* **Abstraction** is one of the fundamental concepts of Object-Oriented Programming (OOP). It refers to the process of hiding the implementation details and showing only the essential features or functionalities to the user. In C++, this is achieved primarily using **abstract classes** and **interfaces**.  
  An **abstract class** is a class that cannot be instantiated on its own and typically contains one or more **pure virtual functions** (functions that do not have any implementation in the base class but must be implemented in derived classes).  
  **Key Points:**
  + **Purpose:** Abstraction simplifies complex systems by breaking them down into smaller, manageable components, exposing only what is necessary for interaction.
  + **How it works in C++:** Through abstract classes and pure virtual functions.
* **Abstraction**   
  Abstraction means providing only some of the information to the user by hiding its internal implementation details. We just need to know about the methods of the objects that we need to call and the input parameters needed to trigger a specific operation, excluding the details of implementation and type of action performed to get the result.  
  Abstraction is selecting data from a larger pool to show only relevant details of the object to the user. It helps in reducing programming complexity and efforts. It is one of the most important concepts of OOPs.  
     
  Real-life example: When you send an email to someone, you just click send, and you get the success message; what happens when you click send, how data is transmitted over the network to the recipient is hidden from you (because it is irrelevant to you).  
     
  We can implement Abstraction in C++ using classes. The class helps us to group data members and member functions using available access specifiers. A Class can decide which data members will be visible to the outside world and not. Access specifiers are the main pillar of implementing abstraction in C++. We can use access specifiers to enforce restrictions on class members.  
     
  **Code Example:**   
     
  cpp  
  Copy code  
  #include <iostream>  
  using namespace std;  
  // Abstract Class  
  class Shape {  
  public:  
   // Pure Virtual Function  
   virtual void draw() = 0; // Makes Shape an abstract class  
  };  
  // Derived Class - Circle  
  class Circle : public Shape {  
  public:  
   void draw() override {  
   cout << "Drawing Circle" << endl;  
   }  
  };  
  // Derived Class - Rectangle  
  class Rectangle : public Shape {  
  public:  
   void draw() override {  
   cout << "Drawing Rectangle" << endl;  
   }  
  };  
  int main() {  
   Shape\* shape1 = new Circle();  
   Shape\* shape2 = new Rectangle();  
  shape1->draw(); // Output: Drawing Circle  
   shape2->draw(); // Output: Drawing Rectangle  
  delete shape1;  
   delete shape2;  
  return 0;  
  }  
  **Explanation:**
  + In this example, Shape is an **abstract class** because it has a pure virtual function draw().
  + The derived classes Circle and Rectangle implement the draw() method, providing specific functionality.
  + The advantage of abstraction is that the user only interacts with the interface (the abstract class), without needing to know the specific details of how the derived classes implement the draw() method.
* **Real-World Analogy:**Abstraction is like driving a car—you only need to know how to operate the steering, accelerator, and brakes without worrying about the inner mechanics of how the engine works.  
     
     
  Example:  
  #include <iostream>  
  using namespace std;  
  class abstraction {  
  private:  
   int a, b;  
  public:  
   // method to set values of private members  
   void set(int x, int y) {  
   a = x;  
   b = y;  
   }  
   void display() {  
   cout << "a = " << a << endl;  
   cout << "b = " << b << endl;  
   }  
  };  
  int main() {  
   abstraction obj;  
   obj.set(10, 20);  
   obj.display();  
   return 0;  
  }  
     
     
  Advantages Of Abstraction  
  Only you can make changes to your data or function, and no one else can.  
  It makes the application secure by not allowing anyone else to see the background details.  
  Increases the reusability of the code.  
  Avoids duplication of your code.  
     
     
     
  Abstract classes can’t be instantiated, i.e., we cannot create an object of this class. However, we can derive a class from it and instantiate the object of the derived class. An Abstract class has at least one pure virtual function.  
     
  **Properties of the abstract classes:**
  + It can have normal functions and variables along with pure virtual functions.
  + Prominently used for upcasting(converting a derived-class reference or pointer to a base-class. In other words, upcasting allows us to treat a derived type as a base type), so its derived classes can use its interface.
  + If an abstract class has a derived class, they must implement all pure virtual functions, or they will become abstract.

#include <iostream>  
using namespace std;  
class Base {  
public:  
 virtual void s() = 0; // Pure Virtual Function  
};  
class Derived : public Base {  
public:  
 void s() { cout << "Virtual Function in Derived\_class"; }  
};  
int main() {  
 Base\* b;  
 Derived d\_obj;  
 b = &d\_obj;  
 b->s();  
}  
Output : Virtual Function in Derived\_class  
If we do not override the pure virtual function in the derived class, then the derived class also becomes an abstract class.   
   
We cannot create objects of an abstract class. However, we can derive classes from them and use their data members and member functions (except pure virtual functions).