**Java Wrapper Class**

In this tutorial, we will learn about the Java Wrapper class with the help of examples.

The wrapper classes in Java are used to convert primitive types (int, char, float, etc) into corresponding objects.

Each of the 8 primitive types has corresponding wrapper classes.

|  |  |
| --- | --- |
| Primitive Type | Wrapper Class |
| byte | Byte |
| boolean | Boolean |
| char | Character |
| double | Double |
| float | Float |
| int | Integer |
| long | Long |
| short | Short |

**Convert Primitive Type to Wrapper Objects**

We can also use the valueOf() method to convert primitive types into corresponding objects.

**Example 1: Primitive Types to Wrapper Objects**

class Main {

public static void main(String[] args) {

// create primitive types

int a = 5;

double b = 5.65;

//converts into wrapper objects

Integer aObj = Integer.valueOf(a);

Double bObj = Double.valueOf(b);

if(aObj instanceof Integer) {

System.out.println("An object of Integer is created.");

}

if(bObj instanceof Double) {

System.out.println("An object of Double is created.");

}

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

An object of Integer is created.

An object of Double is created.

In the above example, we have used the valueOf() method to convert the primitive types into objects.

Here, we have used the instanceof operator to check whether the generated objects are of Integer or Double type or not.

However, the Java compiler can directly convert the primitive types into corresponding objects. For example,

int a = 5;

// converts into object

Integer aObj = a;

double b = 5.6;

// converts into object

Double bObj = b;

This process is known as **auto-boxing**. To learn more, visit [Java autoboxing and unboxing](https://www.programiz.com/java-programming/autoboxing-unboxing).

**Note**: We can also convert primitive types into wrapper objects using Wrapper class constructors. But the use of constructors is discarded after Java 9.

**Wrapper Objects into Primitive Types**

To convert objects into the primitive types, we can use the corresponding value methods (intValue(), doubleValue(), etc) present in each wrapper class.

**Example 2: Wrapper Objects into Primitive Types**

class Main {

public static void main(String[] args) {

// creates objects of wrapper class

Integer aObj = Integer.valueOf(23);

Double bObj = Double.valueOf(5.55);

// converts into primitive types

int a = aObj.intValue();

double b = bObj.doubleValue();

System.out.println("The value of a: " + a);

System.out.println("The value of b: " + b);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

The value of a: 23

The value of b: 5.55

In the above example, we have used the intValue() and doubleValue() method to convert the Integer and Double objects into corresponding primitive types.

However, the Java compiler can automatically convert objects into corresponding primitive types. For example,

Integer aObj = Integer.valueOf(2);

// converts into int type

int a = aObj;

Double bObj = Double.valueOf(5.55);

// converts into double type

double b = bObj;

This process is known as **unboxing**. To learn more, visit [Java autoboxing and unboxing](https://www.programiz.com/java-programming/autoboxing-unboxing).

**Advantages of Wrapper Classes**

* In Java, sometimes we might need to use objects instead of primitive data types. For example, while working with collections.
* // error
* ArrayList<int> list = new ArrayList<>();
* // runs perfectly

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

In such cases, wrapper classes help us to use primitive data types as objects.

* We can store the null value in wrapper objects. For example,
* // generates an error
* int a = null;
* // runs perfectly
* Integer a = null;

**Note**: Primitive types are more efficient than corresponding objects. Hence, when efficiency is the requirement, it is always recommended primitive types.

**Java Arrays**

In this tutorial, we will learn to work with arrays in Java. We will learn to declare, initialize, and access array elements with the help of examples.

An array is a collection of similar types of data.

For example, if we want to store the names of 100 people then we can create an array of the string type that can store 100 names.

String[] array = new String[100];

Here, the above array cannot store more than 100 names. The number of values in a Java array is always fixed.

**How to declare an array in Java?**

In Java, here is how we can declare an array.

dataType[] arrayName;

* dataType - it can be [primitive data types](https://www.programiz.com/java-programming/variables-primitive-data-types#data-types) like int, char, double, byte, etc. or [Java objects](https://www.programiz.com/java-programming/class-objects)
* arrayName - it is an [identifier](https://www.programiz.com/java-programming/keywords-identifiers#identifiers)

For example,

double[] data;

Here, data is an array that can hold values of type double.

**But, how many elements can array this hold?**

Good question! To define the number of elements that an array can hold, we have to allocate memory for the array in Java. For example,

// declare an array

double[] data;

// allocate memory

data = new double[10];

Here, the array can store **10** elements. We can also say that the **size or length** of the array is 10.

