

Forum:[Hack Forums](#)

Thread: [Example program random numbers without repetition](#)

Nemo2478 - August 12, 2019, 1:17 pm

Hello everyone,

used elements:

/* Scanner User Input

" Generate Random Numbers

Exception try-catch Label-continue

foreach loop BigInteger for huge numbers

Methods

*/

Code public class lotto {

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

```
        int givenValue[] = new int[6];
```

```
        int randomValue[] = new int[6];
```

```
        int hits = 0;          //for founded numbers
```

```
        givenValue = determineGivenNumbers(); //Method for User Input
```

```
        randomValue = determineRandomNumbers(); //Generate method for random numbers
```

```
        hits = checkForEquality(givenValue, randomValue); //checks the arrays for equality
```

```
        printArrays(givenValue, randomValue, hits); //returns the arrays and the detected hits
```

```
        binomialcoefficient(); //Calculate the probability of 6 hits out of 49
```

```
    }
```

```
/*-----UserInput-----*/
```

```
    public static int[] determineGivenNumbers(){
```

```
        int givenValue[] = new int[6];
```

```
        Scanner scan = new Scanner(System.in);
```

```
        int checkGivNumber = 0; //so that same numbers can not be entered
```

```
        boolean numOrLetter = false; //so that letters can not be entered
```

```
        System.out.println("Please enter six numbers 1-49!");
```

```
        Böö: for(int i=0; i<givenValue.length; i++){
```

```
            do{
```

```
                try{
```

```
                    numOrLetter = false;
```

```
                    checkGivNumber = scan.nextInt();
```

```
                }catch(Exception e){
```

```
                    System.err.println("Please enter a valid number! " + e.toString());
```

```
                    scan.nextLine();
```

```
                    numOrLetter = true; //if letter is entered error, interrogate again
```

```
            }
```

```

        if(checkGivNumber <= 0 || checkGivNumber > 49) System.err.println("Please give a
number in the specifield range!");
    }while(checkGivNumber <= 0 || checkGivNumber > 49 || numOrLetter == true);

    for(int j=0; j<i; j++){
        if(givenValue[j] == checkGivNumber){    //If entered number twice
            System.err.println("The entered number is already exists!");
            System.out.println("Please enter a new number!");
            i--;    //if equal number then not iterate loop further
            continue Böö;    //then go to Böö, after continue nothing will be executed
        }
    }
    givenValue[i] = checkGivNumber;
}

Arrays.sort(givenValue);

return givenValue;
}
/*-----*/

/*-----Random numbers-----*/
public static int[] determineRandomNumbers(){

    int randomValue[] = new int[6];
    Random rNumber = new Random();
    int checkRanNumber = 0;    //So that same random numbers are not generated

    Böö: for(int i=0; i<randomValue.length; i++){
        checkRanNumber = rNumber.nextInt(49)+1;
        for(int j=0; j<i; j++){
            if(randomValue[j] == checkRanNumber){
                i--;
                continue Böö;
            }
        }
        randomValue[i] = checkRanNumber;
    }

    Arrays.sort(randomValue);

    return randomValue;
}
/*-----*/

/*-----Check arrays for equality-----*/
public static int checkForEquality(int givenValue[], int randomValue[]){
    int hits = 0;
    //Array to check how many numbers are equal
    for(int j=0; j<randomValue.length; j++){
        for(int i=0; i<givenValue.length; i++){
            if(givenValue[i] == randomValue[j]){
                hits++;
            }
        }
    }
    return hits;
}

