
ORACLE
ACADEMY

Java Programming

2-4
Collections – Part 1

ORACLE ACADEMY

Overview

This lesson covers the following topics:

- Create a collection without using generics
- Create a collection using generics
- Implement an ArrayList
- Implement a Set

Collections

- We have seen previously that we can use arrays to store items of the same type.
- The main issue with arrays is that they have to be manually sized before use.

Collections Without Generics Example

- In the generics lesson we defined a simple cell class

```
public class Cell {
  private String data;

  public void set(String celldata)
  {
    data = celldata;
  }
  public String get() {
    return data;
  }
}
```

Collections Without Generics Example

- We could create our own simple data structure to store multiple Cell occurrences.

```
public class CellCollection {
  Cell[] cells;
  int index;
  public void add(Cell c) {
    cells[index]=c;
    index++;
  }
  public Cell get(int i) {
    return cells[i];
  }
//more methods…..
}
```

- We would have to create more methods, such as insert, sort, delete and a constructor.

Collections With Generics Example

- We could try to create a generic collection similar to our non generic example.
- We could try to modify it to use generics.

```
public class CellGenericCollection<T> {
  T[] cells;
  int index;
  public void add(T c) {
    cells[index]=c;
    index++;
  }
  public T get(int i) {
    return cells[i];
  }
}
```

Collections With Generics Example

- The problem is we should not create generic arrays in java.

```
public GenericCell(int size)
{
  cells = new T[size];
}
```

- This would produce an error.
- Java has better ways of handling generic collections.

Collections - Introduction

- Data structures are used a lot in programming and as such java comes with pre-built collection classes for you to utilize.
- These save you and every other developer the task of having to create similar storage structures.
- They come in a few guises, each with its own pros and cons.
- You should check to see if any of the pre-built collections meet your needs before building your own.

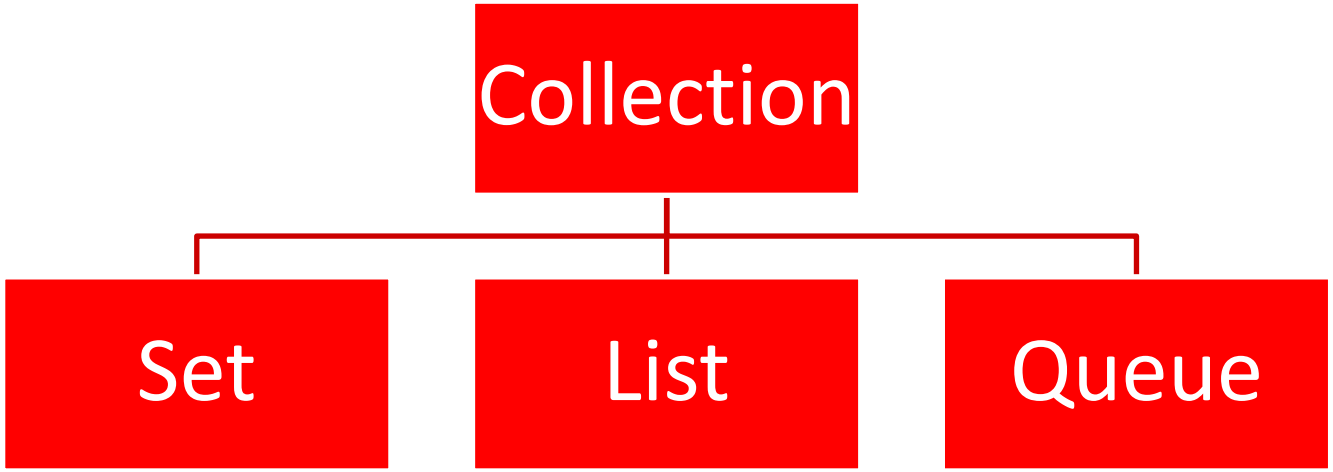
Collections

- A collection is an interface in the java.util package that is used to define a group, or collection of objects.
- It includes sets and lists.
- It is a very important part of data storing.
- Because it is in the java.util package, it will be necessary to import the java.util package into any programs you write using a collection.
- This can be done by typing the following as the first line in your program:

```
import java.util.*;
```

Collections Class Hierarchy

- Java defines three main interface collections.
- This forces any classes that use the collection interface to define its methods such as add().



Lists

- A list, in Java, is an ordered collection that may contain duplicate elements.
- In other words, List extends Collections.
- Important qualities of Lists are:
  - They grow and shrink as you add and remove objects.
  - They maintain a specific order.
  - They allow duplicate elements.

