JAVA WORKSHOP

By

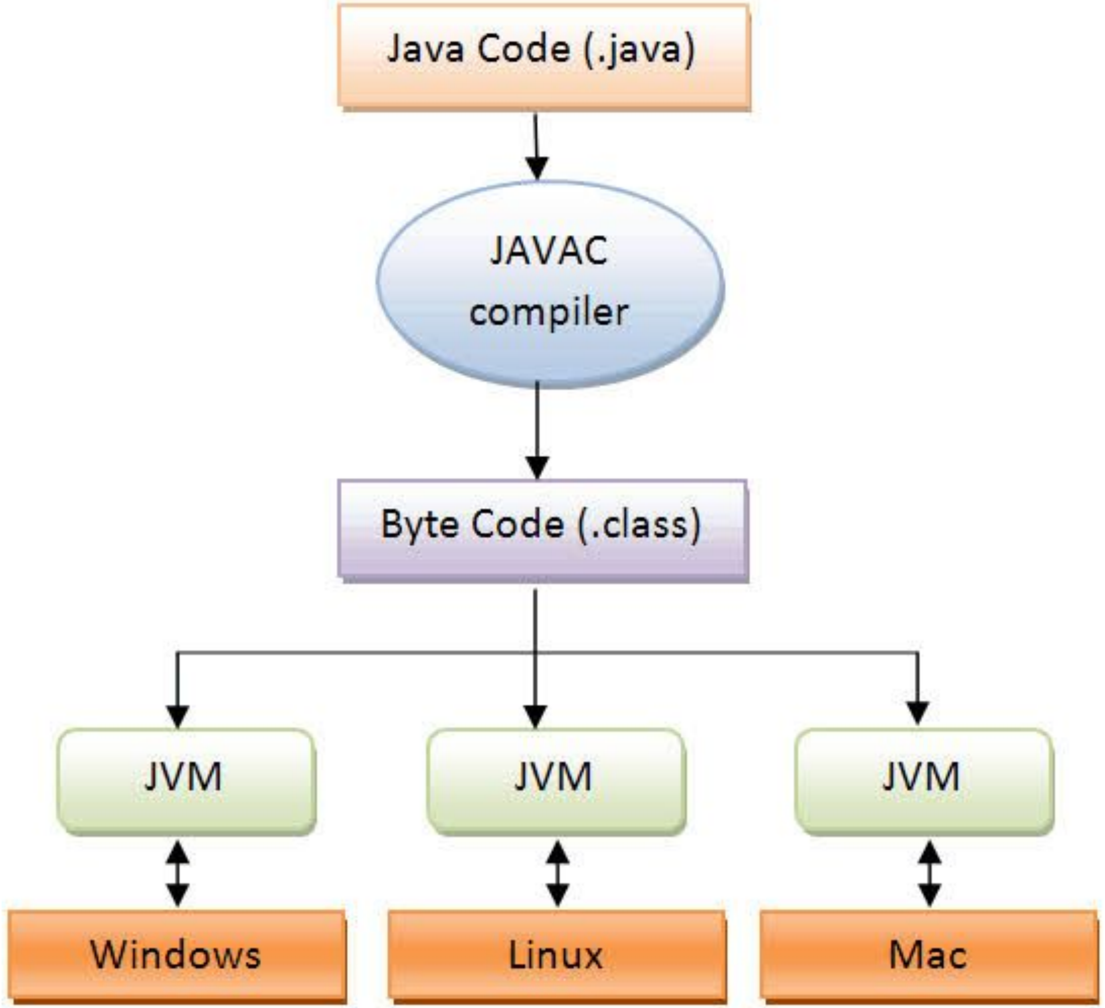
CSI - VESIT

WHAT IS JAVA?