```

```

    }
    }
}
return hits;
}

/*-----*/

/*-----Output arrays and the determined hits-----*/
public static void printArrays(int givenValue[], int randomValue[], int hits){
    for(int i : givenValue){
        System.out.print(i + "\t");
    }
    System.out.println();
    for(int i : randomValue){
        System.out.print(i + "\t");
    }
    System.out.println("You have " + hits + " hits!");
}

/*-----*/

/*-----Calculate the probability of 6 hits out of 49-----*/
public static void binomialcoefficient(){

    BigInteger result = BigInteger.ONE;
    int n = 49;
    int k = 6;

    result = faculty(n).divide(faculty(k).multiply(faculty(n-k)));
    System.out.println("The propability for six correct hits are 1 : " + result);
    double d1 = result.doubleValue(); //convert the BigInteger value to double
    System.out.println(1/d1 + " %"); //so that it can be specified in % number
}

//The BigInteger objects always become as long
//as the corresponding results require space
public static BigInteger faculty(int number){
    if(number <= 1) return BigInteger.ONE;
    return BigInteger.valueOf(number).multiply(faculty(number-1));
    //BigInteger.valueOf(long val) Creates a BigInteger that takes the value val
    //with .multiply will this * multiply(zahl) be done
}
}

```

Regards

PS.: I would like that you add my post to this post: [Welcome to Java :: A complete, updated compilation of Tutorials and Source Codes](#)

MavenGradle - August 14, 2019, 2:02 pm

This is frowned upon (the labelling of your code):

Code Böö: for(int i=0; i<givenValue.length; i++){

This can be simplified:

Code while(checkGivNumber <= 0 || checkGivNumber > 49 || numOrLetter == true);Code

while(checkGivNumber <= 0 || checkGivNumber > 49 || numOrLetter);

For each loop has better readability:

```
Code for(int j=0; j<randomValue.length; j++){
    for(int i=0; i<givenValue.length; i++){
        if(givenValue[i] == randomValue[j]){
            hits++;
        }
    }
}
}Code for (int i : randomValue) {
    for (int value : givenValue) {
        if (value == i) {
            hits++;
        }
    }
}
}
```

Please use the Java array style declaration instead of C:

Code int givenValue[]Code int[] givenValue

Error message does not correspond with the exceptions that might occur:

```
Code try{
    numOrLetter = false;
    checkGivNumber = scan.nextInt();
}catch(Exception e){
    System.err.println("Please enter a valid number! " + e.toString());
    scan.nextLine();
    numOrLetter = true;    //if letter is entered error, interrogate again
}
```

Instead of using an array, your code can be simplified by using an ArrayList and calling the Contains method. You should prefer generic types over arrays. Arrays are inflexible and do not have the expressive power of generic types:

Code public static int[] determineGivenNumbers(){

```
    int givenValue[] = new int[6];
```

```
    ...
```

```
for(int j=0; j<i; j++){
    if(givenValue[j] == checkGivNumber){    //If entered number twice
        System.err.println("The entered number is already exists!");
        System.out.println("Please enter a new number!");
        i--;    //if equal number then not iterate loop further
        continue Boo;    //then go to Böö, after continue nothing will be executed
    }
}
```

Code List<Integer> userIntegers = new ArrayList<>();

```
    if (userIntegers.contains(1)) // Show user a warning because 1 is already in the list;
```

This can be simplified:

Code boolean numOrLetter = false;Code boolean numOrLetter;

Spelling mistake:

Code System.out.println("The propability for six correct hits are 1 : " + result);Code

System.out.println("The probability for six correct hits are 1 : " + result);

You declare a Scanner which implements [Closable](#). It is good practice to CLOSE your Scanner when you're done using it:

```
Code Scanner scan = new Scanner(System.in);
scan.close();
```

numOrLetter = false should not be in your try statement, instead place it above the try statement:

Code try{

```
    numOrLetter = false;
    checkGivNumber = scan.nextInt();
} catch (Exception e) {
    System.err.println("Please enter a valid number! " + e.toString());
    scan.nextLine();
    numOrLetter = true;    //if letter is entered error, interrogate again
} Code numOrLetter = false;
```

try{

```
    checkGivNumber = scan.nextInt();
} catch (Exception e) {
    System.err.println("Please enter a valid number! " + e.toString());
    scan.nextLine();
    numOrLetter = true;    //if letter is entered error, interrogate again
}
```

You call scan.nextLine() without exception handling:

Why should this be added to the Java tutorial if you don't follow general Java coding standards? If I saw someone commit this in my company and try to merge this with the master branch I would immediately deny the merge request. Please adjust accordingly and try again. You write code like an old school C programmer. This is Java.

