Overview

The purpose of the project is to demonstrate good architectural design. The main aspects of such design will be:

1. Creating a clean generic class that takes in an array of any type.
2. Creating a method that can swap values in that array.

Constraints

- No use of GUI
- A written description and/or comments in your code explaining the rationalisation behind your architectural decisions, and what you would have done with more time, is acceptable..

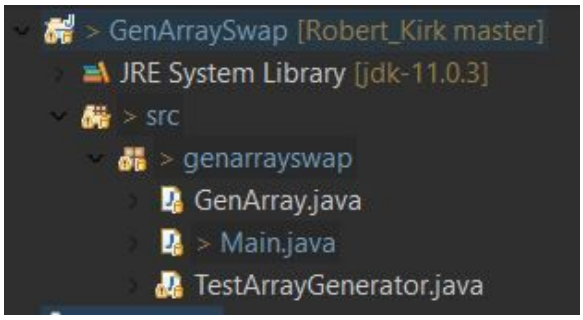
Background

I'm a full stack developer with nearly a one years worth of Java experience about 6 months of which was devoted to the use of Spring Boot.
Considering the amount of time I had for this particular challenge, I tackled it based on what I felt was most important:

1. Neat construction of generic class will allow the class to be used by any type of array.
2. There should be no bounding to the generic types

Screenshots:

File Structure:



- GenArray:
  - Contains a generic class that takes in an array of any type and allows manipulation of it
- Main
  - Controls the starting of the program and runs a test

```
public class Main {

    public static void main(String[] args) {

        Integer[] intArray = new Integer[10];
        GenArray<Integer> genArr = new GenArray<Integer>(intArray);
```

The main produces an Integer array to allow to test the generic class of GenArray.

GenArray is then instantiated taking in the Integer array as an argument

```
        Integer[] intArray = new Integer[10];
        GenArray<Integer> genArr = new GenArray<Integer>(intArray);

        for(int i = 0; i<genArr.getT().length;i++) {
            Integer val = (int)(Math.random()*100)+1;
            genArr.placeValue(val, i);
        }

        System.out.println(genArr.toString());
```

The array held by genArr is then seeded with random values. Then print the array using a custom toString method.

```
    System.out.println(genArr.toString());

    int a = (int) ((Math.random()*intArray.length));
    int b = (int) (Math.random()*intArray.length);

    genArr.exchangeVals(a, b);

    System.out.println(genArr.toString());
```

Generates two values that are to be the index we will be swapped. Then calls exchangeVals which swaps values in the array that is inside of genArr. Then the array is printed out with the toString method.

```
public class GenArray<T> {

    private T[] t;

    public GenArray(T[] arg){
        this.t = arg;
    }
```

GenArray is typed to be generic and it holds a generic array that is to be instantiated by the constructor.

```
public void exchangeVals(int placeA, int placeB) {
    T temp = this.t[placeA];
    this.t[placeA] = this.t[placeB];
    this.t[placeB]= temp;
}
```

This is the method used to swap generic values. It takes in the palaces then uses the temp to hold a value and then rotate the values that are being swaped.

```
@Override
public String toString() {
    String arrStr = "";

    for(T n: this.t) {
        arrStr += n.toString() + " ";
    }

    return arrStr;
}
```

Custom to string method a string by looping through the array and concatenating the values in the array to arrStr. arrString is then returned.

Potential improvements

1. I originally was planning to create a class to generate arrays outside the main method, but I scrapped that for time reasons.

Conclusion:

In conclusion, generics provide an easy way to create reusable code that safely takes care of type issues for you.