Chapter – 4

Object-Oriented Design (OOD) Methodologies

Objectives:

At the end of this Chapter, the students will be able to:

- Define Objects and classes for modeling.
- Identify Quality Classes and Objects and also semantic of classes and their relationship.
- Understands object Modeling and related basic concepts needed for Modeling.
- Differentiate Object modeling from other types of modeling.
- Define states, events, operations and state diagrams etc.
- Construct and describe Dynamic Modeling and Functional Modeling.
- Understands various object-oriented metrics in the design.
- Identify the quality of an object oriented design.
- Discuss different types of Design patterns in the development of software.
- Understand Creational, Structural and Behavioral Design Patterns.

4.0 Introduction

In this method we take a slightly different view of systems development from what we have examined before. In the previous section we tended to view our system from a functional or process-based viewpoint, and related this to the data structure. While this approach does produce well-designed, working systems, the current opinion among many practitioners is that the resulting systems tend to be rigid and make it difficult to respond quickly to changes in user requirements.

Unlike its two predecessors, the object-oriented approach combines data and processes (called methods) into single entities called objects. Objects usually correspond to the real things a system deals with, such as customers, suppliers, contracts, and invoices. Object-oriented models are able to thoroughly represent complex relationships and to represent data and data processing with a reliable notation, which allows an easier mix of analysis and design in a growth process. The aim of the Object-Oriented approach is to make system elements more modular, thus improving system quality and the efficiency of systems analysis and design.

In the Object-Oriented approach we tend to focus more on the behavior of the system. The main feature we document is the Object or Class.

4.1 Building Quality Classes and Objects

A system should be built with a minimum set of unchangeable parts; those parts should be as general as possible; and all parts of the system should be held in a uniform framework. With object-oriented development, these parts are the classes and objects that make up the key abstractions of the system, and the framework is provided by its mechanisms. [2]

In our experience, the design of classes and objects is an incremental, iterative process, except for the most trivial abstractions; we have never been able to define a class exactly right the first time. It takes time to smooth the conceptual jagged edges of our initial abstractions. Of course, there is a cost to refining these abstractions, in terms of recompilation, understandability, and the integrity of the fabric of our system design. [2]

For identifying the semantics of classes and objects

- Select scenarios and analyze
- Assign responsibility to achieve desired behavior
- Partition responsibilities to balance behavior
- Select an object and enumerate its roles and responsibilities
- Define operations to satisfy the responsibilities
- Look for collaborations among objects

4.2 Object Modeling

Links and associations, generalizations, inheritance grouping aggregations, abstract classes, multiple inheritances, metadata, Candidate keys, Dynamic Modeling event states, Operations, Nested State Diagram Concurrency, relation of object & Dynamic Models.

4.2.1 Over-Riding Features

A subclass may override a super class feature by defining a feature with the same name. The overriding feature refines and replaces the overridden features per class feature.

Operation rotate is overridden for performance in class circle to be a null operation.

Tightening the type of an attribute or operation argument to be a subclass of the original type is a form of restriction and must be done with a case.

- A feature should never be overridden so that it is inconsistent with the signature or semantics of the original inherited feature.
- A subclass is a special case of its super class and should be compatible with it in every respect
- A common, practice in object-oriented programming is to "borrow" a class that is similar to a desired class and then modify it by changing and ignoring some of its features even though the new class is not really a special case of the original class.

4.3 Recursive Aggregates

Aggregation can be fixed, variable or recursive. A fixed aggregate has a fixed structure. The number and type of subparts are predefined.

A variable aggregate contains, directly or indirectly, an instance of the same kind of aggregate; the number of potential levels is unlimited. A computer program is an aggregation of blocks, with optionally recursive compound statement; the recursive terminates with simple statements. Blocks can be nested to arbitrary depth.

The following figure shows the usual form of a recursive aggregate: a super-class and two subclasses, one of which is an intermediate node of the aggregate and one of which is a terminal node of the aggregate. The intermediate node is an assembly of instances of the abstract super-class.

