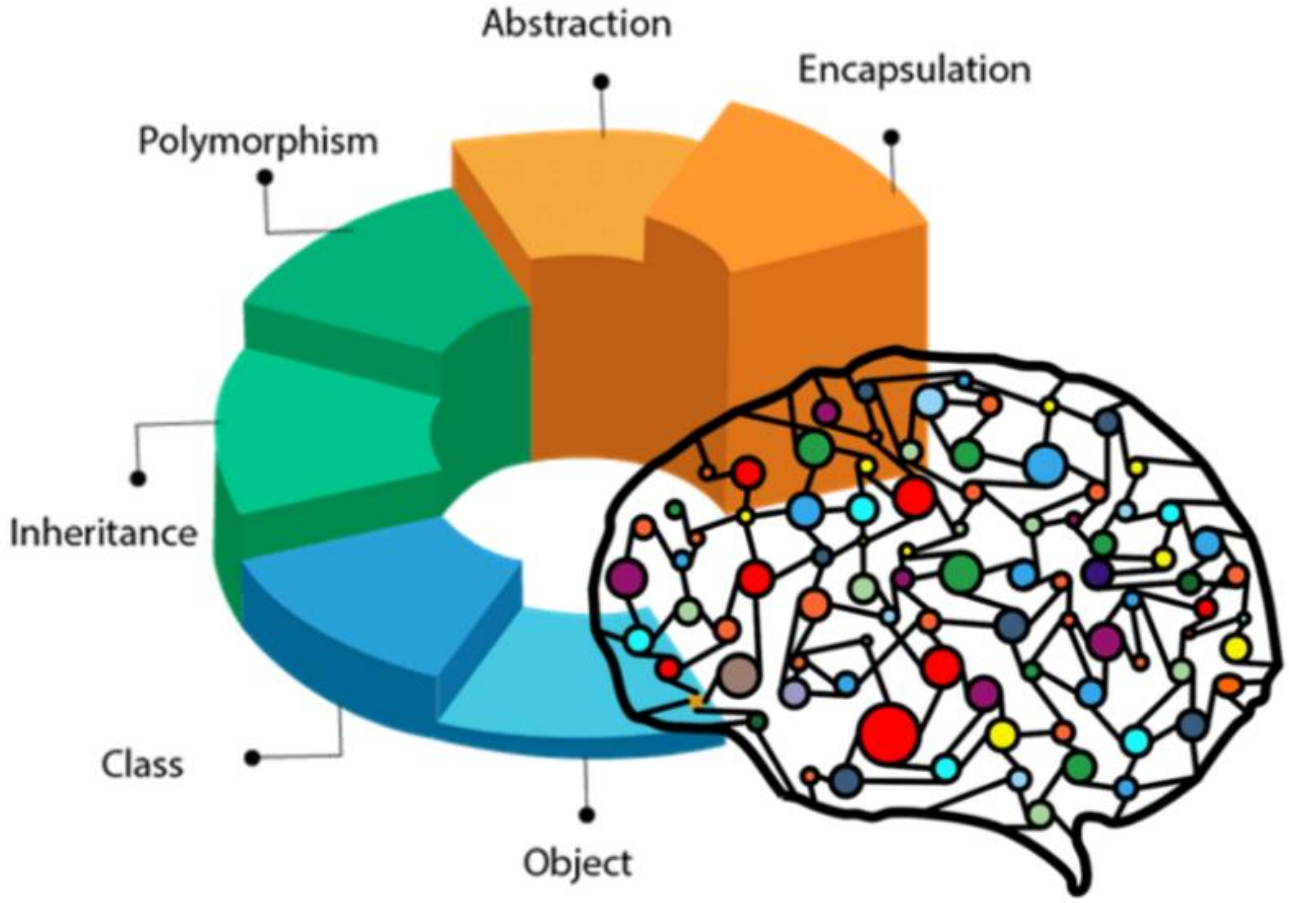
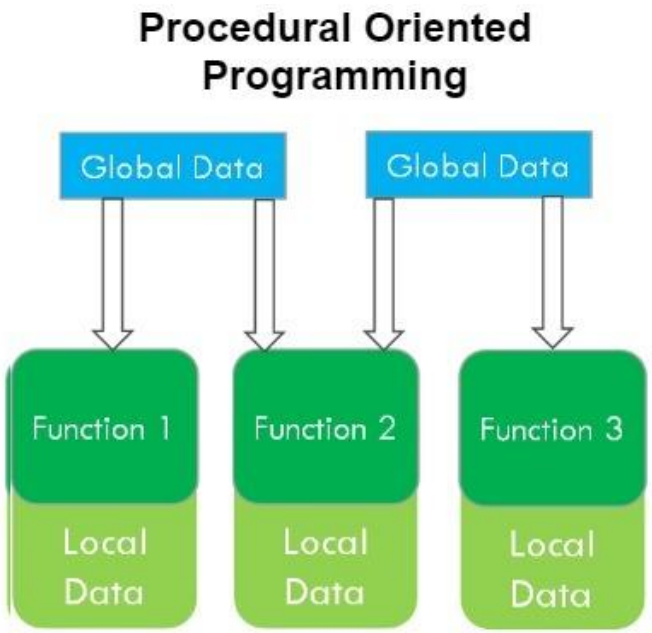
20IS603 Architecture of Intelligent Systems

