UNIVERSITY OF ENGINEERING AND TECHNOLOGY, TAXILA

FACULTY OF TELECOMMUNICATION AND INFORMATION ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

Understanding Object Orientation

LAB MANUAL 2

Lab Objective: Learn to Know

- . How to understand the object-oriented mindset
- . How objects communicate
- . How objects associate with one another
- . How objects combine

Tool used: MS Visio 2003.

Lab Description:

An object is an instance of a class (a category). An object has **structure**. That is, it has attributes (properties) and behavior. An object's behavior consists of the **operations** it carries out. Attributes and operations taken together are called **features**. A class is a template for making objects.
Object-orientation has taken the software world by storm, and rightfully so. As a way of creating programs, it has a number of advantages. It fosters a component-based approach to software development so that you first create a system by creating a set of classes. Then you can expand the system by adding capabilities to components you've already built or by adding new components. Finally, you can reuse the classes you created when you build a new system, cutting down substantially on system development time.

Some Object-Oriented Concepts:

Object-orientation considers aspects of objects. These aspects are called

- Abstraction
- Inheritance
- Polymorphism
- Encapsulation.

Three other important parts of object-orientation are

- Message sending,
- Associations
- Aggregation.

Abstraction:

Abstraction means, simply, to filter out an object's properties and operations until just the ones you need are left. An abstraction relationship is a dependency between model elements that represents the same concept at different levels of abstraction or from

different viewpoints. You can add abstraction relationships to a model in several diagrams, including use-case, class, and component diagrams.

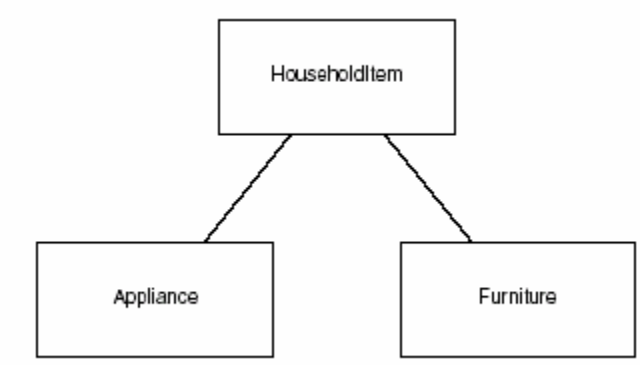
In an abstraction relationship, one model element, the client, is more refined or detailed than the other, the supplier. The different types of abstraction relationships include derivation, realization, refinement, and trace relationships.

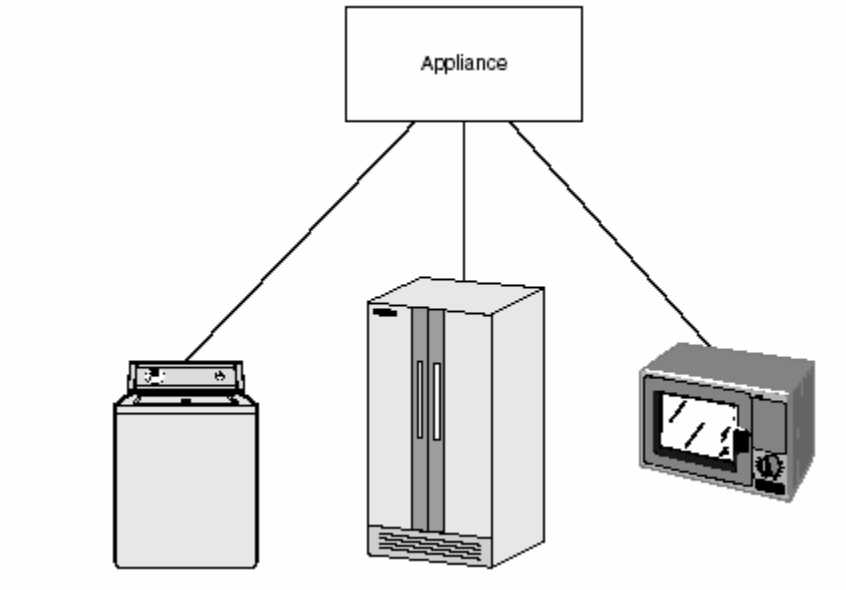
All abstraction relationships can connect model elements that are in the same model or in different models. For example, if you develop an analysis model and then a design model, you can connect the models with a refinement relationship pointing from the analysis model to the design model. This relationship indicates that the design model provides a different level of abstraction of the same system.