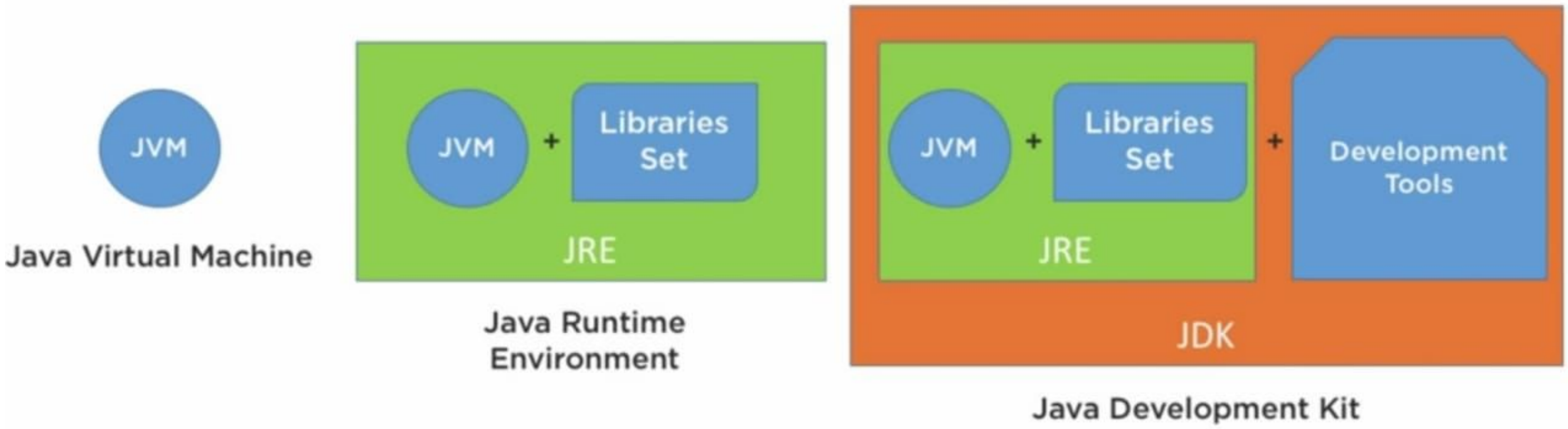
➔ Java is a general-purpose, class-based, object-oriented programming language

➔ It was released by James Gosling at Sun Microsystems in 1995.

➔ Java applications are called **WORA (Write Once Run Anywhere)**


Java Code (.java)
JAVAC compiler
Byte Code (.class)
JVM
JVM
JVM
Windows
Linux
Mac

JVM VS JRE VS JDK



DATA TYPES

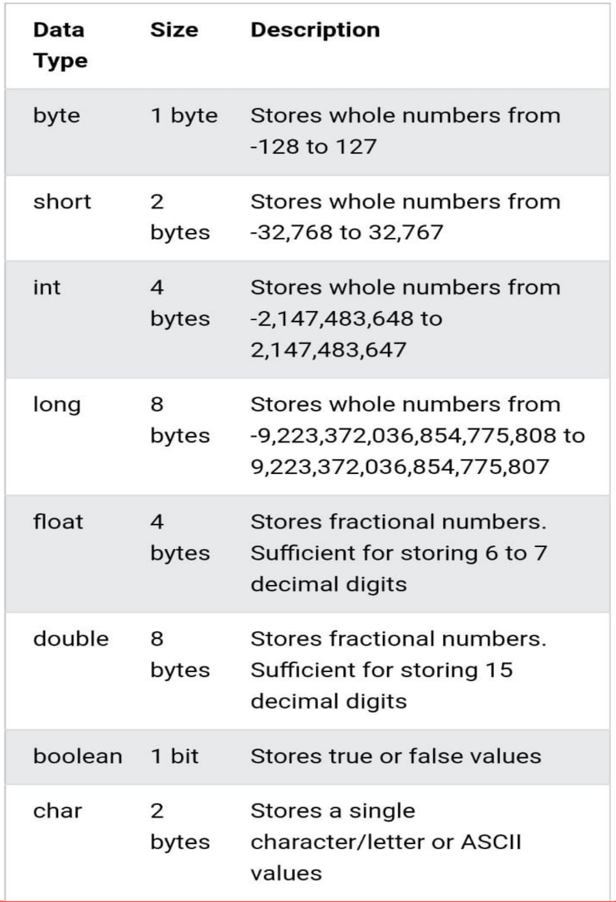
Data types specify the different sizes and values that can be stored in the variable.

