Part01:

Github:

https://github.com/janaashraf888/rowadmasrtasks.git

Self:

1. Upcasting and Downcasting in Relation to Inheritance

Upcasting converts a derived class object to a base class reference and supports polymorphism, allowing a general interface to handle multiple derived types. Downcasting converts a base class reference back to a derived type, which requires caution to avoid runtime errors. Design-wise, upcasting promotes abstraction and flexible code, while downcasting allows access to specialized behavior but should be used sparingly to maintain maintainability.

---

2. LinkedIn Article about Boxing and Unboxing

Boxing converts a value type into a reference type, storing it on the heap, while unboxing retrieves the value type from the reference. This mechanism allows value types to be treated as objects but can affect performance due to memory allocation and type

checking. Understanding boxing and unboxing is essential for writing efficient C# programs.

---

3. Unmanaged Resources Recovery and I/O Operations

Unmanaged resources, such as file handles, network sockets, or database connections, are not managed by the garbage collector and must be explicitly released. Proper handling ensures that these resources are freed correctly, preventing memory leaks or system instability. I/O operations often involve unmanaged resources, making correct management crucial for reliable program execution.

---

4. Customizable Exceptions

Custom exceptions are user-defined classes derived from the base exception class. They allow meaningful and domain-specific error messages, improving readability, maintainability, and clarity in error handling across the application.

---

5. Global Middleware Try-Catch

Global middleware try-catch handles exceptions at a centralized level, catching all unhandled errors in the application. It allows consistent logging, error handling, and response formatting, reducing redundancy and improving maintainability.

---

6. Global Middleware Try-Catch (Alternative View)

Global middleware wraps all request processing in a centralized try-catch block. This ensures that exceptions are consistently managed, supporting clean business logic and unified error handling throughout the application.

---

7. Unmanaged Resources Recovery

Recovering unmanaged resources involves explicitly releasing memory and handles not controlled by the garbage collector. Proper recovery prevents memory leaks and resource locks, ensuring system stability and efficiency.

---

8. I/O Operations

I/O operations involve reading from or writing to external resources like files, databases, or network sockets. Correct handling ensures resources are opened, used, and closed safely, and exceptions are managed to maintain program stability.

---

9. Use Case to Define Your Own Exception

Defining a custom exception allows precise, domain-specific error handling tailored to the application’s business logic. It improves readability, maintainability, and ensures meaningful error messages for specific scenarios.