1. ¿Qué son los modificadores de acceso a atributos y métodos?

* ¿Cuáles son los diferentes modificadores de acceso disponibles en C++?
* ¿Cómo se utilizan los modificadores de acceso para controlar el acceso a los atributos y métodos de una clase?

2. ¿Qué es un constructor de clase?

* ¿Cuál es el propósito de un constructor de clase?
* ¿Cómo se define un constructor de clase en C++?

3. ¿Qué es un destructor de clase?

* ¿Cuál es el propósito de un destructor de clase?
* ¿Cómo se define un destructor de clase en C++?

4. ¿Qué es la parte private y public de una clase en C++?

* ¿Cuáles son las diferencias entre las partes private y public de una clase?
* ¿Cómo se utilizan los modificadores de acceso private y public para controlar el acceso a los atributos y métodos de una clase?

5. ¿Qué es la sobrecarga de métodos?

* ¿Cuál es el propósito de la sobrecarga de métodos?
* ¿Cómo se implementa la sobrecarga de métodos en C++?

6. ¿Qué es la colaboración de clases en C++?

* ¿Cuál es el propósito de la colaboración de clases?
* ¿Cómo se implementa la colaboración de clases en C++?

. 7. ¿Qué es la herencia?

* ¿Cuáles son los diferentes tipos de herencia?
* ¿Cuáles son las ventajas y desventajas que ofrece? ¿
* Cómo se implementa en C++?

8. ¿Cuál es la diferencia entre la colaboración y la herencia en programación orientada a objetos?

Respuestas

1. En C++, los modificadores de acceso se utilizan para controlar la visibilidad y accesibilidad de los atributos y métodos de una clase. Existen los siguientes modificadores:   
   public: Los miembros declarados como public son accesibles desde cualquier parte del programa.

protected: Los miembros protected son accesibles desde la misma clase y desde las clases derivadas.

private: Los miembros private solo son accesibles desde dentro de la misma clase.

Para usarlos, dentro de la clase se encapsulan los métodos y atributos con sus diferentes modificadores, según lo que queramos.

1. Un constructor de clase es una función especial que se llama automáticamente cuando se crea un objeto de la clase. El propósito de un constructor es inicializar los atributos de un objeto. Se define como un método con el mismo nombre que la clase, sin especificar el tipo del método para el retorno.
2. Un destructor de clase es una función especial que se llama automáticamente cuando un objeto de la clase es destruido. El propósito de un destructor es liberar recursos que se hayan ocupado en su uso. Se define como un método con el mismo nombre que la clase (al igual que el constructor), pero precedido por un “~”.
3. La parte privada de una clase es aquello sección donde sus miembros son solo accesibles desde la misma clase, y la parte pública es accesible desde cualquier parte del programa, siendo esta su principal diferencia.   
   Para indicar que parte es privada y pública, se abre de la siguiente manera:  
   class Ejemplo{  
    private:  
    /\*parte privada\*/  
    public:  
    /\*parte pública\*/  
   };
4. La sobrecarga de métodos es la capacidad de definir múltiples métodos con el mismo nombre, pero diferentes parámetros. Su principal propósito es permitir que una función se comporte de manera diferente según los parámetros que reciba. Para implementar la sobrecarga de métodos, hay que hacer lo siguiente:

Class Ejemplo{  
 public:  
 void metodo(int a){  
 //Codigo  
 }

void metodo(int a, string b){  
 //Codigo  
 }

};

1. La colaboración de clases es cuando tenemos varias clases que se relacionan y comunican entre ellas. Su propósito es poder repartir las tareas entre diferentes clases, mejorando la eficiencia y reutilización del código.  
   Para implementarla, podemos realizar lo siguiente  
   class ClaseA{  
   public:  
    void metodoA(){  
    //Codigo  
    }  
   };  
   class ClaseB{  
   private:  
    ClaseA objA;  
   public;  
    void metodoB(){  
    objA.metodoA(); //Colaboración  
    }  
   };
2. La herencia es un mecanismo que permite crear una nueva clase a partir de una clase existente, heredando sus atributos y métodos.  
   Existen diferentes tipos de herencias:  
   Simple: Una clase hereda de una única clase base.  
   Múltiple: Una clase hereda de más de una clase base.  
   Multinivel: Una clase hereda de otra, que a su vez hereda de otra.  
   Jerárquica: Una clase base tiene múltiples clases derivadas.  
   Híbrida: Combinación de los tipos anteriores.  
   Aparte de estos tipos por jerarquía, tenemos los tipos según su modificador de acceso:  
   Herencia pública: todos los miembros públicos y protegidos de la clase base conservan esos mismos niveles de acceso respectivamente en las clases derivadas.  
   Herencia protegida: todos los miembros públicos de la clase base adquieren el nivel de acceso protegido en las clases derivadas, mientras que los miembros protegidos conservan su nivel de acceso. Lo anterior indica que una clase derivada puede luego heredar a otra clase los miembros protegidos que heredó de su clase base.  
   Herencia privada: privada: todos los miembros públicos y protegidos de la clase base adquieren el nivel de acceso privado en las clases derivadas. De ahí se desprende que una clase derivada que haya heredado mediante herencia privada no puede heredar a otras clases los miembros que ha heredado de otras clases.   
   Sus ventajas son:  
   Reutilización de código  
   Facilita la extensión y mantenimiento  
   Sus desventajas son:  
   Puede complicar mucho el diseño  
   Mayor acoplamiento (dependencia) entre clases  
   Potenciales problemas de mantenimiento por un mal uso  
   Se implementa de la siguiente manera:  
   class Base {  
   public:  
    void metodoBase() {  
    //Codigo  
    }  
   };  
   class Derivada : public Base {  
   public:  
    //También tiene el método base  
    void metodoDerivada() {  
    //Codigo  
    }  
   };
3. La colaboración se centra en hacer interactuar a las clases entre sí para cumplir tareas complejas, mientras que la herencia se centra en la creación de una relación jerárquica para reutilizar y extender el comportamiento de una clase existente.