Questions:

4 basic principles of OOP: Encapsulation, Abstraction, Inheritance, and Polymorphism

- Encapsulation

Encapsulation is the practice of containing the data of an object inside of itself and only exposing certain information as needed by the user. Encapsulation also requires defining some fields as private and some as public.

- Abstraction

Abstraction uses classes and objects to hide the internal details of an application. This reduces the complexity and improves the readability of code. It is similar to Encapsulation.

- Inheritance

Inheritance is used to create a hierarchy of classes by inheriting the properties of other classes. Inheritance is also transitive i.e. if German Shepherd extends Dog, and Dog extends Animal, then German Shepherd also extends Animal.

- Polymorphism

Polymorphism (meaning "having multiple forms") uses the methods created by inheritance to perform different tasks. An example would be having subclasses of an Artist class named CubismArtist, RenaissanceArtist, and RomanticismArtist, with each having their own implementation of painting.

What is a class and what is an object? What is the difference?

A class is the blueprint from which individual objects are created. It can contain various data types and methods. An object is created from a class and contains different fields and methods. For example: A bicycle would be an object, while a blueprint for that bicycle would be a class. It would also have different fields (bike color, speed, gear, etc.) and methods (change gear, speed up, break, etc.). The difference is that a class defines the different values of an object.

Abstraction, Encapsulation - how is it different from abstraction?

Abstraction hides complex implementation details and shows only essential features to the user, while encapsulation bundles data and methods that operate on that data within a single unit and restricts direct access to some of the object's components.

Access modifiers

Access modifiers determine whether other classes can use a particular field or invoke a particular method. There are two main types of access modifiers: public and private. Public can be used by all classes everywhere, while private can only be accessed in its own class.

Problems with multiple inheritance; example (not a coding one)

Problems with multiple inheritance come up in certain situations that include collisions with the multiple classes inherited. For example: the so-called "diamond of doom," which occurs when a class inherits multiple classes that each inherit the same base class. This problem creates many issues that need to be resolved, such as: whether the class with multiple inheritance should have one or two copies of the base class and the need to resolve certain types of ambiguous references.

Polymorphism (static one and dynamic one)

Static polymorphism is exhibited during compile time, while dynamic polymorphism is exhibited during run-time. For example, static polymorphism is used in method overloading and dynamic in method overriding.

Data types: value types and reference types?

Variables of reference types store references to their data with, e.g., pointers, while variables of value types directly contain their data.

Data types: Where are we saving them in memory?

Data types are allocated to two different parts of memory: the heap and the stack. The heap stores large data structures and objects with dynamic lifetimes. Its memory may be allocated or deallocated during the program runtime. The stack is used for managing local variables, function arguments, and control information, such as return addresses.

Comparison by value and comparison by reference?

Comparison by value occurs between primitive data types, and comparison by reference occurs with complex data types. This causes where Integers x and y, both having the value 1000, to return false when compared (x == y). This doesn't happen if x and y had primitive data types (such as int) or if the value of x and y was compared.

Garbage collector; phases of cleaning memory?

A Garbage Collector is used in many high-level programming languages in order to automatically manage memory by reclaiming unused or unreachable objects, thereby preventing memory leaks and reducing the risk of program crashes. A Garbage Collector has two phases of cleaning memory, starting with marking, where it identifies which objects are still in use and which are not. The ones in use are marked, and the rest are considered garbage. Then it moves onto the sweep phase, where the heap is traversed to find the gaps between the objects in use. These gaps are recorded in a free list and are made available for new object allocation.

Bonus Questions:

Generics

Generics make it possible to use methods without strictly defining the data types used. Generic classes and methods combine reusability, type safety, and efficiency in a way that their nongeneric counterparts can't. Generic type parameters are replaced with the type arguments during compilation.

Managed & unmanaged code

Managed Code is code that is managed by a runtime. The code is firstly compiled to an “Intermediate Language” (IL) and then that IL code is Just-In-Time compiled by the runtime. This is in contrast to **Unmanaged Code**, in which the code is compiled directly to a binary and started. Managed Code is written in high-level languages, such as C#, and unmanaged code is written in lower-level languages, such as C or C++.

Aggregation and composition

Aggregation and Composition are both subsets of associations with important differences. **Aggregation** implies a relationship where the child can exist independently of the parent. Example: Class (parent) and Student (child). Delete the Class, and the Students still exist. **Composition** implies a relationship where the child cannot exist independent of the parent. Example: House (parent) and Room (child). Rooms don't exist separate from a House.

Serialization and deserialization

Serialization and deserialization are processes used to convert data into a form that can be persisted (saved) or transported. Serialization is the process of converting an object’s data to a byte stream. This byte stream is then able to be deserialized. Deserialization is the process of converting a byte stream back into an object. These two processes are commonly used to convert objects of a programming language to and from JSON in order to share data across the backend and frontend.

Git

merge vs rebase

Merging and rebasing are both used when you want to combine two separate branches together. Merge takes all the changes in one branch and merges them into another branch in one commit. Rebase takes the point at which branch started and moves it to a new starting point. Rebasing is more destructive, but is considered more “clean” for some.

revert and reset

git revert undoes a change part of a commit to the local/origin repo by creating a new commit. Git reset will delete/undo changes which are committed in local repo.

types of reset

There are three different types of resets: soft, mixed, and hard. Soft will move changes from the local commit to the staging area, mixed will move changes from the local commit to the working directory, and hard will delete changes from the working directory.