* 1. What Is Object-Oriented Development?

In traditional software development, the focus is on the process of how the software works, but in object-oriented development, the focus is on the objects and how they interact with each other. This approach allows developers to create software that is more flexible, reusable, and easier to maintain. It also allows for better collaboration among developers and stakeholders, as they can discuss the objects and their interactions in a more concrete and visual way.

The "process-centred" approach to software development involved breaking down a program into smaller, manageable parts called functional modules. 🡪 this approach had a short lifespan because if the program's requirements changed, the entire program would need to be rewritten. This made it difficult to reuse existing code, and software designers began to look for new ways to design software

Key Concepts of Object-Oriented Design

During the development of this way of thinking, many ideas and approaches were tried and rejected. Over time, the field has become more stable and we can now present the main ideas that have proven to be useful and effective.

1-The Central Role of Objects

In software design, object-orientation focuses on using objects to solve problems. In the past, software design was based on processes, which were prone to change and made it hard to reuse old code. An object is a piece of data and the methods that can change it.

2-The Notion of a Class

Why use class using classes in OOP helps us write more organized, efficient, and maintainable code, making it easier to create complex software systems.

class is a way of grouping objects together based on their common characteristics. Just like how animals can be grouped into different species, objects in software design can be grouped into different classes. This allows us to organize and make sense of the many objects in a software system.

3-abstract specification

 the abstract specification is a high-level plan for the system that outlines what the system should do, without specifying how it should do it

4-The Notions of Extendability and Adaptability

**Extendability** means that software (like computer programs) can be easily made bigger or changed. It's like adding new parts to a toy without breaking it. Unlike physical things (like toys), software can be adjusted and expanded more easily.

**Adaptability** means that software can change and adjust to different situations. It's like how you can use the same toy in different ways for different games.

1.3Other Related Concepts

**Object-Oriented Programming** is the way of actually writing computer code using the concepts of objects, properties, and behaviors.

**Object-Oriented Concepts** are the basic ideas that guide us in how to use objects, classes, inheritance, and other things to make our code well-organized and efficient.

In a way, Object-Oriented Programming is like using the cooking recipe, and Object-Oriented Concepts are like the ingredients that make the recipe work.

2.3 Modular Design

We break down a large system into smaller, independent modules. Each module has its own specific job,

2.3 Encapsulation

Encapsulation is like a secret ingredient that makes modular design even better. It's all about hiding the implementation details of a module from the outside world.

The benefits of encapsulation are huge. For one, it helps prevent mistakes. If a module's implementation details are hidden, other modules can't accidentally mess with them. It also makes it easier to change the implementation without affecting the rest of the system.

3.3 Cohesion and Coupling

cohesion of a module tells us how well the entities within a module work together to provide this functionality. Cohesion is a measure of how focused the responsibilities of a module are. If the responsibilities of a module are unrelated or varied and use different sets of data, cohesion is reduced. Highly cohesive modules tend to be more reliable, reusable, and understandable than less cohesive ones.

Coupling refers to how dependent modules are on each other.

The very fact that we split a program into multiple modules introduces some coupling into the system.

Coupling could result because of several factors: a module may refer to variables defined in another module or a module may call methods of another module and use the return values.

CH-2

Implement class and object and explain

Summary:

In object-oriented programming, an attribute is like a property that describes an object. It holds a value that's needed for processing. In this programming approach, we use classes to create objects. A class is like a blueprint that can be used to create multiple objects with similar properties. For example, in a university system, there are different types of objects like students, instructors, staff members, and courses. Before creating these objects, we first create classes that define how these objects should be structured.

Explanation:

In simpler words, think of an attribute as something that helps describe an object. Just like a person has attributes like age, height, and name, objects have attributes too. For example, a "student" object might have attributes like their name, age, and student ID.

Now, classes are like templates or blueprints. They tell the computer how to create objects. Just as you follow a recipe to bake cookies, you use a class to create objects. In the provided example of a university system, there are different types of objects: students, instructors, staff members, and courses. Before creating these specific objects, we first define the classes.

The code 🡪 <https://www.ideone.com/FBLRZ5>

**Constructors for Object Initialization:**

Constructors in object-oriented programming are like special methods that are automatically called when an object is created. They are useful for setting up an object with initial values right from the start.

For example, if you're making a cake, you start by adding the basic ingredients like flour, sugar, and eggs. Similarly, when you create an object, a constructor can be used to add those essential initial values to it.

In simpler terms, constructors are like a way to make sure your objects start off with the right information. Just like you wouldn't want to eat a cake without its main ingredients, you wouldn't want to use an object without its essential values.

So, when you're creating objects and you already know some initial values they should have, that's when you should use constructors. It helps you make your code more organized and your objects ready for action right from the beginning.

