Java Programming

Arrays

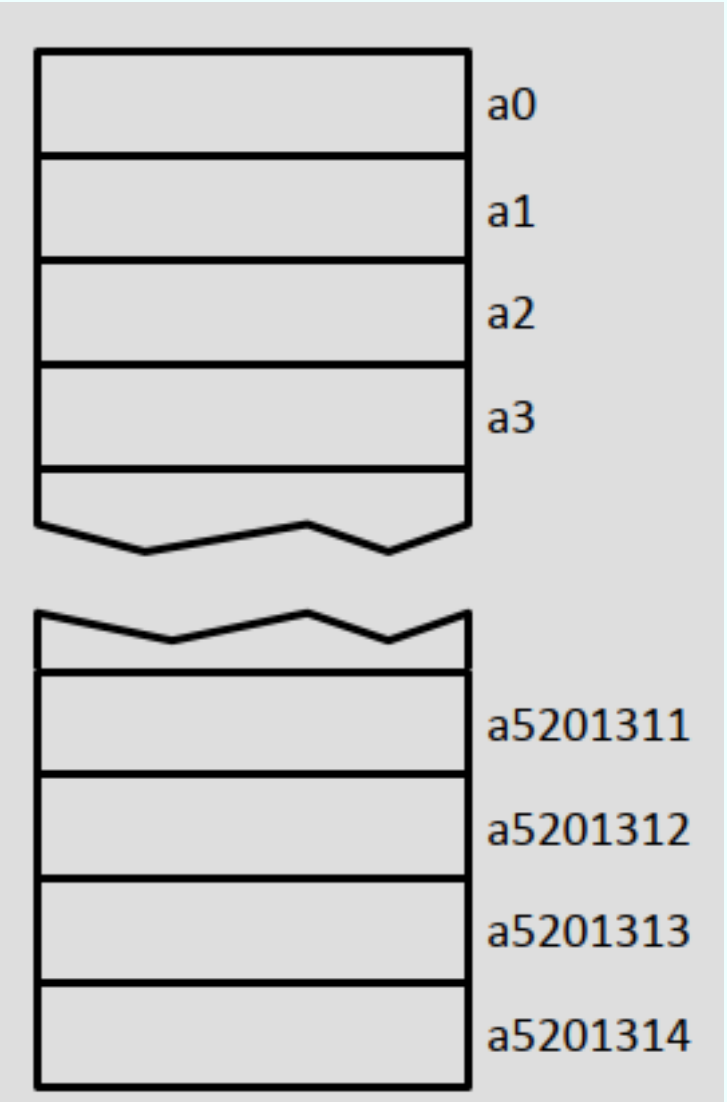
By Kevin 2022/2023

Basic of Arrays

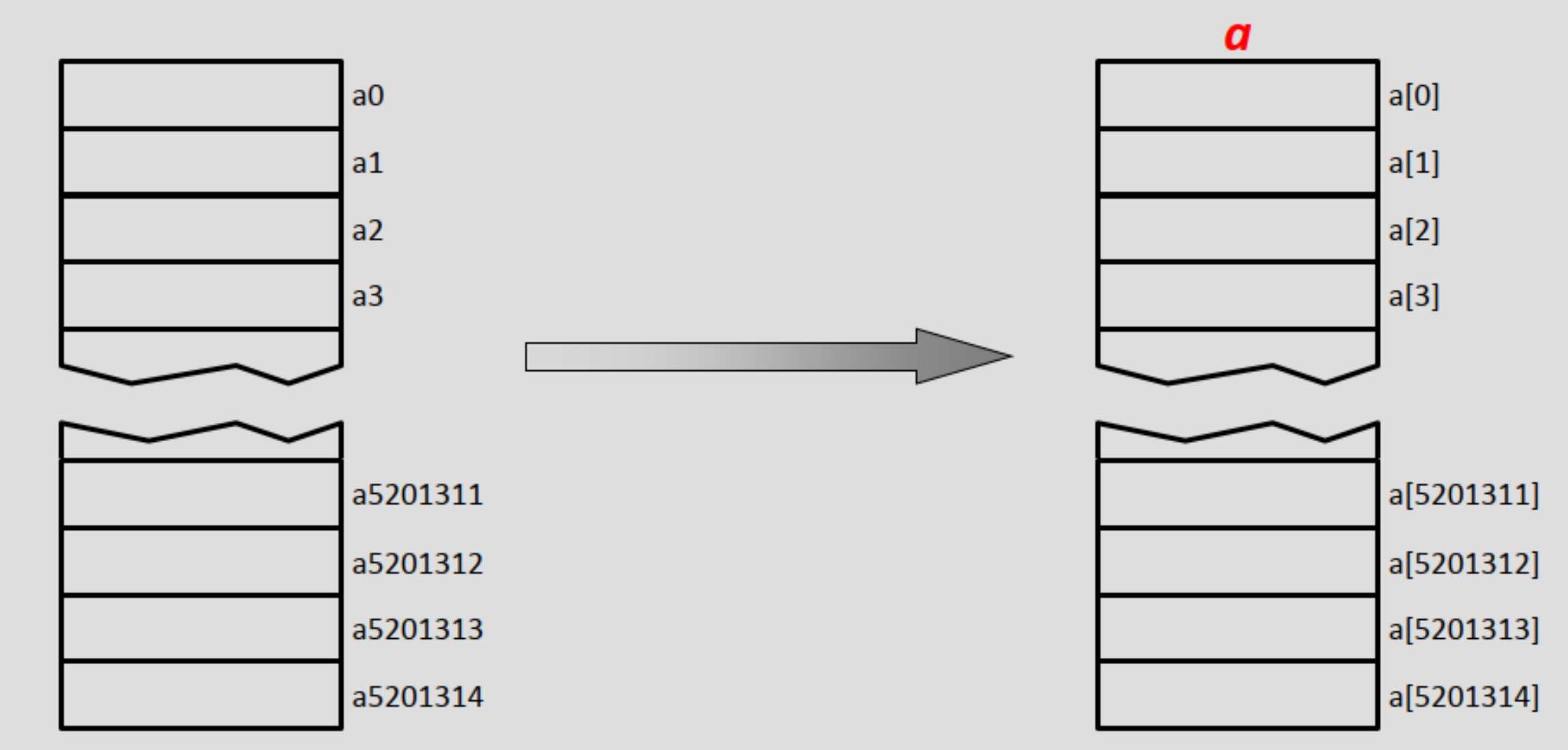
- Suppose you have many variables, all of the same type (e.g., all integers)
- Name them as `a0`, `a1`, `a2`, ..., `a5201314`...
- Let say, find all number > 30...

```
if (a0 > 30)
    System.out.print(a0);
if (a1 > 30)
    System.out.print(a1);
...
if (a5201314 > 30)
    System.out.print(a5201314);
```

- Need a better way to organize your variables.