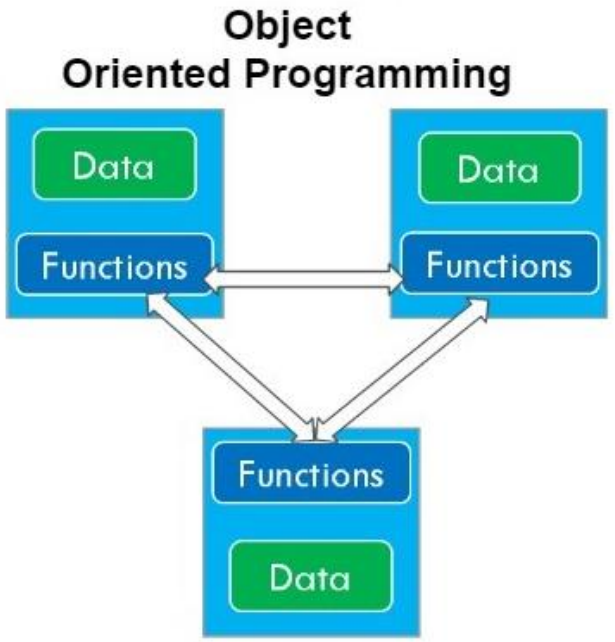
Object-oriented systems



Object Oriented Programming

- Imperative Programming Versus Declarative Programming





- Critical role in many knowledge-based systems, as it offers a way of representing the things that are being reasoned about, their properties and behaviors, and the relationships among them.

