UNIT - I

Unit - 1 Syllabus:

Introduction: The history and evolution of Java, Java Buzz words, object-oriented programming, Data Types, Variables and Arrays, Operators, Control Statements.
Classes and Objects: Concepts, methods, constructors, types of constructors, constructor overloading, usage of static, access control, this keyword, garbage collection, finalize()method, overloading, parameter passing mechanisms, final keyword, nested classes and inner classes.
Utility Classes: Date, Calendar, Scanner, Random

Introduction:

The History and Evolution of Java:

- Java is a programming language and a platform.
- Java was developed by Sun Microsystems (which is now the subsidiary of Oracle) in the year 1995.
- James Gosling is known as the father of Java. Before Java, its name was Oak.
- James Gosling and his team changed the Oak name to Java.

Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.
Many java versions have been released till now.

- JDK Alpha and Beta (1995)
- JDK 1.0 (23rd Jan 1996)
- JDK 1.1 (19th Feb 1997)
- J2SE 1.2 (8th Dec 1998)
- J2SE 1.3 (8th May 2000)
- J2SE 1.4 (6th Feb 2002)
- J2SE 5.0 (30th Sep 2004)
- Java SE 6 (11th Dec 2006)
- Java SE 7 (28th July 2011)
- Java SE 8 (18th Mar 2014)
- Java SE 9 (21st Sep 2017)
- Java SE 10 (20th Mar 2018)

Java Buzz Goals:

The features of Java are also known as Java buzzwords. A list of the most important features of the Java language are:
1. Simple:
Java is very easy to learn, and its syntax is simple, clean and easy to understand.
According to Sun, Java language is a simple programming language because:

Java syntax is based on C++ (so easier for programmers to learn it after C++).
Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

2. Object - Oriented Programming:

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.
Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.
Basic concepts of OOPs are:

Object
Class
Inheritance
Polymorphism
Abstraction
Encapsulation

3. Platform Independent:

JAVA is platform independent because it having its own JVM so that it can run on any platform. Java is platform independent, which means once written you can run it any where.

JVM: Java Virtual Machine(JVM) is a engine that provides runtime environment to drive the Java Code or applications . It converts Java bytecode into machines language.

Byte-Code: Byte-code is a highly optimized set of instructions that is executed by the Java Virtual Machine.Byte code helps Java achieve both portability and security

4. Secured:

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

No explicit pointer

Java Programs run inside a virtual machine sandbox

5. Robust:

Robust simply means strong. Java is robust because:

It uses strong memory management.
There is a lack of pointers that avoids security problems.
There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
There are exception handling and the type checking mechanism in Java. All these points make Java robust.

6. Architecture Neutral:

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

7. Portable:

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

8. High - Perfomance:

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

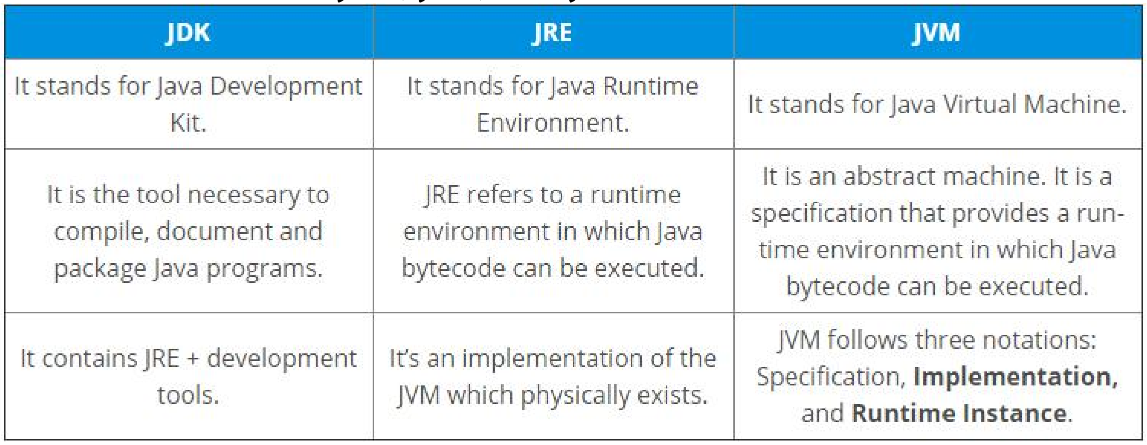
9. Multi - Threaded:

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

10. Dynamic:

- Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.
- Java supports dynamic compilation and automatic memory management (garbage collection).

