Generics exercises

1. Create a class HandyMethods.
    a. Write the generic methods Max and Min with type parameter T, parametrized with a List<T>
        i. Ensure all elements in the list implement the interface IComparable<T>
        ii. Let Max return the greatest element and Min the smallest element, as defined by the IComparable implementation.
    b. Write a generic method copy with a type parameter T that takes two arguments of type T[] and copies the content of one array to the other array. Throw an exception if the arrays have unequal lengths.
    c. Write a generic method shuffle with a type parameter T that takes an argument of type T[]. Permute the array using the following algorithm: Repeatedly generate two random numbers i and j, where i and j must be valid array indices and then swap the entry i with the entry j. Perform this operation n times where n is the length of the array.

    (Hint: The Random class was used in the last exercise session)

    Experiment with marking the methods in HandyMethods using the static keyword.
2. Write a class Pair with two type parameters T1 and T2 to represent a pair of values (i.e. the class should have two fields of type T1 and T2). Add an appropriate constructor and getters. Do not add any setters, as the class should be immutable!
3. Add a method swap to the Pair class. The swap method should return a new pair where the first component becomes the second component and vice versa. For example, for the pair (true, 42) the method should return (42, true). (Hint: You will have to swap the type parameters in the return type.)
4. Add methods setFst and setSnd to the Pair class. Each method should take a type parameter C and return a new pair where the appropriate component has been updated. For example, calling setFst with the integer 42 on the pair (true, "Hello World") should return (42, "Hello World").
5. Write a class Dict that takes two type parameters K and V. The class should represent a dictionary, i.e. a mapping from items of type K to items of type V. Internally, the dictionary should maintain a single list of pairs of type Pair. The dictionary should support the operations: Get(K key) and Put(K key, V value). The get method takes a key argument, searches through the list for an element with that key, and returns its value. If the key is not present, it should throw an exception. The put method updates the list with a new pair for the mapping from key to value. If a pair with the key is already in the map it must be updated.