Collection Types

by Sophia

WHAT'S COVERED

In this lesson, you will learn about common collection types used in Java. These collection types are used for many purposes, which include tasks like data storage and retrieval. Specifically, this lesson covers:

1. Java Collections
2. ArrayList
3. HashSet
4. HashMap

1. Java Collections

Java **collections** are data structures, similar to arrays. They allow multiple values of the same data type to be stored in a container that is accessed by name. The main difference between an array and a collection is that the size of an array is fixed. However, collections can change size.

Java collections are generic classes that take type parameters in angle brackets, like generic methods, to indicate what type of data they contain. Primitive data types require the use of wrapper types, which are similar to those in generic methods.

Here is the list presented when we covered generic methods:

Primitive Type	Wrapper Type
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer

© 2024 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

long	Long

When working with collections, keep in mind that the collection needs to be created before data can be added to the collection. Java features a number of different collection types. Commonly used collection types will be discussed in future tutorials.

TERM TO KNOW

Collection
A collection is a container that allows multiple values of the same data type to be stored. The main difference between an array and a collection is that the size of an array is fixed, but collections can change size.

---

2. ArrayList

As the name indicates, an **ArrayList** is a flexible list collection. It is similar to an array. Like an array, an ArrayList can contain duplicate values. The only constraint is that the values must all be of the same type (as with a plain array).

To use an ArrayList collection in a program, include the following import statement near the top of the file.

EXAMPLE

```
import java.util.ArrayList;
```

When using an `ArrayList`, the collection needs to be created first and then values added to it.

To create an `ArrayList`, use statement like this:

EXAMPLE

```
ArrayList<Integer> list = new ArrayList<Integer>();
```

This constructs an `ArrayList` that will hold `Integer (int)` values.

As with a generic method, the data type is specified in angle brackets:

EXAMPLE

`ArrayList<Integer>` and new `ArrayList<Integer>();`

KEY CONCEPT

© 2024 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

Since the `<Integer>` on the left side of the equal sign clearly indicates the data type, current versions of Java allow leaving out the type on the right side of the equal sign.

This statement does the same thing:

EXAMPLE

```
ArrayList<Integer> list = new ArrayList<>();
```

To put values into the `ArrayList`, use the `add()` method like this:

EXAMPLE

```
list.add(100);
list.add(99);
list.add(98);
```

To access a value in an `ArrayList`, use the `get()` method. The `get()` method takes one parameter. This parameter includes the index of the desired list item. The first item in the list has the index 0, which is consistent with array indices. The last item in the list has an index that is one less than the size of the `ArrayList`. With the number of elements in the array provided by the length property, when working with an ArrayList, the `size()` method is used to get the number of items in the ArrayList.

TRY IT

Directions: To see how an `ArrayList` works, type the following code in the IDE in a file named ArrayListScores.java:

```
import java.util.ArrayList;

class ArrayListScores {
  public static void main(String[] args) {
    // Construct an ArrayList named scores to hold Integer values
    ArrayList<Integer> scores = new ArrayList<>();
    // Add some scores
    scores.add(99);
    scores.add(88);
    scores.add(100);
    scores.add(85);
    System.out.println("First score: " + scores.get(0));
    int listLength = scores.size();
    System.out.println("Last score: " + scores.get(listLength - 1));
```

© 2024 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

```
  }
}
```

The program you ran should produce this output:

```
First score: 99
Last score: 85
```

REFLECT

An ArrayList collection provides a more flexible kind of array, since the size of the collection is not fixed and can grow as items are added at runtime. Additional items automatically appear at the end of the list even if there are repeating items.

There are a number of other similarities between arrays and `ArrayLists`. To print out the values in an `ArrayList`, you would use a `toString()` method.

KEY CONCEPT

Though it is part of the `ArrayList` class itself, it is not part of the `Collections` utilities.

