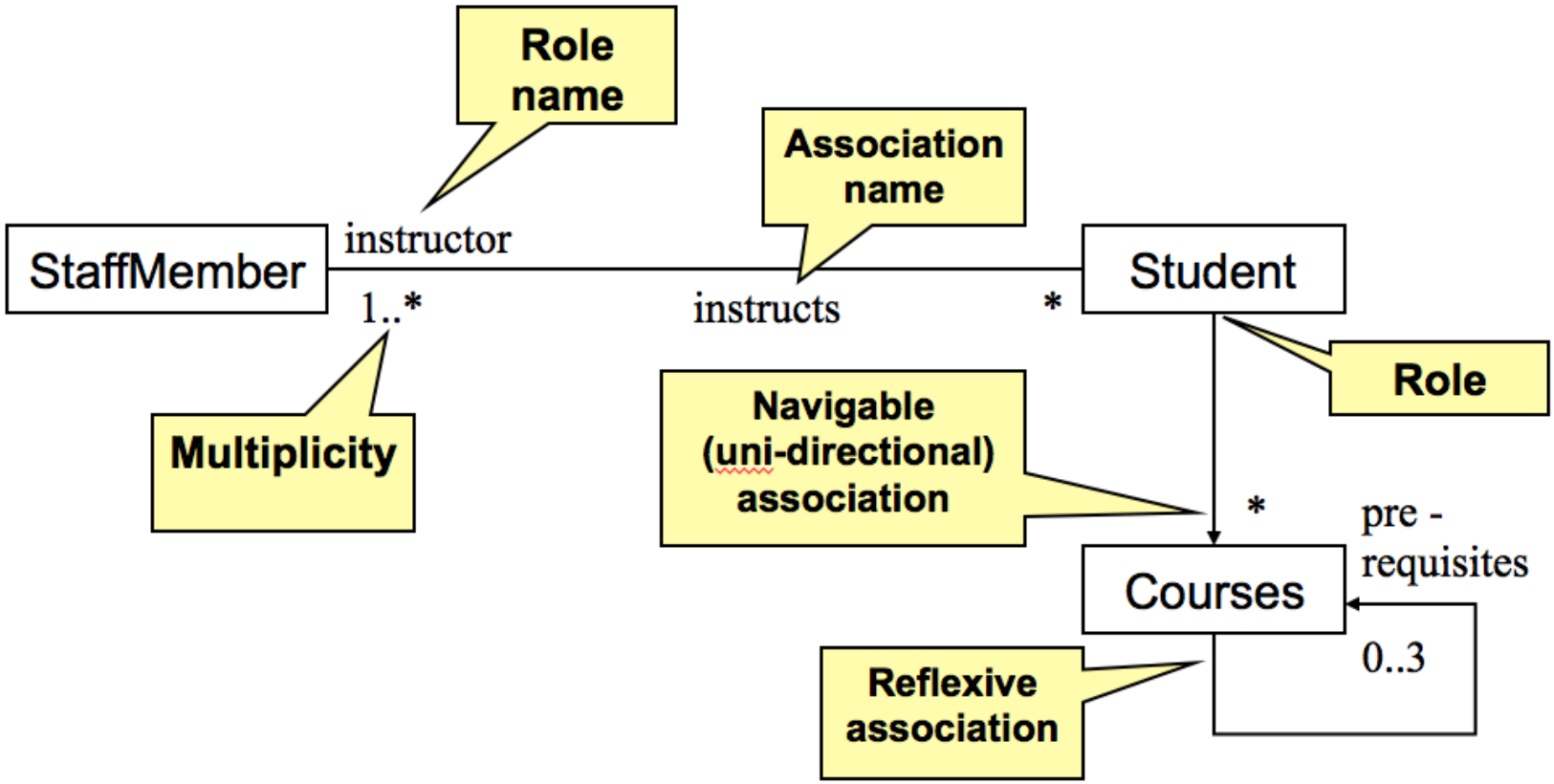
- When you model a system, certain objects will be related to each other, and these relationships themselves need to be modelled for clarity
- There are five types of associations
  - Bi-directional
  - Uni-directional
  - Association class
  - Aggregation
    - Basic aggregation
    - Composition aggregation
  - Reflexive associations

Associations

- An association between two classes indicates that objects at one end of an association "recognise" objects at the other end and may send messages to them
- Example: "An Employee works for a Company"



Associations



Associations

- To clarify its meaning, an association may be named
  - The name is represented as a label placed midway along the association line
  - Usually a verb or a verb phrase
- A **role** is an end of an association where it connects to a class
  - May be named to indicate the role played by the class attached to the end of the association path
    - Usually a noun or noun phrase
    - Mandatory for reflexive associations

