Objective(s):

a. To understand java's Arrays, Collections.sort() and Comparator.comparing()
b. To demonstrate java's generic concept
c. To practice implementing simple sorting algorithm.

Given SillyLuckyNumber.java (think of it as Student.java).

```
package code;

public class SillyLuckyNumber {
   private String breed;
   private int luckyNumber;
   private int threeDigit; // 0 to 999
   public SillyLuckyNumber(String s) {
      breed = s;
      for (int i = 0; i < breed.length(); i++)
         luckyNumber += breed.charAt(i);
      threeDigit = luckyNumber % 1000;
   }
   // getters
   @Override
   public String toString() {
      return "<<"+ breed + " "
       + luckyNumber + " " + threeDigit+">>";
   }
}
```

Task 1: The Arrays class contains various methods for manipulating arrays, including sorting.

```
static void ex0() {
  System.out.println("-ex0---");
  int [] arr = {7, 3, 1, 9, 6, 8, 4, 2, 5};
  println(Arrays.toString(arr));
  Arrays.sort(arr);
  println(Arrays.toString(arr));
}
```

For non-primitive types like SillyluckyNumber, simply tell java what is the definition for putting them in the order. (Think of arranging students ascending order on their ids.

This can be achieved by allocating a Comparator object specifying the sort criteria (see commented of anonymous class code).

```
static void demo1() {
  println("-demo1---");
  SillyLuckyNumber [] arr = {
          new SillyLuckyNumber("Terrier"), new SillyLuckyNumber("Jack"),
          new SillyLuckyNumber("Pom"), new SillyLuckyNumber("Beagle")
       };
  println(Arrays.toString(arr));
  // Comparator<SillyLuckyNumber> engine = new Comparator<>() {
  //   @Override
  //   public int compare(SillyLuckyNumber o1, SillyLuckyNumber o2) {
  //         return Integer.compare(o1.getLuckyNumer(), o2.getLuckyNumer());
  //     }
  // };
  Comparator<SillyLuckyNumber> engine = //your code (sort by luckyNumber);
  Arrays.sort(arr, engine);
  println(Arrays.toString(arr));
}
```

Instruction: fill in the lambda expression code for implementing engine.

Answer task1

(o1, o2) -> Integer.compare(o1.getLuckyNumber(), o2.getLuckyNumber());

Task 2: java Collections also contains .sort() which takes a collection object and comparator as its parameters. In addition, new java syntax allows programmer to implicitly allocate a comparator by in the form of Comparator.comparing(method_reference).

```
static void demo2() {
  println("-demo2----");
  ArrayList<SillyLuckyNumber> list
        = new ArrayList<>( Arrays.asList(
              new SillyLuckyNumber("Terrier"), new SillyLuckyNumber("Jack"),
              new SillyLuckyNumber("Pom"), new SillyLuckyNumber("Beagle")
        ));
  println(list);
  Collections.sort(/* your code */);
  println(list);
}
```

Instruction: fill in the code for sorting list by luckyNumber ascendingly.

Answer task2 **list, Comparator.comparing(SillyLuckyNumber::getLuckyNumber)**

Task 3: Because an arraylist is an object belonged to Collection class, it also contains .sort() method. In addition (Though this task is not on the objectives list), java's collections' always perform a shallow copy when creating copy of an existing collection as shown by .sublist() method.

```
static void demo3() {
  println("-demo3----");
  ArrayList<SillyLuckyNumber> list
        = new ArrayList<>( Arrays.asList(
              new SillyLuckyNumber("Terrier"), new SillyLuckyNumber("Jack"),
              new SillyLuckyNumber("Pom"), new SillyLuckyNumber("Beagle")
        ));
  println(list);
  list.sort(Comparator.comparing(SillyLuckyNumber::getLuckyNumer));
  println(list);

  // demo shallow copy
  ArrayList<SillyLuckyNumber> anotherList
            = new ArrayList<>(list.subList(0, list.size()));
  anotherList.get(0).setBreed("newBreed"); //Terrier
  println(list); //notice how Terrier in list is also effected
}
```

Instruction: Write void setBreed(String b) method with updated luckyNumber and threeDigit. Validate whether the state of object index 0 changes.

Answer task3

```
void setBreed(String breed) {
    this.breed = breed;
}
```

Task 5: One may create a java class accepting type T. (This could be the shortest explanation of generic feature.) With java, one way to achieve this is to create an array of Object. And cast it when accessing it. The @SuppressWarnings("unchecked") would tell java compiler not to worry about casting.

For simplicity, we'll add an item of T to arr like a queue. We'll also need set(int i, T instance) to put instance in to cell j^{th}.

In order to be able to perform a comparison-based sorting, MyArrDemo should be able to swap its 2 cells content.

Instruction: write

void swap(int i, int j)

```
package code;

public class MyArrDemo<T> {
  public final int MAX_SIZE = 9;
  private int size = 0;
  private Object [] arr = new Object[MAX_SIZE];

  public void add(T instance) {
    arr[size++] = instance;
  }
  public void set(int i, T instance) {
    arr[i] = instance;
  }
  public T get(int i) {
    @SuppressWarnings("unchecked")
    final T element = (T)arr[i];
    return element;
  }
  public void swap(int i, int j) {
    // your code
  }
  public int currentSize() {
    return size;
  }
  @Override
  public String toString() {
    StringBuffer sb = new StringBuffer();
    sb.append("My snapshot looks like this -> ");
    for (int i = 0; i < size; i++)
        sb.append(arr[i] + ",");
    return sb.toString();
```

Answer task4

```
if (i >= 0 && i < size && j >= 0 && j < size) {
    T temp = get(i);
    set(i, get(j));
    set(j, temp);
  } else {
    throw new IndexOutOfBoundsException("Invalid index");
  }
```

```
static void demo4() {
  println("-demo4----");
  MyArrDemo<SillyLuckyNumber> arr
                      = new MyArrDemo<>();
  arr.add(new SillyLuckyNumber("Terrier"));
  arr.add(new SillyLuckyNumber("Jack"));
  arr.add(new SillyLuckyNumber("Pom"));
}
  arr.add(new SillyLuckyNumber("Beagle"));
  println(arr);
  //arr.swap(1,3);
  //System.out.println(arr);
```

Task 5: create void selectionSort(MyArrDemo<SillyLuckyNumber> arr).

```
static void demo5() {
    System.out.println("-demo5----");
    MyArrDemo<SillyLuckyNumber> arr = new MyArrDemo<>();
      arr.add(new SillyLuckyNumber("Terrier"));
      arr.add(new SillyLuckyNumber("Jack"));
      arr.add(new SillyLuckyNumber("Pom"));
      arr.add(new SillyLuckyNumber("Beagle"));
      arr.add(new SillyLuckyNumber("Cocker Spaniel"));
      arr.add(new SillyLuckyNumber("Basenji"));
    System.out.println(arr);
    // selectionSort(arr);
    // System.out.println(arr);
}
```

Answer task5

```
static void selectionSort(MyArrDemo<SillyLuckyNumber> arr) {
    int n = arr.currentSize();
    for (int i = 0; i < n - 1; i++) {
      int minIndex = i;
      for (int j = i + 1; j < n; j++) {
        if (arr.get(j).getLuckyNumber() < arr.get(minIndex).getLuckyNumber()) {
          minIndex = j;
        }
      }
      if (minIndex != i) {
        arr.swap(i, minIndex);
      }
    }
  }
```

Submission: this pdf Due date: TBA