Nemo2478 - August 18, 2019, 1:27 am

Hello MavenGradle,

Quote: You should prefer generic types over arrays. Arrays are inflexible and do not have the expressive power of generic types:

I would not generalize that way, especially arrays here prevent boxing and null.

Quote: You declare a Scanner which implements Closable. It is good practice to CLOSE your Scanner when you're done using it:

However, the scanner uses System.in, which should not be closed despite Closeable ...

Regards

Vanagander - August 18, 2019, 6:07 pm

(Aug 18, 2019 01:27 AM) Nemo2478 Wrote: [\[CITE\]](#) Hello MavenGradle,

I would not generalize that way, especially arrays here prevent boxing and null.

However, the scanner uses System.in, which should not be closed despite Closeable ...

Regards

How does arrays prevent null, exactly?

It is good practice to close a stream after you're done with it, since it can cause a memory leak.

Nemo2478 - August 19, 2019, 6:18 am

Hello Vanagander,

Quote:How does arrays prevent null, exactly?

There can not be a null in an int array, but in an ArrayList <integer> can be a null. Arrays of primitive types can not hold null.

Quote:It is good practice to close a stream after you're done with it, since it can cause a memory leak.

This should only be done with streams that you open yourself. And Sytem.in opens the JVM and closes it.

Just closing it where you use it just causes problems - if you close it, then only at the end of the main if there is only one thread in the program.

There are absolutely no problems with System.in, above all there can not be memory leak...

And yes, "after you're done with it". This is difficult with standard input.

Regards

MavenGradle - August 20, 2019, 9:33 am

(Aug 19, 2019 06:18 AM)Nemo2478 Wrote: [\[CITE\]](#) There can not be a null in an int array, but in an ArrayList <integer> can be a null. Arrays of primitive types can not hold null.

You're missing the point entirely. Sure, an ArrayList can contain null values and you should account for that but your code will be simpler, easier to read and generally comply to the code standards which I have explained to you above.

(Aug 19, 2019 06:18 AM)Nemo2478 Wrote: [\[CITE\]](#) Hello Vanagander,

This should only be done with streams that you open yourself. And Sytem.in opens the JVM and closes it.

Just closing it where you use it just causes problems - if you close it, then only at the end of the main if there is only one thread in the program.

There are absolutely no problems with System.in, above all there can not be memory leak...

If you think the Java Virtual Machine is your one way ticket to leave leaks in your code then you are gravely mistaken. You are not properly closing your Scanner. Please adjust accordingly. I don't care if you're using System.in. If you want your System.in to remain open, wrap your System.in stream in a CloseShieldInputStream that prevents it from being closed. Again, you have failed to follow general coding principles and the other advice mentioned above.

I have personally dealt with Java Garbage Collector bugs because of programmers not properly closing certain resources and the GC simply missing them. They can happen and it took many working hours to fix these issues and report the bugs to the JDK vendor (IBM in this case). They originated both due to a developer not closing certain resource streams and the garbage collector not doing its job properly. I am sick and tired of not properly disposed resourced. No exceptions. Write code properly and don't rely on someone else to do it for you.

Nemo2478 - August 22, 2019, 4:11 am

Hello MavenGradle,

Quote:an ArrayList can contain null values

Nobody has claimed something else.

I've found that in a particular case ("here") an array prevents boxing and null, after which Vanagander has asked to what extent an array would prevent null. The answer is that arrays of primitive types can not hold null. In an int [] x you will not find a null. On the other hand, in a List <Integer> you will find a null.

EDIT: just so we do not rubbing against each other: there is no rating included. If the code - as you mentioned in comment #2 - gets better with a List, why not?

Quote:You're missing the point entirely. Sure, an ArrayList can contain null values and you should account for that but your code will be simpler, easier to read and generally comply to the code standards which I have explained to you above.

You also miss the point: just because you have explained your opinion on a certain point, that is far from a standard.

