1 Collection Framework

Overview: http://docs.oracle.com/javase/8/docs/technotes/guides/collections/
Tutorial: http://docs.oracle.com/javase/tutorial/collections/
API Docs: http://docs.oracle.com/javase/10/docs/api/

- Since JDK 1.2
- Since JDK 1.5 (Java 5) with generics
- Since JDK 1.8 (Java 8) with support for lambda expressions

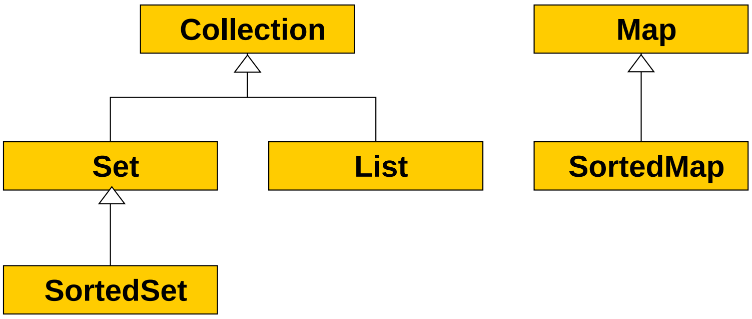
Definitions

Collection
- A *collection* is represented by an instance which combines several objects of a particular type to an entity and which administrates this entity.
- Collections are used to store and retrieve data and to pass data from one method to another.

Framework
1. Set of interfaces — *User view*
   - Set (no duplicates)
   - List (index access supported)
   - Map (key access supported)
2. Set of classes which *implement* the interfaces — *Implementation view*
   - Arrays, linear lists, trees
3. Algorithms which *use* the interfaces — *Generic algorithms*
   - Searching, sorting

↳ Can be extended with new implementation classes and new generic algorithms.

Interfaces:



The designers of the collection framework kept the set of interfaces intentionally small. The following aspects were *not* modeled with special interfaces:
- Immutability (no add/remove)
- Extendability only (no remove)
- Support of `null` as an argument

Solution:
The methods declared in the interfaces of the collection framework may throw runtime exceptions:
- optional methods may throw an *UnsupportedOperationException*
- methods with a restricted value range may throw an *IllegalArgumentException*

Collection

```
interface Collection<E> extends Iterable<E> {
   int        size();
   boolean    isEmpty();

   boolean    contains(Object x);
   boolean    containsAll(Collection<?> c);

   boolean    add(E x);
   boolean    addAll(Collection<? extends E> c);

   boolean    remove(Object x);
   boolean    removeAll(Collection<?> c);
   boolean    retainAll(Collection<?> c);
   void       clear();

   Object[]   toArray();
   <T> T[]    toArray(T[] a);

   Iterator<E>  iterator();
    // default methods: forEach, spliterator, parallelStream, removeIf, stream
}
```

Iterator

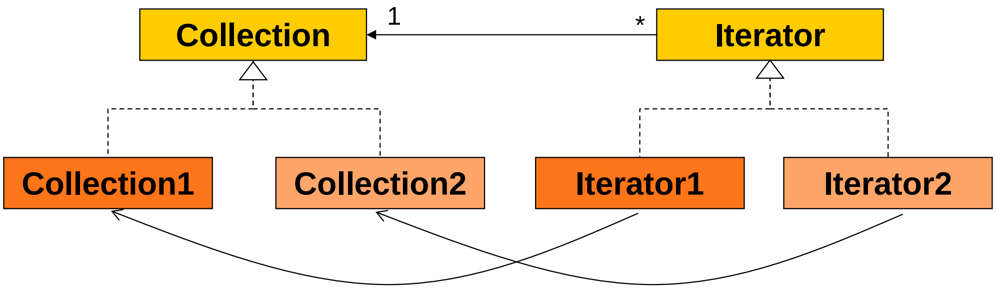
```
interface Iterator<E> {
   boolean    hasNext();
   E          next();
   void       remove();    // default: throws UnsupportedOperationException
}
```

Advantages:

- Several access paths into a collection
- Iterators can be specialized, e.g. an iterator which only returns odd numbers

Disadvantage:

- For each concrete collection implementation a special iterator has to be implemented (as the iterator typically accesses the concrete implementation)

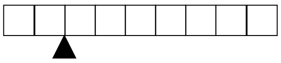


Application: Printing a collection

```
public static void print(Collection<?> c) {
   Iterator<?> it = c.iterator();
   System.out.print("[");
   while (it.hasNext()) {
      System.out.print(it.next());
      if(it.hasNext()) System.out.print(",");
   }
   System.out.println("]"),
}
```

⇨ Such a generic algorithm works with *all* collection instances!

Iterator



Conceptionally an iterator refers to a position *between* two elements.

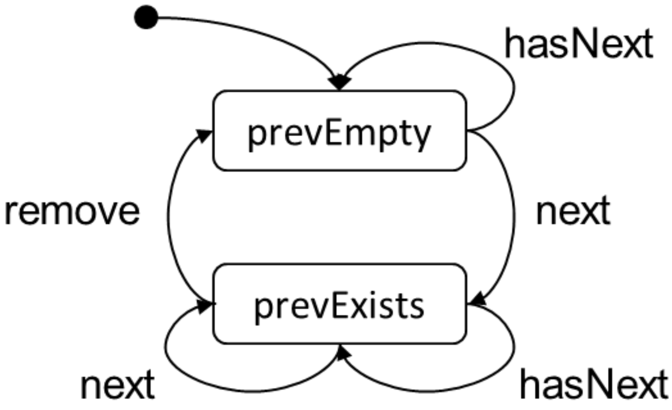
For a collection with *n* elements there are *n+1* possible positions for an iterator.

Methods:

`hasNext` = indicates whether there is another element which can be skipped, i.e. whether the iterator can be moved forward.

`next` = moves the iterator to the next position and returns the skipped element (helpful interpretation for the interface `ListIterator` which provides the additional methods `hasPrevious` and `previous`).

`remove` = removes the object which was returned by the last invocation of either method `next` or `previous`, i.e. that object does not have to be searched again.
=> before remove can be called either next (or previous) must have been called.



If the invocation is not allowed according to the above state diagram, then an `IllegalStateException` is thrown.

Implementing Collections

In order to implement a collection interface, all methods (which do not have a default implementation) have to be implemented.

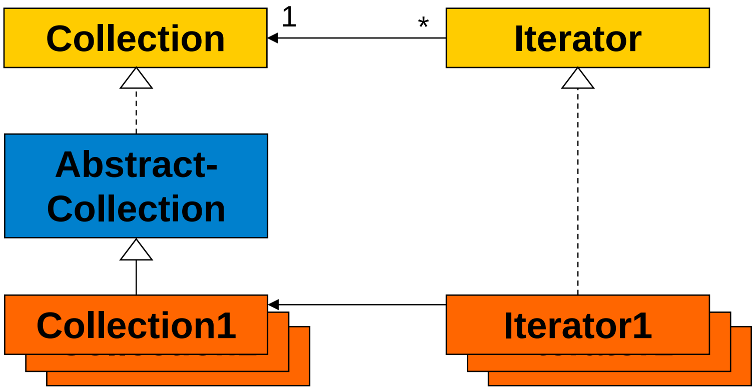
An abstract base class may help as it can provide default implementations in terms of other methods declared in the interface (such default implementations could also be provided as default methods in the interface). These methods do not make any assumptions about the representation of the collection itself.

Tasks:
Which methods can be implemented in an abstract class `AbstractCollection` (or as Java 8 default methods) in terms of the other methods?

```
abstract class AbstractCollection<E> implements Collection<E> {
   ...
}
```

Concrete implementations:
In order to program a concrete collection class the following tasks have to be done:

- Definition of the internal representation (data structure) for the collection.
- Extending the abstract class `AbstractCollection` and implementing the iterator (which accesses the internal representation directly)
- Optionally: replace some inherited default implementations by more efficient ones.


Collection
1
*
Iterator
Abstract-
Collection
Collection1
Iterator1