

Java.lang.ClassCastException: java.util.Arrays\$ArrayList cannot be cast to java.util.ArrayList

In this post, we will see how to fix `java.lang.ClassCastException: java.util.Arrays$ArrayList cannot be cast to java.util.ArrayList`.

`ClassCastException` is runtime exception which indicate that code has tried to cast an object to a subclass which is not an instance.

Reason for java.lang.ClassCastException: java.util.Arrays\$ArrayList cannot be cast to java.util.ArrayList

Although static method `Arrays.asList()` returns `List`, it returns `java.util.Arrays$ArrayList` which is different from `ArrayList`, so when we try to cast `Arrays.asList()` to `ArrayList`, it throws exception.

Let's understand with the help of example:

```
package org.sharma;

import java.util.ArrayList;
import java.util.Arrays;

public class ArrayToList {

    public static void main(String[] args) {
        String[] countries = {"India", "China", "Bhutan"};

        ArrayList<String> listOfCountries = (ArrayList) Arrays.asList(countries);
        System.out.println(listOfCountries);
    }
}
```

OUTPUT

Exception in thread "main" java.lang.ClassCastException: class java.util.Arrays\$ArrayList cannot be cast to class java.util.ArrayList (java.util.Arrays\$ArrayList and java.util.ArrayList are in module java.base of loader 'bootstrap')
at org.arpit.java2blog.ArrayToList.main(ArrayToList.java:11)

Here is source code of `Arrays.asList()` method.

```
@SafeVarargs
@SuppressWarnings("varargs")
public static <T> List<T> asList(T... a) {
    return new ArrayList<>(a);
}
```

```
/**
 * @serial include
 */
private static class ArrayList<E> extends AbstractList<E>
    implements RandomAccess, java.io.Serializable
{
    ...
}
```

As you can see that there is static inner class present in `java.util.Arrays` which is different from `java.util.ArrayList`.

Fixes for `java.lang.ClassCastException: java.util.Arrays$ArrayList cannot be cast to java.util.ArrayList`

Use `ArrayList`'s constructor

If you must have instance of `java.util.ArrayList`, then you can create `ArrayList` instance using [ArrayList's constructor](#).

```
ArrayList<String> listOfCountries = new ArrayList(Arrays.asList(countries));
```

Here is complete program

```
package org.sharma;

import java.util.ArrayList;
import java.util.Arrays;
public class ArrayToList {
    public static void main(String[] args) {
        String[] countries = {"India", "China", "Bhutan"};
        ArrayList<String> listOfCountries = new ArrayList(Arrays.asList(countries));
        System.out.println(listOfCountries);
    }
}
```

Output: [India, China, Bhutan]

Here is source code of ArrayList's constructor

```
/* Constructs a list containing the elements of the specified
 * collection, in the order they are returned by the collection's
 * iterator.
 *
 * @param c the collection whose elements are to be placed into this list
 * @throws NullPointerException if the specified collection is null
 */
public ArrayList(Collection<? extends E> c) {
    elementData = c.toArray();
    size = elementData.length;
    // c.toArray might (incorrectly) not return Object[] (see 6260652)
    if (elementData.getClass() != Object[].class)
        elementData = Arrays.copyOf(elementData, size, Object[].class);
}
```

Assign Arrays.asList() to List reference rather than ArrayList

We can declare List rather than [ArrayList](#) to avoid the exception. Since java.util.Arrays\$ArrayList implements List interface, we can assign it to List reference.

```
List<String> listOfCountries = Arrays.asList(countries);
```

Here is complete program

```
package org.arpit.java2blog;

import java.util.Arrays;
import java.util.List;

public class ArrayToList {

    public static void main(String[] args) {
        String[] countries = {"India", "China", "Bhutan"};
        List<String> listOfCountries = Arrays.asList(countries);
        System.out.println(listOfCountries);
    }
}
```

Output: [India, China, Bhutan]

java.util.HashMap\$Values cannot be cast to class java.util.List

In this post, we will see how to fix error `java.util.HashMap$Values cannot be cast to class java.util.List`.

Why HashMap values cannot be cast to list?

HashMap values returns **java.util.Collection** and you can not cast Collection to List or ArrayList. It will throw **ClassCastException** in this scenario.

Let's understand with the help of example:

```
Package org.sharma;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class HashMapValuesToList {

    public static void main(String[] args) {
        Map<String,Integer> nameAgeMap = new HashMap<>();
        nameAgeMap.put("John",23);
        nameAgeMap.put("Martin",20);
        nameAgeMap.put("Sam",28);

        List<Integer> ageList = (List<Integer>)nameAgeMap.values();
        System.out.println(ageList);
    }
}
```

Output:

```
Exception in thread "main" java.lang.ClassCastException: class java.util.HashMap$Values cannot be cast to class
java.util.List (java.util.HashMap$Values and java.util.List are in module java.base of loader 'bootstrap')
at org.arpit.java2blog.HashMapValuesToList.main(HashMapValuesToList.java:15)
```

