Primitive array to List<>.

The question is we are given array of primitive type for example int[]. And this array needs to be converted to List<Integer>.

As we know that Java supports autoboxing.

This method is used to convert the int (primitive) to Wrapper type. We don’t use any method for this as compiler use Integer.valueOf(i);. Converting a primitive value (an int, for example) into an object of the corresponding wrapper class (Integer) is called autoboxing. The Java compiler applies autoboxing when a primitive value is: Passed as a parameter to a method that expects an object of the corresponding wrapper class. Assigned to a variable of the corresponding wrapper class.

Let us say that we have input as int[] arr. So we can have following code that converts int array to the List<Integer>.

List<Integer> result = **new** ArrayList<Integer>();

**for** (**int** i = 0; i < arr.length; i++) {

result.add(arr[i]);

}

**return** result;

Also, we need to check the corner conditions for this method. What if the passed array is null or its length is 0.

If the input array is null then following code is written to handle this condition.

**if** (arr == **null**) {

**return** **null**;

}

If input size of input array is 0 then we return new ArrayList<Integer>().

**if** (arr.length == 0) {

**returnnew** ArrayList<Integer>();

}

Bringing all of them together.

**public** **static** List<Integer> toList(**final** **int**[] arr) {

**if** (arr == **null**) {

**return** **null**;

}

**if** (arr.length == 0) {

**return** ***INTEGER\_LIST***;

}

List<Integer> result = **new** ArrayList<Integer>();

**for** (**int** i = 0; i < arr.length; i++) {

result.add(arr[i]);

}

**return** result;

}

Now we do same for short[], byte[], long[], double[], float[] and char[].

Below is the code for converting all for the above mentioned primitive array to List<>.

**package** arrays;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** PrimitiveArrayToList {

/\*\* The Constant INTEGER\_LIST. \*/

**public** **static** **final** List<Integer> ***INTEGER\_LIST*** = **new** ArrayList<Integer>();

/\*\* The Constant FLOAT\_LIST. \*/

**public** **static** **final** List<Float> ***FLOAT\_LIST*** = **new** ArrayList<Float>();

/\*\* The Constant LONG\_LIST. \*/

**public** **static** **final** List<Long> ***LONG\_LIST*** = **new** ArrayList<Long>();

/\*\* The Constant DOUBLE\_LIST. \*/

**public** **static** **final** List<Double> ***DOUBLE\_LIST*** = **new** ArrayList<Double>();

/\*\* The Constant CHARACTER\_LIST. \*/

**public** **static** **final** List<Character> ***CHARACTER\_LIST*** = **new** ArrayList<Character>();

/\*\* The Constant SHORT\_LIST. \*/

**public** **static** **final** List<Short> ***SHORT\_LIST*** = **new** ArrayList<Short>();

/\*\* The Constant BYTE\_LIST. \*/

**public** **static** **final** List<Byte> ***BYTE\_LIST*** = **new** ArrayList<Byte>();

**public** **static** List<Integer> toList(**final** **int**[] arr) {

**if** (arr == **null**) {

**return** **null**;

}

**if** (arr.length == 0) {

**return** ***INTEGER\_LIST***;

}

List<Integer> result = **new** ArrayList<Integer>();

**for** (**int** i = 0; i < arr.length; i++) {

result.add(arr[i]);

}

**return** result;

}

/\*\*

\* To list.

\*

\* **@param** arr

\* the arr

\* **@return** the list

\*/

**public** **static** List<Float> toList(**final** **float**[] arr) {

**if** (arr == **null**) {

**return** **null**;

}

**if** (arr.length == 0) {

**return** ***FLOAT\_LIST***;

}

List<Float> result = **new** ArrayList<Float>();

**for** (**int** i = 0; i < arr.length; i++) {

result.add(arr[i]);

}

**return** result;

}

/\*\*

\* To list.

\*

\* **@param** arr

\* the arr

\* **@return** the list

\*/

**public** **static** List<Double> toList(**final** **double**[] arr) {

**if** (arr == **null**) {

**return** **null**;

}

**if** (arr.length == 0) {

**return** ***DOUBLE\_LIST***;

}

List<Double> result = **new** ArrayList<Double>();

**for** (**int** i = 0; i < arr.length; i++) {

result.add(arr[i]);

}

**return** result;

}

/\*\*

\* To list.

