Answers

OOP

1. Class vs. Object:

   Class: It's a template to create Objects which has a name. The class itself doesn't occupy memory. It only tells the object the structure and behavior. A class can have attributes, contains variables and functions.

   Object: It's an instance of a class. You can create multiple instances out of a class if its class is non static. If the class is static, you can only create one object. Every object you create needs memory and you can use the functions of the objects. It doesn't matter if the object is public, protected or private, the object always needs the same amount of space in the heap. The access modifiers only describe where you are able to use the access the object.

   Difference: The class only defines what something should be also the object is concrete and needs the actual definition of attributes.

2. Abstraction:

   Two types: Data abstraction and process abstraction

   Data abstraction: The data is provided through some functions which you can use on the object. get_age()

   Process abstraction: The user doesn't need to know the whole application. As an example if you want to turn on your car, you don't need to know how the car turns itself on. You only have to start the car with a button press and your key.

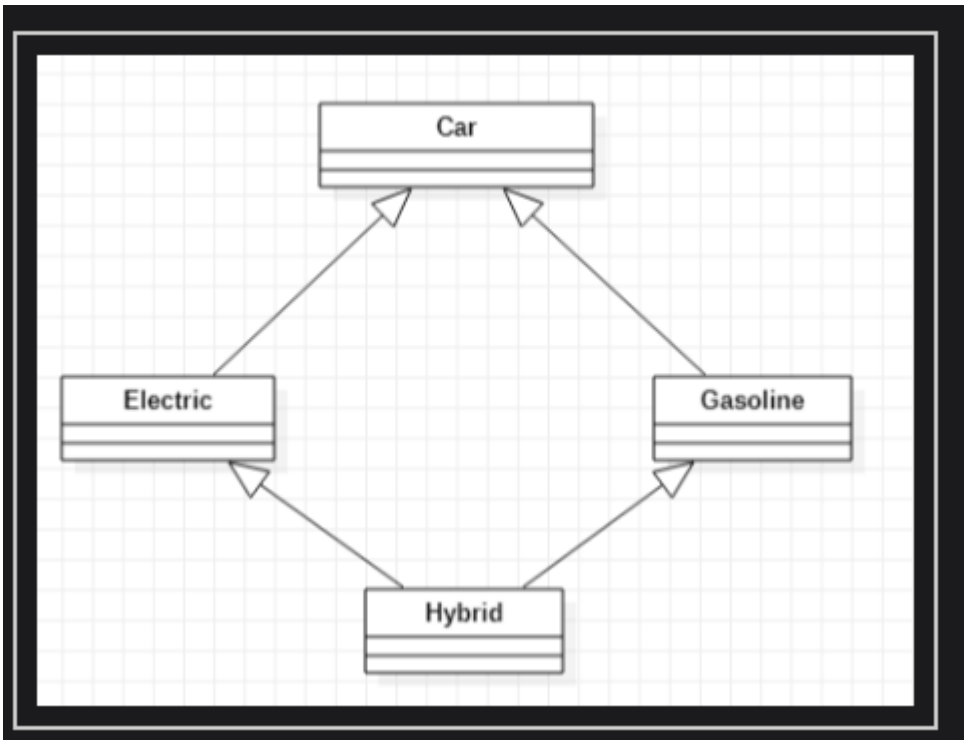
   Encapsulation: You can use different access modifiers to reach encapsulation.

   - Default
   - Private
   - Protected
   - Public

3. Problems with multiple inheritance:

   Many OOP only support single inheritance because if you have multi inheritance it could lead to the diamond problem. The **diamond problem** occurs if one class inherits from two classes as in the example. Now you have **duplicated constructors** and the car constructor gets called twice every time which is inefficient.

   Also if you might have two constructors with the same name it could lead to misunderstandings.



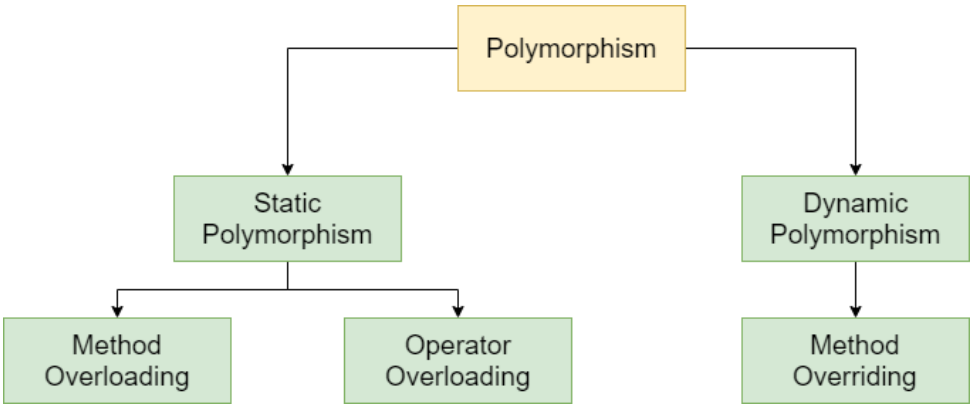
4. Polymorphism (static one and dynamic one):

   Static:

   - while compile time
   - method/operator overloading (You create a second constructor with different amount of attributes)

Dynamic:

- while runtime
- you can override methods with a new child class



5. Data types:

   Value types:

   Stored in stack, they hold the value directly (int, float, double, char, ...)