It needs exactly 6 primitive elements - and that's exactly what an array fulfill. An ArrayList does not make it easy or more readable at the site, nor does it suffice for some popular codestyle. The points you mentioned ("inflexible", "generic") are in this case not a problem, but rather desirable - generic types are more problematic here (yes, it should always be ints) and you do not need flexibility (it should always be 6 sorted elements).

The only advantage of a list would be the contains, but in this case it would work with binarySearch. If you really want something more expressive, are instead of lists SortedSets possible, which are always sorted and directly prevent duplicates - yes, this is exactly what here is needed.

Quote:If you want your System.in to remain open, wrap your System.in stream in a CloseShieldInputStream that prevents it from being closed.

I like the idea (but please not with an external library). Did you visit Stackoverflow? External Lib for such a mini-program is not necessary...

Quote:If you think the Java Virtual Machine is your one way ticket to leave leaks in your code then you are gravely mistaken.

Where do you see memory leaks here? To quote myself:

Quote:There are absolutely no problems with System.in, above all there can not be memory leak... If you think otherwise, then "are you gravely mistaken."

Regards

MavenGradle - August 22, 2019, 9:30 am

Hello Nemo, please do include the correct quote so I get notified of replies. Otherwise it might take days or even until the end of times before you get a reply.

(Aug 22, 2019 04:11 AM)Nemo2478 Wrote: [\[CITE\]](#) Hello MavenGradle,

Nobody has claimed something else.

I've found that in a particular case ("here") an array prevents boxing and null, after which Vanagander has asked to what extent an array would prevent null. The answer is that arrays of primitive types can not hold null. In an int [] x you will not find a null. On the other hand, in a List <Integer> you will find a null.

EDIT: just so we do not rubbing against each other: there is no rating included. If the code - as you mentioned in comment #2 - gets better with a List, why not?

That would be correct. I would use arrays instead of lists in examples like a class that generates ROT13 output. The reason would be because general workings of the methods won't change short term. You yourself just said what kind of List you should use, so why haven't you updated your code yet?

An other example for using lists: Imagine that you worked in a company that currently has 6 active stores open. They ask you to write an application that will get the information from those stores out of the database. Well, you being so eager would use an array and limit it to 6 Store objects, right, because they only have 6 stores. The troubles you'll have next year when they close one of their stores down because their programmers weren't up to par, well, that would certainly play a part in

your application breaking down, now wouldn't it?

You are better off using easy to read and correctly implemented code if you want this to be included in a tutorial. I don't even know why it's so hard to understand that anyone who wants to use your code with more than 6 numbers will have to go over your code, look if you properly used your arrays, instead of just using a list that is flexible and can contain as many numbers as whomever decides, aka make it generic. Your purpose is to be added to a tutorial, then make it applicable to other cases as well or explicitly write that part down. Same thing with your labeled code. No one should do that in production grade code let alone tutorials because it is extremely hard for a new programmer who joins the team to go through said code. You don't just write code for yourself most of the time. This is why it is frowned upon. It makes things unreadable, it makes things harder than they should be. Is that an appropriate explanation or should I go over it again?

You need to split your 'tutorial' in multiple pieces if you want to avoid this kind of advice in the future. You have written something here in the original post that is all over the place. I don't see anything here that anyone can learn from except how to do things absolutely wrong.

(Aug 22, 2019 04:11 AM)Nemo2478 Wrote: [\[CITE\]](#) I like the idea (but please not with an external library). Did you visit Stackoverflow? External Lib for such a mini-program is not necessary... Explain to me what is wrong with using [an external library](#) that has stood the test of time, is open-source and under active development? Under that logic one might not even use the standard Java Class Library.. Why even write Java? Why does Spring exist? If you want to write everything yourself, you should probably get into OS development, hell, maybe even go help TempleOS achieve its purpose.

(Aug 22, 2019 04:11 AM)Nemo2478 Wrote: [\[CITE\]](#) Where do you see memory leaks here? To quote myself:

If you think otherwise, then "are you gravely mistaken."

