UTCN
Computer Science Department
Software Design 2017

LABORATORY 1 – REVISION

==========

I. UML Revision

This section focuses on reviewing the basic concepts regarding UML structural and behavioral modeling. The information provided is based on [1].

1. Structural Modeling

Structural modeling aims to capture the static parts of a model, representing elements that are either conceptual or physical. The main elements of structural modeling are the following: classes, interfaces, collaborations, use cases, artifacts and nodes.

1.1 Classes

Definition: A *class* is a description of a set of objects sharing the same attributes, operations and relationships. There are classes that include abstractions that are part of the problem domain and classes that make up an implementation.

Notation and representation: A class is graphically represented as a rectangle (see Figure 1) in which the following information is specified:

- *Class name*
  - Used to distinguish the class from other classes
  - Simple or qualified (prefixed by the name of the package to which the class belongs)
- *Class attribute*
  - Named property describing a range of values that instances of the property may hold
  - Shared by all objects of that class
- *Class operation*
  - Implementation of a service that can be requested from any object of the class to affect behavior
  - Can be specified by stating its signature

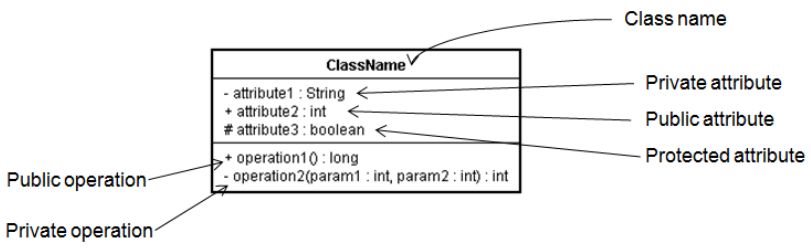

Figure 1. UML class notation

1.2 Relationships

Definition: A relationship models the way things can connect to one another, either logically or physically.

Notation and Representation: A relationship is represented as a path with different kinds of lines used to distinguish the kinds of relationships.

Types: Dependencies, Associations, Generalizations, Realizations

a) *Dependency relationship*

A dependency relationship between two things states that one thing uses the information and services of the other thing, but not necessarily the reverse. In the case of classes, the dependency relationship is used to show that one class uses operations from another class or it uses variables or arguments typed by the other class => if the used class changes, the operation of the other class may be affected as well.

The dependency relationships are represented using dashed directed lines, directed to the things being depended on (see Figure 2).

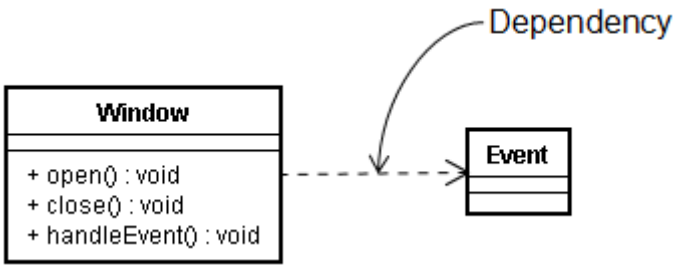

Figure 2. UML dependency relationship

b) *Association relationship*

An association relationship specifies that objects of one thing are connected to objects of another thing. Graphically, an association relationship is represented using a solid line (see Figure 3). An association can have a *name* used to describe the nature of the relationship. When a class participates in an association, it has a specific *role* that it plays in that relationship.

The *multiplicity* of an association's role represents a range of integers specifying the possible size of the set of related objects.

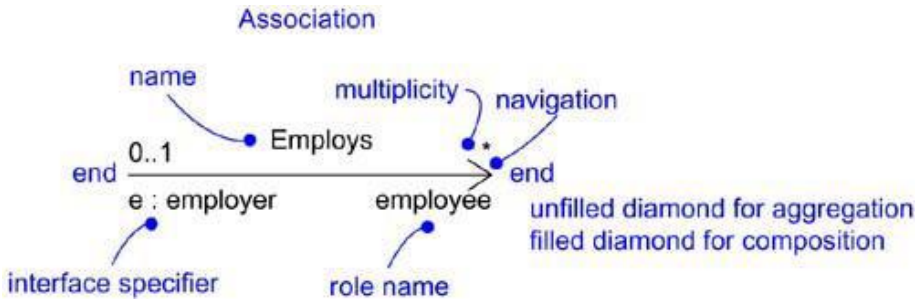

Figure 3. UML association relationship

A plain association between two classes represents a structural relationship between peers, meaning that both classes are conceptually at the same level, no one more important than the other.