Multiplicity

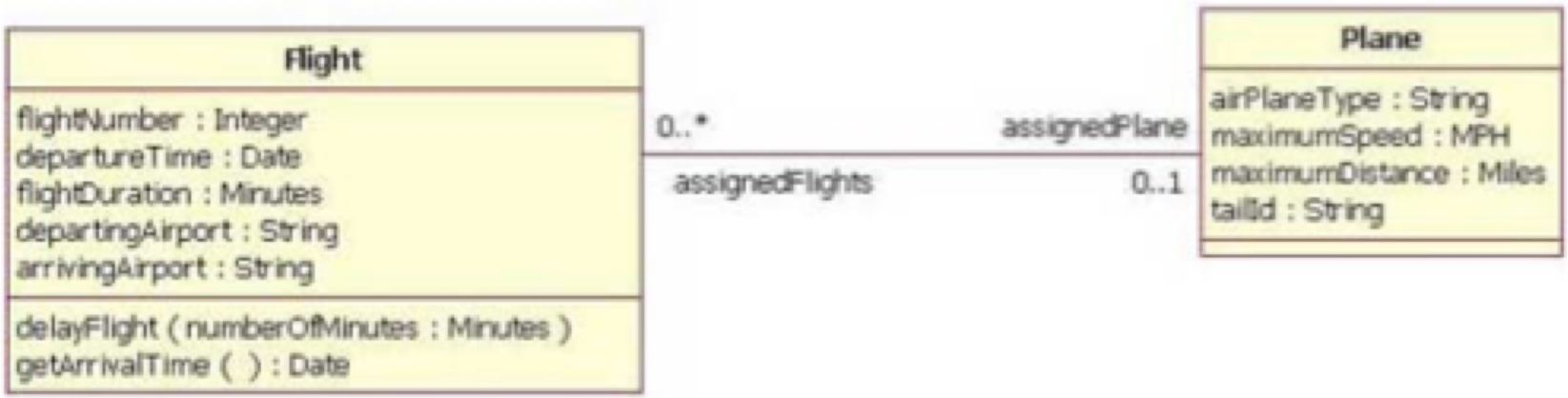
- The number of objects that participate in the association
- Indicates whether or not an association is mandatory

Indicator	**Meaning**
0..1	Zero or One
1	One only
0..*	Zero or more
*	Zero or more
1..*	One or more
3	Three only
0..5	Zero to Five
5..15	Five to Fifteen

Bi-directional (standard) Association

- An association is a linkage between two classes
- Associations are always assumed to be bi-directional
  - This means that both classes are aware of each other and their relationship, unless you qualify the association as some other type
- A bi-directional association is indicated by a solid line between the two classes
- At either end of the line, you place a role name and a multiplicity value

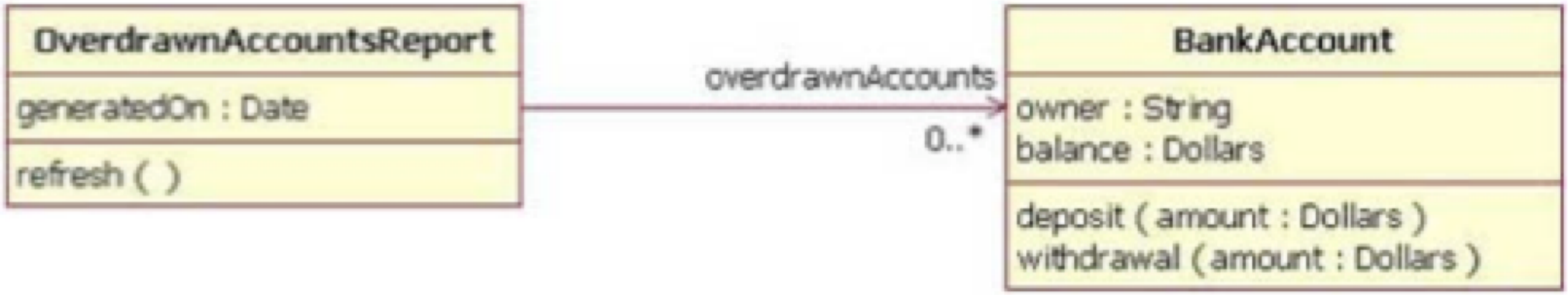
Bi-directional Association Example



Uni-directional Association

- In a uni-directional association, two classes are related, but only one class knows that the relationship exists
- A uni-directional association is drawn as a solid line with an open arrowhead (not the closed arrowhead, or triangle, used to indicate inheritance) pointing to the known class
- Like standard associations, the uni-directional association includes a role name and a multiplicity value, but unlike the standard bi-directional association, the uni-directional association only contains the role name and multiplicity value for the known class