There is no memory leak. You are correct. I never said as much. I said, close the damn thing. If it implements a Closable, you close it. That's not that hard, right? Again, if you want this to be included in a tutorial, you close the damn thing, even if it does nothing. You close it or explain why you don't close it. You think some kid out there won't be like: Oh why is he not closing it, must mean that Scanner objects don't have to be closed then.. You are writing a tutorial. Implement best practices.

All I see here is how NOT to do things. If you want to write a tutorial, why not write a tutorial about code labels on itself (and why you should NOT use them)? Why not write a tutorial on arrays on their own? Why not write a tutorial on how to use a Scanner? Why not write a tutorial on how to use lists? Why not write a tutorial on how to generate a random number generator without repetition using best coding practices? Surely you must see that by now. Do it properly like you know it should be done. Here we are writing about these stupid fucking details that all add up into a tutorial that leads to nothing but bad practice and badly written code and you know it. You needed someone to write paragraph over paragraph over paragraph to explain this to you? You have yet to address the possible crashes of your application. Is this a proper tutorial? How hard do you want me to come down on you?

(Aug 22, 2019 04:11 AM)Nemo2478 Wrote: [\[CITE\]](#) The only advantage of a list would be the contains, but in this case it would work with binarySearch.

If you really want something more expressive, are instead of lists SortedSets possible, which are always sorted and directly prevent duplicates - yes, this is exactly what here is needed.

^This right here shows me at least you know how to do it right. Why don't you?

Hello MavenGradle,

(Aug 22, 2019 09:30 AM)MavenGradle Wrote: [\[CITE\]](#) That would be correct. I would use arrays instead of lists in examples like a class that generates ROT13 output. The reason would be because general workings of the methods won't change short term. And the general workings of a function that returns N numbers, changes? o_O

(Aug 22, 2019 09:30 AM)MavenGradle Wrote: [\[CITE\]](#) An other example for using lists: Imagine that you worked in a company that currently has 6 active stores open. They ask you to write an application that will get the information from those stores out of the database. Well, you being so eager would use an array and limit it to 6 Store objects, right, because they only have 6 stores. The troubles you'll have next year when they close one of their stores down because their programmers weren't up to par, well, that would certainly play a part in your application breaking down, now wouldn't it? So the example is not correct/obvious. Of course, using arrays here is quite straightforward because the information is very static. If you have an extremely high number of accesses, then you can save yourself the overhead, that e.g. a hash solution would entail. The choice of data structures is above all based on the accesses and the resulting requirements.

To spin on the example: when I read data, then I can first determine how many elements I need. And if the number changes then, then I create a new array with the new size and copy once the elements....

(Aug 22, 2019 09:30 AM)MavenGradle Wrote: [\[CITE\]](#) You are better off using easy to read and correctly implemented code if you want this to be included in a tutorial. I don't even know why it's so hard to understand that anyone who wants to use your code with more than 6 numbers will have to go over your code, look if you properly used your arrays, instead of just using a list that is flexible and can contain as many numbers as whomever decides, aka make it generic. Your purpose is to be added to a tutorial, then make it applicable to other cases as well or explicitly write that part down. You could write the same sentence if I used lists or any other collections. The limitation to 6 elements occurs in exactly two places, in the two methods that generate the arrays. If I would use lists, the amount would also be in the two methods. The whole code outside would not look different with collections (or, that what would look different with collections, could also look different with arrays).

The code could be a bit easier with matching collections, but with your suggestion he will not.

(Aug 22, 2019 09:30 AM)MavenGradle Wrote: [\[CITE\]](#) Explain to me what is wrong with using an external library that has stood the test of time, is open-source and under active development? Under that logic one might not even use the standard Java Class Library.. Why even write Java? Why does Spring exist? If you want to write everything yourself, you should probably get into OS development, hell, maybe even go help TempleOS achieve its purpose. It's just about minimizing dependencies. The Standard Class Library is not an external dependency. Likewise, it is not an external dependency, even if another colleague contributes code.

I find it always interesting how developers just add any libraries. And that's where most did not even read the terms of the license. It just come over Maven Central.... Sorry, but that's not so easy. You have to deal with that in detail! And you have to bear the consequences. But that is a decision that is usually not made by the developer.

