UNIT 2

CLASSES AND OBJECTS

When we use object-oriented methods to analyze or design a complex software system, our basic building blocks are classes and objects.

An object is an abstraction of something in a problem domain, reflecting the capabilities of a system to keep information about it, interact with it, or both

- Objects have an internal state that is recorded in a set of attributes.
- Objects have a behavior that is expressed in terms of operations. The execution of operations changes the state of the object and/or stimulates the execution of operations in other objects.
- Objects (at least in the analysis phase) have an origin in a real world entity.

Classes represent groups of objects which have the same behavior and information structures.

- Every object is an instance of a single class
- Class is a kind of type, an ADT (but with data), or an 'entity' (but with methods)
- Classes are the same in both analysis and design
- A class defines the possible behaviors and the information structure of all its object instances.

THE NATURE OF AN OBJECT

The ability to recognize physical objects is a skill that humans learn at a very early age. From the perspective of human, cognition, an object is any of the following.

- A tangible and/or visible thing.
- Something that may be apprehended intellectually.
- Something toward which thought or action is directed.

Informally, object is defined as a tangible entity that exhibits some well defined behavior. During software development, some objects such as inventions of design process whose collaborations with other such objects serve as the mechanisms that provide some higher level behavior more precisely.

An object represents an individual, identifiable item, until or entity either real or abstract, with a well defined role in the problem domain. E.g. of manufacturing plant for making airplane wings, bicycle frames etc. A chemical process in a manufacturing plant may be treated as an object;

because it has a crisp conceptual boundary interacts with certain other objects through a well defined behavior. Time, beauty or colors are not objects but they are properties of other objects. We say that mother (an object) loves her children (another object).

An object has state, behavior and identify; the structure and behavior similar objects are defined an their common class, the terms instance and object are defined in their common class, the terms instance and object are interchangeable.

State

The state of an object encompasses all of the (usually static) properties of the object plus the current (usually dynamic) values of each of these properties.

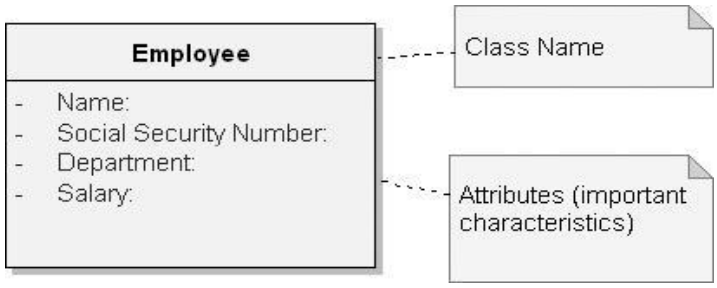

Figure 3–1 Employee Class with Attributes

Figure 3–2 Employee Objects Tom and Kaitlyn

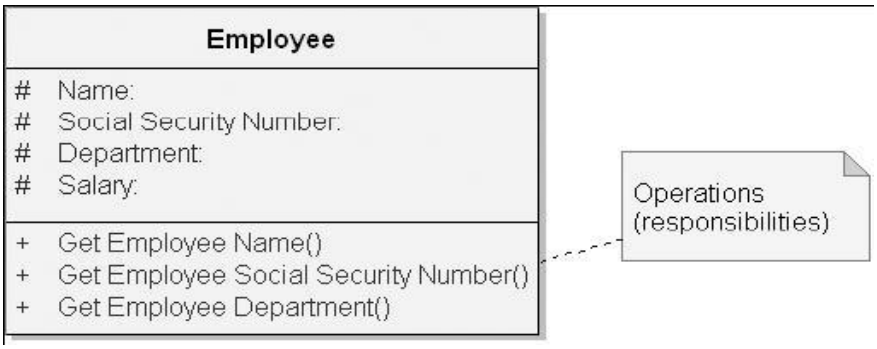

Figure 3–3 Employee Class with Protected Attributes and Public Operations

Behavior

Behavior is how an object acts and reacts, in terms of its state changeable state of object affect its behavior. In vending machine, if we don't deposit change sufficient for our selection, then the machine will probably do nothing. So behavior of an object is a function of its state as well as the operation performed upon it. The state of an object represents the cumulative results of its behavior.

