

## Introduction to Wrapper class:



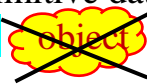
### Let us first understand why wrapper class?

Java is an impure object-oriented programming language because it supports **primitive data type**.

You may be wondering; does it really matter if java is pure or impure object oriented.



To understand this, you must know an important fact about primitive data types which is **Primitive data types are not treated as objects in java.**



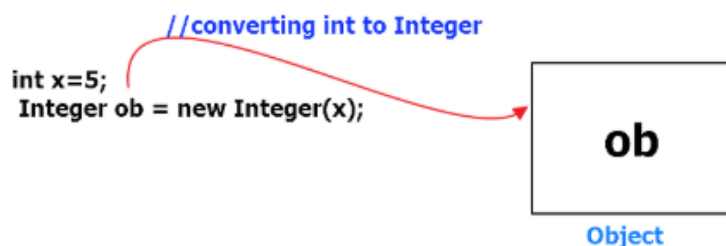
However, 100% pure object-oriented programs can be written in java using **Wrapper Class**.

*Now you understood why wrapper class let us understand what is Wrapper class.*

## What is Wrapper class?

The **wrapper class in Java** is a class provides the mechanism *to convert primitive data types into object and object into primitive data type*.

Well how if you wonder, consider the below example:



In the above example **int** is a **primitive data** type and **Integer** is a **wrapper class**. Let us have a look at one more example of character data type.

Character

if you are using a single character value, you will use the primitive char type

```
char ch = 'a';
```

- ✓ There are times, however, when you need to use a char as an object—for example, as a method argument where an object is expected.
- ✓ The Java programming language provides a wrapper class that "wraps" the char in a Character object for this purpose. An object of type Character contains a single field, whose type is char. This Character class also offers a number of useful class (i.e., static) methods for manipulating characters.
- ✓ You can create a Character object with the Character constructor:

```
Character characterObj= new Character('a');
```

**Advantage of Wrapper class:** The program becomes pure object oriented.

**Disadvantage of wrapper class:** Execution speed decreases.

**Advantage of Primitive data type:** Faster in execution.

**Disadvantage of Primitive data type:** The program becomes impure object oriented.

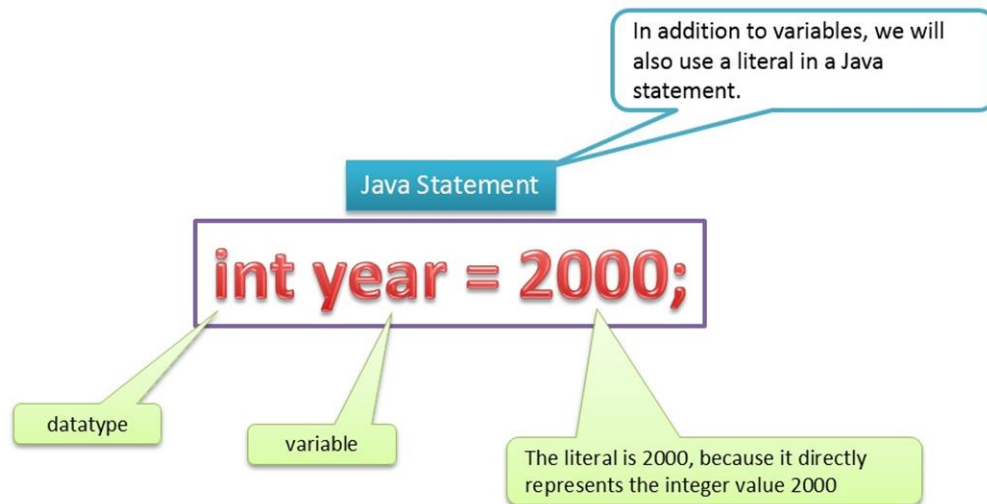
Different wrapper classes for different primitive data types is given below:

**Wrapper Classes for Primitive Data Types**

Primitive Data Types	Wrapper Classes
int	Integer
short	Short
long	Long
byte	Byte
float	Float
double	Double
char	Character
boolean	Boolean