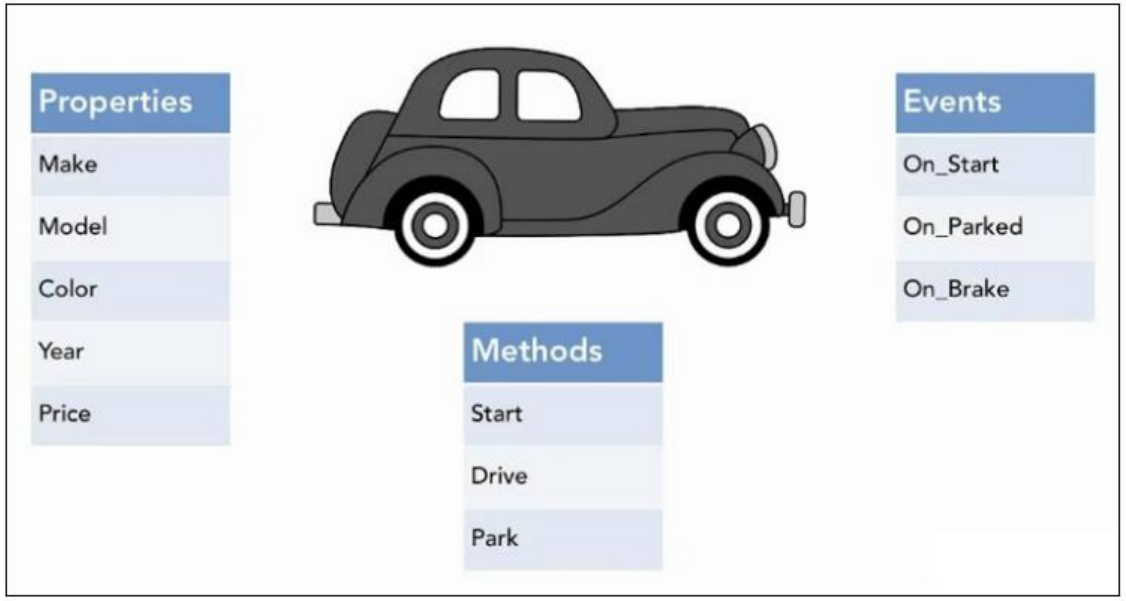
Object Oriented Programming

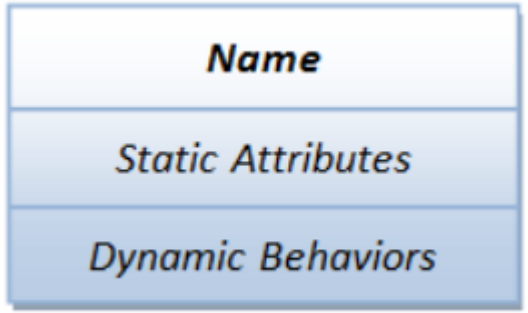
- Programming paradigm that relies on the concept of **classes** and **objects**.
- Solution to a problem is modelled as a collection of collaborating *objects*
- Fundamental building blocks
  - Classes
  - Objects
  - Methods
  - Attributes

Building blocks of an OOP

Classes

- A blueprint or template of entities (objects) that represent the same idea or concept
- Identical to the built-in types
- User-defined classes are called *abstract data types*
- Definition of a class contains the class name, its attributes, its operations and its relationships to other classes





Building blocks of an OOP (2)

