Dasar-Dasar Pemrograman 2: Java Arrays & ArrayList

Fariz Darari (fariz@cs.ui.ac.id)

FAKULTAS
ILMU
KOMPUTER

Why?

Imagine storing a number of integers, what you might do:

```
int num0 = 8;
int num1 = 0;
int num2 = 9;
int num3 = 10;
...
```

This is not convenient!
Arrays make programming lives much easier.

Why?

- The variables we have seen so far are for storing individual values, such as numbers or strings.

- Now, what if we want to store multiple values of the same type?

Answer: Arrays!

What?

- An array is a **sequence** of values of the **same type**.
- The values in an array are called **elements**.
- You can make an array of ints, doubles, objects, etc.

Declaring Arrays

- Array types look like other Java types, except that they are followed by square brackets **[ ]**

```
int[] counts;
double[] values;
```

Creating Arrays

- To create the array itself, you have to use the **new** operator

```
counts = new int[4];
values = new double[size];
```

- The first assignment makes **counts** refer to an array of four integers.
- The second makes **values** refer to an array of doubles, where the number of elements in **values** depends on the value of the variable **size**.

Quiz time

Create an array of 26 chars, and an array of 11 booleans!

Quiz time

Create an array of 26 chars, and an array of 11 booleans!

```
char[] x = new char[26];
boolean[] y = new boolean[11];
```

Creating arrays, with content, in one statement

```
int[] myInts = {9,1,7,7};
boolean[] myBools = {false, false, true};
```

However, be careful not to split the initialization:

```
int[] myInts;
myInts = {9,1,7,7};
```

This gives an error!

If you insist to split, do as follows:

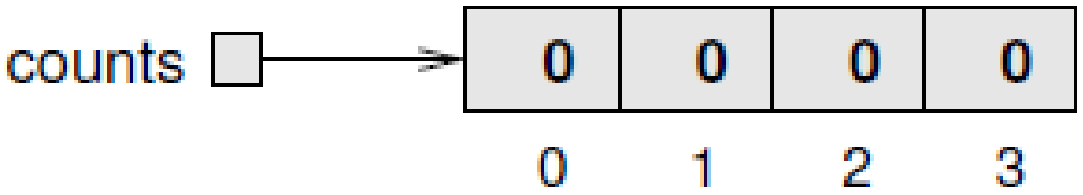
```
int[] myInts;
myInts = new int[]{9,1,7,7};
```

So of course this one is possible too:

```
int[] myInts = new int[]{9,1,7,7};
```

Accessing elements

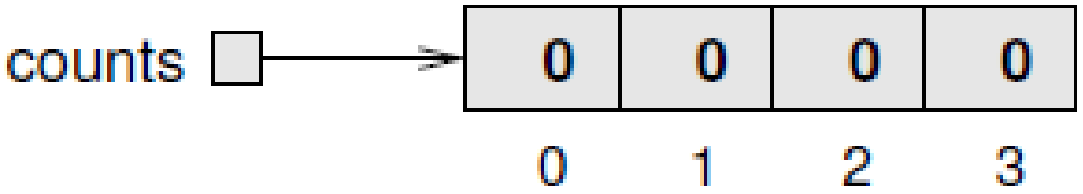
- When creating an array of ints, the elements are initialized to zero.



- The arrow indicates that the value of counts is a **reference** to the array.
- The numbers inside the boxes are the **elements** of the array.
- The numbers outside the boxes are the **indexes**.
- The **[ ]** operator selects elements from an array.
- What's the output of:

```
System.out.println("The zeroth element is " + counts[0]);
```

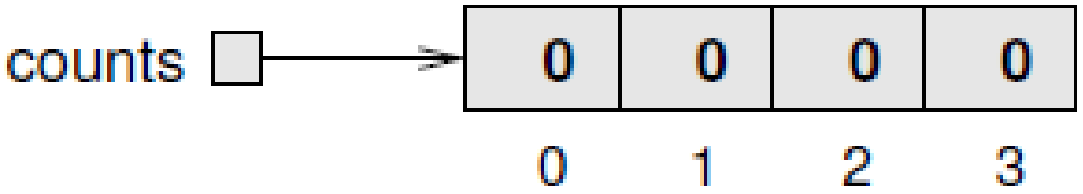
Manipulating array elements



- How's counts look like after executing the following?

```
counts[0] = 7;
counts[1] = counts[0] * 2;
counts[2]++;
counts[3] -= 60;
```