Basic of Arrays

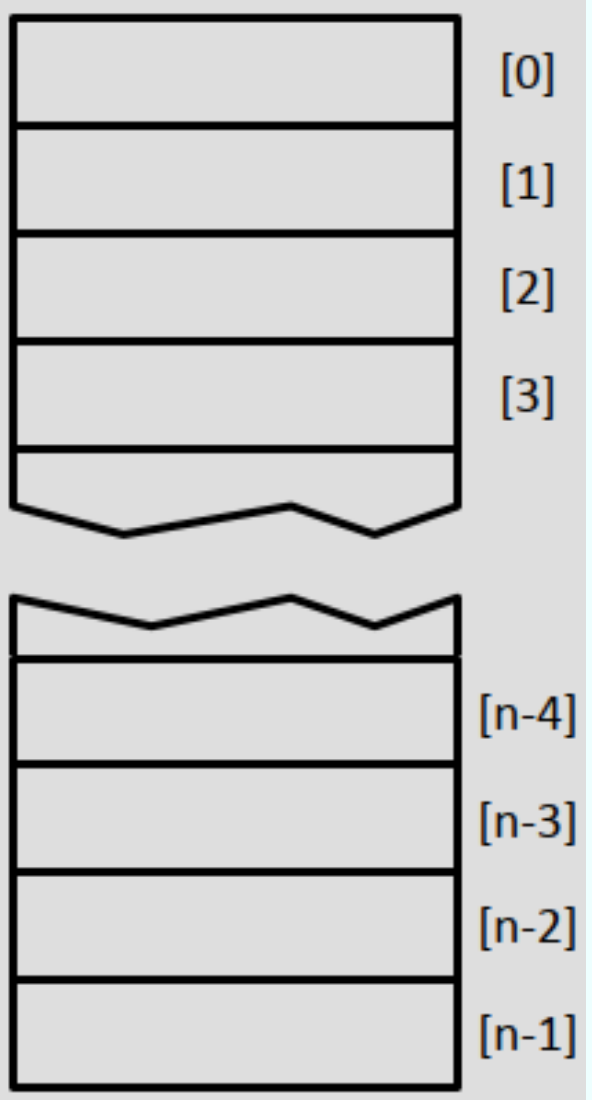


Basic of Arrays

image credit: https://solecollector.com/news/2015/05/how-to-store-sneakers

Basic of Arrays

- A set of variables
- A consecutive memory that consists of the same type (int/float/double/...).
- Allow us to **loop** the variables
- Each element is identified by an index
- Memory location of each element can be calculated from its index



Basic of Arrays

- The first element has the index 0
- The size of an array can be obtained by `array.length`
- Index is represented inside square bracket `[]`:
  `array[0], array[1], array[2], ... array[n-1]`
- Index must be an integer **between 0 to** `array.length - 1`.

Example

```
//assume arr is an int array of size 3.
System.out.println(arr.length); //print 3
arr[0] = 1;
arr[1] = 5;
arr[2] = 6;
for (int i = 0; i < 3; i++)
    System.out.print(arr[i] + " "); //print 1 5 6
```

Array declaration

- To declare an integer variable...

```
int myInt = 100;
```

- To declare an integer array...

```
int[] myArray = new int[5];
```

- This declare `myArray` as an array so that holds 5 integers.

When Declaring a variable

This is what actually happens when you declare and initialize a variable...

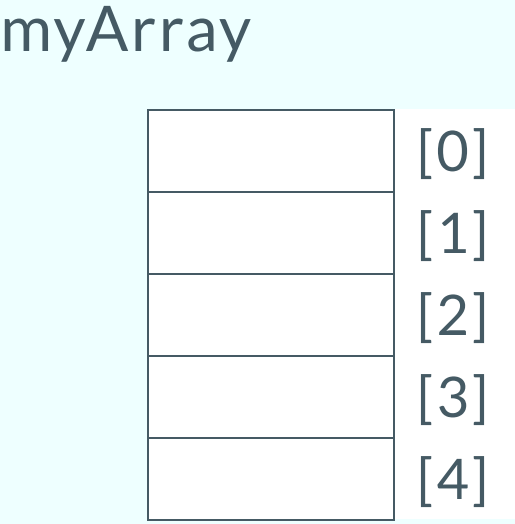
```
int myInt = 100;
```

1. A piece of memory enough for storing an integer (4 bytes) is reserved
2. The value, 100, is stored in that memory
3. From then on, whenever myInt is used, the program refers to the value stored in this memory location
4. Of course, you can assign some other values to it

When Declaring an array

This is what actually happens when you declare and initialize an array...

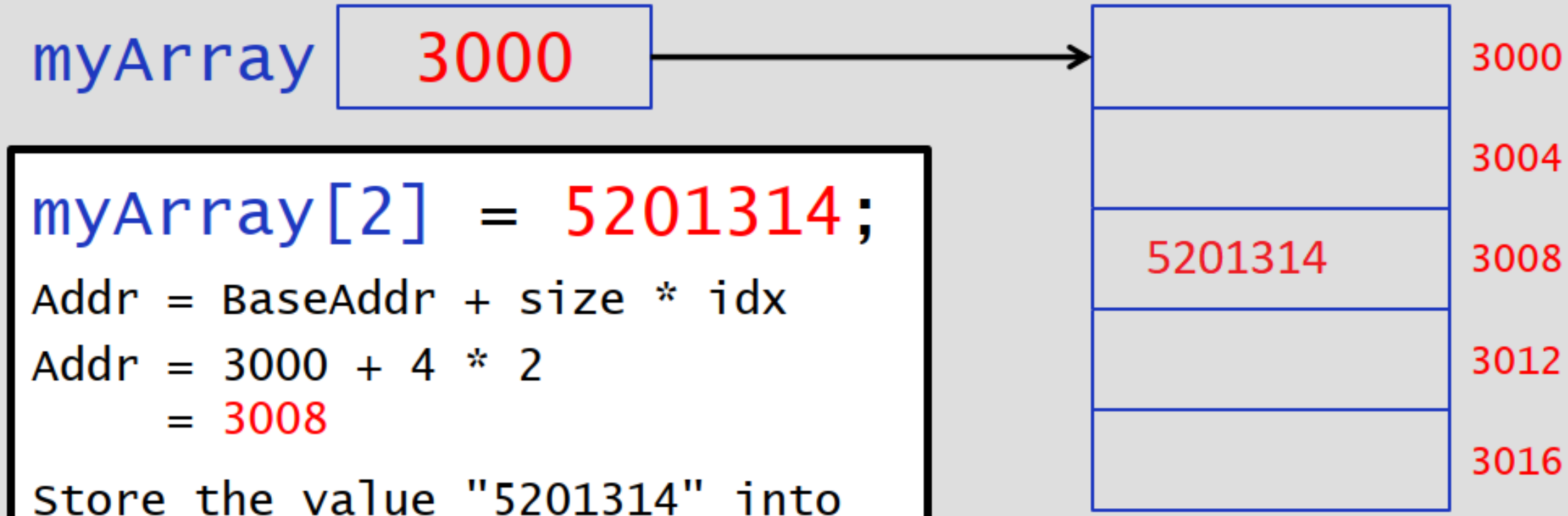
```
int[] myArray = new int[5];
```



1. A piece of memory enough for storing a memory address is reserved (the pointer to an array)
2. A chunk of consecutive memory, good enough for storing 5 integers, is reserved (the array)
3. The address of the array is stored in the pointer
4. From then on, whenever myArray is used, the program refers to the pointer

When Declaring an array

```
int[] myArray = new int[5];
```




```
myArray[2] = 5201314;
Addr = BaseAddr + size * idx
Addr = 3000 + 4 * 2
     = 3008

Store the value "5201314" into
the address, 3008.
```

Syntax for Arrays

- Arrays must be **declared** and **allocated** before use.
- Array Declaration: `<type>[] <array-name>;`

```
int[] intArray;
boolean[] boolArray, anotherBoolArray;
```

- Yet, array can't be used now because they have not been associated with any memory yet.
- Array allocation: `<array-name> = new <type>[<size>];`

```
intArray = new int[10];
boolArray = new boolean[4];
```

- Array Declaration + Allocation:

```
int[] intArray = new int[10];
boolean[] boolArray = new boolean[numOfPeople];
```

Syntax for Arrays

- Use an index to access an array element such as

```
int[] intArray = new int[10];
intArray[0] = 10;
intArray[1] = 5;
```

- An index must be between `0` and `array.length - 1` inclusively.
- Using index outside this range causes an `ArrayIndexOutOfBoundsException`.

```
int[] intArray = new int[10];
intArray[-1] = 10; //error!
intArray[10]; //error! maximum index is 9
```

Syntax for Arrays

- When an array is allocated, all numeric elements are automatically **initialized** with 0, or false for booleans.

```
int[] intArray = new int[3];
boolean[] boolArray = new boolean[1];
System.out.println(intArray[0]); //print 0
System.out.println(boolArray[0]); //print false
```

- You can also allocate an array with a different way

```
int[] myArray = {0,1,4,9,16,25,36};
int[] myArray2 = new int[7];
for (int i = 0; i < 7; i++)
    myArray2[i] = i * i;
//effectively they are the same
```

Ex1: Array Declaration

Error? No Error? That's the Question!