Therefore, I agree with this: Anyone who brings in external dependencies for no real reason makes a clear mistake. This may work on some no-brainer projects in school and college, but in big projects this is a no go.

a) Dependencies are to be minimized.

- b) The lib contains more than you need.
- c) Many libs are dependent on other libs again.
- d) The more libs, the higher the risk of conflict.
- e) Not every lib fulfills the requirements to be used in the company.
- f) Every lib hide a legal risk.

And all the fun, just to make this saving?:

```
Code public class NonClosingInputStream extends FilterInputStream {
    public NonClosingInputStream(InputStream delegate) { super(delegate); }
    public void close() {}
}
```

I did not say it was wrong, but that it is not necessary for such a short code snippet. Integrating external libs here simply does not fit this format. In a "real project" that looks quite different.

Open source and active development are not pros. You do not need additional libs for very simple functions.

(Aug 22, 2019 09:30 AM)MavenGradle Wrote: [\[CITE\]](#) There is no memory leak. You are correct. I never said as much. I said, close the damn thing. If it implements a Closable, you close it. That's not that hard, right? Again, if you want this to be included in a tutorial, you close the damn thing, even if it does nothing. You close it or explain why you don't close it. You think some kid out there won't be like: Oh why is he not closing it, must mean that Scanner objects don't have to be closed then.. You are writing a tutorial. Implement best practices. So I have a completely different point of view on it!

When someone writes a tutorial, he sets clear priorities and has to weigh things up. If a tutorial does not cover a topic, it does not cover that topic. One thinks over a statement and that is then described as understandable as possible. (Typical example: A complex situation is presented in a very simplified way. Then the presentation is possibly even simple wrong, just because certain things were omitted!).

External library in a beginner's tutorial is also extremely wrong: A beginner wants just to copy your code and your code just does not work for him.

But on the one hand you are of course right: I see this AutoCloseable as well as the Disposable at .Net: It has to be closed. Therefore, this is either in a try with ressources or, if it is stored, in an instance variable; then the class will just implement AutoClosable and the problem will be passed on then.

And the solution here in my eyes relatively simple: I just set a public final static scanner
inputScanner = new Scanner (System.in); and then it is created exactly once and stays over the entire runtime and is never closed. (And spelling adapted here to System.in, because here it is not a simple constant for a literal... So similar to System.in). ==> Short and painless for a tutorial. In a real project, I would also think about how I would do it. But would probably be similar to the principle. While I could also imagine an unclosable system.in stream, but then I would regularly have duplicate code in the form of try (Scanner in = new Scanner (SomeClass.in)) {}, so that I would probably stay with the static Scanner object.

(Aug 22, 2019 09:30 AM)MavenGradle Wrote: [\[CITE\]](#) If it implements a Closable, you close it. That's not that hard, right? Yes, it is and will fall on your feet if you just close each Closeable directly. :)

If it implements a Closable, you close it.

Simply closing any Closeable that you find falls on your feet, especially if it's static, and thus you can not control who else is accessing it. My variant is the most meaningful to make Scanners static and leave it open from start to finish.

Regards

Hello MavenGradle,

however, you can overdo it with criticism, and a "You can not use arrays!" falls completely under it.

If one desires to have Streams for generating 6 random, different numbers from 1-49, this would be done e.g. with:

`Code new Random().ints(1, 49+1).distinct().limit(6).toArray()`The 49 would one also pull out as a constant, as well as the 6. That this is not done, serves the format here.

The addition should serve as a clue and avoid a magical 50, because that would really not be conducive to understanding. A "+1", on the other hand, should be easy to understand for most of you who have worked with random or intervals.

You could also pull out the direct addition there,

- e.g. with an `int upperBound = 49 + 1` and `ints(1, upperBound)`,
- or `ints(1, upto(49))` (Method name not set),
- or, if you stand on Lombok, `new Random.intsClosed(1, 49)`,
- or [insert X alternatives here].

Regards