Different types of problems require different amounts of information, even if those problems are in the same general area.
In any case, what you're left with, after you've made your decisions about what to include and what to exclude, is an abstraction.

Inheritance:

Similarities often exist between different classes. Very often two or more classes will share the same attributes and/or the same methods. Because you don't want to have to write the same code repeatedly, you want a mechanism that takes advantage of these similarities. Inheritance is that mechanism. Inheritance models "is a" and "is like" relationships, enabling you to reuse existing data and code easily. When *A* inherits from *B,* we say *A* is the subclass of *B* and *B* is the superclass of *A*. Furthermore, we say we have "pure inheritance" when *A* inherits all the attributes and methods of *B*. The UML modeling notation for inheritance is a line with a closed arrowhead pointing from the subclass to the superclass.i.e Washing machines, refrigerators, microwave ovens, toasters, dishwashers, radios, waffle makers, blenders, and irons are all appliances. In the world of object orientation, we would say that each one is a **subclass** of the Appliance class. Another way to say this is that Appliance is a **superclass** of all those others.





Polymorphism:

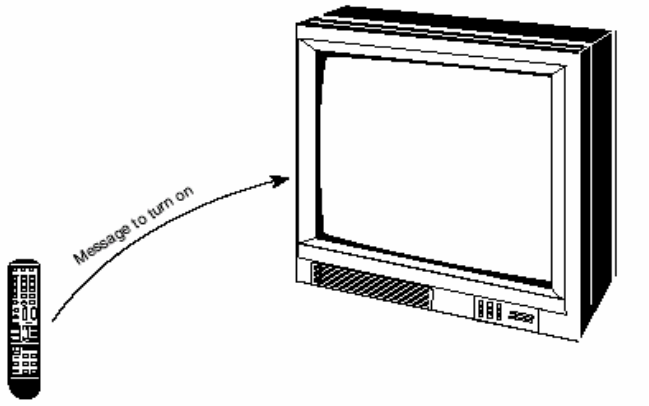
Polymorph means many shapes. Polymorphism is a way for people to conceptualize things, which are not entirely the same thing, using the same name In the UML it is assumed that if an operation is applied to an object and there are several alternative classes that have the operation defined then the object to which the operation is applied always determines the operation that is executed. Sometimes an operation has the same name in different classes. For example, you can open a door, you can open a window, and you can open a newspaper, a present, a bank account, or a conversation. In each case you're performing a different operation. In object-orientation each class "knows" how that operation is supposed to take place. This is called **polymorphism.**

Encapsulation:

Encapsulation means that an object hides what it does from other objects and from the outside world, encapsulation is also called **information hiding**. When an object carries out its operations, those operations are hidden that's the essence of **encapsulation**. In a system that consists of objects, the objects depend on each other in various ways. If one of them happens to malfunction and software engineers have to change it in some way, hiding its operations from other objects means that it probably won't be necessary to change those other objects.i.e When most people watch a television show, they usually don't know or care about the complex electronics components that sit in back of the TV screen and all the many operations that have to occur in order to paint the image on the screen. The TV does what it does and hides the process from us. Most other appliances work that way, too.

