CHAPTER 2
Object Oriented Concepts and Modeling

2.1 Introduction to class, Object, inheritance, polymorphism

Class:

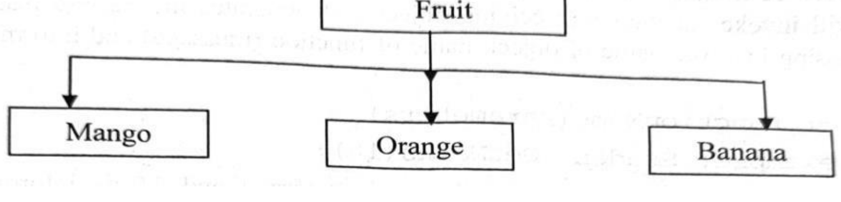
- A class represents a group of objects that share common properties, behavior and relationships. There are various objects like computer, furniture, building, bus, box etc. in real world.
- The computers in our college lab or other college's lab or our home are similar objects. These different computers may be manufactured by different companies in different models. Although they have different models, capacities and appearance, they have common characteristics to identify them. For example: all have memory, hard disk, monitor, CPU and others.
- The general or common name to describe objects of common or similar characteristics and behaviors is called class. Thus, the name 'Computer' may be class name which represents my personal computer, lab computer and other computers.
- A class is blue print or template which describes characteristics of similar objects.
- In other word, a class is an identifier which is general name (i.e. family/group name) defined to represent objects of similar characteristics.

Objects:

- An object is an identifiable entity with some characteristics and behavior.
- An object is an instance of a class.
- Creating an object of a class is like defining a variable of a data type. Once a class has been declared, we can create variables of that type by using the class name.
- An object doesn't exist until an instance of the class has been created; the class is just definition. When the object is physically created, space for that object is allocated in primary memory.
- It is possible to have multiple objects created from one class.

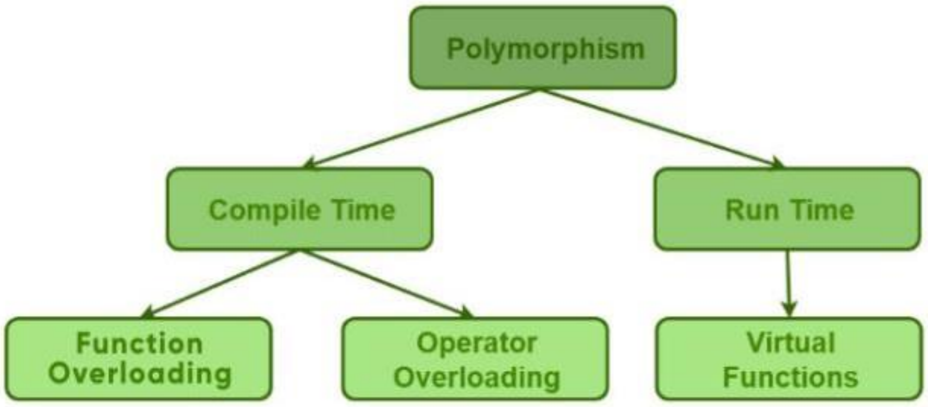
Inheritance:

- Inheritance is the ability of creating new class from existing class with new data methods.
- The existing class is called base class and the newly created classes al called derived classes.
- The derived class inherits all the features, except private members, inherent in the base class. Thus, inheritance supports program reusability and reliability.
- For example, a programmer can create a base class named Fruit and define derived classes as Mango, Orange, Banana, etc. Each of them has some common characteristics like taste, price and season. The common characteristics are written in base class. Each of these derived classes, has all the features of the base class (i.e. Fruit) with additional attributes or features specific to these newly created derived classes. Mango would have its own defined features, orange would have its own defined features, and Banana would have its own defined features, and so on.



Polymorphism:

- The word polymorphism is derived from two Greek words poly and morphe. The word poly means many and morphe means forms. Thus, polymorphism means the ability to take more than one form.
- The OOP supports polymorphism through function overloading, operator overloading, function overriding and template.



Compile time polymorphism:

- The function to be invoked is known at the compile time.
- It is achieved by function overloading and operator overloading.
- It provides fast execution as it is known at the compile time.
- It is less flexible as mainly all the things execute at the compile time.