There are 2 types of data types:

1) **Primitive data types**: The primitive data types include boolean, char, byte, short, int, long, float and double.

2) **Non-primitive data types**: The non-primitive data types include Classes, Interfaces, Arrays and Strings.

PRIMITIVE DATA TYPE



Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Formula for Range :

-2^(n-1) to (2^(n-1)-1)

where

n = no. of bits of primitive data type

OUTPUT METHODS IN JAVA

print()

This method prints the text on the console and the cursor remains at the end of the text. print() takes a single argument.

Syntax: System.out.print(*parameter*);

Input

```
int no = 8;
int num1 = 2, num2 = 5;
float ans= (float)num1/num2;
System.out.print("Hello World");
System.out.print(no);
System.out.print(num1+"/"+num2+" = "+ans);
```

Output

```
Hello World82/5 = 0.4
```

println()

This method prints the text on the console and the cursor moves to the start of next line . println() takes a single argument.

Syntax: System.out.println(*parameter*);

Input

```
int no = 8;
int num1 = 2, num2 = 5;
float ans= (float)num1/num2;
System.out.println("Hello World");
System.out.println(no);
System.out.println(num1+"/"+num2+" = "+ans);
```

Output

```
Hello World
8
2/5 = 0.4
```

printf()

This is similar to printf in C. printf() may take multiple arguments. This is used to format the output in Java.

Syntax: System.out.printf(*String*, *var1*, *var2*,......);

Input

```
int no = 8;
int num1 = 2, num2 = 5;
float ans= (float)num1/num2;
System.out.printf("Hello World");
System.out.printf("%d",no);
System.out.printf("%d/%d = %f",num1,num2,ans);
```

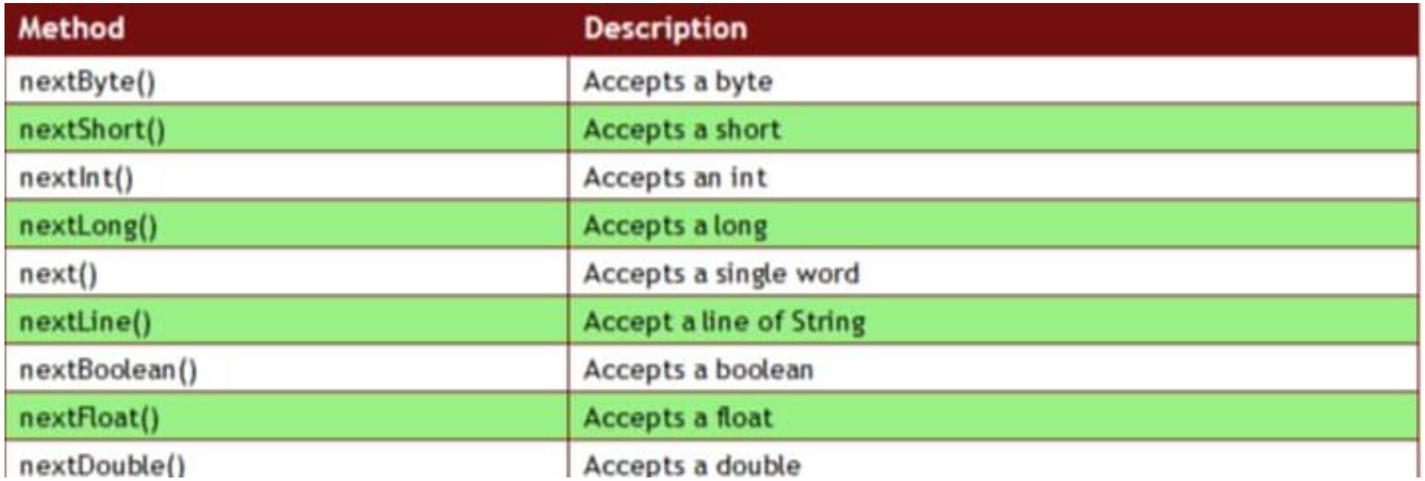
Output

```
Hello World82/5 = 0.400000
```

