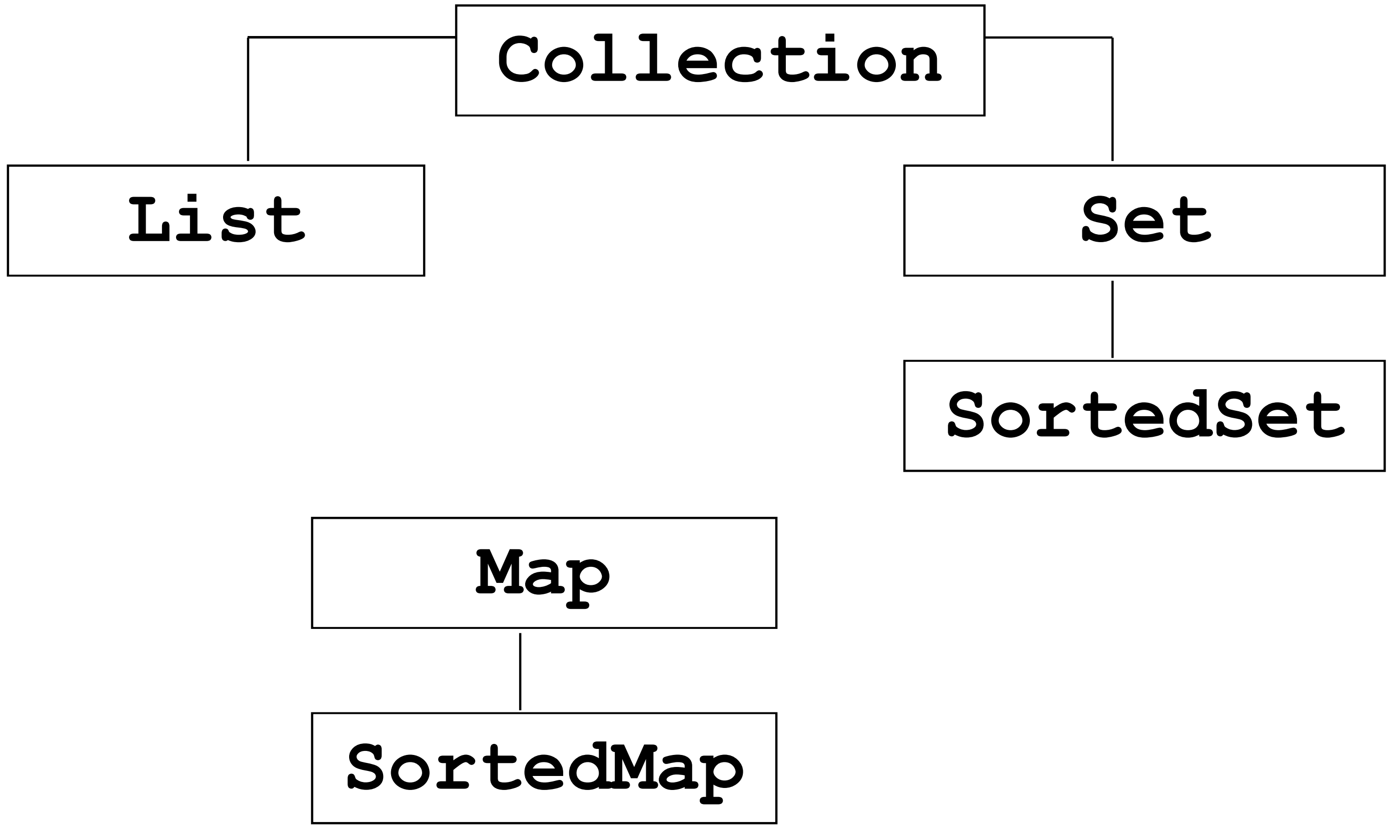
Collections Framework

By Rahul Barve

Collections Framework



Collection

By Rahul Barve

Collection

- It is the root interface in the hierarchy.
- Represents a group of objects known as elements.
- Provides generic utility methods to work upon different types of collections.

List

`List`

- It is inherited from `Collection`.
- It is an ordered collection (Index Based) and permits duplicate values.

List

- It has several implementations like:
  - `Stack`
  - `Vector`
  - `ArrayList`
  - `LinkedList`

Iterating Over Collections

Iterating Over Collections

- It is possible to iterate over a collection using a for loop or an iteration API.

Iterating Over Collections

- Java provides several interfaces to iterate over collections:
  - `Iterator`
  - `ListIterator`
  - `Enumeration`

Iterating Over Collections

- `Iterator` interface provides following methods:
  - `hasNext()`
  - `next()`
  - `remove()`

Iterating Over Collections

- `ListIterator` is an extention to `Iterator`.
- It provides additional methods like `hasPrevious()` and `previous()` to perform reverse traversal.

Iterating Over Collections

- `Enumeration` is a legacy interface provides following methods:
  - `hasMoreElements()`
  - `nextElement()`

Generics

Generics

- Generics is a newly added feature since java version 1.5, which allows developers to create classes and methods that work with objects of any type.
- Generics also allows to create type-safe collections.

Generics

- A generic notation is denoted using a pair of angular brackets `'<>'`.
- Typically it is used for interfaces and the implementation class specifies the actual type.

Generics

E.g.

```
public interface Test<T> {
     boolean doTest(T t);
}
```

Generics

```
public class NameTest implements Test<String> {
    boolean doTest(String s){…}
}
public class AgeTest implements Test<Integer> {
    boolean doTest(Integer i){…}
}
```

Type Safe Collections

Type Safe Collections

- The generic feature is also used in case of type safe collections.
- Type safe collections ensure that every element is of the specified type only.

Type Safe Collections

- Early type checking is possible at compilation time.
- Explicit cast is not required while retrieving objects from collection.

Type Safe Collections

- `List<string> cities =`

  ```
      new ArrayList<String>();
  ```
- Instructs compiler that collection `cities` can accept only objects of type `String`.

Type Safe Collections

- Therefore, `cities.add(100)` results into a compilation error.
- No casting is required while retrieving the data.

```
String firstCity = cities.get(0);
```