Aggregation is a special kind of association used to model a „whole/part" relationship, in which one class represents a larger thing („the whole"), which consists of smaller things (the „parts"). The aggregation relationship is graphically represented using a solid line adorned with an unfilled diamond at the whole end (see Figure 4).

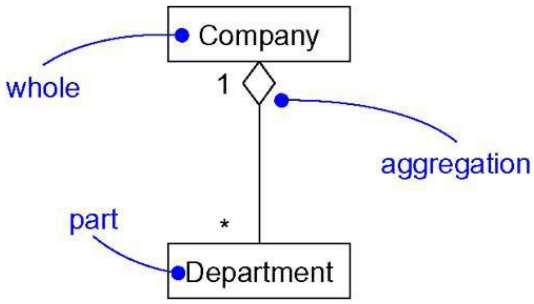

Figure 4. UML aggregation relationship

Composition is a variation of simple aggregation, which introduces a strong ownership and coincident lifetime as part of the whole. Parts with non-fixed simplicity may be created after the composite itself, but once created they live and die with it. Such parts can also be explicitly removed before the death of the composite. The composition relationship is graphically represented using a solid line adorned with a filled diamond at the whole end (see Figure 5).

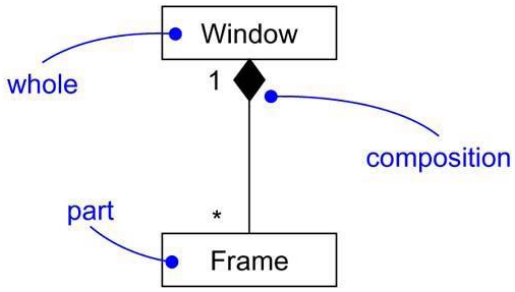

Figure 5. UML composition relationship

c) Generalization relationship

A generalization relationship is established between a general kind of thing (superclass) and a more specific kind of thing (subclass) => the child is substitutable for a declaration of the parent. The child has attributes and operations in addition to those found in its parents. An implementation of an operation in a child overrides an implementation of the same operation of the parent => *polymorphism*.

The generalization relationships are represented as solid directed lines with a large unfilled triangular arrowhead pointing to the parent (see Figure 6).

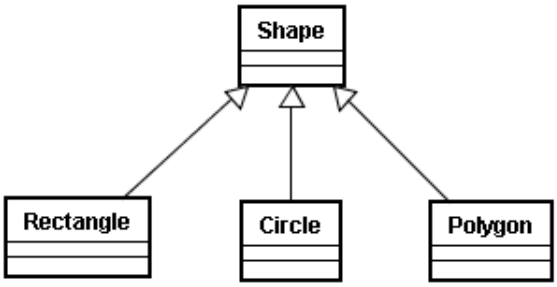

Figure 6. UML generalization relationship

d) Realization relationship

A realization relationship is a semantic relationship between classifiers (e.g classes, interfaces, collaborations use cases) in which one classifier specifies a contract that another classifier guarantees to carry out. Graphically, a realization is represented as a dashed directed line with a large open arrowhead pointing to the classifier that specifies tha contract. Generrally, a realization is used in the context of interfaces (see Figure 7) and collaborations (see Figure 8).