2 WAYS TO GET INPUT IN JAVA

SCANNER

Scanner is a class in java.util package used for obtaining the input of the primitive data types like int, double, etc. and strings.

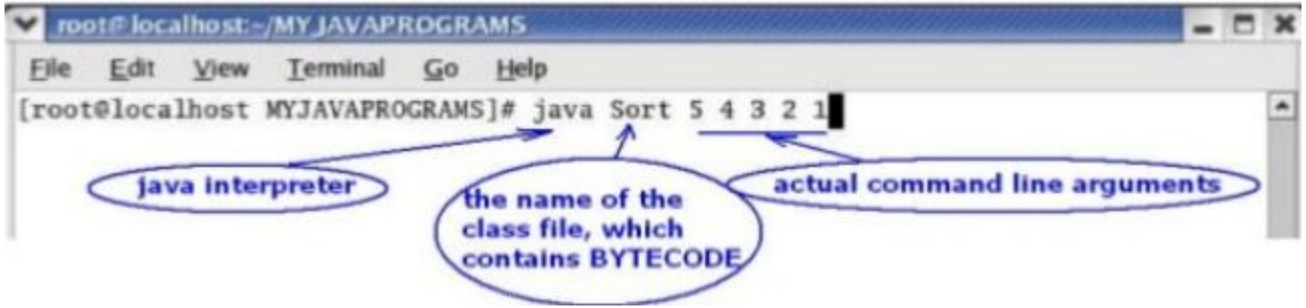


Method	Description
nextByte()	Accepts a byte
nextShort()	Accepts a short
nextInt()	Accepts an int
nextLong()	Accepts a long
next()	Accepts a single word
nextLine()	Accept a line of String
nextBoolean()	Accepts a boolean
nextFloat()	Accepts a float
nextDouble()	Accepts a double

COMMAND LINE ARGUMENTS

The java command-line argument is an argument i.e. passed at the time of running the java program.

The arguments passed from the console can be received in the java program and it can be used as an input.


root@localhost:~/MYJAVAPROGRAMS
File Edit View Terminal Go Help
java interpreter
the name of the class file, which contains BYTECODE
actual command line arguments

```
[root@localhost MYJAVAPROGRAMS]# java Sort 5 4 3 2 1
```

CONDITIONAL STATEMENTS

If … Else

- Use `if` to specify a block of code to be executed, if a specified condition is true
- Use `else` to specify a block of code to be executed, if the same condition is false
- Use `else if` to specify a new condition to test, if the first condition is false

Syntax

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
 }
```

SWITCH STATEMENT

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

SWITCH CASE SYNTAX

```
switch(variable){
        case value1:
                //code to be executed;
                break;  //optional
        case value2:
                //code to be executed;
                 break;  //optional
        …
        default:
                 // code to be executed if all cases are not matched;
}
```

LOOPS

FOR LOOP

The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is **fixed**, it is recommended to use for loop.

Syntax

```
for(initialization; condition; increment/decrement){
        //statement or code to be executed
}
```

WHILE LOOP

The Java *while loop* is used to iterate a part of the program repeatedly until the specified Boolean condition is true. As soon as the Boolean condition becomes false, the loop automatically stops.

Syntax

```
while (condition){
        //code to be executed
        Increment / decrement statement
}
```

DO WHILE LOOP

The Java *do-while loop* is used to iterate a part of the program repeatedly, until the specified condition is true. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use a do-while loop.

Syntax

```
do{
        //code to be executed / loop body
        //update statement
}while (condition);
```

FUNCTIONS / METHODS

WHAT? WHY? HOW?!

- A unit of code you can use multiple number of times.
- Always does the particular task for you.
- Parameterized & Non-parameterized.
- May or may not **return** a value.
- Syntax for Declaration:

```
returnType  functionName (<parameters>) {

        // Body of the Function

}
```

HOW TO ACCESS A METHOD?