Operations

An operation denotes a service that a class offers to its clients. A client performs 5 kinds of operations upon an object.

- Modifier: An operation that alters the state of an object.
- Selector: An operation that accesses the state of an object but does not alter the state.
- Iterator: An operation that permits all parts of an object to be accessed in some well defined order. In queue example operations, clear, append, pop, remove) are modifies, const functions (length, is empty, front location) are selectors.
- Constructor: An operation that creates an object and/or initializes its state.
- Destructor: An operation that frees the state of an object and/or destroys the object itself.

Identity

Identity is that property of an object which distinguishes it from all other objects.

Object life span

The lifeline of an object extends from the time it is first created (and this first consumes space) until that space is recalled, whose purpose is to allocate space for this object and establish an

initial stable state. Often objects are created implicitly in C++ programming an object by value creates a new objection the stack that is a copy of the actual parameters.

Roles and Responsibilities

we may say that the state and behavior of an object collectively define the roles that an object may play in the world, which in turn fulfill the abstraction's responsibilities.

Most interesting objects play many different roles during their lifetime such as:

- A bank account may have the role of a monetary asset to which the account owner may deposit or withdraw money. However, to a taxing authority, the account may play the

  role of an entity whose dividends must be reported on annually.

RELATIONSHIP AMONG OBJECTS

Objects contribute to the behavior of a system by collaborating with one another. E.g. object structure of an airplane. The relationship between any two objects encompasses the assumptions that each makes about the other including what operations can be performed. Two kinds of objects relationships are links and aggregation.

Links

A link denotes the specific association through which one object (the client) applies the services of another object (the supplier) or through which are object may navigate to another. A line between two object icons represents the existence of pass along this path. Messages are shown as

directed lines representing the direction of message passing between two objects is typically unidirectional, may be bidirectional data flow in either direction across a link.

As a participation in a link, an object may play one of three roles:

- Controller: This object can operate on other objects but is not operated on by other objects. In some contexts, the terms active object and controller are interchangeable.
- Server: This object doesn't operate on other objects; it is only operated on by other objects.
- Proxy: This object can both operate on other objects and be operated on by other objects. A proxy is usually created to represent a real-world object in the domain of the application.

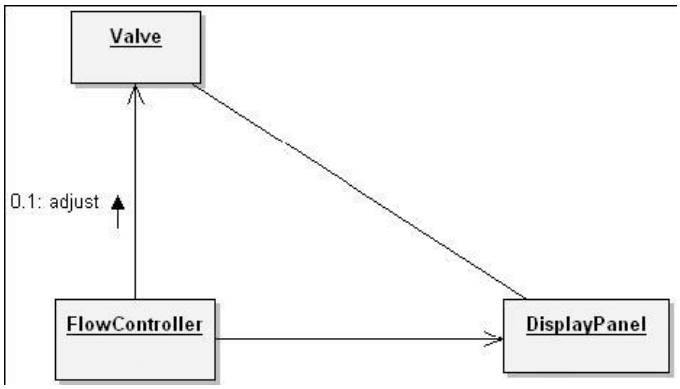

Figure 3–5 Links

In the above figure, FlowController acts as a controller object, DisplayPanel acts as a server object, and Valve acts as a proxy.

Visibility

Consider two objects, A and B, with a link between the two. In order for A to send a message to object B, B must be visible to A. Four ways of visibility

- The supplier object is global to the client
- The supplier object is a programmer to some operation of the client
- The supplier object is a part of the client object.
- The supplier object is locally declared object in some operation of the client.

Synchronization

Wherever one object passes a message to another across a link, the two objects are said to be synchronized. Active objects embody their own thread of control, so we expect their semantics to be guaranteed in the presence of other active objects. When one active object has a link to a passive one, we must choose one of three approaches to synchronization.

