This is demonstrating the variables that are accessed from inner and outer class

```
public class OuterClass {
```

```
    private static String name ="amit";
    private String nameAgain ="amit1";

    public class InnerClass
    {
        private static final String name1 = "amit2";
        private String nameAgain1="amit3";

        public void test()
        {
            System.out.println(OuterClass.name); // to access a static member, this is ok
            System.out.println(OuterClass.this.nameAgain); // to access a non-static member

            System.out.println(this.name1); //access a non-static member of the inner class
            System.out.println(nameAgain1); //access a static member of the inner class
```

```
        }
    }

    public static void main(String[] args) {
        System.out.println("Welcome to Online IDE!! Happy Coding :)");

        OuterClass.InnerClass ic = new OuterClass().new InnerClass();
        ic.test();
    }
}
```

We can see that we are able to readily drop the this keyword for System.out.println for bullet point 3... This makes perfect sense given that name1 is visible within the method... The scope of private variable is ok.

The difficulty in understanding is system output for bullet point 2 as described below.#

```
public class OuterClass {
```

```
    private static String name ="amit";
    private String nameAgain ="amit1";

    public class InnerClass
    {
        private static final String name1 = "amit2";
        private String nameAgain1="amit3";

        public void test()
        {
            System.out.println(OuterClass.name); // to access a static member, this is ok
            System.out.println(OuterClass.this.nameAgain); // to access a non-static member

            System.out.println(name1); //access a non-static member of the inner class
            System.out.println(nameAgain1); //access a static member of the inner class
```

I am examining this from perspective of accessing variables. There is no shadowing at ALL

IT IS QUITE DIFFICULT TO UNDERSTAND WHY this KEYWORD IS REQUIRED. IT APPEARS IT IS A WORKAROUND TO CREATING AN INSTANCE OF THE OUTERCLASS AND ACCESSING A NON-STATIC VARIABLE?

We can see it clearly accesses and no errors in execution..

```
Welcome to Online IDE!! Happy Coding :)
amit
amit1
amit2
amit3


** Process exited - Return Code: 0 **
```

OUTPUT

I know if I remove the this keyword, it will cause error in execution since nameAgain is not a class level variable

```
public class OuterClass {

    private static String name ="amit";
    private String nameAgain ="amit1";

    public class InnerClass
    {
        private static final String name1 = "amit2";
        private String nameAgain1="amit3";

        public void test()
        {
            System.out.println(OuterClass.name); // to access a static member, this is ok
            System.out.println(OuterClass.nameAgain); // to access a non-static member

            System.out.println(name1); //access a non-static member of the inner class
            System.out.println(nameAgain1); //access a static member of the inner class
```

I am examining this from perspective of accessing variables. There is no shadowing at ALL

I am removing the this keyword and I would expect an error to occur

The error is as expected:

```
OuterClass.java:23: error: non-static variable nameAgain cannot be referenced from a static context
            System.out.println(OuterClass.nameAgain); // to access a non-static member
                                          ^
1 error


** Process exited - Return Code: 1 **
```

OUTPUT. I WOULD NORMALLY DECLARE THE VARIABLE nameAgain AS STATIC FOR A WORKAROUND.

OR CREATE AN INSTANCE VARIABLE OF TYPE OuterClass. And reach member field

```
OuterClass oc= new OuterClass();
System.out.println(oc.nameAgain); // to access a non-static member
```

So the overall confusion is as to whether OuterClass • this • nonClassLevelVariable

(use of no constructor, difficult to understand hierarchy)

is equivalent to creating instance (initialisation via new keyword which I am familiar with) of the OuterClass and accessing via:
OuterClass oc = new OuterClass();

oc.nameAgain (object reference • nonClassLevelVariable)