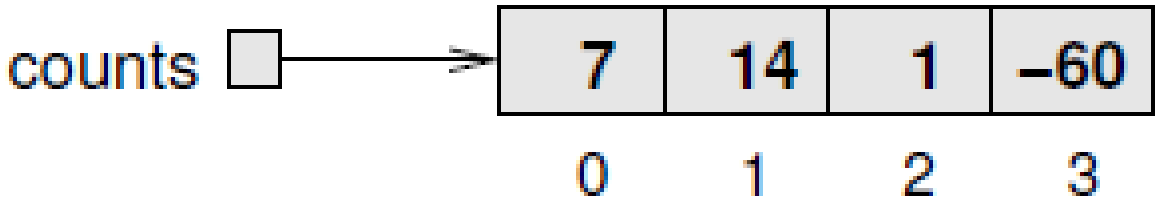
Manipulating array elements



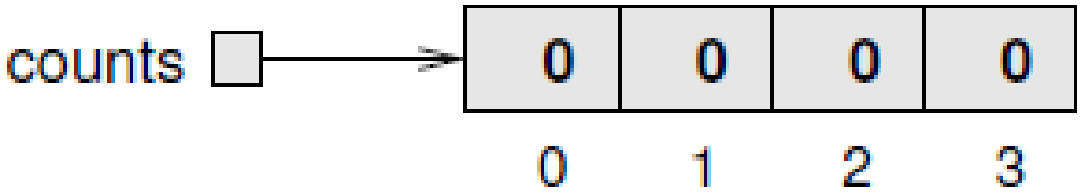
- How's counts look like after executing the following?

```
counts[0] = 7;
counts[1] = counts[0] * 2;
counts[2]++;
counts[3] -= 60;
```

Answer:

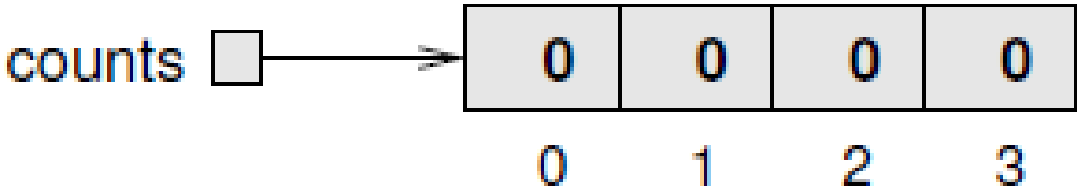


Visiting every element of an array



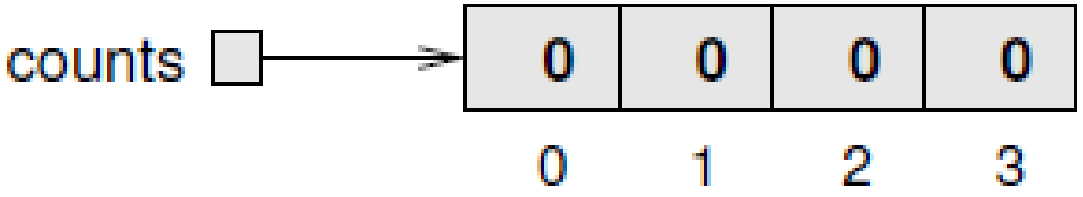

```
for (int i = 0; i < 4; i++) {
    System.out.println(counts[i]);
}
```

Visiting every element of an array (oops)




```
for(int i = 0; i < 5; i++) {
    System.out.println(counts[i]);
}
```

Visiting every element of an array (oops)




```
for(int i = 0; i < 5; i++) {
    System.out.println(counts[i]);
}
```

Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException:
Index 4 out of bounds for length 4

Foreach loop (enhanced for loop)

```
int[] myInts = new int[]{9,1,7,7};
for(int element:myInts) {
    System.out.println(element);
}
```

Output:

```
9
1
7
7
```

Foreach loop (enhanced for loop)

```
int[] myInts = new int[]{9,1,7,7};
for(int element:myInts) {
     System.out.println(element);
}
```

Output:

```
9
1
7
7
```

Foreach loop is best when:

- No need to manipulate array content.
- No need to use the indexes for something.

Quiz time

Create a method **printEven** to print only array elements of even indexes.