1. Sequential: The semantics of the passive object are guaranteed only in the presence of a single active object at a time.
2. Guarded: The semantics of the passive object are guaranteed in the presence of multiple threads of control, but the active clients must collaborate to achieve mutual exclusion.
3. Concurrent: The semantics of the passive object are guaranteed in the presence of multiple threads of control, and the supplier guarantees mutual exclusion.

Aggregation

Whereas links denote peer to peer or client/supplier relationships, aggregation denotes a whole/part hierarchy, with the ability to navigate from the whole (also called the aggregate) to its parts. Aggregation is a specialized kind of association. Aggregation may or may not denote physical containment. E.g. airplane is composed of wings, landing gear, and so on. This is a case of physical containment. The relationship between a shareholder and her shares is an aggregation relationship that doesn't require physical containment.

There are clear trade-offs between links and aggregation. Aggregation is sometimes better because it encapsulates parts as secrets of the whole. Links are sometimes better because they permit looser coupling among objects.

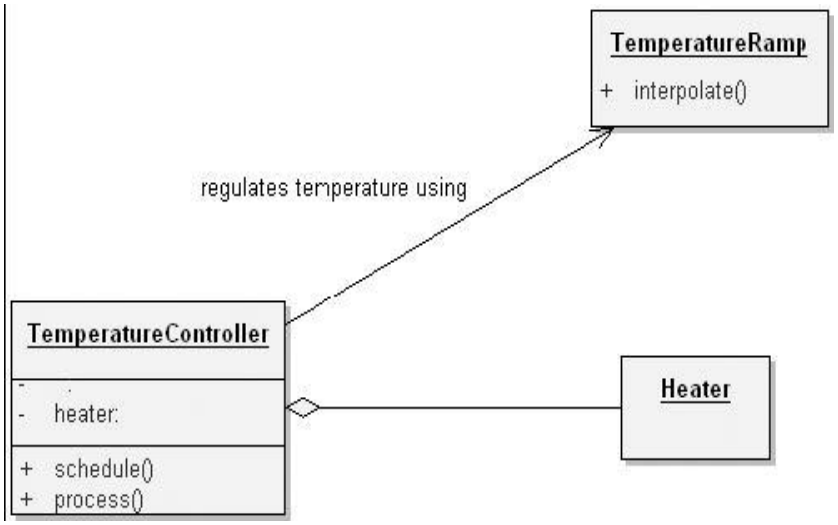

Figure 3–6 Aggregation

THE NATURE OF A CLASS

A class is a set of objects that share a common structure, common behavior and common semantics. A single object is simply an instance of a class. Object is a concrete entity that exists in time and space but class represents only an abstraction. A class may be an object is not a class. **Interface and Implementation**: The interface of a class provides its outside view and therefore emphasizes the abstraction while hiding its structure and secrets of its behavior. The interface primarily consists of the declarations of all the operators applicable to instance of this class, but it may also include the declaration of other classes, constants variables and exceptions as needed to complete the abstraction. The implementation of a class is its inside view, which encompasses the secrets of its behavior. The implementation of a class consists of the class. Interface of the class is divided into following four parts.

- **Public**: a declaration that is accessible to all clients
- **Protected**: a declaration that is accessible only to the class itself and its subclasses
- **Private:** a declaration that is accessible only to the class itself
- **Package:** a declaration that is accessible only by classes in the same package

RELATIONSHIP AMONG CLASSES

We establish relationships between two classes for one of two reasons. First, a class relationship might indicate some kind of sharing. Second, a class relationship might indicate some kind of semantic connection.

There are three basic kinds of class relationships.

- The first of these is generalization/specialization, denoting an “is a” relationship. For instance, a rose is a kind of flower, meaning that a rose is a specialized subclass of the more general class, flower.
- The second is whole/part, which denotes a “part of” relationship. A petal is not a kind of a flower; it is a part of a flower.
- The third is association, which denotes some semantic dependency among otherwise unrelated classes, such as between ladybugs and flowers. As another example, roses and candles are largely independent classes, but they both represent things that we might use to decorate a dinner table.

