OOP concepts

Tuesday, April 6, 2021 3:34 PM

What is Object Oriented Programming

- Object Oriented programming (OOP) is a programming paradigm that relies on the concept of **classes** and **objects**.
- It is used to structure a software program into simple, reusable pieces of code blueprints (usually called classes), which are used to create individual instances of objects.
- A **class** is an abstract blueprint used to create more specific, concrete objects
- Classes often represent broad categories, like `Car` or `Dog` that share **attributes**
- These classes define what attributes an instance of this type will have, like `color`, but not the value of those attributes for a specific object.
- Classes can also contain functions, called **methods** available only to objects of that type. These functions are defined within the class and perform some action helpful to that specific type of object.

Benefits of OOP

- OOP models complex things as reproducible, simple structures
- Reusable, OOP objects can be used across programs
- Allows for class-specific behavior through polymorphism
- Easier to debug, classes often contain all applicable information to them
- Secure, protects information through encapsulation

How to structure OOP programs

1. Create a parent class
2. Create a child class
3. Add unique attributes and behaviors
4. Create objects from the child class

Building blocks of OOP

1. **Classes** : are essentially user defined data types, where we create a blueprint for the structure of methods and attrivbutes
2. **Objects**: are instances of classes created with specific data
3. **Attributes** : are the information that is stored, defined in the class template, when objects are instaniated individual objects contain data stored in the Attributes field
4. **Methods**: represent behaviors, perform actions, retrun information about an object or update an object's data. Defined in the class definition

Principles of OOP

There are four principles of object - oriented programming are: encapsulation, abstraction, inheritence, and polymorphism

Encapsulation

- Encapsulation is achieved when each object keeps its state **private**, inside a class. Other objects don't have direct access to this state. Instead, they can only call a list of public functions — called methods.
- Object manages its own state via methods - no other class can touch it unless explicity allowed
- You just can use the methods provided to communicate with the object
- By default, you can't change the state

Example : The Cat class

○ **private variables** mood, hungry and energy. ○ private method meow()	It can be happy, depressed, excited, hungry, energetic anytime, you never know and it can meow when ver it wants, other classes can't make it meow, or be happy,…
public methods sleep(), play() and feed()	each of them modifies the internal state, and can invoke meow()

Abstraction

- Abstraction can be thought of as a natural extension of encapsulation.
- Maintaining a large codebase like this for years — with changes along the way — is difficult while in OO design, program are often extremely large -> Abstraction can ease this problem
- Applying abstraction means that each object should **only** expose a high-level mechanism for using it.
- This mechanism should hide internal implementation details. It should only reveal operations relevant for the other objects.

Example: The coffee machine

- does a lot of stuff and makes quirky noises under the hood.
- all you have to do is put in coffee and press a button.
- this mechanism should be easy to use and should rarely change over time

➢ Think of it as a small set of public methods which any other class can call without "knowing" how they work.

Inheritance

- Objects are often very similar. They share common logic. But they're not **entirely** the same.
- If we want to reuse the common logic and extract the unique logic into a seprate class, we will use inheritance
- It means that you create a (child) class by deriving from another (parent) class. This way, we form a hierarchy.

Example: Teacher and student system

If our program needs to manage public and private teachers, but also other types of people like students, we can implement this class hierarchy.
This way, each class adds only what is necessary for it while reusing common logic with the parent classes.

Polymorphism

- Polymorphism means "many shapes" in Greek.

? Say we have a parent class and a few child classes which inherit from it. Sometimes we want to use a collection — for example a list — which contains a mix of all these classes. Or we have a method implemented for the parent class — but we'd like to use it for the children, too.

- polymorphism gives a way to use a class exactly like its parent so there's no confusion with mixing types
- But each child class keeps its own methods as they are.
- This typically happens by defining a (parent) interface to be reused. It outlines a bunch of common methods. Then, each child class implements its own version of these methods.

Example: The shape of circle,rectangle and triangles

- Having these three figures inheriting the parent Figure Interface lets you create a list of mixed triangles, circles, and rectangles
- if this list attempts to calculate the surface for an element, the correct method is found and executed. If the element is a triangle, triangle's CalculateSurface() is called. If it's a circle — then circle's CalculateSurface() is called
- You can define it once and accept a Figure as an argument. Whether you pass a triangle, circle or a rectangle — as long as they implement CalculateParamter()