```
public class Method  {
        void add ( ) {
                System.out.println("In addition");
        }
        public static void main (String[] args)  {
                Method obj = new Method( );
                obj.add();
         }
}
```

METHOD WITH PARAMS

```
public class Method {
        public int add ( int a, int b ) {
                return  a + b;
        }
        public static void main (String[] args) {
                Method obj = new Method( );
                obj.add ( 3, 4 );
        }
}
```

WHAT'S A STATIC METHOD?

```
public class Method {
        static void add ( ) {
                System.out.println(“In addition”);
        }
        public static void main (String[] args) {
                add();
        }
}
```

PULLING THE STRINGS...

Common Methods / Functions

- length
- charAt
- indexOf
- lastIndexOf
- substring
- toLowerCase
- toUpperCase
- replace
- concat
- equals
- equalsIgnoreCase
- compareTo
- trim
- endsWith
- startsWith

OOPS....

PRINCIPLES

- Objects
- Classes
- Data Abstraction
- Data Hiding
- Encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding

OBJECTS AND HOW TO CREATE IT?

- A unique entity which contains characteristics (**data**) and behavior (**functions**).
- An object is **an instance of a class.**
- **Syntax:**

```
ClassName  objectName  =  new  ClassName( );
```

CLASSES AND HOW TO CREATE IT?

- A class is an **object factory**.
- It is a template or blueprint from which objects are created.
- Example -

```
public class Test {
        int num = 30; // Attribute
        // Method
        void square (int x) {
                System.out.println(x * x);
        }
 }
```

HOW TO ACCESS ATTRIBUTES?

```
public class FirstClass {
   int s=7;
   public static void main (String[] args) {
      FirstClass obj = new FirstClass();
      int x = obj.s;
   }
}
```

CONSTRUCTORS

➔ A constructor in Java is a special method that is used to initialize objects.

➔ The constructor is called when an object of the class is created.

➔ It can be used to set initial values for object attributes.

```
public class FirstClass {
  int x;  // Create a class attribute

  // Create a class constructor for the FirstClass class
  public FirstClass() {
    x = 5;  // Set the initial value for the class attribute x
  }

  public static void main(String[] args) {
    FirstClass obj = new FirstClass(); // Create an object of class FirstClass (This will call the
constructor)
    System.out.println(obj.x); // Print the value of x
  }

}
```

INHERITANCE

- ➔ **Inheritance** in Java is a concept that **acquires the properties from one class to other classes**; for example, the relationship between father and son.

- ➔ In simple terms base class gets all the properties of its parent class.

```
 class Parent{
  public void show(){
       System.out.println(“In show”);
   }
}
class Student extends Parent{
   //It can now access all the features of Parent Class   eg.Show() method
}
```

ENCAPSULATION

➔ **Encapsulation** in Java is a mechanism to **wrap up variables(data) and methods(code) together as a single unit**.

➔ It is the process of **hiding information details** and **protecting data and behavior of the object**.