ArrayLists

- Until now we have been using arrays inside a collections class to create a collections interface.
- With arrays, you are restricted to the size of the array that you initiate inside the constructor.
- ArrayLists are very similar to arrays, except that you do not need to know the size of the ArrayList when you initialize it, like you would with an array.
- You may alter the size of the ArrayList by adding or removing elements.
- The only information you need when initializing an ArrayList is the object type that it stores.

Code to Initialize an ArrayList

- The following code initializes an ArrayList of Accounts.

```
ArrayList<Account> account= new ArrayList<Account>();
```

ArrayList Methods

- The ArrayList class already exists in Java. It contains many methods, including the following:

Method	Method Description
boolean add(E e)	Appends the specified element to the end of the list.
void add(int index, E element)	Inserts the specified element at the specified position
E get(int index)	Returns the element at the specified position.
E set(int index, E element)	Replaces the element at the specified position in the list with the specified element.
E remove(int index)	Removes the element at the specified position in the list
boolean remove(Object o)	Removes the first occurrence of the specified element from this list, if it is present.

ArrayList Activity

- Try the following activity:
  - Get into groups and initialize an ArrayList of Strings called GroupNames.
  - Add each of your names into the ArrayList.
  - Collections contains a method called sort that takes in a List as a parameter and lexicographically sorts the list.
  - An ArrayList is a List.
  - Use Collections sort method on your ArrayList of names.
  - What does your ArrayList look like after sorting it?

ArrayList Activity: Sample Answer 1

- Get into groups and initialize an ArrayList of Strings called GroupNames.
- Add each of your names into the ArrayList.

```
ArrayList<String> GroupNames;
GroupNames = new ArrayList<String>();
GroupNames.add("Mark");
GroupNames.add("Andrew");
GroupNames.add("Beth");
```

ArrayList Activity: Sample Answer 2

- Collections contains a method called sort that takes in a List as a parameter and lexicographically sorts the list.
- An ArrayList is a List.
- Use Collections sort method on your ArrayList of names.
- What does your ArrayList look like after sorting it?

```
Collections.sort(GroupNames);
GroupNames would look like:
Andrew, Beth, Mark
```

Sets

- A set is a collection of elements that does not contain duplicates.
- For example, a set of integers 1, 2, 2, 3, 5, 3, 7 would be:
  - {1, 2, 3, 5, 7}
- All elements of a set must be of the same type.
- For example, you can not have a set that includes integers and Strings, but you could have a set of integers and a separate set of Strings.

Implementing Sets with a HashSet

- Lists, which are a collection that may contain duplicates, are implemented through ArrayLists.
- Similarly, Sets are commonly implemented with a HashSet.
- A HashSet:
  - Is similar to an ArrayList, but does not have any specific ordering.
  - Is good to use when order is not important.

HashSets Coin Example

- For example, imagine you have 35 coins in a bag.
- There is no special order to these coins, but we can search the bag to see if it contains a certain coin.
- We can add to or remove coins from the bag, and we always know how many different coins are in the bag.
- Think of the bag as the HashSet.
- There is no ordering and therefore no indexes of the coins, so we cannot increment through or sort HashSets.

HashSets Coin Example

- Even if we incremented through or sorted the coins, with one little shake of the bag, the order is lost.
- HashSets have no guarantee that the order will be the same throughout time.

HashSets Coin Code Example

- The code below demonstrates the initialization of HashSet bagOfCoins.
- The HashSet bagOfCoins is a set of Coin objects.
- This assumes that the class Coin has already been created.

```
HashSet<Coin> bagOfCoins = new HashSet<Coin>();
```

Searching Through HashSets

- Although HashSets do not have ordering, we can search through them, just like we could search through the bag of coins to see if the coin we are looking for is in the bag.
- For example, to search for a coin in the HashSet bagOfCoins, use HashSet's contains(element) as demonstrated below:

```
bagOfCoins.contains(quarter);
//returns true if bagOfCoins contains the coin
```

More HashSet Methods

Method	Method Description
boolean add(E e)	Adds the specified element to this set if it is not already present.
boolean remove(Object o)	Removes the specified element from the list if present.
int size()	Returns the number of elements in the set.

Terminology

Key terms used in this lesson included:

- Generics
- Collection
- List
- ArrayList
- Set
- HashSet

Summary

In this lesson, you should have learned how to:

- Create a collection without using generics
- Create a collection using generics
- Implement an ArrayList
- Implement a HashSet


ORACLE
ACADEMY