Quiz time

Create a method **printEven** to print only array elements of even indexes.

```
public static void printEven(int[] intArray) {

    for(int i = 0; i < intArray.length; i = i + 2)
        System.out.println(intArray[i]);

}
```

Displaying arrays

Be careful when printing arrays:

```
int[] myInts = new int[]{9,1,7,7};
System.out.println(myInts);
```

Output:

```
[I@3ac3fd8b
```

Displaying arrays

Instead, do this:

```
import java.util.Arrays;

int[] myInts = new int[]{9,1,7,7};
System.out.println(Arrays.toString(myInts));
```

Output:

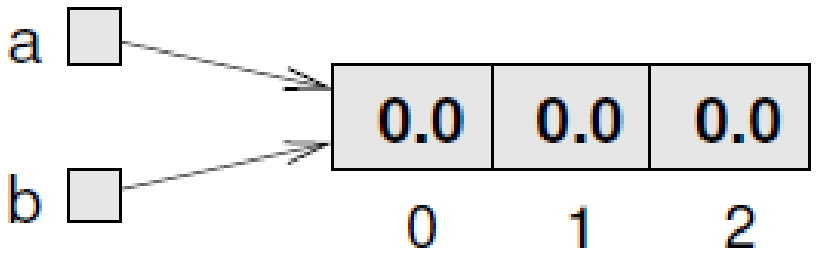
```
[9, 1, 7, 7]
```

Copying arrays

Or copying references? Have a look below:

```
double[] a = new double[3];
double[] b = a;
```

Here's what really happens:



Copying arrays

Or copying references? Have a look below:

```
int[] myInts = new int[]{9,1,7,7};
int[] copyInts = myInts;

System.out.println(myInts);
System.out.println(copyInts);
```

Output:

Copying arrays

Or copying references? Have a look below:

```
int[] myInts = new int[]{9,1,7,7};
int[] copyInts = myInts;

System.out.println(myInts);
System.out.println(copyInts);
```

Output:

```
[I@3ac3fd8b
[I@3ac3fd8b
```

Quiz time: Guess the output

```
int[] myInts = new int[]{9,1,7,7};
int[] copyInts = myInts;
copyInts[2] = 66;
System.out.println(Arrays.toString(myInts));
System.out.println(Arrays.toString(copyInts));
```

Output:

Quiz time: Guess the output

```
int[] myInts = new int[]{9,1,7,7};
int[] copyInts = myInts;
copyInts[2] = 66;
System.out.println(Arrays.toString(myInts));
System.out.println(Arrays.toString(copyInts));
```

Output:

[9, 1, 66, 7]
[9, 1, 66, 7]

Copying arrays, now for real

```
int[] myInts = new int[]{9,1,7,7};
int[] copyInts = new int[4];

for(int i = 0; i<myInts.length; i++)
    copyInts[i] = myInts[i];

copyInts[2] = 66;
System.out.println(Arrays.toString(myInts));
System.out.println(Arrays.toString(copyInts));
```

Output?

Copying arrays, now for real

```
int[] myInts = new int[]{9,1,7,7};
int[] copyInts = new int[4];

for(int i = 0; i<myInts.length; i++)
    copyInts[i] = myInts[i];

copyInts[2] = 66;
System.out.println(Arrays.toString(myInts));
System.out.println(Arrays.toString(copyInts));
```

Output? [9, 1, 7, 7]
[9, 1, 66, 7]

Better copying arrays

```
int[] myInts = new int[]{9,1,7,7};
int[] copyInts = Arrays.copyOf(myInts, myInts.length);
```

The second parameter is the number of elements you want to copy, so you can also use copyOf to copy just part of an array.

Alternative copying arrays

```
int[] myInts = new int[]{9,1,7,7};
int[] copiedInts = new int[3];
System.arraycopy(myInts, 1, copiedInts, 0, 2);
System.out.println(Arrays.toString(copiedInts));
```

Parameters for System.arraycopy(sourceArr, sourcePos, destArr, destPost, length):

- sourceArr: array to be copied from
- sourcePos: starting position in source
- destArr: array to be copied in
- destPos: starting position in destination
- length: copy how many?