Static Members

static member is a member of a class that is shared by all instances of the class. This means that all instances of the class share the same memory location for the static member, and any changes made to the static member will affect all instances of the class.

The code -> <https://www.ideone.com/p9fgsk>

Interfaces

help programmers design classes by giving them a clear set of rules to follow. They're like a common language that both humans and the program can understand to make sure everything works smoothly.

They tell the program how an object should look and what it can do, If you inherit from an interface, you must implement all the functions defined in the interface class

In C++, there is no native "interface" keyword like in some other programming languages (e.g., Java or C#). Instead, you can create an interface-like structure using abstract classes with pure virtual functions. Here's an example:

Code 🡪 <https://www.ideone.com/ji1LcQ>

Abstract Classes

we know that every shape has an area, we have defined the double field area and the method getArea() to return the area of the shape. We require that there be a method to compute the area of a shape, so we have written the method getArea(). But since the formula to compute the area is different for the three possible shapes, we have left out the implementation and declared the method itself as abstract.

Code -> <https://www.ideone.com/H0OCmv>

What the different between interface and Abstract Classes

Abstract Classes:

1. They often contain a mix of both concrete (implemented) methods and pure virtual methods (methods without an implementation). A class becomes abstract if it has at least one pure virtual method.
2. Abstract classes can have data members, concrete methods with implementation, and pure virtual methods.

Interface class

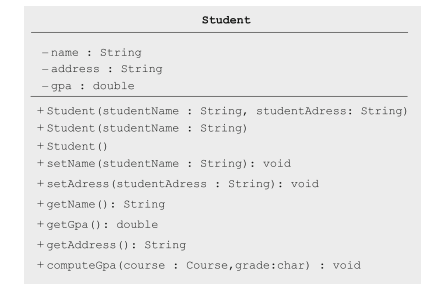
1. interfaces are like contracts. They declare a set of pure virtual methods that any class implementing the interface must provide
2. Classes implement interfaces by providing definitions for all the pure virtual methods declared in the interface.

Comparing Objects for Equality

We have seen the need to use the equals method to compare two objects.

Code -> <https://www.ideone.com/cYpUzh>

Class Diagrams



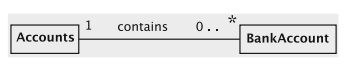
is an example of a class diagram. Each class is represented by a box, which is divided into three rectangles. The name of the class is given in the top rectangle. The attributes are shown with their names and their types in the second box. The third box shows the methods with their return types and parameters (names and types). The access specifier for each field and method is given just in front of the field name or method name. A − sign indicates private access, + stands for public access.

Sequence Diagrams

* we need to decide what each class is responsible
* One of the major goals of design is to determine the classes and their responsibilities

Association

"Association" is like a way of showing that the "Accounts" group can have one or more connections to different "BankAccount" objects, and each "BankAccount" can be linked to zero or more "Accounts" groups. It's a way of representing how things are connected or related in a clear and organized manner.



Interfaces Their and Implementation

in the context of UML (Unified Modeling Language) diagrams refer to a way of representing how a class in a computer program follows a set of rules defined by an interface.

In a diagram, you'll see two boxes: one for the interface and another for the class that implements it.

CH-3

In the previous chapter we studied classes and objects as the two building blocks of object-oriented systems. The structure of a software system is defined by the way in which these building blocks relate with one another and the behaviour of the system is defined by the manner in which the objects interact with one another. Therefore, in order to construct a software system, we need mechanisms that create connections between these building blocks. In this chapter we introduce the basic types of relationships between classes (and objects) that make the connections.

Association

which simply indicates that the objects of the two classes are related in some non-hierarchical way.

Code -> <https://www.ideone.com/A1gr6o>

Inheritance

Classes connected by inheritance share some commonalities and therefore, this kind of relationship is more restrictive than association.

Genericity

(templates) is a way to create classes or containers that can hold different types of things.

flexible container or box that can hold different types of things. Imagine you have a magic container that can hold anything - toys, books, or even snacks.

Explain Association

An association is like a connection between two or more groups of things. It shows that one group of things is using another group of things. Imagine it as a line drawn between two pictures that represent these groups.

For example, we have a class called "Student" that keeps track of information about the courses a student is taking. This information is shown in a picture. In our example, when a student needs information about a course, like when they want their grades or need to pay for it, they use the course information. This connection between a student and a course is what we call an association.

Characteristics of Associations

* associations represent very general relationships , they allow for variation in the nature of the connection.
* how many objects of class A can be linked to one object of class B
* An association normally represents something that will be stored as part of the data
* usually results in some information being added to the system since it adds a path connecting two objects

Arity of Relationships

1. one–one
2. one–many
3. many–many