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

```
int[] intArray = new int[-1];
```

```
int[] intArray = new int[3.14];
```

```
int[] intArray = new float[10];
```

```
int[] intArray = new short[10];
```

 `int[] intArray = new int[0];` is weird but no error.

Syntax for Array

- Recall extended for loop (for-each loop), which works with arrays nicely.

```
int[] intArray = {1, 4, 7, 11, 14};
for (int i : intArray) {
    System.out.printf("%d * %d = %d  \n", i, i, i*i);
}
```

```
1 * 1 = 1
4 * 4 = 16
7 * 7 = 49
11 * 11 = 121
14 * 14 = 196
```

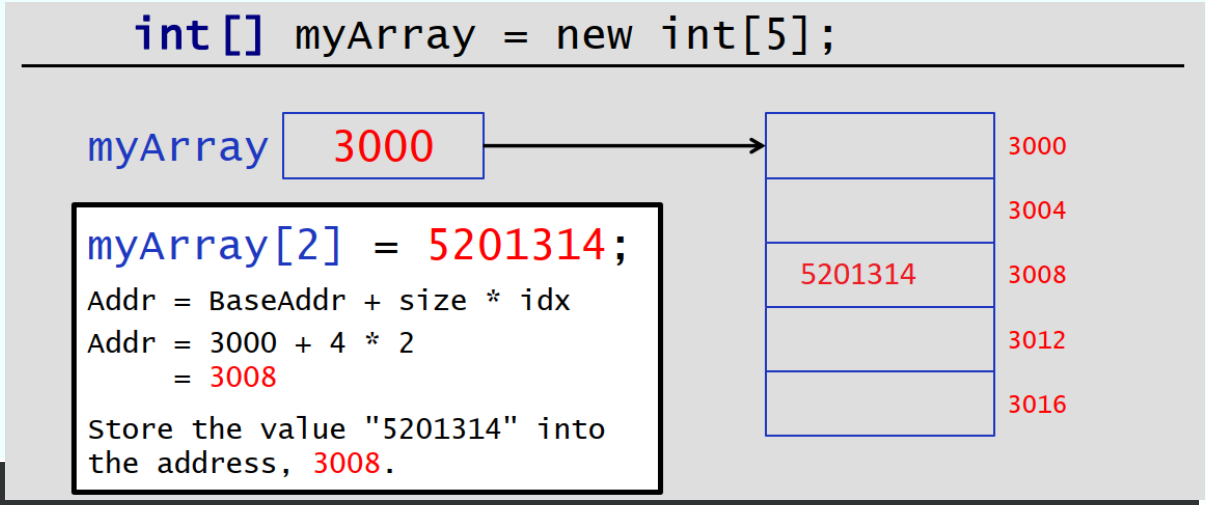
- However, updating value in the counter does not affect the value in the array

```
int[] intArray = {1, 4, 7, 11, 14};
for (int i : intArray)
    i = 1;
for (int i : intArray) {
    System.out.printf("%d * %d = %d  \n", i, i, i*i);
}
```

```
1 * 1 = 1
4 * 4 = 16
7 * 7 = 49
11 * 11 = 121
14 * 14 = 196
```

Reference

- `myArray` is a variable which contains a special type of value called ***address*** or ***reference***.
- An extended for simply copy the content from the array to the counter `i`




```
for (int i : myArray) { ... }
```

basically means

```
for (int counter = 0; counter < myArray.length; counter++) {
    int i = myArray[counter];
    ...
}
```


Reference and address are not exactly the same, treat them the same here for convenient.

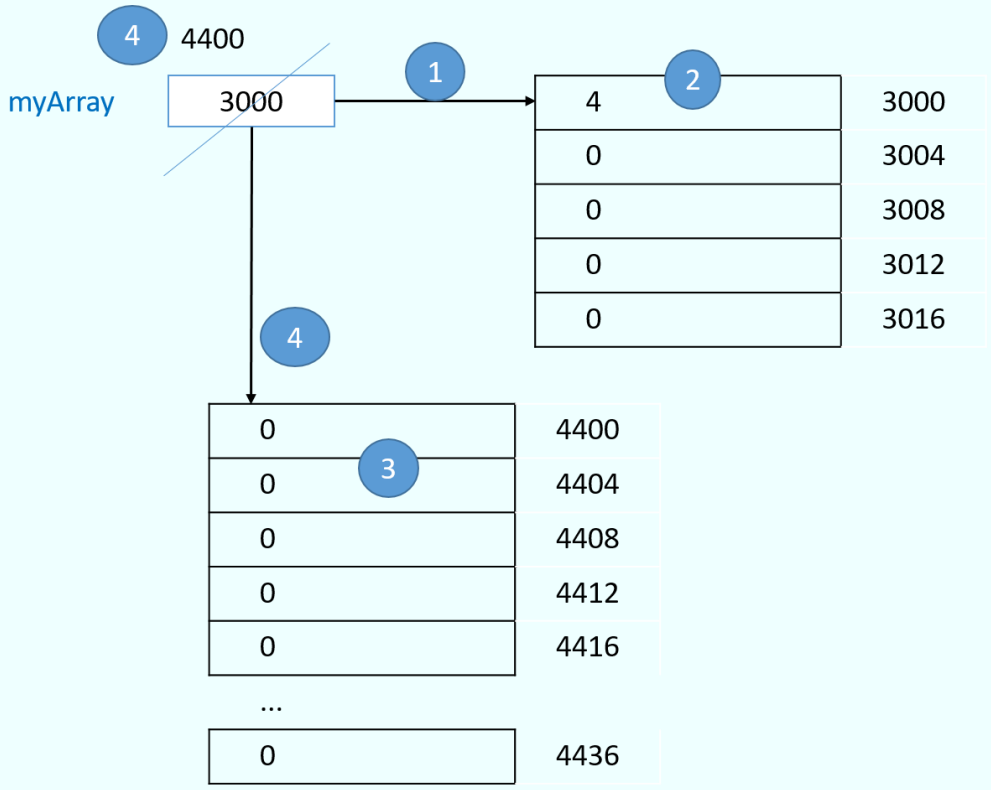
Reference - new array

What happen if...

```
int[] myArray = new int[5];
myArray[0] = 4;
myArray = new int[10];
System.out.println(myArray[0]); //print ??
```

Reference - new array

1. myArray was first allocated with a 5-elements memory block from the address 3000.
2. The value of `myArray[0]` is changed to 4.
3. A new block of 10-elements is allocated from the address 4400.
4. The value of myArray (which is a reference) is changed to where the new array is.



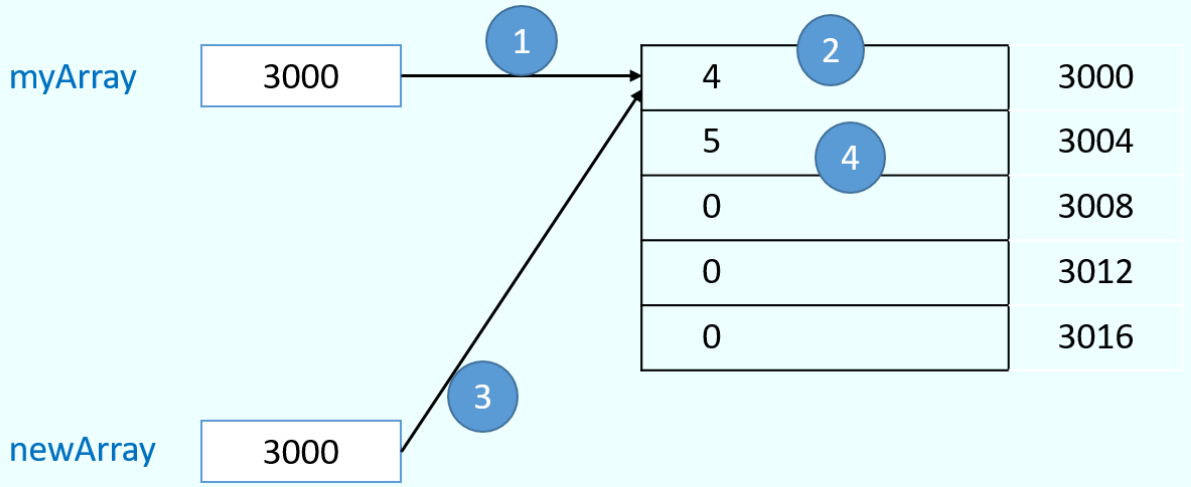
Reference - another variable

What happen if...

```
int[] myArray = new int[5];
int[] newArray = myArray;
myArray[0] = 4;
System.out.println(newArray[0]); //print ??
newArray[1] = 5;
System.out.println(myArray[1]); //print ??
```