Here is source code of HashMap's values method.

```
public Collection<V> values() {
```

```
Collection<V> vs = values;  
return (vs != null ? vs : (values = new Values()));  
}
```

Fix for java.util.HashMap\$Values cannot be cast to class java.util.List

We can use [ArrayList](#)'s [constructor](#) which takes Collection as parameter to resolve this issue.

```
List<Integer> ageList = new ArrayList<>(nameAgeMap.values());
```

Here is complete program

```
package org.sharma;  
  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
public class HashMapValuesToList {  
    public static void main(String[] args) {  
        Map<String,Integer> nameAgeMap = new HashMap<>();  
        nameAgeMap.put("John",23);  
        nameAgeMap.put("Martin",20);  
        nameAgeMap.put("Sam",28);  
  
        // Use ArrayList's constructor  
        List<Integer> ageList = new ArrayList<>(nameAgeMap.values());  
        System.out.println(ageList);  
    }  
}
```

Output: [23, 20, 28]

Here is source code of ArrayList's constructor

```
/* Constructs a list containing the elements of the specified
 * collection, in the order they are returned by the collection's
 * iterator.
 *
 * @param c the collection whose elements are to be placed into this list
 * @throws NullPointerException if the specified collection is null
 */
public ArrayList(Collection<? extends E> c) {
    elementData = c.toArray();
    size = elementData.length;
    // c.toArray might (incorrectly) not return Object[] (see 6260652)
    if (elementData.getClass() != Object[].class)
        elementData = Arrays.copyOf(elementData, size, Object[].class);
}
```

Unable to obtain LocalDateTime from TemporalAccessor

In this article, we will see how to fix **Unable to obtain LocalDateTime from TemporalAccessor** in Java 8.

Unable to obtain LocalDateTime from TemporalAccessor : Reason

You will generally get this error, when you try to [convert String to LocalDateTime](#) and Formatted String does not have time related information.

Let's understand with the help of example:

```
package org.sharma;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
public class StringToLocalDateTime {

    public static void main(String[] args) {

        // Custom formatted String
        String dateStr = "2022-02-16";

        // Create DateTimeFormatter instance with specified format
        DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");

        // Convert String to LocalDateTime using Parse() method
        LocalDateTime localDateTime = LocalDateTime.parse(dateStr,dateTimeFormatter);

        // Print LocalDateTime object
        System.out.println("LocalDateTime obj: "+localDateTime);
    }
}
```

Output:

```
Exception in thread "main" java.time.format.DateTimeParseException: Text '2022-02-16' could not be parsed: Unable to obtain LocalDateTime from TemporalAccessor: {},ISO resolved to 2022-02-16 of type java.time.format.Parsed
    at java.base/
```

```
java.time.format.DateTimeFormatter.createError(DateTimeFormatter.java:2017)
    at java.base/java.time.format.DateTimeFormatter.parse(DateTimeFormatter.java:1952)
    at java.base/java.time.LocalDateTime.parse(LocalDateTime.java:492)
    at org.arpit.java2blog.StringToLocalDateTime.main(StringToLocalDateTime.java:17)
Caused by: java.time.DateTimeException: Unable to obtain LocalDateTime from TemporalAccessor: {},ISO resolved to 2022-02-16 of type java.time.format.Parsed
    at java.base/java.time.LocalDateTime.from(LocalDateTime.java:461)
    at java.base/java.time.format.Parsed.query(Parsed.java:235)
    at java.base/java.time.format.DateTimeFormatter.parse(DateTimeFormatter.java:1948)
    ... 2 more
```

Caused by: java.time.DateTimeException: Unable to obtain LocalDateTime from TemporalAccessor: {},ISO resolved to 2022-02-16 of type java.time.format.Parsed
at java.base/java.time.LocalDateTime.from(LocalTime.java:431)
at java.base/java.time.LocalDateTime.from(LocalDateTime.java:457)
... 4 more

Unable to obtain LocalDateTime from TemporalAccessor : Fix

LocalDate's `parse()` method with `atStartOfDay()`

As Formatted String does not contain time information, we need to use LocalDate's `parse()` method and call `atStartOfDay()` method to get `LocalDateTime` object.

```
package org.sharma;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class StringToLocalDateTime {
    public static void main(String[] args) {
        // Custom formatted String
        String dateStr = "2022-02-16";

        // Create DateTimeFormatter instance with specified format
        DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
        // Convert String to LocalDateTime using LocalDate's parse() method
        LocalDateTime localDateTime = LocalDate.parse(dateStr,dateTimeFormatter).atStartOfDay();
        // Print LocalDateTime object
        System.out.println("LocalDateTime obj: "+localDateTime);
    }
}
```

Output: LocalDateTime obj: 2022-02-16T00:00

Use `LocalDate` instead of `LocalDateTime`

Since formatted String does not contain time information, we may want to use `LocalDate` rather than `LocalDateTime`.

