

A Java collection of value pairs? (tuples?)

[Ask Question](#)

I like how Java has a Map where you can define the types of each entry in the map, for example `<String, Integer>`.

What I'm looking for is a type of collection where each element in the collection is a pair of values. Each value in the pair can have its own type (like the String and Integer example above), which is defined at declaration time.

The collection will maintain its given order and will not treat one of the values as a unique key (as in a map).

Essentially I want to be able to define an ARRAY of type `<String,Integer>` or any other 2 types.

I realize that I can make a class with nothing but the 2 variables in it, but that seems overly verbose.

I also realize that I could use a 2D array, but because of the different types I need to use, I'd have to make them arrays of OBJECT, and then I'd have to cast all the time.

I only need to store pairs in the collection, so I only need two values per entry. Does something like this exist without going the class route? Thanks!

java

edited Feb 6 '09 at 17:18



toolkit

40.6k

12

91

126

asked Feb 6 '09 at 17:07



DivideByHero

6,925

20

47

59

i wonder Guava might have a class for this also. – Sikorski Jul 23 '13 at 5:31

Guava is pretty anti- Pair , and the folks at Google have gone so far as to create a much better alternative - Auto/Value. It lets you easily create *well-typed* value type classes, with proper equals/hashCode semantics. You'll never need a Pair type again! – dimo414 Dec 7 '17 at 21:06

16 Answers

The Pair class is one of those "gimme" generics examples that is easy enough to write on your own. For example, off the top of my head:

```
public class Pair<L,R> {

    private final L left;
    private final R right;

    public Pair(L left, R right) {
        this.left = left;
        this.right = right;
    }

    public L getLeft() { return left; }
    public R getRight() { return right; }

    @Override
    public int hashCode() { return left.hashCode() ^ right.hashCode(); }

    @Override
    public boolean equals(Object o) {
        if (!(o instanceof Pair)) return false;
        Pair paio = (Pair) o;
        return this.left.equals(paio.getLeft()) &&
            this.right.equals(paio.getRight());
    }
}
```

And yes, this exists in multiple places on the Net, with varying degrees of completeness and feature. (My example above is intended to be immutable.)

edited May 30 '17 at 9:40



Jarvis

2,390

2

12

27

answered Feb 6 '09 at 17:24



Paul Brinkley

4,633

3

17

31

15 I like this, but what do you think about making the left and right fields public? It's pretty clear that the Pair class is never going to have any logic associated and all clients will need access to 'left' and 'right,' so why not make it easy? – Outlaw Programmer Feb 6 '09 at 17:39

36 Um...no it wouldn't. The fields are marked as final, so they can't be reassigned. And it's not threadsafe because 'left' and 'right' could be mutable. Unless getLeft()/getRight() returned defensive copies (useless in

this case), I don't see what the big deal is. – [Outlaw Programmer](#) Feb 6 '09 at 17:46

- 15 Note that hashCode() as-is does gives the same value if left and right are swapped. Perhaps: `long l = left.hashCode() * 2654435761L; return (int)l + (int)(l >> 32) + right.hashCode();` – [karmakaze](#) Apr 28 '12 at 21:40
- 52 So if I understand correctly, rolling out a simple pair class yields a syntax error, a subpar hashCode method, null pointer exceptions, no compareTo method, design questions ... and people still advocate rolling out this class while it exists in Apache commons. Please, just copy the code if you don't want to include the JAR but stop reinventing the wheel! – [cquezel](#) Mar 18 '13 at 20:20
- 24 What do you mean "easy enough to write on your own"? That's *terrible* software engineering. Are the N^2 adapter classes to convert between MyPair and SomeoneElsesPair also considered easy enough to write on one's own? – [djechlin](#) Nov 1 '13 at 18:59

AbstractMap.SimpleEntry

Easy you are looking for this:

```
java.util.List<java.util.Map.Entry<String,Integer>> pairList= new
java.util.ArrayList<>();
```

How can you fill it?

```
java.util.Map.Entry<String,Integer> pair1=new java.util.AbstractMap.SimpleEntry<>
("Not Unique key1",1);
java.util.Map.Entry<String,Integer> pair2=new java.util.AbstractMap.SimpleEntry<>
("Not Unique key2",2);
pairList.add(pair1);
pairList.add(pair2);
```