Quiz time: What goes wrong?

```
int[] myInts = new int[]{9,1,7,7};
System.out.println(myInts[myInts.length]);
```

Quiz time: What's this code doing?

```
for (int i = 0; i < a.length; i++) {
    a[i] = Math.pow(a[i], 2.0);
}
```

Quiz time: What's this code doing?

```
for (int i = 0; i < a.length; i++) {
    a[i] = Math.pow(a[i], 2.0);
}
```

Answer: squares the elements of a double array

Quiz time: What's this code doing?

```
public static int s(int[] a, int target) {
    for (int i = 0; i < a.length; i++) {
        if (a[i] == target) {
            return i;
        }
    }
    return -1;
```

Quiz time: What's this code doing?

```
public static int s(int[] a, int target) {
    for (int i = 0; i < a.length; i++) {
        if (a[i] == target) {
            return i;
        }
    }
    return -1;
```

Answer: the method takes an int array and an integer value, and it returns the index where the value appears.

Quiz time: Create a method that returns the sum of an array of integers.

Quiz time: Create a method that returns the sum of an array of integers.

```
public static int sum(int[] a) {
    int total = 0;
    for (int i = 0; i < a.length; i++) {
        total += a[i];
    }
    return total;
}
```

Quiz time: Create a method that returns the sum of an array of integers.

```
public static int sum(int[] a) {
    int total = 0;
    for (int i = 0; i < a.length; i++) {
        total += a[i];
    }
    return total;
}
```

Create a foreach loop version of the above method!

Sorting arrays

```
int[] myInts = new int[]{9,1,7,7};
Arrays.sort(myInts);
System.out.println(Arrays.toString(myInts));
```

Sorting arrays

```
int[] myInts = new int[]{9,1,7,7};
Arrays.sort(myInts);
System.out.println(Arrays.toString(myInts));
```

The sorting is done **in-place,**
no need to return to any variable.

Quiz time: Sorting arrays in descending order

```
public static int[] sortDescending(int[] arr) {

    int[] copyArr = Arrays.copyOf(arr,arr.length);
    Arrays.sort(copyArr);
    return reverse(copyArr);

}

public static int[] reverse(int[] arr) {

    int[] reversedArr = new int[arr.length];
    int maxIndex = arr.length-1;
    for(int i = maxIndex; i >= 0; i--)
        reversedArr[maxIndex-i] = arr[i];
    return reversedArr;

}
```

Filling arrays

```
int[] intArr = new int[3];
Arrays.fill(intArr, 100);
System.out.println(Arrays.toString(intArr));
Arrays.fill(intArr, 1,3,7);
System.out.println(Arrays.toString(intArr));
```

Output:

[100, 100, 100]
[100, 7, 7]

Check array content equality

```
int[] intArr = new int[3];
Arrays.fill(intArr, 100);
int[] intArrAnother = new int[3];
Arrays.fill(intArrAnother, 100);
System.out.println(intArr == intArrAnother);
System.out.println(Arrays.equals(intArr, intArrAnother));
```

Output:

false

true

Pass by reference

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

    int[] intArr = {1,3,2};
    mystery(intArr);
    System.out.println(Arrays.toString(intArr));
}

public static void mystery(int[] arr) {
    arr[1] = 899;
}
```

Output: [1, 899, 2]

Enlarge array capacity

You can't. What you can do is to copy your array to a larger array.

```
int[] arr = {5,1,2,1,3};
int[] bigArr = new int[10];
for(int i = 0; i < arr.length; i++) {
    bigArr[i] = arr[i];
}
arr = bigArr;
System.out.println(Arrays.toString(arr));
```

Multi-dimensional arrays

- So far our arrays are one-dimensional:

```
double[] arr = new double[]{0.5,2.5,2.0,5.0};
```

- Suppose you want to store the content of a numeric table or a matrix, then we have to go deeper, adding more dimensions!

```
double[][] twoDArr = {{0.5,2.5,2.0,5.0},
                      {1.5,0.5,1.0,7.0},
                      {3.5,1.5,3.0,1.0}};
```

2-D arrays: Guess the output?

```
double[][] twoDArr = {{0.5,2.5,2.0,5.0},
                      {1.5,0.5,1.0,7.0},
                      {3.5,1.5,3.0,1.0}};


System.out.println(Arrays.toString(twoDArr));
```

2-D arrays: Guess the output?

```
double[][] twoDArr = {{0.5,2.5,2.0,5.0},
                      {1.5,0.5,1.0,7.0},
                      {3.5,1.5,3.0,1.0}};
```

```
System.out.println(Arrays.toString(twoDArr));
```

[[D@3ac3fd8b, [D@5594a1b5, [D@6a5fc7f7]

2-D arrays: Guess the output?

