
Australian
National
University
Introductory Java 14
J14
forEach
Ordering Collections
Introduction to Software Systems 1110/1140/1510/6710

forEach

- Collections implement the `forEach` method, which applies an action to every element in the collection.

  Instead of:

```
for (Thing t : things) {
   System.out.println(t);
}
```

  You can do this:

```
things.forEach((t) -> System.out.println(t));
```

Ordering Collections

- The `Comparable` interface defines a 'natural' ordering for all instances of a given type, `T`:

```
public interface Comparable<T> {
    public int compareTo(T o);
}
```

The return value is either -ve, `0`, or +ve according to whether the receiver comes before, equal, or after the argument, `o`.

- The `Comparator` interface allows a type T to be ordered in additional ways:

```
public interface Comparator<T> {
    int compare(T o1, T o2);
}
```

Collections.sort()

- No arguments
  - uses *natural* order for type
- Single Lambda argument:
  - uses order defined by lambda expression
  - `(a T, b T) -> { return <expression>;}`

Josh Bloch Item 25: Prefer lists to arrays

- Why?
  - Arrays are covariant, Generics are invariant
    - if `A extends B`, then `A[]` is a subclass of `B[]`
    - but `List<A>` has no relationship to `List<B>`

```
// Fails at runtime!
Object[] objectArray = new Long[1];
objectArray[0] = "I don't fit in";       // Throws ArrayStoreException

// Won't compile!
List<Object> ol = new ArrayList<Long>(); // Incompatible types
ol.add("I don't fit in");
```