HSBC Technology Graduate Training

Spring Boot

Day 7 (Morning)

Tuesday 3 November 2020 | 9am

Contents

ArrayList<T>

ARRAYLIST

- We've been using standard Java arrays to store values.
- The problem with using standard Arrays is that you cannot change the size of an array after it has been initialized.
- Instead, we can use a built-in collection type called ArrayList<T>.
- The <T> indicates that ArrayList<T> requires a type declaration.
- For example, if we are storing Strings in the ArrayList<T> we declare ArrayList<String>.
- Below is an example of an ArrayList<String> being initialized.

```java
public class Driver {

    public static void main(String[] args) {

        // Arrays using ArrayList<T>.
        ArrayList<String> fruits = new ArrayList<String>();

        // Traditional Arrays.
        String[] fruits = {};

    }

}
```

ArrayList<T> is a class, so we create a new ArrayList<T> just like we create a new object of our own classes.

ARRAYLIST

- When do we use ArrayList<T> and when do we use traditional Arrays?
- Generally, traditional Arrays are much faster than ArrayList<T>.
- However, ArrayList<T> is more accessible, it can change in size meaning you can add and remove elements from the ArrayList<T> after initialization and the ArrayList<T> will automatically adjust its size.
- Additionally, ArrayList<T> contains various methods that make interacting with it a lot easier.

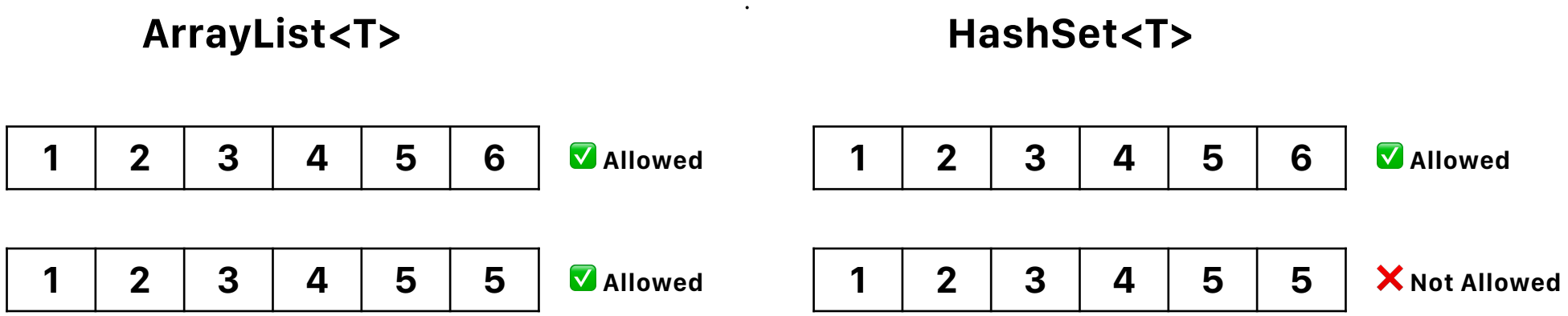
- Here are some methods that we can use on ArrayList<T> objects.

  - ArrayList<T>.contains(x): returns true if and only if the element x exists in ArrayList.
  - ArrayList<T>.add(x): adds the element x to the end of the ArrayList.
  - ArrayList<T>.remove(x): removes the first occurrence of element x from ArrayList.
  - ArrayList<T>.get(x): returns the element in the ArrayList stored at index x.
  - ArrayList<T>.clear(): clears the ArrayList.

HashSet<T>

HASHSET

- A HashSet<T> is similar to an ArrayList<T> with one key difference:
  - Duplicates are NOT allowed.



HashMap<T, E>

- In ArrayLists<T>, elements are referred to by its index position.
- For instance, to get the 3rd element in the ArrayList, we call .get(2).

```java
public class Driver {

    public static void main(String[] args) {

        // Arrays using ArrayList<T>.
        ArrayList<String> fruits = new ArrayList<String>();

        // Add elements to ArrayList.
        fruits.add("Apple");
        fruits.add("Orange");
        fruits.add("Banana");
        fruits.add("Pineapple");
        fruits.add("Grapes");

        fruits.get(0); // Returns Apple
        fruits.get(1); // Returns Orange
        fruits.get(2); // Returns Banana
        fruits.get(3); // Returns Pineapple
        fruits.get(4); // Returns Grapes

    }

}
```

HASHMAP

- A HashMap<T, E> stores Key, Value pairs.
- Technically, an ArrayList<T> stores Key, Value pairs too, but the key is the index position of the element.
- With a HashMap, we can customize our key.

Index	0	1	2	3	4
Value	6	2	8	3	2

Calling .get(2) returns 8 in an ArrayList.

Index	Chris	Ben	Ryan	Alex	Lucy
Value	6	2	8	3	2

Calling .get("Ben") returns 8 in a HashMap

HASHMAP

- When initializing a HashMap<T, E> we must provide two types T and E.
  - T is the type of the key we are going to store.
  - E is the type of the value we are going to store.

```java
public class Driver {

    public static void main(String[] args) {

        // Create HashMap<T, E>
        HashMap<String, String> favouriteFruits = new HashMap<String, String>();

        // Add elements to HashMap.
        favouriteFruits.put("Ben", "Apple");
        favouriteFruits.put("Jennifer", "Orange");
        favouriteFruits.put("Nathan", "Durian");
        favouriteFruits.put("Joseph", "Banana");
        favouriteFruits.put("Niklas", "Lychee");

        // Getting values.
        favouriteFruits.get("Joseph"); // Returns Banana

        // Removing values.
        favouriteFruits.get("Niklas"); // (Niklas, Lychee) pair removed from HashMap
    }

}
```

We must specify the types of our keys and values. In this case, names will be Strings and fruits will also be Strings.

To add a key-value pair to a HashMap, we use a slightly different method .put(x) instead of .add(x).

For Each Loops

FOR EACH LOOPS

- Creating a For loop requires you to specify 3 things:
  - (1) Initial variable state.
  - (2) Condition for loop to stop/continue.
  - (3) Code statement to be executed after loop executed.
- To print all elements inside an ArrayList<T> using a For loop, we can do the following:

```
1 public class Driver {
2
3     public static void main(String[] args) {
4
5         // Arrays using ArrayList<T>.
6         ArrayList<String> fruits = new ArrayList<String>();
7
8         // Add elements to ArrayList.
9         fruits.add("Apple");
10        fruits.add("Orange");
11        fruits.add("Banana");
12        fruits.add("Pineapple");
13        fruits.add("Grapes");
14
15                (1)        (2)          (3)
16        for (int i = 0; i < fruits.size(); i++) {
17
18            System.out.println(fruits.get(i));
19
20        }
21
22    }
23
24 }
```

FOR EACH LOOPS

- Creating a traditional For loop can be error-prone since we have to specify 3 things.
- If we wanted a loop to perform some code for each and every element of a collection such as an ArrayList<T>, we can use a ForEach loop.
- The advantage of a For Each loop is that we don't have to specify conditions.
- It will automatically loop through each element of a given collection.
- For each loop, it assigns the element to a local variable that we name.

```java
public class Driver {

    public static void main(String[] args) {

        // Arrays using ArrayList<T>.
        ArrayList<String> fruits = new ArrayList<String>();

        // Add elements to ArrayList.
        fruits.add("Apple");
        fruits.add("Orange");
        fruits.add("Banana");
        fruits.add("Pineapple");
        fruits.add("Grapes");


        for (String fruit:fruits) {

            System.out.println(fruit);

        }



}
```