## Literals in java:

Any constant value which can be assigned to the variable is called as literal. Consider the example shown below:



*Let us have a look at some valid and invalid syntaxes of variable names and literals.*

### **Variable Names:**

*The symbols that are allowed in variable names are → **\_** and **\$***

**Valid Syntax:**

```
int rooman_ = 99;  
int _rooman_ = 99;  
int _r_ooman_ = 99;  
int roman$ = 99;  
int $rooman_ = 99;  
int r_o$$oman_ = 99;
```

**Invalid Syntax:**

```
int &rooman_ = 99;  
int _roo%man$ = 99;  
int roo man = 99;  
int roo " man = 99;
```


## Literals:

The only symbol that is allowed in literal is  $\Rightarrow$  **"\_"(underscore)**

**Valid Syntax:**    `int rooman_ = 9_9;`  
                      `int _rooman_ = 9__9;`  
                      `int _r_ooman_ = 9____9;`

**Invalid Syntax:**    `int rooman_ = 99_;`  
                      `int rooman = _99;`  
                      `float rooman = 99._9f;`  
                      `float rooman = 99.9_9f;`  
                      `float rooman = 99.99_f;`

## Introduction to Arrays:

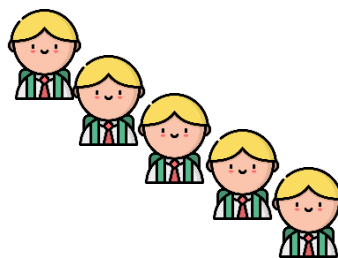
*In Java array is an **object** which contains elements of a similar data type.* 

After getting to know the definition of array, let us now understand why array-based approach was introduced.

To understand this, let us first understand the limitations of variable approach by considering three different cases:

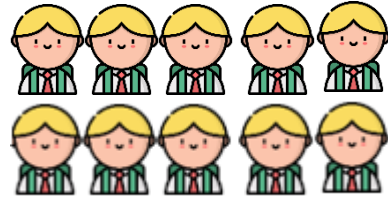
Case i) Write code to store the ages of 5 students.

```
int a;  
int b;  
int c;  
int d;  
int e;
```



Caseii) Write code to store the ages of 10 students.

```
int a;  
int b;  
int c;  
int d;  
int e;  
int f;  
int g;  
int h;  
int i;  
int j;
```



Caseiii) Write code to store the ages of 20 students.

```
int a;  
int b;  
int c;  
int d;  
int e;  
int f;  
int g;  
int h;  
int i;  
int j;  
int h;  
int i;  
int j;  
int k;  
int l;  
int m;  
int n;  
int o;  
int p;  
int q;  
int r;  
int s;  
int t;
```



After considering three different cases, if I consider case iii and ask you which variable stores the marks of 18<sup>th</sup> student, you'll now have to check each line to answer this question which is one of the limitations of variable approach.

The variable approach has two limitations:

- 1) *Creation is difficult.*
- 2) *Remembering multiple names and accessing them is difficult.*

Due to these limitations, array-based approach was invented.

Arrays are of two types:

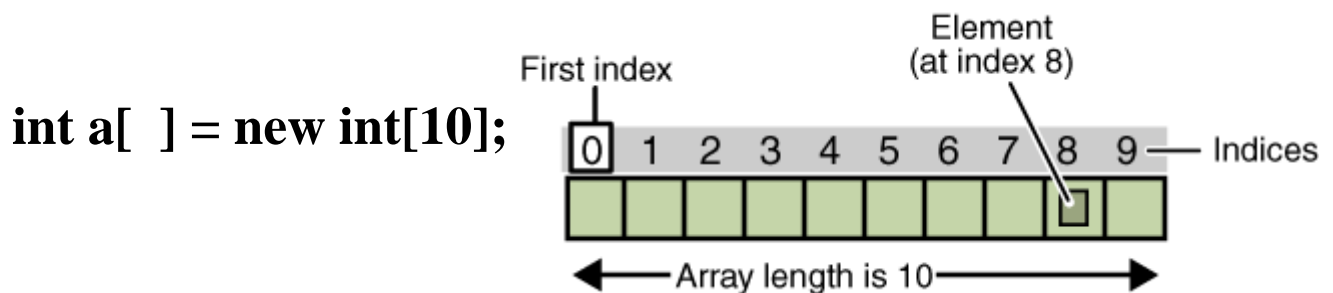
- 1) **Regular array.**
- 2) **Jagged array.**