Reference - another variable

1. `myArray` has been allocated with a 5-elements memory block.
2. The value of the first element of the memory block (`myArray[0]`) is updated to 4
3. `newArray` **points** to the same 5-elements memory block as `myArray` does.
4. The value of the first element of the memory block (`newArray[1]` / `myArray[0]`) is updated to 4



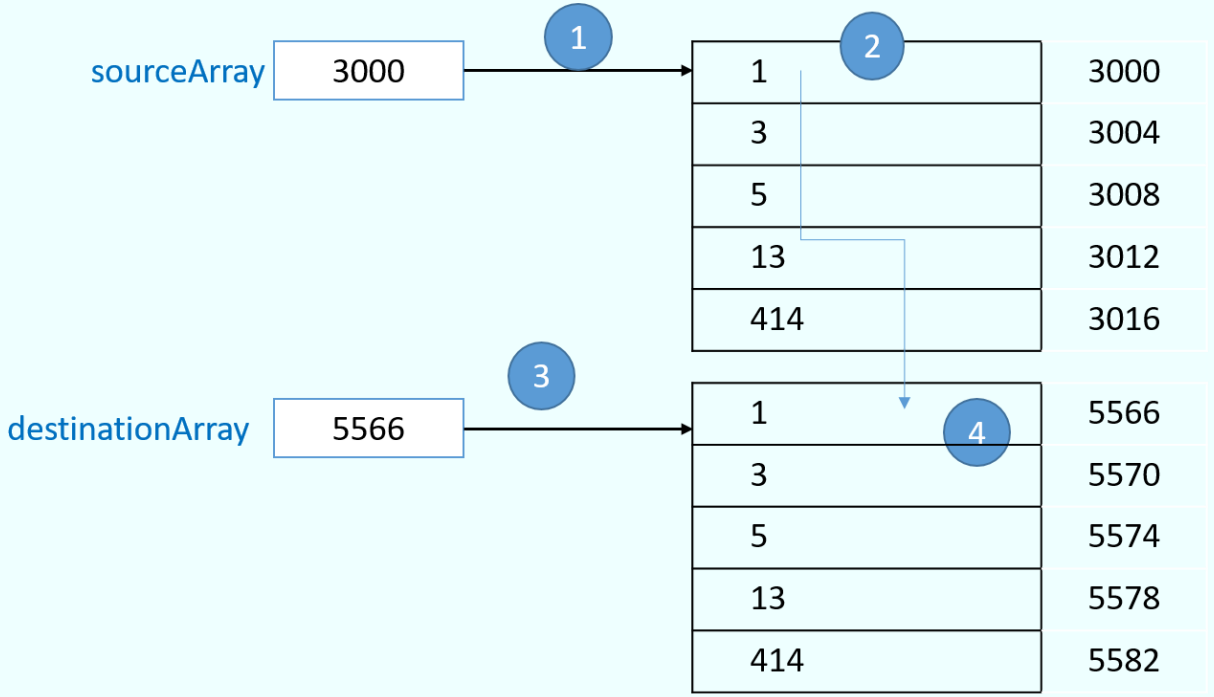
Common mistake: students wrongly copy an array by:

```
newArray = myArray;
```

Ex - Copying an array

A proper way to copy an array

```
int[] sourceArray = {1,3,5,13,414};
//declare and allocate an array with the same length of the source
int[] destinationArray = new int[sourceArray.length];
for (int i = 0; i < sourceArray.length; i++)
    destinationArray[i] = sourceArray[i];
```



Ex - Summing an array

```
int[] array = {15, 22, 34, 5, 41, 56, 9, 33};
int sum = 0;  //important

for (int i = 0; i < array.length; i++) {
    sum += array[i];
}
System.out.println("Sum: " + sum);  //print Sum: 215
```

or using extended for loop

```
int[] array = {15, 22, 34, 5, 41, 56, 9, 33};
int sum = 0;  //important

for (int i : array) {
    sum += i;
}
System.out.println("Sum: " + sum);  //print Sum: 215
```

Ex - Finding Max

```
int[] array = {15, 22, 34, 5, 41, 56, 9, 33};
int max = array[0];                    // why this line??
for (int i : array)
    if (max < i)
        max = i;
System.out.println("Max: " + max); //print Max: 56
```

or

```
int[] array = {15, 22, 34, 5, 41, 56, 9, 33};
int max = array[0];                    // why this line??
for (int i = 1; i < array.length; i++) {   //why i starts from 1?
    if (max < array[i]) {
        max = array[i];
    }
}
System.out.println("Max: " + max); //print Max: 56
```

Ex - Finding the best Student

```
int[] scores = {67, 50, 69, 83, 55, 94};
String[] names = {"Alice", "Bob", "Carol", "Dave", "Eva", "Fred"};
int maxIndex = 0;
for (int i = 1; i < scores.length; i++)
    if (scores[maxIndex] < scores[i])
        maxIndex = i;
System.out.println("Best student: " + names[maxIndex]);
System.out.println("Score: " + scores[maxIndex]);
```

Indices	Scores	Names	Remarks
0	67	Alice	initial value of `maxIndex` = 0
1	50	Bob	
2	69	Carol	`maxIndex` updates to 2 when `i` is 2
3	83	Dave	`maxIndex` updates to 3 when `i` is 3
4	55	Eva	
5	94	Fred	`maxIndex` updates to 5 when `i` is 5

Ex - Finding best students

- What if there are multiple students top the list at the same time?

```
int[] scores = {83, 50, 69, 83, 55, 83};
String[] names = {"Alice", "Bob", "Carol", "Dave", "Eva", "Fred"};
int max = scores[0];
for (int i : scores)
    if (max < i)
        max = i;

System.out.print("Best Student(s): ");
for (int i = 0; i < scores.length; i++)
    if (scores[i] == max)
        System.out.print(names[i] + " ");
System.out.println();
System.out.println("Score: " + max);
```

```
Best Student(s): Alice Dave Fred
Score: 83
```

Ex - Histogram

```
int[] array = {15, 22, 34, 5, 41, 56, 9, 33};
for (int i = 0; i < array.length; i++) {
    System.out.print(i + "  ");
    for (int j = 0; j < array[i]; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

```
0  ***************
1  **********************
2  ********************************
3  *****
4  ****************************************
5  ****************************************************************
6  *********
7  ********************************
```

Ex - Voting

Given the following array, print a histogram

```
String[] ballots = {"Ada", "Bob", "Ada", "Eva", "Ada", "Bob", "Eva", "Bob", "Ada",...};
```

```
Ada   ***************
Bob   ********************
Eva   *******
Tom   *****
Nic   **********
Tim   **********
```

Yet, we don't know the name of the candidates.

Fun fact: US presidential election allow a voter to write in a name that is not on the election ballot. Search *write-in candidate*.

Ex - Voting - Build candidate list

Two stages of design:

1. Building the candidate
2. Counting the vote and print

```
String[] ballots = {"Ada", "Bob", "Ada", "Eva",  ...}; //list too long
String[] candidates = new String[ballots.length];
int numOfCand = 0; //keep track the number of candidates
for (int i = 0; i < ballots.length; i++) {
    //check if candidate is already in the candidates list
    if (....) {
        //skip
    } else {
        //add the candidate into the list
        candidates[numOfCand] = ballots[i];
        numOfCand++;
    }
}
for (int i = 0; i < numOfCand; i++)
    System.out.println("Candidate " + i + " : " + candidates[i]);
```

Ex - Voting - Build candidate list