We can use LocalDate's `parse()` method to convert String to LocalDate in Java.

```
package org.sharma;

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
```



```
public class StringToLocalDate {  
  
    public static void main(String[] args) {  
        // Custom formatted String  
        String dateStr = "2022-02-16";  
  
        // Create DateTimeFormatter instance with specified format  
        DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");  
  
        // Convert String to LocalDateTime using LocalDate's parse() method  
        LocalDate localDate = LocalDate.parse(dateStr,dateTimeFormatter);  
  
        // Print LocalDateTime object  
        System.out.println("LocalDate obj: "+localDate);  
    }  
}
```

Output: LocalDate obj: 2022-02-16

Class names are only accepted if annotation processing is explicitly requested

In this post, we will see how to resolve **class names are only accepted if annotation processing is explicitly requested** in java.

Problem: class names are only accepted if annotation processing is explicitly requested

You will get this error when you are trying to compile java program without **.java** extension.

Let's reproduce this issue with the help of an example:

```
package org.sharma;  
  
public class HelloWorldExample {  
    public static void main(String[] args) {  
        System.out.println("Hello world");  
    }  
}
```

When we will compile the above class with javac as below:

```
C:\Users\Arpit\Desktop\javaPrograms>javac HelloWorldExample  
error: Class names, 'HelloWorldExample', are only accepted if annotation processing is explicitly  
requested  
1 error
```

As you can see got **error:class names are only accepted if annotation processing is explicitly requested** because we did not add **.java** suffix in **HelloWorldExample**.


Solution: class names are only accepted if annotation processing is explicitly requested

We can simply resolve this issue by appending **.java** at the end of file name and it should resolve the issue.

```
C:\Users\Arpit\Desktop\javaPrograms>javac HelloWorldExample.java
```

```
C:\Users\Arpit\Desktop\javaPrograms>
```

Errors in Java

 By. Prabhat Kumar

As you can see, we did not get any error now.

It may also happen if you are doing improper capitalization of `.java` extension while compiling.

```
C:\Users\Arpit\Desktop\javaPrograms>javac HelloWorldExample.Java
error: Class names, 'HelloWorldExample.Java', are only accepted if annotation processing is
explicitly requested
1 error
```

If you notice, filename should be HelloWorldExample.java rather than HelloWorldExample.Java. That's all about how to fix `error: class names are only accepted if annotation processing is explicitly requested`.

Exception in thread “main” `java.util.InputMismatchException`

In this tutorial, we will discuss about exception in thread "main" `java.util.InputMismatchException`.

What causes `java.util.InputMismatchException`?

A Scanner throws this [exception](#) to indicate that the token retrieved does not match the expected type pattern, or that the token is out of range for the expected type.

In simpler terms, you will generally get this error when user input or file data do not match with expected type.

Let's understand this with the help of simple example.

```
package org.sharma;

import java.util.Scanner;

public class ReadIntegerMain {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter any integer: ");

        // This method reads the integer from user
        int num = scanner.nextInt();
        scanner.close();

        // Displaying the integer
        System.out.println("The Integer provided by user: "+num);
    }
}
```

If you put `NA` as user input, you will get below exception.

Output:

Enter any integer: NA

```
Exception in thread "main" java.util.InputMismatchException
at java.base/java.util.Scanner.throwFor(Scanner.java:939)
at java.base/java.util.Scanner.next(Scanner.java:1594)
at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
at org.arpit.java2blog.ReadIntegerMain.main(ReadIntegerMain.java:13)
```

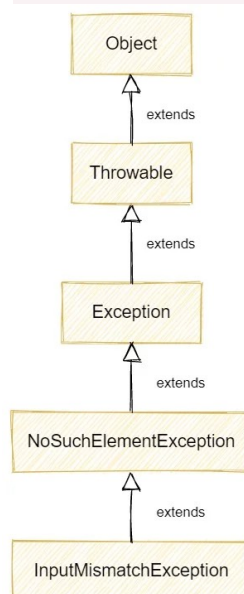
As you can see, we are getting **Exception in thread "main" java.util.InputMismatchException** for input int because user input **NA** is String and does not match with expected input Integer.

Hierarchy of java.util.InputMismatchException

InputMismatchException extends **NoSuchElementException** which is used to denote that request element is not present.

NoSuchElementException class extends the **RuntimeException**, so it does not need to be declared on compile time.

Here is a diagram for hierarchy of **java.util.InputMismatchException**.



Constructor of java.util.InputMismatchException

There are two constructors exist for `java.util.InputMismatchException`

- `InputMismatchException()`: Creates an `InputMismatchException` with null as its error message string.
- `InputMismatchException (String s)`: Creates an `InputMismatchException`, saving a reference to the error message string `s` for succeeding retrieval by the `getMessage()` method.

How to solve java.util.InputMismatchException?

In order to fix this exception, you must verify the input data and you should fix it if you want application to proceed further correctly. This exception is generally caused due to bad data either in the file or user input.

That's all about how to fix exception in thread "main" `java.util.InputMismatchException`.

uses unchecked or unsafe operations. recompile with -xlint:unchecked for details.

In this post, we will see about warning message `uses unchecked or unsafe operations. recompile with -xlint:unchecked for details` in java.

