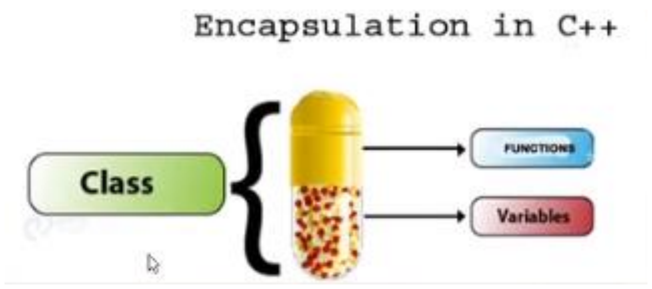
OBJECT ORIENTED PROGRAMMING USING C++

PARADIMS:

1 Encapsulation: The concept behind realizing an object as a single unit, consisting of variables and member functions is called Encapsulation.

- A mechanism that associates the code(function) and the data(variables) that it manipulates into a single unit is called an object.
- It is the mechanism to keep the data and code safe from external access.
- The construction that is used in c++ to support encapsulation is class.

EX:



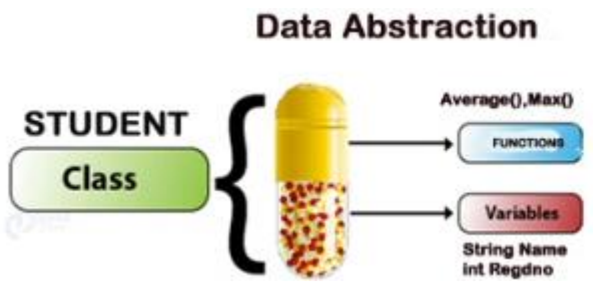
Encapsulation is the bundling of data and methods that operate on that data within a single unit (class). It restricts direct access to some of an object's components, which is why it's often referred to as **data hiding**.

- **Public Interface**: Methods that access or modify the data are provided to the user. **Private Data**: The data itself is hidden and only accessible through the public interface (e.g., methods).

2 Data abstraction: The process of understanding a real world concept as a combination of attributes (variables) and behaviour (Member functions) is called Data abstraction

- The art of creating user defined data types for a given problem.
- Class name itself is a user defined data
- They are also called abstract data types.

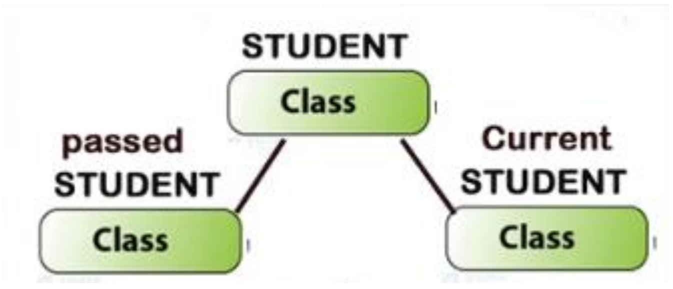
EX:



Data abstraction is the concept of hiding the complex implementation details and showing only the essential features of an object. It is achieved through **abstract classes** or **interfaces**.

3 Inheritance & Extensibility:

- It allows code extension and code reuse.
- The real world concept of class and subclass is atimulated.
- Derived class and base class concepts.



EX:

Inheritance allows one class to inherit the attributes and methods from another class. This promotes **code reusability** and **extensibility**—the ability to add new features to existing classes without modifying their code.

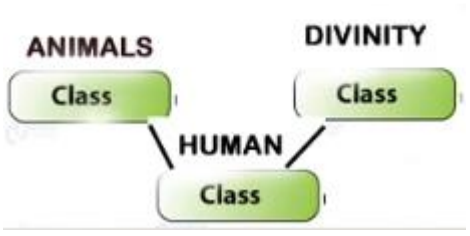
A SOFTWARE USED BY A COMPANY ALLOWS TWO TYPES OF USERS A) ADMIN B) END USER.

Which of the following concepts in C++ is best suited for implementing the above scenario is

s (in heritance)

4 Multiple inheritance:

- One derived class having multiple base classes



EX:

Multiple inheritance allows a class to inherit from more than one parent class. This allows a class to combine functionalities from multiple sources, though it can sometimes create complexity if the parent classes have conflicting methods.

5 Polymorphism:

Vendor calculates the bidding amount using a function **BA().** But the calculations made for the function **BA()** takes three different depending upon the client

<u>The concept In C++ close to the above description is:</u> **POLYMORPHISM**

- Single name/Operator having multiple Roles.

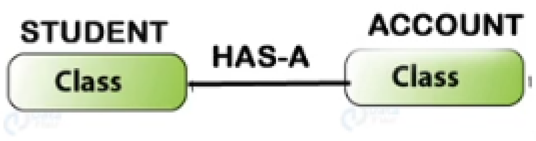
- Implemented in C++ using function Overloading, Operator Overloading and Virtual functions.

EX:

Polymorphism allows different classes to be treated as instances of the same class through inheritance. It allows methods to have different implementations based on the object type.

6 Delegation:

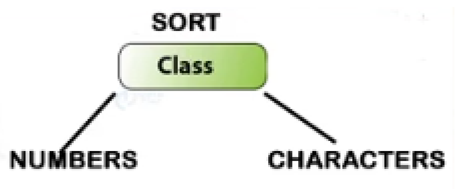
- Objects of one class being the member of another class.
- Implementing in c++ using features like Composition and aggregation forming new relations.
  EX:



Delegation is a design pattern where an object passes (delegates) a task to another object rather than implementing the functionality directly. The delegating object doesn't perform the task but relies on the delegate object to do so. This promotes **flexibility** and **separation of concerns**.

7 Genericity:

- Components having more than one interpretation depending upon their usage.
- Implementing in C++ using Functions templates and class templates.



EX:

Genericity (or Generics) allows classes, methods, and functions to operate on objects of **any type** while providing compile-time type safety. It enables **code reuse** by writing flexible, type-agnostic algorithms or data structures that work for different data types.

,-Persistence ,-Message passage