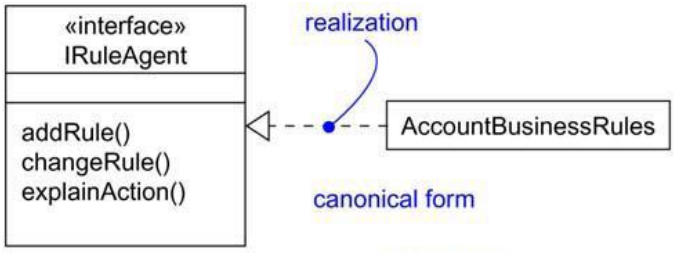

Figure 7. Realization of an interface

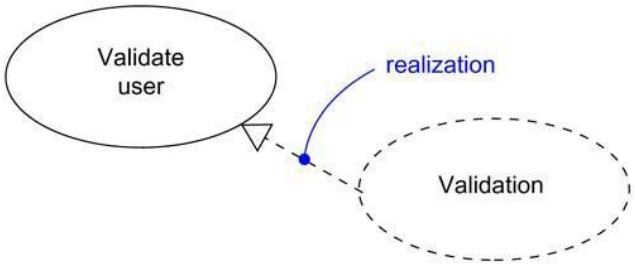

Figure 8. Realization of a Use Case

1.3 Class Diagrams

A class diagram shows a set of classes, interfaces and collaborations and their relationships. Class diagrams may also contain packages or sub-systems, both of which are used to group elements of the model.

Class diagrams are used to model the static design view of a system and are typically used in one of the following three ways: (1) to model the vocabulary of a system, (2) to model simple collaborations and (3) to model a logical database schema.

2. Behavioral Modeling

Behavioral modeling aims to capture the dynamic parts of a model. These are the verbs of a model, representing behavior over time and space. There are three types of behavioral things: interactions, state machines and activities.

2.1 Interactions

An *interaction* is a behavior that comprises a set of messages exchanged among objects in a set of roles within a context to accomplish a purpose. The objects that participate in an interaction are either concrete things of protypical things.

A *link* is a semantic connection among objects. In general, a link is an instance of an association. A link specifies a path along which one object can dispatch a message to another (or the same) object.

A *message* is a specification of a communication between objects. In the UML the following types of messages can be modeled: *call* – invokes an operation of an object (an object may send a message to itself, resulting in the local invocation of an operation), *return* – returns a value to the caller, *send* – sends a signal to an object, *create* – creates an object, and *destroy* – destroys an object (see Figure 9).

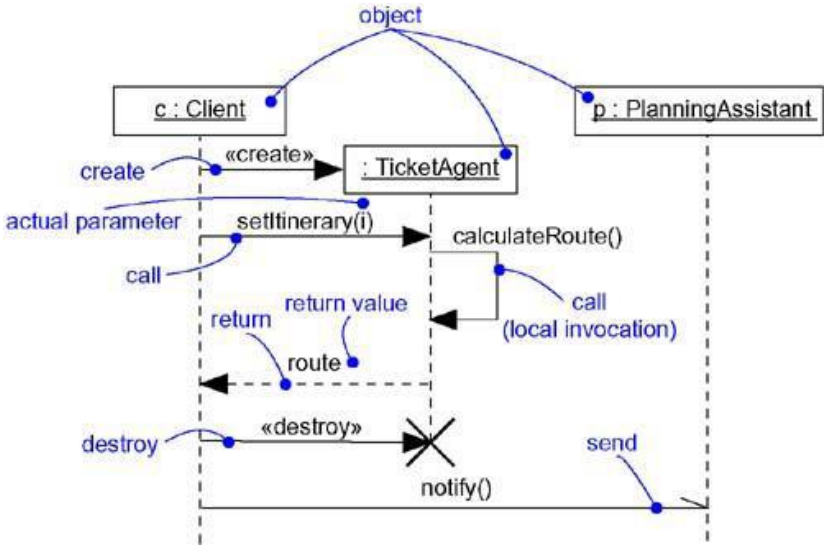

Figure 9. Messages

2.2 Sequencing

A *sequence* is formed of a stream of messages in which an object passes a message to another object, the receiving object might in turn send a message to another object which might send a meesage to yet a different object and so on.

A communication diagram shows the message flow between roles within a collaboration. Messages flow along connections of the collaboration (see Figure 10).

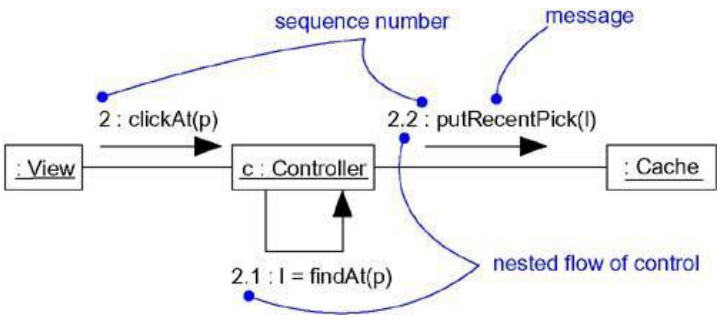

Figure 10. Sequence

2.3 Use Cases

A *use case* is a description of a set of sequences of actions, including variants that a system performs to yield an observable result of value to an actor. Graphically, a use case is represented as an ellipse.