What is warning message: uses unchecked or unsafe operations

`uses unchecked or unsafe operations` is displayed when you compile code which compiler considers to be lacking in error checking or unsafe in some way. It's a warning, not an error, and will not prevent you from compiling the code.

You will generally get this warning when you are using collection without using type specifier (e.g. `ArrayList()` instead of `ArrayList()` in your code.

Example:

Let's say you have simple java code to print [ArrayList](#).

```
package org.sharma;

import java.util.ArrayList;

public class PrintArrayListMain {

    public static void main(String[] args) {
        ArrayList countryList=new ArrayList();
        countryList.add("India");
        countryList.add("China");
        countryList.add("Bhutan");
        System.out.println(countryList);
    }
}
```

When you will compile the code, you will get below output:

```
C:\Users\Arpit\Desktop\javaPrograms>javac PrintArrayListMain.java
Note: PrintArrayListMain.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

As you can see, compile gives us warning message that `PrintArrayListMain.java` uses unchecked or unsafe operations.

We are getting this warning message because we did not use [generics](#) with `ArrayList`.

If you want to know about unchecked or unsafe operations, you can use below command:


```
C:\Users\Arpit\Desktop\javaPrograms>javac PrintArrayListMain.java
Note: PrintArrayListMain.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
C:\Users\Arpit\Desktop\javaPrograms>javac PrintArrayListMain.java -Xlint:unchecked
PrintArrayListMain.java:9: warning: [unchecked] unchecked call to add(E) as a member of the raw
type ArrayList
countryList.add("India");
^
where E is a type-variable:
E extends Object declared in class ArrayList
PrintArrayListMain.java:10: warning: [unchecked] unchecked call to add(E) as a member of the
raw type ArrayList
countryList.add("China");
^
where E is a type-variable:
E extends Object declared in class ArrayList
PrintArrayListMain.java:11: warning: [unchecked] unchecked call to add(E) as a member of the
raw type ArrayList
countryList.add("Bhutan");
^
where E is a type-variable:
E extends Object declared in class ArrayList
3 warnings
```

How to resolve warning message: uses unchecked or unsafe operations.

You can resolve this warning message by using [generics](#) with `Collections`.

In our example, we should use `ArrayList<String>` rather than `ArrayList()`.

Errors in Java

 By. Prabhat Kumar

```
package org.sharma;  
  
import java.util.ArrayList;  
  
public class PrintArrayListMain {  
  
    public static void main(String[] args) {  
        ArrayList<String> countryList=new ArrayList<>();  
        countryList.add("India");  
        countryList.add("China");  
        countryList.add("Bhutan");  
        System.out.println(countryList);  
    }  
}
```

When you will compile above code, you won't get warning message anymore.

```
C:\Users\Arpit\Desktop\javaPrograms>javac PrintArrayListMain.java
```

That's all about how to fix **uses unchecked or unsafe operations. recompile with -xlint:unchecked for details** in java.

uses or overrides a deprecated api. recompile with -xlint:deprecation for details.

In this post, we will see about warning message `uses or overrides a deprecated api. recompile with -xlint:deprecation for details` in java.

What is warning message: uses or overrides a deprecated api

You will get this warning when you are using deprecated api in your code.

Example:

Let's say you have simple swing code which displays a `JFrame` with `JButton` on it.

```
package org.sharma;

import java.awt.FlowLayout;
import javax.swing.*;

public class JFrameDeprecatedAPIExample {
    public static void main(String str[]) {
        JFrame frame = new JFrame("JFrame Example");
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());
        JButton button = new JButton();
        button.setText("Button");
        panel.add(button);
        frame.add(panel);
        frame.setSize(200, 300);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.show();
    }
}
```

When you will compile the code, you will get below output:

```
C:\Users\Arpit\Desktop\javaPrograms>javac JFrameDeprecatedAPIExample.java
Note: JFrameDeprecatedAPIExample.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
```

As you can see, compile gives us warning message that `JFrameDeprecatedAPIExample.java uses or overrides a deprecated API.`

We are getting this warning message because JFrame's `show()` method is deprecated and should not be used.

If you want to know about deprecated API, you can use below command:

```
C:\Users\Arpit\Desktop\javaPrograms>javac JFrameDeprecatedAPIExample.java
-Xlint:deprecation
JFrameDeprecatedAPIExample.java:18: warning: [deprecation] show() in Window has been
deprecated
frame.show();
^
1 warning
```

If you are using maven, you can use `-Dmaven.compiler.showDeprecation=true` with maven goals.

How to resolve warning message: uses or overrides a deprecated api.

You can resolve this warning message by using recommended API.

In our example, we should use `setVisible(true)` rather than `show()` method.

```
package org.sharma;
import java.awt.FlowLayout;
import javax.swing.*;
public class JFrameDeprecatedAPIExample {
    public static void main(String str[]) {
        JFrame frame = new JFrame("JFrame Example");
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());
        JButton button = new JButton();
        button.setText("Button");
        panel.add(button);
        frame.add(panel);
        frame.setSize(200, 300);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

When you will compile above code, you won't get warning message anymore.

```
C:\Users\Arpit\Desktop\javaPrograms>javac JFrameDeprecatedAPIExample.java
```

That's all about how to fix `uses or overrides a deprecated api. recompile with -xlint:deprecation for details` in java.