```
class Account{
private account_no;
public void getAccount(Account  a) {     //initialize getters and setters for the private variables
return a.account_no;
          }
}
```

Polymorphism

- ➔ **Polymorphism** in Java can be defined as a task that can perform a single action in different ways.
- ➔ Polymorphism occurs when there is **inheritance**, i.e. there are many classes that are related to each other.
- ➔ Polymorphism in java allows us to use **inherited properties to perform different tasks**.
- ➔ Polymorphism can be achieved by two methods:
  1. **Method Overloading**
  2. **Method Overriding**

TYPES OF POLYMORPHISM

- ➔ Static / Compile-Time Polymorphism
- ➔ Dynamic / Runtime Polymorphism

COMPILE TIME POLYMORPHISM

- In this process, the call to the method is resolved at **compile-time**.
- Compile-Time polymorphism is achieved through **Method Overloading**.
- **Method Overloading** is when a class has multiple methods with the same name, but the number, types and order of parameters and the return type of the methods are different.

RUN-TIME POLYMORPHISM

- ➔ In this process, the call to an overridden method is resolved dynamically at runtime rather than at compile-time
- ➔ Runtime polymorphism is achieved through **Method Overriding**.
- ➔ Method Overriding is done when a child or a subclass has a method with the same name, parameters and return type as the parent or the superclass.

ABSTRACTION

- ➔ **Data Abstraction** is the property by virtue of which only the essential details are displayed to the user.
- ➔ The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.
- ➔ Data Abstraction may also be defined as the **process of identifying only the required characteristics of an object ignoring the irrelevant details**.

INTERFACE

- ➔ An **interface** is a reference type in Java. It is similar to class.
- ➔ A class **implements** an interface, thereby inheriting the abstract methods of the interface.
- ➔ Unless the class that implements the interface is **abstract**, all the methods of the interface need to be defined in the class.

ABSTRACTION VS INTERFACE

➔ An abstract class permits you to make functionality that subclasses can implement or override whereas an interface only permits you to state functionality but not to implement it.

➔ A class can extend only one abstract class while a class can implement multiple interfaces.

➔ The interface does not have access modifiers. Everything defined inside the interface is assumed public modifier.

➔ Abstract Class can have an access modifier.

ABSTRACTION VS ENCAPSULATION

ABSTRACTION	ENCAPSULATION
In abstraction, problems are solved at the design or interface level.	While in encapsulation, problems are solved at the implementation level.
Abstraction is the method of hiding the unwanted information.	Whereas encapsulation is a method to hide the data in a single entity or unit along with a method to protect information from outside.
We can implement abstraction using abstract class and interfaces.	Whereas encapsulation can be implemented using by access modifier i.e. private, protected and public.
In abstraction, implementation complexities are hidden using abstract classes and interfaces.	While in encapsulation, the data is hidden using methods of getters and setters.

ARRAYS

ARRAY

- ➔ Arrays are used to store multiple values in a single variable.
- ➔ The elements of an array are stored in a contiguous memory location.
- ➔ Array is an object which contains elements of a similar data type.

TYPES OF ARRAY

	One-dimensional Array	Multi-dimensional Array
DEFINITION	Store a single list of the elements of the same data type.	Store a 'list of lists' of the elements of the same data type.
INITIALIZATION	datatype [] variable_name= new datatype[size]	datatype [][]....n variable_name= new datatype[size1][size2]...[sizen]
EXAMPLES	eg: int arr[ ] = new int [5]; //An array with five elements would be created. [1 2 3 4 5]	eg: int multi[ ][ ] = new int[3][3] //An array with three rows, three columns would be created. [1 2 3] [4 5 6] [7 8 9]

INITIALISING A 1-D ARRAY

Declaring An Array:

```
data_type arr[ ];     //First way
data_type[ ] arr;     //Second way
```

Instantiation Of An Array:

```
arr =new data_type[size];  //Allocating memory to the array
```

Example:

```
int arr[ ]; //Declaring an array of integers
arr = new int[10]; // Array would have 10 integers.
```

We can do the above step in a single line.   `int arr[ ] = new int[10]`

PROPERTIES OF AN ARRAY

1. **Length Property:** We can calculate the length of the array using the length property.
   eg: int arr[] = {1,2,3,4}; // Another method to initialize the array.
   return arr.length; /* This would return the length of the array. In this case, the length of the array is 4. */

2. **Clone Property:** This would make a clone/copy of the array. Only works for One Dimensional Array. We will store the array into another array.

   eg: int arr[ ] = {1, 2, 3};
   int new_arr[] = arr.clone();

PASSING ARRAY TO METHOD

We can pass arrays, like we add variables to a method.

Eg: We will make a method to return the largest number in the array.