In Java, we can declare and allocate the memory of an array in one single statement. For example,

double[] data = new double[10];

**How to Initialize Arrays in Java?**

In Java, we can initialize arrays during declaration. For example,

//declare and initialize and array

int[] age = {12, 4, 5, 2, 5};

Here, we have created an array named age and initialized it with the values inside the curly brackets.

Note that we have not provided the size of the array. In this case, the Java compiler automatically specifies the size by counting the number of elements in the array (i.e. 5).

In the Java array, each memory location is associated with a number. The number is known as an array index. We can also initialize arrays in Java, using the index number. For example,

// declare an array

int[] age = new int[5];

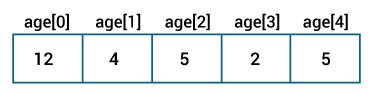
// initialize array

age[0] = 12;

age[1] = 4;

age[2] = 5;

..

Java Arrays initialization

**Note**:

* Array indices always start from 0. That is, the first element of an array is at index 0.
* If the size of an array is n, then the last element of the array will be at index n-1.

**How to Access Elements of an Array in Java?**

We can access the element of an array using the index number. Here is the syntax for accessing elements of an array,

// access array elements

array[index]

Let's see an example of accessing array elements using index numbers.

**Example: Access Array Elements**

class Main {

public static void main(String[] args) {

// create an array

int[] age = {12, 4, 5, 2, 5};

// access each array elements

System.out.println("Accessing Elements of Array:");

System.out.println("First Element: " + age[0]);

System.out.println("Second Element: " + age[1]);

System.out.println("Third Element: " + age[2]);

System.out.println("Fourth Element: " + age[3]);

System.out.println("Fifth Element: " + age[4]);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

Accessing Elements of Array:

First Element: 12

Second Element: 4

Third Element: 5

Fourth Element: 2

Fifth Element: 5

In the above example, notice that we are using the index number to access each element of the array.

We can use loops to access all the elements of the array at once.

**Looping Through Array Elements**

In Java, we can also loop through each element of the array. For example,

**Example: Using For Loop**

class Main {

public static void main(String[] args) {

// create an array

int[] age = {12, 4, 5};

// loop through the array

// using for loop

System.out.println("Using for Loop:");

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

System.out.println(age[i]);

}

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

Using for Loop:

12

4

5

In the above example, we are using the [for Loop in Java](https://www.programiz.com/java-programming/for-loop) to iterate through each element of the array. Notice the expression inside the loop,

age.length

Here, we are using the length property of the array to get the size of the array.

We can also use the [for-each loop](https://www.programiz.com/java-programming/enhanced-for-loop) to iterate through the elements of an array. For example,

**Example: Using the for-each Loop**

class Main {

public static void main(String[] args) {

// create an array

int[] age = {12, 4, 5};

// loop through the array

// using for loop

System.out.println("Using for-each Loop:");

for(int a : age) {

System.out.println(a);

}

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

Using for-each Loop:

12

4

5

**Example: Compute Sum and Average of Array Elements**

class Main {

public static void main(String[] args) {

int[] numbers = {2, -9, 0, 5, 12, -25, 22, 9, 8, 12};

int sum = 0;

Double average;

// access all elements using for each loop

// add each element in sum

for (int number: numbers) {

sum += number;

}

// get the total number of elements

int arrayLength = numbers.length;

// calculate the average

// convert the average from int to double

average = ((double)sum / (double)arrayLength);

System.out.println("Sum = " + sum);

System.out.println("Average = " + average);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

Sum = 36

Average = 3.6

In the above example, we have created an array of named numbers. We have used the for...each loop to access each element of the array.

Inside the loop, we are calculating the sum of each element. Notice the line,

int arrayLength = number.length;

Here, we are using the [length attribute](http://stackoverflow.com/questions/8755812/array-length-in-java) of the array to calculate the size of the array. We then calculate the average using:

average = ((double)sum / (double)arrayLength);

As you can see, we are converting the int value into double. This is called type casting in Java. To learn more about typecasting, visit [Java Type Casting](https://www.programiz.com/java-programming/typecasting).

**Multidimensional Arrays**

Arrays we have mentioned till now are called one-dimensional arrays. However, we can declare multidimensional arrays in Java.

A multidimensional array is an array of arrays. That is, each element of a multidimensional array is an array itself. For example,

double[][] matrix = {{1.2, 4.3, 4.0},

{4.1, -1.1}

};

Here, we have created a multidimensional array named matrix. It is a 2-dimensional array. To learn more, visit the [Java multidimensional array](https://www.programiz.com/java-programming/multidimensional-array).