Java.lang.class not found exception: sun.jdbc.odbc.jdbcodbcdriver

In this post, we will see how to resolve java.lang.classnotfoundexception: sun.jdbc.odbc.jdbcodbcdriver [exception](#) in java.

Problem: java.lang.classnotfoundexception: sun.jdbc.odbc.jdbcodbcdriver

This exception comes in [Java 8](#) because `sun.jdbc.odbc.jdbcodbcdriver` has been removed from JDK and JRE. This class is used to connect database using Object database connectivity driver such as Microsoft access.

Since Java 8 does not support jdbc odbc bridge, you need to use some other alternative solutions to resolve this issue.

Solution 1 : Using UCanAccess jars

You can use `UCanAccess` jars to resolve this issue.

```
<dependency>
  <groupId>net.sf.ucanaccess</groupId>
  <artifactId>ucanaccess</artifactId>
  <version>5.0.0</version>
</dependency>
```

Maven will add all other required dependencies for you.

Here is example code to connect using UCanAccess libraries.

```
package org.sharma;

import java.sql.Connection;
import java.sql.DriverManager;

public class MSAccessUsingUCanAccess {
    public static void main(String args[]){

        String conUrl = "jdbc:ucanaccess://C:\\Users\\Arpit\\Databases\\Employee.accdb";

        try {
            //Getting connection
            Connection con = DriverManager.getConnection(conUrl);

            if(con!=null){
                System.out.println("Connected successfully");
                con.close();
            }
        }
    }
}
```

```
    }  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}  
}
```

Please note that if you are working with large database and using default memory setting, it is recommended to use `mxm` and `xms` parameter to allocate sufficient memory, otherwise you have to set driver's memory property to false as below

```
String conUrl = "jdbc:ucanaccess://C:\\Users\\Arpit\\Databases\\Employee.accdb;memory=false";
```

Solution 2 : Revert Java version to 7 or before

If you are not using features of Java 8, then you can revert your java version to 7 and it will work fine.

That's all about how to resolve `java.lang.classnotfoundexception: sun.jdbc.odbc.jdbcodbcdriver` exception in java.

Create Array from 1 to n in Java

Introduction

In this article, we will look at How to **Create an Array from 1 to n in Java**. We need to Initialize Arrays in a sequence of values ranging from 1 to number N.

Knowing how we can initialize arrays in different ways will give us deeper insights into Array in Java and allow us to gain more experience with handling arrays under critical situations. Let us have a quick peek at the concept of Arrays in Java.

Array in Java

Arrays are a collection of homogenous data i.e. stores values of the same data type. In Java, we create arrays with the new keyword and allocate memory dynamically similar to the concept of [Objects](#) in Java. They can be of primitive type or an Object type.

Syntax to create Array:

```
int arr[] = new int[10];
```

Here, we create an array **arr** of type **int** and size 10.

Create Array from 1 to N in Java

Now, let us discuss in detail different ways to Create Arrays from 1 to N in Java and initialize them with a sequence of values.

Using Simple For loop in Java

When it comes to initializing arrays with a sequence of values, what better than a For loop to perform an action in a sequence.

Important points to note:

- We use for loop and initialize each **index** of our array with the value: **index + 1**
- This ensures that when iterating over the **current index** the array fills its values from 1 to a given number N.
- We print the arrays using the **Arrays.toString()** method, which gives a String representation of an array object.

Let us look at the code snippet.

```
import java.util.*;

public class Java2Blog {

    public static void main(String[] args) {

        System.out.println("Enter Input N:");
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        //Create Array of Size N
        int arr[] = new int[n];
        for(int i=0;i<n;i++)
        {
            //we fill array with values starting from 1 to 10
            arr[i] = i+1;
        }
        System.out.println("Array with values 1 to "+n);
        System.out.println(Arrays.toString(arr));

    }


}
```

Output:

```
Enter Input N:
10
Array with values 1 to 10
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Using IntStream.range() method of Stream API in Java

Errors in Java

 By. Prabhat Kumar

With the advent of Java 8, the [Stream API](#) was introduced in java which brought into the picture the aspects and functionalities of Functional programming. This reduced code redundancy as well.

We will use the **range()** method of [IntStream](#) class that generates a sequence of increasing values between **start** and **end** of Integral values which is exclusive of the range value.

Important points to note:

- If we want to create a stream of numbers from **1 to 10** we need to pass values into the **range()** method as -> **range(1,11)**.
- This will create a stream of integers that can be returned as an array using the **toArray()** method.

Note: [JDK version 8](#) or higher must be installed to locate this approach.

Let us look at the code.

