Object-Oriented Analysis and Design using JAVA (20B12CS334)

B.Tech (CSE/IT) 5th SEM
2021-2022

Principles of object-orientation

Principles of object-orientation

Object model The collection of principles that form the foundation of object-oriented design; a software engineering paradigm emphasizing the principles of abstraction, encapsulation, modularity, hierarchy

Object-oriented technology is built on a sound engineering foundation, whose elements we collectively call the *object model of development* or simply the *object model*. The object model encompasses the principles of

- abstraction,
- encapsulation,
- modularity,
- hierarchy,

Abstraction

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

“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”

-Dahl, Dijkstra

“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”

-Shaw

“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”

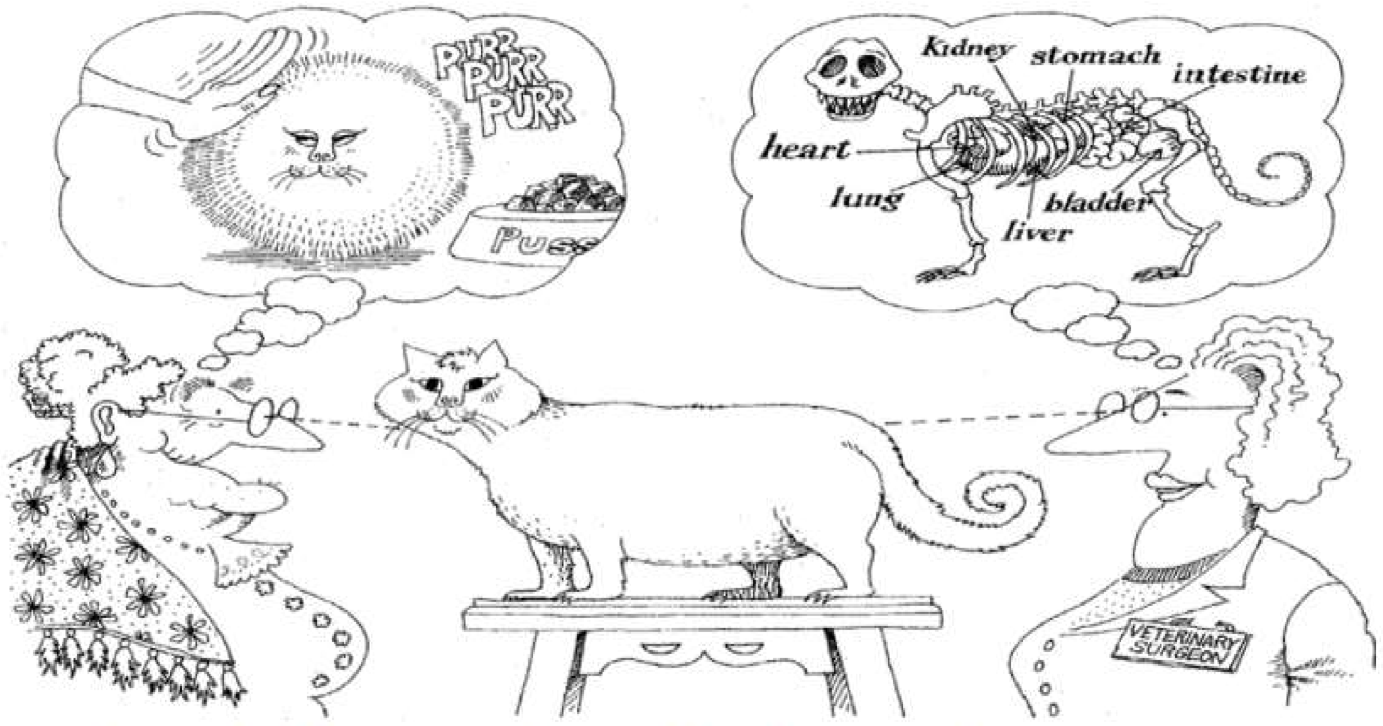
-Berzins, Gray, and Naumann

“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”

-Grady Booch

Referred sources mentioned in Key References

An abstraction focuses on the outside view of an object and so serves to separate an object's essential behavior from its implementation



Abstraction focuses on the essential characteristics of some object, relative to the perspective of the viewer.