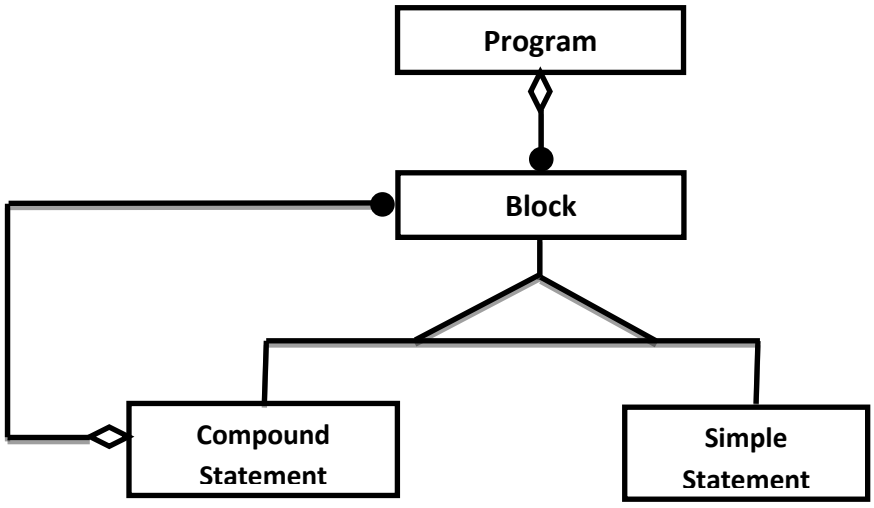

Fig 4.1: Recursive

4.3.1 Propagation of operations

Propagation (also called triggering) is the automatic application of an operation to a network of objects when the operation is applied to some starting object.

For e.g.

Moving an aggregate moves its parts; the move operation propagates to the parts. Propagation of operations to parts is often a good indicator of aggregation. Example of propagation is shown in the figure

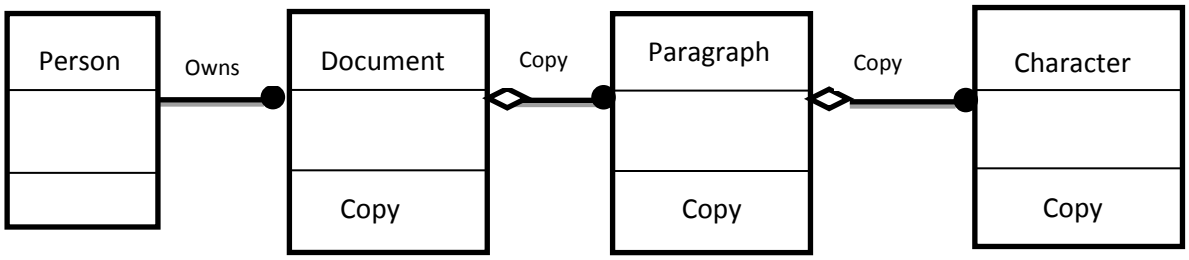

Fig-2: Propagation of operations

A person owns multiple documents. Each document is composed of paragraphs that are, in turn, composed of characters. The copy operation propagates from documents to paragraphs to characters. Copying a paragraph copies all the characters in it. The operation does not propagate in the reverse direction; a paragraph can be copied without copying the whole document. Similarly, copying a document copies the owner link but does not spawn a copy of the person who is owner.

An operation can be thought of as starting at some initial object and flowing from object to object through links according to propagation rules. Propagation is possible for other operations including save restore, destroy, print, lock and display.

Propagation is indicated on object models with a special notation. The propagation behavior is bound to an association, direction & operation. Propagation is indicated with a small arrow and operation name next to the affected association. The arrow indicates the direction of propagation.

4.4 Abstract Classes

An abstract class is a class that has no direct instances but whose descendent classes have direct instances. A concrete class is a class that is instant able. i.e., it can have direct instances. A concrete class may have abstract subclasses. A concrete class may be leaf class in the inheritance tree; only concrete classes may be leaf classes in the inheritance tree.