```
import java.util.Arrays;
import java.util.Scanner;
import java.util.stream.IntStream;

public class Java2Blog {

    public static void main(String[] args) {

        System.out.println("Enter Input N:");
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        //Stream Range end = n+1 to include value n in array
        int arr[] = IntStream.range(1, n+1).toArray();

        System.out.println("Array with values 1 to "+n+" using IntStream.range() method");
        System.out.println(Arrays.toString(arr));

    }

}
```

Output:

Enter Input N:

5

Array with values 1 to 5 using `IntStream.range()` method
[1, 2, 3, 4, 5]

Using `IntStream.rangeClosed()` method of Stream API in Java

The **`rangeClosed()`** method is a variation of the `range()` method of the **`IntStream`** class as shown in the above example. It has similar properties to its counterpart. The only difference is that the method creates a Stream of Integers inclusive of the range value.

Hence, if we want to create a stream of numbers from **1 to 10** we need to pass values into the **`rangeClosed()`** method as -> **`rangeClosed(1,10)`**.

Let us look at the code snippet for this method.

```
import java.util.Arrays;
import java.util.Scanner;
import java.util.stream.IntStream;

public class Java2Blog {

    public static void main(String[] args) {

        System.out.println("Enter Input N:");
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        //Stream Range end = n to include value n
        int arr[] = IntStream.rangeClosed(1,n).toArray();

        System.out.println("Array with values 1 to "+n+" using IntStream.rangeClosed() method");
        // Printing String Representation of Array.
        System.out.println(Arrays.toString(arr));

    }

}
```

Output:

```
Enter Input N:
7
Array with values 1 to 7 using IntStream.rangeClosed() method
[1, 2, 3, 4, 5, 6, 7]
```

Using IntStream.iterate() method of Stream API in Java

The **IntStream** class provides the **iterate()** method with the same functionality as a for-loop. It iterates over a given value and generates a sequence of numbers in increasing order.

Important points to note:

- If we need to generate a sequence of values we call the **IntStream.iterate()** method and we determine the increment value for the sequence like **IntStream.iterate(1, i -> i + 1)**, where **1** indicates the starting number of the sequence.
- This ensures that the values follow a sequence. Now, to generate the number up to a certain limit we use the **limit()** method with the **iterate()** method sequence as -> **limit(n)**, where **n** is the input size.
- Next, we collect the generated sequence of integers using the **toArray()** method. The **IntStream** class by default returns an **Integer []** i.e. an **Integer Wrapper class** type array so there is no need to typecast the array.

Now, let us have a look at the code implementation for this approach as well.

```
import java.util.Arrays;
import java.util.Scanner;
import java.util.stream.IntStream;

public class Java2Blog {

    public static void main(String[] args) {

        System.out.println("Enter Input N:");
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        // Determine increment for i = i+1 then limit output to n
        int[] arr = IntStream.iterate(1, i -> i + 1).limit(n).toArray();

        System.out.println("Array with values 1 to "+n+" using IntStream.iterate() method");
```

```
// Printing String Representation of Array.  
System.out.println(Arrays.toString(arr));  
  
}  
  
}
```

Output:

```
Enter Input N:  
6  
Array with values 1 to 6 using IntStream.iterate() method  
[1, 2, 3, 4, 5, 6]
```

In **Java 9**, the overloaded version of the [iterate\(\)](#) method was introduced that provided an argument for the predicate to determine when the stream must terminate or to limit the sequence stream.

Features of this iterate() method in Java 9:

- This eliminates the concern to use the `limit()` method as we now add the predicate to limit the number of elements to **n** in the stream.
- Hence, to use this `iterate()` method with the limiting functionality we define it like this: **`iterate(1, i -> i <= n, i -> i+1)`**, where **1** indicates the starting number of the sequence.

Let us have a quick look at this code snippet as well.

```
import java.util.Arrays;  
import java.util.Scanner;
```

```
import java.util.stream.IntStream;

public class Java2Blog {

    public static void main(String[] args) {

        System.out.println("Enter Input N:");
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        //Adding predicate to terminate the stream when it reaches value n
        int[] arr = IntStream.iterate(1, i -> i<=n, i -> i + 1).toArray();

        System.out.println("Array with values 1 to "+n+" using Java 9 - IntStream.iterate() method");
        // Printing String Representation of Array.
        System.out.println(Arrays.toString(arr));

    }

}
```

Output:

```
Enter Input N:
6
Array with values 1 to 6 using Java 9 - IntStream.iterate() method
[1, 2, 3, 4, 5, 6]
```

Using the Stream class of Stream API in Java

We can also use the **Stream** class as an alternative to the above-discussed ways with the **IntStream** class. We can use the **iterate()** method of the Stream class in the same way we use the **IntStream.iterate()** method.

Important points to note:

- We will use an array of **Integer Wrapper class** type then use the **limit()** method to terminate the stream and collect the stream into an array using the **toArray()** method.
- The **toArray()** method of the Stream class returns an **Object[]** i.e. **Object class** type array by default. Hence, It becomes necessary to typecast this **Object[]** into an **Integer[]** type.
- We typecast the generated stream to **Integer[]** type using the **toArray()** method like this: **toArray(Integer[]::new)** . This converts the returned array to an Integer array.

Let us look at the code implementation below.