1) **Regular array:** Regular array is further divided into 1-Dimensional, 2-Dimensional, 3-Dimensional array and so on.

### 1-Dimensional regular array/Rectangular array.

Let us learn the creation of 1-D array using an example.

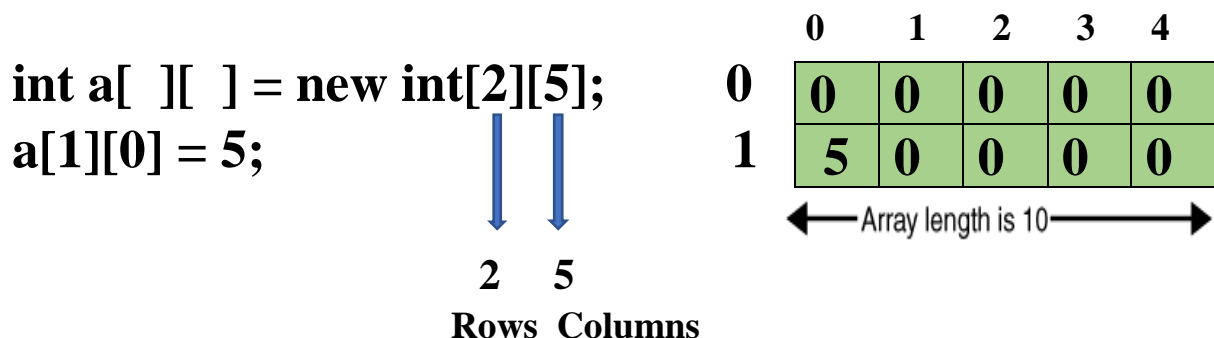
**Ex: Create an array to store the ages of 10 students:**



### 2-Dimensional regular array/Rectangular array.

Let us learn the creation of 2-D array using an example.

**Ex: Create an array to store the ages of students belonging to 2 classrooms with 5 students each.**



## How to take input from users???

After knowing arrays in java, let's move towards how take inputs from scanner class in Java.



To import scanner class all you have to do is:

- 1) Import `java.util.Scanner`; //Add this line at top of the code.
- 2) Create object of Scanner class by saying :  
`Scanner scan = new Scanner(System.in);`
- 3) To read different types of input, look at the table below:

Method	Description
<code>nextBoolean()</code>	Reads a <b>Boolean</b> value from the user.
<code>nextByte()</code>	Reads a <b>Byte</b> value from the user.
<code>nextDouble()</code>	Reads a <b>Double</b> value from the user.
<code>nextFloat()</code>	Reads a <b>Float</b> value from the user.
<code>nextInt()</code>	Reads a <b>Integer</b> value from the user.
<code>nextLine()</code>	Reads a <b>Line</b> value from the user.
<code>nextLong()</code>	Reads a <b>Long</b> value from the user.
<code>nextShort()</code>	Reads a <b>Short</b> value from the user.

## Continuation with arrays....

Now that you know why we go for array approach and what are different types of arrays, let's start with how to make use of it.

