Characteristic of Object

Citra N., MT
UNIKOM

Why Object

- Objects are easier for people to understand: This is because the objects are derived from the business that we’re trying to automate, rather than being influenced too early by computer-based procedures or data storage requirements.
- For example, in a bank system, we program in terms of bank accounts, bank tellers and customers, instead of diving straight into account records, deposit and withdrawal procedures, and loan qualification algorithms.

- Data and processes are not artificially separated
- In traditional methods, the data that needs to be stored is separated early on from the algorithms that operate on that data and they are then developed independently.
- In Object oriented development, data and processes are kept together in small, easy-to-manage packages; data is never separated from the algorithms.

- Code can be reused more easily: With the traditional approach, we start with the problem that needs to be solved and allow that problem to drive the entire development. We end up with a monolithic solution to today's problem. But tomorrow always brings a different problem to solve; no matter how close the new problem is to the last one we dealt with, we're unlikely to be able to break open our monolithic system and make it fit.
- Object orientation is mature and well proven. Applying objects in such areas as software, databases and networks is now well understood.

- A model is a representation of a problem domain or a proposed solution that allows us to talk, or reason, about the real thing. This allows us to increase our understanding and avoid potential pitfalls.
- A model allows us to learn a without actually building anything. Much of software development involves creating and refining models, rather than cutting lines of code.

- From the perspective of human cognition, an object is any of the following:
- A tangible and/or visible thing
- Something that may be comprehended intellectually
- Something toward which thought or action is directed

- “An object is an entity that has state, behavior, and identity. The structure and behavior of similar objects are defined in their common class. The terms instance and object are interchangeable.”

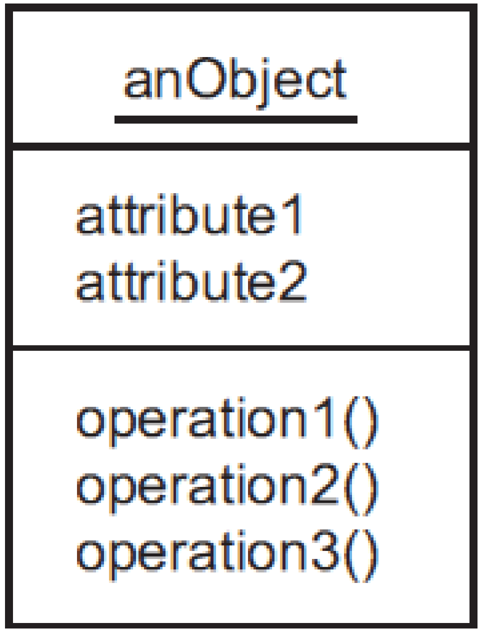
Object

- All objects have **attributes**: for example, a car has a manufacturer, a model number, a color and a price; a dog has a breed, an age, a color and a favorite toy. Objects also have **behavior**: a car can move from one place to another and a dog can bark.

Attributes

- An attribute is a property of an object, such as its size, position, name, price, font, interest rate, or whatever. In UML, each attribute can be given a type, which is either a class or a primitive. If we choose to specify a type, it should be shown to the right of the attribute name, after a colon. (We might choose not to specify attribute types during analysis, either because the types are obvious or because we don't want to commit ourselves yet.)
- Attributes can be shown on a class diagram by adding a compartment under the class name. To save space, we can document them separately instead as an attribute list, complete with descriptions.

Object notation



standard notation but in non-standard location

Classes

- 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.
- A class is a category. All objects that belong to the same category have the same attributes and operations (but the values of the attributes may change from object to object).

Class

- When designing a class hierarchy, you should bear in mind that most superclasses are abstract. This follows from the fact that inheritance hierarchies are naturally derived from the bottom up:

  1. We look for the concrete concepts that exist in our problem domain and reason about their knowledge and behavior.

  2. We look for commonalities between the concrete classes so that we can introduce more general superclasses.

  3. We group superclasses into more superclasses, until we arrive at our most general root class (Fruit or Collection, for example).