```
import java.util.Arrays;
import java.util.Scanner;
import java.util.stream.Stream;

public class Java2Blog {

    public static void main(String[] args) {

        System.out.println("Enter Input N:");
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        //We iterate over stream and return Integer type array by type casting with new keyword
        Integer[] arr = Stream.iterate(1, i -> i + 1).limit(n).toArray(Integer[]::new);

        System.out.println("Array with values 1 to "+n+" using Stream API: ");
        // Printing String Representation of Array.
        System.out.println(Arrays.toString(arr));

    }
}
```

```
Enter Input N:
10
Array with values 1 to 10 using Stream API:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Output:

Using Arrays.setAll() method in Java

In **Java 8**, the **setAll()** method was introduced in the **Arrays** class. It is a very useful method to set the array elements with sequential integers or a fixed value through the entire length of the array.

Important points to note:

- It accepts two arguments – the array and the predicate that computes the value for each index and returns nothing as it is a setter method.
- Using the **setAll()** method we can set the array with a fixed value as well. Here we write a predicate that generates a sequence of integers and call the method as **setAll(arr, i -> i+1)**, where arr is the array to fill.

Let us look at the code implementation for this approach.

```
import java.util.Arrays;
import java.util.Scanner;

public class Java2Blog {
    public static void main(String[] args) {

        System.out.println("Enter Input N:");
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        // we define an array of size n.
        int arr[] = new int [n];
        //We call the setAll() method pass the array and the condition for sequence
        Arrays.setAll(arr, i -> i + 1);

        System.out.println("Array with values 1 to "+n+" using Arrays.setAll() method: ");
        // Printing String Representation of Array.
        System.out.println(Arrays.toString(arr));

    }
}
```

Output:

```
Enter Input N:
15
Array with values 1 to 15 using Arrays.setAll() method:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

Unsupported class file major version 61 in Java

In this post, we will see how to fix Unsupported class file major version 61 in Java.

Reason for Unsupported class file major version 61 in Java

You will get this error when you have compiled the class with Java 17(major version 17) and you are trying to run class file under lower java version (version 16 or below).

All compiled java classes have major version associated with it and it defines which java version has compiled the class.

For example:

Java 17 has major version of **61** where as Java 8 has major version of **52**.

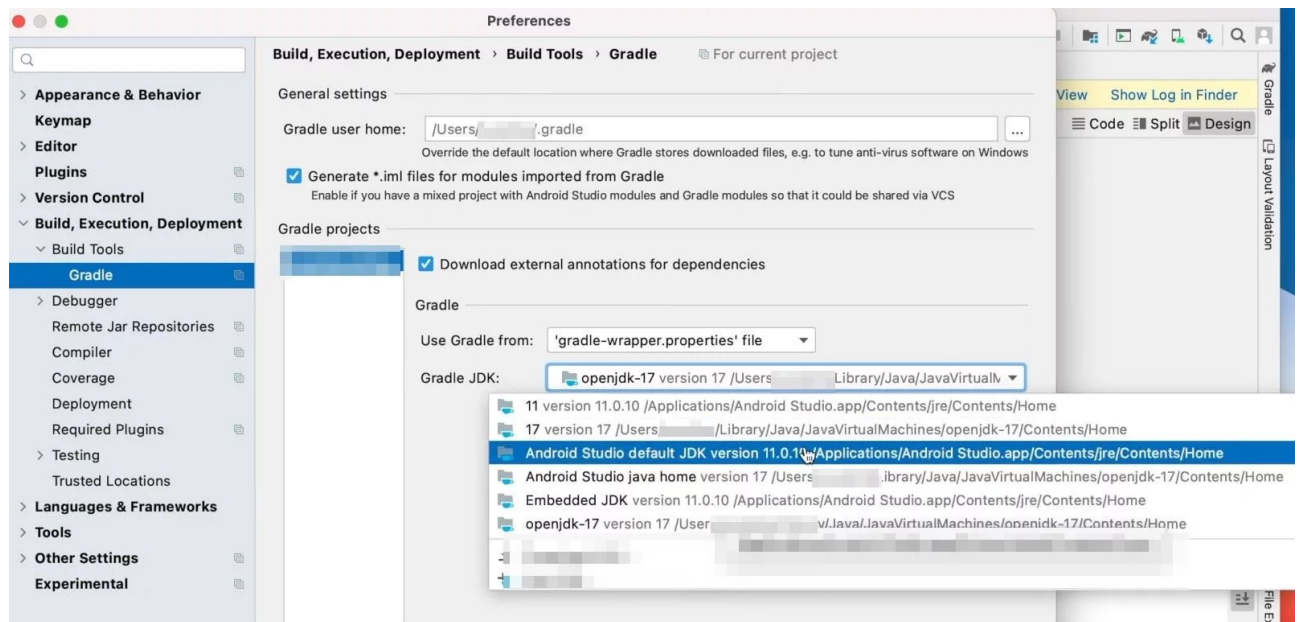
Solution for Unsupported class file major version 61 in Java

You can get this issue in scenarios.

Android studio/IntelliJ Idea with gradle

if you are getting this issue with gradle in Android studio/IntelliJ IDE. You can follow below steps to resolve this.

- Go to **Android Studio --> Preferences**.
- Go to **Build, Execution, Deployment --> Build Tools --> Gradle**.
- Change the Java version to either **Android Studio default JDK** or **Embedded JDK**.
-



You can also try to [upgrade the gradle version](#) to fix this issue.

Any other scenario

If maven/gradle is using Java 17 to build the jar, but you are using Java 11 to deploy the jar to production, you will get **Unsupported class file major version 61** version.

In general, you should check java version in which you are compiling the class and compare it with java version in which class was compiled.

Please note that different IDEs can have different JDKs to compile the class file. You need to make sure that both compile time and run time environment are same.

Find Difference Between Two Instant in Java

In this post, we will see how to find difference between two Instant in Java.

Ways to find difference between two Instant in Java

There are multiple ways to find difference between two Instant in various time units such as Days, Hours, Minutes, Seconds etc.

Using Instant's until() method

To find difference between two Instant, We can use Instant's **until()** method.

Instant's **until()** method returns amount of time with another date in terms of specified unit. You can pass specific **TemporalUnit** such as DAYS, SECONDS to get result in specified units.

For example:

If you want amount of seconds between two Instant, you can use below code:

```
long noOfSeconds = instantBefore.until(instantAfter, ChronoUnit.SECONDS);
```

Here is an example:

```
package org.sharma;

import java.time.Instant;
import java.time.temporal.ChronoUnit;

public class DifferenceBetweenTwoInstantUntil {
    public static void main(String[] args) {
        Instant instantBefore = Instant.parse("2022-09-14T18:42:00Z");
        Instant instantAfter = Instant.parse("2022-09-16T05:25:00Z");

        long noOfDays = instantBefore.until(instantAfter, ChronoUnit.DAYS);
        long noOfHours = instantBefore.until(instantAfter, ChronoUnit.HOURS);
        long noOfMinutes = instantBefore.until(instantAfter,
ChronoUnit.MINUTES);
```

```
    long noOfSeconds = instantBefore.until(instantAfter,
ChronoUnit.SECONDS);

    System.out.println("Days: "+noOfDays);
    System.out.println("Hours: "+noOfHours);
    System.out.println("Minutes: "+noOfMinutes);
    System.out.println("Seconds: "+noOfSeconds);
}
}
```

Output

```
2 Days: 1
3 Hours: 34
4 Minutes: 2083
5 Seconds: 124980
6
```

Using ChronoUnit Enum

We can use **ChronoUnit** method to find difference between two Instant.

It internally calls **until()** method to find the difference between two time units.

For example:

If you want amount of seconds between two Instant, we can use following syntax:

```
long noOfSeconds = ChronoUnit.SECONDS.between(instantBefore,instantAfter);
```

Here is an example:

```
package org.sahrma;

import java.time.Instant;
import java.time.temporal.ChronoUnit;

public class DifferenceBetweenTwoInstant {
    public static void main(String[] args) {
        Instant instantBefore = Instant.parse("2022-09-14T18:42:00Z");
```

```
Instant instantAfter = Instant.parse("2022-09-16T05:25:00Z");

long noOfDays = ChronoUnit.DAYS.between(instantBefore, instantAfter);
long noOfHours = ChronoUnit.HOURS.between(instantBefore, instantAfter);
long noOfMinutes =
ChronoUnit.MINUTES.between(instantBefore, instantAfter);
long noOfSeconds =
ChronoUnit.SECONDS.between(instantBefore, instantAfter);

System.out.println("Days: "+noOfDays);
System.out.println("Hours: "+noOfHours);
System.out.println("Minutes: "+noOfMinutes);
System.out.println("Seconds: "+noOfSeconds);
}
}
```

Output

```
Days: 1
Hours: 34
Minutes: 2083
Seconds: 124980
```

Using Duration's between() method

We can use Duration's **between()** method to find difference between two Instant.

Duration's **between()** method returns amount of time with another date in terms of minutes, hours, seconds and nanoseconds. You can use **toXXX()** method to convert into required unit.

For example:

If you want amount of seconds between two Instant, we can use following syntax:

Here is an example:

```
package org.sharma;

import java.time.Duration;
import java.time.Instant;
```

```
public class DifferenceBetweenTwoInstantDuration {
    public static void main(String[] args) {
        Instant instantBefore = Instant.parse("2022-09-14T18:42:00Z");
        Instant instantAfter = Instant.parse("2022-09-16T05:25:00Z");

        long noOfDays = Duration.between(instantBefore, instantAfter).toDays();
        long noOfHours = Duration.between(instantBefore,
instantAfter).toHours();
        long noOfMinutes = Duration.between(instantBefore,
instantAfter).toMinutes();
        long noOfSeconds = Duration.between(instantBefore,
instantAfter).toSeconds();

        System.out.println("Days: "+noOfDays);
        System.out.println("Hours: "+noOfHours);
        System.out.println("Minutes: "+noOfMinutes);
        System.out.println("Seconds: "+noOfSeconds);
    }
}
```

Output

```
Days: 1
Hours: 34
Minutes: 2083
Seconds: 124980
```