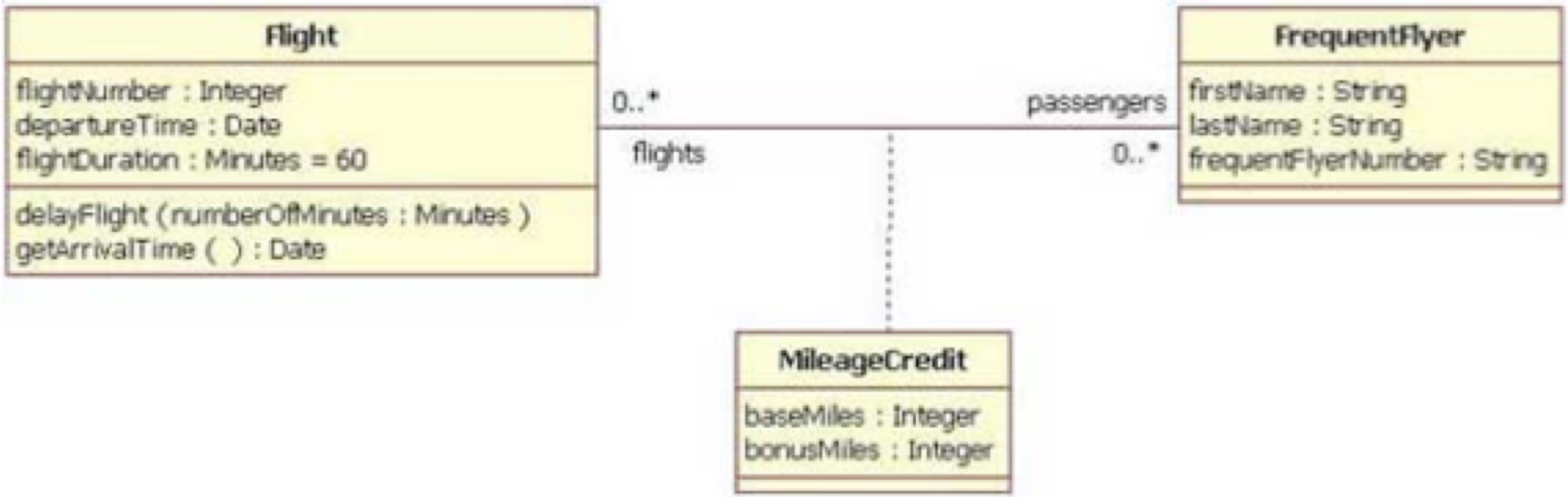
Uni-directional Association Example



Association Class

- In modelling an association, there are times when you need to include another class because it includes valuable information about the relationship
- For this you would use an association class that you tie to the primary association
- An association class is represented like a normal class
- The difference is that the association line between the primary classes intersects a dotted line connected to the association class

Association Class Example



Aggregation

- Aggregation is a special type of association used to model a "whole to its parts" relationship
- In basic aggregation relationships, the lifecycle of a part class is independent from the whole class's lifecycle
- For example, we can think of Car as a whole entity and Car Wheel as part of the overall Car
- The wheel can be created weeks ahead of time, and it can sit in a warehouse before being placed on a car during assembly
- In this example, the Wheel class's instance clearly lives independently of the Car class's instance

Aggregation

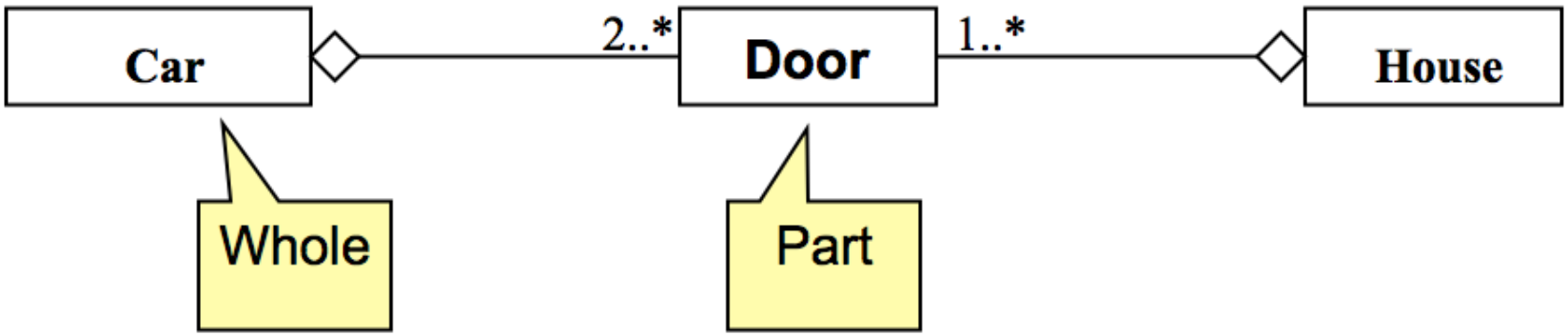
- However, there are times when the part class's lifecycle is not independent from that of the whole class - this is called composition aggregation
- Consider, for example, the relationship of a company to its departments
- Both Company and Departments are modelled as classes, and a department cannot exist before a company exists
- Here the Department class's instance is dependent upon the existence of the Company class's instance

Aggregation

- Aggregation tests:
  - Is the phrase part of used to describe the relationship?
    - A door is part of a car
  - Are some operations on the whole automatically applied to its parts?
    - Move the car, move the door
- Are some attribute values propagated from the whole to all or some of its parts?
  - The car is blue, therefore the door is blue
- Is there an intrinsic asymmetry to the relationship where one class is subordinate to the other?
  - A door **is** part of a car. A car **is not** part of a door