Difference between JDK, JRE, and JVM

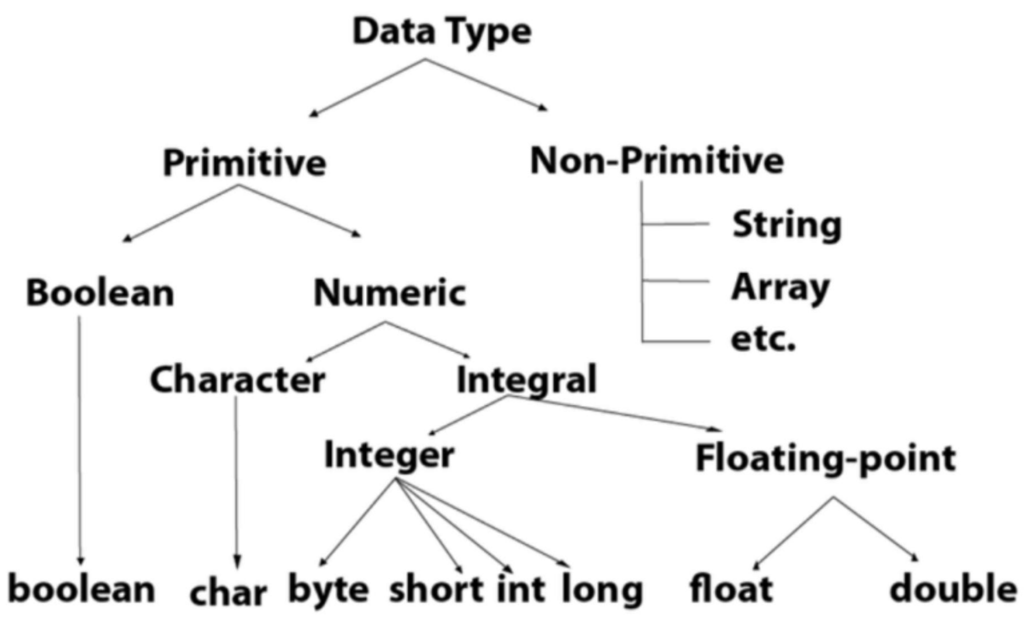


JDK	JRE	JVM
It stands for Java Development Kit.	It stands for Java Runtime Environment.	It stands for Java Virtual Machine.
It is the tool necessary to compile, document and package Java programs.	JRE refers to a runtime environment in which Java bytecode can be executed.	It is an abstract machine. It is a specification that provides a run-time environment in which Java bytecode can be executed.
It contains JRE + development tools.	It's an implementation of the JVM which physically exists.	JVM follows three notations: Specification, **Implementation,** and **Runtime Instance**.

Data Types:

Two types of data types in Java

- Primitive data types
- Non-primitive data types



Primitive Data Types

These are the most basic data types available in Java language.
There are 8 types of primitive data types:

- boolean
- byte
- char
- short
- int
- long
- float
- double

Boolean

- Store only two possible values: true and false
- This data type is used for simple flags that track true/false conditions.
- The Boolean data type specifies one bit of information

Example: Boolean one = false

Byte

- It is an 8-bit signed integer.
- Its value-range lies between -128 to 127
- Its minimum value is -128 and maximum value is 127.
- The byte data type is used to save memory in large arrays where the memory savings is most required.
- byte is 4 times smaller than an integer.

Short

- short data type is a 16-bit signed integer.
- Its value-range lies between -32,768 to 32,767.
- Its minimum value is -32,768 and maximum value is 32,767
- short data type can also be used to save memory just like byte data type.
- A short data type is 2 times smaller than an integer.

Int

- The int data type is a 32-bit signed integer.
- Its value-range lies between - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive).
- Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647.
- The int data type is generally used as a default data type for integral values

Long

- The long data type is a 64-bit integer.
- Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive).
- Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807.
- long data type is used when you need a range of values more than those provided by int.

Float

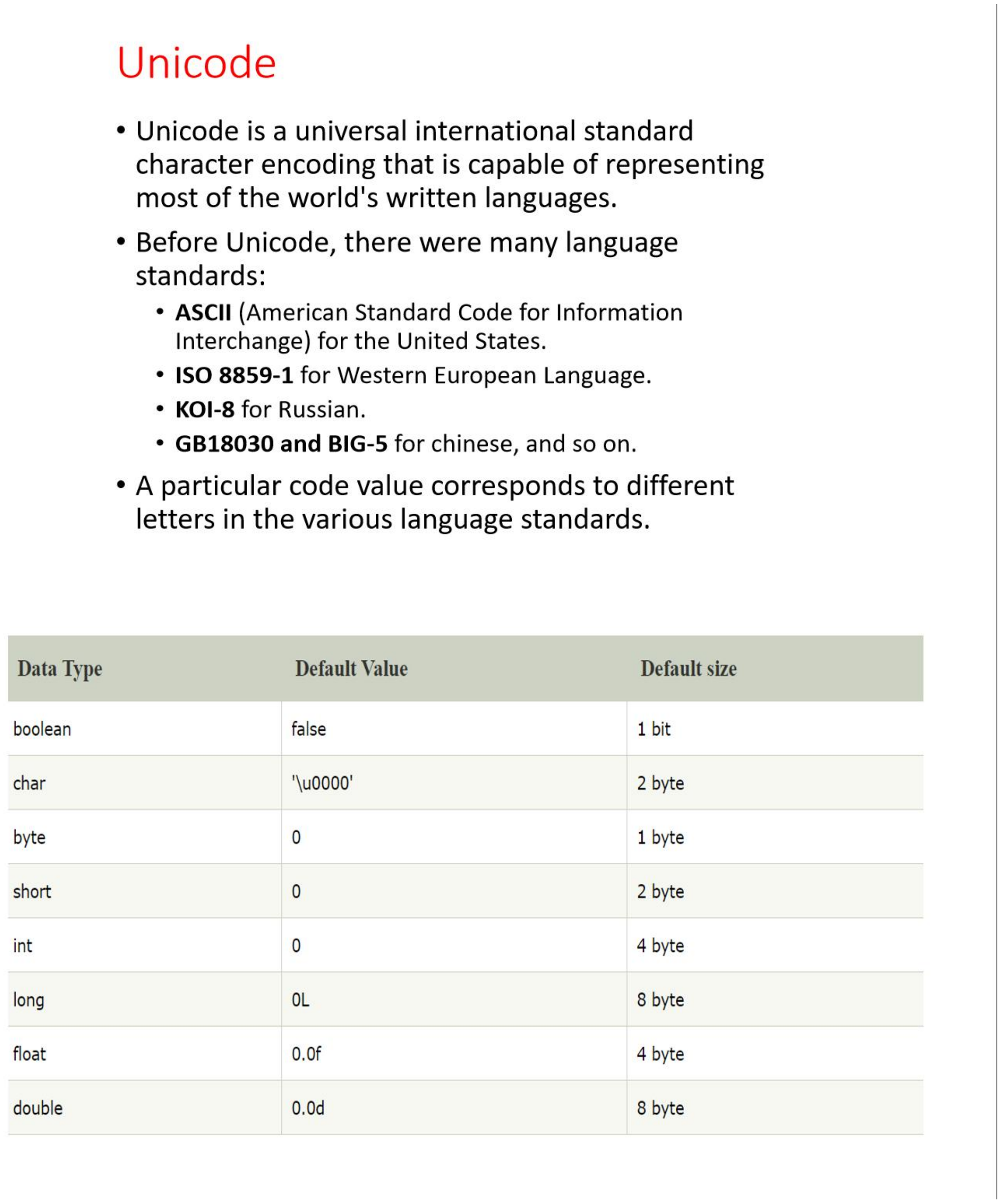
- The float data type is a single-precision 32-bit IEEE 754 floating point.
- Its value range is unlimited.
- The float data type should never be used for precise values, such as currency.