```
double[][] twoDArr = {{0.5,2.5,2.0,5.0},
                      {1.5,0.5,1.0,7.0},
                      {3.5,1.5,3.0,1.0}};
```

```
System.out.println(Arrays.toString(twoDArr));
```

[[D@3ac3fd8b, [D@5594a1b5, [D@6a5fc7f7]

Why?
Because twoDArr is an array of 3 arrays!

2-D arrays: Guess the output?

```
double[][] twoDArr = {{0.5,2.5,2.0,5.0},
                      {1.5,0.5,1.0,7.0},
                      {3.5,1.5,3.0,1.0}};


System.out.println(Arrays.toString(twoDArr[1]));
```

2-D arrays: Guess the output?

```
double[][] twoDArr = {{0.5,2.5,2.0,5.0},
                      {1.5,0.5,1.0,7.0},
                      {3.5,1.5,3.0,1.0}};
```

```
System.out.println(Arrays.toString(twoDArr[1]));
```

[1.5, 0.5, 1.0, 7.0]

2-D arrays: Guess the output?

```
double[][] twoDArr = {{0.5,2.5,2.0,5.0},
                      {1.5,0.5,1.0,7.0},
                      {3.5,1.5,3.0,1.0}};
```

```
System.out.println(Arrays.toString(twoDArr[1][3]));
```

2-D arrays: Guess the output?

```
double[][] twoDArr = {{0.5,2.5,2.0,5.0},
                      {1.5,0.5,1.0,7.0},
                      {3.5,1.5,3.0,1.0}};
```

```
System.out.println(Arrays.toString(twoDArr[1][3]));
```

Error: java: no suitable method found for toString(double)

2-D arrays: Guess the output?

```
double[][] twoDArr = {{0.5,2.5,2.0,5.0},
                      {1.5,0.5,1.0,7.0},
                      {3.5,1.5,3.0,1.0}};


System.out.println(twoDArr[1][3]);
```

2-D arrays: Guess the output?

```
double[][] twoDArr = {{0.5,2.5,2.0,5.0},
                      {1.5,0.5,1.0,7.0},
                      {3.5,1.5,3.0,1.0}};

System.out.println(twoDArr[1][3]);
```

7.0

2-D arrays: Guess the output?

```
double[][] twoDArr = {{0.5,2.5,2.0,5.0},
                      {1.5,0.5,1.0,7.0},
                      {3.5,1.5,3.0,1.0}};


System.out.println(twoDArr[2][1]);
```

2-D arrays: Guess the output?

```
double[][] twoDArr = {{0.5,2.5,2.0,5.0},
                      {1.5,0.5,1.0,7.0},
                      {3.5,1.5,3.0,1.0}};

System.out.println(twoDArr[2][1]);
```

1.5

2-D arrays: Guess the output?

```
double[][] twoDArr = {{0.5,2.5,2.0,5.0},
                      {1.5,0.5,1.0,7.0},
                      {3.5,1.5,3.0,1.0}};


System.out.println(twoDArr[0][2]);
```

2-D arrays: Guess the output?

```
double[][] twoDArr = {{0.5,2.5,2.0,5.0},
                      {1.5,0.5,1.0,7.0},
                      {3.5,1.5,3.0,1.0}};

System.out.println(twoDArr[0][2]);
```

2.0

Quiz time: Create a method to print the content of a 2D-array of doubles!

For example, given:

```
double[][] twoDArr = {{0.5,2.5,2.0,5.0},
                      {1.5,0.5,1.0,7.0},
                      {3.5,1.5,3.0,1.0}};
```

`print2D(twoDArr)` will print:

```
0.5 2.5 2.0 5.0
1.5 0.5 1.0 7.0
3.5 1.5 3.0 1.0
```

Quiz time: Create a method to print the content of a 2D-array of doubles!

```
public static void print2D(double[][] arr) {

    }
```

Quiz time: Create a method to print the content of a 2D-array of doubles!

```
public static void print2D(double[][] arr) {

        for(int i = 0; i < arr.length; i++) {
            double[] insideArr = arr[i];
            for(int j = 0; j < insideArr.length; j++) {
                System.out.print(insideArr[j] + " ");
            }
            System.out.println();
        }

    }
```

Quiz time: What's the output?

```
double[][] twoDArr = {{0.5,2.5,2.0,5.0},
                      {},
                      {3.5,1.5}};
print2D(twoDArr);
```

Quiz time: What's the output?