Relationship

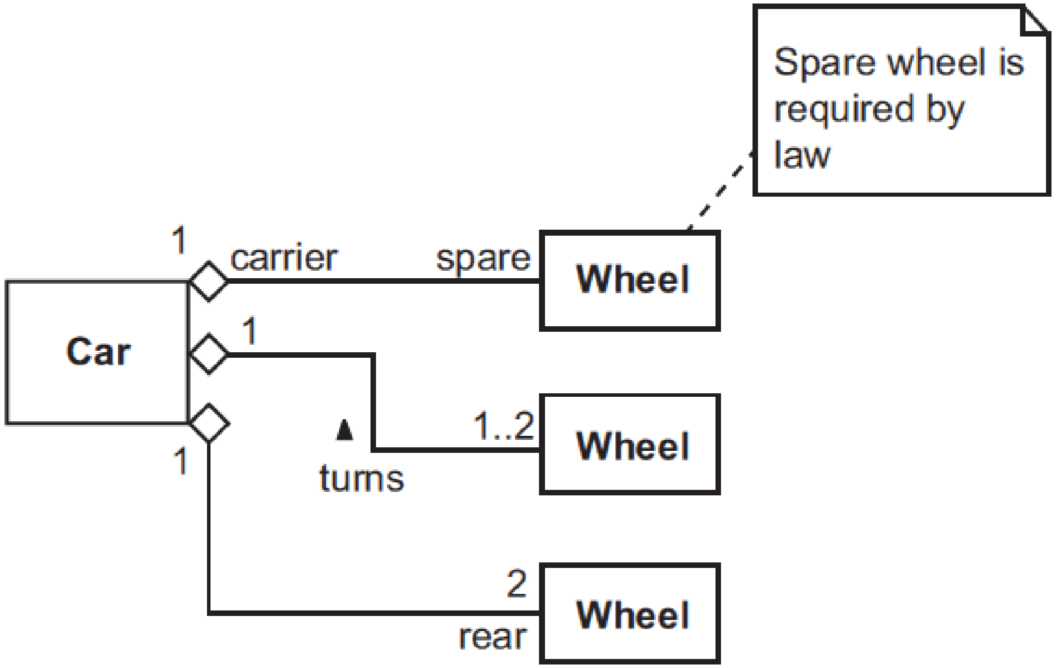
- The relationship between any two objects encompasses the assumptions that each makes about the other, including what operations can be performed and what behavior results. We have found that two kinds of object relationships are of particular interest in object-oriented analysis and design, namely:

  1. Links, denote peer-to-peer or client/supplier relationships

  2. Aggregation, denotes a whole/part hierarchy, with the ability to navigate from the whole (also called the aggregate) to its parts. In this sense, aggregation is a specialized kind of association

Association, Labels, Roles and Comments

- All relationships, except inheritance, can be given an association label, indicating the nature of the association. If it's not obvious which way the association name shc l.