```
String[] ballots = {"Ada", "Bob", "Ada", "Eva",  ...}; //list too long
String[] candidates = new String[ballots.length];
int numOfCand = 0; //keep track the number of candidates
for (int i = 0; i < ballots.length; i++) {
    //check if candidate is already in the candidates list
    boolean onList = false;
    for (int j = 0; j < numOfCand; j++)
        if (ballots[i].equals(candidates[j]))
            onList = true;
    if (onList) {
        continue;   //we can invert the logic to save keep it clean
    } else {
        //add the candidate into the list
        candidates[numOfCand] = ballots[i];
        numOfCand++;
    }
}
for (int i = 0; i < numOfCand; i++)
    System.out.println("Candidate " + i + " : " + candidates[i]);
```

Ex - Voting - Build candidate list

```
String[] ballots = {"Ada", "Bob", "Ada", "Eva",  ...}; //list too long
String[] candidates = new String[ballots.length];
int numOfCand = 0; //keep track the number of candidates

for (int i = 0; i < ballots.length; i++) {
    //check if candidate is already in the candidates list
    boolean onList = false;
    for (int j = 0; j < numOfCand; j++)
        if (ballots[i].equals(candidates[j]))
            onList = true;

    if (!onList) {
        //add the candidate into the list
        candidates[numOfCand] = ballots[i];
        numOfCand++;
    }
}
for (int i = 0; i < numOfCand; i++)
    System.out.println("Candidate " + i + " : " + candidates[i]);
// Count votes...
```

Ex - Voting - Build candidate list

```
Candidate 0 : Ada
Candidate 1 : Bob
Candidate 2 : Eva
Candidate 3 : Tom
Candidate 4 : Nic
Candidate 5 : Tim
```

- Count the vote would be very similar.
- Need another `int[]` array to keep track of the votes

```
int[] votes = new int[numOfCand];
for (String v : ballots)
    for (int index = 0; index < numOfCand; index++)
        if (v.equals(candidates[index])) {
            votes[index]++;
            break;
        }
```

Ex - Voting - Combining two steps

```
String[] ballots = {"Ada", "Bob", "Ada", "Eva",  ...}; //list too long
String[] candidates = new String[ballots.length];
int[] votes = new int[ballots.length];
int numOfCand = 0; //keep track the number of candidat
for (int i = 0; i < ballots.length; i++) {
    //check if candidate is already in the candidates list
    boolean onList = false;
    for (int j = 0; j < numOfCand; j++)
        if (ballots[i].equals(candidates[j])) {
            onList = true;
            votes[j]++;
        }
    if (!onList) {
        //add the candidate into the list
        candidates[numOfCand] = ballots[i];
        votes[numOfCand] = 1;
        numOfCand++;
    }
}
//con't
```

Ex - Voting - Combining two steps

```
for (int i = 0; i < numOfCand; i++) {
    System.out.print(candidates[i] + ":");
    for (int j = 0; j < votes[i]; j++)
        System.out.print("*");
    System.out.println();
}
```

Ex - Enlarging Arrays

- Once memory spaces are allocated for an array, the size can't be changed

```
array.length = array.length + 10; //error! read-only data
```

- To enlarge an array dynamically we should **declare a new array and copy the content**

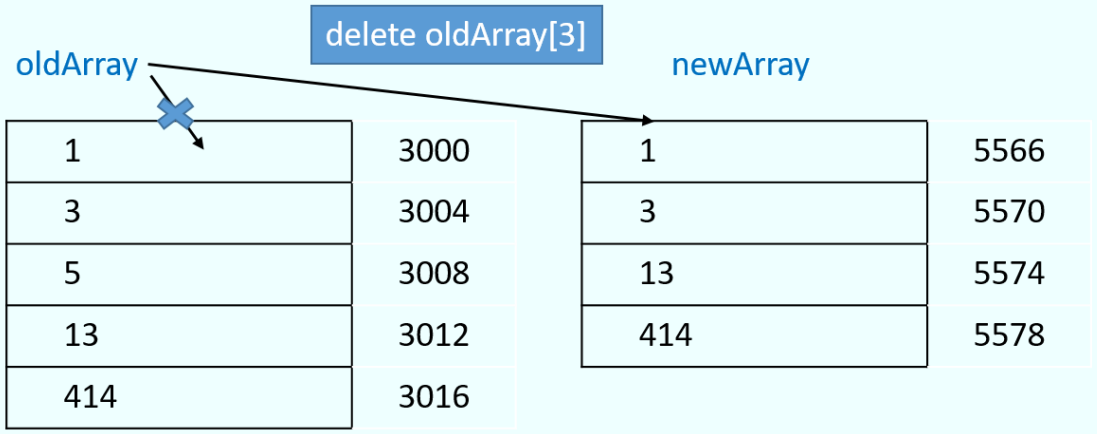
```
String[] oldArray = {"apple", "banana", "carrot"};
//wish to add one more element into oldArray
String[] newArray = new String[oldArray.length + 1];
for (int i = 0; i < oldArray.length; i++)
    newArray[i] = oldArray[i];
newArray[newArray.length - 1] = "new data";

oldArray = newArray; //replace the existing reference by the new reference
```

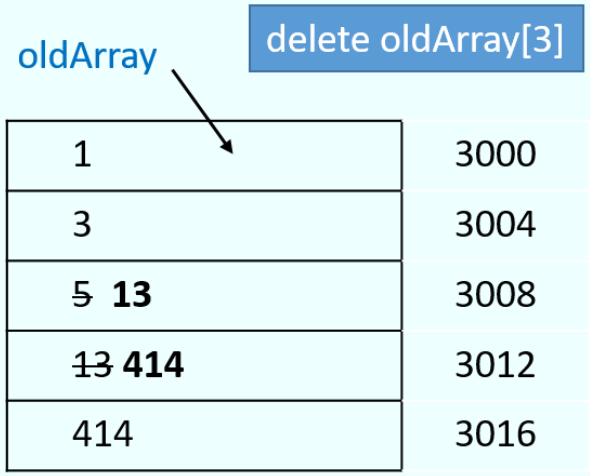
Ex - Shrinking Arrays

If you wish to remove an element from an array, there are two options

1. Recreate a smaller size array and copy all content except the deleted one; or



2. Move the content of the array like *music-chairs*.



You need another variable to tell the *effective* size of the array.

Ex - Shrinking Arrays - 1

```
String[] oldArray = {"apple", "banana", "carrot", "mango", "lemon"};
//wish to remove third item - oldArray[2]
String[] newArray = new String[oldArray.length - 1];
int indexToRemove = 2;

if (indexToRemove < oldArray.length) {
    for (int i = 0; i < indexToRemove; i++)
        newArray[i] = oldArray[i];
    for (int i = indexToRemove; i < newArray.length; i++ )
        newArray[i] = oldArray[i + 1];
    //replace the existing reference by the new reference
    oldArray = newArray;
}
for (String s : oldArray)
    System.out.println(s);
System.out.println("The lenth of oldArray is: " + oldArray.length);
System.out.println("The lenth of newArray is: " + newArray.length);
```

Ex - Shrinking Arrays - 2

```
String[] oldArray = {"apple", "banana", "carrot", "mango", "lemon"};
int size = oldArray.length;

//wish to remove third item - oldArray[2]
int indexToRemove = 2;

if (size > 0) {
  for (int i = indexToRemove; i < size - 1; i++ )
    oldArray[i] = oldArray[i + 1];
  size--;
}
for (int i = 0; i < size; i++)
  System.out.println(oldArray[i]);

System.out.println("The length of oldArray is: " + oldArray.length);
System.out.println("The value of size is: " + size);
```

Common Mistakes

```
int[] array;
array[0] = 10;
```

❌ Array has not been allocated yet

```
int array[10];
array[0] = 10;
```

❌ C-style array declaration. Invalid in Java.

```
int[] array = new int[];
array[0] = 10;
```

❌ Allocate array without a size

Common Mistakes

```
int[] array = new int[10];
array.length++;
```

❌ `array.length` is read-only

```
int[] array = new char[10];
```

❌ incompatible type int vs char.

```
int[] array = new int[10];
for (int i = 1; i <= 10; i++)
    array[i] = i;
```

❌ arrays always start with 0 end with length - 1.

Common Mistakes

```
int[] array = new int[10];
array[] = {5, 5, 3};
```

❌ `array[]` is an incorrect syntax.