Double

- The double data type is a double-precision 64-bit IEEE 754 floating point.
- Its value range is unlimited.
- The double data type is generally used for decimal values just like float.

Char

- The char data type is a single 16-bit Unicode character.
- The char data type is used to store characters.
- **Example:** char letterA = 'A'



Unicode

- Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages.
- Before Unicode, there were many language standards:
  - **ASCII** (American Standard Code for Information Interchange) for the United States.
  - **ISO 8859-1** for Western European Language.
  - **KOI-8** for Russian.
  - **GB18030 and BIG-5** for chinese, and so on.
- A particular code value corresponds to different letters in the various language standards.

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Variables:

There are three types of variables in Java:

local variable
instance variable
static variable

Local Variable:

A variable declared inside the body of the method is called local variable.
We can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.
A local variable cannot be defined with "static" keyword.

Instance Variable:

A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.

It is called instance variable because its value is instance specific and is not shared among instances.

Static Variable:

A variable which is declared as static is called static variable. It cannot be local.

You can create a single copy of static variable and share among all the instances of the class.

Memory allocation for static variable happens only once when the class is loaded in the memory.

Arrays:

Array is a collection of similar type of elements which has contiguous memory location.

The elements of an array are stored in a contiguous memory location.

It is a data structure where we store similar elements.

We can store only a fixed set of elements in a Java array.

There are two types of array.

Single Dimensional Array

Multidimensional Array

Single Dimensional Arrays:

Syntax to Declare an Array in Java

```
dataType[] arr; (or)
dataType []arr; (or)
dataType arr[];
```

Instantiation of an Array in Java

```
arrayRefVar=new datatype[size
```

Example:

```
class Testarray{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
for(int i=0;i<a.length;i++)
System.out.println(a[i]);
}}
```

Multi-Dimensional Arrays:

Data is stored in row and column based index

Syntax

```
dataType[][] arrayRefVar; (or)
dataType [][]arrayRefVar; (or)
dataType arrayRefVar[][]; (or)
dataType []arrayRefVar[];
```

Example:

```
class Testarray3{
public static void main(String args[]){
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
//printing 2D array
for(int i=0;i<3;i++){
 for(int j=0;j<3;j++){
   System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}
}}
```

Operators:

* Operators in java are same as in C.

Control Statements:

* if, if-else, if-else if, nested if, switch, while, for, do-while, break, continue all these control structures follow the same syntax and working mechanism just as in C.

Apart from above, there are some other control statements such as for each loop, labelled break and labelled continue.

for-each loop:

The for-each loop is used to traverse array or collection in java.
It is easier to use than simple for loop because we don't need to increment value and use subscript notation.

Syntax:

```
for(Type var:array){
//code to be executed
}
```

Example:

```
public class ForEachExample {
public static void main(String[] args) {
     int arr[]={12,23,44,56,78};
   //Printing array using for-each loop
   for(int i:arr){
       System.out.println(i);
   }
}
}
```

Labelled Break:
In Labelled Break Statement, we give a label/name to a loop.
When this break statement is encountered with the label/name of the loop, it skips the execution any statement after it and takes the control right out of this labelled loop.
And, the control goes to the first statement right after the loop.
Example:

```
public class LabelledBreak
{
public static void main(String... ar)
{
loop2:
for(int i=0;i<2;i++)
for(int j=0;j<5;j++)
{
        if(j==2)
                break loop2;
        System.out.println("i = "+i);
        System.out.println("j = "+j);
}
System.out.println("Out of the loop");
}
}
```

Labelled Continue:
In Labelled Continue Statement, we give a label/name to a loop.
When this continue statement is encountered with the label/name of the loop, it skips the execution any statement within the loop for the current iteration and continues with the next iteration and condition checking in the labelled loop
Example:

```
class LabelledContinue
{
public static void main(String... ar)
{
loop:
for(int i=0;i<2;i++)
for(int j=0;j<5;j++)
{
        if(j==2)
                continue loop;
        System.out.println("i ="+i);
        System.out.println("j ="+j);
}
System.out.println("Out of the loop");
}
}
```

Classes and Objects:

Concepts:

OOP - Concept:

Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects.

It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object:

- Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc.
- It can be physical or logical.
- **An Object can be defined as an instance of a class.**
- An object contains an address and takes up some space in memory.
- Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class:

- **Collection of objects is called class. (OR) Class is the blueprint for creating object.**
- It is a logical entity.
- A class can also be defined as a blueprint from which you can create an individual object.
- Class doesn't consume any space.