The program demonstrates this:

```
import java.util.ArrayList;

class ArrayListScores {
  public static void main(String[] args) {
    // Construct an ArrayList named scores to hold Integer values
    ArrayList<Integer> scores = new ArrayList<>();
    // Add some scores
    scores.add(99);
    scores.add(88);
    scores.add(100);
    scores.add(85);
    System.out.println("Scores list: " + scores.toString());
    System.out.println("First score: " + scores.get(0));
    int listLength = scores.size();
    System.out.println("Last score: " + scores.get(listLength - 1));
  }
}
```

The results should look like this:

```
Scores list: [99, 88, 100, 85]
```

© 2024 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

```
First score: 99
Last score: 85
```

Similar to the way an array can be sorted using the `Arrays.sort()` method, there is a corresponding `Collections.sort()` method that can be called like this:

EXAMPLE

```
Collections.sort(scores);
```

The sort is done using the "natural order," which is similar to an array. It also uses the numeric order for numeric data types and alphabetic order for character and string data.

See how to use a sorted list to get the lowest and highest values in the ArrayList below:

```
import java.util.ArrayList;
import java.util.Collections;

class ArrayListScores {
  public static void main(String[] args) {
    // Construct an ArrayList named scores to hold Integer values
    ArrayList<Integer> scores = new ArrayList<>();
    // Add some scores
    scores.add(99);
    scores.add(88);
    scores.add(100);
    scores.add(85);
    System.out.println("First score: " + scores.get(0));
    int listLength = scores.size();
    System.out.println("Last score: " + scores.get(listLength - 1));
    Collections.sort(scores);
    System.out.println("Lowest score: " + scores.get(0));
    System.out.println("Highest score: " + scores.get(listLength - 1));
  }
}
```

Running this code shows ArrayListScores.java 85-100 as the output.:

```
First score: 99
Last score: 85
Lowest score: 85
Highest score: 100
```

TERM TO KNOW

© 2024 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

ArrayList
An ArrayList is a flexible list collection that has similarities to an array, except that its size is not fixed.

---

3. HashSet

A **HashSet** is a Java collection that stores unique values. A `HashSet` never contains duplicate values. Adding a duplicate value to a `HashSet` has no effect. After a `HashSet` has been declared, values of the appropriate type are added using the `add()` method. There is also a `remove()` method that removes the value passed as a parameter.

The following code shows the use of the `size()` method to get the number of items in the `HashSet`. The program also demonstrates the use of the `HashSet`'s `contains()` method.

This program will return a boolean value, or a true if the value passed in is present; otherwise, it will return false.

```
import java.util.HashSet;
import java.util.Scanner;

public class PetsHashSet {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    HashSet<String> petSet = new HashSet<>();
    petSet.add("hamster");
    petSet.add("cat");
    petSet.add("fish");
    petSet.add("dog");
    petSet.add("dog"); // duplicate accepted but not kept
    System.out.println("There are " + petSet.size() + " pets in the HashSet.");
    System.out.println("Pets: " + petSet.toString());

    System.out.print("Enter a type of pet: ");
    String pet = input.nextLine();
    // convert entry to lowercase
    pet = pet.toLowerCase();
    // contains() returns true if value is present in set, otherwise false
    if(petSet.contains(pet)) {
      System.out.println("The HashSet contains " + pet + ".");
    }
    else {
      System.out.println("The HashSet does not contain " + pet + ".");
```

© 2024 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

```
    }
  }
}
```

The output for this program should look like the following output screens:

EXAMPLE The output screen shows PetsHashSet.java as the output:

```
There are 4 pets in the HashSet.
Pets: [hamster, cat, fish, dog]
Enter a type of pet: dog
The HashSet contains dog.
```

EXAMPLE The output screen shows PetsHashSet.java no bird as the output:

```
There are 4 pets in the HashSet.
Pets: [hamster, cat, fish, dog]
Enter a type of pet: bird
The HashSet does not contain bird.
```

KEY CONCEPT

It is important to note that a `HashSet` does not provide any mechanism to access individual items in the set. It can only determine if a `HashSet` contains a given value.