```
int[] array = new int[10];
for (int i = 0; i < 10; i++)
    array = i;
```

❌ `array` is an array, not to store just one integer. You may want to type `array[i]` instead.

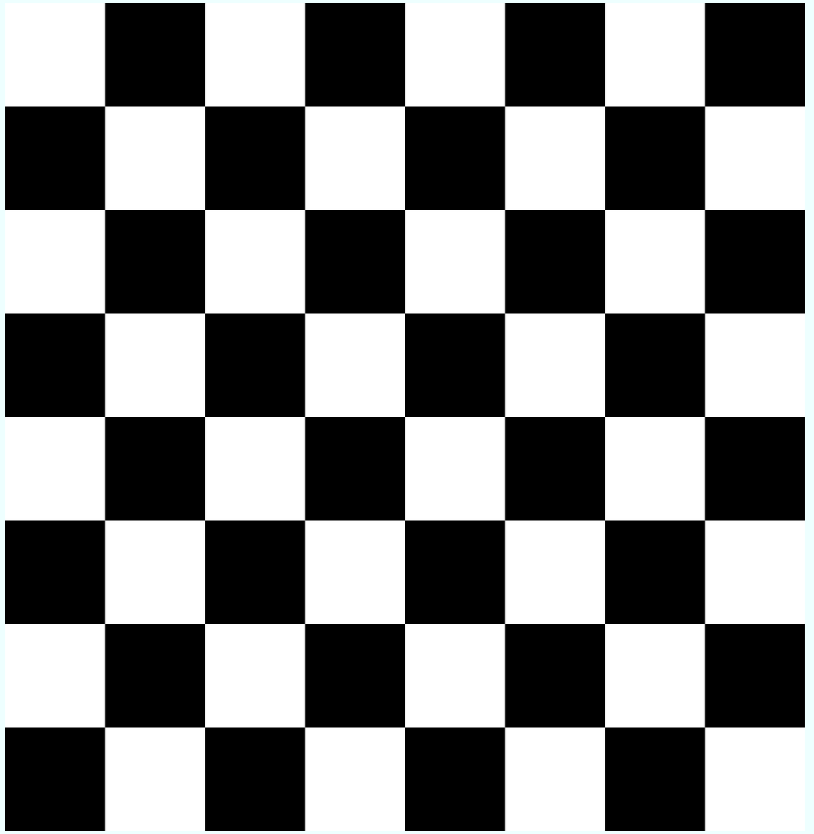
2D Arrays

By Kevin 2022/2023

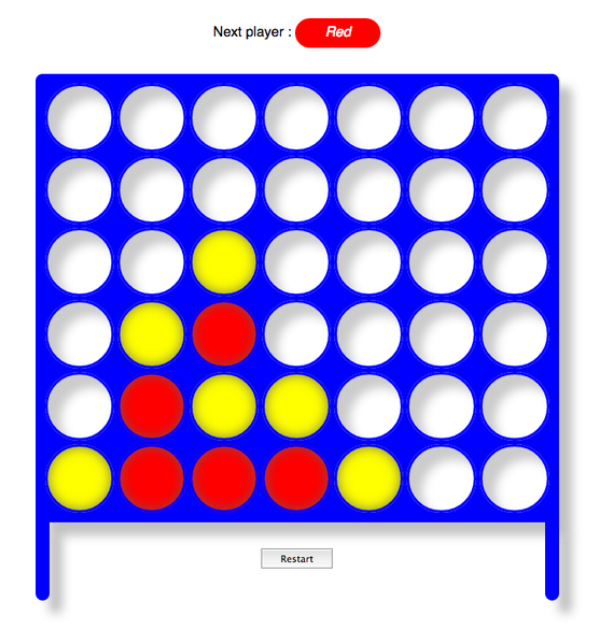
2D Arrays

What we have discussed so far is Linear Array

- But there are applications where linear isn't just enough
- Suppose we wanted to model a chess board
  - not a linear group of squares
  - more like a grid of squares
  - linear array would not be appropriate



2D Arrays - other applications



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

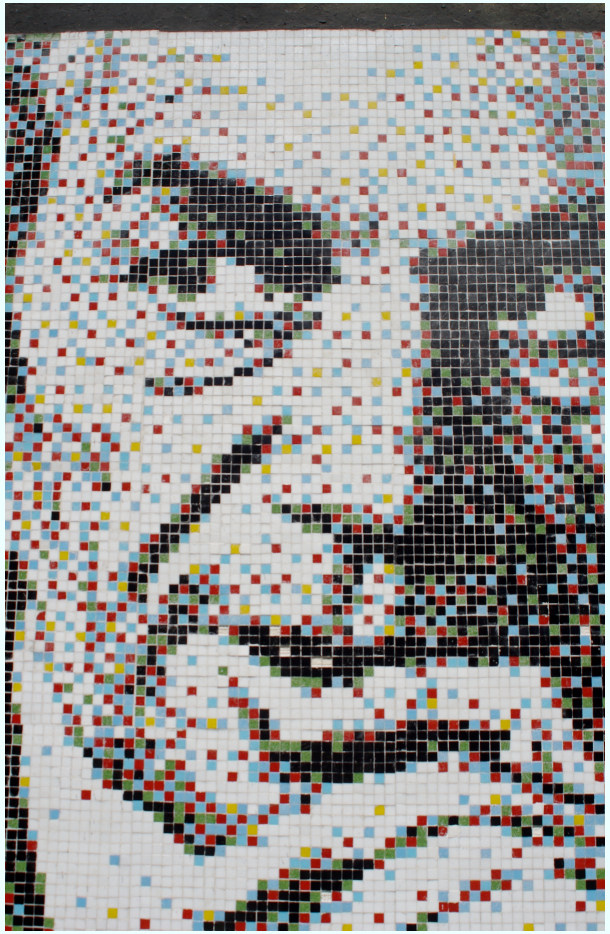
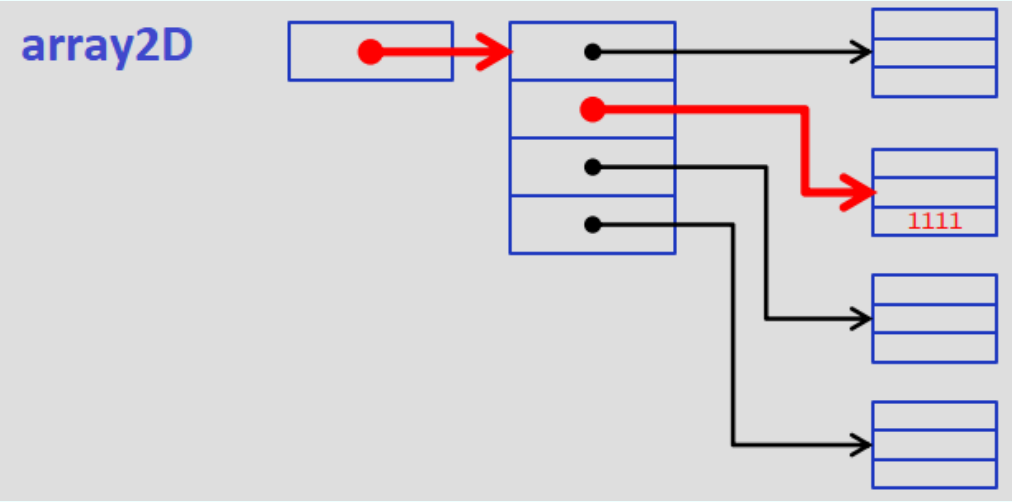


image credit: 1, 2 3

2D Array - Declaration and Allocation

We declare an 2D array by