Inheritance:

- **When one object acquires all the properties and behaviors of a parent object, it is known as inheritance.**
- It provides code reusability.
- To know about different types of inheritance refer to: https://www.dotnettricks.com/learn/oops/understanding-inheritance-and-different-types-of-inheritance

Polymorphism:

- **If one task is performed in different ways, it is known as polymorphism.**
- In Java, we use method overloading and method overriding to achieve polymorphism.

Abstraction:

Hiding internal details and showing functionality is known as abstraction.

For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.

https://docs.google.com/document/d/10zyrTxUwTnnjpMGuEB14dHH2JEZRANCE/edit?usp=share_link&ouid=113267319362652288542&rtpof=true&sd=true

Encapsulation:

Binding (or wrapping) code and data together into a single unit are known as encapsulation.

For example, a capsule, it is wrapped with different medicines.

A java class is the example of encapsulation..

Introducing Classes and Objects

Declaring a class:

```
class classname {
type instance-variable1;
type instance-variable2;
// ...
type instance-variableN;
type methodname1(parameter-list) {
// body of method
}
type methodname2(parameter-list) {
// body of method
}
// ...
type methodnameN(parameter-list) {
// body of method
}
}
```

The data, or variables, defined within a class are called instance variables.

The code is contained within methods.

The methods and variables defined within a class are called members of the class.

Java classes do not need to have a main( ) method.

Example:

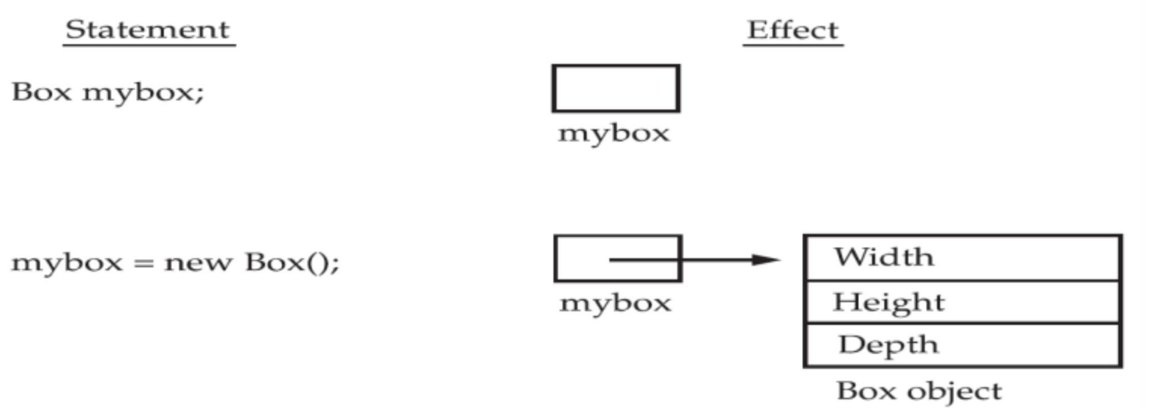
```
class Box {
double width;
double height;
double depth;
}
```

It is important to remember that a class declaration only creates a template; it does not create an actual object.

Declaring object:

A class is a logical construct. An object has physical reality.

Box mybox; // declare reference to object
mybox = new Box(); // allocate a Box object



To access these variables, you will use the dot (.) operator.

The dot operator links the name of the object with the name of an instance variable.

mybox.width = 100;

Use the dot operator to access both the instance variables and the methods within an object.

Example:

```
class Box {
double width;
double height;
double depth;
}
class BoxDemo {
public static void main(String args[]) {
Box mybox = new Box();
double vol;
mybox.width = 10;
mybox.height = 20;
mybox.depth = 15;
vol = mybox.width * mybox.height * mybox.depth;
System.out.println("Volume is " + vol);
}
}
```

Methods:

The general form of a method:

```
type name(parameter-list) {
// body of method
}
```

Example:

```
class Box {
double width;
double height;
double depth;
void volume() {
System.out.print("Volume is ");
System.out.println(width * height * depth);
}
}
class BoxDemo3 {
public static void main(String args[]) {
Box mybox1 = new Box();
Box mybox2 = new Box();
mybox1.width = 10;
mybox1.height = 20;
mybox1.depth = 15;
mybox2.width = 3;
mybox2.height = 6;
mybox2.depth = 9;
mybox1.volume();
mybox2.volume();
}
}
```

Constructors:

In Java, a constructor is a block of codes similar to the method.

It is called when an instance of the class is created.

At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java:

No-arg constructor.

Parameterized constructor.

It is called constructor because it constructs the values at the time of object creation.

It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

There are some rules defined for the constructor.

Constructor name must be the same as its class name

A Constructor must have no explicit return type

A Java constructor cannot be abstract, static, final, and synchronized

Default Constructor:

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax

<class_name>(){}
The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Example:

```
class Bike1{

Bike1()
{
System.out.println("Bike is created");
}
public static void main(String args[]){
Bike1 b=new Bike1();
}
}
```

Parameterized Constructor:

A constructor which has a specific number of parameters is called a parameterized constructor.

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example:

```
class Student4{
  int id;
  String name;
    Student4(int i,String n){
  id = i;
  name = n;
  }
     void display(){System.out.println(id+" "+name);}
    public static void main(String args[]){
    Student4 s1 = new Student4(111,"Karan");
  Student4 s2 = new Student4(222,"Aryan");
   s1.display();
  s2.display();
 }
}
```

Usage of static keyword:

The static keyword in Java is used for memory management mainly.

We can apply static keyword with

Variables

Methods

Blocks

If we declare any variable as static, it is known as a static variable.

The static variable can be used to refer to the common property of all objects (which is not unique for each object)
The static variable gets memory only once in the class area at the time of class loading.