This simplifies to:

```
Entry<String,Integer> pair1=new SimpleEntry<>("Not Unique key1",1);
Entry<String,Integer> pair2=new SimpleEntry<>("Not Unique key2",2);
pairList.add(pair1);
pairList.add(pair2);
```

And, with the help of a `createEntry` method, can further reduce the verbosity to:

```
pairList.add(createEntry("Not Unique key1", 1));
pairList.add(createEntry("Not Unique key2", 2));
```

Since `ArrayList` isn't final, it can be subclassed to expose an `of` method (and the aforementioned `createEntry` method), resulting in the syntactically terse:

```
TupleList<java.util.Map.Entry<String,Integer>> pair = new TupleList<>();
pair.of("Not Unique key1", 1);
pair.of("Not Unique key2", 2);
```

edited Nov 26 '16 at 20:38



Dave Jarvis

20.1k 30 128 245

answered Jun 28 '12 at 22:17



JavaHelp4u

2,339 1 7 4

- 6 FYI: That suggested `SimpleEntry` has a sibling class omitting `setValue` method to be immutable. Thus the name `SimpleImmutableEntry`. – [Basil Bourque](#) Apr 4 '16 at 23:49

This is better than self-implementation. The fact that it's easy to implement is not an excuse for implementing it each time. – [pevogam](#) Jan 5 '17 at 18:53

- 3 I wouldn't consider it "standard." `Entry` is meant to be a key-value pair, and it's fine for that. But it has weaknesses as a true tuple. For example the `hashCode` implementation in `SimpleEntry` just xors the codes of the elements, so `<a,b>` hashes to the same value as `<b,a>`. Tuple implementations often sort lexicographically, but `SimpleEntry` doesn't implement `Comparable`. So be careful out there... – [Gene](#) Jan 16 '17 at 20:22

These built-in classes are an option, too:

- [AbstractMap.SimpleEntry](#)
- [AbstractMap.SimpleImmutableEntry](#)

edited Jan 9 '17 at 15:24



Didier L.

6,221 25 60

answered Feb 6 '09 at 17:28



Johannes Weiss

36.6k 12 78 116

- 3 They're not just an option, they're the right answer. I think some people just prefer to reinvent the wheel. – [CurtainDog](#) Jun 28 '12 at 1:42

Java 9+

In Java 9, you can simply write: `Map.entry(key, value)` to create an immutable pair.

Note: this method does not allow keys or values to be null. If you want to allow null values, for example, you'd want to change this to: `Map.entry(key, Optional.ofNullable(value))`.

Java 8+

In Java 8, you can use the more general-purpose `javafx.util.Pair` to create an immutable, serializable pair. This class *does* allow null keys and null values. (In Java 9, this class is included in the `javafx.base` module).

Java 6+

In Java 6 and up, you can use the more verbose `AbstractMap.SimpleImmutableEntry` for an immutable pair, or `AbstractMap.SimpleEntry` for a pair whose value can be changed. These classes also allow null keys and null values, and are serializable.

Android

If you're writing for Android, just use `Pair.create(key, value)` to create an immutable pair.

Apache Commons

[Apache Commons Lang](#) provides the helpful `Pair.of(key, value)` to create an immutable, comparable, serializable pair.

Eclipse Collections

If you're using pairs that contain primitives, [Eclipse Collections](#) provides some very efficient primitive pair classes that will avoid all the inefficient auto-boxing and auto-unboxing.

For instance, you could use `PrimitiveTuples.pair(int, int)` to create an `IntIntPair`, or `PrimitiveTuples.pair(float, long)` to create a `FloatLongPair`.

Otherwise

If none of the above solutions float your boat, you can simply copy and paste the following code (which, unlike the class listed in the accepted answer, guards against `NullPointerException`):

```
import java.util.Objects;

public class Pair<K, V> {

    public final K key;
    public final V value;

    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }


    public boolean equals(Object o) {
        return o instanceof Pair && Objects.equals(key, ((Pair<?,?>)o).key) &&
            Objects.equals(value, ((Pair<?,?>)o).value);
    }

    public int hashCode() {
        return 31 * Objects.hashCode(key) + Objects.hashCode(value);
    }

    public String toString() {
        return key + "=" + value;
    }
}
```

edited Apr 1 at 1:48

answered Sep 2 '14 at 2:42

 [Hans Brende](#)
2,625 2 17 32

1 JavaFX however is desktop-only, adding an unnecessary dependency for non-desktop environments (e.g. servers). – [foo](#) Jun 29 '17 at 17:19