Basic Aggregation

- A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts
  - Models a "is a part-part of" relationship



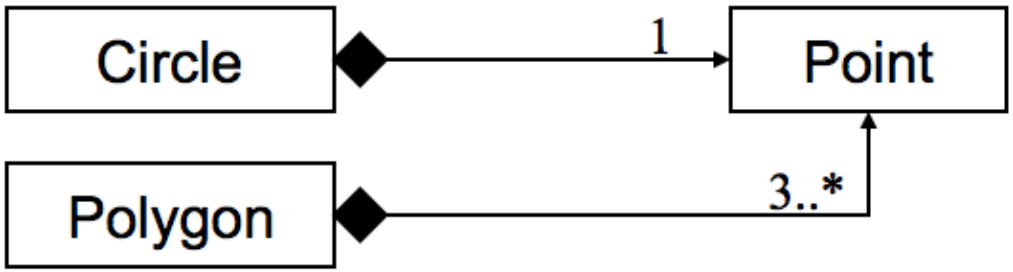
Basic Aggregation

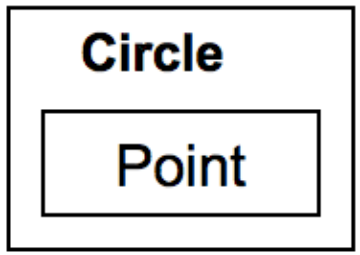
- An association with an aggregation relationship indicates that one class is a part of another class
- In an aggregation relationship, the child class instance can outlive its parent class
- To represent an aggregation relationship, you draw a solid line from the parent class to the part class, and draw an unfilled diamond shape on the parent class's association end

Basic Aggregation Example

Composition Aggregation

- A strong form of aggregation
  - The whole is the sole owner of its part
    - The part object may belong to only one whole
  - Multiplicity on the whole side must be zero or one
  - The life time of the part is dependent upon the whole
    - The composite must manage the creation and destruction of its parts





Composition Aggregation

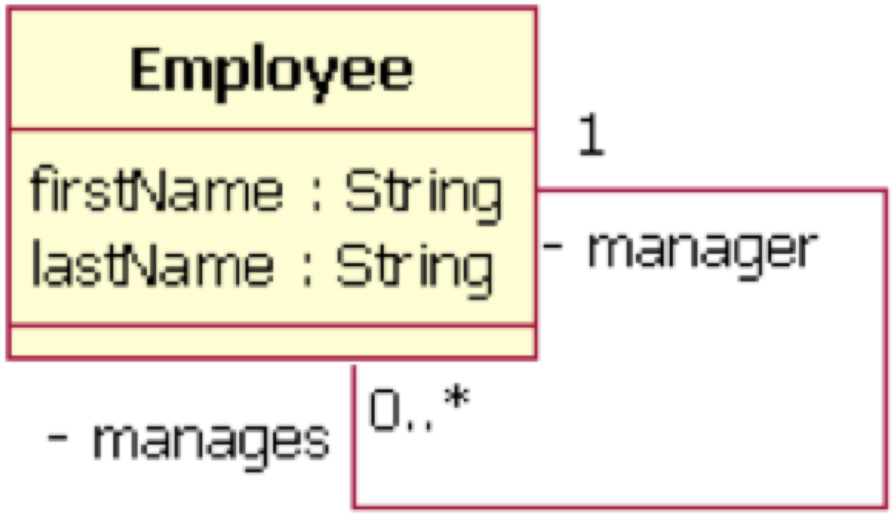
- The composition aggregation relationship is just another form of the aggregation relationship, but the child class's instance lifecycle is dependent on the parent class's instance lifecycle
- Another important feature of composition aggregation is that the part class can only be related to one instance of the parent class (e.g. the Company class in our example)

Composition Aggregation Example

Reflexive Associations

- All our examples have shown a relationship between two different classes
- However, a class can also be associated with itself, using a reflexive association
- This may not make sense at first, but remember that classes are abstractions. An Employee class could be related to itself through the manager/manages role
- When a class is associated to itself, this does not mean that a class's instance is related to itself, but that an instance of the class is related to another instance of the class

Reflexive Associations

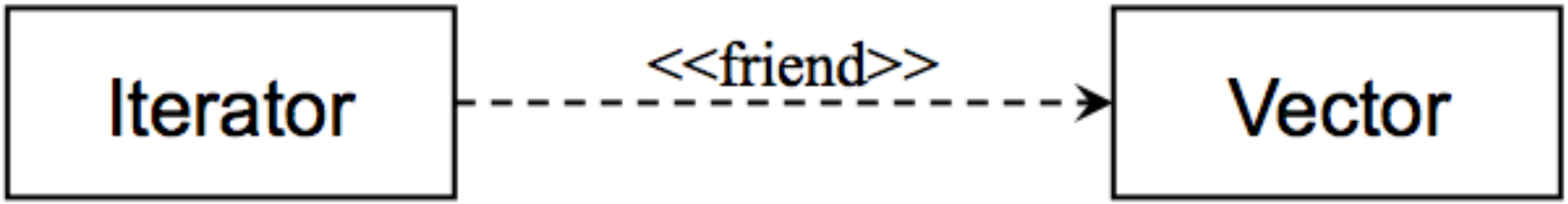


Realisation

- A realisation relationship indicates that one class implements a behaviour specified by another class (an interface or protocol)
- An interface can be realised by many classes
- A class may realise many interfaces

