**\*Topic 2 : Java Primitive Types , Literals and Operators.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Identifiers :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Rules to declare an Identifier :**

Identifier = Name in Java Program. (Variable , Method , Class).

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. The allowed characters are :

A to Z.

a to z.

0 to 9.

$.

\_.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. Identifiers cannot have same name as Java Keywords.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. Identifiers are case sensitive.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. We can use all the predefined Java Class names and interface names as Identifiers but it is not recommended.

Eg :

String String = “Gaurav”;

🡪 Correct.

import static java.lang.System.out;

class String

{

public static void main(String…args)

{

out.println(“Hello World”);

}

}

* Incorrect.

import static java.lang.System.out;

class String

{

public static void main(java.lang.String…args)

{

out.println(“Hello World”);

}

}

* Correct.

Note : Like above example JDK will always consider your defined class over predefined classes. So you have to clarify that it is the predefined class and not your own defined class.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Q. Which of he following are valid declarations in Java?**

**A. int int = 10; 🡪 Invalid.**

**B. int Int = 10; 🡪 Valid.**

**C. String string = “Gaurav”; 🡪 Valid.**

**D. String String = “Gaurav”; 🡪 Valid.**

**E. int #count = 10; 🡪 Invalid.**

**F. int 123total = 10; 🡪 Invalid.**

**G. class class = Test.class; 🡪 Invalid.**

**H. int total = 20; 🡪 Valid.**

**I. int \_total = 20; 🡪 Valid.**

**Primitive Data Types :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Java is a strongly typed programming language.

Java is not a PURE Object Oriented Programming Language.

As it does not includes features like Multiple Inheritance , Operator Overloading and primarily depends on Primitive Data Type. It is a different story that we can convert Primitive to Object but we still use Primitive which is a non Object.

The following are the Primitive Data Types in Java :

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **byte.**
2. **short.**
3. **int.**
4. **long.**
5. **double.**
6. **float.**
7. **char.**
8. **boolean.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Information about each Primitive Data Type :

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**BYTE :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Size : 1 Byte.**

**Range : -128 to 127.**

**Wrapper Class : Byte.**

**Default value : 0.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**SHORT :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Size : 2 Bytes.**

**Range : -32,768 to 32,767.**

**Wrapper Class : Short.**

**Default value : 0.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**INT :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Size : 4 Bytes.**

**Range : -2,14,74,83,648 to 2,14,74,83,647.**

**Wrapper Class : Integer.**

**Default value : 0.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**LONG :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Size : 8 Byte.**

**Range : -263 to 263 – 1.**

**Wrapper Class : Long.**

**Default value : 0.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**DOUBLE :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Size : 8 Bytes.**

**Range : -1.7e308 to 1.7e308**

**Wrapper Class : Double.**

**Default value : 0.0.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**FLOAT :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Size : 4 Bytes.**

**Range : -3.4e38 to 3.4e38**

**Wrapper Class : Float.**

**Default value : 0.0.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**CHAR :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Size : 2 Bytes.**

**Range : 0 to 65,535.**

**Wrapper Class : Character.**

**Default value : 0 [Represented by Space “ “].**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**BOOLEAN :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Size : Not Applicable.**

**Range : Not Applicable[But allowed values are “true” and “false”].**

**Wrapper Class : Boolean.**

**Default value : false.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Note : Null is not applicable for primitives. It is applicable only for Object types.**

What exceptions could be raised when misused these Primitive Data Types :

**If we try to convert from numeric to numeric :**

**For example :**

**int to double.**

**short to float.**

**……**

**Exception : Possible loss of precision / Lossy conversion.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**If we try to convert from numeric to non-numeric or vice versa :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**For example :**

**int to char.**

**boolean to byte.**

**…..**

**Exception : Incompatible Types.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Note : If we try to declare any variable that is non decimal point numeric other than int we will get Exception as Possible loss of precision because all the non decimal numbers are by default considered as int by the JVM and same applies for decimal point numbers. Where they are by default considered as double.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Literals :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Rules for char literals :

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Single char inside single quotes :

char ch = ‘a’;

**Valid.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Single char without quotes :

char ch = a;

**Invalid.**

**CE : Cannot find symbol.**

**Symbol : Variable a.**

**Location : Class class\_name.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Single char with double quotes :

char ch = “a”;

**Invalid.**

**CE : Incompatible Types.**

**Found : java.lang.String.**

**Required : char.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Multiple char with single quotes :

char ch = ‘ab’;

**Invalid.**

**CE : Unclosed char literal.**

**CE : Unclosed char literal.**

**CE : Not a statement.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Integer values including octal and hexadecimal too :

char ch = 97;

char ch = 0777;

char ch = OxFace;

char ch = 65,535;

**Valid.**

char ch = 65,536;

**Invalid.**

**CE : Possible loss of precision.**

**Found : int.**

**Required : char.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Unicode values :

char ch = ‘\u0000’; // 0000 here is a 4 digit hexadecimal number.

char ch = ‘\u0061’;

**Valid.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Escape Characters :

char ch = ‘\t’; // \\_ should be an escape character.