Example:

```
class Student{
    int rollno;
   String name;
   static String college="ITS";
   Student(int rollno,String name, String college){
   this.rollno = rollno;
   this.name = name;
   this.college = college;
   }
   void display(){System.out.println(rollno+" "+name+" "+college);}
}
public class Main{
  public static void main(String args[]){
  Student s1 = new Student(111,"Karan","RVR");
  s1.display();
  Student s2 = new Student(222,"Aryan","VVIT");
  s2.display();
  s1.display();
  s2.display();
  }
}
```

Output:

```
111 Karan RVR
222 Aryan VVIT
111 Karan VVIT
222 Aryan VVIT
```

Static Method:

If we apply static keyword with any method, it is known as static method.
They can only directly call other static methods.
They can only directly access static data.
They cannot refer to this or super in any way.

Example:

```
class Student{
int rollno;
String name;
static String college="RVR";
static void change(){
college = "VVIT";
}
Student(int r,String n){
rollno=r;
name=n;
}
void display(){
        System.out.println(rollno+" "+name+" "+college);}
}
public class Main{
public static void main(String args[]){
Student.change();
Student s1=new Student(111,"Karan");
Student s2=new Student(222,"Aryan");
Student s3=new Student(333,"Sonoo");
s1.display();
s2.display();
s3.display();
}
}
```

Output:

```
111 Karan VVIT
222 Aryan VVIT
333 Sonoo VVIT
```

Access Control:

Encapsulation links data with the code that manipulates it.
Encapsulation provides another important attribute: access control.
How a member can be accessed is determined by the access modifier attached to its declaration.
Java's access modifiers are public, private, and protected.
protected applies only when inheritance is involved.
When a member of a class is modified by public, then that member can be accessed by any other code.
When a member of a class is specified as private, then that member can only be accessed by other members of its class.

Example for declaration:

```
public int i;
private double j;
private int myMethod(int a, char b) { //...
```

this Keyword:

Sometimes a method will need to refer to the object that invoked it.
To allow this, Java defines the this keyword.
this can be used inside any method to refer to the current object.
That is, this is always a reference to the object on which the method was invoked.
it is illegal in Java to declare two local variables with the same name inside the same or enclosing scopes.
Interestingly, we have local variables, including formal parameters to methods, which overlap with the names of the class' instance variables.

However, when a local variable has the same name as an instance variable, the local variable hides the instance variable.

We can use it to resolve any namespace collisions that might occur between instance variables and local variables.

Consider another version of Box( ), which uses width, height, and depth for parameter names and then uses this to access the instance variables by the same name.

```
Box(double width, double height, double depth) {
this.width = width;
this.height = height;
this.depth = depth;
}
```

Garbage Collection:

Objects are dynamically allocated by using the new operator, We might be wondering how such objects are destroyed and their memory released for later reallocation.

In some languages, such as C++, dynamically allocated objects must be manually released by use of a delete operator.

Java takes a different approach; it handles deallocation automatically.

The technique that accomplishes this is called garbage collection.

It works like this: when no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed.

Finalize() Method:

Finalize() is the method.

This method is called just before an object is garbage collected.

finalize() method overrides to dispose system resources, perform clean-up activities and minimize memory leaks.

Syntax:

```
 void finalize()
```

Overloading:

Having same name but different in parameters, it is known as Overloading.

Two types of overloadings

Method Overloading

Constructor Overloading

Method Overloading:

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

Method overloading increases the readability of the program.

There are two ways to overload the method in java

By changing number of arguments

By changing the data type

In Java, Method Overloading is not possible by changing the return type of the method only.

Example:

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

Constructor Overloading:

In Java, a constructor is just like a method but without return type.

It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists.

They are arranged in a way that each constructor performs a different task.

Example:

```
class Student5{
   int id;
   String name;
   int age;
   Student5(int i,String n){
   id = i;
   name = n;
   }
   Student5(int i,String n,int a){
   id = i;
   name = n;
   age=a;
   }
   void display(){System.out.println(id+" "+name+" "+age);}
   public static void main(String args[]){
   Student5 s1 = new Student5(111,"Karan");
   Student5 s2 = new Student5(222,"Aryan",25);
   s1.display();
   s2.display();
  }
}
```

Parameter passing mechanisms:

There are two types:

1. Call by Value
2. Call by reference

Formal Parameter : A variable and its type as they appear in the prototype of the function or method.

Syntax:

Function_name(datatype variable_name)

Actual Parameter : The variable or expression corresponding to a formal parameter that appears in the function or method call in the calling environment.

Syntax:

func_name(variable_name)

Call by Value:

If we call a method passing a value, it is known as call by value.

The changes being done in the called method, is not affected in the calling method.

Example:

```
class Operation{
 int data=50;

 void change(int data){
 data=data+100;//changes will be in the local variable only
 }

 public static void main(String args[]){
   Operation op=new Operation();

   System.out.println("before change "+op.data);
   op.change(500);
   System.out.println("after change "+op.data);

 }
}
```

Call by Reference:

If we pass object in place of any primitive value, original value will be changed.

Example:

```
class Operation2{
 int data=50;
 void change(Operation2 op){
 op.data=op.data+100;//changes will be in the instance variable
 }
 public static void main(String args[]){
   Operation2 op=new Operation2();
  System.out.println("before change "+op.data);
   op.change(op);//passing object
   System.out.println("after change "+op.data);

 }
}
```

Final Keyword:

The final keyword in java is used to restrict the user.