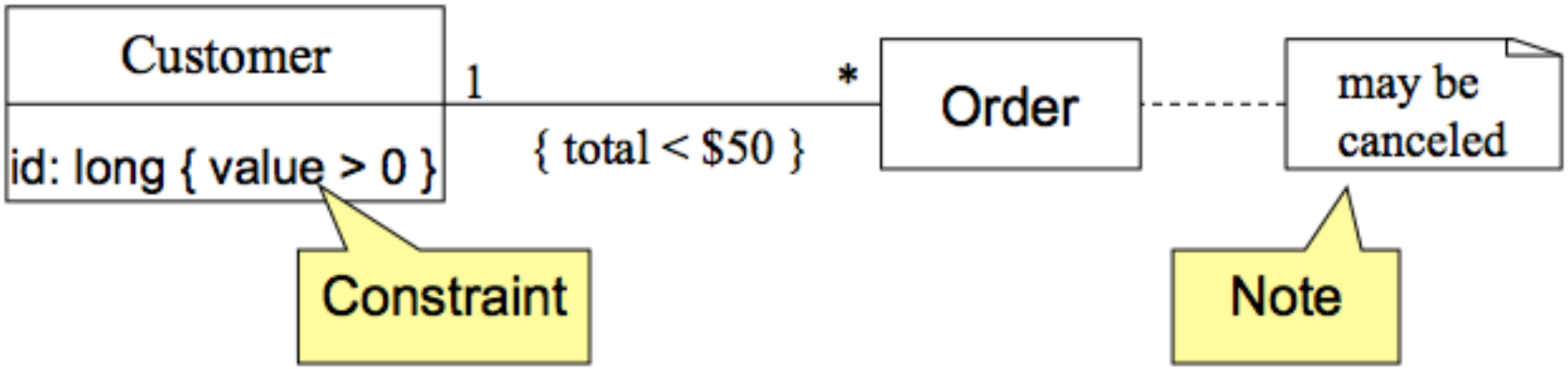
Dependency

- Dependency is a weaker form of relationship which indicates that one class depends on another because it uses it at some point in time
- One class depends on another if the independent class is a parameter variable or local variable of a method of the dependent class
- This is different from an association, where an attribute of the dependent class is an instance of the independent class