Run Time Polymorphism:

- The function to be invoked is known at the run time.
- It is achieved by virtual functions and pointers.
- It provides slow execution as it is known at the run time.
- It is more flexible as all the things execute at the run time.

Encapsulation:

- Encapsulation is an ability to bind (package) functions and data together in a place and hide (or prevent) the data from unauthorized use from other parts of the program.
- Thus, function and data in OOP are always together within an object that provides data hiding, also known as encapsulation. Thus, data is hidden within an object.
- It enables data hiding, hiding irrelevant information from the users of a class and exposing only the relevant details required by the user. We can expose our operations to the outside world and hide the details of what is needed to perform that operation. We can thus protect the internal state of an object by hiding its attributes from the outside world.
- Example from real world: Consider the smart phone you are using now. You are not worried about the internal operations of the smart phone. You only know and care about the operations or functions it exposes to you such as making call, sending SMS or using your apps.

Data Abstraction:

- Abstraction refers to an act of representing essential features without including background details or explanations. Thus, data abstraction is methodology that supports use of compound or complex object without its detail knowledge.
- For example, a class car would be made up of engine, gearbox, steering objects and many more components. To build car class, one does not need to know how the different components of the car work internally, but only how to interface with them.

Advantages of object oriented software development

- Some of the advantages of object oriented software development are:
- Less maintenance cost mostly because it is modular.
- Better code reusability due to features such as inheritance and hence faster development.
- Improved code reliability and flexibility
- Easy to understand due to real world modelling.
- Better abstraction at object level.
- Reduced complexity during the transitions from one development phase to another.

2. Object Oriented Concepts and Modeling Object Oriented System Development:

- In Object-Oriented Development, we apply object orientation across every system development activity such as requirement analysis, design, programming, testing, and maintenance.
- For instance, an object oriented analysis (OOA) will usually have the following steps:
  - Identifying the system functionality
  - Identifying the involved objects
  - Recognizing the object classes
  - Analyzing the objects to fulfil the system functionality

Object Oriented Development Cycle

- The motive of the software development process that consists of analysis, design, implementation, testing, and refinement is to transform user's needs into a software solution that satisfies those needs.
- Some people view software development process as interesting but feel it has little importance in developing software.
- In general, dynamics of software development provides little room for such shortcuts, and bypasses have been less than successful.
- The object oriented approach requires a more rigid process to do things right.

- You need not see code until after about 25 percent of the development time, because you need to spend more time in gathering requirements, developing a requirement model and an analysis model, then turning them into the design model.