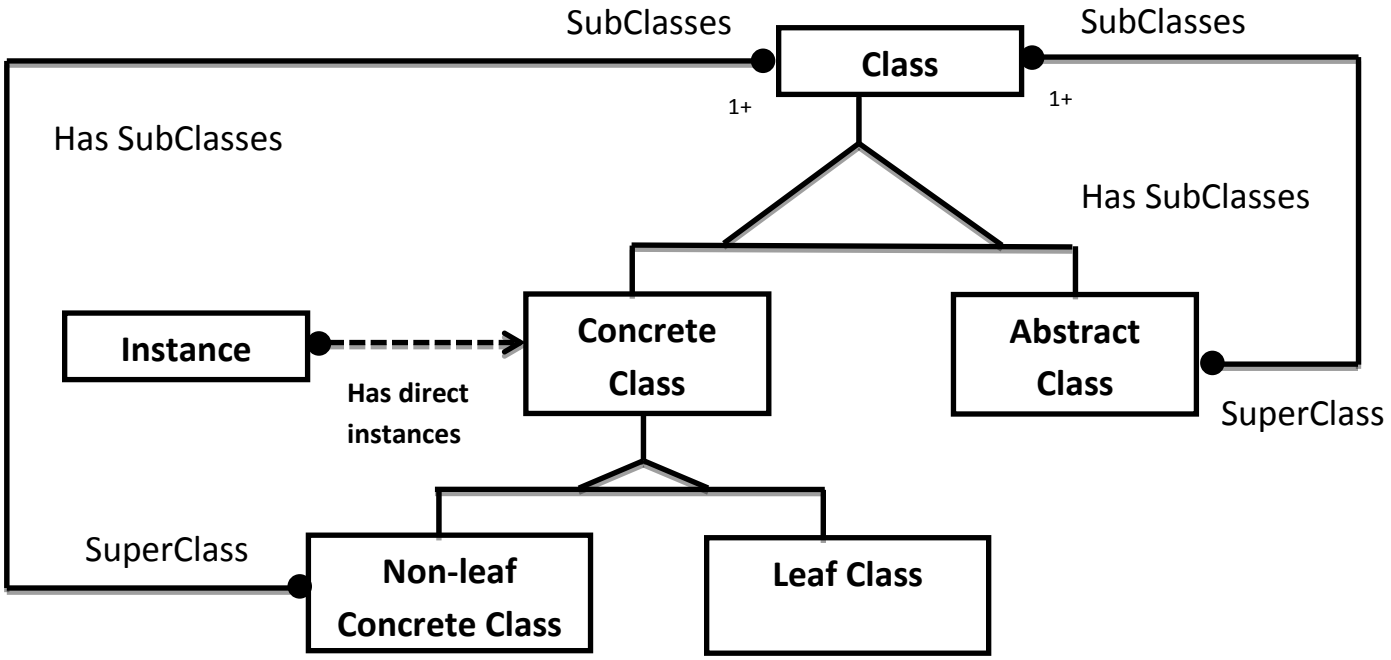

Fig-3: Object model defining abstract and concrete class

All the occupations shown in the following figure are concrete classes. Butcher, baker and candlestick maker are concrete classes because they have direct instances. The ellipse notation (…) indicates that additional subclasses exist but have been omitted from the diagram, perhaps for lack of space or lack of relevance to the present concern. Worker also is a concrete class because some occupations may not be further specified.

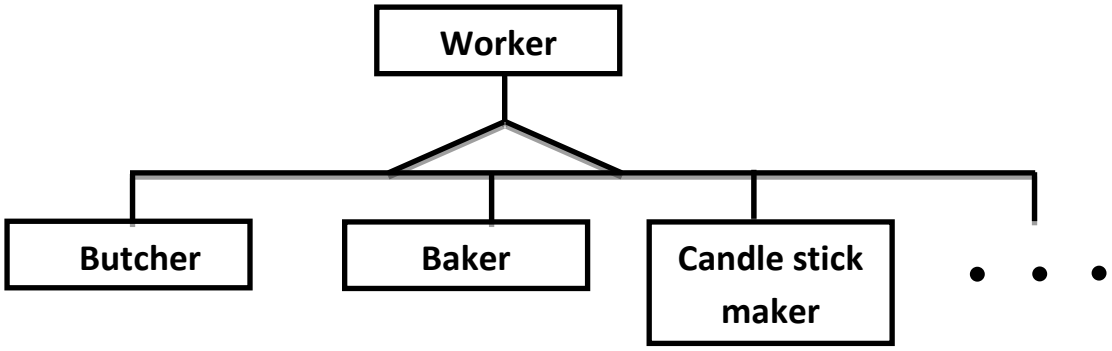

Fig-4: Concrete

Class employee in the following figure is an example of an abstract class. All employees must be hourly, salaried or exempt.