An *actor* represents a set of roles that users of use cases play when interacting with these use cases. Typically, an actor represents a role that a human or hardware device or even another system plays with a system.

2.4 Use Case Diagrams

A *use case diagram* shows a set of use cases and actors together with their relationships. Use case diagrams commonly contain the following elements: subject, use cases, actors, and dependency, generalization and association relationships (see Figure 11).

The subject is shown as a rectangle containing a set of use case ellipses. The name of the subject is placed within the rectangle.

The actors are shown as stick figures placed outside the rectangle having their names placed under them.

Lines connect actor icons to the use case ellipses with which they communicate.

Relationships among use cases (e.g. extend, include) are drawn inside the rectangle.

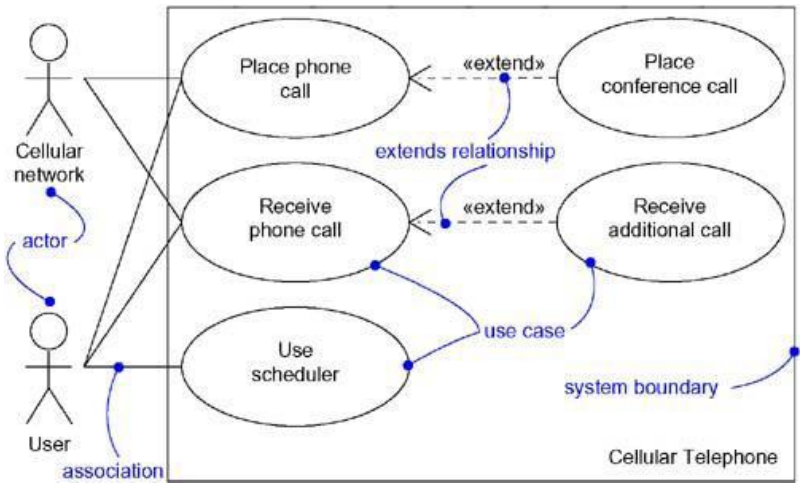

Figure 11. A Use Case diagram

2.5 Interaction Diagrams

An interaction diagram shows an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them.

A *sequence diagram* is an interaction diagram that emphasizes the time ordering of messages (see Figure 12).

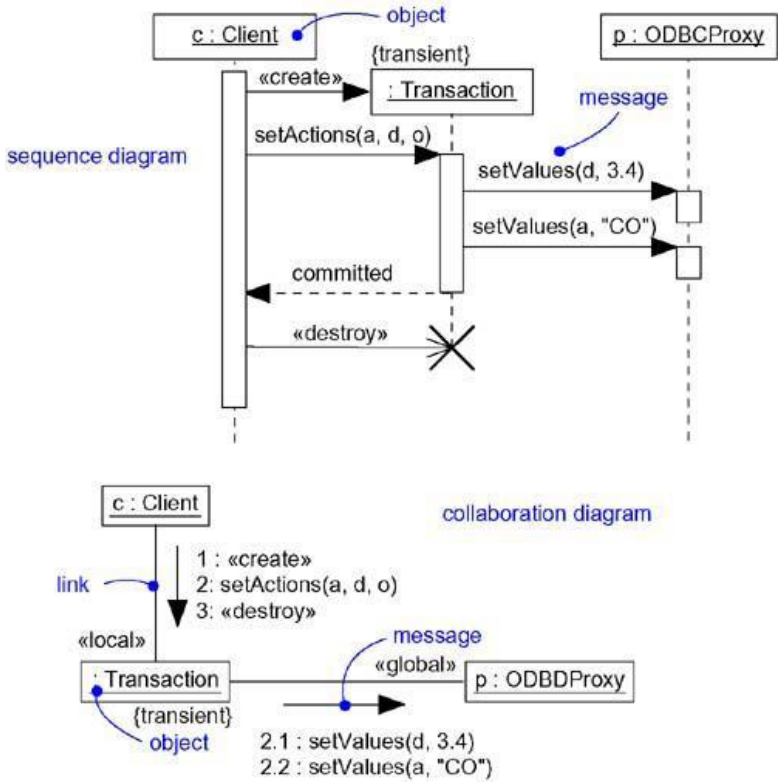

Figure 12. Interaction diagrams

A *communication diagram* is an interaction diagram that emphasizes the structural organization of objects that send and receive messages (see Figure 12). A communication diagram is formed by first placing the objects that participate in the interaction as the vertices in a graph and then the links that connect these objects are rendered as the arcs of this graph. Finally, the links are adorned with the messages that objects send and receive.

