School of Computing and Informatics

Advance Software Engineering Course

UML Class Diagram Lab

Prepared by:

Eng. Shatha Al Hawawsheh

Lab Objectives:

- To understand Class Diagram concept.
- To familiar with class diagram annotations.
- To draw class diagram based on given requirements.

Class Diagram Overview:

The class diagram is a structural diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing and documenting different aspects of a system but also for constructing executable code of the software application. The class diagram describes the attributes and operations of a class and the constraints imposed on the system. The class diagrams are widely used in the modelling of object-oriented systems because they are the only UML diagrams which can be mapped directly with object-oriented languages.

The class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a static diagram.

Purpose:

The purpose of the class diagram is to model the static view of an application. The class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction.

UML diagrams like activity diagram, sequence diagram can only give the sequence flow of the application, but class diagram is a bit different. So, it is the most popular UML diagram in the coder community So the purpose of the class diagram can be summarized as:

• Analysis and design of the static view of an application.

• Describe responsibilities of a system.

• Base for component and deployment diagrams.

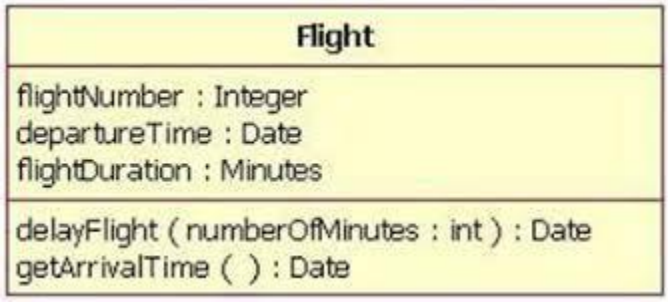
• Forward and reverse engineering.

Class Diagram Annotations:

Class:

The UML representation of a class is a rectangle containing three compartments stacked vertically. The top compartment shows the class's name. The middle compartment lists the class's attributes. The bottom compartment lists the class's operations. When drawing a class element on a class diagram, must use the top compartment, and the bottom two compartments are optional. (The bottom two would be unnecessary on a diagram

depicting a higher level of detail in which the purpose is to show only the relationship between the classifiers.)



Class Attribute List:

The attribute section of a class (the middle compartment) lists each of the class's attributes on a separate line. The attribute section is optional, but when used it contains each attribute of the class displayed in a list format. The line uses the following format:

name : attribute type
flightNumber : Integer

Continuing with our Flight class example, we can describe the class's attributes with the attribute type information, as shown in table below. The Flight class's attribute names with their associated types.

Attribute Name	Attribute Type
FlightNumber	Integer
departureTime	Date
flightDuration	Minutes

In business class diagrams, the attribute types usually correspond to units that make sense to the likely readers of the diagram (i.e., minutes, dollars, etc.). However, a class diagram that will be used to generate code needs classes whose attribute types are limited to the types provided by the programming language, or types included in the model that will also be implemented in the system. Sometimes it is useful to show on a class diagram that a particular attribute has a default value. (For example, in a banking account application a new bank account would start off with a zero balance.) The UML specification allows for the identification of default values in the attribute list section by using the following notation:

name : attribute type = default value

balance : Dollars = 0

Showing a default value for attributes is optional; Figure below shows a Bank Account class with an attribute called balance, which has a default value of 0.

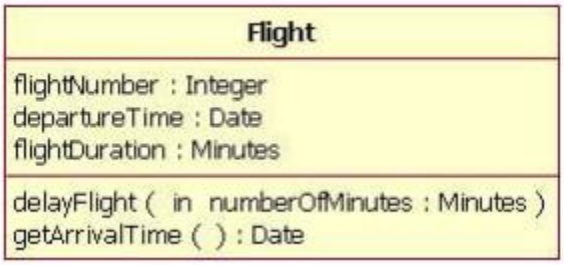
Class Operations List:

The class's operations are documented in the third (lowest) compartment of the class diagram's rectangle, which again is optional. Like the attributes, the operations of a class are displayed in a list format, with each operation on its own line. Operations are documented using the following notation:

name(parameter list) : type of value returned

Operation Name	Parameters Return		Value Type
	Name	Type	
delayFlight	numberOf Minutes	Minutes	N/A
getArrivalTime	N/A		Date

Figure below shows that the delayFlight operation has one input parameter (numberOfMinutes) of the type Minutes. However, the delayFlight operation does not have a return value. [Note: The delayFlight does not have a return value because I made a design decision not to have one. One could argue that the delay operation should return the new arrival time, and if this were the case, the operation signature would appear as delayFlight (numberOfMinutes :Minutes) : Date. When an operation has parameters, they are put inside the operation's parentheses; each parameter uses the format "parameter name : parameter type".