```
public class LargestInArray {
        public static void main(String args[]) {
                int arr[ ] = {1, 2, 3, 4, 5};
                largest(arr); // Calling the method
        }

        public static void largest(int arr[ ]) //Passing array to the method {
                int largest = arr[0];
                for( int i=1; i<arr.length; i++ ) {
                        if( arr[i] > largest)
                                largest = arr[i];
                }
                System.out.println("Largest Number "+largest);
        }
}
```

OPERATION ON MULTIDIMENSIONAL ARRAY

Printing a 2-dimensional array in the form of a matrix.

```
public class DisplayMatrix {
        public static void main(String args[]) {
                int arr[ ][ ] = new int[3][3];
                display(arr);
        }

        public static void display(int arr[][]) {
                for( int i=0; i<3; i++ ) {
                        for( int j=0; j<3; j++) {
                                System.out.print(arr[i][j] + “  “);
                        }
                        System.out.println();
                }
        }
}
```

ADVANTAGES & DISADVANTAGES OF ARRAY

ADVANTAGES:

➔ **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
➔ **Random access:** We can get any data located at an index position.
➔ **Fast:** Arrays are fast as compared to primitive data types.

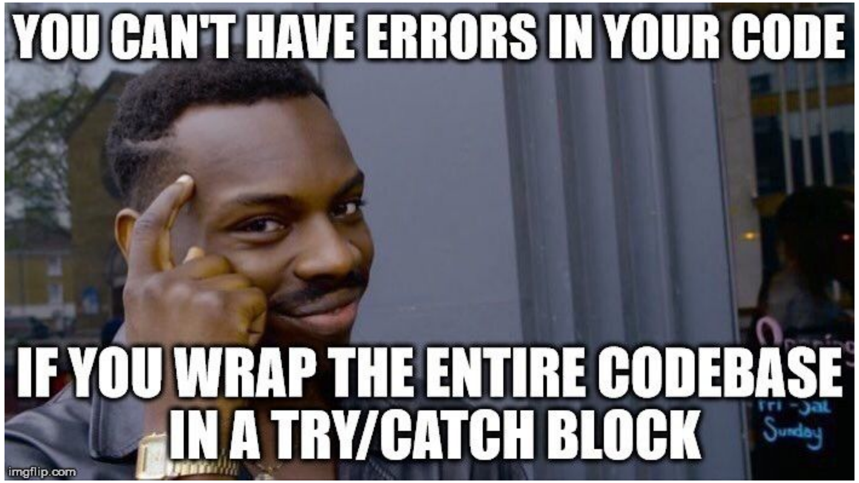
DISADVANTAGES:

➔ We can only insert/delete from the end of the array.
➔ We can store objects in an array, but we cannot store objects of different types.
➔ Arrays are static, i.e. we have to declare the size of the array during initialization.

EXCEPTION HANDLING

- ➔ Exception Handling is a method to handle errors, which may have occurred during the execution of the program.

- ➔ This would help us to maintain the normal flow of the program, as an exception would disrupt the flow of the program.


YOU CAN'T HAVE ERRORS IN YOUR CODE
IF YOU WRAP THE ENTIRE CODEBASE IN A TRY/CATCH BLOCK
imgflip.com

EXCEPTION & TYPES OF EXCEPTION

Exception are any unexpected event, which occurs during the execution of a program i.e at run time, that disrupts the normal flow of the program's instructions.

There are two types of Exceptions:

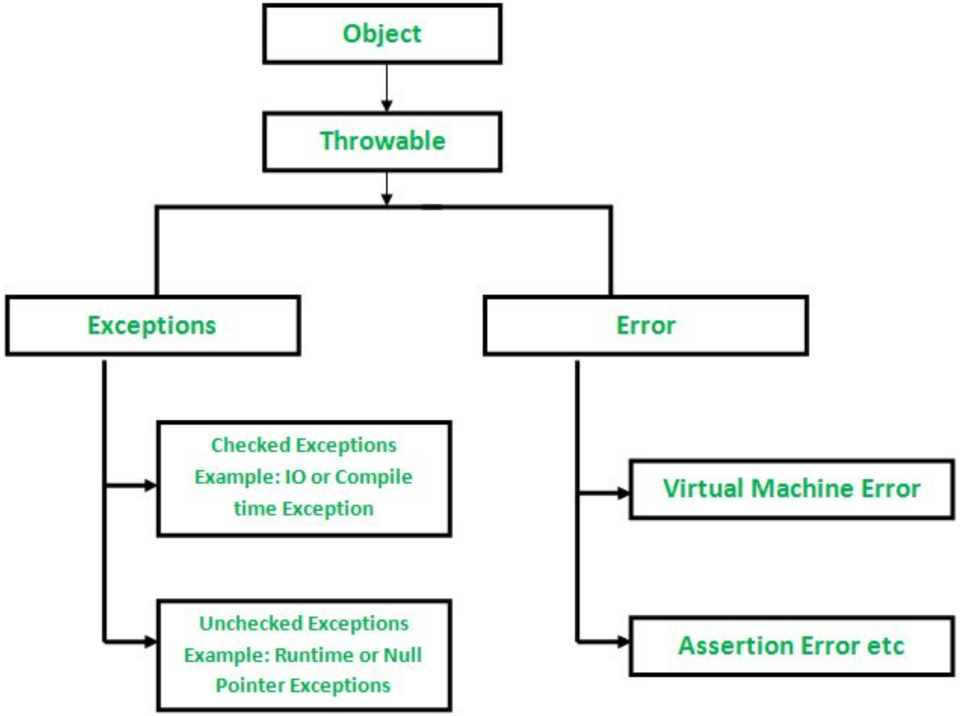