Differences between sequence diagrams and communication diagrams:

- Sequence diagrams define the concept of *object lifeline* – the vertical dashed line that represents the existence of an object over a period of time
- Sequence diagrams define the concept of *focus of control* – tall, thin rectangle that shows the period of time during which an object is performing an action, either directly or through a subordinate procedure.

Sequence diagrams allow the definition of structured control sequences: optional execution, conditional execution, parallel execution, loop (iterative) execution.

3. Forward Engineering Basics

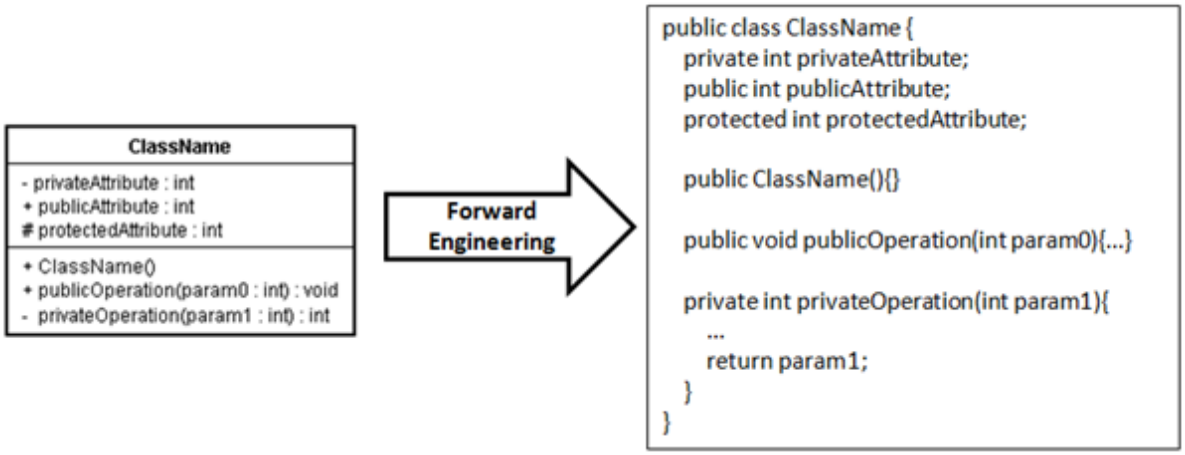

Figure 13. Forward engineering for classes

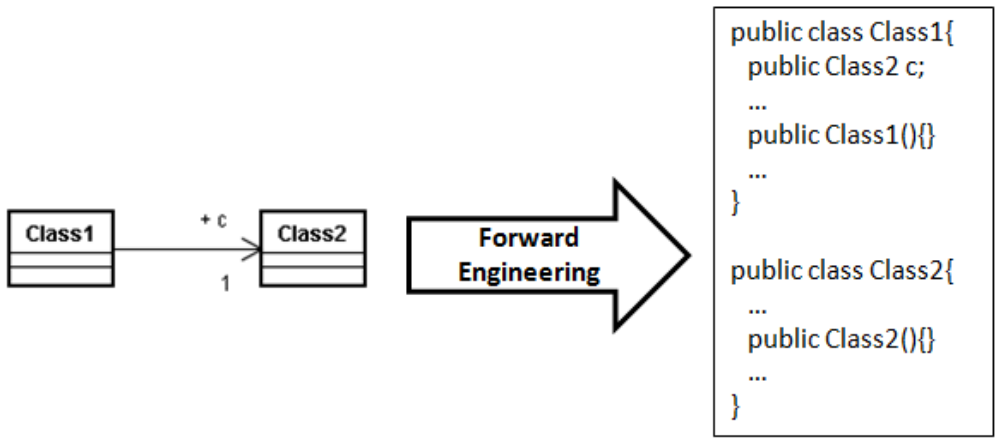

Figure 14. Forward engineering for association relationship (1)

