Collections and Generic Types

The Collections framework

- It was designed to meet several goals:
  - Storing and manipulating groups of objects
  - Had to be high-performance
  - Had to allow different types of collections to work in a similar manner and with good interoperability
  - Had to extend or adapt a collection easily
- Towards this end, the entire collections framework was designed around a set of standard interfaces.
- All collections frameworks contain the following:
  - Interfaces. Abstract data types that represent collections.
  - Classes. The concrete implementations of the collection interfaces
  - Methods. They perform useful computations, like searching or sorting.

The Collection Super Interface

- The foundation upon which the collections framework is built.
- It declares the core methods that all collections will have in common.
- Several of these methods can throw UnsupportedOperationException
- See examples for each method

boolean add(Object obj)	Adds obj to the invoking collection. Returns true if obj was added.
boolean addAll(Collection c)	Adds all the elements of c to the invoking collection. Returns true if it succeeds.
void clear()	Removes all elements from the invoking collection
boolean contains(Object obj)	Returns true if obj is an element of the invoking collection
boolean containsAll(Collection c)	Returns true if the invoking collection contains all elements of c
boolean equals(Object obj)	Returns true if the invoking collection and obj are equal.
int hashCode()	Returns the hash code for the invoking collection
boolean isEmpty()	Returns true if the invoking collection is empty
Iterator iterator()	Returns an iterator for the invoking collection
boolean remove(Obj obj)	Removes one instance of obj from the invoking collection. Returns true if the element was removed
boolean removeAll(Collection c)	Removes all elements of c from the invoking collection. Returns true if the collection changed

boolean retainAll(Collection c)	Removes all elements from the invoking collection except those in c. Returns true if the collection changed
int size()	Returns the number of elements held in the invoking collection
Object[] toArray()	Returns an array that contains all the elements stored in the invoking collection. The array elements are copies of the collection elements.
Object[] toArray(Object array[])	Returns an array containing only those collection elements whose type matches that of array.

Lists

- The List interface extends Collection
- Elements can be inserted or accessed by their position in the list, using a zero-based index.
- A list may contain duplicate elements.
- In addition to the methods defined by Collection, List defines some of its own.
- Several of the list methods will throw an UnsupportedOperationException if the collection cannot be modified, and a ClassCastException is generated when one object is incompatible with another.

Sets

- A Set is a Collection that cannot contain duplicate elements. It models the mathematical set abstraction.
- The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited
- SortedSet: It extends Set and declares the behavior of a set sorted in an ascending order.

Maps

- The Map interface maps unique keys to values.
- Given a key and a value, you can store the value in a Map object. After the value is stored, you can retrieve it by using its key.

Iterator

- Abstract mechanism for cycling through the elements in a collection
    - Defines hasNext() and next() methods
    - Throws ConcurrentModificationException if underlying collection is modified
    - Also provides remove() method

Collection Algorithms

- The collections framework defines several algorithms that can be applied to collections and maps.
- These algorithms are defined as static methods within the Collections class.
- Notable methods:
    - Collections.copy(list1, list2) - Copy a collection to another
    - Collections.reverse(list) - Reverse the order of the list
    - Collections.shuffle(list) - Shuffle the list
    - Collections.sort(list) - Sort the list (ascending)

Collections and Generic Types

- Since Java5 it's possible to add Generic Types as arguments to Collection objects
- Usage: Collection<Type> name = new Collection<Type>();
- Used to avoid creating collections with Raw types.
- Generic type syntax:
    - <?> means any type
    - <? extends E> means any subtype of E