**Valid.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**List of escape characters for Java :**

**\t** **-> New Line.**

**\n -> Horizontal Tab.**

**\r -> Carriage returns.**

**\b -> Back Space.**

**\f -> Form Feed.**

**\’ -> Single Quote.**

**\” -> Double Quote.**

**\\ -> Back Slash.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Integral Literals (byte , short , int , long) :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. Decimal form : (Base 10)

int x = 100;

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. Octal form : (Base 8)

From 0 to 8 .

Must be prefixed with 0.

int x = 076;

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. Hexa-Decimal form : (Base 16)

From 0 to 9 and a to f including (A to F).

Must be prefixed with 0x or 0X.

int x = 0xbeef;

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. Binary from : (Base 2)

* Was introduced in 1.7 V.

0 or 1.

Must be prefixed with 0b or 0B.

int x = 0b111;

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Q. Which of the following are valid declarations :**

**A. int x = 0777; ----> Valid.**

**B. int x = 0786; ----> Invalid.**

**C. int x = 0XFace; ----> Valid.**

**D. int x = 0xBeef; ----> Valid.**

**E. int x = 0xBeer; ----> Invalid.**

**F. int x = 0xadda ----> Valid.**

**G. int x = 0B1010; ----> Valid.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Note : You can take all these four ways to declare integral literals and use them in the code , but when you try to print it. JVM will always print it in the decimal format.

**By default Integral type :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

By default every integral literal is of int type. If we suffixed it with l or L then it is treated as long type.

If we are assigning integral literal to byte variable and the value is in the range of byte type (-128 to 127) , then it is automatically considered as byte type.

byte b = 127; 🡪 Valid.

byte b = 128; 🡪 Invalid.

And same goes for short.

short s = 32767; 🡪 Valid.

short s = 32768; 🡪 Invalid.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Floating point literals :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

float f = 123.456; 🡪 Invalid.

float f = 123.456F; 🡪 Valid.

float f = 123.456f; 🡪 Valid.

double d = 123.456; 🡪 Valid.

double d = 1.234D; 🡪 Valid.

double d = 1.234d; 🡪 Valid. 🡪Not required.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Floating point literal only in decimal form :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

double d = 1.234; 🡪 Valid.

double d = 01.234; 🡪 Valid.

double d = 0x1.234; 🡪 Invalid.

double d = 0777; 🡪 Valid.

double d = 0786; 🡪 Invalid.

double d = 0786.0; 🡪 Valid.

double d = 0xFace; 🡪 Valid.

🡪 We can assign integral literals to floating point literals and then it will be converted by JVM.

Double d = 1.2e3; 🡪 Valid.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Usage of ‘\_’ symbol from 1.7 version for numeric literals :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Just for readability.

After compilation these underscore will be removed by Compiler.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Int i = 1\_234; 🡪 Valid.

Double d = 1\_\_\_2\_\_3\_45.678; 🡪 Valid.

Double d = \_123.456; 🡪 Invalid.

Double d = 123\_.456; 🡪 Invalid.

Double d = 123.456\_; 🡪 Invalid.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Char literal :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

char ch = ‘a’; 🡪 Valid.

char ch = “a”; 🡪 Invalid.

char ch = ‘ab’; 🡪 Invalid.

char ch = 97; 🡪 Valid.

char ch = 65535; 🡪 Valid.

char ch = 65536; 🡪 Invalid.

char ch = ‘\u0061’ 🡪 Valid.

Char ch = ‘\ubeef’ 🡪 Valid.

Char ch = ‘\uabcd’ 🡪 Valid.

Char ch = ‘\iface’ 🡪 Invalid.

Char ch = \u0061 🡪 Invalid.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Primitive Type Casting :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **Implicit Type Casting.**
2. **Explicit Type Casting.**

**Implicit :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

int x = ‘a’; 🡪 Valid.

Here we are assigning smaller data type value to a bigger data type.

Character = 2 Bytes.

Integer = 4 Bytes.

1. Compiler is responsible.
2. Smaller Data Type value to Bigger Data Type variable.
3. No loss of information.
4. Widening or upcasting.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Explicit :**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Left to Right 🡪 Implicit.

Explicit 🡨 Right to Left.

byte 🡪 short 🡪 int 🡪 long 🡪 float 🡪 double.

char 🡪 int.

The reason why we can perform implicit type casting from long to float is because memory representation of integral data types and floating point data types is different.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

byte b = (byte) integral/floating value.

short s = (short) integral/floating value.

int i = (int) integral/floating value.

long l = (long) floating value.

float f = (float) double;

char c = (char) int/long/floating value.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Note : We cannot perform type casting between char and byte/short.

The only case where we can perform types casting is when the char data type is final.

Eg.

final char ch = ‘a’;

byte b = (byte)ch;

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**