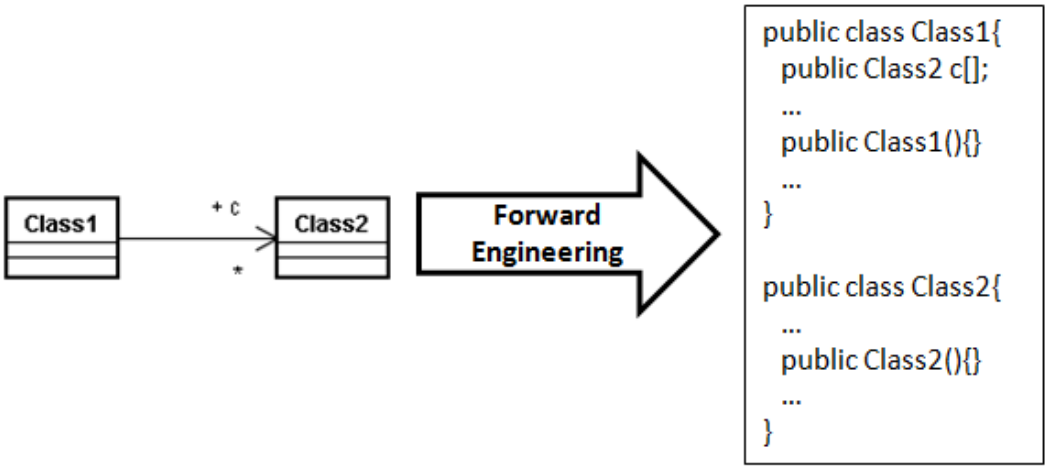

Figure 15. Forward engineering for association relationship (2)