The java final keyword can be used in many context. Final can be:

variable
method
class

Final Variable:

If you make any variable as final, you cannot change the value of final variable(It will be constant).

```
class Bike9{
 final int speedlimit=90;
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Bike9 obj=new  Bike9();
 obj.run();
 }
}
```

Final Method:

If you make any method as final, you cannot override it.

```
class Bike{
  final void run(){System.out.println("running");}
}
class Honda extends Bike{
   void run(){System.out.println("running safely with 100kmph");}
   public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
   }
}
```

Final Class:

If you make any class as final, you cannot extend it.

```
final class Bike{}
class Honda1 extends Bike{
  void run(){System.out.println("running safely with 100kmph");}
  public static void main(String args[]){
  Honda1 honda= new Honda1();
  honda.run();
  }
}
```

Blank Final:

A final variable that is not initialized at the time of declaration is known as blank final variable.

If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed.

Example:

```
class Bike10{
 final int speedlimit;//blank final variable
 Bike10(){
 speedlimit=70;
 System.out.println(speedlimit);
 }
 public static void main(String args[]){
  new Bike10();
 }
}
```

Nested Classes:

There are two types of nested classes

- Static nested classes.
- Non-static classes.

Static Nested class:

- A static class i.e. created inside a class is called static nested class in java.
- It can be accessed by outer class name.
- It can access static data members of outer class including private.
- Static nested class cannot access non-static (instance) data member or method.

Example:

```
class TestOuter1{
 static int data=30;
 static class Inner{
  void msg(){System.out.println("data is "+data);}
 }
 public static void main(String args[]){
 TestOuter1.Inner obj=new TestOuter1.Inner();
 obj.msg();
 }
}
```

Output:

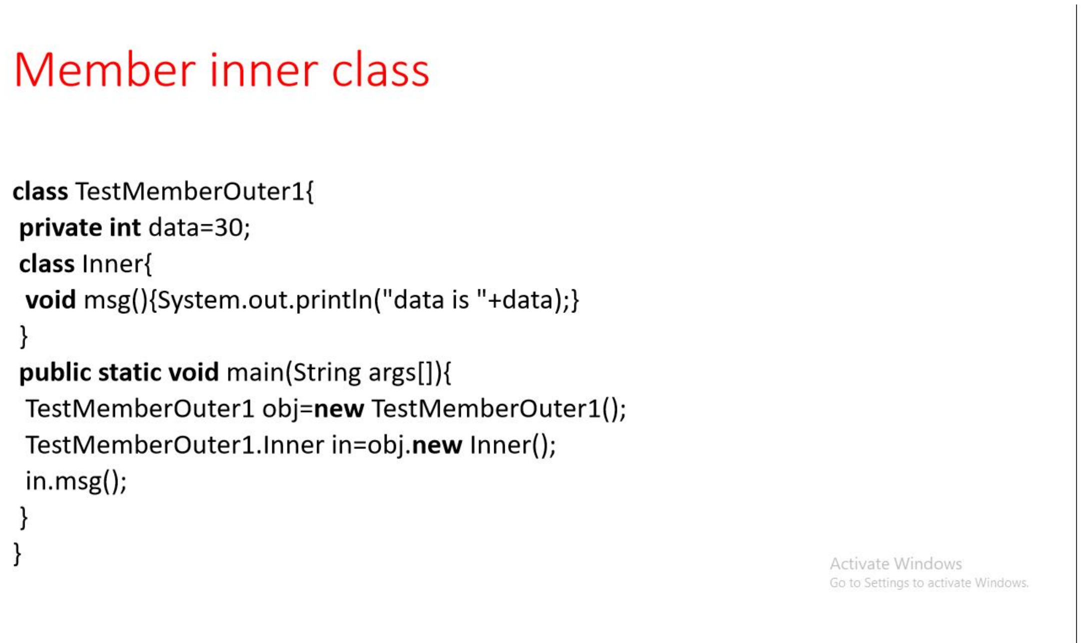
```
data is 30
```

Inner Class:

- The non-static nested classes are also known as inner classes.
- Java inner class is a class which is declared inside the class or interface.
- We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.
- Additionally, it can access all the members of outer class including private data members and methods.

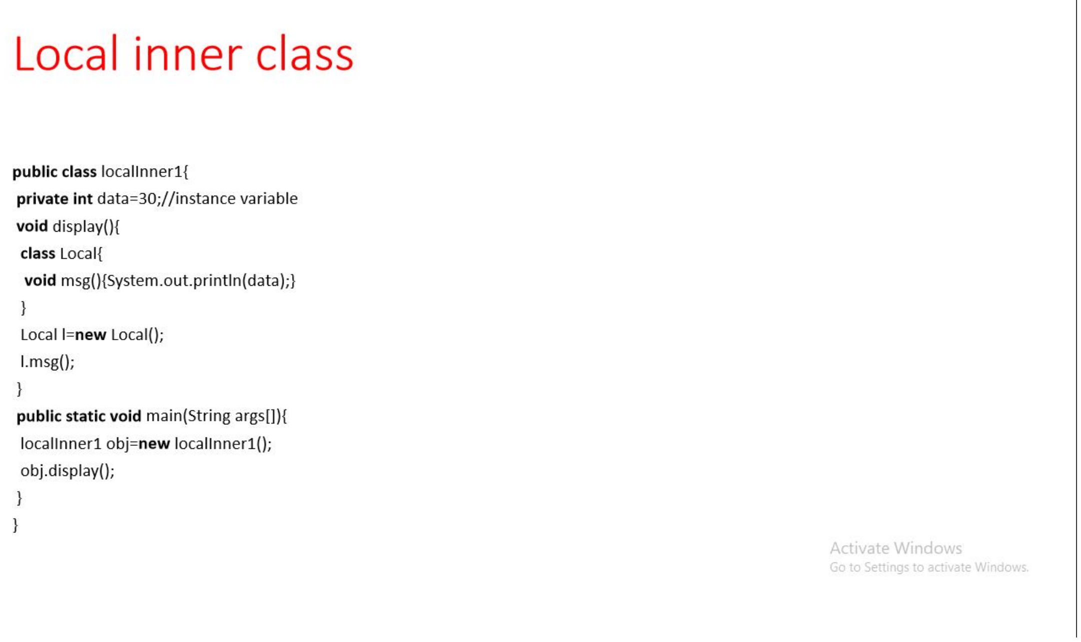
Syntax of Inner class

```
class Java_Outer_class{
 //code
 class Java_Inner_class{
  //code
 }
}
```