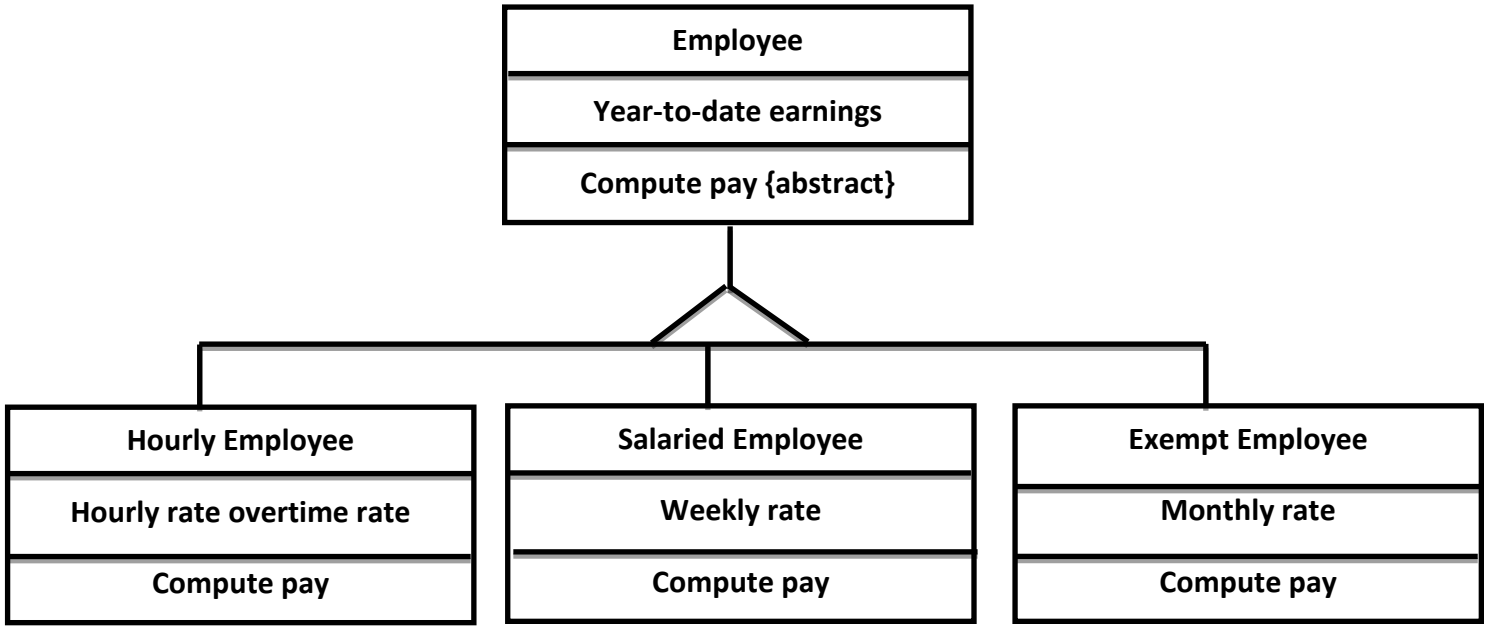

Fig-5: Abstract class and abstract operation

Abstract classes organize features common to several classes. It is often useful to create abstract super-class to encapsulate classes that participate in the same association or aggregation. Some abstract classes appear naturally in the application domain. Other abstract classes are artificially introduced as a Mechanism for promoting code reuse.

Abstract classes are frequently used to define methods to be inherited by subclasses on the other hand, an abstract class can definite the protocol for an operation without supplying a corresponding method. We call this an abstract operation.

