Introduction to Collections Framework

By Rahul Barve

Introduction to Collections Framework

- A collection is a data structure that contains different types of objects.
- The framework provides interfaces that have some contract or behavior which is applicable to the relevant collection.

Array Vs. Collection

By Rahul Barve

Array Vs. Collection

Array

- Contains elements of similar types.
- Has a fixed dimension. Cannot be resized.
- Can work upon primitives as well as object types.

Collection

- Can contain elements of different types.
- Grows dynamically as elements are added.
- Works only upon object types.

Legacy Classes

Legacy Classes

- Java library provides several predefined classes right from the first version to handle collection specific functionalities.
- Collection specific APIs belong to a package `java.util`.

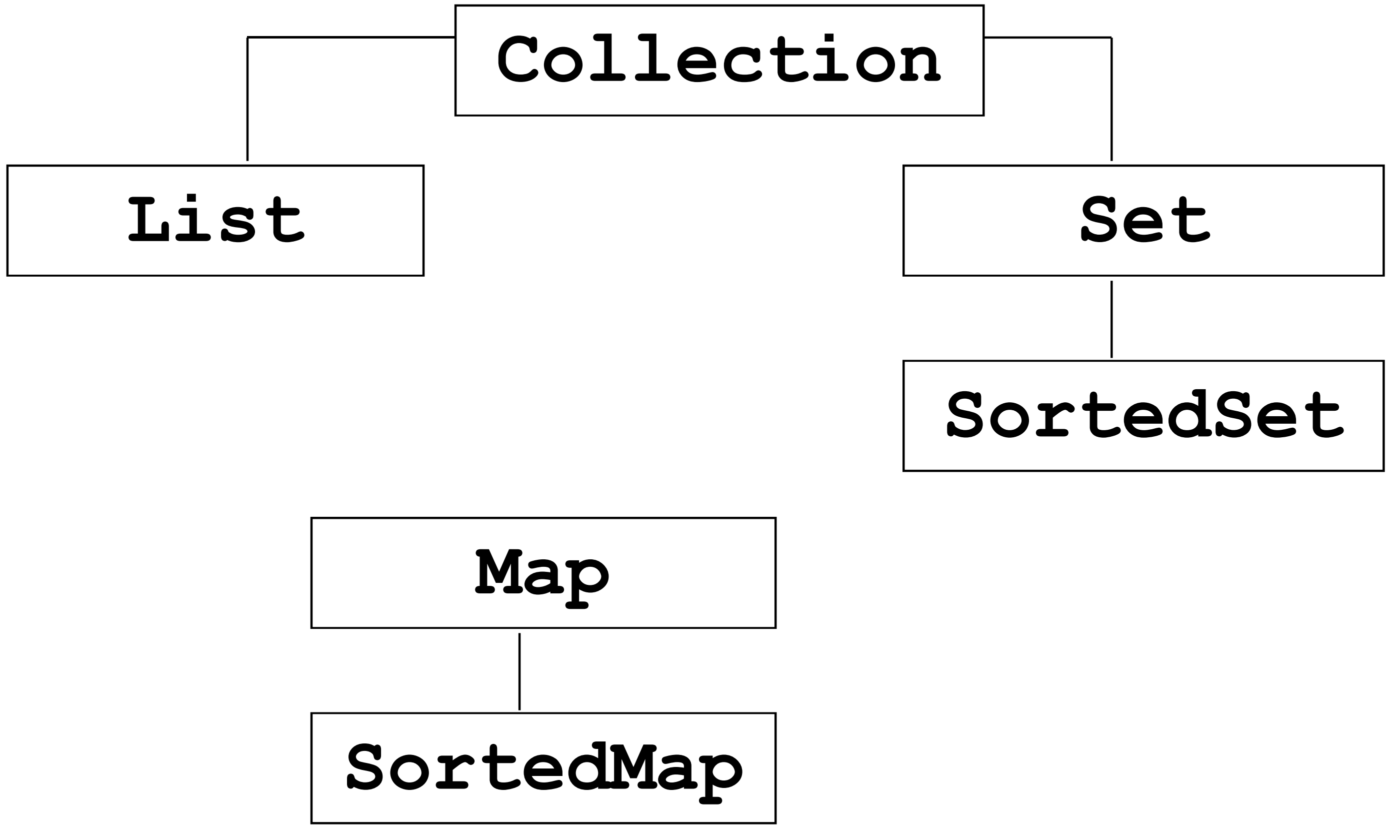
Legacy Classes

- Stack
- Vector
- Dictionary
- Hashtable
- Properties

Collections Framework

By Rahul Barve

Collections Framework



Collection

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`

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);
```

Set

Set

- It is also inherited from `Collection`.
- It is an unordered collection and prevents duplicate values.

Set

- It is implemented by `HashSet`.
- Uses a hashing algorithm instead of index to store the elements.

Set

- To acquire appropriate behavior of `Set`, the element specific class must override `hashCode()` and `equals()`.

More on `hashCode()` and `equals()`

By Rahul Barve

More on `hashCode()` and `equals()`

- If two objects are equal, their hash codes are always equal whereas if two objects are unequal still they may have the same hash code.