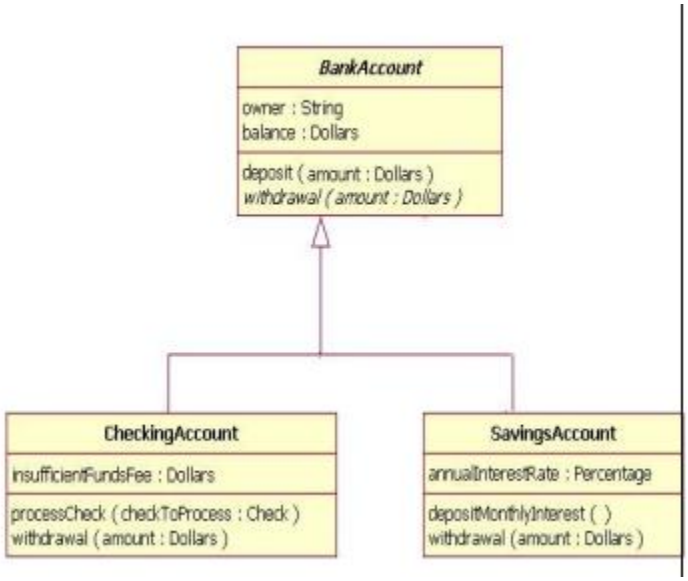


When documenting an operation's parameters, you may use an optional indicator to show whether or not the parameter is input to, or output from, the operation. This optional indicator appears as an "in" or "out" as shown in the operations compartment in Figure above. Typically, these indicators are unnecessary unless an older programming language

such as Fortran will be used, in which case this information can be helpful. However, in C++ and Java, all parameters are "in" parameters and since "in" is the parameter's default type according to the UML specification, most people will leave out the input/output indicators.

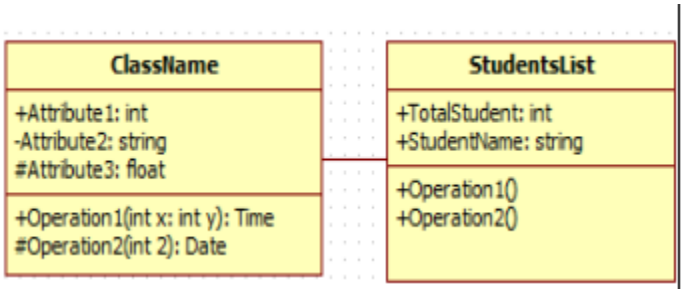
Inheritance:

A very important concept in objectoriented design, 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 behavior) with a closed, unfilled arrowhead (or triangle) pointing to the super class. Consider types of bank accounts: Figure below shows how both CheckingAccount and SavingsAccount classes inherit from the BankAccount class.



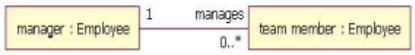
Associations:

Basic Association Associations represent static relationships between classes. Place association names above, on, or below the association line. Use a filled arrow to indicate the direction of the relationship. Place roles near the end of an association. Roles represent the way the two classes see each other.



Multiplicity (Cardinality)

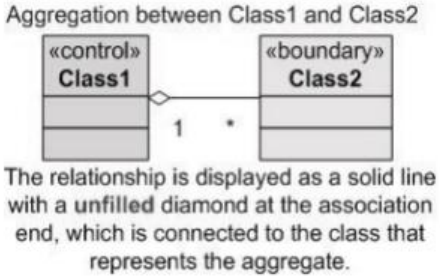
- Place multiplicity notations near the ends of an association.
- These symbols indicate the number of instances of one class linked to one instance of the other class.
- For example, one manager could assigned for zero or many team member.



0	Collection must be empty
1	Exactly one instance
5	Exactly 5 instances
*	Zero or more instances
0..1	No instances or one instance
1..1	Exactly one instance
0..*	Zero or more instances
1..*	At least one instance
m..n	At least m but no more than n instances

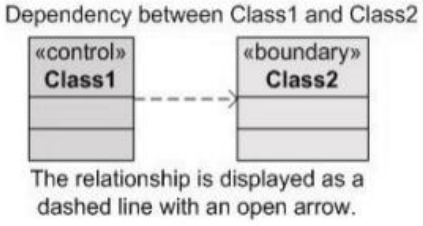
Aggregation

A special type of association. It represents a "part- of" relationship. • Class2 is part of Class1. Many instances (denoted by the *) of Class2 can be associated with Class1. • Objects of Class1 and Class2 have separate lifetimes.



Composition

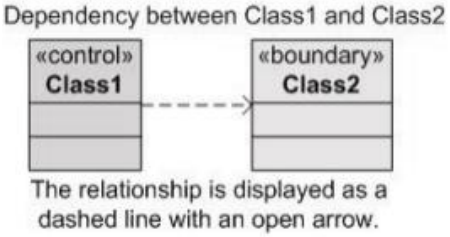
• A special type of aggregation where parts are destroyed when the whole is destroyed.
• Objects of Class2 live and die with Class1. Class2 cannot stand by itself.