Association

Of the different kinds of class relationships, associations are the most general. The identification of associations among classes is describing how many classes/objects are taking part in the relationship. As an example for a vehicle, two of our key abstractions include the vehicle and wheels. As shown in Figure 3–7, we may show a simple association between these two classes: the class Wheel and the class Vehicle.

Figure 3–7 Association

Multiplicity/Cardinality

This multiplicity denotes the cardinality of the association. There are three common kinds of multiplicity across an association:

1. One-to-one
2. One-to-many

3. Many-to-many

Inheritance

Inheritance, perhaps the most semantically interesting of the concrete relationships, exists to express generalization/specialization relationships. Inheritance is a relationship among classes wherein one class shares the structure and/or behavior defined in one (single inheritance) or more (multiple inheritance) other classes. Inheritance means that subclasses inherit the structure of their superclass.

Space probe (spacecraft without people) report back to ground stations with information regarding states of important subsystems (such as electrical power & population systems) and different sensors (such as radiation sensors, mass spectrometers, cameras, detectors etc), such relayed information is called telemetry data. We can take an example for Telemetry Data for our illustration.

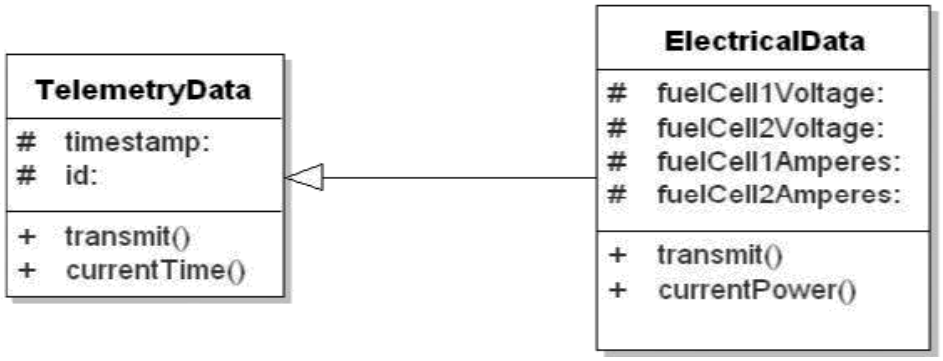

Figure 3–8 ElectricalData Inherits from the Superclass TelemetryData

As for the class ElectricalData, this class inherits the structure and behavior of the class TelemetryData but adds to its structure (the additional voltage data), redefines its behavior (the function transmit) to transmit the additional data, and can even add to its behavior (the function currentPower, a function to provide the current power level).

