Do you know **Collections** class in Collection API ?

If don’t, let me tell you that **Collections** class is a utility class for Collection API which has lots of utility method.

And the interesting thing about **Collections** class, every method of this class is static in nature.

So the point comes out is that it is good practice when we creates utility classes so every method should be static in nature.

**Utility classes should only contain static methods !**

**BEFORE JAVA 8**

Before java 8, java developers used to create a different utility class for their API. You can also observe that, we have **Collection** **interface** (or Collection API) and on the other hand we have **Collections** class for utility methods.

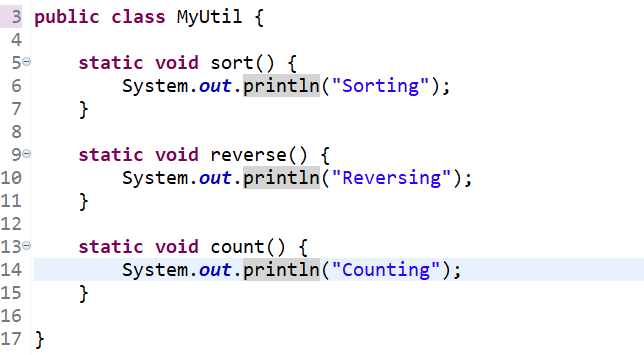
Java developers noticed few drawback in this approach !

**Drawback 1 :** We are separating utility methods from our API.

**Drawback 2 :** We are introducing one more class in our API whose whole purpose is to provide utility methods.

**Drawback 3 :**

To understand 3rd drawback, look at the below class



There is a point about above class for which we need to care about i.e. all the methods we are creating in this class are static in nature.

If a class only containing static members that means we won’t never going to create object of this class to access members.

When we create a class, JVM adds few default stuff in class i.e. default constructor, extending Object class, importing lang package etc.

Text

Description automatically generated

But as we know if a class only have static members in it so there no need of this default settings.

Still by putting all such methods in a class we are increasing JVM overhead.

**Drawback 1 Solution :** Now instead of separating utility methods from our API, we can directly put all those utility methods in our API.

If this feature exists before, then all the methods of **Collections** class is present in the **Collection** interface.

**Drawback 2 Solution :** Now we don’t have any separate utility class.

**Drawback 3 Solution**

Solution for all three drawbacks is **interface static methods**.

Java developers come up with a solution that if a class is never going to instantiate and only going to be call using class name then instead of creating a class why don’t we create a interface because interface don’t instantiate.

Now question arises is ? Why inside interface ? What wrong with class ?

So the answer is simple, Classes are heavy weight entity in compare with interfaces. It is because in case of classes, JVM manages lots of thing and JVM take many step before creating Object of class.

But when It comes to interface, JVM no needs to worry about all these things at all.

By Java 8, Java developers started supporting static methods inside interface and said that if your class is containing only static methods then it is better to have an interface rather than a class.

**Syntax**

Syntax is similar as we declare static method in class. Only thing to remember is we don’t need to specify any access modifier because in interface by default every methods are public in nature.

**Text

Description automatically generated**

**Access**

There is only one way to access interface static methods.

A picture containing text

Description automatically generated

**Point to remember :**

1. We can access class static methods in three different way :

* We can directly access static member of parent class in child class only by specifying method name.
* By creating object of class and we can call the static method using object.
* We can call static method using class name also.

1. We can only access interface static methods only one way :
   * Using interface name