Referred sources mentioned in Key References

Deciding on the right set of abstractions for a given domain is the central problem in object-oriented design. Because this topic is so important,

Kinds of abstraction

- **Entity abstraction** An object that represents a useful model of a problem domain or solution domain entity
- **Action abstraction** An object that provides a generalized set of operations, all of which perform the same kind of function
- **Virtual machine abstraction** An object that groups operations that are all used by some superior level of control, or operations that all use some junior-level set of operations
- **Coincidental abstraction** An object that packages a set of operations that have no relation to each other

Example of Abstraction

On a hydroponics farm, plants are grown in a nutrient solution, without sand, gravel, or other soils.

One must control diverse factors such as temperature, humidity, light, pH, and nutrient concentrations. On a large farm, it is not unusual to have an automated system that constantly monitors and adjusts these elements. (Automated gardener)

One of the key abstractions in this problem is that of a sensor. Actually, there are several different kinds of sensors. Anything that affects production must be measured, so we must have sensors for air and water temperature, humidity, light, pH, and nutrient concentrations, among other things.

What are the responsibilities of a temperature sensor? Our design decision is that a sensor is responsible for knowing the temperature at a given location and reporting that temperature when asked

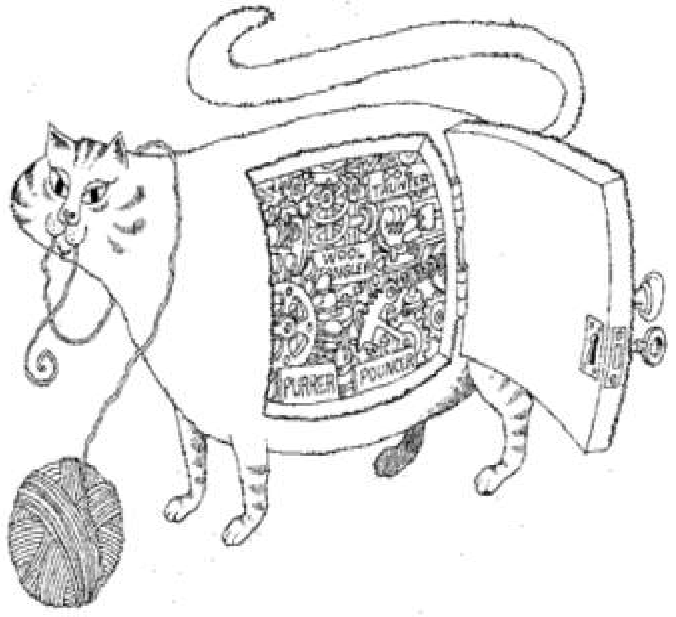
Abstraction: Temperature Sensor
Important Characteristics: temperature Location
Responsibilities: report current temperature calibrate

Encapsulation

- 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.

- “No part of a complex system should depend on the internal details of any other part”. Whereas abstraction “helps people to think about what they are doing,” encapsulation “allows program changes to be reliably made with limited effort”



Encapsulation hides the details of the implementation of an object

- “For abstraction to work, implementations must be encapsulated”.
- In practice, this means that each class must have two parts: an interface and an implementation.
- The interface of a class captures only its outside view, encompassing our abstraction of the behavior common to all instances of the class.
- The implementation of a class comprises the representation of the abstraction as well as the mechanisms that achieve the desired behavior.
- The interface of a class is the one place where we assert all of the assumptions that a client may make about any instances of the class; the implementation encapsulates details about which no client may make assumptions.

To summarize, we define encapsulation as follows:

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.

Modularity

"The act of partitioning a program into individual components can reduce its complexity to some degree.

In some languages, such as Smalltalk, there is no concept of a module, so the class forms the only physical unit of decomposition.