```
double[][] twoDArr = {{0.5,2.5,2.0,5.0},
                      {},
                      {3.5,1.5}};
print2D(twoDArr);
```

```
0.5 2.5 2.0 5.0


3.5 1.5
```

Quiz time: What's the output?

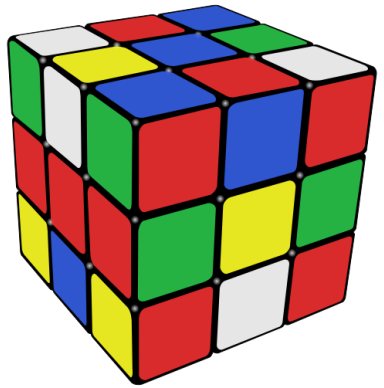
```
double[][] twoDArr = {{0.5,2.5,2.0,5.0},
                      {},
                      {3.5,1.5}};
print2D(twoDArr);
```

```
0.5 2.5 2.0 5.0


3.5 1.5
```

The arrays inside an array do not have to have the same length!

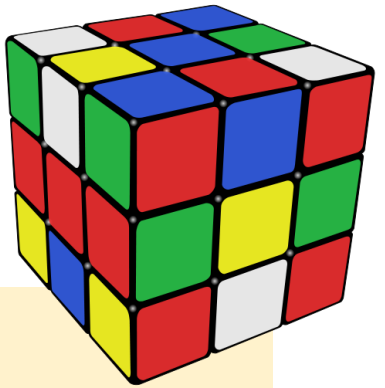
3-D arrays



- Yes, we can go further.
- Previously, our arrays contain arrays. Now, what if our arrays contain arrays of arrays?

```
int[][][] threeDArr1 =  new int[3][3][3];
int[][][] threeDArr2 = {{{1,2,3},{3,2,1},{2,1,5}},
                        {{5,2,5},{1,1,1},{7,1,0}},
                        {{4,6,7},{4,5,4},{4,6,6}}};
```

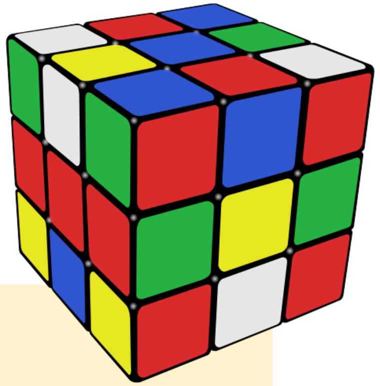
Quiz time: What's the output?



```
int[][][] threeDArr1 = new int[3][3][3];
int[][][] threeDArr2 = {{{1,2,3},{3,2,1},{2,1,5}},
                        {{5,2,5},{1,1,1},{7,1,0}},
                        {{4,6,7},{4,5,4},{4,6,6}}};

System.out.println(threeDArr1[0][0][1]);
System.out.println(Arrays.toString(threeDArr1[0][0]));
System.out.println(threeDArr2[0][1][2]);
System.out.println(threeDArr2[2][1][1]);
System.out.println(Arrays.toString(threeDArr2[1][2]));
System.out.println(Arrays.toString(threeDArr2[0]));
```

Quiz time: What's the output?



```
int[][][] threeDArr1 = new int[3][3][3];
int[][][ ...                                  }},
                                              }},
                                              }}};