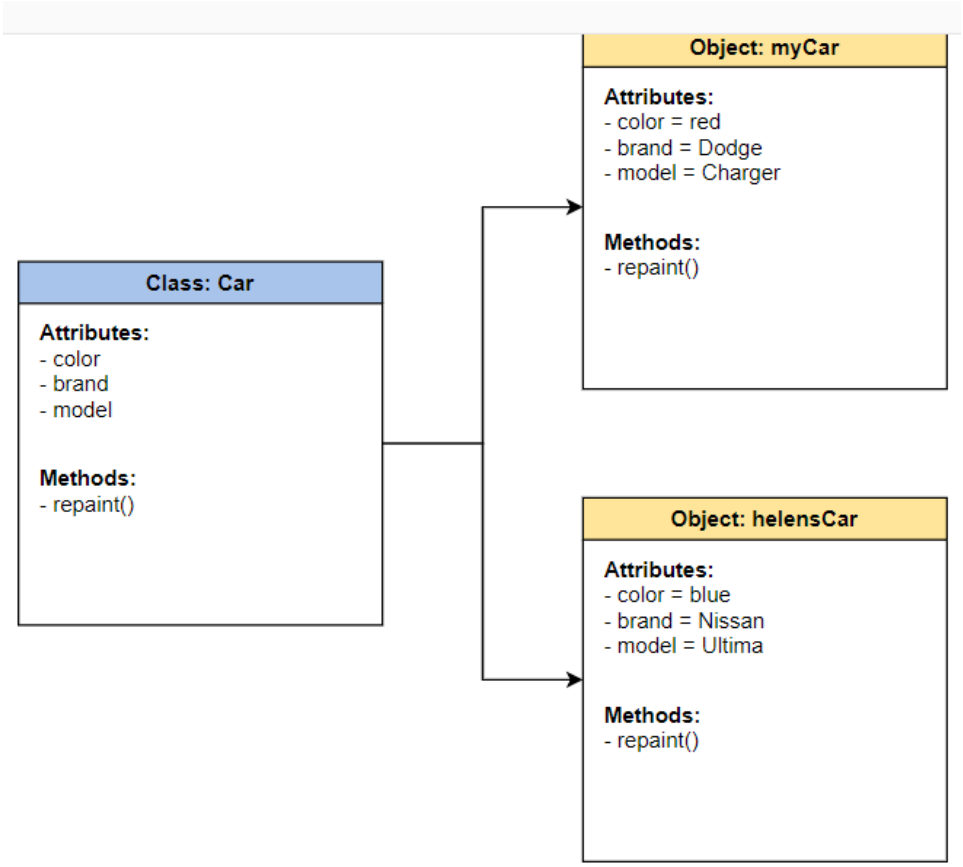
Instances

- Classes form templates for “the objects themselves,” - object *instances*.
- Once a class has been defined, instances of the class can be created that have the properties defined for the class.
- ‘My car’ is an instance of the class of car - specifies the characteristics and behavior of its instances.
- *Instantiate* was to make a real (data-filled) object from an abstract object

Building blocks of an OOP (3)

Objects

- Object means a real-world entity.
- Any entity that has state and behavior is an object – physical or logical
- Objects are instances of classes created with specific data
- Object encapsulates data and the functions that operate on that data - data is usually hidden from other objects



Building blocks of an OOP (4)

Attributes (or Data members)

- Quantities that describe instances of that class
- Example: *pulse* [amplitude, position, speed, and direction],
  car [number of wheels, doors, seating capacity]
- Only the names and the types of the attributes need to be declared within the class
- Attributes can be of any type - Amplitude and speed of type float, whereas position and direction of type vector
- Class attributes (or class variables) - same for all instances of that class
- Instance attributes (or instance variables) - belonging to only one object.

Building blocks of an OOP (5)

Methods or Operations or Member functions

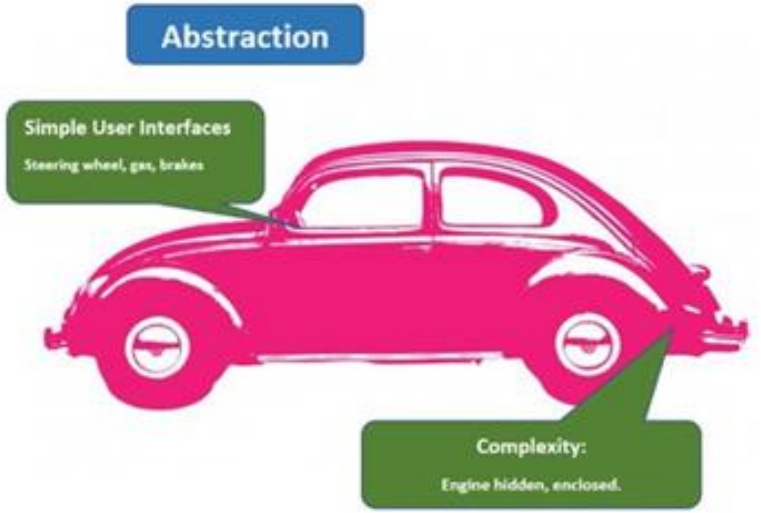
- A collection of statements that perform some specific task and return result
- Represent behaviors; perform actions; return information about an object, or update an object's data - Operations belonging to objects

Creation and Deletion of Instances

- Objects that are created must be explicitly destroyed when they are no longer needed - release of memory when no longer required.
- *Constructor* – create new instance - a member function whose name is identical to the class name, and used to set the initial values of some attributes.
- *Destructor* - used to release memory when an instance is deleted.