Member inner class

```java
class TestMemberOuter1{
 private int data=30;
 class Inner{
  void msg(){System.out.println("data is "+data);}
 }
 public static void main(String args[]){
  TestMemberOuter1 obj=new TestMemberOuter1();
  TestMemberOuter1.Inner in=obj.new Inner();
  in.msg();
 }
}
```



Local inner class

```java
public class localInner1{
 private int data=30;//instance variable
 void display(){
  class Local{
   void msg(){System.out.println(data);}
  }
  Local l=new Local();
  l.msg();
 }
 public static void main(String args[]){
  localInner1 obj=new localInner1();
  obj.display();
 }
}
```

Utility Classes:

The Collections utility class consists exclusively of static methods that operate on or return collections.

Date:

The class Date represents a specific instant in time, with millisecond precision. The Date class of java.util package implements Serializable, Cloneable and Comparable interface. It provides constructors and methods to deal with date and time with java.

Constructors

- **Date() :** Creates date object representing current date and time.
- **Date(long milliseconds) :** Creates a date object for the given milliseconds since January 1, 1970, 00:00:00 GMT.
- **Date(int year, int month, int date)**
- **Date(int year, int month, int date, int hrs, int min)**
- **Date(int year, int month, int date, int hrs, int min, int sec)**
- **Date(String s)**

Note : The last 4 constructors of the Date class are Deprecated.
Example:

```
// Java program to demonstrate constuctors of Date
import java.util.*;

public class Main
{
        public static void main(String[] args)
        {
                Date d1 = new Date();
                System.out.println("Current date is " + d1);
                Date d2 = new Date(2003,0,16,11,27,7);
                System.out.println("Date represented is "+ d2 );
        }
}
```

Output:

```
Current date is Sun Dec 04 15:44:40 GMT 2022
Date represented is Fri Jan 16 11:27:07 GMT 3903
```

Important Methods

- **boolean after(Date date) :** Tests if current date is after the given date.
- **boolean before(Date date) :** Tests if current date is before the given date.
- **int compareTo(Date date) :** Compares current date with given date. Returns 0 if the argument Date is equal to the Date; a value less than 0 if the Date is before the Date argument; and a value greater than 0 if the Date is after the Date argument.
- **long getTime() :** Returns the number of milliseconds since January 1, 1970, 00:00:00 GMT represented by this Date object.
- **void setTime(long time) :** Changes the current date and time to given time.

Example:

```
// Program to demonstrate methods of Date class
import java.util.*;
```

```
public class Main
{
        public static void main(String[] args)
        {
                // Creating date
                Date d1 = new Date(2000, 11, 21);
                Date d2 = new Date(); // Current date
                Date d3 = new Date(2010, 1, 3);

                boolean a = d3.after(d1);
                System.out.println("Date d3 comes after " +
                                                "date d2: " + a);

                boolean b = d3.before(d2);
                System.out.println("Date d3 comes before "+
                                                "date d2: " + b);

                int c = d1.compareTo(d2);
                System.out.println(c);

                System.out.println("Miliseconds from Jan 1 "+
                                "1970 to date d1 is " + d1.getTime());

                System.out.println("Before setting "+d2);
                d2.setTime(204587433443L);
                System.out.println("After setting "+d2);
        }
}
```

Output:

```
Date d3 comes after date d2: true
Date d3 comes before date d2: false
1
Miliseconds from Jan 1 1970 to date d1 is 60935500800000
Before setting Sun Dec 04 15:50:19 GMT 2022
After setting Fri Jun 25 21:50:33 GMT 1976
```

Calender:

Calendar class in Java is an abstract class that provides methods for converting date between a specific instant in time and a set of calendar fields such as MONTH, YEAR, HOUR, etc. It inherits Object class and implements the Comparable, Serializable, Cloneable interfaces.

As it is an Abstract class, so we cannot use a constructor to create an instance. Instead, we will have to use the static method Calendar.getInstance() to instantiate and implement a sub-class.

- **Calendar.getInstance():** return a Calendar instance based on the current time in the default time zone with the default locale.
- **Calendar.getInstance(TimeZone zone)**
- **Calendar.getInstance(Locale aLocale)**
- **Calendar.getInstance(TimeZone zone, Locale aLocale)**