\*

\* **@param** arr

\* the arr

\* **@return** the list

\*/

**public** **static** List<Long> toList(**final** **long**[] arr) {

**if** (arr == **null**) {

**return** **null**;

}

**if** (arr.length == 0) {

**return** ***LONG\_LIST***;

}

List<Long> result = **new** ArrayList<Long>();

**for** (**int** i = 0; i < arr.length; i++) {

result.add(arr[i]);

}

**return** result;

}

**public** **static** List<Character> toList(**final** **char**[] arr) {

**if** (arr == **null**) {

**return** **null**;

}

**if** (arr.length == 0) {

**return** ***CHARACTER\_LIST***;

}

List<Character> result = **new** ArrayList<Character>();

**for** (**int** i = 0; i < arr.length; i++) {

result.add(arr[i]);

}

**return** result;

}

**public** **static** List<Short> toList(**final** **short**[] arr) {

**if** (arr == **null**) {

**return** **null**;

}

**if** (arr.length == 0) {

**return** ***SHORT\_LIST***;

}

List<Short> result = **new** ArrayList<Short>();

**for** (**int** i = 0; i < arr.length; i++) {

result.add(arr[i]);

}

**return** result;

}

**public** **static** List<Byte> toList(**final** **byte**[] arr) {

**if** (arr == **null**) {

**return** **null**;

}

**if** (arr.length == 0) {

**return** ***BYTE\_LIST***;

}

List<Byte> result = **new** ArrayList<Byte>();

**for** (**int** i = 0; i < arr.length; i++) {

result.add(arr[i]);

}

**return** result;

}

**public** **static** **void** main(String[] args) {

**byte**[] b = { 1, 2, 3, 4, 5 };

**char**[] c = { 'a', 'b', 'c', 'd', 'e' };

**int**[] i = { 1, 2, 3, 4, 5 };

**long**[] l = { 1, 2, 3, 4, 5 };

**double**[] d = { 1, 2, 3, 4, 5 };

**float**[] f = { 1, 2, 3, 4, 5 };

**short**[] s = { 1, 2, 3, 4, 5 };

*display*("byte[] to List<Byte> ", *toList*(b));

*display*("char[] to List<Character> ", *toList*(c));

*display*("int[] to List<Integer> ", *toList*(i));

*display*("long[] to List<Long> ", *toList*(l));

*display*("double[] to List<Double> ", *toList*(d));

*display*("float[] to List<Float> ", *toList*(f));

*display*("short[] to List<Short> ", *toList*(s));

}

**static** **void** display(String type, List<?> list) {

System.***out***.println(type + " " + list);

}

}

Output:

byte[] to List<Byte> [1, 2, 3, 4, 5]

char[] to List<Character> [a, b, c, d, e]

int[] to List<Integer> [1, 2, 3, 4, 5]

long[] to List<Long> [1, 2, 3, 4, 5]

double[] to List<Double> [1.0, 2.0, 3.0, 4.0, 5.0]

float[] to List<Float> [1.0, 2.0, 3.0, 4.0, 5.0]

short[] to List<Short> [1, 2, 3, 4, 5]

Following are the test cases.

**package** arrays;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**import** org.junit.Assert;

**import** org.junit.Test;

**public** **class** PrimitiveArrayToListTest {

**byte**[] b = { 1, 2, 3, 4, 5 };

**byte**[] bNull=**null**;

**char**[] c = { 'a', 'b', 'c', 'd', 'e' };

**char**[] cNull=**null**;

**int**[] i = { 1, 2, 3, 4, 5 };

**int**[] iNull=**null**;

**long**[] l = { 1, 2, 3, 4, 5 };

**long**[] lNull=**null**;

**double**[] d = { 1, 2, 3, 4, 5 };

**double**[] dNull=**null**;

**float**[] f = { 1, 2, 3, 4, 5 };

**float**[] fNull=**null**;

**short**[] s = { 1, 2, 3, 4, 5 };

**short**[] sNull=**null**;

/\*\* The Constant INTEGER\_LIST. \*/

**public** **static** **final** List<Integer> ***INTEGER\_LIST\_FILLED*** = **new** ArrayList<Integer>(Arrays.*asList*(1,2,3,4,5));