# Java Multidimensional Arrays

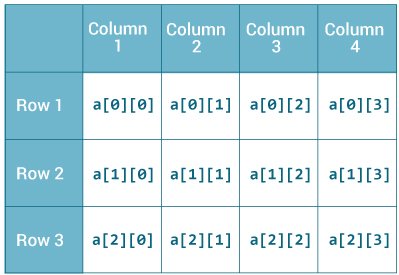
In this tutorial, we will learn about the Java multidimensional array using 2-dimensional arrays and 3-dimensional arrays with the help of examples.

Before we learn about the multidimensional array, make sure you know about [Java array](https://www.programiz.com/java-programming/arrays).

A multidimensional array is an array of arrays. Each element of a multidimensional array is an array itself. For example,

int[][] a = new int[3][4];

Here, we have created a multidimensional array named a. It is a 2-dimensional array, that can hold a maximum of 12 elements,

2-dimensional Array

Remember, Java uses zero-based indexing, that is, indexing of arrays in Java starts with 0 and not 1.

Let's take another example of the multidimensional array. This time we will be creating a 3-dimensional array. For example,

String[][][] data = new String[3][4][2];

Here, data is a 3d array that can hold a maximum of 24 (3\*4\*2) elements of type String.

## How to initialize a 2d array in Java?

Here is how we can initialize a 2-dimensional array in Java.

int[][] a = {

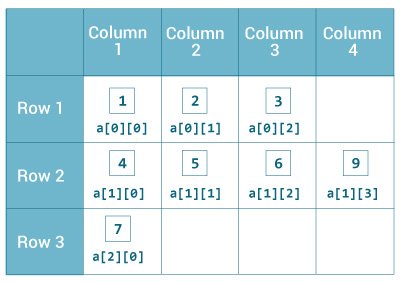
{1, 2, 3},

{4, 5, 6, 9},

{7},

};

As we can see, each element of the multidimensional array is an array itself. And also, unlike C/C++, each row of the multidimensional array in Java can be of different lengths.

Initialization of 2-dimensional Array

### Example: 2-dimensional Array

class MultidimensionalArray {

public static void main(String[] args) {

// create a 2d array

int[][] a = {

{1, 2, 3},

{4, 5, 6, 9},

{7},

};

// calculate the length of each row

System.out.println("Length of row 1: " + a[0].length);

System.out.println("Length of row 2: " + a[1].length);

System.out.println("Length of row 3: " + a[2].length);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

Length of row 1: 3

Length of row 2: 4

Length of row 3: 1

In the above example, we are creating a multidimensional array named a. Since each component of a multidimensional array is also an array (a[0], a[1] and a[2] are also arrays).

Here, we are using the length attribute to calculate the length of each row.

### Example: Print all elements of 2d array Using Loop

class MultidimensionalArray {

public static void main(String[] args) {

int[][] a = {

{1, -2, 3},

{-4, -5, 6, 9},

{7},

};

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

for(int j = 0; j < a[i].length; ++j) {

System.out.println(a[i][j]);

}

}

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

1

-2

3

-4

-5

6

9

7

We can also use the [for...each loop](https://www.programiz.com/java-programming/enhanced-for-loop) to access elements of the multidimensional array. For example,

class MultidimensionalArray {

public static void main(String[] args) {

// create a 2d array

int[][] a = {

{1, -2, 3},

{-4, -5, 6, 9},

{7},

};

// first for...each loop access the individual array

// inside the 2d array

for (int[] innerArray: a) {

// second for...each loop access each element inside the row

for(int data: innerArray) {

System.out.println(data);

}

}

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

1

-2

3

-4

-5

6

9

7

In the above example, we are have created a 2d array named a. We then used for loop and for...each loop to access each element of the array.

## How to initialize a 3d array in Java?

Let's see how we can use a 3d array in Java. We can initialize a 3d array similar to the 2d array. For example,

// test is a 3d array

int[][][] test = {

{

{1, -2, 3},

{2, 3, 4}

},

{

{-4, -5, 6, 9},

{1},

{2, 3}

}

};

Basically, a 3d array is an array of 2d arrays. The rows of a 3d array can also vary in length just like in a 2d array.

### Example: 3-dimensional Array

class ThreeArray {

public static void main(String[] args) {

// create a 3d array

int[][][] test = {

{

{1, -2, 3},

{2, 3, 4}

},

{

{-4, -5, 6, 9},

{1},

{2, 3}

}

};

// for..each loop to iterate through elements of 3d array

for (int[][] array2D: test) {

for (int[] array1D: array2D) {

for(int item: array1D) {

System.out.println(item);

}

}

}

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

1

-2

3

2

3

4

-4

-5

6

9

1

2

3

**Java Comments**

In this tutorial, you will learn about Java comments, why we use them, and how to use comments in right way.