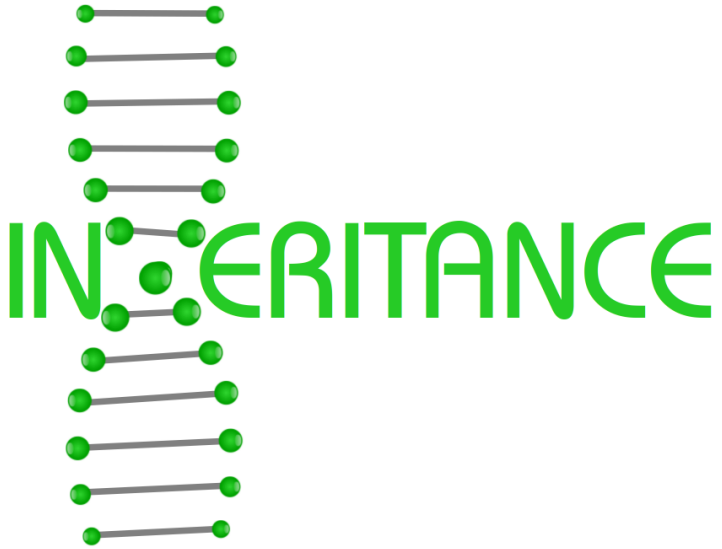
Data Abstraction

- Abstraction is method of hiding the implementation details and showing only the functionality.
- Data abstraction and Process abstraction
- When the object data is not visible to the outer world, it creates data abstraction. If needed, access to the Objects' data is provided
- Data Abstraction - the process of identifying only the required characteristics of an object ignoring the irrelevant details.
- The properties and behaviours of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects
- It increases the readability of the code as it eliminates the possibility of displaying the complex working of the code.



Inheritance

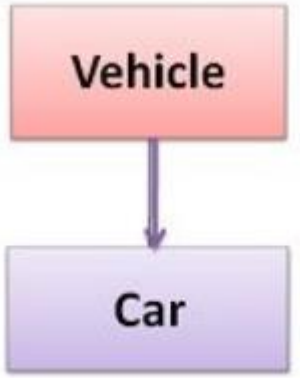
- When there are also many similarities between classes - common information had to be specified twice - problem can be avoided by the use of inheritance.
- Allows programmers to create classes that are built upon existing classes, and this enables a class created through inheritance to inherit the attributes and methods of the parent class.
- Different Forms of Inheritance
  - Single Inheritance
  - Multiple Inheritance
  - Multi-level Inheritance
  - Hierarchical Inheritance
  - Hybrid Inheritance



Single Inheritance

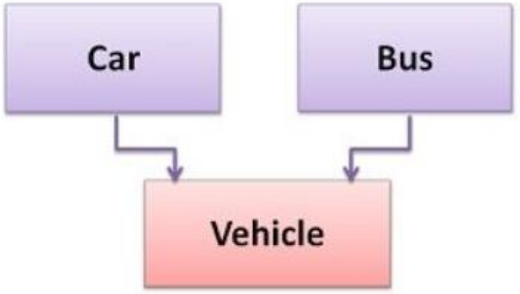
- A class inherits only one parent class
- The sub/super class relationship can be thought of as *is-a-kind-of*,
  - a subclass is-a-kind-of superclass;
  - an offspring is-a-kind-of parent;
  - a derived class is-a-kind-of base class;
  - a specialized class is-a-kind-of generalized class; and
  - a specialized class inherits from a generalized class.
- The is-a-kind-of relationship is transitive, that is,

```
if x is-a-kind-of y and y is-a-kind-of z, then x is-a-kind-of z
```

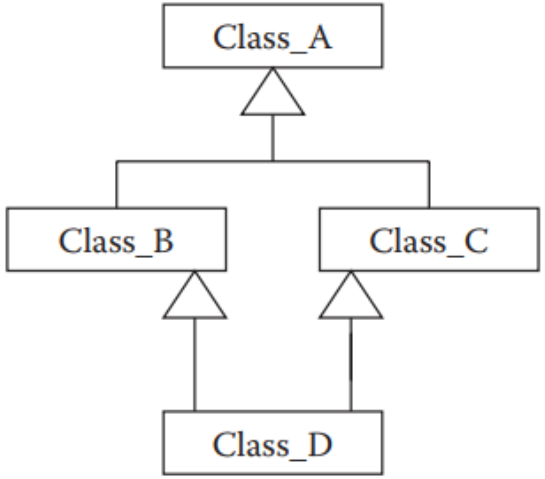


Multiple and Repeated Inheritance

- Multiple inheritance - An offspring inherits from more than one parent.