Single Inheritence

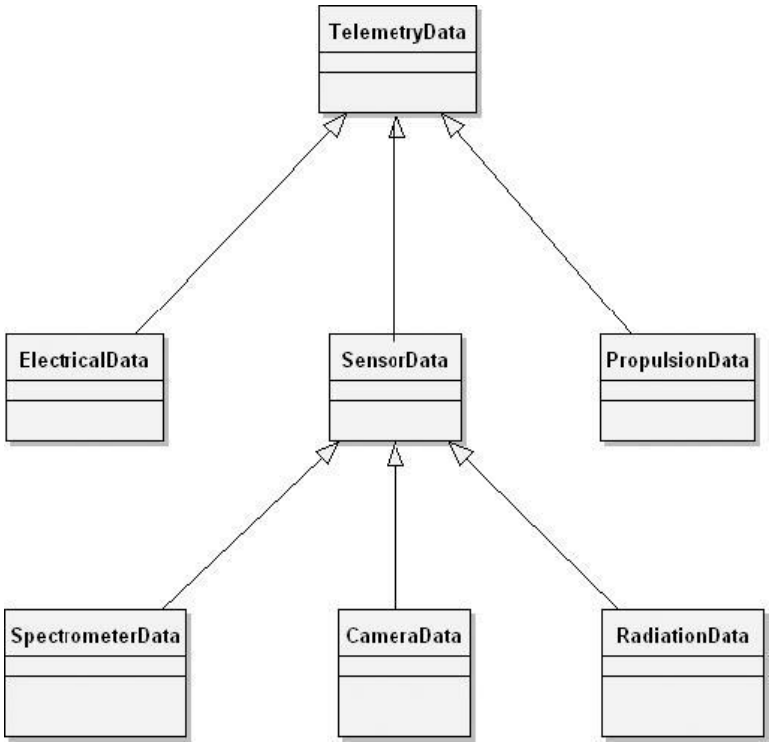

Figure 3–9 Single Inheritance

Figure 3–9 illustrates the single inheritance relationships deriving from the superclass TelemetryData. Each directed line denotes an “is a” relationship. For example, CameraData “is a” kind of SensorData, which in turn “is a” kind of TelemetryData.

