COM6516
Object Oriented Programming and Software Design

The contents of this module has been developed by Adam Funk, Kirill Bogdanov, Mark Stevenson, Richard Clayton and Heidi Christensen

Practical 5

Java Collections Framework

- `List`
- `HashSet` and `TreeSet`
- Sorting and counting
- `Set` interface

List

List returned by `Arrays.asList` cannot be modified:

```
List<String> fixedList = Arrays.asList("elephant", "lion",
                                       "leopard", "tiger");
```

Copy to a modifiable list and replace each item with upper case:

```
List<String> myList = new LinkedList<String>(fixedList);
ListIterator<String> iter = myList.listIterator();
while (iter.hasNext()) {
    String animal = iter.next();
    String upperCaseAnimal = animal.toUpperCase();
    iter.remove();
    iter.add(upperCaseAnimal);
}
```

List

List returned by `Arrays.asList` cannot be modified:

```
List<String> fixedList = Arrays.asList("elephant", "lion",
                                       "leopard", "tiger");
```

Alternative — add upper case items converted from the fixed list to a new list:

```
List<String> newList = new LinkedList<String>();
for (String animal : fixedList) {
    newList.add(animal.toUpperCase());
}
```

HashSet and TreeSet

8 entries from `person.txt`:

```
David|James|34
Joey|Barton|22
Bradley|Wright-Phillips|23
Bradley|Wright-Phillips|99
Bradley|Wright-Phillips|100
Bradley|Wright-Phillips|99
Brad|Wright-Phillips|99
Andrew|Cole|34
```

repeated entry

single entry in `HashSet`

Printout by the `HashSetTest`:

```
Person[firstName=“Bradley”,surname="Wright-Phillips",age=99] …
Person[firstName=“Brad”,surname="Wright-Phillips",age=99] …
Person[firstName=“Joey”,surname="Barton",age=22] …
Person[firstName=“Bradley”,surname="Wright-Phillips",age=100] …
Person[firstName=“Bradley”,surname="Wright-Phillips",age=23] …
Person[firstName=“David”,surname="James",age=34] …
Person[firstName=“Andrew",surname="Cole",age=34] …
```

HashSet and TreeSet

8 entries from `person.txt`:

```
David|James|34
Joey|Barton|22
Bradley|Wright-Phillips|23
Bradley|Wright-Phillips|99
Bradley|Wright-Phillips|100
Bradley|Wright-Phillips|99
Brad|Wright-Phillips|99
Andrew|Cole|34
```

repeated entry

Printout by the `TreeSetTest`:

ordered differently from `HashSet`

```
Person[firstName="Joey",surname="Barton",age=22] …
Person[firstName="Andrew",surname="Cole",age=34] …
Person[firstName="David",surname="James",age=34] …
Person[firstName="Brad",surname="Wright-Phillips",age=99] …
Person[firstName="Bradley",surname="Wright-Phillips",age=23] …
Person[firstName="Bradley",surname="Wright-Phillips",age=99] …
Person[firstName="Bradley",surname="Wright-Phillips",age=100] …
```

single entry in `TreeSet`

`HashSet` and `TreeSet`

Recall that

- `HashSet` implements `Set`
- `TreeSet` implements `SortedSet`, a subinterface of `Set`

A `TreeSet` is a sorted collection:

- Elements may be inserted in any order
- Iterating through the collection returns them in sorted order
- Objects stored in a `TreeSet` must implement `Comparator` interface, so must provide a `compare` method

HashSet and TreeSet

8 entries from `person.txt`:

```
David|James|34
Joey|Barton|22
Bradley|Wright-Phillips|23
Bradley|Wright-Phillips|99
Bradley|Wright-Phillips|100
Bradley|Wright-Phillips|99
Brad|Wright-Phillips|99
Andrew|Cole|34
```

age 34

age 99

Printout by the `TreeSetTest`:

```
Person[firstName="Bradley",surname="Wright-Phillips",age=100]
Person[firstName="Bradley",surname="Wright-Phillips",age=99]
Person[firstName="David",surname="James",age=34]
Person[firstName="Bradley",surname="Wright-Phillips",age=23]
Person[firstName="Joey",surname="Barton",age=22]
```

one entry per each age

HashSet and TreeSet

(note) https://docs.oracle.com/javase/8/docs/api/java/util/TreeSet.html

```
Set<Person> people = new TreeSet<Person>();
```

constructs a new, empty tree set, sorted according to the natural ordering of its elements, while

```
Set<Person> people = new TreeSet<Person>(new AgeComparator());
```

uses `AgeComparator` to order its elements:

```
public class AgeComparator implements Comparator<Person> {
    public int compare(Person a, Person b) {
        return b.getAge() - a.getAge();
    }
}
```

Sorting words

Case insensitive sorting of words:

```
    words.sort(new CaseInsensitiveComparator());
```

and the `CaseInsensitiveComparator` class is defined as below:

```
class CaseInsensitiveComparator implements Comparator<String> {
    @Override
    public int compare(String o1, String o2) {
        return o1.toLowerCase().compareToIgnoreCase(o2.toLowerCase());
    }
}
```

Counting words

Assume that `words` is already a sorted list of words when the following code starts:

```
Iterator<String> i1 = words.iterator();
String previous = null;
String current;
int counter = 0;
while (i1.hasNext()) {
    current = i1.next();
    if (current.equals(previous)) {
        counter ++;
    }
    else {
        if (previous != null) {
            System.out.println(previous + "  count = " + counter);
        }
        previous = current;
        counter = 1;
    }
}
```

`Set` interface

Both `list0` and `list1` contain some words:

```
List<String> test = eitherNotBoth(list0, list1);
```

and the `eitherNotBoth` method uses the `Set` interface:

```
public static <T> List<T> eitherNotBoth(List<T> listA, List<T> listB) {
    Set<T> union = new HashSet<T>(listA);
    union.addAll(listB);
    Set<T> intersection = new HashSet<T>(listA);
    intersection.retainAll(listB);
    Set<T> result = new HashSet<T>();
    result.addAll(union);
    result.removeAll(intersection);
    return new ArrayList<T>(result);
}
```