**Example 1: Create an array to store the ages of 5 students**

### Solution:

```
import java.util.Scanner;

class Demo
{
    public static void main(String[] args)
    {
        int a[] = new int[5];

        Scanner scan = new Scanner(System.in);

        System.out.println("Enter the age:");
        a[0] = scan.nextInt();

        System.out.println("Enter the age:");
        a[1] = scan.nextInt();

        System.out.println("Enter the age:");
        a[2] = scan.nextInt();

        System.out.println("Enter the age:");
        a[3] = scan.nextInt();

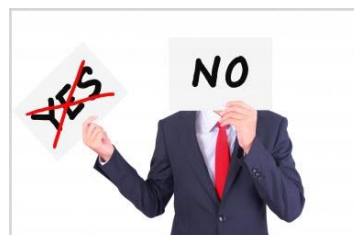
        System.out.println("Enter the age:");
        a[4] = scan.nextInt();

        System.out.println("Enter the age:");
        a[5] = scan.nextInt();

    }
}
```



**IS IT**  
***CORRECT?***





Whenever a set of instructions is repeating it's wise to use loops.

Let's see how to make the code efficient with the help of for loop.



```
import java.util.Scanner;

class Demo
{
    public static void main(String[] args)
    {
        int a[] = new int[5];

        Scanner scan = new Scanner(System.in);

        for(int i =0; i<=4; i++)
        {
            System.out.println("Enter the age:");

            a[i] = scan.nextInt();

        }
    }
}
```



But instead of specifying array size in the for loop condition part we can make use of a built-in property called as length. Which will give the length of the array by saying **a.length-1 ( -1 because the array indexing starts from 0).**

Let's see how length works for 2D, 3D array

Consider we have a 2D array,

```
int a[][] = new int[2][5];
```

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

**a.length** → Number of rows

**a[0].length** → Number of elements in a[0]

**a[1].length** → Number of elements in a[1]

**a[i].length** → Number of columns

Let's now consider 3D array,

```
int a[][][] = new int [2][3][5];
```

**a.length** → Number of blocks

**a[i].length** → Number of rows

**a[i][j].length** → Number of Columns

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

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

**Let us now try to code a 2D array**

```
import java.util.Scanner;

class Demo
{
    public static void main(String[] args)
    {
        int a[][] = new int[2][5];

        Scanner scan = new Scanner(System.in);

        for(int i =0; i<=a.length-1; i++)
        {
            for(int j=0; j<=a[i].length-1; j++)
            {
                System.out.println("Enter the age of class "+ i
                +"student"+ j );

                a[i][j] = scan.nextInt();
            }
        }
    }
}
```

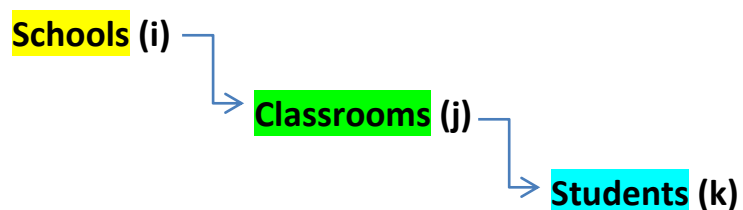


## Let us now try to understand a 3D array

**Example 3)** Create an array to store the ages of students belonging to 2 schools having 3 classrooms with 5 students each.

### Solution:

It is very important to know the different dimensions and order of it, here we have 3 dimensions (schools, classrooms, students).



Below given is the syntax of defining the 3D arrays in Java:

```
data_type array_name [ ] [ ] [ ] = new array_name [i] [j] [k];
```

- Here **data\_type**: data type of elements that will be stored in the array.  
**array\_name**: name of the array
- **new**: keyword to create an object in Java
- **i, j, k**: holds the numeric values for the various dimensions.

Let's now consider 3D array,

```
int a[][][] = new int [2][3][5];
```

**a.length** → Number of blocks

**a[i].length** → Number of rows

**a[i][j].length** → Number of Columns

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

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

Let's try to code this

```
import java.util.Scanner;
class Demo
{
    public static void main(String[] args)
    {
        int a[][][] = new int[2][3][5];
        Scanner scan = new Scanner(System.in);
        for(int i =0;i<=a.length-1;i++)
        {
            for(int j=0;j<=a[i].length-1;j++)
            {
                for(int k=0;k<=a[i][j].length-1;k++)
                {
                    System.out.println
                        ("Enter the age of student from school"
                        +i+" classroom "+j+" student "+k);
                    a[i][j][k] = scan.nextInt();
                }
            }
        }
    }
}
```



Although we now know how to create a multi-dimensional array, do you in above case every classroom was expecting same number of students. Whereas in reality inside a classroom there can be varying number of students. To achieve this in java we have **jagged arrays**.