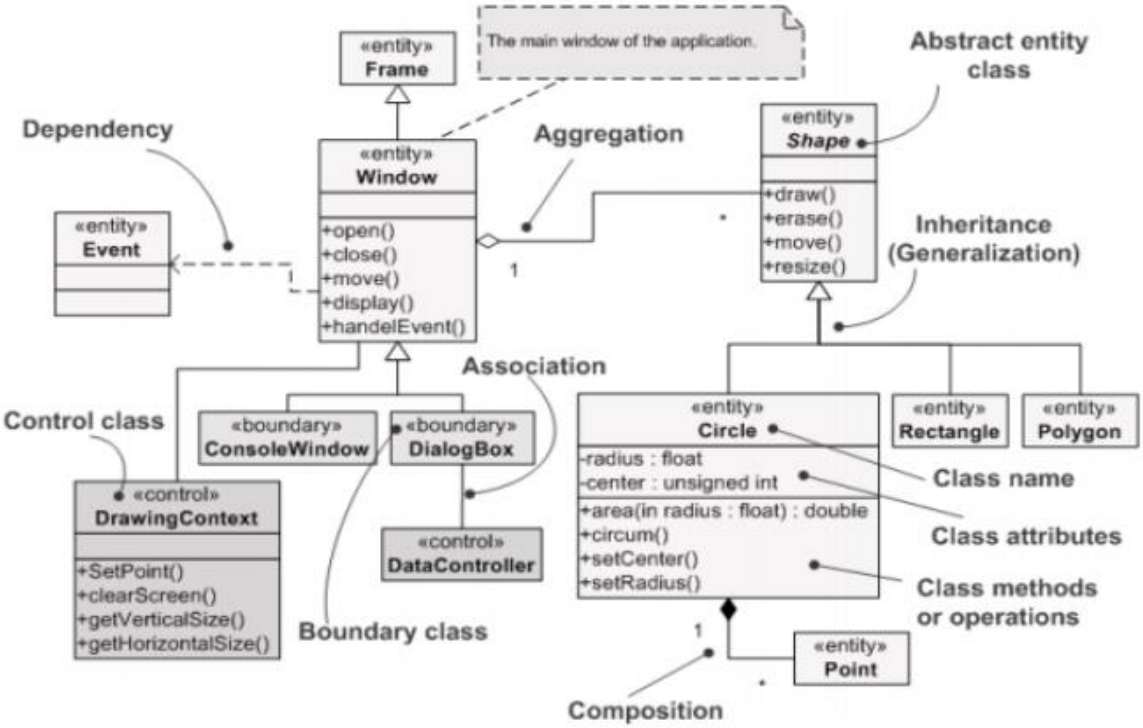
Dependency:

• Exists between two classes if changes to the definition of one may cause changes to the other (but not the other way around).

• Class1 depends on Class2



Example of Complete Class Diagram:



The following can be observed from the above diagram:

1. Shape is an abstract class. It is shown in Italics.
2. Shape is a superclass. Circle, Rectangle and Polygon are derived from Shape. In other words,a Circle is-a Shape. This is a generalization / inheritance relationship.
3. There is an association between DialogBox and DataController.
4. Shape is part-of Window. This is an aggregation relationship. Shape can exist without Window.
5. Point is part-of Circle. This is a composition relationship. Point cannot exist without a Circle.
6. Window is dependent on Event. However, Event is not dependent on Window.
7. The attributes of Circle are radius and center. This is an entity class.

8. The method names of Circle are area(), circum(), setCenter() and setRadius().
9. The parameter radius in Circle is an in parameter of type float.
10. The method area() of class Circle returns a value of type double.
11. The attributes and method names of Rectangle are hidden. Some other classes in the diagram also have their attributes and method names hidden.

✓ Lab Exercises:

1- A company consists of multiple departments. Departments are located in one or more offices. One office acts as headquarter. Each department has a manager who is recruited from the set of employees. Draw a class diagram which consists of all the classes in this system, their attributes, relationships between the classes, multiplicity specifications, and other model elements that you find appropriate.

Notes:

- ❖ A company consists of multiple departments and multiple offices.
- ❖ Departments are located in one or more offices.
- ❖ One office acts as headquarter.
- ❖ One or more employee works at the department.
- ❖ Each department has a manager who is recruited from the set of employees.

2- .Create a class diagram for a Car Engine System. The car engine is constructed by motor, fuel pump and gas tank. The piston and Cylinder is the component of the motor. The motor is connected to the fuel pump from the gas tank. Both Diesel and petrol can be used in this system. The class diagram should contain the following class: Car, Fuel Pump, Gas Tank, Motor, Fuel, Cylinder, Piston, Diesel, Petrol.

3- UniMAP library system is maintain by librarian. The librarian have the access and record of checking availability of books, verify members, issuing book, return book. The member of library have the access of checking availability of the books, issue book, and return book. Fine will be impose if the book returned after the due date. Draw class diagram for above system.

4- . Draw a UML Class Diagram representing the following elements from the problem domain for a hockey league. A hockey league is made up of at least four hockey teams. Each hockey team is composed of six to twelve players, and one player captains the team. A team has a name and a record. Players have a number and a position. Hockey teams play games against each other. Each game has a score and a location. Teams are sometimes lead by a coach. A coach has a level of accreditation and a number of years of experience, and can coach multiple teams. Coaches and players are people, and

people have names and addresses. Draw a class diagram for this information, and be sure to label all associations with appropriate multiplicities.