Multiple Inheritence

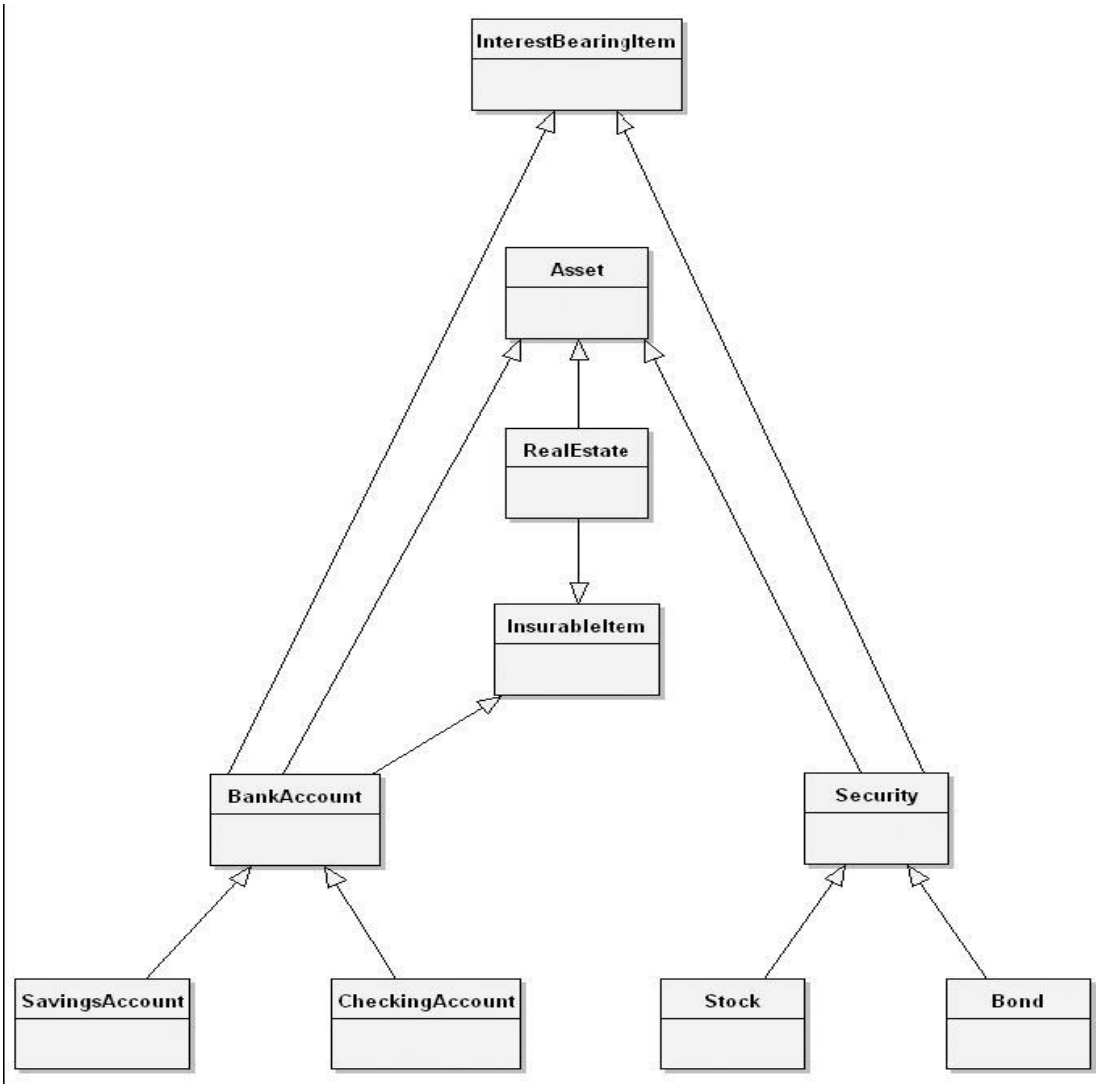

Figure 3–10 Multiple Inheritance

Consider for a moment how one might organize various assets such as savings accounts, real estate, stocks, and bonds. Savings accounts and checking accounts are both kinds of assets typically managed by a bank, so we might classify both of them as kinds of bank accounts, which in turn are kinds of assets. Stocks and bonds are managed quite differently than bank accounts, so we might classify stocks, bonds, mutual funds, and the like as kinds of securities, which in turn are also kinds of assets.

Unfortunately, single inheritance is not expressive enough to capture this lattice of relationships, so we must turn to multiple inheritance. Figure 3–10 illustrates such a class structure. Here we see that the class Security is a kind of Asset as well as a kind of InterestBearingItem. Similarly, the class

BankAccount is a kind of Asset, as well as a kind of InsurableItem and InterestBearingItem.

Polymorphism

Polymorphism is a concept in type theory wherein a name may denote instances of many different classes as long as they are related by some common superclass. Any object denoted by this name is thus able to respond to some common set of operations in different ways. With polymorphism, an operation can be implemented differently by the classes in the hierarchy.

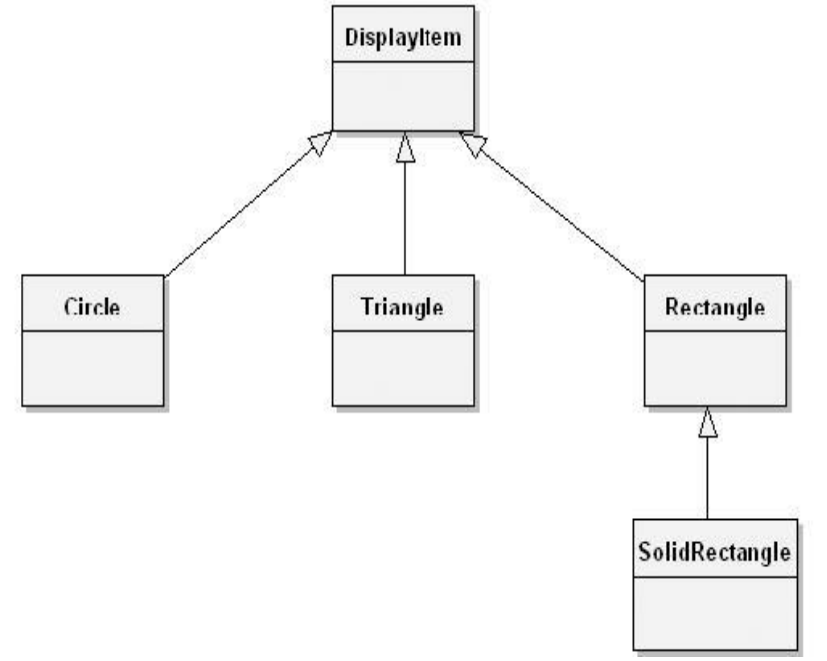

Figure 3-11 Polymorphism

Consider the class hierarchy in Figure 3–11, which shows the base class DisplayItem along with three subclasses named Circle, Triangle, and Rectangle. Rectangle also has one subclass, named SolidRectangle. In the class DisplayItem, suppose that we define the instance variable theCenter (denoting the coordinates for the center of the displayed item), along with the following operations:

- draw: Draw the item.
- move: Move the item.
- location: Return the location of the item.

