Question Answering Bonus

1. What is operator overloading ?

   What is Operator Overloading?

   - Operator overloading allows you to specify how operators work for instances of your class or struct.
   - You define special methods (using the operator keyword) that are called when an operator is used with your type.
   - This is useful for creating natural and readable code, especially for types that represent mathematical or logical entities (e.g., vectors, complex numbers, or custom data structures).

   Key Points:

   - Not all operators can be overloaded. C# allows overloading of operators like +, -, *, /, ==, !=, >, <, ++, --, etc., but not operators like &&, ||, or =.
   - Overloading must be done thoughtfully to maintain clarity and avoid unexpected behavior.
   - Operator overloading is declared as public static methods in the class or struct.

   Syntax for Operator Overloading

   To overload an operator, you define a method with the following characteristics:

   - Use the operator keyword followed by the operator symbol (e.g., +).
   - The method must be public and static.

- The method must return a value (often the same type as the class) and take parameters that represent the operands.

General Syntax:

```
public static ReturnType operator OperatorSymbol(Type1 operand1, Type2
operand2)
{
   // Implementation
}
```

Rules and Best Practices for Operator Overloading

1. **Overload in Pairs**:
    - If you overload ==, you must also overload !=.
    - If you overload <, you must also overload >.
    - If you overload ==, override Equals and GetHashCode for consistency.
2. **Maintain Intuitive Behavior**:
    - Operators should behave in ways that align with user expectations. For example, + should represent addition or combination, not something unrelated like subtraction.
3. **Handle Nulls Carefully**:
    - As shown in the == operator, check for null to avoid NullReferenceException.
4. **Keep It Simple**:
    - Avoid overloading operators in ways that make code confusing. Only overload when it makes sense for the type (e.g., mathematical types like vectors or matrices).
5. **Immutability Consideration**:

    - Operators should typically return a new instance rather than modifying existing ones to avoid side effects, as shown in the + operator returning a new Vector2D.

6. **Performance**:
    - Ensure the implementation is efficient, as operators are often used in performance-critical scenarios (e.g., vector calculations in graphics).

Common Operators You Can Overload

- **Arithmetic**: +, -, *, /, %
- **Comparison**: ==, !=, <, >, <=, >=
- **Unary**: ++, --, +, -, !, ~
- **Others**: true, false (for logical operations), <<, >>, etc.

Non-overloadable Operators:

- Logical operators (&&, ||)
- Assignment (=)
- Conditional (?:)
- Null-coalescing (??)