Understanding static:

When a member is declared static, it can be accessed before any objects of its class are created,

and without reference to any object. You can declare both methods and variables to be static.

The most common example of a static member is main( ).

main( ) is declared as static because it must be called before any objects exist.

Static method in Java is a method which belongs to the class and not to the object.

A static method can access only static data. It cannot access non-static data (instance variables)

A non-static member belongs to an instance. It's meaningless without somehow resolving which instance of a class you

are talking about. In a static context, you don't have an instance, that's why you can't access a non-static member

without explicitly mentioning an object reference.

In fact, you can access a non-static member in a static context by specifying the object reference explicitly :

```
public class Human {

   String message = "Hello World";

   public static void display(Human human){
      System.out.println(human.message);
   }

   public static void main(String[] args) {
      Human kunal = new Human();
      kunal.message = "Kunal's message";
      Human.display(kunal);
   }

}
```

A static method can call only other static methods and cannot call a non-static method from it.

A static method can be accessed directly by the class name and doesn’t need any object

A static method cannot refer to "this" or "super" keywords in anyway

If you need to do computation in order to initialize your static variables,

you can declare a static block that gets executed exactly once, when the class is first loaded.

```
// Demonstrate static variables, methods, and blocks.
    class UseStatic {
      static int a = 3;
      static int b;
      static void meth(int x) {
        System.out.println("x = " + x);
        System.out.println("a = " + a);
        System.out.println("b = " + b);
      }
      static {
        System.out.println("Static block initialized.");
        b = a * 4;
       }
      public static void main(String args[]) {
        meth(42);
   }
}
```

As soon as the UseStatic class is loaded, all of the static statements are run. First, a is set to 3,

then the static block executes, which prints a message and then initializes b to a*4 or 12. Then main( ) is called,

which calls meth( ), passing 42 to x. The three println( ) statements refer to the two static variables a and b,

as well as to the local variable x.

Here is the output of the program:

Static block initialized. x = 42

a = 3

b = 12

Note: main method is static, since it must be accessible for an application to run, before any instantiation takes place.

NOTE: Only nested classes can be static.

NOTE: Static inner classes can have static variables

You cant override the inherited static methods, as in java overriding takes place by resolving the type of object at

run-time and not compile time, and then calling the respective method.

Static methods are class level methods, so it is always resolved during compile time.

Static INTERFACE METHODS are not inherited by either an implementing class or a sub-interface.

NOTE:

```
public class Static {

   // class Test  // ERROR
   static class Test{
      String name;

      public Test(String name) {
         this.name = name;
      }
   }

   public static void main(String[] args) {
      Test a = new Test("Kunal");

      Test b = new Test("Rahul");
```

```
        System.out.println(a.name); // Kunal
        System.out.println(b.name); // Rahul
    }
}
```

Because :

The static keyword may modify the declaration of a member type C within the body of a non-inner class or interface T.

Its effect is to declare that C is not an inner class. Just as a static method of T has no current instance of T in its

body, C also has no current instance of T, nor does it have any lexically enclosing instances.

Here, test does not have any instance of it's outer class Static. Neither does main.

But main & Test can have instances of each other.