accenture
Latvia ATC

Test Automation Engineering Fundamentals: Java

Module 9: Collections

GROW
CONFI
DENTLY

Module Objectives

accenture
Latvia ATC

- At the end of this module, participants will be able to:
  - Define collections
  - Explain List
  - Explain Map
  - Explain HashSets

Java Collections

accenture
Latvia ATC

- The Java Collections API's provide Java developers with a set of classes and interfaces that makes it easier to handle collections of objects.
- In a sense Collection's works a bit like arrays, except their size can change dynamically, and they have more advanced behavior than arrays.

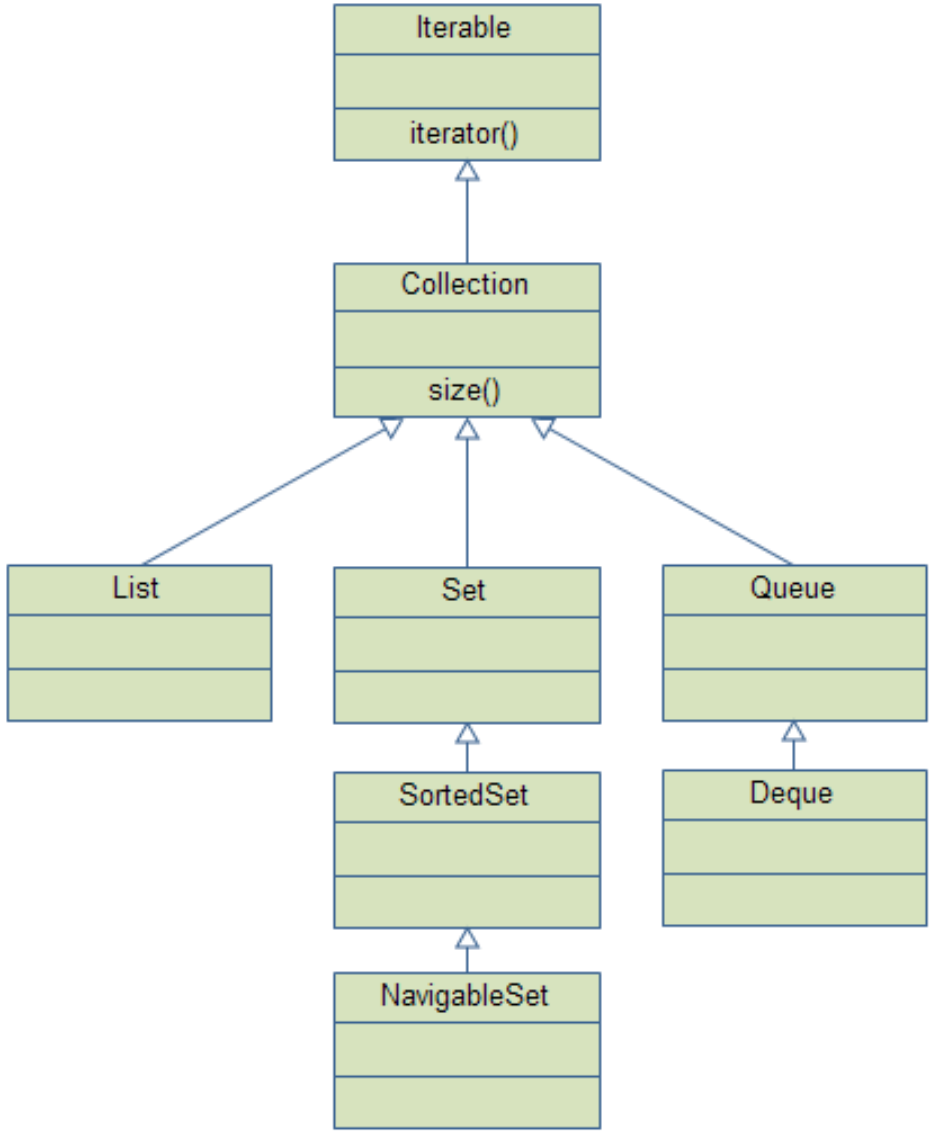
Collections types

accenture
Latvia ATC

- The following interfaces (collection types) extends the Collection interface:
  - List
  - Set
  - SortedSet
  - NavigableSet
  - Queue
  - Deque

Java Collections example

accenture
Latvia ATC



Java Collections - Iterable

accenture
Latvia ATC

- The Iterable interface (*java.lang.Iterable*) is one of the root interfaces of the Java collection classes. The Collection interface extends Iterable, so all subtypes of Collection also implement the Iterable interface.

- The Collection interface just defines a set of methods (behaviour) that each of these Collection subtypes share. This makes it possible ignore what specific type of Collection you are using, and just treat it as a Collection.