/\*\* The Constant FLOAT\_LIST. \*/

**public** **static** **final** List<Float> ***FLOAT\_LIST\_FILLED*** = **new** ArrayList<Float>(Arrays.*asList*(1f,2f,3f,4f,5f));

/\*\* The Constant LONG\_LIST. \*/

**public** **static** **final** List<Long> ***LONG\_LIST\_FILLED*** = **new** ArrayList<Long>(Arrays.*asList*(1l,2l,3l,4l,5l));

/\*\* The Constant DOUBLE\_LIST. \*/

**public** **static** **final** List<Double> ***DOUBLE\_LIST\_FILLED*** = **new** ArrayList<Double>(Arrays.*asList*(1d,2d,3d,4d,5d));

/\*\* The Constant CHARACTER\_LIST. \*/

**public** **static** **final** List<Character> ***CHARACTER\_LIST\_FILLED*** = **new** ArrayList<Character>(Arrays.*asList*( 'a', 'b', 'c', 'd', 'e' ));

/\*\* The Constant SHORT\_LIST. \*/

**public** **static** **final** List<Short> ***SHORT\_LIST\_FILLED*** = **new** ArrayList<Short>(Arrays.*asList*((**short**)1,(**short**)2,(**short**)3,(**short**)4,(**short**)5));

/\*\* The Constant BYTE\_LIST. \*/

**public** **static** **final** List<Byte> ***BYTE\_LIST\_FILLED*** = **new** ArrayList<Byte>(Arrays.*asList*((**byte**)1,(**byte**)2,(**byte**)3,(**byte**)4,(**byte**)5));

@Test

**public** **void** testByteNull() {

Assert.*assertNull*(PrimitiveArrayToList.*toList*(bNull));

}

@Test

**public** **void** testByteFilled() {

Assert.*assertEquals*(***BYTE\_LIST\_FILLED***, PrimitiveArrayToList.*toList*(b));

}

@Test

**public** **void** testIntNull() {

Assert.*assertNull*(PrimitiveArrayToList.*toList*(iNull));

}

@Test

**public** **void** testIntFilled() {

Assert.*assertEquals*(***INTEGER\_LIST\_FILLED***, PrimitiveArrayToList.*toList*(i));

}

@Test

**public** **void** testFloatNull() {

Assert.*assertNull*(PrimitiveArrayToList.*toList*(fNull));

}

@Test

**public** **void** testFloatFilled() {

Assert.*assertEquals*(***FLOAT\_LIST\_FILLED***, PrimitiveArrayToList.*toList*(f));

}

@Test

**public** **void** testLongNull() {

Assert.*assertNull*(PrimitiveArrayToList.*toList*(lNull));

}

@Test

**public** **void** testLongFilled() {

Assert.*assertEquals*(***LONG\_LIST\_FILLED***, PrimitiveArrayToList.*toList*(l));

}

@Test

**public** **void** testDoubleNull() {

Assert.*assertNull*(PrimitiveArrayToList.*toList*(dNull));

}

@Test

**public** **void** testDoubleFilled() {

Assert.*assertEquals*(***DOUBLE\_LIST\_FILLED***, PrimitiveArrayToList.*toList*(d));

}

@Test

**public** **void** testCharNull() {

Assert.*assertNull*(PrimitiveArrayToList.*toList*(cNull));

}

@Test

**public** **void** testCharFilled() {

Assert.*assertEquals*(***CHARACTER\_LIST\_FILLED***, PrimitiveArrayToList.*toList*(c));

}

@Test

**public** **void** testShortNull() {

Assert.*assertNull*(PrimitiveArrayToList.*toList*(sNull));

}

@Test

**public** **void** testShortFilled() {

Assert.*assertEquals*(***SHORT\_LIST\_FILLED***, PrimitiveArrayToList.*toList*(s));

}

}