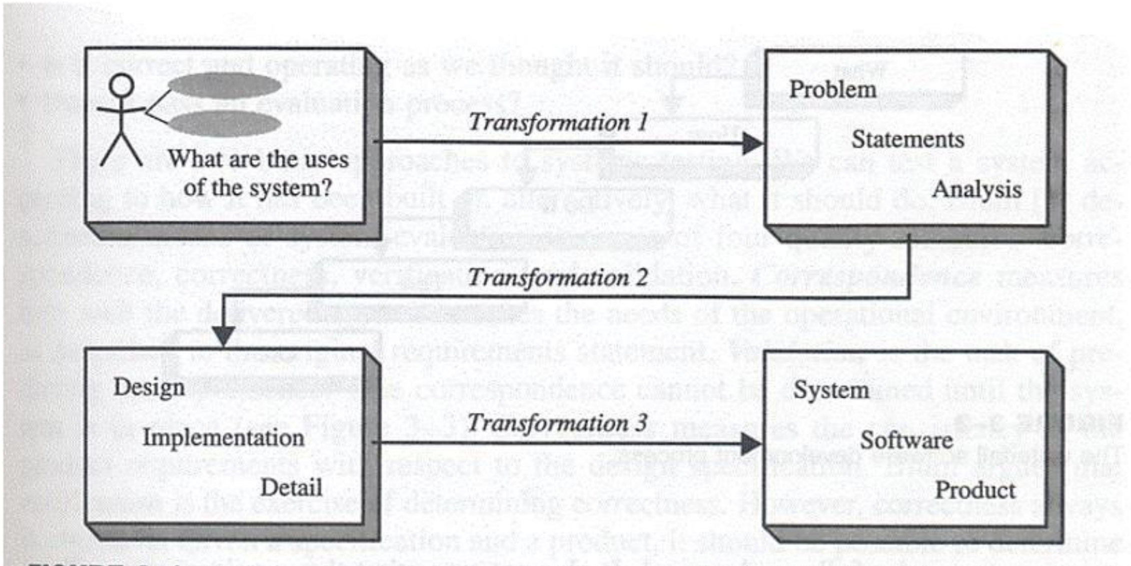


FIGURE 3–1
Software process reflecting transformation from needs to a software product that satisfies those needs.

- **transformation 1(analysis)** - translates user's need into system's requirements & responsibilities
  - how they use system can give insight into requirements, e.g.: analyzing incentive payroll - capacity must be included in requirements
- **transformation 2 (design)** - begins with problem statement, ends with detailed design that can be transformed into operational system
  - Includes bulk of development activity, include definition on how to build software, its development, its testing, design description + program+ testing material
- **Transformation 3 (implementation)** - refines detailed design into system deployment that will satisfy user's needs.

Object Oriented Analysis:

- The OOA phase deals with the investigation of the problem and requirements, rather than finding a solution to the problem.
  - The phase of OOA the typical question starts with what...? Like
    - "What will my program need to do?"
    - "What will the classes in my program be?" and
    - "What will each class be responsible for?"
- Priority is given to find and describe the objects or concepts in the problem domain.
- Object oriented analysis emphasizes the building of real-world model using the object oriented view of the world.

- Object Oriented Analysis includes following activities:
  - Finding the objects
  - Organizing the objects
  - Describing how the object interacts
  - Defining the operations of the objects
  - Defining the objects internally

Elements in Object Model

- Object Diagrams are derived from class diagrams that uses notations similar to class diagram.
- Objects and associations are the main two elements of the diagram.
- Example: A simple Object model of Student Enrollment System