The above figure shows an abstract operation. An abstract operation is designed by a comment in braces. Compute- pay is an abstract operation of class employee. The original class of a feature is the top most defining class. The original class defines the protocol of the feature.

4.4.1 Multiple Inheritances

Multiple inheritance permits a class to have more than one super-class and to inherit features from all parent this permits mixing of information from two or more sources. This is a more complicated form of generalization than single inheritance, which restricts the class hierarchy to a tree.

Definition:

A class may inherit features from more than one super-class is called a join class.

Each generalization should cover a single property for example where a vehicle travels. If a can be refined on several distinct and independent dimensions, then use multiple generalizations.

The term multiple inheritances is used somewhat imprecisely to mean either the conceptual relationship between classes or the language mechanism that implements that relationship by sharing of behavior and data.

4.4.2 Accidental Multiple Inheritance

An instance of a join class is inherently an instance of all the ancestors of the join class

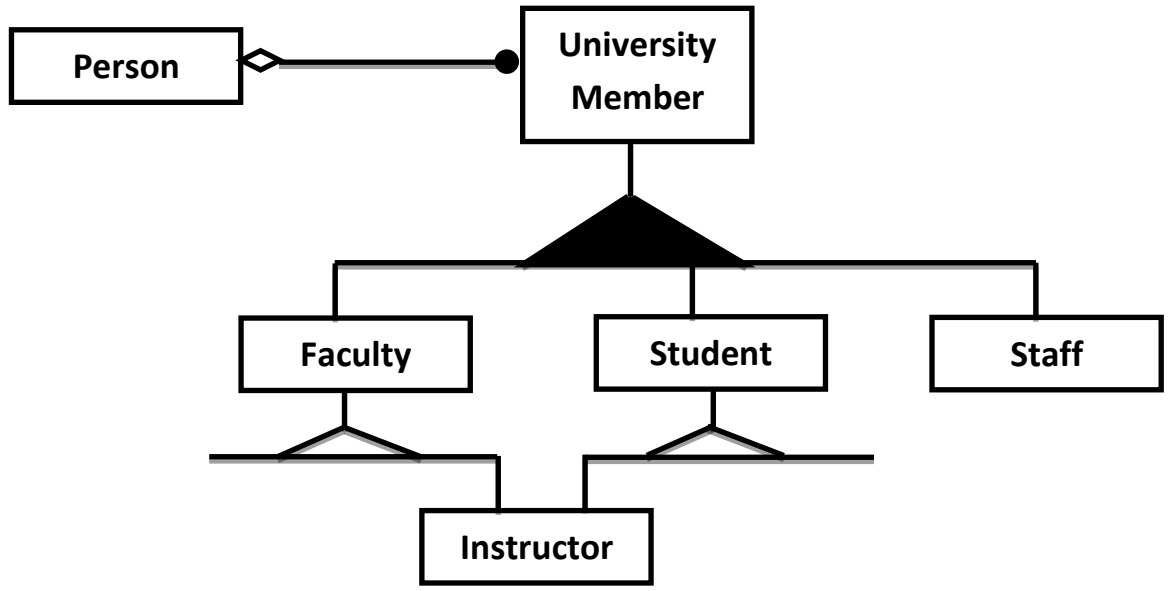

Fig-6: Workaround for accidental multiple inheritance

4.4.3 Meta Data

Meta data is data that describes other data. Computer language implementations also use metadata heavily.

Relational database management systems also use metadata. A person can define database tables for storing information. Similarly, a relation DBMS has several meta tables that store table definition.

Metadata is frequently confusing because it bliss the normal separation between the model and the real world.

4.4.4 Patterns and Metadata

A class describes a set of object instances of a given form. Instantiation relates a class to its instances.

Real world things may be metadata. There are real-world things that describe other world things. A part describes in a catalog describes manufactured parts.

4.4.5 Class Description

Classes can be considered as objects, but they are meta-objects and not real world objects class description objects have features, and they in turn have their own classes, which are called meta-classes.

