# Lab 3. Sorting Collections

Author: Yida Tao

Lab material: `Item.java`, `SortingDemo.java`

Suppose we have an `Item` class like this:

```
public class Item
{
   private String description;
   private int partNumber;

   public Item(String desc, int pnumber)
   {
      description = desc;
      partNumber = pnumber;
   }

   public String toString()
   {
      return "[no=" + partNumber + ", desc=" + description + "]";
   }
}
```

And we've created several `Item` objects, and we want to sort these items by different criteria.

```
Item i1 = new Item("Toaster", 2);
Item i2 = new Item("Widget", 8);
Item i3 = new Item("Modem", 5);
Item i4 = new Item("Laptop", 6);
Item i5 = new Item("Chair", 6);
Item i6 = new Item("Book", 2);
```

## Sort by partNumber

We may put items to a `List`, then using the `sort` method by providing a `Comparator`, which compares two items by their partNumber.

```
List<Item> list = new ArrayList<>();
list.add(i1);
list.add(i2);
list.add(i3);
list.add(i4);
list.add(i5);
list.add(i6);
```

```
// sort items by partNumber
Collections.sort(list, new Comparator<>() {
   @Override
   public int compare(Item o1, Item o2) {
      return o1.getPartNumber() - o2.getPartNumber();
   }
});
```

**Think**: Can we further simplify this code? Ask IDEA for help 😉

### Sort by length of the description

Similarly, we could sort the list using another `Comparator`.

### Sort by partNumber, then by description

#### Approach 1

The `Comparator` interface has several convenient static methods for creating comparators. For example, the static `comparing` method takes a "key extractor" function that maps a type (e.g., `Item`) to a comparable type (e.g., `String`). You can also chain comparators with the `thenComparing` method for breaking ties.

#### Approach 2

We've learned that `java.util.TreeSet` is able to maintain orderings, so probably it's also a good idea to use `TreeSet` for this problem.

```
      Set<Item> parts = new TreeSet<>();
      parts.add(i1);
      parts.add(i2);
      parts.add(i3);
      parts.add(i4);
      parts.add(i5);
      parts.add(i6);
      System.out.println(parts);
```

Ideally, executing `TreeSetTest.java` should output:

```
[[no=2, desc=Book], [no=2, desc=Toaster], [no=5, desc=Modem], [no=6, desc=Chair],
[no=6, desc=Laptop], [no=8, desc=Widget]]
```

But things seem to go wrong. What happened and why? How can we fix the problem to get the expected result?