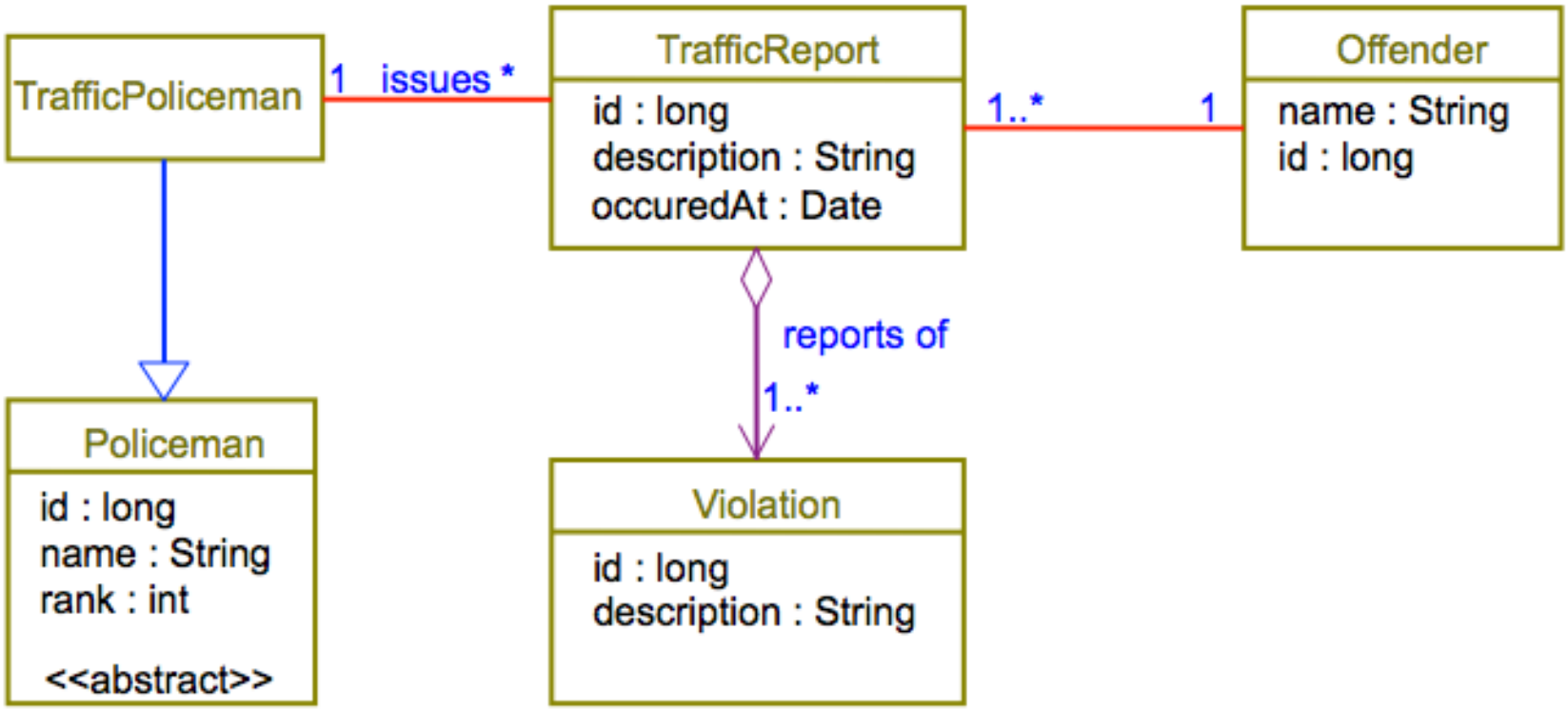


Constraint Rules and Notes

- **Constraints** and **notes** annotate among other things associations, attributes, operations and classes
- Constraints are semantic restrictions noted as Boolean expressions
  - UML offers many pre-defined constraints



Traffic Violation Report System Example



Analysis Classes Elicitation

- Consider main perspectives of the system
- Interface between the system and its actors
  - Protocols for information exchange
  - Don't concentrate on visual aspects
- Data the system uses
  - The core of the system, key concepts
- The system logic
  - Controls and coordinates the behaviour
  - Delegates the work to other classes
  - Decouples interface and data classes

Tips

- Don't try to use all the various notations
- Don't draw models for everything, concentrate on the key areas

Analysis Classes

- A technique for finding analysis classes which uses three different perspectives of the system:
  - The boundary between the system and its actors (Boundary)
  - The information the system uses (Entity)
  - The control logic of the system (Control)

Boundary Classes

- Models the interaction between the systems surroundings and its inner workings
  - User interface classes
    - Concentrate on what information is presented to the user
    - Don't concentrate on user interface details
  - System/Device interface classes
    - Concentrate on what protocols must be defined
    - Don't concentrate on how the protocols are implemented
- Boundary classes are environment dependent
  - UI dependent
  - Communication protocol dependent

Entity Classes

- Models the key concepts of the system
- Usually models information that is persistent
- Contains the logic that solves the system problem
- Is environment independent
- Can be used in multiple use cases
  - For example: Violation, Report, Offender

Control Classes

- Controls and coordinates the behaviour of a use case
- Delegates the work of the use case to classes
  - A control class should tell other classes to do something and should never do anything except for directing
- Control classes decouple boundary and entity classes
- Often, there is one control class per use case