- A class attribute describes a value common to an entire class of objects, rather than data peculiar to each instance.
- Class attributes are useful for storing default information for creating new objects or summary information about instances of the class.
- Class operation is an operation on the class itself. The most common kind of class operations are operations to create new class instances.

A class window with class features indicated by leading dollar signs windows has class attributes for the set of window size, and maximum window.

Window contains class operations to creates a new window object and to find the existing window with the highest priority.

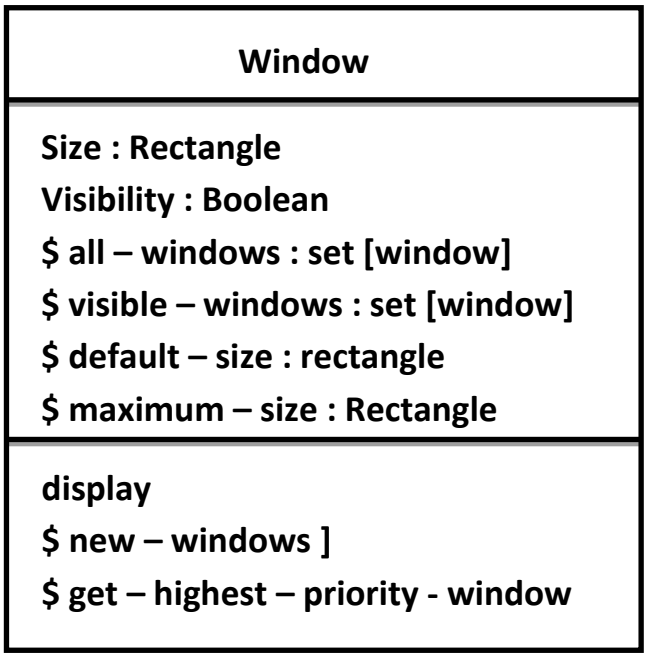

Fig-7: Class with class features

4.4.6 Candidate keys

A candidate key is a minimal set of attributes that uniquely identifies and object or link. By minimal, an attribute cannot be discarded from the candidate key and still distinguish all objects and links. A class or association may have one or more candidate keys, each of which may have different combinations and number of attributes.

Candidate key is a term commonly used within the database community. However candidate key is really not a database concept, a candidate key is a logical concept. Each candidate key constraints the instances in a class or the multiplicity of an association Most programming languages lack the notion of a candidate key. A candidate key is delimited in an object model with braces.

Multiplicity and candidate key have nearly the same expressive power for binary associations. Multiplicity also includes the notion of existence dependency. A many –to- many association requires both related objects to uniquely identify each link.

A one-to-many association has a single candidate key: the object on the many end. A one-to –many association has two candidate keys either of the objects.