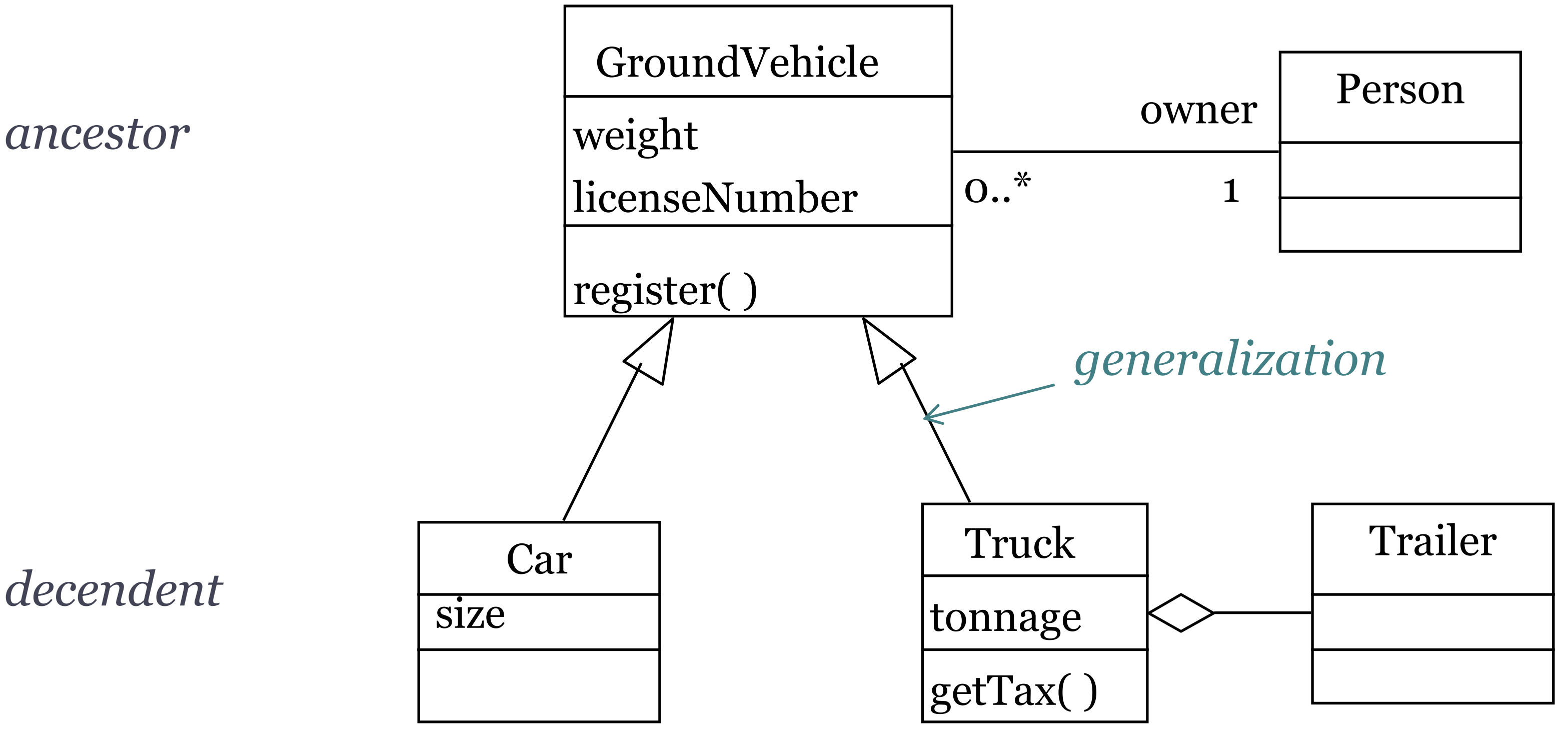


Generalization

- The concept of a class allows us to make statements about a set of objects that we treat exactly the same way. But sometimes we run into objects that are only partially alike.
- A generalization is a taxonomic relationship between a more general classifier and a more specific classifier. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier inherits the features of the more general classifier.

What is Generalization?

- One class inherits from another



Association

- An association specifies a semantic relationship that can occur between typed instances. An instance of an association is called a link
- An association between two classes indicates that objects (instances) of one class may be related (linked) to objects of the other class. You specify an association at the class level; you specify a link at the object level.

Association

- Association is a weak form of connection: the objects may be part of a group, or family, of objects but they’re not completely dependent on each other.
- For example, consider a car, a driver, a passenger and another passenger. When the driver and the two passengers are in the car, they’re associated: they all go in the same direction, they occupy the same volume in space, and so on. But the association is loose: the driver can drop off one of the passengers to go their separate way, so that the passenger is no longer associated with the other objects.

Aggregation (“is a part of”)

- Aggregation: A special form of association that specifies a whole-part relationship between the aggregate (whole) and a component part.
- Formally, in the UML, aggregation is considered to be a specific type of association, where the class on one end of the association represents a whole and the class at the other end represents a part.

Aggregation

- Aggregation means putting objects together to make a bigger object. Manufactured items usually form aggregations: for example, a microwave is made up of a cabinet, a door, an indicator panel, buttons, a motor, a glass plate, a magnetron, and so on.
- Aggregations usually form a part–whole hierarchy. Aggregation implies close dependency, at least of the whole to the part; for example, a magnetron is still a magnetron if you take it out of its microwave, but the microwave would be useless without the magnetron, because it wouldn't be able to cook anything.