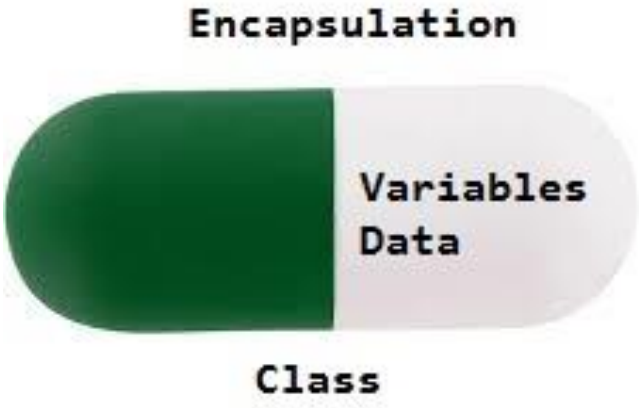


- One class may find itself indirectly inheriting from another via more than one route – repeated inheritance



Encapsulation

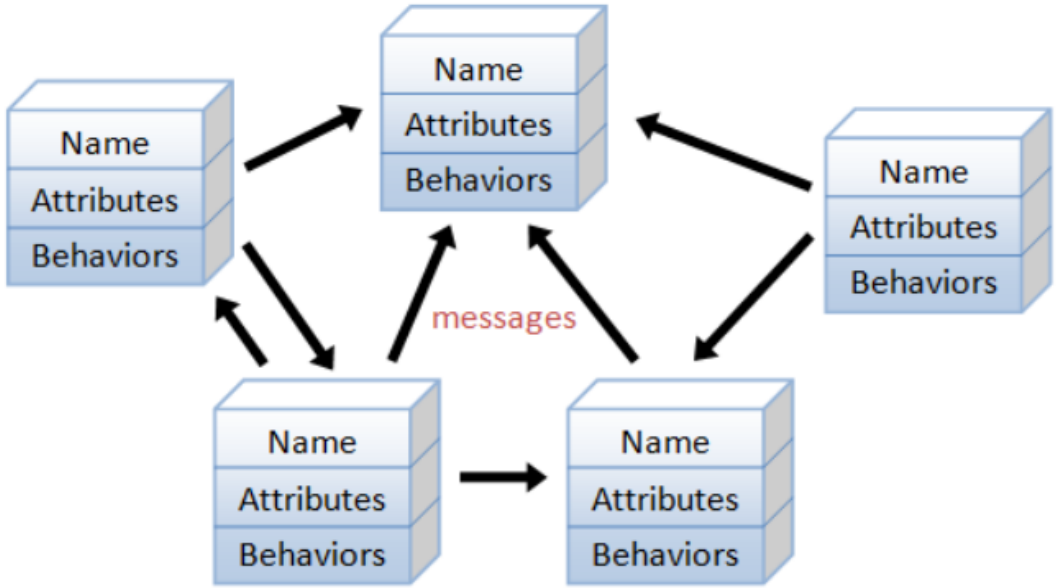
- Information-hiding
- Containing all important information inside an object, and only exposing selected information to the outside world.
- The methods and attributes are therefore private and are said to be encapsulated within the object.
- The interface to each object reveals as little as possible of the inner workings of the object.
- The object has control over its own data, and those data cannot be directly altered by other objects.



Encapsulation

Four access levels:

- Private: access from member functions of this class only (the default);
- Protected: access from member functions of this class and of derived classes;
- Public: access from any part of the program; and
- Friend: access from member functions of nominated classes.



