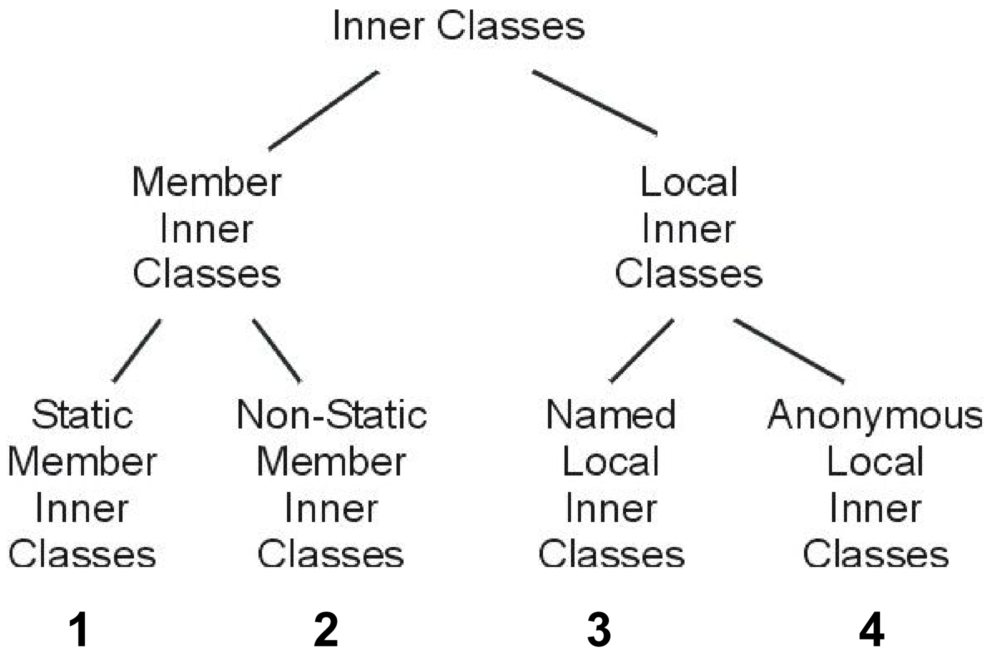
Inner Classes

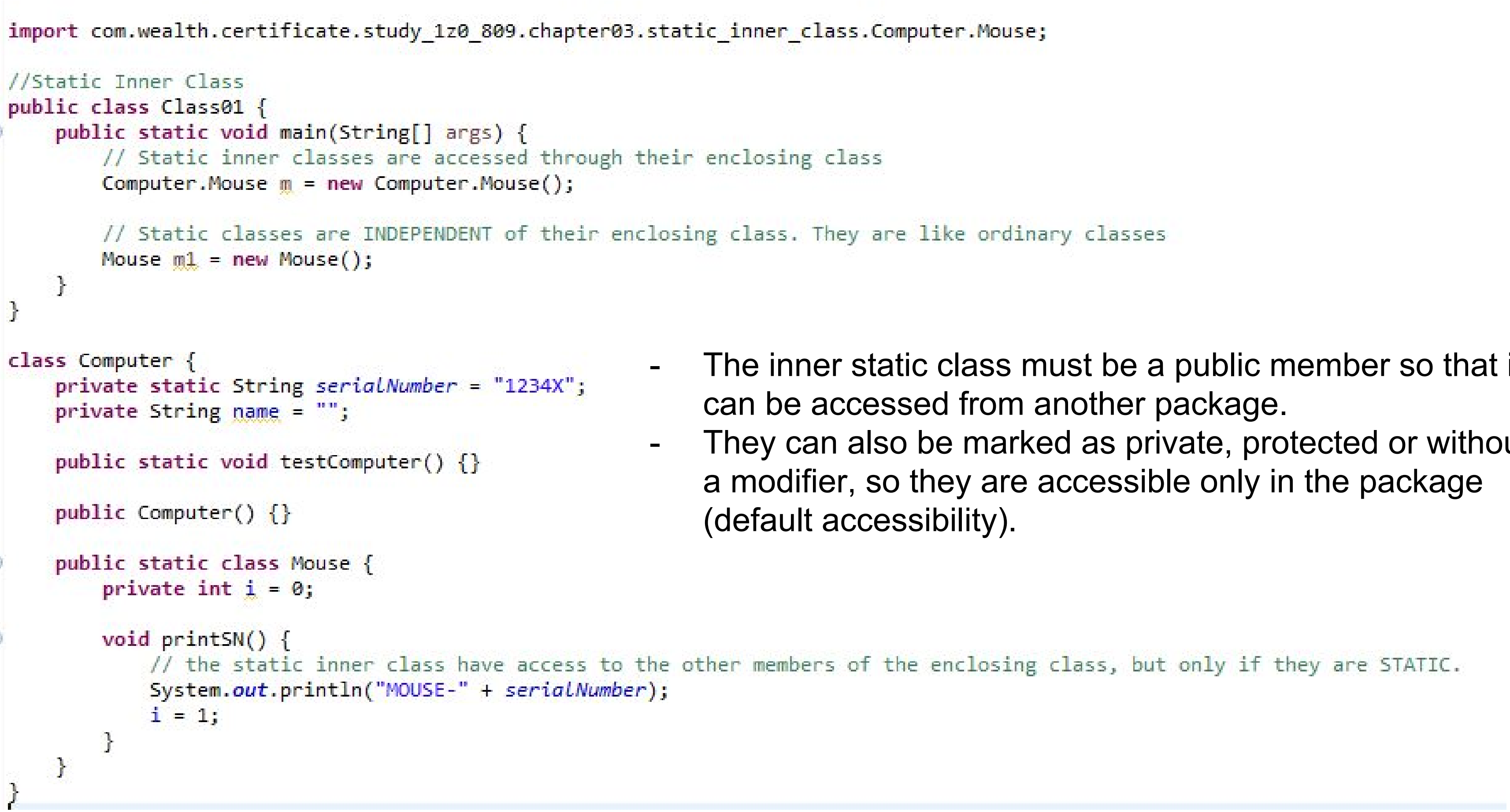
Chapter 3

Inner Classes

1. STATIC inner classes
2. NON-STATIC inner classes
3. LOCAL classes
4. ANONYMOUS classes


Inner Classes
Member Inner Classes
Local Inner Classes
Static Member Inner Classes
Non-Static Member Inner Classes
Named Local Inner Classes
Anonymous Local Inner Classes
1
2
3
4