```
int[][] array2D = new int[4][];
array2D[0] = new int[3];
array2D[1] = new int[3];
array2D[2] = new int[3];
array2D[3] = new int[3];
array2D[1][2] = 1111;
```

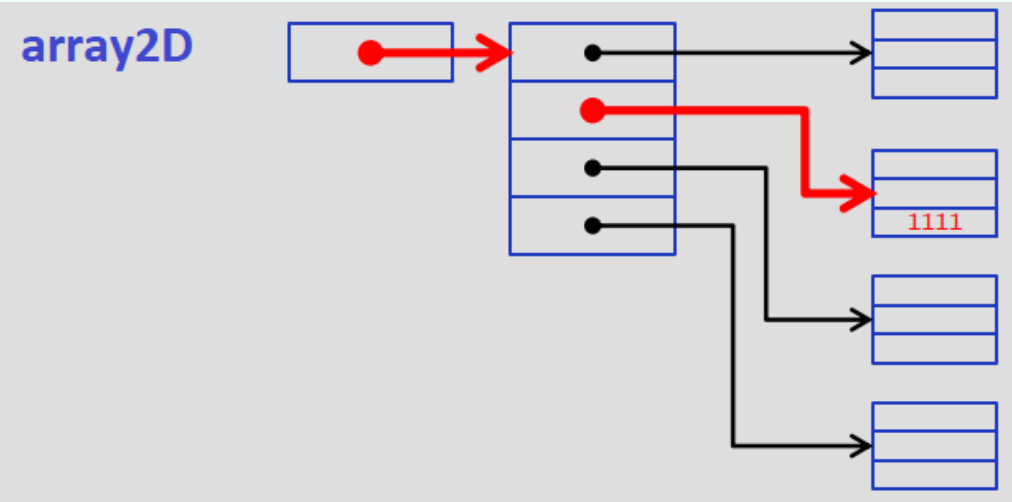


- `array2D` - It is an array of type `int[]`
  - It has four elements. Each element is an int array.
- `array2D[0]` - It is an array of type `int`
  - It has three elements. Each element is an int.
- `array2D[0][0]` - It is an element of type `int`

2D Array - Declaration and Allocation

We declare an 2D array by

```
int[][] array2D = new int[4][3];
array2D[1][2] = 1111;
```



- `array2D` - It is an array of type `int[]`
  - It has four elements. Each element is an int array.
- `array2D[0]` - It is an array of type `int`
  - It has three elements. Each element is an int.
- `array2D[0][0]` - It is an element of type `int`

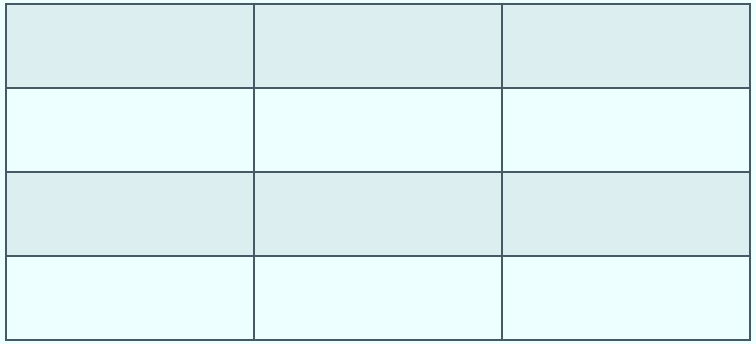
2D Array - length

```
int[][] array2D = new int[4][3];
array2D[1][2] = 1111;
```

- `array2D.length` is 4. Because it has four elements in this array.
- `array2D[2].length` is 3. Because it has three elements in this array.
- `array2D[0][1].length` is undefined because it is not an array!

Conceptually it has almost no different if you draw the array is a 3-by-4 rectangle or 4-by-3 rectangle on a paper. By convention, we treat the first bracket as row and the second bracket as column.

array2D



2D Array

```
int [][] array = new int [5][4];
System.out.println("array: " + array.length + " rows");
for (int r = 0; r < array.length; r++)
    for (int c = 0; c < array[r].length; c++) {
        if ((r+c) % 2 == 0) {
            array[r][c] = 1;
        } else {
            array[r][c] = 0;
        }
    }
for (int r = 0; r < array.length; r++) {
    System.out.print(r + " (" + array[r].length + " columns): ");
    for (int c = 0; c < array[r].length; c++) {
        System.out.print(array[r][c] + " ");
    }
    System.out.println();
}
```

	0	1	2	3
0	1	0	1	0
1	0	1	0	1
2	1	0	1	0
3	0	1	0	1
4	1	0	1	0

```
array: 5 rows
0 (4 columns): 1 0 1 0
1 (4 columns): 0 1 0 1
2 (4 columns): 1 0 1 0
3 (4 columns): 0 1 0 1
4 (4 columns): 1 0 1 0
```

2D Array Alternative Allocation

- Similar to 1D array, you can allocate and initialize a 2D array using bracelet directly

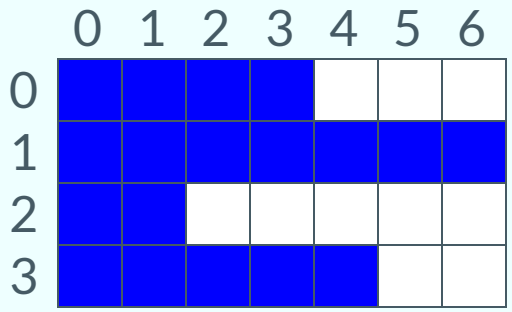
```
int[][] array = {{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
```

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12

Irregular 2D Array

With crazy reasons you may even do the following!

```
int[][] twoDArray = new int[4][];
twoDArray[0] = new int[4];
twoDArray[1] = new int[7];
twoDArray[2] = new int[2];
twoDArray[3] = new int[5];
```



Same as the following

```
int[][] twoDArray = {{0, 0 ,0 ,0}, {0, 0 ,0, 0, 0, 0, 0}, {0, 0}, {0, 0, 0, 0, 0}};
//twoDArray.length = 4
//twoDArray[3].length = 5
```

Multi-Dimensional Arrays

- Can declare an array to be 2-dimensional

```
int[][] twoDArray = new int[5][4];
twoDArray[2][3] = 9;
```

- Can declare an array to be 5-dimensional

```
int[][][][][] fiveDArray = new int[7][3][19][32][3];
fiveDArray[3][0][15][3][0] = 7428;
```

- In fact, you can create array of any dimension!

Multi-Dimensional Arrays

```
int[][]...[] mDArray = new int[a1][a2][a3]...[aM];
////|-- m --|                        |------ m -------|
```

```
int[][][][][] array = new int[7][3][19][32][3];
int cnt = 0;
for (int d1 = 0; d1 < array.length; d1++) {
   for (int d2 = 0; d2 < array[d1].length; d2++) {
      for (int d3 = 0; d3 < array[d1][d2].length; d3++) {
         for (int d4 = 0; d4 < array[d1][d2][d3].length; d4++) {
            for (int d5 = 0; d5 < array[d1][d2][d3][d4].length; d5++) {
               array[d1][d2][d3][d4][d5] = d1 * d2 * d3 * d4 * d5;
               cnt++;
            }
         }
      }
   }
}
System.out.println("cnt: " + cnt);   //print cnt:38304
```

Tic-tac-toe

```
Player 1, your turn! 0 1
_ x _
_ _ _
_ _ _
Player 2, your turn! 0 1
invalid!
Player 2, your turn! 2 1
_ x _
_ _ _
_ o _
Player 1, your turn! 0 2
_ x x
_ _ _
_ o _
...
x x x

_ _ _
o o _
Player 1 you win!
```

Tic-tac-toe

Ingredients

- Arrays
- Loops
- If

We can try a 1D and a 2D version

Strategy

- Create a no-loop version

Tic-tac-toe 1D version

```
0 1 2
3 4 5
6 7 8
```

```
char[] cells = new char[9];
for (int i = 0; i < cells.length; i++)
    cells[i] = '_';
//input
...
//print cells
for (int i = 0; i < 3; i++)
    System.out.print(cells[i] + " ");
System.out.println();
for (int i = 3; i < 6; i++)
    System.out.print(cells[i] + " ");
System.out.println();
for (int i = 6; i < 9; i++)
    System.out.print(cells[i] + " ");
System.out.println();
...

