
Question: What are the benefits of using a generic sorting algorithm over a non-generic one?

Answer: The main benefits are reusability and efficiency, since the same algorithm can work with any data type without code duplication, while ensuring type safety and better performance.

Question: How do lambda expressions improve the readability and flexibility of sorting methods?

Answer: Lambda expressions make code shorter and clearer, allowing the comparison logic to be written inline instead of creating separate methods, which increases flexibility when changing sorting rules.

Question: Why is it important to use a dynamic comparer function when sorting objects of various data types?

Answer: A dynamic comparer allows defining custom sorting logic at runtime based on the data type, making it possible to handle multiple types without writing separate sorting algorithms for each.

Question: How does implementing IComparable in derived classes enable custom sorting?

Answer: By implementing IComparable, a class defines its own CompareTo method, which specifies how its objects should be compared, enabling custom and consistent sorting.

Question: What is the advantage of using built-in delegates like Func<T, T, TResult> in generic programming?

Answer: They simplify passing custom logic as parameters, promote reusability, and reduce boilerplate code by avoiding the need to create separate delegate types.

Question: How does the usage of anonymous functions differ from lambda expressions in terms of readability and efficiency?

Answer: Anonymous functions use a longer, older syntax, while lambda expressions are shorter, more expressive, and generally more efficient, improving code readability.

Question: Why is the use of generic methods beneficial when creating utility functions like Swap?

Answer: Generic methods allow utility functions like Swap to work with any data type without duplication, ensuring reusability and type safety.

Question: What are the challenges and benefits of implementing multi-criteria sorting logic in generic methods?
Answer: The benefit is flexibility in sorting by multiple fields or rules, while the challenge lies in handling complex comparison logic and maintaining performance with different data types.

Question: Why is the default(T) keyword crucial in generic programming, and how does it handle value and reference types differently?
Answer: default(T) provides a safe way to initialize variables when the type is unknown. For value types, it returns zero or equivalent; for reference types, it returns null.

Question: How do constraints in generic programming ensure type safety and improve the reliability of generic methods?
Answer: Constraints restrict the types that can be used with generics, ensuring only valid operations are performed, which improves safety, reliability, and prevents runtime errors.

Question: What are the benefits of using delegates for string transformations in a functional programming style?
Answer: Delegates allow passing transformation logic as parameters, enabling flexible, reusable, and modular string operations in a clean functional style.

Question: How does the use of delegates promote code reusability and flexibility in implementing mathematical operations?
Answer: Delegates let developers pass mathematical logic dynamically, allowing the same method to perform different operations without rewriting code, thus improving reusability.

Question: What are the advantages of using generic delegates in transforming data structures?
Answer: Generic delegates can operate on any type of data, making them versatile for transformations while ensuring type safety and reducing code duplication.

Question: How does Func simplify the creation and usage of delegates in C#?
Answer: Func provides a predefined delegate type for methods that return a value, avoiding the need to define custom delegates and simplifying delegate usage.

Question: Why is Action preferred for operations that do not return values?

Answer: Action is specifically designed for methods with no return value, making the code more explicit and reducing unnecessary complexity compared to using Func.

Question: What role do predicates play in functional programming, and how do they enhance code clarity?

Answer: Predicates represent conditions returning a boolean value. They make code more readable and expressive, especially in filtering and searching operations.

Question: How do anonymous functions improve code modularity and customization?

Answer: They allow defining custom logic inline without creating separate named methods, making the code more modular and easily customizable.

Question: When should you prefer anonymous functions over named methods in implementing mathematical operations?

Answer: Anonymous functions are preferred for short, one-time operations where defining a named method would add unnecessary complexity.

Question: What makes lambda expressions an essential feature in modern C# programming?

Answer: They provide a concise, expressive, and powerful way to write inline logic, widely used in LINQ, event handling, and functional programming patterns.

Question: How do lambda expressions enhance the expressiveness of mathematical computations in C#?

Answer: Lambda expressions allow mathematical formulas and logic to be expressed directly in code, making computations clearer, shorter, and easier to understand.