System.o ...
System.o ...                                  ][0]));
System.o ...
System.o ...
System.o ...                                  ][2]));
System.out.println(Arrays.toString(threeDArr2[0]));
```

Output:

0

[0, 0, 0]

1

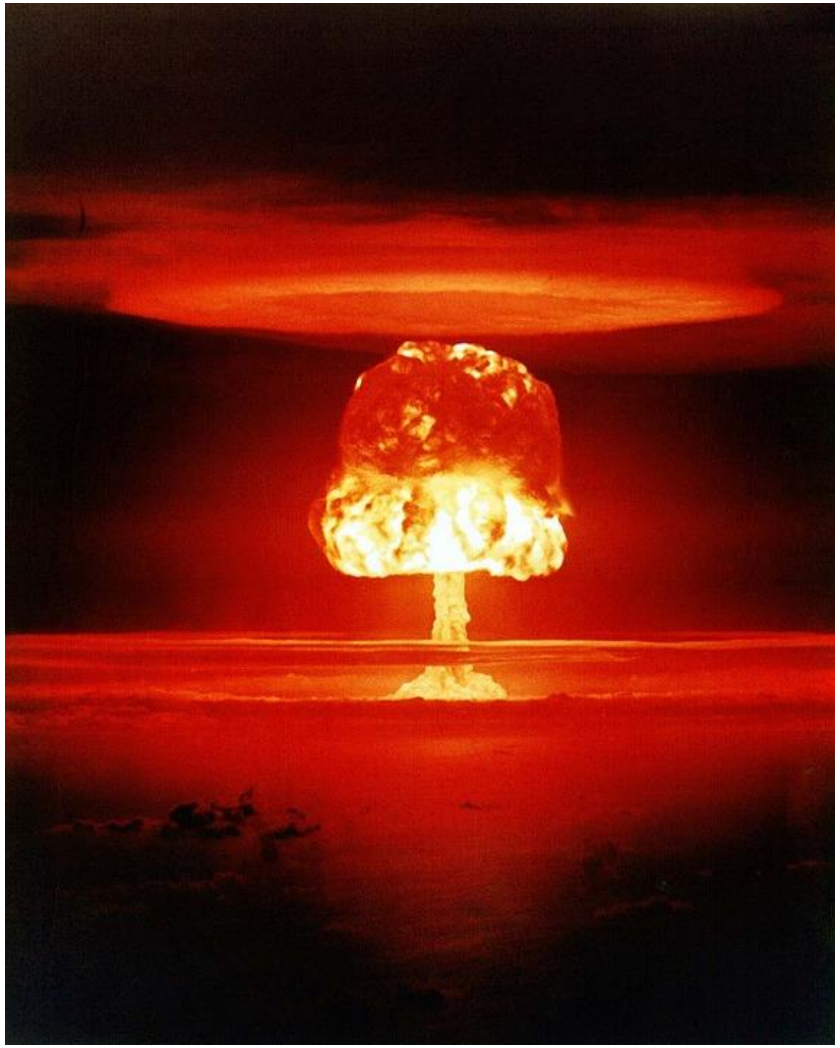
5

[7, 1, 0]

[[I@3ac3fd8b, [I@5594a1b5, [I@6a5fc7f7]

4-D arrays and more..

4-D arrays and more..



Boom!

ArrayList

- So far our arrays are of **fixed-size**.
- Can be a limitation when we need to store things with no predefined size.
- Thankfully, Java has provided **ArrayList** to make our programming lives easier:
  - Flexible size.
  - Can store any kind of objects.
  - Comes with a set of useful methods to work with.

Declaring and creating ArrayList

- First things first, import the ArrayList class:

```
import java.util.ArrayList;
```

- Then, we create an ArrayList, like this:

```
ArrayList lst = new ArrayList();
```

- Or like this, only if **you know the type** of objects to be put in:

```
ArrayList<T> lst = new ArrayList<T>();
```

where T is a type (Integer, Double, String, etc).

ArrayList example (1)

```
ArrayList<String> todoList = new ArrayList<String>();
todoList.add("Eat");
todoList.add("Pray");
todoList.add("Code");
System.out.println(todoList.get(2));
System.out.println(todoList);
System.out.println(todoList.size());
```

ArrayList example (1)

```
ArrayList<String> todoList = new ArrayList<String>();
todoList.add("Eat");
todoList.add("Pray");
todoList.add("Code");
System.out.println(todoList.get(2));
System.out.println(todoList);
System.out.println(todoList.size());
```

Output:

Code

[Eat, Pray, Code]

3

ArrayList example (2)

```
ArrayList<String> todoList = new ArrayList<String>();
todoList.add("Eat");
todoList.add("Pray");
todoList.add("Code");
todoList.remove(2);
todoList.add("Game");
todoList.remove("Eat");
todoList.add(1,"Coffee");
System.out.println(todoList);
```

ArrayList example (2)

```
ArrayList<String> todoList = new ArrayList<String>();
todoList.add("Eat");
todoList.add("Pray");
todoList.add("Code");
todoList.remove(2);
todoList.add("Game");
todoList.remove("Eat");
todoList.add(1,"Coffee");
System.out.println(todoList);
```

Output:

[Pray, Coffee, Game]

ArrayList example (3)

```
ArrayList<String> todoList = new ArrayList<String>();
todoList.add("Eat");
todoList.add("Pray");
todoList.add("Code");
todoList.set(2,"Love");
todoList.set(0,"Shopping");
todoList.set(2,"Code");
System.out.println(todoList);
```

ArrayList example (3)

