Lecture: Properties of OOP

2

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which are data structures that contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods.

1. Objects - structures that contain both data and procedures
2. Classes - definitions for the data format and available procedures for a given type or class of object; may also contain data and procedures (known as class methods) themselves.

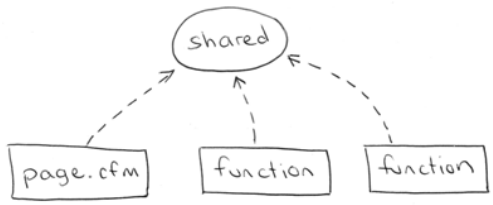
Object-oriented programming (OOP) is a programming language model organized around objects rather than "actions" and data rather than logic.

OOP is a design philosophy. It stands for Object Oriented Programming. Object-Oriented Programming (OOP) uses a different set of programming languages than old procedural programming languages (C, Pascal, etc.). Everything in OOP is grouped as self-sustainable "objects". Hence, you gain reusability by means of four main object-oriented programming concepts.

In procedural programing our code is organized into small "procedures" that use and change our data. The key idea here is that our functions have no intrinsic relationship with the data they operate on. As long as you provide the correct number and type of arguments, the function will do its work and faithfully return its output.

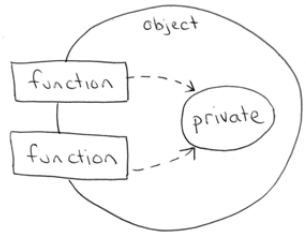
Sometimes our functions need to access data that is not provided as a parameter, i.e., we need access data that is outside the function. Data accessed in this way is considered "global" or "shared" data.

So in a procedural system our functions use data they are "given" (as parameters) but also directly access any shared data they need.



In object oriented programming, the data and related functions are bundled together into an "object". Ideally, the data inside an object can only be manipulated by calling the object's functions. This means that your data is locked away inside your objects and your functions provide the only means of doing

something with that data. In a well-designed object oriented system objects never access shared or global data, they are only permitted to use the data they have, or data they are given.



1.1 Object vs. Class

1.1.1 Object

An object is a software construct that encapsulates state and behavior. Objects allow you to model your software in real-world terms and abstractions. Strictly speaking, an object is an instance of a class.

An object is a thing, an entity, a noun, anything you can imagine that has its own identity. Some objects are living, some aren't. Examples from the real world include a car, a person, a house, a table, a dog, a pot plant, a check book or a raincoat.

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.

Objects encapsulate both state and behavior. In particular, they consist of OBJECT a collection of instance variables, representing the state of the object, and a INSTANCE VARIABLE collection of methods, representing the behavior that the object is capable of METHOD performing. We sometimes refer to instance variables as the fields of an object. The methods are routines that are capable of accessing and manipulating the values of the instance variables of the object. When a message is sent to an MESSAGE object, the corresponding method of the object is executed. (In C++, instance variables are referred to as member fields or variables and methods as member functions.)

1.1.1.1 Attributes

Attributes are the outwardly visible characteristics of a class. Eye color and hair color are example of attributes.

1.1.1.2 Behavior

A behavior is an action taken by an object when passed a message or in response to a state change: It's something that an object does. One object can exercise another object's behavior by performing an operation on that object. You might see the terms method call, function call, or pass a message used in place of performing an operation.

1.1.2 Class

Just like objects in the real world, the OOP world groups objects by their common behaviors and attributes. A class defines all of those characteristics common to a type of object. Specifically, the class defines all of those attributes and behaviors exposed by the object. A class defines the common attributes and behaviors shared by a type of object. Objects of a certain type or classification share the same behaviors and attributes.

1.2 Benefits of OOP over procedural language

1. Better code organization
2. Reusable
3. Extensible
4. Natural
5. Reliable
6. Maintainable
7. Extendable

1.3 Properties of OOP

1.3.1 Abstraction

Abstraction layer

An abstraction layer (or abstraction level, or a layer of abstraction) is a way of hiding the implementation details of a particular set of functionality.

Reduce complexity

The first, and most important, has to do with management of complexity. In this sense, abstraction is a primary concept in all engineering disciplines and is, in fact, a basic property of how people approach the world. We simply can't cope with the full complexity of what goes on around us, so we have to find models or approximations that capture the salient features we need to address at a given time, and gloss over issues not of immediate concern.

1.3.2 Encapsulation

The property of encapsulation is the property of information hiding. Encapsulation typically refers to the **hiding of data** and of the implementation of an object. Data and code, when encapsulated, are hidden from external view. When an external observer views an encapsulated object, only the exterior interface is visible; the internal details are invisible and cannot be accessed. Thus, data which is encapsulated cannot directly be manipulated and, in particular, cannot be directly updated.

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. Some programming languages (for example, Smalltalk) automatically make attributes private and some languages (for example, Java) leave it to the programmer.

Another way to think of encapsulation is to imagine that objects are courteous to one another. If you wanted to borrow some money from a colleague to buy food in the staff canteen, you wouldn't grab their wallet and look through it to see if they had enough cash. Instead, you would ask them whether they could lend you some money and they would look in their own wallet.

Encapsulation is a term that is found in Object-Oriented paradigm and refers to keeping the data in private fields and modify it only through methods.
Encapsulation is the technique of making the fields in a class private and providing access to the fields via public methods. If a field is declared private, it cannot be accessed by anyone outside the class, thereby hiding the fields within the class. For this reason, encapsulation is also referred to as data hiding

Thus **encapsulation may be seen as a way of achieving data hiding in object-oriented systems**.

1.3.3 Abstraction vs. Encapsulation

Abstraction - hiding implementation.
Encapsulation - hiding data.

Abstraction focuses on the outside view of an object (i.e. the interface)

Encapsulation (information hiding) prevents clients from seeing its inside view.

Abstraction means hiding the internal details and just exposing the functionality. For Example:-When you change the gear of your car, you know the gears will be changed without knowing how they are functioning internally. Abstraction focuses on the outside view of an object (i.e. the interface).

Encapsulation means put the data and the function that operate on that data in a single unit (information hiding). Encapsulation prevents clients from seeing it's inside view, where the behavior of the abstraction is implemented.

1.3.4 Polymorphism.

The word "polymorphism" literally means "having many forms". In programming languages, polymorphism is most often taken to be that property of procedures by which they can accept and/or return values of more than one type. For example, a procedure which takes a single argument is said to be polymorphic if it can accept actual parameters of more than one type.

1.3.5 Inheritance

Inheritance allows us to specify that a class gets some of its characteristics from a parent class and then adds unique features of its own – this leads to the description of whole families of objects. Inheritance allows us to group classes into more and more general concepts, so that we can reason about larger chunks of the world that we live in.

A subclass inherits all of the fields, messages, methods (and assertions) of its superclass. For example, if we wanted to model land vehicles, we might come up with the hierarchy shown in Figure