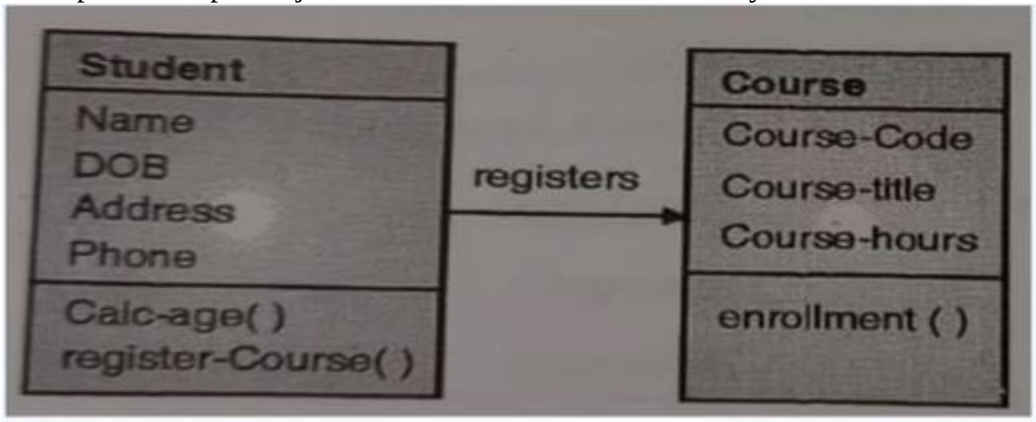


Fig: Class Diagram for Student Enrollment

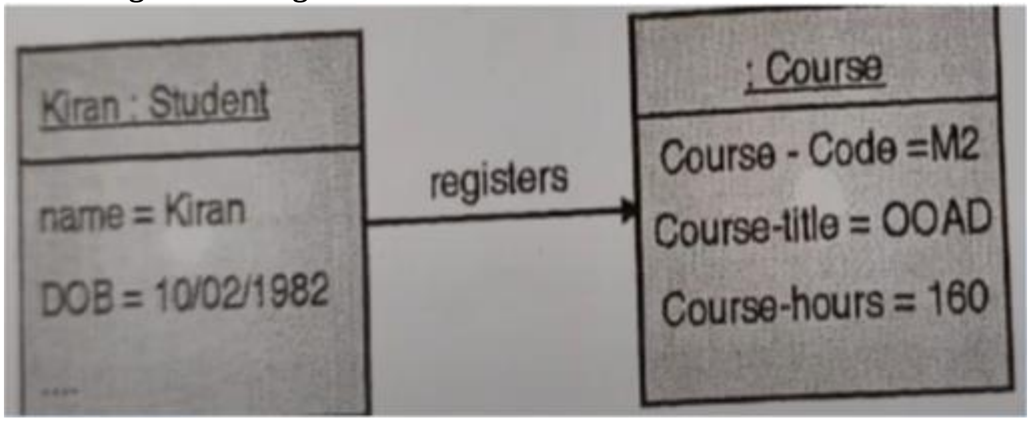


Fig: Object Diagram for Student Enrollment

Identifying the Elements of an Object Model

- If you look around a room, there is a set of physical objects that can be easily identified, classified, and defined (in terms of attributes and operations). But when you "look around" the problem space of a software application, the objects may be more difficult to comprehend.
- We can begin to identify objects by examining the problem statement or performing a "grammatical parse" on the processing narrative for the system to be built.
- Objects are determined by underlining each noun or noun clause and entering it in a simple table. Synonyms should be noted.
- If the object is required to implement a solution, then it is part of the solution space; otherwise, if an object is necessary only to describe a solution, it is part of the problem space. What should we look for once all of the nouns have been isolated? Objects manifest

themselves in one of the ways represented in figure.

- Objects can be:
  - **External entities** (e.g., other systems, devices, people) that produce or consume information to be used by a computer-based system.
  - **Things** (e.g., reports, displays, letters, signals) that are part of the information domain for the problem.
  - **Occurrences or events** (e.g., a property transfer or the completion of a series of robot movements) that occur within the context of system operation.
  - **Roles** (e.g., manager, engineer, salesperson) played by people who interact with the system.
  - **Organizational units** (e.g., division, group, team) that are relevant to an application.
  - **Places** (e.g., manufacturing floor or loading dock) that establish the context of the problem and the overall function of the system.
  - **Structures** (e.g., sensors, four-wheeled vehicles, or computers) that define a class of objects or in the extreme, related classes of objects.

Example: Defining objects during early stages of analysis: SafeHome system

- To illustrate how objects might be defined during the early stages of analysis, we return to the SafeHome security system example. The processing narrative is reproduced:

Input:

- SafeHome software enables the homeowner to configure the security system when it is installed, monitors all sensors connected to the security system, and interacts with the homeowner through a keypad and function keys contained in the SafeHome control panel.
- During installation, the SafeHome control panel is used to "program" and configure the system. Each sensor is assigned a number and type, a master password is programmed for arming and disarming the system, and telephone number(s) are input for dialing when a sensor event occurs.
- When a sensor event is sensed by the software, it rings an audible alarm attached to the system. After a delay time that is specified by the homeowner during system configuration activities, the software dials a telephone number of a monitoring service, provides information about the location, reporting and the nature of the event that has been detected. The number will be redialed every 20 seconds until telephone connection is obtained.
- All interaction with SafeHome is managed by a user-interaction subsystem that reads input provided through the keypad and function keys, displays prompting messages on the LCD display, and displays system status information on the LCD display. Keyboard interaction takes the following form

Output:

- SafeHome software enables the homeowner to configure the security system when it is installed, monitors all sensors connected to the security system, and interacts with the homeowner through a keypad and function keys contained in the SafeHome control panel.
- During installation, the SafeHome control panel is used to "program" and configure the system. Each sensor is assigned a number and type, a master password is programmed for arming and disarming the system, and telephone number(s) are input for dialing when a sensor event occurs.
- When a sensor event is recognized, the software invokes an audible alarm attached to the system. After a delay time that is specified by the homeowner during system configuration activities, the software dials a telephone number of a monitoring service, provides information about the location, reporting the nature of the event that has been detected. The telephone number will be redialed every 20 seconds until telephone connection is obtained.