//check wins
```

Tic-tac-toe v1D - simplified print cells

```
Scanner scanner = new Scanner(System.in);
char[] cells = new char[9];
for (int i = 0; i < cells.length; i++)
    cells[i] = '_';
//input
...
//print cells
for (int j = 0; j < 3; j++) {
    for (int i = 0; i < 3; i++)
        System.out.print(cells[j * 3 + i] + " ");
    System.out.println();
}
...
```

```
0 1 2
3 4 5
6 7 8
```

i	j	j * 3 + i
1	1	4
0	2	6
2	1	5

Tic-tac-toe v1D - adding input

```
Scanner scanner = new Scanner(System.in);
char[] cells = new char[9];
for (int i = 0; i < cells.length; i++)
    cells[i] = '_';
//input
System.out.print("Player 1, your turn! ");
int row = scanner.nextInt();
int col = scanner.nextInt();
//validate inputs
...
//print cells
for (int j = 0; j < 3; j++) {
    for (int i = 0; i < 3; i++)
        System.out.print(cells[j * 3 + i] + " ");
    System.out.println();
}
//check wins
...
```

```
0 1 2
3 4 5
6 7 8
```

Tic-tac-toe v1D - validate inputs

```
...
//input
System.out.print("Player 1, your turn! ");
int row = scanner.nextInt();
int col = scanner.nextInt();
//validates input
boolean valid = true;
if (row < 0 || row > 2 || col < 0 || col > 2)
    valid = false;
if (!valid || cells[row * 3 + col] != '_')
    valid = false;
// if invalid, input again
//print cells
...
//check wins
...
```

Tic-tac-toe v1D - check horizontal

```
0 1 2
3 4 5
6 7 8
```

```
...
//print cells
...
//input
int row, col;
row = col = 0;
do {
    System.out.print("Player 1, your turn! ");
    row = scanner.nextInt();
    col = scanner.nextInt();
    if (row >= 0 && row <= 2 && col >= 0 && col <= 2 && cells[row * 3 + col] == '_')
        break;
} while (true);
cells[row * 3 + col] = 'x'; //for player 1
//check wins
boolean win = false;
//check horizontal
if (cells[0] == cells[1] && cells[0] == cells[2] && cells[0] == 'x')
    win = true;
if (cells[3] == cells[4] && cells[3] == cells[5] && cells[3] == 'x')
    win = true;
if (cells[6] == cells[7] && cells[6] == cells[8] && cells[6] == 'x')
    win = true;
//check vertical
//check diagonal
...
```

Tic-tac-toe v1D - check wins

```
//check wins
boolean win = false;
//check horizontal
for (int r = 0; r < 3; r++)
    if (cells[r * 3] == cells[r * 3 + 1] &&
        cells[r * 3] == cells[r * 3 + 2] && cells[r * 3] == 'x')
        win = true;
//check vertical
for (int c = 0; c < 3; c++)
    if (cells[c] == cells[c + 3] &&
        cells[c] == cells[c + 6] && cells[c] == 'x')
        win = true;
//check diagonal
if (cells[0] == cells[4] && cells[0] == cells[8] && cells[0] == 'x')
    win = true;
if (cells[3] == cells[4] && cells[3] == cells[6] && cells[3] == 'x')
    win = true;
if (win)
    System.out.println("Player 1 you win!");
...
```

Tic-tac-toe v1D - adding loops and player

```
boolean win = false;
boolean p1Turn = true;
do {
    char symbol = p1Turn ? 'x' : 'o';
    ...
    //validate
    ...
    if (cells[0] == cells[4] && cells[0] == cells[8] && cells[0] == symbol)
        win = true;
    ...
    if (win)
        System.out.println("Player " + (p1Turn ? 1 : 2) + " you win!");
    //change player
    p1Turn = !p1Turn;
} while (!win);
```

Tic-tac-toe v1D - final

```
boolean win = false;
boolean p1Turn = true;
Scanner scanner = new Scanner(System.in);
char[] cells = new char[9];
for (int i = 0; i < cells.length; i++)
    cells[i] = '_';

do {
    char symbol = p1Turn ? 'x' : 'o';
    String player = p1Turn ? "Player 1" : "Player 2";
    int row = 0 , col = 0;

    do {
        System.out.print(player + ", your turn! ");
        row = scanner.nextInt();
        col = scanner.nextInt();
        if (row >= 0 && row <= 2 && col >= 0 &&
            col <= 2 && cells[row * 3 + col] == '_')
            break;
        System.out.println("Invalid");
    } while (true);
    cells[row * 3 + col] = symbol;
    //print cells
    for (int j = 0; j < 3; j++) {
        for (int i = 0; i < 3; i++)
            System.out.print(cells[j * 3 + i] + " ");
        System.out.println();
    }
```

```
    //check horizontal
    for (int r = 0; r < 3; r++)
        if (cells[r * 3] == cells[r * 3 + 1] &&
            cells[r * 3] == cells[r * 3 + 2] &&
            cells[r * 3] == symbol)
            win = true;
    //check vertical
    for (int c = 0; c < 3; c++)
        if (cells[c] == cells[c + 3] &&
            cells[c] == cells[c + 6] &&
            cells[c] == symbol)
            win = true;
    //check diagonal
    if (cells[0] == cells[4] &&
        cells[0] == cells[8] &&
        cells[0] == symbol)
        win = true;
    if (cells[2] == cells[4] &&
        cells[2] == cells[6] &&
        cells[2] == symbol)
        win = true;

    if (win)
        System.out.println(player + " you win!");
    p1Turn = !p1Turn;
} while (!win);
```

Tic-Tac-Toe 2D version

- 2D version works very similar to the 1D version
- Set the cell be a 3x3 arrays

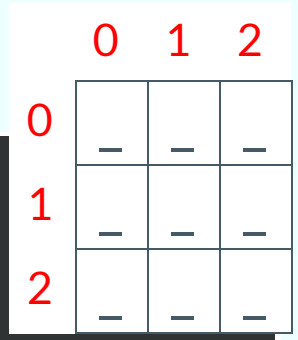
```
char[][] cells = new char[3][3];
for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)
        cells[i][j] = '_';
```

- Alternatively, you can also allocate a 2D array by

```
char[][] cells = {{'_','_','_'},
                  {'_','_','_'},
                  {'_','_','_'}};
```

Tic-Tac-Toe v2D - checking

- To check a horizontal line




```
boolean win = false;
for (int r = 0; r < 3; r++) {
    boolean formLine = true; //assume that row forms a line
    for (int c = 0; c < 3; c++)
        if (cells[r][c] != symbol)
            formLine = false; //oh it is not
    if (formLine) //only happen when all cells == symbol
        win = true;
}
//if win == true, the cells form at least one horizontal line.
//check vertical
//check diagonals
```

Revisit Crossing the Bridge Game

Ref: https://www.inwebson.com/demo/cross-the-bridge/

- 6 family members need to cross a bridge within 30 seconds.
- Need to bring a lamp with them (someone need to take the lamp back)
- Times required to cross the bridge for different members are different:
- Max two people can cross a bridge at the same time
- These two people will walk at the same pace.

People	Time Require To Cross Bridge
Alex	1 sec
Bob	2 sec
Carol	4 sec
Dave	6 sec
Eva	8 sec
Fred	12 sec

Revisit Crossing the Bridge Game

```
Time: 0
ABCDEF (*) _______________
Enter two initials or one followed by -: A B
Time: 2
CDEF _____________ AB (*)
Enter two initials or one followed by -: D E
Invalid selection!
Enter two initials or one followed by -: D A
Invalid selection!
Enter two initials or one followed by -: A -
Time: 3
ACDEF (*) _____________ B
Enter two initials or one followed by -: C F
Time: 15
ADE _____________ BCF (*)
...
```

Revisit Crossing the Bridge Game

Strategy:

- Store the initials of each person into an array
- Find the indices of the two people crossing the bridge
- Store the position of each person into an array
- Store the times required for each person into an array

Do it as an exercise

Summary

- Array Syntax
- Array Applications
- 2D-Array

Updates on Errata

On 5/10/2021

- Page 47 - correct the diagram to 5 rows
- Page 49 - correct the number of zeros
- Page 59 - correct the vertical validation
- Page 62 - correct the type `new cell[3][3]` -> `new char[3][3]`