   Reference type:

   Stored in heap (object, class, arrays, ... but also string!) a string in C# is immutable so you can't change the value of it. If you change it (str1 = "test") a new string gets created and the reference changes.

6. Comparation & Garbage collector

   Comparation by value:

   "==" checks if content of two values or objects are equal but doesn't check the reference.

   Comparation by reference:

   Needs the method "ReferenceEquals" (It checks if two objects have the same reference in the heap)

Garbage collector (GC):

The GC got added to C# so it's easier and faster to develop because you don't have to manage the memory by your own. It's still possible in C# to manually trigger the GC but there aren't many cases where you need to do it and normally it works fine by its own.

What GC does:

Stack: As soon as a variable isn't in the scope or in use anymore it will get removed from the stack.

Heap: If there is a reference which doesn't get used anymore, the GC will automatically remove it.

Generics

1. Managed and unmanaged code

   Managed code:

   The code is managed by the CLR (Common Language Runtime). It provides the compilation of the current language, security management, memory management, type safety and more.

   Unmanaged code:

   It's executed directly by the OS so the programmer is responsible for everything the CLR was doing for him like type safety etc.

   If we are using C# in Visual Studio we have the CLR which also includes the GC by default which means we don't have to check the type safety and we don't have to handle the memory by ourselves.

2. Aggregation(Introvert) and composition(extrovert)

   Aggregation:

   It's the weaker relationship than the composition. One thing of the relation can be for itself without being together with anything else. If you remove the Engine of a car the wheels still exist and can be used for a different car.

   Composition:

It means that two things must be together and nothing is able to survive alone. If one gets removed the other thing must get removed too because it has a strong reliance.

3. Serialization and deserialization

   Serialization:

   It's the process to convert the database information/objects in a format like (JSON, XML, HTML, Protobuf etc.). You need to serialize the data before the transfer so the other end is able to understand and work with the data.

   Deserialization:

   It's the opposite of Serialization. It happens when you send an update Request from the frontend with the JSON data to the backend as an example. Before you are able to update the database you have to deserialize it so the database is able to understand the information you got from the frontend.

Git

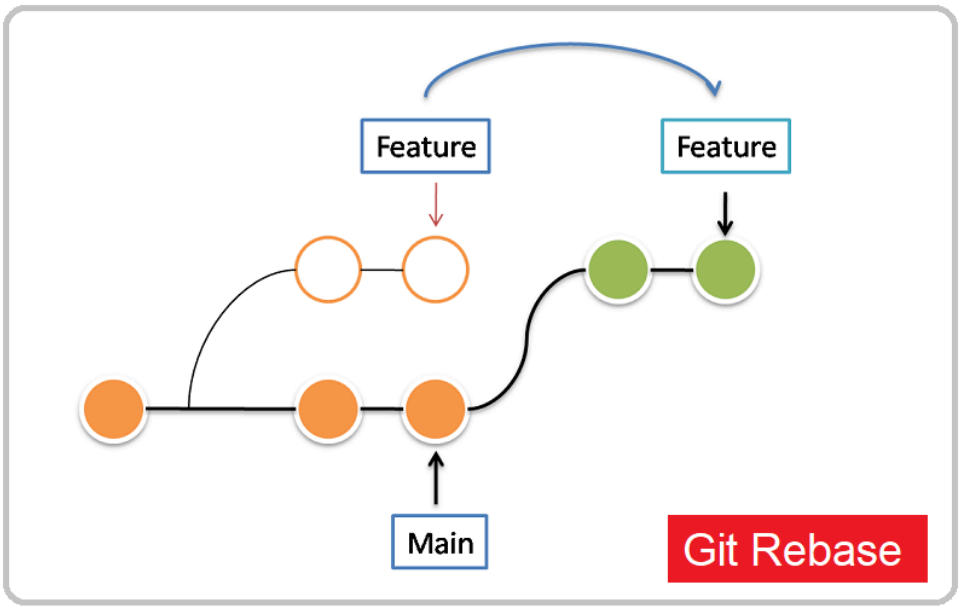
1. Merge vs rebase

   Merge:

   Combine changes of two branches so they are on the same level. After the merge you have everything of your current branch + the merged branch.

   Rebase:

   With the command rebase you are able to move changes from one branch to another. It can make the git history clearer and easier to read.



2. Revert and reset

   Revert: Git revert resets the branch to an older commit of the branch but adds a new commit on top of the head. You can use this command instead of reset if you don't wanna delete your history because of some reason.
   Reset
   : With the reset command you are able to reset your changes and go to the earlier commit.

3. Types of reset

   git reset **—soft** / **—mixed(default)** / **—hard**

   meaning of...

   ...soft: Resets only the head of the branch but keeps the rest saved. Commits and local changes stay.

   ...mixed: Resets the head, deletes the staging area, keeps the local changes

   ...hard: Resets everything to the chosen commit