- All interaction with SafeHome is managed by a user-interaction subsystem that reads input provided through the keypad and function keys, displays prompting messages on the LCD display, and displays system status information on the LCD display. Keyboard interaction takes the following form

- All verbs are SafeHome processes
- All nouns are either external entities , data or control objects , or data stores
- Nouns and verbs can be attached to one another
- Extracting the nouns, creates a number of potential objects:

Potential Object/Class	General Classification
homeowner	role or external entity
sensor	external entity
control panel	external entity
installation	occurrence
system (alias security system)	thing
number, type	not objects, attributes of sensor
master password	thing
telephone number	thing
sensor event	occurrence
audible alarm	external entity
monitoring service	organizational unit or external entity

Selection characteristics that should be used when considering each potential object for inclusion in the analysis model:

1. **Retained information**: The potential object will be useful during analysis only if information about it must be remembered so that the system can function.
2. **Needed services**: The potential object must have a set of identifiable operations that can change the value of its attributes in some way.
3. **Multiple attributes**: During requirement analysis, the focus should be on "major" information; an object with a single attribute may, in fact, be useful during design, but is probably better represented as an attribute of another object during the analysis activity.
4. **Common attributes**: A set of attributes can be defined for the potential object and these attributes apply to all occurrences of the object.
5. **Common operations**: A set of operations can be defined for the potential object and these operations apply to all occurrences of the object.
6. **Essential requirements**: External entities that appear in the problem space and produce or consume information essential to the operation of any solution for the system will almost always be defined as objects in the requirements model
   - To be a legitimate object for inclusion in the requirements model, a potential object should satisfy all (or almost all) of these characteristics.
   - The decision for inclusion of potential objects in the analysis model is somewhat subjective
   - Applying these selection characteristics to the list of potential SafeHome objects gives:

Potential Object/Class	Characteristic Number That Applies
homeowner	rejected: 1, 2 fail even though 6 applies
sensor	accepted: all apply
control panel	accepted: all apply
installation	rejected
system (alias security system)	accepted: all apply
number, type	rejected: 3 fails, attributes of sensor
master password	rejected: 3 fails
telephone number	rejected: 3 fails
sensor event	accepted: all apply
audible alarm	accepted: 2, 3, 4, 5, 6 apply
monitoring service	rejected: 1, 2 fail even though 6 applies

Specifying the Attributes

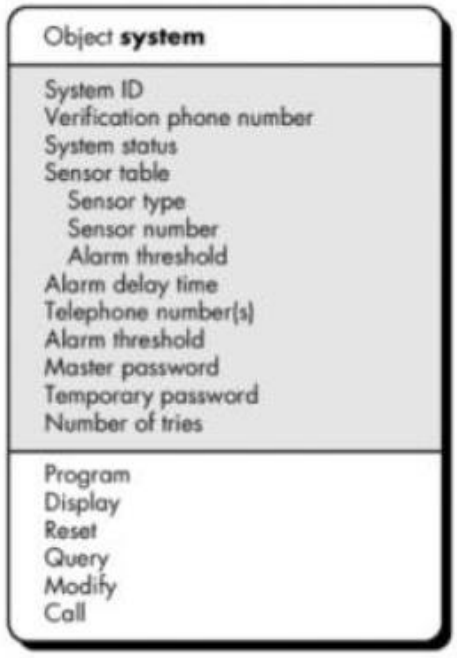
- Attributes describe an object that has been selected for inclusion in the analysis model. In essence, it is the attributes that define the object—that clarify what is meant by the object in the context of the problem space.
- To determine object attributes:
  - Study the processing narrative (or statement of scope) for the problem and select those things that reasonably "belong" to the object.
  - Answer the following question for each object: "What data items (composite and/or elementary) fully define this object in the context of the problem at hand?"

Defining Operations

- Operations change one or more attribute values that are contained within an object.
- Three categories of objects:
  i. Operations that manipulate data in some way (e.g., adding, deleting, reformatting, selecting)
  ii. Operations that perform a computation
  iii. Operations that monitor an object for the occurrence of a controlling event.
- To derive a set of operations for the objects of the analysis model, study the processing narrative (or statement of scope) for the problem and select those operations that reasonably belong to the object.
  - Study the grammatical parse again to isolate verbs.
  - Some verbs will be legitimate operations and can be easily connected to a specific object.
  - e.g. from SafeHome processing narrative:
    - "sensor is assigned a number and type"
    - "A master password is programmed for arming and disarming the system."
  - These two phrases indicate a number of things:
    - An assign operation is relevant for the sensor object.
    - A program operation will be applied to the system object.
  - Also consider communication between objects.