Java Collections - Iterable

accenture
Latvia ATC

- The Iterable interface (*java.lang.Iterable*) is one of the root interfaces of the Java collection classes. The Collection interface extends Iterable, so all subtypes of Collection also implement the Iterable interface.

```
List list = new ArrayList();
for(int i=0; i<list.size; i++)
{
        Object o = list.get(i);
        //do something o;
        // o -> list.get(i);
}
```

Java Collections - Iterable

accenture
Latvia ATC

- The Iterable interface (*java.lang.Iterable*) is one of the root interfaces of the Java collection classes. The Collection interface extends Iterable, so all subtypes of Collection also implement the Iterable interface.

```
List list = new ArrayList();
for(int i=0; i<list.size; i++)
{
        Object o = list.get(i);
        //do something o;
        // o -> list.get(i);
}
```

```
List list = new ArrayList();
for(Object o : list)
{
        //do something o;
        // o -> list.get(i);
}
```

Collections methods

accenture
Latvia ATC

Add

collection.add("I am a part of collection");

Remove

collection.remove(1);

AddAll

- collection.addAll(List);

RemoveAll

- collection.removeAll(List);

RetainAll

Oposite to RemoveAll

Contains

True/false

collection.contains("I am a part of collection");

ContainsAll

collection.containsAll(List);

Size

collection.size; Return int = count of elements.

Generic Collections

accenture
Latvia ATC

- It is possible to generify the various Collection and Map types and subtypes in the Java collection API.

```
Collection<String> stringCollection = new HashSet<String>();
```

Java Collections - List

accenture
Latvia ATC

- It represents an ordered list of objects, meaning you can access the elements of a List in a specific order, and by an index too.
- You can also add the same element more than once to a List

```
List listA = new ArrayList();

listA.add(15);
listA.add("element 1");
listA.add("element 1");

//access via index
String element0 = (Int) listA.get(0);
String element1 = listA.get(1);
String element3 = listA.get(2);
```

Java Collections - List

accenture
Latvia ATC

- It represents an ordered list of objects, meaning you can access the elements of a List in a specific order, and by an index too. You can also add the same element more than once to a List

```
List listA = new ArrayList();

listA.add(15);
listA.add("element 1");
listA.add("element 1");

//access via index
String element0 = (Int) listA.get(0); ---------> 15
String element1 = listA.get(1); ---------> element 1
String element3 = listA.get(2); ---------> element 1
```

Java Collections - Set

accenture
Latvia ATC

- It represents set of objects, meaning each element can only exists once in a Set.

```
Set setA = new HashSet();

setA.add("element 0");
setA.add("element 1");
setA.add("element 2");

//access via index
String element0 = setA.get(0);
String element1 = setA.get(1);
String element3 = setA.get(2);
```

Java Collections - Set

accenture
Latvia ATC

- It represents set of objects, meaning each element can only exists once in a Set.

```
Set setA = new HashSet();

setA.add("element 0");
setA.add("element 1");
setA.add("element 2");

//access via index
String element0 = setA.get(0);  ---------->  element 2
String element1 = setA.get(1);  ---------->  element 0
String element3 = setA.get(2);  ---------->  element 1
```

Java Collections - Map

accenture
Latvia ATC

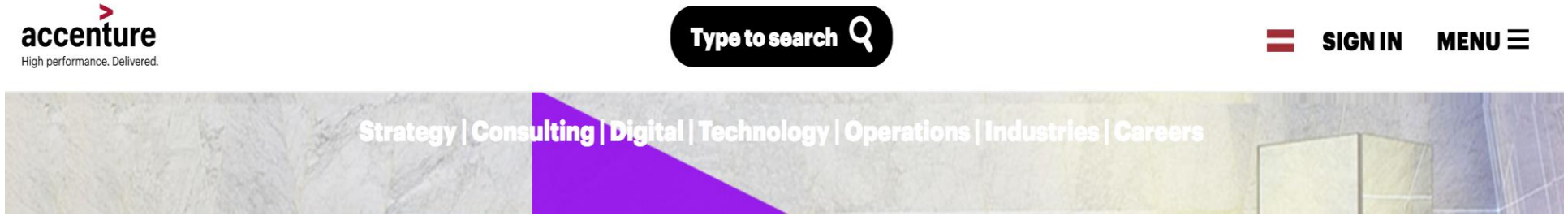
- The Map interface represents a mapping between a key and a value.
- The Map interface is not a subtype of the Collection interface.
- Therefore it behaves a bit different from the rest of the collection types.

```
Map mapA = new HashMap();
mapA.put("key1", "element 1");
mapA.put("key2", "element 2");
mapA.put("key3", "element 3");

mapA.get(”key3”); // -> element3
```

Map in real life

accenture
Latvia ATC




```
webObject search = new webObject(“search”);
webObject menu = new webObject(“menu”);
webObject signIn = new webObject(“sign in”);

Map<String, webObject> mapA = new HashMap(String, webObject);
mapA.put(”searchField", search);
mapA.put(”menuButton", menu);

mapA.get(“menuButton”); // -> webObject menu
```

Questions and Comments

accenture
Latvia ATC

- What questions or comments do you have?