**java.lang package – Part-01**

* **Agenda:**

1. **Introduction**
2. **Object class**
3. **String class**
4. **StringBuffer class**
5. **StringBuilder class (introduced in 1.5)**
6. **Wrapper Classes**
7. **Auto-Boxing & Auto Unboxing.**

* **Introduction:**

For writing any Java program whether it is simple or complex. The most commonly required classes and interface are grouped into a separate package which is nothing but java.lang package.

We are not required to import java.lang packge explicitly because all classes and interfaces present in lang package by default available to every Java program.

* **java.lang.Object:**

The most commonly required methods for every Java class (whether it are predefined or user-defined class) are defined in a separate class, which is nothing but Object class.

Every class in Java is the child class of Object either directly or indirectly so that Object class methods by default available to every java class. Hence, Object class is considered as root of all java classes.

Note:

1. If our class doesn’t extend any other class, then only our class is the direct child class of Object.

class A{

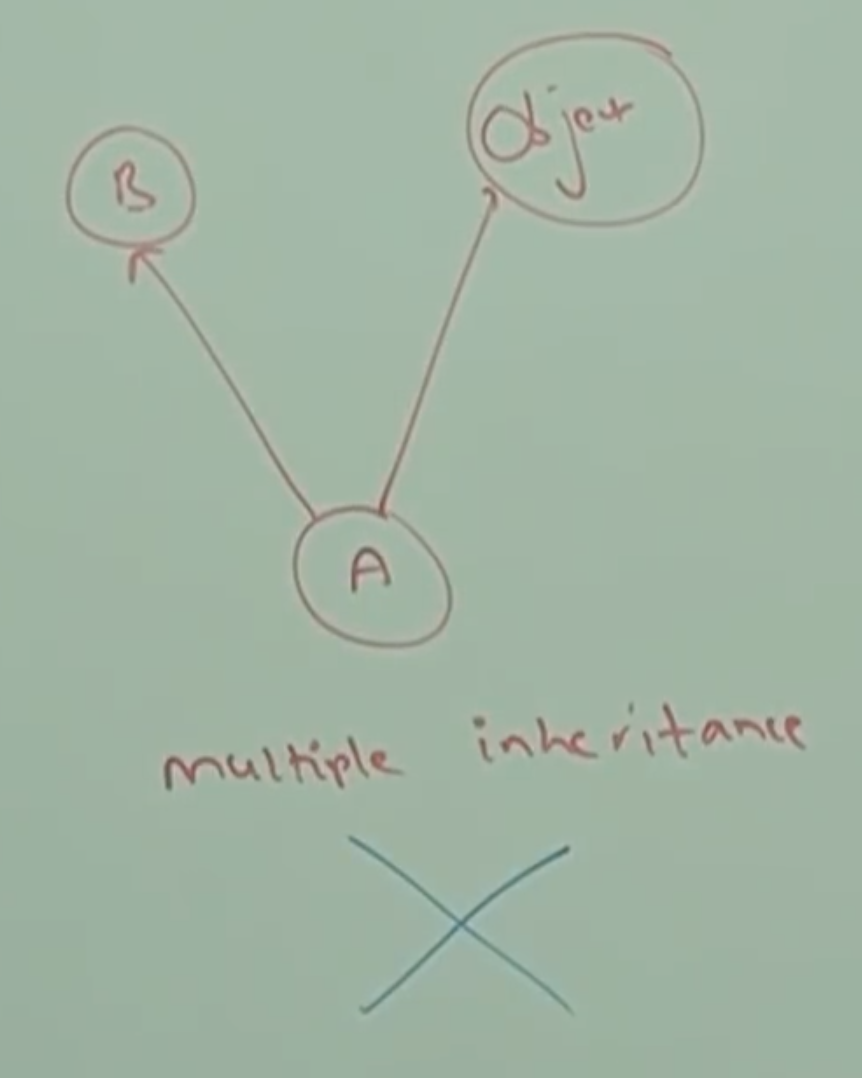
}

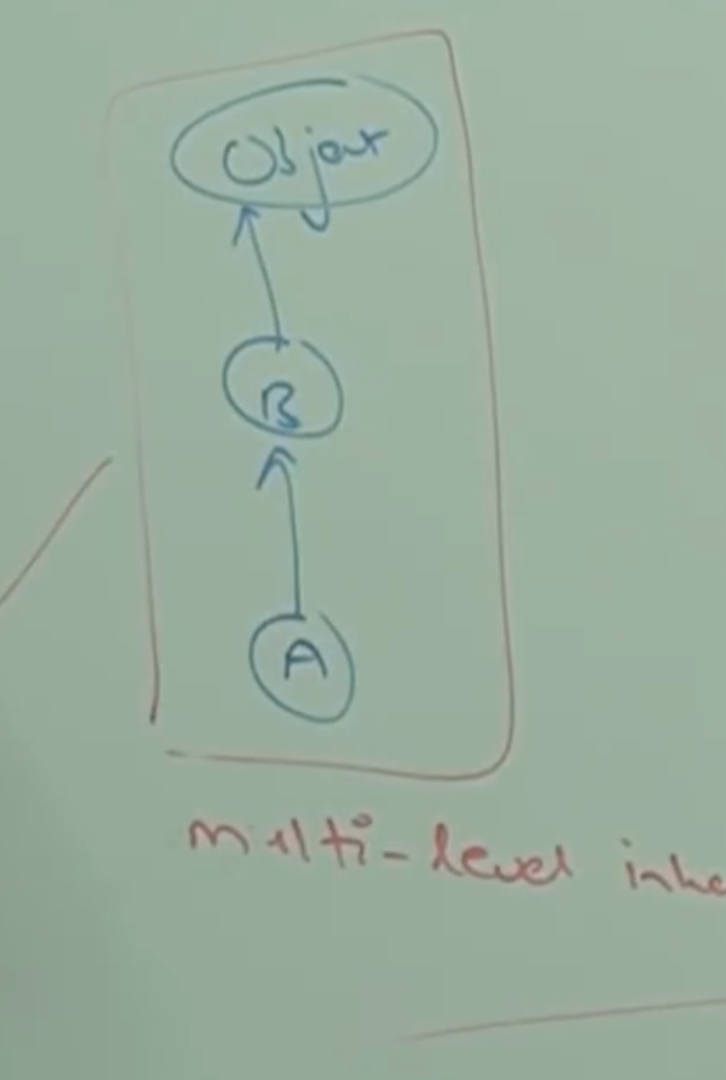
Object

1. If our class extends any other class, then our class is indirect child class of Object.

class A extends B{

}





Conclusion:

Either directly or indirectly java won’t provide support for multiple inheritance with respect to classes.

* **Object class methods:**

Object class defines the following 11 methods:

public String toString()

public native int hashCode()

public boolean equals(Object o)

protected native Object clone() throws CloneNotSupportedException

protected void finalize() throws Throwable

protected final Class getClass()

public final void wait() throws InterruptedException

public final native void wait(long ms) throws InterruptedException

public final void wait(long ms, int ns) throws InterruptedException

public native final void notify()

public native final void notifyAll();

Note:

Strictly speaking Object class contains 12 methods that extra method is “registerNatives”.

private static native void registerNatives();

This method is internally required for Object class and not available to the child classes. Hence, we are not required to consider this method.

Example:

import java.lang.reflect.\*;

class Test{

public static void main(String[] args) throws Exception{

Class class = Class.forName(“java.lang.Object”);

Method[] methods = class.getDeclaredMethod();

for(Method m: methods){

System.out.println(m.getName());

}

}

}

* **toString()**

We can use toString() method to get string representation of an object.

String s = obj.toString();

Whenever we are trying to print object reference internally toString() method will be called.

System.ou.println(s) 🡪 System.out.println(s.toString());

Example:

Student s = new Student();

System.out.println(s) 🡪 System.out.println(s.toString());

If our class doesn’t contain toString() then Object class toString() method will be executed.

Example:

class Student{

String name;

int rollNo;

Student(String name, int rollNo){

this.name = name;

this.name = name;

}

public static void main(String[] args){

Student s1 = new Student(“Saravana”, 101);

Student s2 = new Student(“Ravi”, 102);

System.out.println(s1); //Student@1888759

System.out.println(s1.toString()); //Student@1888759

System.out.println(s2.toString()); //Student@6e1408

}

}

In the above example Object class toString() method got executed which is implement as follows.

public String toString(){

return getClass().getName() + “@” + Integer.toHexString(hashCode());

}

Classname@hashCode in hexa decimal form

Based on our requirement we can override toString() method to provide our own string representation.

For example, whenever we are trying to print student object reference to print his name and rollno, we have to override toString() method as follows.

public String toString(){

return name+” …”+rollno;

}

pubic String toString(){

return “This is the student with name:”+name+”and rollno:”+rollno;

}