Finalizing the Object Definition

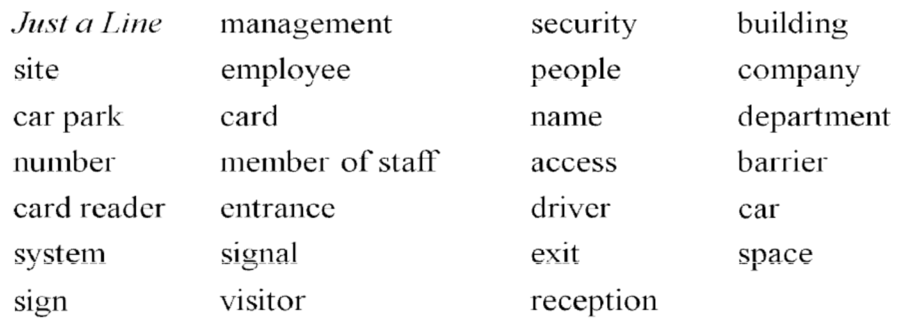
- Definition of operations is the last step in completing the specification of an object.
- Generic life history of an object can be defined by recognizing that the object must be created, modified, manipulated or read in other ways, and possibly deleted.
- Some of the operations can be ascertained from likely communication between objects.
- e.g.:
  - sensor event will send a message to system to display the event location and number
  - control panel will send system a reset message to update system status
  - audible alarm will send a query message
  - control panel will send a modify message to change one or more attributes without reconfiguring the entire system object;
  - Sensor event will also send a message to call the phone number(s) contained in the object.
- **Final Result**


Object **system**
System ID
Verification phone number
System status
Sensor table
Sensor type
Sensor number
Alarm threshold
Alarm delay time
Telephone number(s)
Alarm threshold
Master password
Temporary password
Number of tries
Program
Display
Reset
Query
Modify
Call

Example of candidate objects

Just a Line management wishes to increase security, both in their building and on site, without antagonizing their employees. They would also like to prevent people who are not part of the company from using the Just a Line car park.
It has been decide to issue identity cards to all employees, which they are expected to wear while on the Just a Line site. The cards records the name, department and number of the member of staff, and permit access to the Just a Line car park.
A barrier and a card reader are placed at the entrance to the car park. The driver of an approaching car insert his or her numbered card in the card reader, which then checks that the card number is known to the Just a Line system. If the card is recognized, the reader sends a signal to raise the barrier and the car is able to enter the car park.
At the exit, there is also a barrier, which is raised when a car wishes to leave the car park.
When there are no spaces in the car park a sign at the entrance display "Full" and is only switched off when a car leaves.
Special visitor's cards, which record a number and the current date, also permit access to the car park. Visitor's cards may be sent out in advance, or collected from reception. All visitor's cards must be returned to reception when the visitor leaves Just a Line.

Candidate objects in given case are:

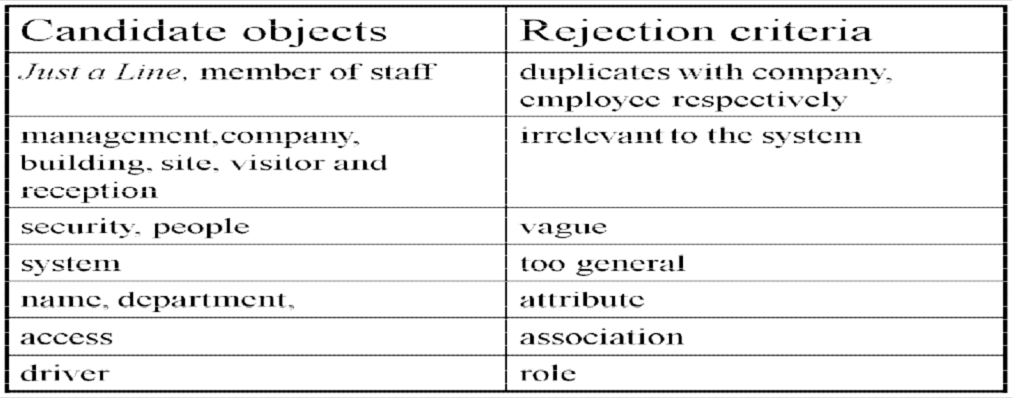


Just a Line	management	security	building
site	employee	people	company
car park	card	name	department
number	member of staff	access	barrier
card reader	entrance	driver	car
system	signal	exit	space
sign	visitor	reception	

Candidate objects' rejection rule

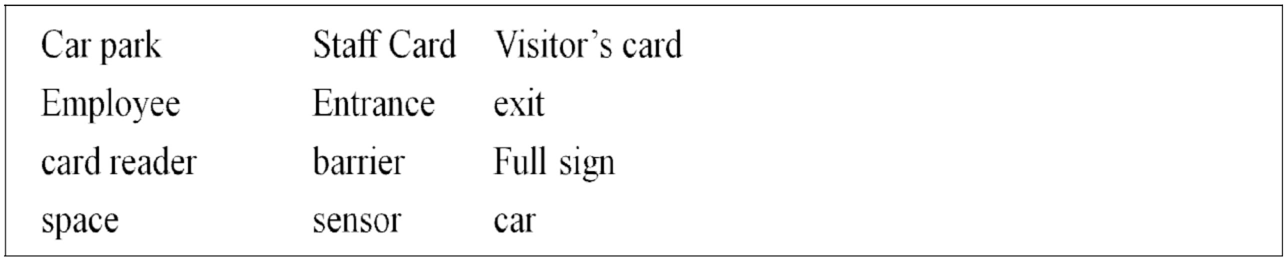
- ***Duplicates**: if two or more objects are simply different names for the same thing.*
- ***Irrelevant**: objects which exists in the problem domain, but which are not intended.*
- ***Vague**: when considering words carefully it sometimes becomes clear that they do not have a price meaning and cannot be the basis of a useful in the system.*
- ***General**: the meaning is too broad.*
- ***Attributes**: as the attribute of objects.*
- ***Associations**: actually represents the relationships between objects.*
- ***Roles**: sometimes objects referred to by the role they play in a particular part of the system.*

Rejected Candidate objects



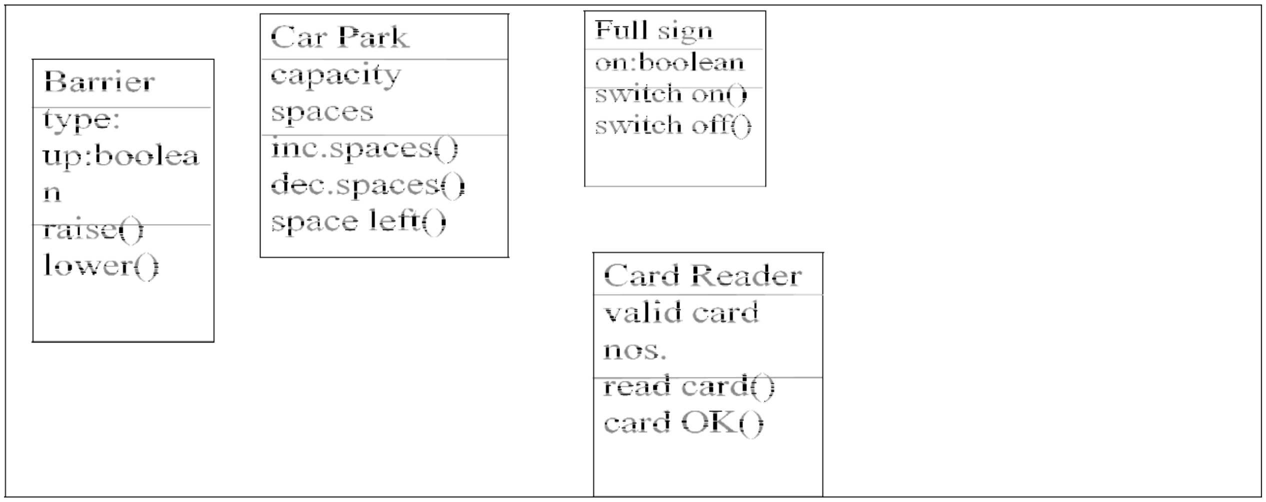
Candidate objects	Rejection criteria
Just a Line, member of staff	duplicates with company, employee respectively
management,company, building, site, visitor and reception	irrelevant to the system
security, people	vague
system	too general
name, department,	attribute
access	association
driver	role

