1.¿Qué es una relación de herencia?

La herencia es una de las características principales de la programación orientada a objetos, con ella podemos reutilizar el mismo código para varias clases que pueden derivar de una misma clase padre. Vamos a hacer un ejemplo con videojuegos, todos los enemigos y el player pueden moverse, atacar, recibir daño, etc. Pero para el player necesitamos que se mueva a por medio de un input del teclado por ejemplo en cambio un enemigo necesitamos que sea de manera automática, entonces cada uno tiene una función diferente hasta llegar a la función de moverse, así que lo que hacemos es una clase padre como ser "Character" que tenga las funciones de Move(), Attack(), etc. y 2 clases hijo una Player y otra Enemy que hereden esas funciones y las usen cuando les parezca.

2.¿Qué particularidad tiene un campo o método protected?

La particularidad es que a diferencia de un campo o función private en un caso de herencia las clases derivadas o hijas pueden recurrir a las mismas, en el ejemplo anterior si quisiéramos que "Enemy" tenga una función que haga un random para hacer un movimiento random y pasárselo a Move(int x, int y) como parámetro necesitaríamos que Move en la clase padre este público o protected. En caso de que sea público tendríamos la vulnerabilidad que cualquier clase u objeto instanciado podría llamar a move, si no quisiéramos eso tendría que estar como protected.

3.¿Qué particularidades existen con respecto al orden de ejecución de los constructores y destructores cuando hay herencia?

En cuanto a los constructores que construye primero el padre y luego el hijo, es decir si tenemos una clase Base, y luego Derivada1, y luego una Derivada2 que hereda de Derivada1, los constructores serán Base-Derivada1-Derivada2. En cambio cuando se llaman los destructores primero se destruyen los hijos y luego el padre, es decir Derivada2-Derivada1-Base. Esto es porque las clases hijas requieren de sus padres para poder existir.

4.¿Qué es el polimorfismo?

El polimorfismo es una particularidad de la programación orientada a objetos que permite mediante una misma función de la clase base sus clases derivadas pueden hacer cada una acciones distintas, por ejemplo supongamos que tenemos una clase base que sea FiguraGeometrica y tiene una función Draw(), en caso de un círculo el dibujo se hace de una forma X, y en caso de un triángulo de una forma Y, entonces para poder hacer esto y que desde el programa simplemente llamemos a FiguraGeometrica.Draw(); y el mismo se encargue de dibujar cada uno lo que debe usamos el polimorfismo. Esto lo podemos ver en varios patrones de diseño, y es muy útil ya que a veces necesitamos que un manager active alguna función como "attack" o "move" pero no todos los objetos instanciados deben hacerlo de la misma forma.

5.¿Qué es un método virtual?

Un método virtual es la forma de usar polimorfismo, en el caso de la figura geométrica y su función Draw(), necesitaríamos que esta sea virtual, para por ejemplo en triángulo usar una función Draw() override para sobrescribir la funcionalidad de la función base. Lo que hace esto es que si ninguna clase hija sobrescribe la función Draw(), realizará la de la clase base pero si alguna le hace el override esta sobrescribirá la funcionalidad base.

6.¿Por qué es importante usar destructores virtuales cuando usamos polimorfismo?

Porque si no lo usamos los objetos no se destruyen de manera correcta.

7.¿Qué es una clase abstracta?

Es una clase que no se puede instanciar, es decir para poder ser instanciada debemos hacerlo por medio de un hijo. En C++ esto se hace haciendo todas sus funcionalidades de manera virtual, por lo que ninguna de estas funciones estará implementada a no ser que sea la de un hijo.

8.¿A qué se conoce como “método virtual puro”?

Es un método que no está implementado en la clase base, es decir está vacío.