Assignment: Generics in Java

Part I: Basic Assignment [Not Studied Collections]

Generic Classes

1. **Generic Class Implementation:**

Create a generic class `Container` that can store an item of any type. Implement methods to get and set the item, and demonstrate usage with different types.

2. **Generic Class with Multiple Type Parameters:**

Define a generic class `Pair<K, V>` that can store two related objects. Provide methods to get and set both values. Create instances with different type parameters.

3. **Generic Class with Bounded Type Parameter:**

Implement a generic class `BoundedBox<T extends Number>` that can store numerical values only. Add methods to perform basic arithmetic operations on the stored value.

4. **Generic Class with Collections:**

Design a generic class `Stack<T>` that simulates a stack data structure. Implement push, pop, and peek methods.

5. **Nested Generic Classes:**

Create a nested generic class `Outer<T>` with an inner class `Inner<U>`. Show how to instantiate and use both classes.

Generic Methods

6. Generic Method with Multiple Parameters:

Create a generic method `combine` that takes two parameters of different types and returns a `Pair` object containing both values.

7. Generic Method for Searching:

Write a generic method `linearSearch` that searches for an element in an array and returns its index. The method should work with arrays of any type.

8. Generic Method with Wildcards:

Develop a generic method `printCollection` that takes a `Collection<?>` and prints all its elements.

Generic Interfaces

9. Generic Interface Definition:

Define a generic interface `ComparablePair<T>` with a method `int compare(T o1, T o2)`. Implement this interface in a class `StringPair`.

10. Generic Interface with Multiple Type Parameters:

Create a generic interface `Transformer<I, O>` with a method `O transform(I input)`. Implement this interface in a class that converts strings to integers.

11. Generic Interface with Bounded Type Parameters:

Create a generic interface `BoundedProcessor<T extends Number>` with a method `void process(T number)`. Implement this interface to perform arithmetic operations.

Assignment: Generics in Java

Part II: Basic Assignment [Studied Collections]

Generic Methods

1. Generic Method for Sorting:

Write a generic method `sortArray` that takes an array of any type and sorts it. Ensure the method works with arrays of `Comparable` types.

2. Generic Method for Maximum Value:

Implement a generic method `findMax` that takes an array of elements and returns the maximum element. Ensure the type parameter extends `Comparable`.

Generic Interface

3. Generic Interface for Sorting:

Implement a generic interface `Sorter<T>` with a method `void sort(List<T> list)`. Provide a concrete implementation for sorting integers.

Generic Wildcards

4. Generic Wildcards Upper Bound:

Write a method `calculateSum` that accepts a list of numbers (`List<? extends Number>`) and returns the sum of the elements. Test this method with lists of `Integer`, `Double`, and `Float`.

5. Generic Wildcards Lower Bound:

Implement a method `addElements` that adds a series of integers to a list (`List<? super Integer>`). Demonstrate this method by adding integers to a list of `Number`.

6. Generic Wildcards with Collections:

Create a method `copyList` that copies elements from one list to another. Use wildcards to ensure the method works with lists of any type.

7. Generic Wildcards with Upper Bound in Methods:

Implement a method `printNumbers` that accepts a list of numbers (`List<? extends Number>`) and prints each element. Demonstrate this method with different numeric types.

8. Generic Wildcards with Lower Bound in Methods:

Write a method `fillList` that fills a list (`List<? super T>`) with elements of type `T`. Test this method with lists of different types.

Complex Examples

9. Combining Multiple Concepts:

Create a generic class `DataStorage<T>` that uses a generic method `void store(T item)` to add items to a list, and another method `T retrieve(int index)` to get items from the list. Use wildcards to create a method `printAllItems(DataStorage<? extends Number> storage)` that prints all numerical items from a given `DataStorage` instance.

10. Generic Tree Structure:

Implement a generic binary tree class `BinaryTree<T>` with methods to insert, find, and traverse elements. Ensure the type parameter extends `Comparable`.

11. Generic Caching Mechanism:

Design a generic class `Cache<K, V>` that stores key-value pairs. Implement methods to add, retrieve, and remove items from the cache.

12. Generic Graph Implementation:

Create a generic class `Graph<T>` to represent a graph data structure. Implement methods to add vertices, add edges, and perform depth-first search.

13. Generic Priority Queue:

Implement a generic priority queue class `PriorityQueue<T>` using a heap. Ensure the type parameter extends `Comparable`.

14. Generic Utility Class:

Write a generic utility class `ArrayUtils` with static methods to reverse an array, find the minimum value, and find the maximum value. Ensure these methods work with arrays of any type.

15. Generic Pair Processor:

Create a generic class `PairProcessor<T1, T2>` with methods to process pairs of values. Implement a method to combine two pairs into a new pair.

16. Generic Comparator Implementation:

Implement a generic comparator class `GenericComparator<T extends Comparable<T>>` with a method `compare(T o1, T o2)`. Use this comparator to sort a list of custom objects.

17. Generic Algorithm for Merging Lists:

Write a generic method `mergeLists` that merges two lists into one. Use wildcards to ensure the method works with lists of any type.

18. Generic Data Transformation:

Create a generic method `transformList` that applies a transformation function to each element in a list and returns a new list of the transformed elements. Use a functional interface to represent the transformation function.