**Pair in Java**

**Overview**

**A Pair is a really useful data structure which is used to store two values together. Pairs are useful when we need to return two values from a method which are associated with each other such as a Student name and marks, a number and its square, etc.**

Java supports the Pair data structure using the Pair class available in javafx.util package post Java 8. Pair class in Java stores paired data in a key-value pair combination, this combination is also known as tuple.

**Introduction to Java Pair Class**

Since Java 8, Java provides a Pair class to store a pair of values together. These two values can be of any data type independent of each other. For example, one value can be of String type, and the second one can be an Integer.

Now, let's see how we can use the in-built Pair class in Java.

**1. Importing Pair class**

For importing Pair class in Java we use:

import javafx.util.Pair;

**2. Declaring a Pair object**

Let us now see how to declare a Pair object in Java.

**Syntax:**

Pair<Key\_Type, Value\_Type> pairName = new Pair<>(key, value);

**Example:**

Pair<Integer, String> pr = new Pair<>(7, "Hello");

As a result of this statement, a Pair object of the type <Integer, String> is created. The constructor will take the value 7 and Hello and store them in the Pair object pr.

**3. Accessing values**

Using getKey() and getValue() methods we can access the two values stored in a Pair object.

1. **getKey()**: gets the first value.
2. **getValue()**: gets the second value

Now, let's see an example of implementing Pair class in Java.

*// Java program to implement in-built Pair class*

import javafx.util.Pair;

public class Main {

public static void main(String[] args) {

Pair < Integer, String > pr = new Pair <>(7, "Hello");

*// Printing the key and value pair*

System.out.println("The first value is: " + pr.getKey());

System.out.println("The second value is: " + pr.getValue());

}

}

**Output:**

The first value is: 7

The second value is: Hello

**Explanation:**

As we can see, using the getKey() method, we can access the first value in the pair and using the getValue() method, we can access the second value. Both these methods are defined in the Pair class to access the first and second field values of a Pair object.

**Note:**

Here, <Key, Value> simply refers to a pair of values that are stored together.

**Do not confuse it with the <Key, Value> used in a HashMap or HashTable.**

**Methods Provided by the javafx.util.Pair Class**

Pair class in Java contains methods which make it easier to operate on Pair objects. Let's see some of these methods, their use cases and how to implement them in our Java programs.

**1. boolean equals()**

Two Pair objects are said to be equal if and only if the corresponding keys and values in both objects are equal. equals() performs a deep comparison which means only the basic values need to be equal.

**Declaration:**

public boolean equals(Object o)

**Example:**

import javafx.util.Pair;

public class PairExample {

public static void main(String[] args) {

Pair < String, Integer > pair1 = new Pair <>("apple", 1);

Pair < String, Integer > pair2 = new Pair <>("apple", 1);

Pair < String, Integer > pair3 = new Pair <>("banana", 2);

Pair < String, Integer > pair4 = new Pair <>("pineapple", 4);

*// equals() method*

System.out.println("pair1 and pair2 are equal: " + pair1.equals(pair2));

System.out.println("pair1 and pair3 are equal: " + pair1.equals(pair3));

System.out.println("pair3 and pair4 are equal: " + pair3.equals(pair4));

}

}

**Output:**

pair1 and pair2 are equal: true

pair1 and pair3 are equal: false

pair3 and pair4 are equal: false

**2. String toString()**

toString() method returns a String representation of a Pair object. toString() method in Pair class overrides the toString() method defined in Object class and returns a String view of the Pair object. The default name/value delimiter '=' is always used between the key and value of the Pair object.

**Declaration:**

public String toString()

**Example:**

import javafx.util.Pair;

import java.util.ArrayList;

public class PairExample {

public static void main(String[] args) {

Pair < String, Integer > pair1 = new Pair <>("apple", 1);

Pair < String, Integer > pair2 = new Pair <>("banana", 2);

Pair < String, Integer > pair3 = new Pair <>("pineapple", 4);

ArrayList<Pair<String, Integer>> list = new ArrayList<Pair<String, Integer>>();

list.add(pair1);

list.add(pair2);

list.add(pair3);

*//iterating over list and using toString function*

for (Pair < String, Integer > p: list) {

System.out.println(p.toString());

}

}

}

**Output:**

apple=1

banana=2

pineapple=4

**3. getKey() Method**

This method returns the key for the given specific pair in Java and is declared as:

public K getKey() *// K is the data type of the Key*

**Example:**

import java.util.ArrayList;

import javafx.util.Pair;

public class PairExample {

public static void main(String[] args) {

Pair<String, Integer> pair1 = new Pair<>("apple", 1);

Pair<String, Integer> pair2 = new Pair<>("banana", 2);

Pair<String, Integer> pair3 = new Pair<>("pineapple", 4);