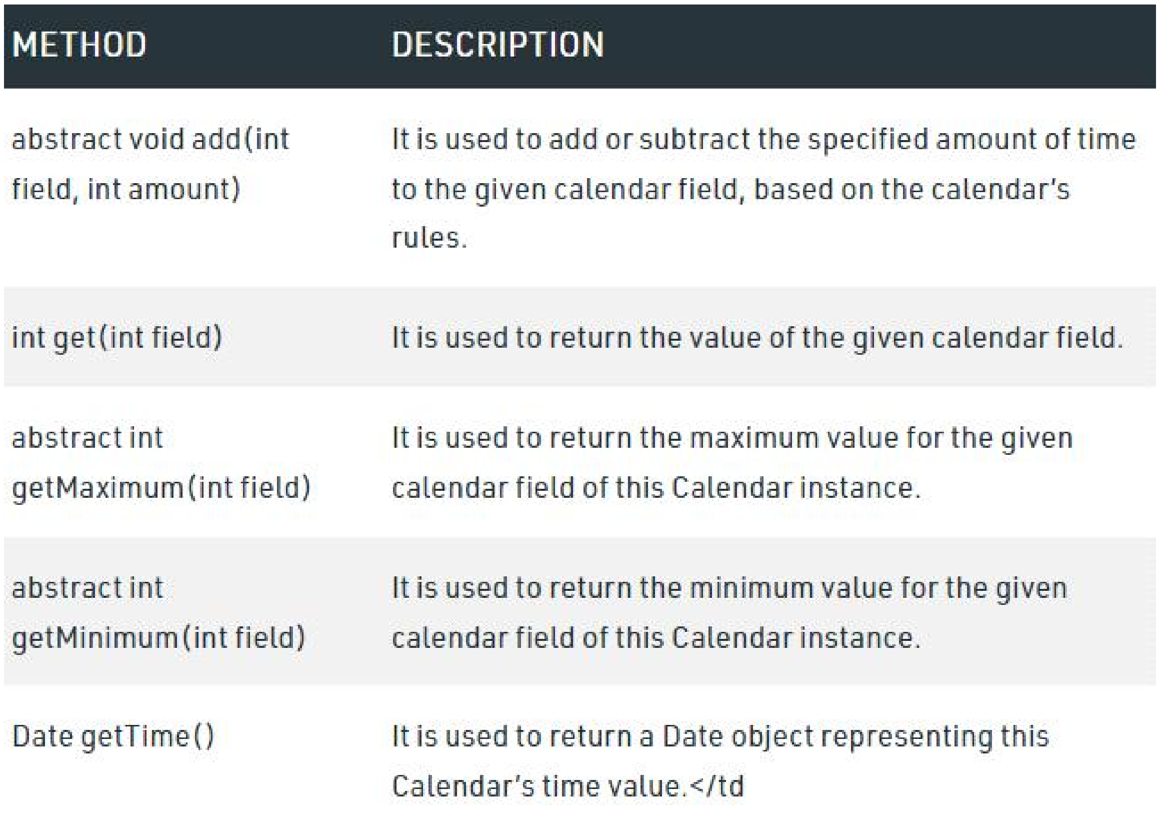
Example:

```
// Date getTime(): It is used to return a
// Date object representing this
// Calendar's time value.

import java.util.*;
public class Main{
        public static void main(String args[])
        {
                Calendar c = Calendar.getInstance();
                System.out.println("The Current Date is:" + c.getTime());
        }
}
```

Output:

```
The Current Date is:Sun Dec 04 15:53:45 GMT 2022
```



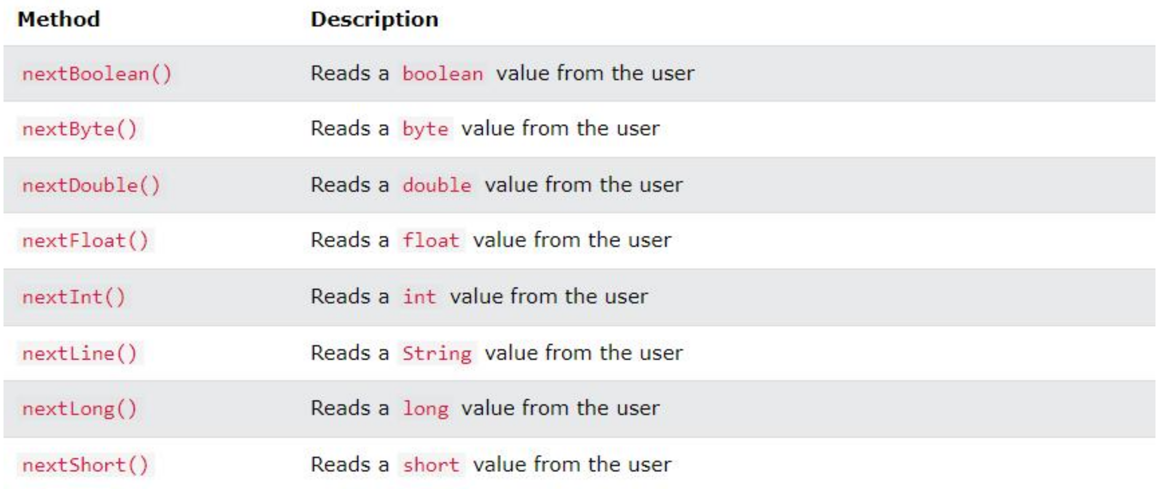
METHOD	DESCRIPTION
abstract void add(int field, int amount)	It is used to add or subtract the specified amount of time to the given calendar field, based on the calendar's rules.
int get(int field)	It is used to return the value of the given calendar field.
abstract int getMaximum(int field)	It is used to return the maximum value for the given calendar field of this Calendar instance.
abstract int getMinimum(int field)	It is used to return the minimum value for the given calendar field of this Calendar instance.
Date getTime()	It is used to return a Date object representing this Calendar's time value.</td

for examples refer to:
https://www.geeksforgeeks.org/calendar-class-in-java-with-examples/

Scanner:

The Scanner class is used to get user input, and it is found in the java.util package.

To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation.



Method	Description
`nextBoolean()`	Reads a `boolean` value from the user
`nextByte()`	Reads a `byte` value from the user
`nextDouble()`	Reads a `double` value from the user
`nextFloat()`	Reads a `float` value from the user
`nextInt()`	Reads a `int` value from the user
`nextLine()`	Reads a `String` value from the user
`nextLong()`	Reads a `long` value from the user
`nextShort()`	Reads a `short` value from the user

Note: We can use next() instead of nextLine() for reading a String as Input.

Random:

Refer to : https://www.javatpoint.com/post/java-random

***