The operation location is common to all subclasses and therefore need not be redefined, but we expect the operations draw and move to be redefined since only the subclasses know how to draw and move themselves.

Aggregation

We also need aggregation relationships, which provide the whole/part relationships manifested in the class's instances. Aggregation relationships among classes have a direct parallel to aggregation relationships among the objects corresponding to these classes. As shown in Figure 3–12, the class TemperatureController denotes the whole, and the class Heater is one of its parts.

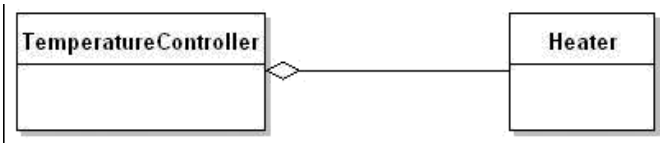

Figure 3–12 Aggregation

THE INTERPLAY OF CLASSES AND OBJECTS

Relationships Between Classes and Objects

•Classes and objects are separate yet intimately related concepts.

•Classes are static- Their existence, semantics and relationships are fixed prior to execution of a program.

•Class of most objects is static.

•Objects are typically created and destroyed at a rapid rate during lifetime of an application.Eg. Air Traffic Control system Planes, flight plans, runways and airspaces

Role of Classes and Objects in Analysis and DesignTwo primary tasks:

1. Identify the classes and objects that form the vocabulary of the problem domain.
2. Invent the structures whereby sets of objects work together to provide behaviors that satisfy the requirements of the problem.•Focus must be outside view of both(Class structure and object structure).

On Building Quality Classes and Objects

Measuring quality of an abstraction

•A system should be built with a minimum set of unchangeable parts; those parts should be as general as possible and all parts should be held in a uniformframework.

How can one know if a given class or object is well designed?

•Coupling

•Cohesion

•Sufficiency

•Completeness

•Primitiveness

IMPORTANCE OF PROPER CLASSIFICATION:

What is Classification?

Classification is the means by which we order knowledge.

Why Classify?

To identify relationships among classes Kinds of relationships, Generalization, Association,Aggregation, Dependences.

Classification is the means where by we order knowledge. There is no any golden path to classification. Classification and object oriented development: The identification of classes and objects is the hardest part of object oriented analysis and design, identification involves both discovery and invention. Discovery helps to recognize the key abstractions and mechanisms that form the vocabulary of our problem domain. Through invention, we desire generalized abstractions as well as new mechanisms that specify how objects collaborate discovery and inventions are both problems of classifications.

classify helps us to identify generalization, specialization, and aggregation hierarchies among classes classify also guides us making decisions about modularizations.

The difficulty of classification

In word processing system, is character class or words are class? Intelligent classification is difficult

The incremental and iterative nature of classification

Intelligent classification is intellectually hard work, and that it best comes about through an incremental and iterative process. Such processes are used in the development of software technologies such as GUI, database standards and programming languages.

IDENTIFYING CLASSES AND OBJECTS

Classical and modern approaches: There are three general approaches to classifications.

- Classical categorization
- Conceptual clustering
- Prototypal theory

Classical categorizations

All the entities that have a given property or collection of properties in common forms a category. Such properties are necessary and sufficient to define the category. i.e. married people constitute a category i.e. either married or not. The values of this property are sufficient to decide

to which group a particular person belongs to the category of tall/short people, where we can agree to some absolute criteria.

Conceptual clustering

It is a more modern variation of the classical approach and largely derives from attempts to explain how knowledge is represented in this approach, classes are generated by first formulating conceptual description of these classes and then classifying the entities according to the descriptions.

Prototype theory

It is more recent approach of classify where a class of objects is represented by a prototypical object, an object is considered to be a member of this class if and only if it resembles this prototype in significant ways. e.g. category like games, not in classical since no single common properties shared by all games, e.g. classifying chairs (beanbag chairs, barber chairs, in prototypes theory, we group things according to the degree of their relationship to concrete prototypes.