Yes, thank you for the changes! I think it's pretty complete now. – [foo](#) Jun 29 '17 at 19:47

Apache common lang3 has Pair class and few other libraries mentioned in this thread [What is the equivalent of the C++ Pair<L,R> in Java?](#)

Example matching the requirement from your original question:

```
List<Pair<String, Integer>> myPairs = new ArrayList<Pair<String, Integer>>();
myPairs.add(Pair.of("val1", 11));
myPairs.add(Pair.of("val2", 17));

//...

for(Pair<String, Integer> pair : myPairs) {
    //following two lines are equivalent... whichever is easier for you...
    System.out.println(pair.getLeft() + ": " + pair.getRight());
    System.out.println(pair.getKey() + ": " + pair.getValue());
}
```

edited May 23 '17 at 11:55



Community ♦

1 1

answered Dec 11 '12 at 23:56



changed

1,036 5 25 50

1 This is the simplest option if Apache Commons is available. – [Kip](#) Nov 2 '16 at 14:55

To anyone developing for Android, you can use [android.util.Pair](#). :)

answered May 15 '14 at 6:09



[Xâppli'-l0llwlg'!](#) -

9,419 17 85 131

What about "Apache Commons Lang 3" `Pair` class and the relative subclasses ?

```
import org.apache.commons.lang3.tuple.ImmutablePair;
import org.apache.commons.lang3.tuple.Pair;
...
@SuppressWarnings("unchecked")
Pair<String, Integer>[] arr = new ImmutablePair[]{
    ImmutablePair.of("A", 1),
    ImmutablePair.of("B", 2)};

// both access the 'left' part
String key = arr[0].getKey();
String left = arr[0].getLeft();

// both access the 'right' part
Integer value = arr[0].getValue();
Integer right = arr[0].getRight();
```

`ImmutablePair` is a specific subclass that does not allow the values in the pair to be modified, but there are others implementations with different semantic. These are the Maven coordinates, if you need them.

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <version>3.4</version>
</dependency>
```

answered Nov 14 '15 at 19:34



[danidemi](#)

1,155 2 19 27

You could write a generic `Pair<A, B>` class and use this in an array or list. Yes, you have to write a class, but you can reuse the same class for all types, so you only have to do it once.

answered Feb 6 '09 at 17:16



[Dan Dyer](#)

44.3k 11 115 162

I'd love to see an example of that! – [DivideByHero](#) Feb 6 '09 at 17:18

1 Dan, the bad thing about that is that it is not possible to accept e.g. only `Pair<String,Integer>`s because of type erasure, no? But so is Java... – [Johannes Weiss](#) Feb 6 '09 at 17:26

Interesting, Johannes. java.sun.com/docs/books/tutorial/java/generics/erasure.html – [JMD](#) Feb 6 '09 at 17:42

I don't think this will work for plain arrays, but it definitely will work for other Collections. – [Outlaw Programmer](#) Feb 6 '09 at 17:49

3 the thing about type erasure: sorry, nonsense... – [Johannes Weiss](#) Feb 6 '09 at 18:02

|

The preferred solution as you've described it is a List of Pairs (i.e. List).

To accomplish this you would create a Pair class for use in your collection. This is a useful utility class to add to your code base.

The closest class in the Sun JDK providing functionality similar to a typical Pair class is `AbstractMap.SimpleEntry`. You could use this class rather than creating your own Pair class, though you would have to live with some awkward restrictions and I think most people would frown on this as not really the intended role of `SimpleEntry`. For example `SimpleEntry` has no `setKey()` method and no default constructor, so you may find it too limiting.

Bear in mind that Collections are designed to contain elements of a single type. Related utility interfaces such as `Map` are not actually Collections (i.e. `Map` does not implement the `Collection` interface). A `Pair` would not implement the `Collection` interface either but is obviously a useful class in building larger data structures.

answered Feb 6 '09 at 17:51

 [Jeremy Rishel](#)
239 1 2

I find this solution way better than the "winning" one with the generic `Pair<K, V>`. It does everything requested and comes out of the box. I use this one for my implementation. – [Sauer](#) Oct 15 '14 at 10:46

I was going to ask if you would not want to just use a `List<Pair<T, U>>` ? but then, of course, the JDK doesn't have a `Pair<>` class. But a quick Google found one on both [Wikipedia](#), and [forums.sun.com](#). Cheers

answered Feb 6 '09 at 17:19

 [JMD](#)
6,642 3 22 33

Expanding on the other answers a generic immutable Pair should have a static method to avoid cluttering your code with the call to the constructor:

```
class Pair<L,R> {
    final L left;
    final R right;

    public Pair(L left, R right) {
        this.left = left;
        this.right = right;
    }

    static <L,R> Pair<L,R> of(L left, R right){
        return new Pair<L,R>(left, right);
    }
}
```

if you name the static method "of" or "pairOf" the code becomes fluent as you can write either:

```
list.add(Pair.of(x,y)); // my preference
list.add(pairOf(x,y)); // use with import static x.y.Pair.pairOf
```

its a real shame that the core java libraries are so sparse on such things that you have to use commons-lang or other 3rd parties to do such basic stuff. yet another reason to upgrade to scala...

answered Feb 27 '13 at 23:15

 [simbo1905](#)
2,806 25 47

This is based on JavaHelp4u 's code.

Less verbose and shows how to do in one line and how to loop over things.

```
//====> Imports
import java.util.AbstractMap.SimpleEntry;
import java.util.ArrayList;
import java.util.List;
import java.util.Map.Entry;

//====> Single Entry
SimpleEntry<String, String> myEntry = new SimpleEntry<String, String>("ID",
"Text");
System.out.println("key: " + myEntry.getKey() + "    value:" +
myEntry.getValue());
System.out.println();

//====> List of Entries
List<Entry<String,String>> pairList = new ArrayList<>();

//-- Specify manually
Entry<String,String> firstButton = new SimpleEntry<String, String>("Red ", "Way
out");
pairList.add(firstButton);

//-- one liner:
pairList.add(new SimpleEntry<String,String>("Gray", "Alternate route"));
//Anonomous add.

//-- Iterate over Entry array:
for (Entry<String, String> entr : pairList) {
    System.out.println("Button: " + entr.getKey() + "    Label: " +
entr.getValue());
}
```

answered Jun 19 '14 at 20:45



Leo Ufimtsev

1,497 2 15 25

Apache Crunch also has a Pair class:

<http://crunch.apache.org/apidocs/0.5.0/org/apache/crunch/Pair.html>

answered Jul 22 '13 at 22:15



Bobak_KS

467 2 10

just create a class like

```
class tuples
{
    int x;
    int y;
}
```

then create List of this objects of tuples

```
List<tuples> list = new ArrayList<tuples>();
```

so you can also implement other new data structures in the same way.

answered Nov 1 '13 at 15:42



user93

1,122 1 12 33

I mean, even though there is no Pair class in Java there is something pretty simmilar: Map.Entry

[Map.Entry Documentation](#)

This is (simplifying quite a bit) what HashMap , or actually any Map stores.

You can create an instance of Map store your values in it and get the entry set. You will end up with a Set<Map.Entry<K,V>> which effectively is what you want.

So:

```
public static void main(String []args)
{
    HashMap<String, Integer> values = new HashMap<String,Integer>();
    values.put("A", 235); //your custom data, the types may be different
}
```

```
//more data insertions....  
Set<Map.Entry<String,Integer>> list = values.entrySet();//your list  
//do as you may with it  
}
```

answered Sep 5 '16 at 11:16



SomeDude

31 2

This option has the problem of the unique keys. Let's say you have attributes of persons (e.g. {(person1, "blue-eyed"), (person1, "red-haired"), (person2, "shortsighted"), (person2, "quick-thinker")}) you would not be able to store them as pairs in a map, since each person is the key to the map and would only allow one attribute per person. – [manuelvigarcia](#) Oct 27 '16 at 10:13

What about com.sun.tools.javac.util.Pair?

answered Jun 19 '12 at 6:22



Rene H.

19

6 This type is in tools.jar - part of the "Sun" implementation of the javac compiler. It is only distributed as part of the JDK, may not be present in other vendors implementations and is unlikely to be part of the public API. – [McDowell](#) Jun 19 '12 at 8:27