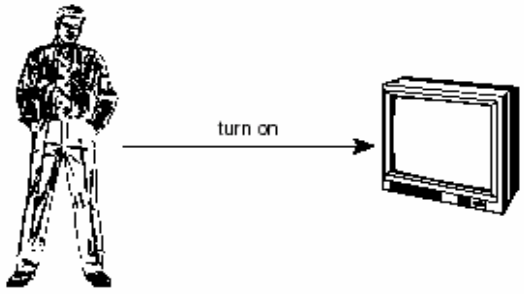
Message Sending:

In a system, objects work together. They do this by sending messages to one another. One object sends another a message—a request to perform an operation and the receiving object performs that operations.i.e A TV and a remote present a nice intuitive example. When you want to watch a TV show, you hunt around for the remote, settle into your favorite chair, and push the On button. What happens? The remote-object sends a message (literally!) to the TV-object to turn itself on. The TV-object receives this message, knows how to perform the turn-on operation, and turns itself on. When you want to watch a different channel, you click the appropriate button on the remote, and the remote-object sends a different message—"change the channel"—to the TV object. The remote can also communicate with the TV via other messages for changing the volume, muting the volume, and setting up closed captioning.

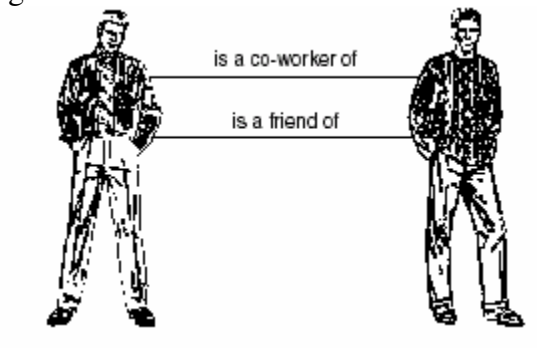


Associations:

Another common occurrence is that objects are typically related to one another in some fashion. For example, when you turn on your TV, in object-oriented terms, you're in an **association** with your TV. Association may be unidirectional and bidirectional.
The "turn-on" association is unidirectional (one-way),. That is, you turn your TV on. Unless you watch way too much television, however, it doesn't return the favor.

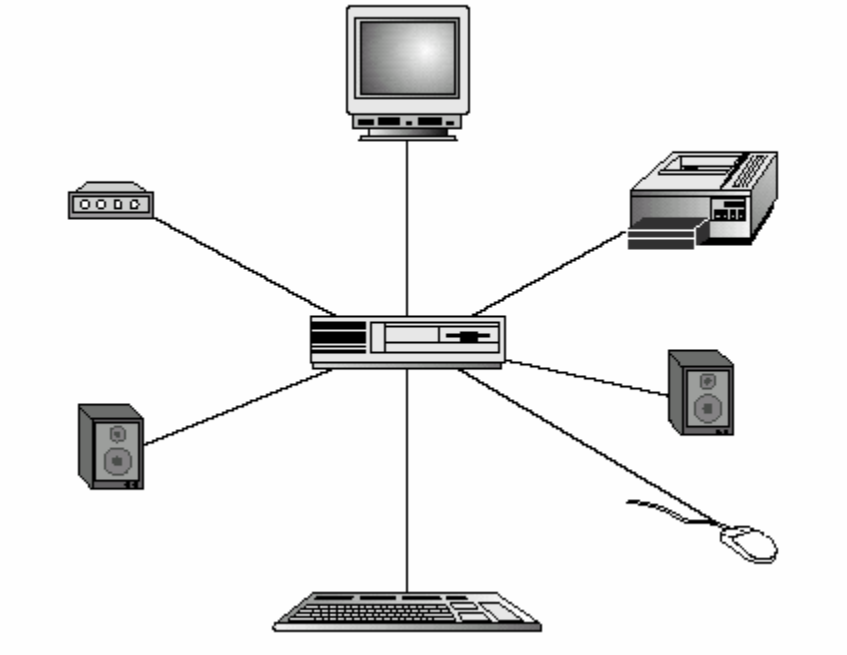


Sometimes an object might be associated with another in more than one way.



Aggregation:

A special form of an association that specifies a whole-part relationship between the aggregate (whole) and a component part. One form of aggregation involves a strong relationship between an aggregate object and its component objects. This is called **composition**. The key to composition is that the component exists as a component only within the composite object.



Lab performed on (date): _____ Reg #: _____
Checked by: _____ Date: _____
Marks Awarded: _____