ArrayList<Pair<String, Integer>> list = new ArrayList<Pair<String, Integer>>();

list.add(pair1);

list.add(pair2);

list.add(pair3);

*// iterating over list and using getKey function*

for (Pair p : list) {

System.out.println("key: " + p.getKey());

}

}

}

**Output:**

key: apple

key: banana

key: pineapple

**4. getValue() Method**

This method returns value for the given pair in Java and is declared as:

public V getValue() *// V is the data type of the Value*

Example to understand the `getValue()`` method of Pair class in Java:

import javafx.util.Pair;

import java.util.ArrayList;

public class PairExample {

public static void main(String[] args) {

Pair < String, Integer > pair1 = new Pair < > ("apple", 1);

Pair < String, Integer > pair3 = new Pair < > ("banana", 2);

Pair < String, Integer > pair4 = new Pair < > ("pineapple", 4);

ArrayList < Pair < String, Integer >> list = new ArrayList < Pair < String, Integer >> ();

list.add(pair1);

list.add(pair2);

list.add(pair3);

list.add(pair4);

*//iterating over list and using getValue function*

for (Pair p: list) {

System.out.println("key: " + p.getValue());

}

}

}

**Output:**

value: 1

value: 2

value: 4

**5. int hashCode()**

This method generates a hash code for the given pair in Java, which is calculated using the key and value associated with the Pair object. This method overrides the hashCode() method declared in the Object class.

**Declaration:**

public int hashCode()

**Example:**

import javafx.util.Pair;

import java.util.ArrayList;

public class PairExample {

public static void main(String[] args) {

Pair < String, Integer > pair1 = new Pair < > ("apple", 1);

Pair < String, Integer > pair2 = new Pair < > ("apple", 1);

Pair < String, Integer > pair3 = new Pair < > ("banana", 2);

Pair < String, Integer > pair4 = new Pair < > ("pineapple", 4);

ArrayList < Pair < String, Integer >> list = new ArrayList < > ();

list.add(pair1);

list.add(pair2);

list.add(pair3);

list.add(pair4);

for (Pair p: list) {

System.out.println("Pair: " + p.toString() + " and hashCode: " + p.hashCode());

}

}

}

**Output:**

Pair: apple=1 and hashCode: 1209379731

Pair: apple=1 and hashCode: 1209379731

Pair: banana=2 and hashCode: -972748765

Pair: pineapple=4 and hashCode: -1349826842

**Note:** The **hashCode** values for pair1 and pair2 are the same as the key, value are the same in both Pair objects.

**Table for Pair Class Methods**

| **Method** | **Return Type** | **Description** |
| --- | --- | --- |
| **getKey()** | K | Returns the key of the pair. |
| **getValue()** | V | Returns the value of the pair. |
| **hashCode()** | int | Generates a hash code for the given pair. Hashcode for equal Pair objects will be the same. |
| **toString()** | String | String representation of the pair. |
| **equals(Object o)** | boolean | If the object is not a Pair object or is null, equals() method returns false |

**Types of Pair Classes**

1. **MutablePair Class**: Mutable classes allow the value to be altered at any time during our program. getters and setters methods can be used to alter and access the value of a given object after being defined.

MutablePair is defined in the **Apache Commons Lang** which is why before using it in our program we have to make sure the package has been added to the **classpath**.

We can add the below dependencies section of pom.xml if we're using Maven.

**Maven** is a build automation tool used primarily for Java projects.

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

<version>3.12.0</version>

</dependency>

**How to import:**

import org.apache.commons.lang3.tuple.MutablePair;

**Creating MutablePair objects:**

*// using constructor method*

MutablePair<String, String> mutablePair = new MutablePair<>(

"leftValue",

"rightValue"

);

*// using of() method*

MutablePair<String, String> mutablePair = MutablePair.of(

"leftValue",

"rightValue"

);

**Update Values:**

mutablePair.setLeft("left");

mutablePair.setRight("right");

**Accessing Elements:**

String leftElement = mutablePair.left;

String rightElement = mutablePair.right;

*// OR*

String leftElement = mutablePair.getLeft();

String rightElement = mutablePair.getRight();

**Example:**

*// Java program to implement MutablePair class*

import org.apache.commons.lang3.tuple.MutablePair;

public class Main {

public static void main(String[] args) {

MutablePair < String, String > mutablePair = new MutablePair < > ();

mutablePair.setLeft("leftValue");

mutablePair.setRight("rightValue");

*/\* printing left and right elements \*/*

System.out.println("Left Element - " + mutablePair.getLeft());

System.out.println("Right Element - " + mutablePair.getRight());

}

}

**Output**

Left Element - leftValue

Right Element - rightValue

1. **ImmutablePair Class**: In the ImmutablePair class we cannot change the value of an object once it’s defined, it remains constant and the setValue() method cannot be used to alter the constant value.

