Why would you want to write your code in an object-oriented programming language?

Writing code in an object-oriented programming (OOP) language provides several advantages. OOP allows for modularity, where different parts of the code can be organized into separate entities (classes and objects) that are easier to manage, maintain and update. It promotes readability because you can create new objects based on existing classes, making your code more efficient and reducing redundancy. Additionally OOP enhances code readability. It allows for easier debugging, testing and collaboration between teams working on different components of the system.

What's the difference between a class and an object?

A class is a blueprint or template for creating objects. It defines the attributes (properties) and behaviors (methods) that the objects created from it will have. Think of a class as an idea or a design, like a Train class that has attributes like brand and model and behaviors like moving or stopping. An object, on the other hand, is an instance of that class. It's a specific realization of the class with concrete values. For example, if the Train class is the design, an object would be a specific train, like a NS sprinter. Essentially, a class defines the structure and actions, while objects are actual representations of that structure in memory.

What is Composition?

Composition is an object oriented programming principle where a complex object is constructed using multiple smaller objects. Instead of writing everything in one large class, composition allows different parts to be separated into their own classes and combined into a bigger structure. For example a Train can contain multiple wheels and seats as its components. Each component is a separate class and together they form a complete Train.

Why do we use Composition?

Composition helps to keep a system well structured and organized by breaking it down into smaller, independent components. This makes the code more readable and modular, as each part has a clear responsibility. By using composition, individual components can be reused in multiple objects, which reduces redundancy and improves efficiency. Additionally, maintainability is improved because modifying one component does not require changes to the entire system.

How do we use Composition in our code?

To use composition in our code, we create objects that contain instances of other objects rather than extending a base class. This enables better code reusability and maintainability.

What is Inheritance?

Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a class (called the subclass) to derive properties and behaviors from another class (called the superclass). This enables code reuse. The subclass inherits attributes and methods from the superclass and can override or extend them to implement specific behaviours.

Why do we use Inheritance?

We use Inheritance because it allows us to reuse existing code, reducing duplication and improving maintainability. It also helps define relationships between classes, creating a structured and organized codebase. New functionality can also be added to subclasses without modifying the parent class, making the system more adaptable.

How do we use Inheritance in our code?

In object-oriented programming (OOP) languages, Inheritance is implemented using a specific syntax. For example a Charmander class inherits from a Pokemon class. This demonstrates how Inheritance helps in organizing and reusing code efficiently.

What are static variables and methods?

A Static Variable is a variable that is declared with the static keyword, meaning it retains its value across all instances of the class. A Static method is a method that is also declared with the static keyword, allowing it to be called without an object. It can only directly access static variables and other static methods within the class. Static variables and methods belong to the class itself rather than to any specific instance of that class. This means they are shared among all instances of the class. A static variable holds a value that is common to all objects of the class, while a static method can be called on the class itself without needing an object.

Why do we use static variables and methods?

Static variables and methods are useful when you want to share a single piece of data or behavior across all instances of a class, or when you want to perform operations that are independent of object state. They save memory because they are stored only once, and they can simplify access to shared resources or utility functions.

How do we use static variables and methods in our code?

To declare a static variable or method, you use the static keyword. For example totalrounds is a static variable shared across all instances of the Arena class.

What is Encapsulation?

Encapsulation is a fundamental principle of Object-Oriented Programming (OOP) that restricts direct access to an object’s data while allowing controlled access through methods. It helps protect the integrity of the data and prevents unintended modifications. In C# Encapsulation is implemented using modifiers such as private and public. Encapsulation ensures that an object’s internal state is hidden from the outside and only exposed through controlled mechanisms.

Why do we use Encapsulation?

Encapsulation simplifies debugging and modification of the class without affecting other parts of the program. It also encourages modular code design, making components easier to reuse. By using Encapsulation, we can enforce constraints on data and ensure that objects remain in a valid state.

How do we use Encapsulation in our code?

In code, we use Encapsulation by making fields of a class private, and providing getter and setter methods (properties) to access and modify those fields. For example, aantalOVPaaltjes is private and can't be accessed directly from outside the class. Another example is getAantalOVPaaltjes and setAantalOVPaaltjes they are public properties, providing controlled access to the private field. The getter retrieves the value, and the setter updates the value. This way, Encapsulation ensures that the internal data of the object is protected and modified only through valid operations, providing data integrity and beter code structure.