Composition Aggregation

- Composite aggregation, also known as composition, is a special form of aggregation where in each part may belong to only one whole at a time.
- Composite aggregation is a strong form of aggregation that requires a part instance be included in at most one composite at a time. If a composite is deleted, all of its parts are normally deleted with it.
- Formally, composition is a specific kind of aggregation. In aggregation, a part may belong to more than one whole at the same time; in composite aggregation, however, the object may belong to only one whole at a time. The parts are destroyed whenever the whole is destroyed—except for those parts that have been removed prior to the deletion of the whole.

Inheritance

- Inheritance: The mechanism by which more specific elements incorporate structure and behavior of more general elements.
- Inheritance refers to the mechanism by which a specialized class adopts—that is, inherits—all the attributes, operations, and relationships of a generalized class

Encapsulation

- Every day you use objects without knowing how they work or what their internal structure is. This is a useful aspect of the way we human objects interact with other objects. It keeps us from having to know too much. It also means that we can easily switch to another object with a different internal structure as long as it behaves the same way externally.
- Encapsulation refers to an object hiding its attributes behind its operations (it seals the attributes in a capsule, with operations on the edge). Hidden attributes are said to be private.

Polymorphism

- Polymorphism means the ability to take on many forms. The term is applied both to objects and to operations.
- 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. In this manner, a subclass can extend the capabilities of its superclass or override the parent's operation,

- **One Operation, Many Methods**
- A polymorphic operation is one whose method may take on many forms based on the class of the object carrying it out.
- **One Interface, Many Implementations**
- Polymorphism means "one interface, many possible implementations." Cars, for example, are designed with polymorphism in mind. They all use the same interface—an accelerator pedal—to change speed, even though the internal method may differ from model to model. The auto industry designs cars this way so that the drivers do not have to learn a new interface for each newmodel of car.

Elements of Object

Abstraction

- Dahl, Dijkstra, and Hoare suggest that “abstraction arises from a recognition of similarities between certain objects, situations, or processes in the real world, and the decision to concentrate upon these similarities and to ignore for the time being the differences” [42].
- Shaw defines an abstraction as “a simplified description, or specification, of a system that emphasizes some of the system’s details or properties while suppressing others. A good abstraction is one that emphasizes details that are significant to the reader or user and suppresses details that are, at least for the moment, immaterial or diversionary” [43].
- Berzins, Gray, and Naumann recommend that “a concept qualifies as an abstraction only if it can be described, understood, and analyzed independently of the mechanism that will eventually be used to realize it” [44].

Abstraction

- An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer.

- Abstraction is one of the fundamental ways that we as humans cope with complexity.

Encapsulation

- Encapsulation is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation.
- Abstraction and encapsulation are complementary concepts: Abstraction focuses on the observable behavior of an object, whereas encapsulation focuses on the implementation that gives rise to this behavior. Encapsulation is most often achieved through information hiding (not just data hiding), which is the process of hiding all the secrets of an object that do not contribute to its essential characteristics; typically, the structure of an object is hidden, as well as the implementation of its methods.

Modularity/Decomposition

- Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules.
- Modularization consists of dividing a program into modules which can be compiled separately, but which have connections with other modules.
- “The act of partitioning a program into individual components can reduce its complexity to some degree. . . . Although partitioning a program is helpful for this reason, a more powerful justification for partitioning a program is that it creates a number of well-defined, documented boundaries within the program. These boundaries, or interfaces, are invaluable in the comprehension of the program”

- Thus, the principles of abstraction, encapsulation, and modularity are synergistic. An object provides a crisp boundary around a single abstraction, and both encapsulation and modularity provide barriers around this abstraction.

Hierarchy

- Hierarchy is a ranking or ordering of abstractions.
- The two most important hierarchies in a complex system are its class structure the “is a” hierarchy) and its object structure (the “part of” hierarchy).