In computer programming, comments are a portion of the program that are completely ignored by Java compilers. They are mainly used to help programmers to understand the code. For example,

// declare and initialize two variables

int a =1;

int b = 3;

// print the output

System.out.println("This is output");

Here, we have used the following comments,

* declare and initialize two variables
* print the output

**Types of Comments in Java**

In Java, there are two types of comments:

* single-line comment
* multi-line comment

**Single-line Comment**

A single-line comment starts and ends in the same line. To write a single-line comment, we can use the // symbol. For example,

// "Hello, World!" program example

class Main {

public static void main(String[] args) {

// prints "Hello, World!"

System.out.println("Hello, World!");

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

Hello, World!

Here, we have used two single-line comments:

* "Hello, World!" program example
* prints "Hello World!"

The Java compiler ignores everything from // to the end of line. Hence, it is also known as **End of Line** comment.

**Multi-line Comment**

When we want to write comments in multiple lines, we can use the multi-line comment. To write multi-line comments, we can use the /\*....\*/ symbol. For example,

/\* This is an example of multi-line comment.

\* The program prints "Hello, World!" to the standard output.

\*/

class HelloWorld {

public static void main(String[] args) {

System.out.println("Hello, World!");

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

Hello, World!

Here, we have used the multi-line comment:

/\* This is an example of multi-line comment.

\* The program prints "Hello, World!" to the standard output.

\*/

This type of comment is also known as **Traditional Comment**. In this type of comment, the Java compiler ignores everything from /\* to \*/.

**Use Comments the Right Way**

One thing you should always consider that comments shouldn't be the substitute for a way to explain poorly written code in English. You should always write well structured and self explaining code. And, then use comments.

Some believe that code should be self-describing and comments should be rarely used. However, in my personal opinion, there is nothing wrong with using comments. We can use comments to explain complex algorithms, regex or scenarios where we have to choose one technique among different technique to solve problems.

**Note**: In most cases, always use comments to explain '**why**' rather than '**how**' and you are good to go.

**1. Overview**

In this tutorial, You'll learn **ArrayList with Real-Time examples**. If you are new to java programming, you'll get a question "**What are the real-life examples of the ArrayList in Java?**". Initial days when I was in engineering the second year, my professor was teaching [ArrayList](https://java-w3schools.blogspot.com/2017/09/javautilarraylist-class-in-java-with.html" \t "_blank) in java.  
  
I have learned about it. **ArrayList** is a dynamic array to store the elements and also it grows the size automatically if it reaching its threshold value. But when we should be using the ArrayList in realtime applications.

Remember, for now, **In the software world there is no application deployed in production without ArrayList. Now think about the usage of ArrayList. ArrayList can be used in many more scenarios in realtime.**  
  
We will be seeing a few real-time examples of ArrayList in Java.

**2. Collecting database records into ArrayList**

JDBC is used to connect to the database and perform the operations on the tables. **Every application needs to save user data and activities into the database**. Once a Select query is executed, a ResultSet instance is returned. This ResultSet will contain all records.  
  
For Example, the Amazon website has many customer's records in its database. If they want to retrieve the customers from the database and show it on a web screen. In this case, ArrayList will be used to add customer records from ResultSet.  
  
Let us take a look at the below code.

**Customer class:**

package com.java.w3schools.blog.arraylist;

import java.io.Serializable;

import java.util.Date;

public class Customer implements Serializable {

private String fullName;

private String email;

private String password;

private String mobileNumber;

private Date dateOfBirth;

public String getFullName() {

return fullName;

}

public void setFullName(String fullName) {

this.fullName = fullName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getMobileNumber() {

return mobileNumber;

}

public void setMobileNumber(String mobileNumber) {

this.mobileNumber = mobileNumber;

}

public Date getDateOfBirth() {

return dateOfBirth;

}

public void setDateOfBirth(Date dateOfBirth) {

this.dateOfBirth = dateOfBirth;

}

}

**Loading into ArrayList:**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class SelectDataDemo {

    public static void main(String[] args) {

        Connection connection = null;

        Statement selectStmt = null;

List customerList = new ArrayList;

        try

        {

            Class.forName("com.mysql.jdbc.Driver");

            connection = DriverManager.getConnection("jdbc:mysql://localhost:8080/AMAZONDB", "scoot", "tiger");

            selectStmt = connection.createStatement();

            ResultSet rs = selectStmt.executeQuery("SELECT FULL\_NAME, EMAIL, PASSWORD, DOB, MOBILE\_NUMBER FROM CUSTOMER ");

            while(rs.next())