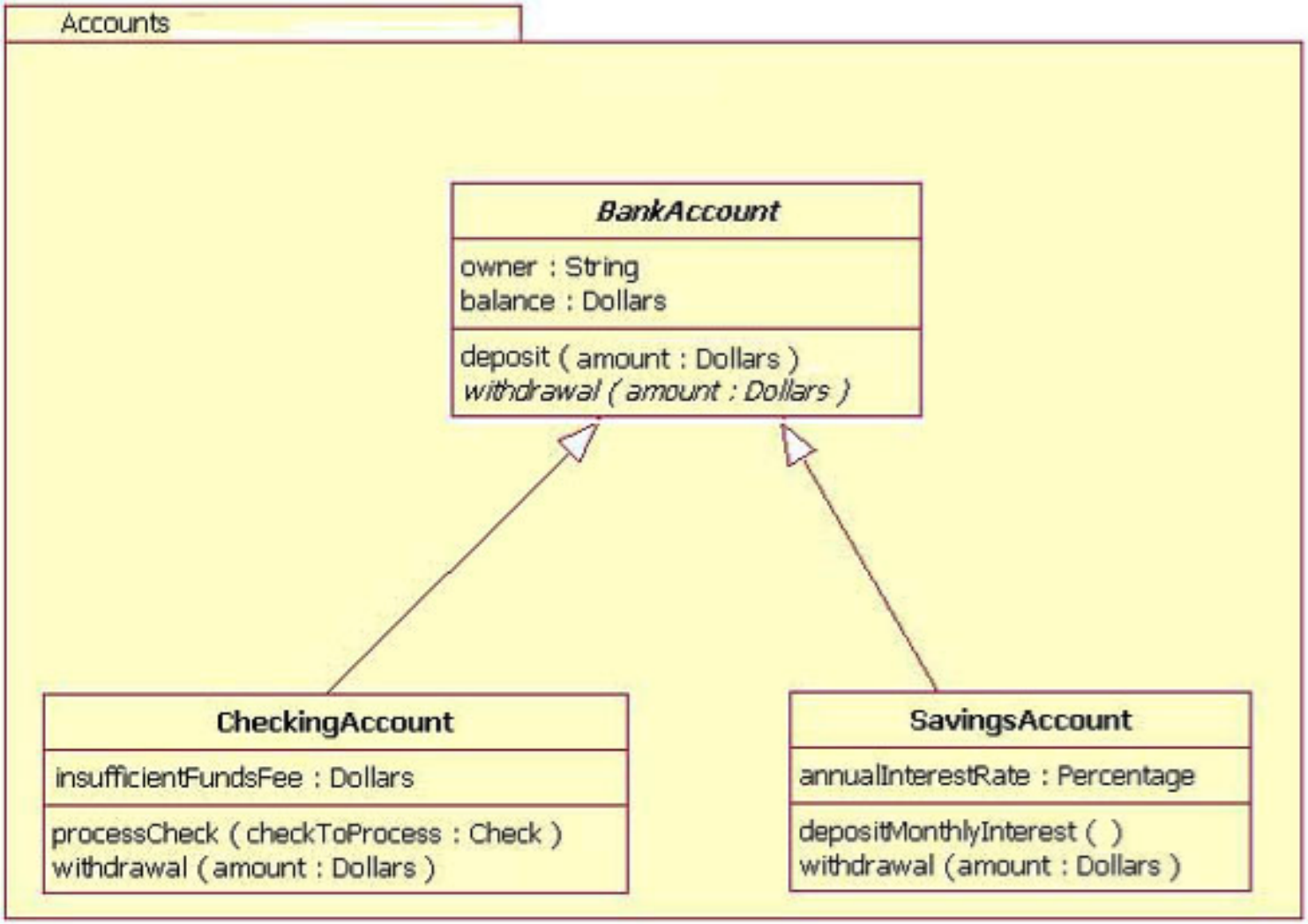
Packages

- Packages enable modellers to organise the model's classifiers into namespaces, which is sort of like folders in a filing system
- Dividing a system into multiple packages makes the system easier to understand, especially if each package represents a specific part of the system
- In cases where your packages have lots of classes, it is better to use multiple topic-specific class diagrams instead of just producing one large class diagram