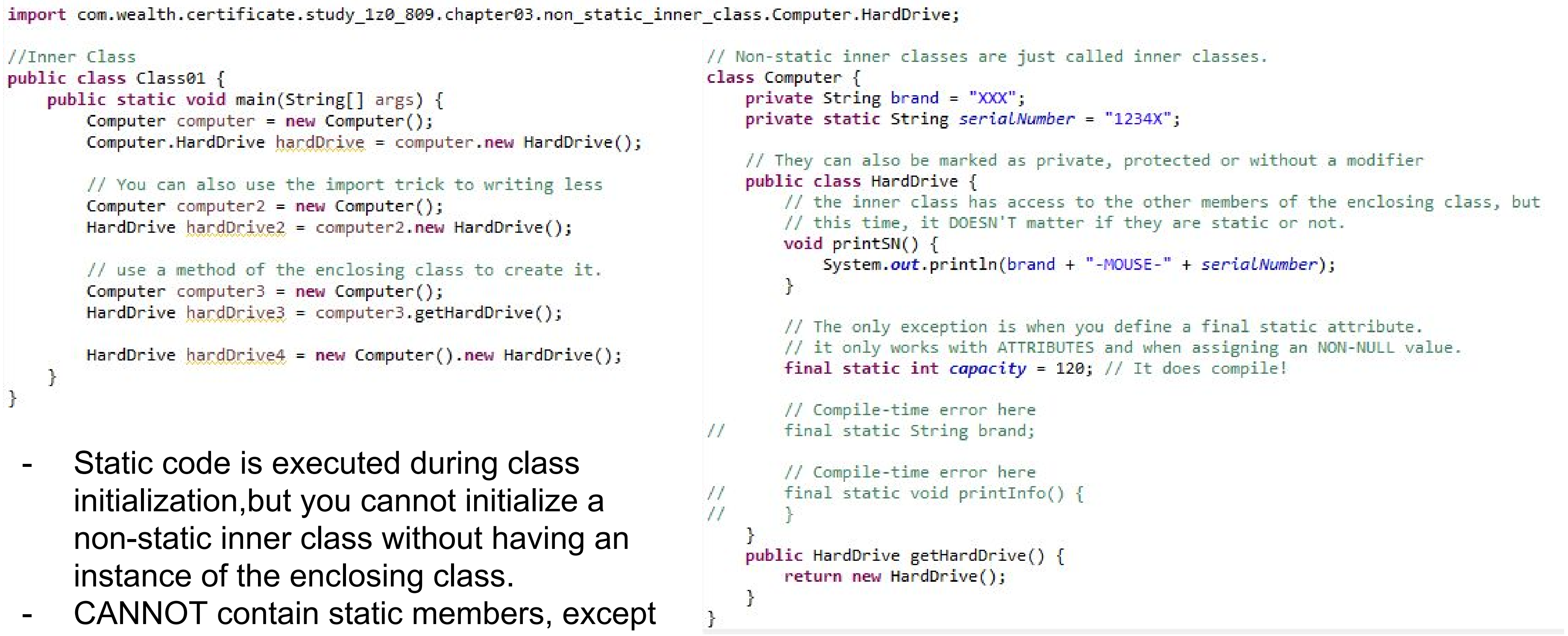
STATIC Inner Class



```
import com.wealth.certificate.study_1z0_809.chapter03.static_inner_class.Computer.Mouse;

//Static Inner Class
public class Class01 {
    public static void main(String[] args) {
        // Static inner classes are accessed through their enclosing class
        Computer.Mouse m = new Computer.Mouse();

        // Static classes are INDEPENDENT of their enclosing class. They are like ordinary classes
        Mouse m1 = new Mouse();
    }
}

class Computer {
    private static String serialNumber = "1234X";
    private String name = "";

    public static void testComputer() {}

    public Computer() {}

    public static class Mouse {
        private int i = 0;

        void printSN() {
            // the static inner class have access to the other members of the enclosing class, but only if they are STATIC.
            System.out.println("MOUSE-" + serialNumber);
            i = 1;
        }
    }
}
```

- The inner static class must be a public member so that it can be accessed from another package.
- They can also be marked as private, protected or without a modifier, so they are accessible only in the package (default accessibility).

NON-STATIC Inner Class = Inner Class

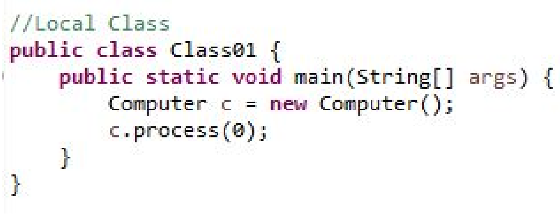


```java
import com.wealth.certificate.study_1z0_809.chapter03.non_static_inner_class.Computer.HardDrive;

//Inner Class
public class Class01 {
    public static void main(String[] args) {
        Computer computer = new Computer();
        Computer.HardDrive hardDrive = computer.new HardDrive();

        // You can also use the import trick to writing less
        Computer computer2 = new Computer();
        HardDrive hardDrive2 = computer2.new HardDrive();

        // use a method of the enclosing class to create it.
        Computer computer3 = new Computer();
        HardDrive hardDrive3 = computer3.getHardDrive();

        HardDrive hardDrive4 = new Computer().new HardDrive();
    }
}
```