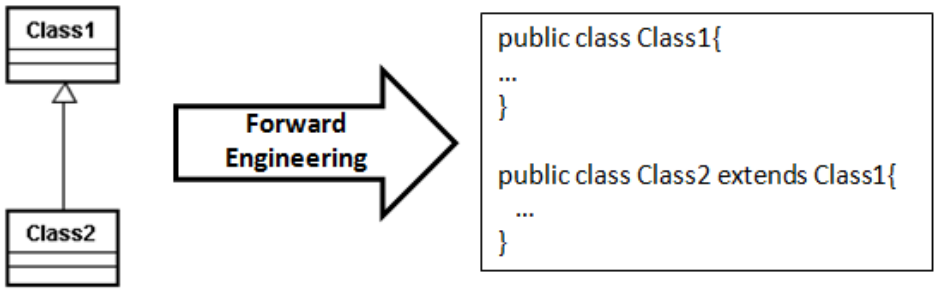

Figure 16. Forward engineering for the generalization relationship

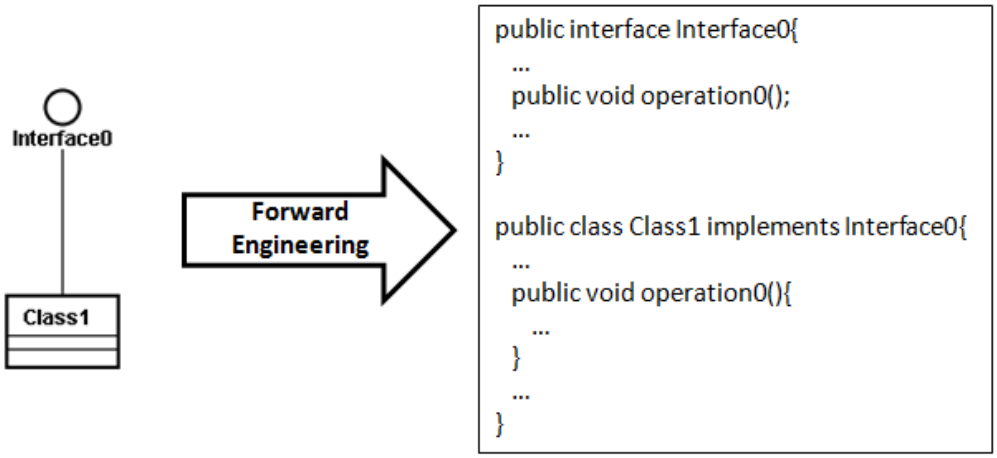

Figure 17. Forward engineering for the realization relationship

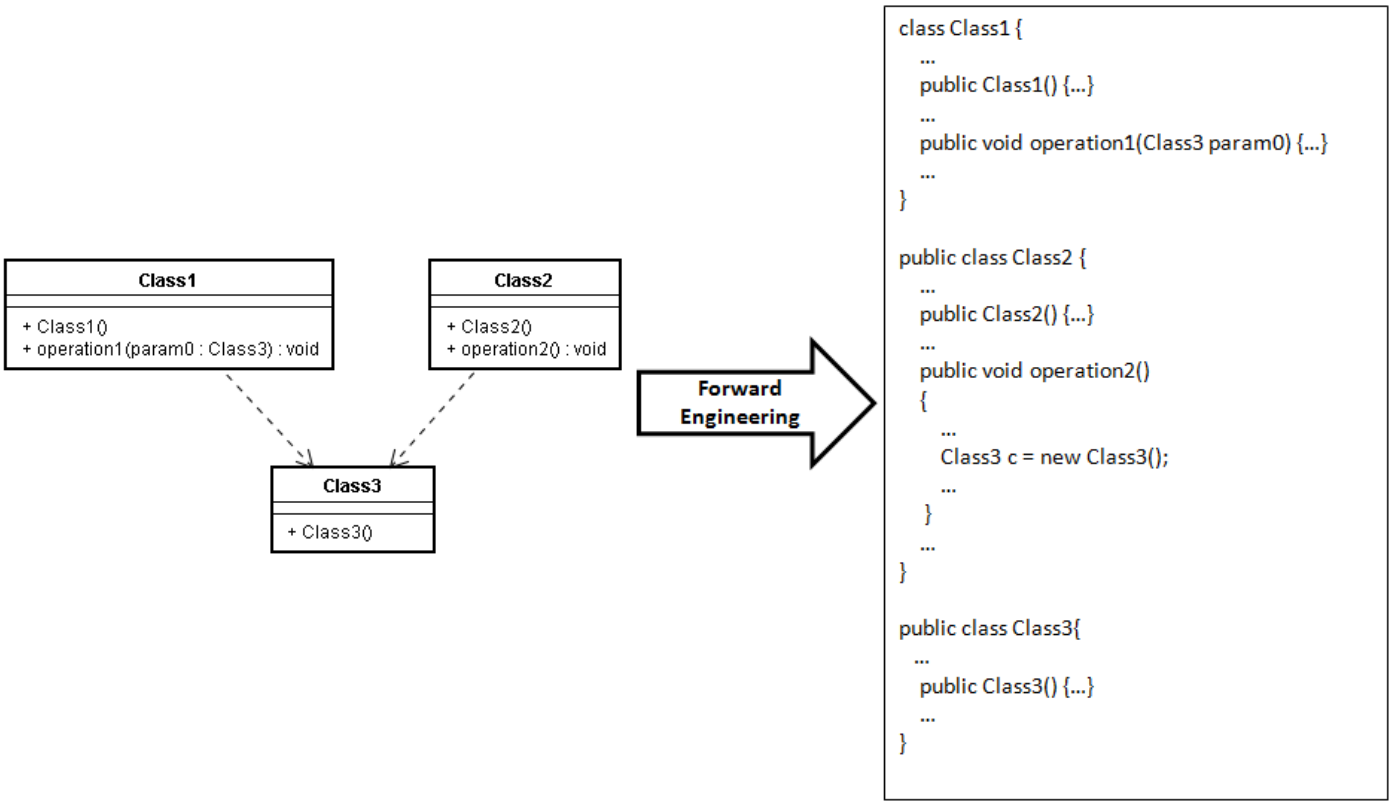

Figure 18. Forward engineering for the dependency relationship

II. Design Patterns Revision

Design patterns represent solutions to problems that arise when developing software within a particular context. They capture static and dynamic structure and collaboration among key participants in software designs. Consequently, design patterns facilitate reuse of successful software architectures and design.