            {

  Customer customer = new Customer();

               customer.setFullName(rs.getString(1));

               customer.setEmail(rs.getString(2));

               customer.setPassword(rs.getString(3));

               customer.setDateOfBirth(new java.util.Date(rs.getDate(4).getTime()));

  customer.setMobileNumber(rs.getString(5));

  customerList.add(customer)

            }

        }

        catch (Exception e) {

            e.printStackTrace();

        }finally {

            try {

                selectStmt.close();

                insertStmt.close();

                connection.close();

            } catch (Exception e) {

                e.printStackTrace();

            }

        }

System.out.println("Customer records count : "+customerList.size());

    }

}

This is the best case to use the ***ArrayList*** to load the records. Every application must be having this kind of use case.

**3. Returning List of transactions for Credit Card**

Let us take a scenario where application A is using a third party API to get the credit card transactions. That API will returns all transactions for the recent month for a given credit card number.  
  
Let us take a look at the response returned by API.

**CreditCardResponse.java:**

This class has instance variable transactions which are a type of *List<Transaction>*. This holds a list of **all transactions and cardholder name, expiry date** as well.

package com.java.w3schools.blog.arraylist;

import java.util.Date;

import java.util.List;

public class CreditCardResponse {

private long cardNumber;

private String nameOnCard;

private Date expDate;

private List transactions;

public long getCardNumber() {

return cardNumber;

}

public void setCardNumber(long cardNumber) {

this.cardNumber = cardNumber;

}

public String getNameOnCard() {

return nameOnCard;

}

public void setNameOnCard(String nameOnCard) {

this.nameOnCard = nameOnCard;

}

public Date getExpDate() {

return expDate;

}

public void setExpDate(Date expDate) {

this.expDate = expDate;

}

public List getTransactions() {

return transactions;

}

public void setTransactions(List transactions) {

this.transactions = transactions;

}

}

**Transaction.java**

package com.java.w3schools.blog.arraylist;

import java.util.Date;

public class Transaction {

private long id;

private Date processedDate;

private double amount;

private String description;

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

public Date getProcessedDate() {

return processedDate;

}

public void setProcessedDate(Date processedDate) {

this.processedDate = processedDate;

}

public double getAmount() {

return amount;

}

public void setAmount(double amount) {

this.amount = amount;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

}

**4. Use anywhere no removal and insertions in the middle**

***ArrayList*** is implemented on behavior to [add](https://java-w3schools.blogspot.com/2019/04/java-arraylist-add-method-examples.html) the elements or values at the end of the list.  
For example, if you are working with huge volume data and after adding the values to ***ArrayList*** if you do not want to remove the values or add new values in the between the exiting values then you are good to use the ***ArrayList*** in such scenario's.

**5. Analyze the use case before using ArrayList**

Before Blindly using the ArrayList, understand the scenario and data set. If no insertions and removals are then good to go with ArrayList.  
  
PROCESS SEQUENTIALLY and ACTIONS VALIDATE, CONVERSOIONS, PROCESSING, POST VALIDATING?  
  
Let us talk about another scenario. You have to process a file and it has 5 stages.  
  
**A) File reading**

**B) Validation**  
**C) Conversion**  
**D) PROCESSING**  
**E) PostValidation**  
  
All these are the processes need to be run sequentially. First, add all of these processes to the ArrayList.

List steps = new ArrayList();

steps.add(FileReadingProcess.getInstance());

steps.add(ValidationProcess.getInstance());

steps.add(ConversionProcess.getInstance());

steps.add(ProcessingProcess.getInstance());

steps.add(PostValidationProcess.getInstance());

Now, pass the steps list to another service to follow the execution of the steps in order.

// invoking the service.

parsingService.sendStepsForFeed(steps, fileLocaion);

// apply all these phases to the file.

**6. ArrayList Example Programs**

Read more articles on ArrayList.

* [**ArrayList API and usage Examples**](https://java-w3schools.blogspot.com/2017/09/javautilarraylist-class-in-java-with.html)
* [**How to add values to ArrayList?**](https://java-w3schools.blogspot.com/2017/11/arraylist-add.html)
* [**How to remove elements from ArrayList?**](https://java-w3schools.blogspot.com/2019/04/how-to-remove-element-from-arraylist.html)
* [**Removing values by index from ArrayList?**](https://java-w3schools.blogspot.com/2017/12/java-arraylist-remove.html)

**7. Conclusion**

In this article, We've seen **when and where to use ArrayList in real time applications. Explained real life scenarios such as adding the retrieved records from the database, adding credit card transactions**.  
  
And also discussed where not to use ArrayList and analyzing the input data to take the decision.