TERM TO KNOW

HashSet
A HashSet is a Java collection that stores unique values (i.e., there are no duplicate values).

---

4. HashMap

Java includes many collection types, but the last specific type that will be considered in this tutorial is the **HashMap**. Each element in a HashMap contains two values. These include the key value that identifies a unique element. The second is the value that is associated with the key. Since each element contains two values and two data types, they need to be specified when declaring a HashMap. The first identifies the key and the second identifies the value.

The code below declares the HashMap like this:

EXAMPLE

```
HashMap<String, Integer> scores = new HashMap<>();
```

© 2024 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

In this case, the key is a student or user ID that is a string value and the value is an integer value representing the score for the associated ID.

After a `HashMap` has been declared, the key-value pairs are added using the `put()` method. To access an item in the `HashMap`, the `get()` method is used. This method takes a parameter for passing in the key value being sought. The `get()` method returns `null` (no value) if the key is not present in the `HashMap`. The code below also shows the use of the `contains()` method, which returns true or false, and allows the programmer to check if a key is found in the `HashMap`.

TRY IT

Directions: To see how a `HashMap` collection works, type in the following code in the IDE in a file named ScoresHashMap.java:

```
import java.util.HashMap;
import java.util.Scanner;

public class ScoresHashMap {

  public static void main(String[] args) {
    // HashMap holds key-value pairs.
    // The key (user ID) is a String (case sensitive).
    // The value (score) is an Integer (int)
    HashMap<String, Integer> scores = new HashMap<>();
    scores.put("ssmith04", 88);
    scores.put("tlang01", 100);
    scores.put("glewis03", 99);
    System.out.println("Scores: " + scores.toString());

    Scanner input = new Scanner(System.in);

    System.out.print("Enter an ID: ");
    String id = input.nextLine();
    // Check if the HashMap contains the key (id)
    if(scores.containsKey(id)) {
      // Only safe to use get() to retrieve value if key exists in HashMap
      int score = scores.get(id);
      System.out.println(id + " has a score of " + score + ".");
    }
    else {
      System.out.println("There is no score for " + id + ".");
    }
```

© 2024 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

```
    }
}
```

The output tables below are the return of a couple sample runs of the program:

EXAMPLE The output screen shows ScoresHashMap.java as the output.

```
Scores: {ssmith04=88, glewis03=99, tlang01=100}
Enter an ID: tlang01
tlang01 has a score of 100.
```

EXAMPLE The output screen shows ScoresHashMap.java as the output.

```
Scores: {ssmith04=88, glewis03=99, tlang01=100}
Enter an ID: cjones03
There is no score for cjones03.
```

REFLECT

This example has shown how to use a Java HashMap collection to store data that is organized into key-value pairs. As with any collection, a HashMap doesn't have a fixed size (though it cannot contain duplicate keys).

TERM TO KNOW

HashMap

A HashMap is a Java collection that stores key-value pairs. The key is a unique value that identifies the pair, and the value is the data associated with the key.

SUMMARY

In this lesson, you have learned about Java collection types. You learned that **Java collections** are an important data construct in Java, since they can contain a flexible range of data. You discovered that, unlike arrays, collections do not have fixed sizes. You also learned that there are a large number of collection types in Java that are useful for storing data in a wide range of programming constructs. Finally, you learned that a few of the most commonly used types are **ArrayList**, **HashSet**, and **HashMap**.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source py4e.com/html3/

TERMS TO KNOW

ArrayList

© 2024 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

An ArrayList is a flexible list collection that has similarities to an array, except that its size is not fixed.

Collection

A collection is a container that allows multiple values of the same data type to be stored. The main difference between an array and a collection is that the size of an array is fixed, but collections can change size.

HashMap

A HashMap is a Java collection that stores key-value pairs. The key is a unique value that identifies the pair, and the value is the data associated with the key.

HashSet

A HashSet is a Java collection that stores unique values (i.e., there are no duplicate values).

© 2024 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.