**So let's see what exactly is jagged array..**

Jagged array is array of arrays such that **member arrays can be of different sizes**, i.e., we can create a 2-D or 3-D arrays but with **variable number of columns in each row**. These type of arrays are also known as Jagged arrays.

# understood

**Example 4)** Create an array to store the ages of students belonging to 2 classrooms where the first classroom has 2 students and second classroom has 5 students.

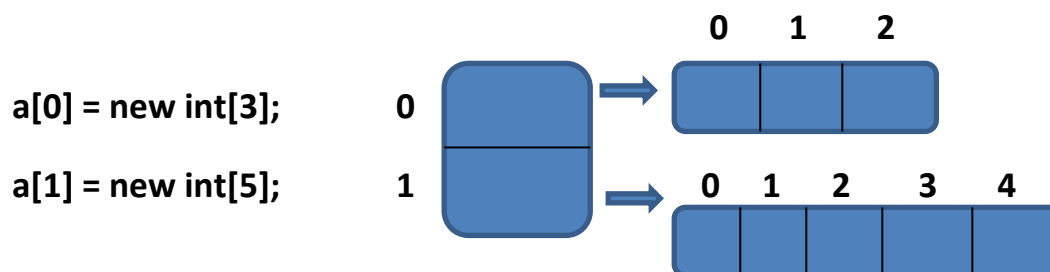
**Solution:**

Initialisation of jagged array →

```
int a[ ][ ] = new int[2][ ];
```



When not sure how many students are in each class,  
just leave it blank and specify separately.



```
import java.util.Scanner;
class Demo
{
    public static void main(String[] args)
    {
        int a[][] = new int[2][];
        a[0] = new int [3];
        a[1] = new int [5];
        Scanner scan = new Scanner(System.in);
        for(int i =0;i<=a.length-1;i++)
        {
            for(int j=0;j<=a[i].length-1;j++)
            {
                System.out.println
                    ("Enter the age of student from classroom "
                     +i+" student "+j);
                a[i][j] = scan.nextInt();
            }
        }
    }
}
```

Similarly try doing for 3D jagged array.

GOOD → GREAT

*Let us try to write the code for few other examples.*

**Example 1):** Create an array to store the ages of students belonging to 2 classrooms with 5 students each.

**Solution:**

```
import java.util.Scanner;
class Demo
{
    public static void main(String[] args)
    {
        int a[][] = new int[2][5];
        Scanner scan = new Scanner(System.in);

        for(int i=0;i<=a.length-1;i++)
        {
            for(int j=0;j<=a[i].length-1;j++)
            {
                System.out.println("Enter the ages of school"+i+"class"+j+"student: ");
                a[i][j] = scan.nextInt();
            }
        }
        System.out.println("The ages are: ");
        for(int i=0;i<=a.length-1;i++)
        {
            for(int j=0;j<=a[i].length-1;j++)
            {
                System.out.println(a[i][j]);
            }
        }
    }
}
```

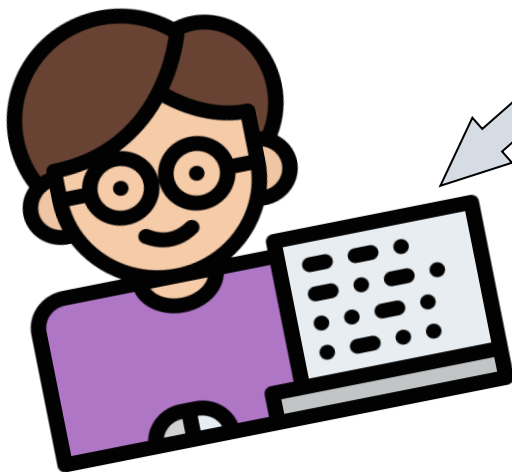


**Example 2): Create an array to store the ages of students belonging to 2 schools having 3 classrooms with 5 students each.**