```
ArrayList<String> todoList = new ArrayList<String>();
todoList.add("Eat");
todoList.add("Pray");
todoList.add("Code");
todoList.set(2,"Love");
todoList.set(0,"Shopping");
todoList.set(2,"Code");
System.out.println(todoList);
```

Output:

[Shopping, Pray, Code]

ArrayList example (4)

```
ArrayList<String> todoList = new ArrayList<String>();
todoList.add("Eat");
todoList.add("Pray");
todoList.add("Code");
System.out.println(todoList.indexOf("Pray"));
System.out.println(todoList.indexOf("Sleep"));
todoList.clear();
System.out.println(todoList);
```

ArrayList example (4)

```
ArrayList<String> todoList = new ArrayList<String>();
todoList.add("Eat");
todoList.add("Pray");
todoList.add("Code");
System.out.println(todoList.indexOf("Pray"));
System.out.println(todoList.indexOf("Sleep"));
todoList.clear();
System.out.println(todoList);
```

Output:

1

-1

[]

Copying array to ArrayList

```
int[] arr =  {3,1,3};
ArrayList<Integer> lst = new ArrayList<Integer>();
for(int i = 0; i < arr.length; i++)
   lst.add(arr[i]);
System.out.println(lst);
```

Copying array to ArrayList

```
int[] arr =  {3,1,3};
ArrayList<Integer> lst = new ArrayList<Integer>();
for(int i = 0; i < arr.length; i++)
   lst.add(arr[i]);
System.out.println(lst);
```

Output:

[3,1,3]

Copying array to ArrayList

```
int[] arr =  {3,1,3};
ArrayList<Integer> lst = new ArrayList<Integer>();
for(int i = 0; i < arr.length; i++)
   lst.add(arr[i]);
System.out.println(lst);
```

Output:

[3,1,3]

Auto-Boxing: Automatically convert from primitive type to object type (via wrapper class), here, it's from int to Integer.

Can also be applied for double (to Double), boolean (to Boolean), and so on..

Copying array to ArrayList, shorter code

```
String[] arr = new String[]{"A","B","C"};
ArrayList<String> lst = new ArrayList<String>(Arrays.asList(arr));
```

Copying ArrayList to array

```
String[] arr = new String[]{"A","B","C"};
ArrayList<String> lst = new ArrayList<String>(Arrays.asList(arr));
String[] arrNew = new String[lst.size()];
lst.toArray(arrNew);
System.out.println(Arrays.toString(arrNew));
```

Copying ArrayList to array

```
String[] arr = new String[]{"A","B","C"};
ArrayList<String> lst = new ArrayList<String>(Arrays.asList(arr));
String[] arrNew = new String[lst.size()];
lst.toArray(arrNew);
System.out.println(Arrays.toString(arrNew));
```

Output:

```
[A, B, C]
```

Quiz time: Remove all occurrences of a specific String from an ArrayList of String

Quiz time: Remove all occurrences of a specific String from an ArrayList of String

```
String[] arr = new String[]{"A","B","C","A","A"};
ArrayList<String> lst = new ArrayList<String>(Arrays.asList(arr));
while(lst.contains("A"))
    lst.remove("A");
System.out.println(lst);
```

Useful methods from java.util.Collections for ArrayList

```
import java.util.Collections;
```

```
String[] arr = new String[]{"A","B","C","A","A"};
ArrayList<String> lst = new ArrayList<String>(Arrays.asList(arr));
Collections.sort(lst);
System.out.println(lst);
System.out.println(Collections.min(lst));
System.out.println(Collections.max(lst));
Collections.shuffle(lst);
System.out.println(lst);
Collections.fill(lst, "Z");
System.out.println(lst);
```

Useful methods from java.util.Collections for ArrayList

```
import java.util.Collections;
```

```
String[] arr = new String[]{"A","B","C","A","A"};
ArrayList<String> lst = new ArrayList<String>(Arrays.asList(arr));
Collections.sort(lst);
System.out.println(lst);
System.out.println(Collections.min(lst));
System.out.println(Collections.max(lst));
Collections.shuffle(lst);
System.out.println(lst);
Collections.fill(lst, "Z");
System.out.println(lst);
```

Output:

[A, A, A, B, C]
A
C
[A, A, C, A, B]
[Z, Z, Z, Z, Z]

Credits: Chapter 8 of Think Java book by Allen Downey and Chris Mayfield