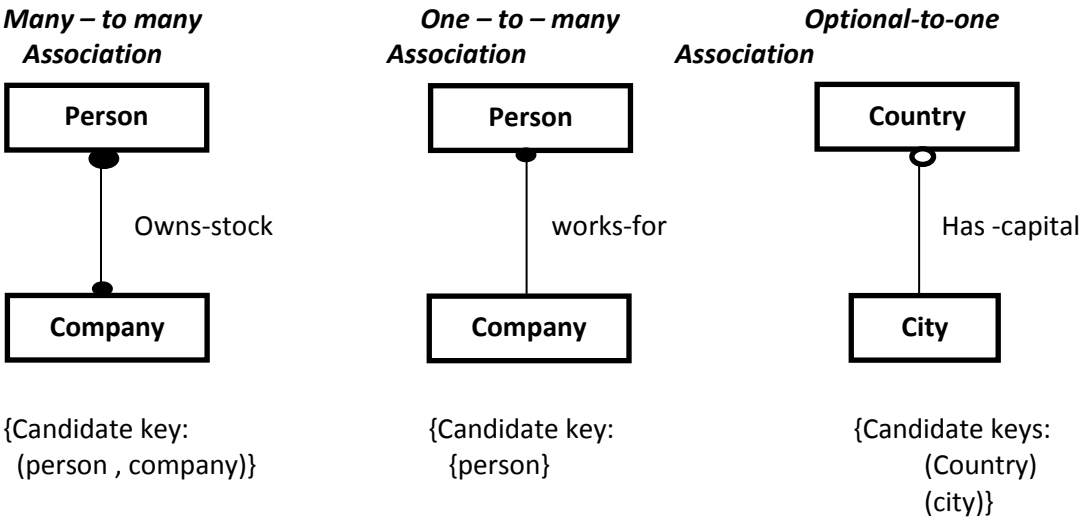

Fig-8: Comparison of multiplicity with candidate keys for binary associations

4.4.7 Constraints

Constraints are functional dependencies between objects that are not related by an input-output dependency. Constraints can be on two objects at the same time or between instances of the same object at different times or between different objects at different times.

Preconditions on functions are constraints that the input values must satisfy and post constraints are that the output values are guaranteed to hold

State the times or conditions under which the constraints hold.

4.5 Dynamic Modeling

4.5.1 Events and States [2]

An event is something that happens at a point of time, such as user depresses left button. An event has no duration of course, nothing is really instantaneous, and event is simply an occurrence that is fast compared to the granularity of the time scale of a given abstraction.

An event is a one –way transmission of information from one object to another. It is not like subroutine call that returns a value.

In the real world, all objects exist concurrently. An object sending an event to another object may expect a reply, but the reply is a separate event under the control of the second object, which may or may not choose to send it.

- Every event is a unique occurrence.
- This structure is hierarchical just as object class structure is hierarchical.

- An event conveys information from one object to another.
- The term event is often used ambiguously. Sometimes event refers to an event instance at other times to an event class. In practice, this ambiguity is usually not a problem and the precise meaning is apparent from the context

Scenarios and Event Traces

A scenario is a sequence of events that occurs during one particular execution of a system. The scope of a scenario can vary, it may include all events in the system, or it may include only those events impinging on or generated by certain objects in the system. A scenario can be the historical record of executing a system or a thought experiment of executing a proposed system.

States

A state is an abstraction of the attribute values and links of an object sets of values are grouped together into a state according to properties that affect the gross behavior of the object.

A state specifies the response of the object to input event. The response of the object to an event received by an object may vary quantitatively depending on the exact values of its attributes but the response is qualitatively the same for all values within the same state and may be qualitatively different for value in different state.

- A state corresponds to the internal between two events received by an object. Events represent points in fine; states represent intervals of time.
- The state of an object depends on the past sequence of events it has received, but in most cases past events are events are eventually hidden by subsequent events.
- A state has duration it occupies an interval time
- A state cum be characterized in various ways.

4.5.2 State Diagrams

A state diagram relates events and states when an event is received, the next state depends on the current state as well as the event, a change of state caused by an event is called a transition.

A state diagram specifies the state sequence caused by an event sequence. If an object is in a state and an event labeling one of its transitions occurs, the object enters the state on the target end of the transition. The transition is said to fire.

A sequence of events corresponds to a path through the graph.

Conditions:

A condition is a Boolean function of object values, such as “the temperature is below freezing” A condition is valid over an interval of time.

Conditions can be used as guards on transitions. A guarded transition fires when its event occurs but only if the guard condition is true. A guard condition on a transition is shown as a Boolean expression in brackets following the event name.

4.5.3 Operations

State diagrams would be of little use if they just described patterns of events. A behavioral description of an object must specify what the object does in response to events. Operations attached to states or transitions performed in response to the corresponding states or events.

- An activity is an operation that takes time to complete. An activity is associated with a state.
- A state may control a continuous activity, such as ringing a telephone bell, that persists until an event terminates it by causing a transition from the state.
- The same notation “do; A” indicates that sequential activity A begins on entry to the state and stops when complete.
- An action is an instantaneous operation. An action is associated with an event. An action represents an operation whose directly is insignificant compared to the resolution of the state diagram.
- Actions can also represent internal control operations, such as setting attributes or generating other events. Such actions have the real-world count experts but instead are mechanism.

4.5.4 Event Generalization

Events can be organized into a generalization hierarchy with inheritance of event attributes.

- Keyboard characters can be divided into central characters and graphic characters
- Providing an event hierarchy permits different level of abstraction to be used at different places in a model.