**Solution:**

```
import java.util.Scanner;
class Demo
{
    public static void main(String[] args)
    {
        int a[][][] = new int[2][3][5];
        Scanner scan = new Scanner(System.in);

        for(int i=0;i<=a.length-1;i++)
        {
            for(int j=0;j<=a[i].length-1;j++)
            {
                for(int k=0;k<=a[i][j].length-1;k++)
                {
                    System.out.println("Enter the ages of school "+i+"class "+j+"student"+k);
                    a[i][j][k]= scan.nextInt();
                }
            }
        }
        System.out.println("The ages are: ");
        for(int i=0;i<=a.length-1;i++)
        {
            for(int j=0;j<=a[i].length-1;j++)
            {
                for(int k=0;k<=a[i][j].length-1;k++)
                {
                    System.out.println(a[i][j][k]);
                }
            }
        }
    }
}
```



***HE GOT THE OUTPUT. DID YOU?***



*Let us look at an example of two-dimensional jagged array.*

Example 3): Create an array to store the ages of students belonging to 2 classrooms where the first classroom has 3 students and second classroom has 5 students.

**Solution:**

```
import java.util.Scanner;
class Demo
{
    public static void main(String[] args)
    {
        int a[][] = new int[2][];
        a[0] = new int[3];
        a[1] = new int[5];
        Scanner scan = new Scanner(System.in);

        for(int i=0;i<=a.length-1;i++)
        {
            for(int j=0;j<=a[i].length-1;j++)
            {
                System.out.println("Enter the ages of class "+i+"student"+j);
                a[i][j]= scan.nextInt();
            }
        }
        System.out.println("The ages are: ");
        for(int i=0;i<=a.length-1;i++)
        {
            for(int j=0;j<=a[i].length-1;j++)
            {
                System.out.println(a[i][j]);
            }
        }
    }
}
```

# DID YOU KNOW?

**Bill Gates** began programming  
Computers at the age of 13.



Similarly let us try coding for 3D jagged array.

**Example 3):** Create an array to store the data given below:

School	Classrooms	Students
0	0	0-1
	1	0-2
	2	0-2
1	0	0-1
	1	0-2

**Solution:**

```
import java.util.Scanner;
class Demo
{
    public static void main(String[] args)
    {
        int a[][][] = new int[2][][];
        a[0] = new int[3][];
        a[1] = new int[2][];
        a[0][0] = new int[2];
        a[0][1] = new int[3];
        a[0][2] = new int[3];
        a[1][0] = new int[2];
        a[1][1] = new int[3];
        Scanner scan = new Scanner(System.in);

        for(int i=0;i<=a.length-1;i++)
        {
            for(int j=0;j<=a[i].length-1;j++)
            {
                for(int k=0;k<=a[i][j].length-1;k++)
                {
                    System.out.println("Enter the ages of school "+i+"class "+j+"student"+k);
                    a[i][j][k]= scan.nextInt();
                }
            }
        }

        System.out.println("The ages are: ");
        for(int i=0;i<=a.length-1;i++)
        {
            for(int j=0;j<=a[i].length-1;j++)
            {
                for(int k=0;k<=a[i][j].length-1;k++)
                {
                    System.out.println(a[i][j][k]);
                }
            }
        }
    }
}
```

## Disadvantages of Array

*It is important for one to understand when to use an array and when not to use an array.*

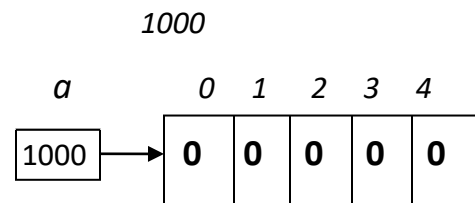
*And to understand that, let us have a look at few Cases on Array.*

### Case-1:

Let us consider a one-dimensional array,

**`int a[ ] = new int [5];`**

which will be represented in the memory as,



In the above statement, we tell JVM to create an array of size 5 which can store only integer type values. Once this information is given to the JVM, what type of data it has to store, then only those types of data are allowed to store within the array.