Since ImmutablePair is also defined in the **Apache Commons Lang**, we need to add the package and import the same way we did in MutablePair.

Let's see how we can create an ImmutablePair object:

*/\* using constructors \*/*

ImmutablePair<String, String> immutablePair = new ImmutablePair<>("leftValue", "rightValue");

*/\*Using of() method\*/*

ImmutablePair<String, String> immutablePair = ImmutablePair.of("leftValue", "rightValue");

**Accessing Elements:**

*// Accessing left and right elements*

String leftElement = immutablePair.left;

String rightElement = immutablePair.right;

*// or*

String leftElement = immutablePair.getLeft();

String rightElement = immutablePair.getRight();

Now let's implement ImmutablePair class in Java program.

*// Java program to implement ImmutablePair class*

import org.apache.commons.lang3.tuple.ImmutablePair;

public class Main {

public static void main(String args[]) {

ImmutablePair < String, String > immutablePair = ImmutablePair.of("leftValue", "rightValue");

System.out.println("Left Element - " + immutablePair.getLeft());

System.out.println("Right Element - " + immutablePair.getRight());

}

}

**Output**

Left Element - leftValue

Right Element - rightValue

**Why do We Need Pair Class?**

The following are a few cases when we need to use the Pair class:

1. The Pair class in Java is used for managing a pair of values which are somehow associated with each other, for example, a player and score, a student and marks, etc.
2. We can use a pair class in Java for returning a pair of values from a method.
3. Pair class is especially useful while performing operations on a Tree. When we perform recursion on a tree data structure, it becomes easy to return values to the tree nodes towards the top with certain computations performed on them like a max or min operation on the subtree of a tree node.

**Custom Pair class in Java**

We can create a Generic Pair class in Java with **Java Generics**. Generics means the data types will be dynamic or parameterized types. The data type of the member fields used inside a class will be provided in the class declaration.

**Example:**

class CustomClass<T1, T2> {

T1 id, T2 var;

CustomClass(T1 id, T2 var) {

this.id = id;

this.var = var;

}

public T1 getID() {

return this.id;

}

public T2 getVar() {

return this.var;

}

}

Here we have created a Generic class which specifies the types of its member fields in its declaration.

**Object Creation:**

*// Creating a CustomClass object by calling its constructor.*

*// Notice that T1 = Integer, T2 = String*

CustomClass<Integer, String> obj = new CustomClass<Integer, String> (2, "ScalerTopics");

In a similar way, we can design a custom Pair class in Java using Java Generics. The Pair class will have its own public constructor for instantiation as we have seen above.

We can define two data fields in the Pair class i.e. first and second. The data types of first and second data fields will be defined in the Pair class declaration as shown in the code below.

We can define additional methods such as:

* getKey(): To retrieve the values of first data field.
* getValue(): To retrieve the values of second data field.
* toString(): To return a String presentation of a Pair object.

**Example:**

import java.util.ArrayList;

class Pair<F, S> {

private F first; *//first member of pair*

private S second; *//second member of pair*

public Pair(F first, S second) {

this.first = first;

this.second = second;

}

public F getKey() {

return first;

}

public S getValue() {

return second;

}

@Override

public String toString() {

return "(" + this.first + ", " + this.second + ")";

}

}

public class Main {

public static Pair<String, Integer> getManOfTheMatch(ArrayList<Pair<String, Integer>> matchDetails) {

Pair<String, Integer> manOfTheMatch = new Pair<String, Integer>("", 0);

int maxScore = Integer.MIN\_VALUE;

for (Pair<String, Integer> player : matchDetails) {

int score = player.getValue();

if (score > maxScore) {

maxScore = score;

manOfTheMatch = player;

}

}

return manOfTheMatch;

}

public static void main(String args[]) {

ArrayList<Pair<String, Integer>> match = new ArrayList<Pair<String, Integer>>();

match.add(new Pair<String, Integer>("Kohli", 65));

match.add(new Pair<String, Integer>("Rohit", 44));

match.add(new Pair<String, Integer>("Dhoni", 126));

match.add(new Pair<String, Integer>("Bhuvi", 43));

match.add(new Pair<String, Integer>("Shikhar", 98));

Pair<String, Integer> ans = getManOfTheMatch(match);

System.out.println(

ans.getKey() + " is man of the match " + "with score of " + ans.getValue()

);

}

}

**Output**

Dhoni is man of the match with score of 126

**Explanation:**

* Here we have implemented a custom generic Pair class with standard methods implemented inside them.
* Then we have created a method getManOfTheMatch() to find out the maximum scorer and return the name of the player and score with it.
* As it is visible in the data, Dhoni is the maximum scorer with the total score of 99. Hence, its the output