Design patterns are classified as follows:

- **Creational Patterns**
  - Deal with initializing and configuring classes and objects
  - „*How am I going to create my objects?* ”
  - Examples: Singleton, Factory Method, Prototype
- **Structural Patterns**
  - Deal with decoupling the interface and implementation of classes and objects
  - „*How classes and objects are composed to build larger structures*? ”
  - Examples: Composite, Decorator, Proxy
- **Behavioral Patterns**
  - Deal with dynamic interactions among societies of classes and objects
  - „H*ow to manage complex control flows (communications)*? ”
  - Examples: Strategy, Command, Observer

III. Problems

P1. Prepare a class model to describe undirected graphs. An undirected graph consists of a set of vertices and a set of edges. Edges connect pairs of vertices. Your model should capture only the structure of graphs (i.e. connectivity).

P2. Consider the following specification: "*A company consists of several departments. Each department is located in one or more buildings*. ". Draw a class diagram to model the concepts above (no attributes and operations, just classes and the relationships between them including multiplicities).

P3. Prepare a class diagram for a graphical document editor that supports grouping. Assume that a document consists of several sheets. Each sheet contains drawing objects, including text, geometrical objects, and groups. A group is simply a set of drawing objects, possibly including other groups. A group must contain at least two drawing objects. A drawing object can be a direct member of at most one group. Geometrical objects include circles, ellipses, rectangles, lines and squares.

P4. A* is an advertising company in New York. A* deals with other companies that it calls clients. A record is kept of each client company. Clients have advertising campaigns, and a record is kept of every campaign. Each campaign includes one or more adverts. A* nominates members of creative team, which work on campaigns. One member of the creative team manages each campaign. Staff may be working on more than one project at a time. When a campaign starts, the manager responsible estimates the likely cost of the client and agrees it with the client. A finish date may be set for a campaign at any time, and may be changed. When the campaign is completed, an actual completion date and the actual cost are recorded. When the client pays, the date is recorded. The manager checks the campaign budget periodically. The system should also hold the staff grades, keep a record of the contact information for each staff member, and should calculate staff salaries.

Requirements

a) Draw a UML use case diagram for the A* information system.
b) Draw a UML class diagram. The class diagram should represent all of the classes, their attributes and operations, relationships between classes, multiplicity specifications.
c) Draw a UML sequence diagram for the use case *Add Advertisement*.

P5. Design a photo/video sharing Web application. The application should allow a potential new user to create an account, activate the account via an email link or view photos and videos without being logged in. Login would allow him to edit his profile, post videos and post photos either stand-alone or in an album he creates. Also, upon viewing a photo or a video, anyone, logged in or not, can flag the content for questionable content. A content advisor should review each flagged item once logged in, then decide on whether the content is acceptable for the scope of the Web application. If the content is inappropriate, the content advisor can block the owning user's account.

Requirements

a) Draw a UML use case diagram for the photo/video sharing Web application.
b) Draw a UML class diagram. The class diagram should represent all of the classes, their attributes and operations, relationships between classes, multiplicity specifications.
c) Draw a UML sequence diagram for the use case *Block User*.

P6. Draw an UML diagram for the following C++ segment of code:

```
class Person
{
public:
        Person(const char* aName);
        void speak();
        void drive();
private:
        char* name; // Persons Name
        Brain brain; // persons brain
        Car* pCar; // Car owned by the person
        double height;
        double weight;
};
```

P7. Given the following code:

```
class Memory {…} // Assume a copy constructor is provided for this class.

class Memory1 extends Memory {…} // Assume a copy constructor is provided.

class Computer
{
  private Memory theMemory;
  public Computer(Computer another)
 {
     if (another.theMemory instanceof Memory1)
          theMemory = new Memory1(another.theMemory);
     else
          theMemory = new Memory(another.theMemory);
 }
}
```

Draw a UML class diagram showing the relationship between these classes.

P8. Choose a design pattern studied in previous courses. Think of a problem in which it could be applied, draw the associated class diagram and write the associated source code.

IV. Bibliography

[1] G. Booch, J. Rumbaugh and I. Jacobson – T*he Unified Modeling Laguage User Guide Second Edition* – Addison-Wesley, ISBN 0-321-26797, 2005.

[2] Unified Modeling Language (UML) Tutorial – http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/index.htm

[3] http://edn.embarcadero.com/article/31863