**Arrays once created can only store data of same type.**

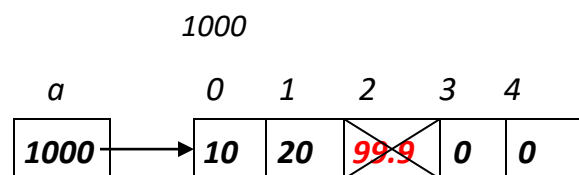
*Look at the below example:*

**`int a[ ] = new int [5];`**

**`a[0] = 10;`**

**`a[1] = 20;`**

**`a[2] = 99.9;`** ← **Error**



Here, in the example in `a[2] = 99.9;` we are trying to store float type of data which is not possible.

**Arrays can only store homogeneous data.**

### Case-2:

Let us consider a one-dimensional array,

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

```
a[0] = 10;
```

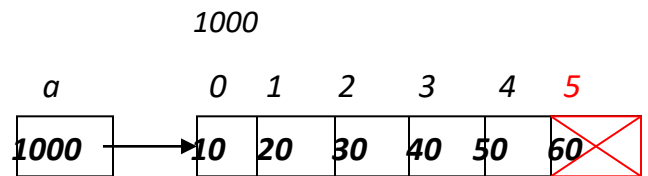
```
a[1] = 20;
```

```
a[2] = 30;
```

```
a[3] = 40;
```

```
a[4] = 50;
```

```
a[5] = 60;
```



In the above example, we have created an array of size 5. But by assigning **a[5] = 60;**

We are trying to store value 60 at the index 5, which means adding an extra size to the array. But this is not possible because the size of array is fixed, it cannot grow or shrink in size.

**Arrays cannot grow or shrink in size.**

### Case-3:

We know that RAM is the main memory and it is collection of bytes, if an array is created even that makes use of RAM.

Let's assume we create an array of size 5.

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

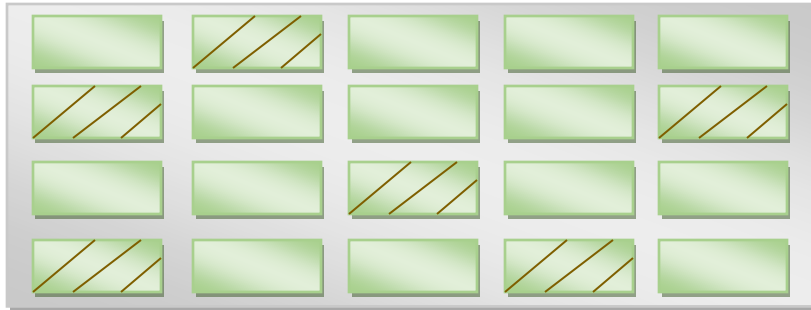
Arrays have an expectation that the size we mention to an array in terms of memory, those 5 memory locations must and should be **contiguous** (which means one next to the other) as shown below.



Contiguous memory

*But in real life, in our computer multiple software's would be working and all make use of RAM. Therefore, free memory locations in reality are rarely available in RAM in continuous way, most of time it is dispersed.*

*Arrays demand continuous memory allocations and cannot make use of **dispersed/scattered memory allocations.***



*Dispersed memory*

***Arrays require contiguous memory allocation.***

***Points to remember:***

- Arrays are objects
- They can even hold the reference variables of other objects
- They are created during runtime
- They are dynamic, created on the heap
- The Array length is fixed
- Array requires contiguous memory allocation
- They can only store homogeneous data.

**The original name for Java was**  
**Oak.**