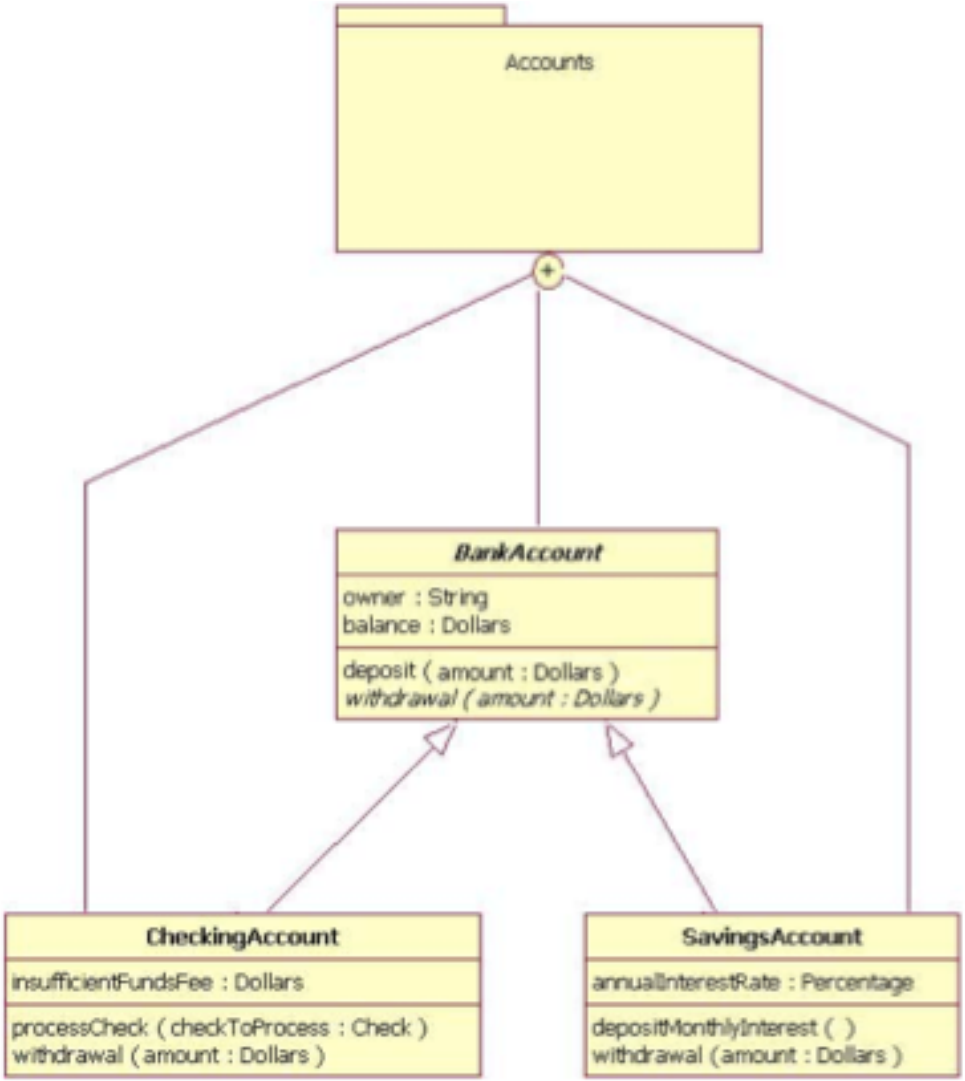
Packages

- There are two ways of drawing packages on diagrams
- Both ways begin with a large rectangle with a smaller rectangle (tab) above its upper left corner
- If the modeller decides to show the package's members within the large rectangle, then all those members need to be placed within the rectangle
- If the modeller decides to show the package's members outside the large rectangle then all the members that will be shown on the diagram need to be placed outside the rectangle
- To show what classifiers belong to the package, a line is drawn from each classifier to a circle that has a plus sign inside the circle attached to the package

Packages


Accounts
BankAccount
owner : String
balance : Dollars
deposit (amount : Dollars)
withdrawal (amount : Dollars)
CheckingAccount
insufficientFundsFee : Dollars
processCheck (checkToProcess : Check)
withdrawal (amount : Dollars)
SavingsAccount
annualInterestRate : Percentage
depositMonthlyInterest ( )
withdrawal (amount : Dollars)

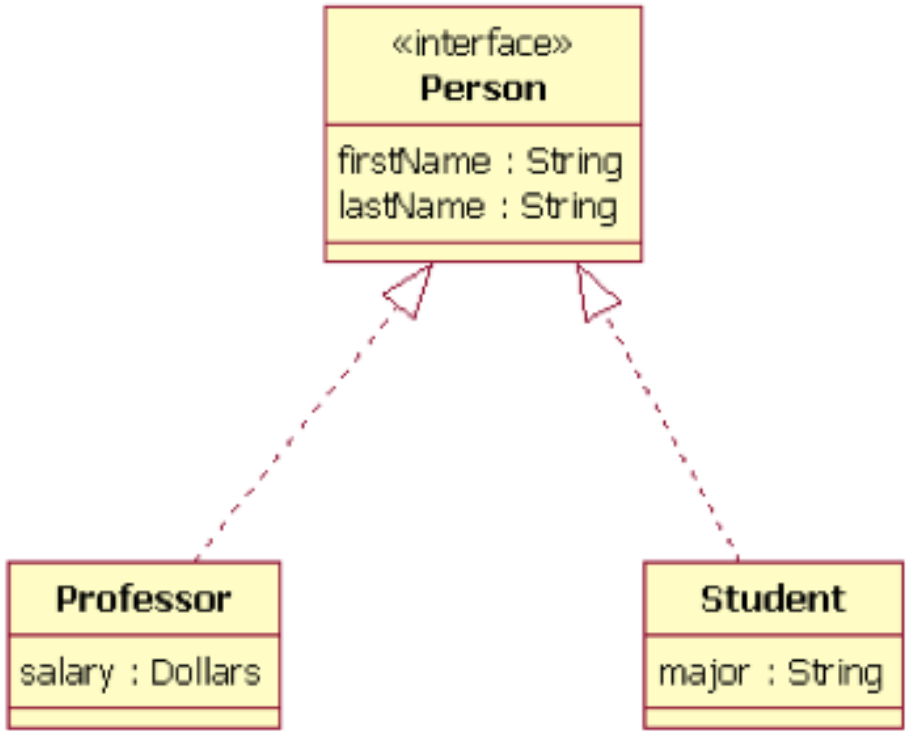
Packages


Accounts
BankAccount
owner : String
balance : Dollars
deposit (amount : Dollars)
withdrawal (amount : Dollars)
CheckingAccount
insufficientFundsFee : Dollars
processCheck (checkToProcess : Check)
withdrawal (amount : Dollars)
SavingsAccount
annualInterestRate : Percentage
depositMonthlyInterest ( )
withdrawal (amount : Dollars)

Interfaces

- A class and an interface differ
  - A class can have an actual instance of its type, whereas an interface must have at least one class to implement it
- In UML 2, an interface is considered to be a specialisation of a class modelling element
- Therefore, an interface is drawn just like a class, but the top compartment of the rectangle also has the text "≪interface≫

Interfaces


«interface»
Person
firstName : String
lastName : String
Professor
salary : Dollars
Student
major : String

References

- Donald Bell, (2004), The class diagram, An introduction to structure diagrams in UML 2.
  https://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/