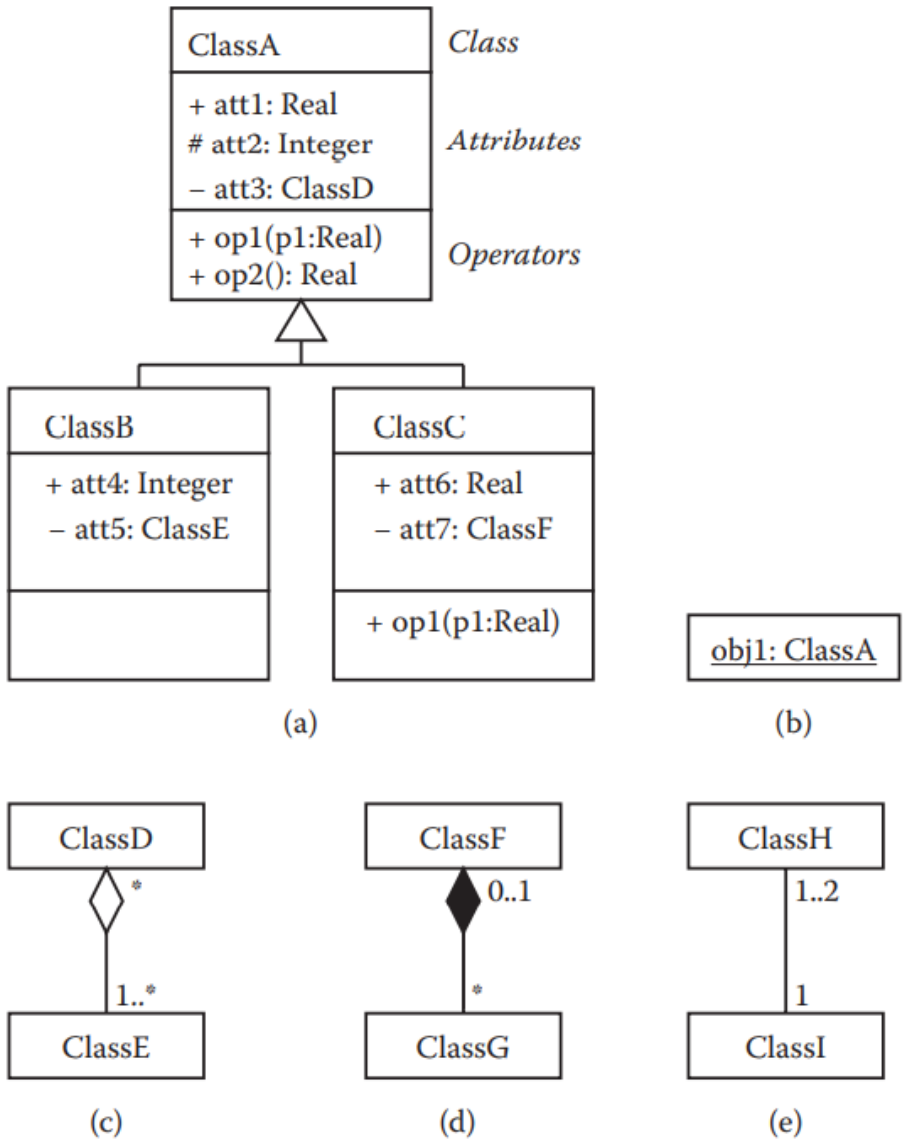
Unified Modeling Language (UML)

- Provides a means of specifying the relationships between classes and instances, and representing them diagrammatically.
- Specialization describes the is-a-kind-of relationship between a subclass and superclass, and involves the inheritance of common information.
- Instantiation describes the relation between a class and an instance of that class.
- An instance of a class and a subclass of a class are both said to be clients of that class, since they both derive information from it, but in different ways
- Three other types of relationships:
  - Aggregation
  - Composition
  - Association

Unified Modeling Language (UML)

- An *aggregation* relationship exists when an object can be viewed as comprising several subobjects.
  - Software library can be seen as an aggregation of many component modules. The degree of ownership of the modules by the software library is rather weak, in the sense that the same module could also belong to another software library.
- The *composition* relationship has a strong degree of ownership of the component parts.
  - For instance, a car comprises an engine, chassis, doors, seats, wheels, etc. – this can be recognized as a composition relation since duplication or deletion of a car would require duplication or deletion of these component objects.
- *Associations* are loose relationships between objects.
  - For instance, when one object makes use of another by sending messages to it.
  - The senders of messages are termed *actors*, and the recipients are *servers*. Objects that both send and receive messages are sometimes termed *agents*

Unified Modeling Language (UML)



Class relationships in UML. (a) ClassA is a generalization of ClassB and ClassC; (b) obj1 is an instance of ClassA; (c) aggregation; (d) composition; (e) association. Numbers represent permissible number of objects; * indicates any number. Access controls: + indicates public, # indicates protected, and – indicates private.

Dynamic (or Late) Binding

- Although the parent classes of objects may be known at compilation time, the actual (derived) classes may not be.
- The actual class is not bound to an object name at compilation time, but instead the binding is postponed until run-time - known as dynamic binding
- Dynamic Binding is the type of binding that collects all required information to call a function during run time.
- The combined effect of inheritance and dynamic binding is that the same function call can have more than one meaning, and the actual meaning is not interpreted until run-time - this effect is known as polymorphism.
- The code that uses monomorphism is much longer and is likely to include considerable duplication.
- Polymorphism avoids duplication and allows the commonality between classes to be made explicit.