Final Objects

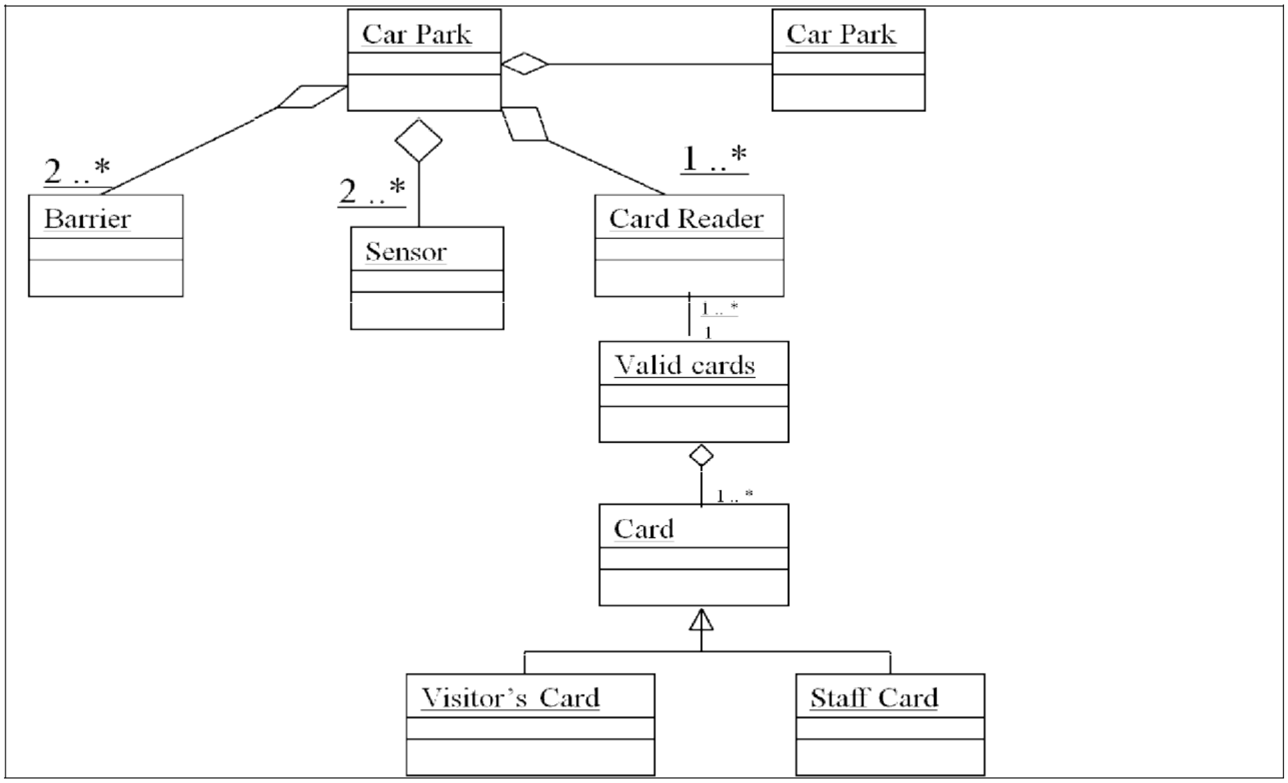


Car park	Staff Card	Visitor's card
Employee	Entrance	exit
card reader	barrier	Full sign
space	sensor	car

D. Define class Attributes and Operations



E. Define Objects Relationship



Car Park
Car Park
2 ..*
Barrier
2 ..*
Sensor
1 ..*
Card Reader
1 .. *
1
Valid cards
1 .. *
Card
Visitor's Card
Staff Card