1. **Checked Exceptions:** These exceptions are checked during compile time of the program. It should be handled during compile-time, or else it would show a compilation error. eg: **ClassNotFoundException** : When the class is not found.

2. **Unchecked Exceptions:** These exceptions occur during running time of the program. eg: **ArithmeticException** : If we divide a number with zero.
   ArrayIndexOutOfBoundsException : If we try to print an array element which is not between 0 and the last element(length-1).

HIERARCHY OF JAVA EXCEPTION CLASS

The java.lang.Throwable is the root class of Java Exception hierarchy inherited by two subclasses: Exception and Error.

Any exceptions which are not handled within a Java Program, are caught by the JRE i.e. Java **R**untime **E**nvironment.


Object
Throwable
Exceptions
Error
Checked Exceptions Example: IO or Compile time Exception
Unchecked Exceptions Example: Runtime or Null Pointer Exceptions
Virtual Machine Error
Assertion Error etc

KEYWORDS USED IN EXCEPTION HANDLING

try	The "try" keyword is used to specify a block which contains an exception, which needs to be handled. The try block must be followed by catch.
catch	The "catch" block is used to handle the exception. It is always written after the try block.
finally	The "finally" block is used to execute the code i.e. written after the catch block. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception,explicitly in the code, inside the function or the block of code.
throws	The "throws" keyword is used in the method signature to declare an exception which might be thrown by the function.

SYNTAX FOR EXCEPTION HANDLING

For handling the error, first we will place the code that contains the error in the try block, and handle the error in the catch block.

```
public class ExceptionHandlingDemo {
        public static void main(String args[]) {
                int a = 10, b = 0;
                try {
                        int c = a/b; // Arithmetic Exception
                }
                catch(ArithmeticException e) {
                        System.out.println(e); //Prints the exception
                } // We can have multiple catch blocks
                finally {
                        // Remaining code
                }
        }
}
```

EXCEPTION HANDLING DEMO

In this demo, we will learn how to handle an Exception

ExceptionHandlingDemo.java

THANK YOU