Java has packages that contain classes. In many other languages, including Object Pascal, C++, and Ada, the module is a separate language construct and therefore warrants a separate set of design decisions.

In these languages, classes and objects form the logical structure of a system; we place these abstractions in modules to produce the system's physical architecture.

Especially for larger applications, in which we may have many hundreds of classes, the use of modules is essential to help manage complexity

"Modularization consists of dividing a program into modules which can be compiled separately, but which have connections with other modules".

Modularity packages abstractions into discrete units

Referred sources mentioned in Key References

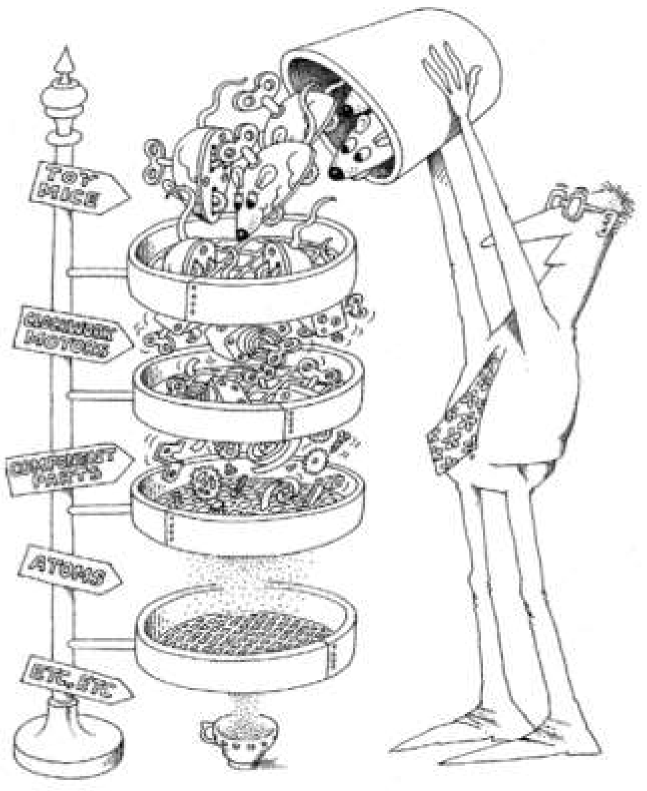
Hierarchy

- Abstraction is a good thing, but in all except the most trivial applications, we may find many more different abstractions than we can comprehend at one time.

- Encapsulation helps manage this complexity by hiding the inside view of our abstractions. Modularity helps also, by giving us a way to cluster logically related abstractions.

- Still, this is not enough. A set of abstractions often forms a hierarchy, and by identifying these hierarchies in our design, we greatly simplify our understanding of the problem.

- 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)



Abstractions form a hierarchy

Examples of Hierarchy

Inheritance is the most important "is a" hierarchy, and as we noted earlier, it is an essential element of object-oriented systems. Basically, inheritance defines a relationship among classes, wherein one class shares the structure or behavior defined in one or more classes (denoting single inheritance and multiple inheritance, respectively). Inheritance thus represents a hierarchy of abstractions, in which a subclass inherits from one or more super classes. Typically, a subclass augments or redefines the existing structure and behavior of its super classes

Semantically, inheritance denotes an "is a" relationship. For example, a bear "is a" kind of mammal, a house "is a" kind of tangible asset, and a quick sort "is a" particular kind of sorting algorithm. Inheritance thus implies a generalization/ specialization hierarchy, wherein a subclass specializes the more general structure or behavior of its superclasses. Indeed, this is the litmus test for inheritance: If `B` is not a kind of `A`, then `B` should not inherit from `A`.

- Single Inheritance
- Multiple Inheritance
- Aggregation

Key references

1. Object-Oriented Analysis and Design with Applications-Third Edition-Addition Wesley Authors-Grady Booch Robert A. Maksimchuk Michael W. Engle Bobbi J. Young, Ph.D. Jim Conallen Kelli A. Houston

Thank You