Introduction

For writing any java program the most commonly required package is javalang.

Example:

```
Class Test {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

In the previous example we depends on `javalang` to create:

- the class Test because every class inherits from class Object.
- String, because the String class exists in javalang.
- System, also exists in javalang.

For writing any java program whether it is simple or complex, the most commonly and interfaces are grouped into a separate package which javalang.

We are not required to import javalang explicity because all class and interfaces are available by default to every java program.

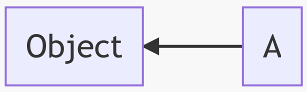
javalang.Object:

- The most commonly required methods (whether they are predefined or customer classes) are defined in a separate class which is nothing but `Object` class.
- Every class in java is a `child` class of object either directly or indirectly, so that *object* class methods available to every java class.
- Object class is considered as **root** for all java class.

NOTE:

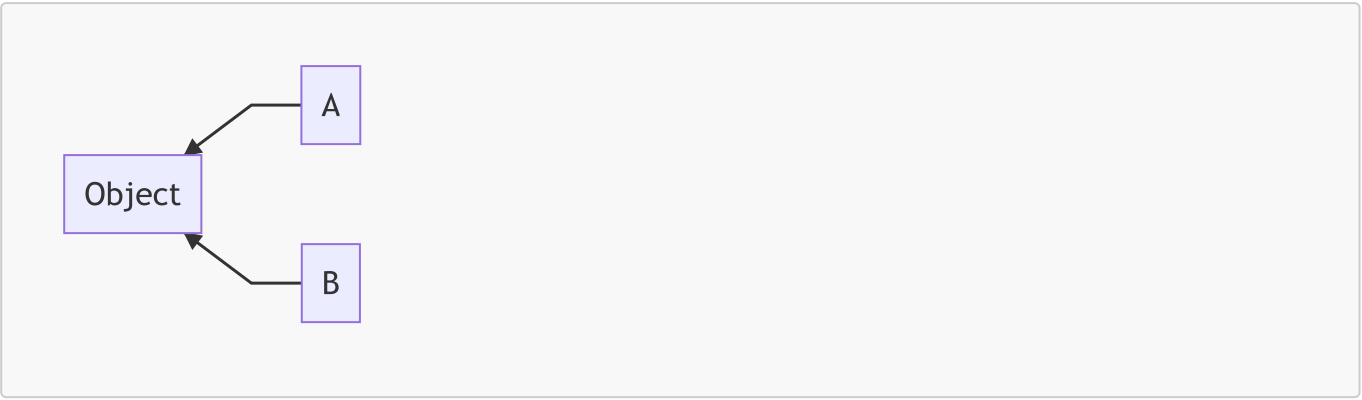
- If our class doesn't extend any class then only our class is a direct child of *Object*.
- **Example:**

```
Class A {

}
```

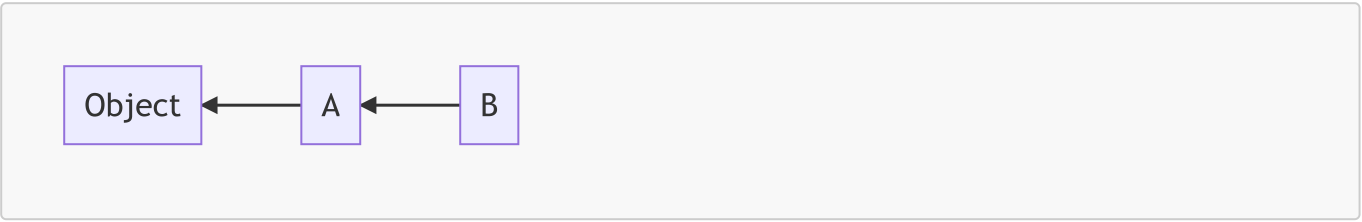


- A is the child class of `Object`.
- If our class extends any other class, then our class **indirect** child of `Object`.

```
Class A extends B {

}
```



This is called *multiple inheritence* which is **ILLEGAL**.



This is called *multi-level inheritance* which is **LEGAL**. **Conclusion:**

- either directly or indirectly, java doensn't provide support multi inheritence with respect to class.

Object class methods:

- Oject class defines the following 11 methods:
    1. `public String toString()`
    2. `public native int hashCode()`
    3. `public boolean equals(Object o)`
    4. `protected native Object clone() throws CloneNotSupportedException`
    5. `protected void finalize() throws Throwable`
    6. `public final Class getClass()`
    7. `public final void wait() throws InterruptedException`
    8. `public final native void wait(long ms) throws InterruptedException`
    9. `public final void wait(long ms, int ns) throws InterruptedException`
    10. `public native final void notify()`
    11. `public native final void notifyAll()`
- To check the methods in a given class we can use:

```
import java.lang.reflect.*
class Test {
    public static void main(String[] args) throws Exception {
        int count = 0;
        Class c = Class.forName("java.lang.Object");
        Method[] m = c.getDeclaredMethods();
        for (Method m1 : m) {
            System.out.println(m1.getName());
        }
        System.out.println("The number of methods:" + count);
    }
}
```

toString():

- We can use `toString()` method to get a string representation of an object, `String s = obj.toString()`.
- Whenever we are trying to print object reference, internally `toString()` method will be called.
- *Example:*
  `Student s = new Student()`
  `System.out.println(s); <=> System.out.println(s.toString())`
- If our class doesn't contain `toString()` method, then *Object* class `toString()` method will be executed.

```
class Student {
    String name;
    int rollno,
    Student (String name, int rollno) {
        this.name = name;
        this.rollno = rollno;
    }
    public static void main(String[] args) {
        Student s1 = new Student ("Ahmed", 101);
        Student s2 = new Student ("John", 102);

        System.out.println(s1);
        System.out.println(s1.toString());
        System.out.println(s2);
    }
}
```

output:

- Student@1888759
- Student@1888759
- Student@6e1408

In the previous example, Object class toString() method got executed, which is implemented as follows:

```
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

classname@hashCode_in_hexadecimal_form

Overriding toString():

- based on our requirements we can *override* `toString()` method to provide our own representation.

```
@Override
public String toString() {
    return "Student name: " + name + "\nStudent rollno: " + rollno;
}
```

Wrapper classes:

- In all wrapper classes and collection classes, string class, string buffer and builder class `toString()` is orriden for meaningful string representation. Hence, it's highly recommended to override `toString()` in our class as well.

```
import java.util.ArrayList
class Test {
    public String toString() {
        return "This is a test";
    }

    public static void main(String[] args) {
        String s = new String("Ahmed");
        System.out.println(s); => ahmed

        Integer i = new Integer(10);
        System.out.println(i); => 10

        Arraylist l = new Arraylist();
        l.add("A");
        l.add("B");
        System.out.println(l); => [A, B]

        Test t = new Test();
        System.out.println(t); => This is a test
    }
}
```