Object oriented Analysis

An analysis, we seek to model the world by discovering. The classes and objects that form the vocabulary of the problem domain and in design, we invent the abstractions and mechanisms that provide the behavior that this model requires following are some approaches for analysis that are relevant to object oriented system.

Classical approaches

It is one of approaches for analysis which derive primarily from the principles of classical categorization. e.g. Shlaer and Mellor suggest that classes and objects may come from the following sources:

- Tangible things, cars, pressure sensors
- Roles – Mother, teacher, politician
- Events – landing, interrupt
- Interactions – meeting

(i) People – human who carry out some function

(ii) Places – Areas set for people or thing

(iii) Things – Physical objects (tangible)

(iv) Organizations – organized collection of people resources

(v) Concepts – ideas

(vi) Events – things that happen

(i) Structure

(ii) Dences
(iii) Events remembered (historical)

(iv) Roles played (of users)
(v) Locations (office, sites)

(vi) Organizational units (groups)

Behavior Analysis

Dynamic behavior also be one of the primary source of analysis of classes and objects things can are grouped that have common responsibilities and form hierarchies of classes (including superclasses and subclasses).

Domain Analysis

Domain analysis seeks to identify the classes and objects that are common to all applications within a given domain, such as patient record tracking, compliers, missile systems etc. Domain analysis defined as an attempt to identify the objects, operations and, relationships that are important to particular domain.

Use case Analysis

Use case is defined as a particular form pattern or sequence of interrelated events. Use case analysis is applied as early as requirements analysis, at which time end users, other domain experts and the development team enumerate the scenarios that are fundamental to system's operation.

CRC cards

CRC are a useful development tool that facilitates brainstorming and enhances communication among developers. It is 3 x 5 index card (class/Responsibilities/collaborators i.e. CRC) upon

which the analyst writes in pencil with the name of class (at the top of card), its responsibilities(on one half of the card) and its collaborators (on the other half of the card). One card is created for each class identified as relevant to the scenario. CRC cards are arranged to represent generalization/specialization or aggregation hierarchies among the classes.

Informal English Description

Proposed by Abbott. It is writing an English description of the problem (or a part of a problem) and then underlining the nouns and verbs. Nouns represent candidate objects and the verbs represent candidate operations upon them. it is simple and forces the developer to work in the vocabulary of the problem space.

Structured Analysis

Same as English description as an alternative to the system, many CASE tools assists in modeling of the system. In this approach, we start with an essential model of the system, as described by data flow diagrams and other products of structured analysis. From this model we may proceed to identify the meaningful classes and objects in our problem domain in 3 ways.

- Analyzing the context diagrams, with list of input/output data elements; think about what they tell you or what they describe e.g. these make up list of candidate objects.
- Analyzing data flow domains, candidate objects may be derived from external entities, data stores, control stores, control transformation, candidate classes derive from data flows and candidate flows.
- By abstraction analysis: In structured analysis, input and output data are examined and followed inwards until they reach the highest level of abstraction.

KEY ABSTRACTIONS AND MECHANISMS

Identifying key abstractions Finding key abstractions

A key abstraction is a class or object that forms part of the vocabulary of the problem domain. The primary value of identifying such abstractions is that they give boundaries to our problems.

Refining key abstractions

Once we identify a certain key abstraction as a candidate, we must evaluate it. Programmer must focus on questions. How we objects of this class created? What operations can be done on such objects? If there are not good answers to such questions, then the problem is to be thought again and proposed solution is to be found out instead of immediately starting to code among the problems placing classes and objects at right levels of abstraction is difficult.

Identifying Mechanisms Finding Mechanism

A mechanism is a design decision about how collection of objects cooperates. Mechanisms represent patterns of behavior e.g. consider a system requirement for an automobile: pushing the accelerator should cause the engine to run faster and releasing the accelerator should cause the engine to run slower. Any mechanism may be employed as long as it delivers the required behavior and thus which mechanism is selected is largely a matter of design choice.