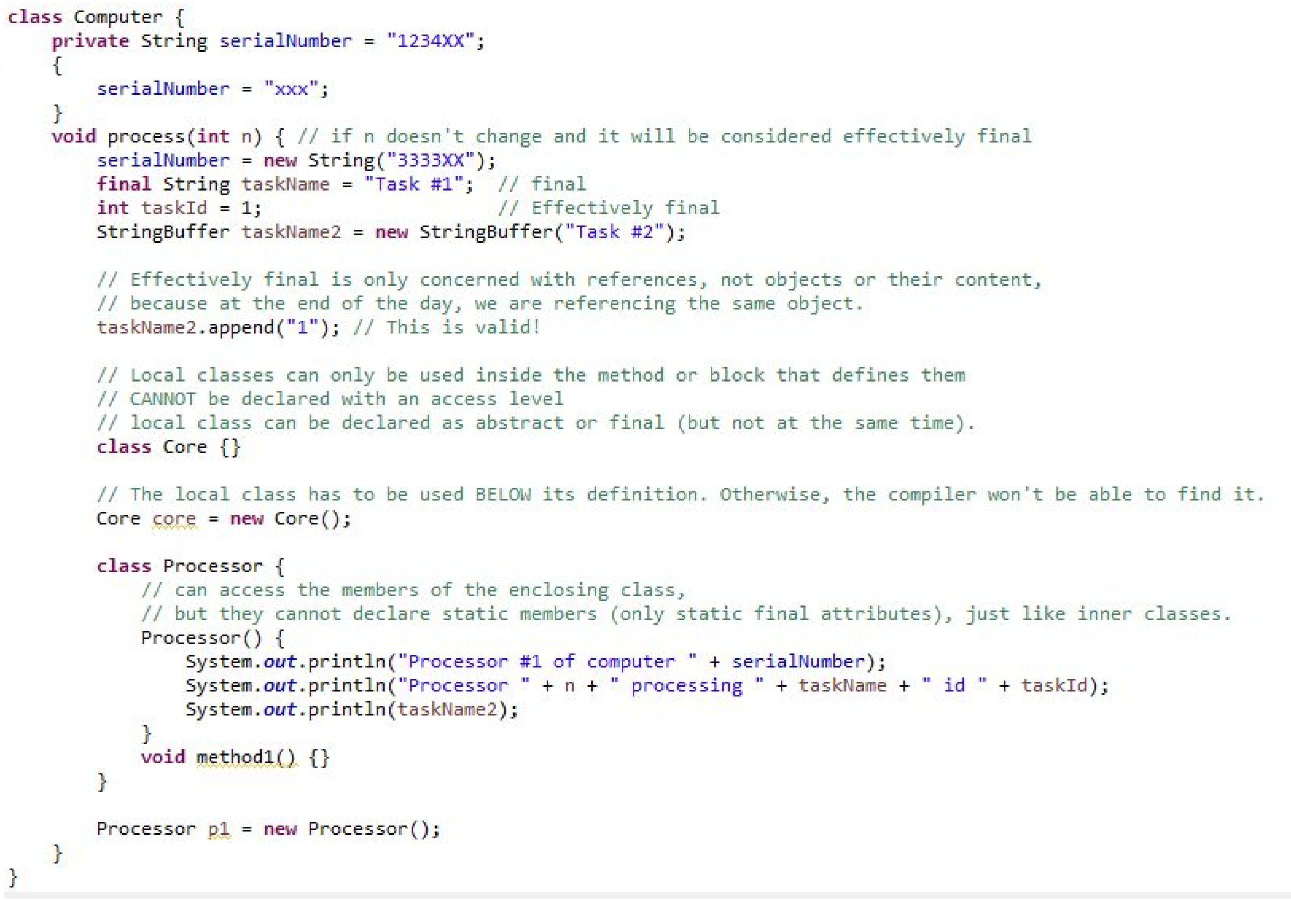
```java
// Non-static inner classes are just called inner classes.
class Computer {
    private String brand = "XXX";
    private static String serialNumber = "1234X";

    // They can also be marked as private, protected or without a modifier
    public class HardDrive {
        // the inner class has access to the other members of the enclosing class, but
        // this time, it DOESN'T matter if they are static or not.
        void printSN() {
            System.out.println(brand + "-MOUSE-" + serialNumber);
        }

        // The only exception is when you define a final static attribute.
        // it only works with ATTRIBUTES and when assigning an NON-NULL value.
        final static int capacity = 120; // It does compile!

        // Compile-time error here
//      final static String brand;

        // Compile-time error here
//      final static void printInfo() {
//      }
    }
    public HardDrive getHardDrive() {
        return new HardDrive();
    }
}
```

- Static code is executed during class initialization,but you cannot initialize a non-static inner class without having an instance of the enclosing class.
- CANNOT contain static members, except final static attribute

LOCAL Class

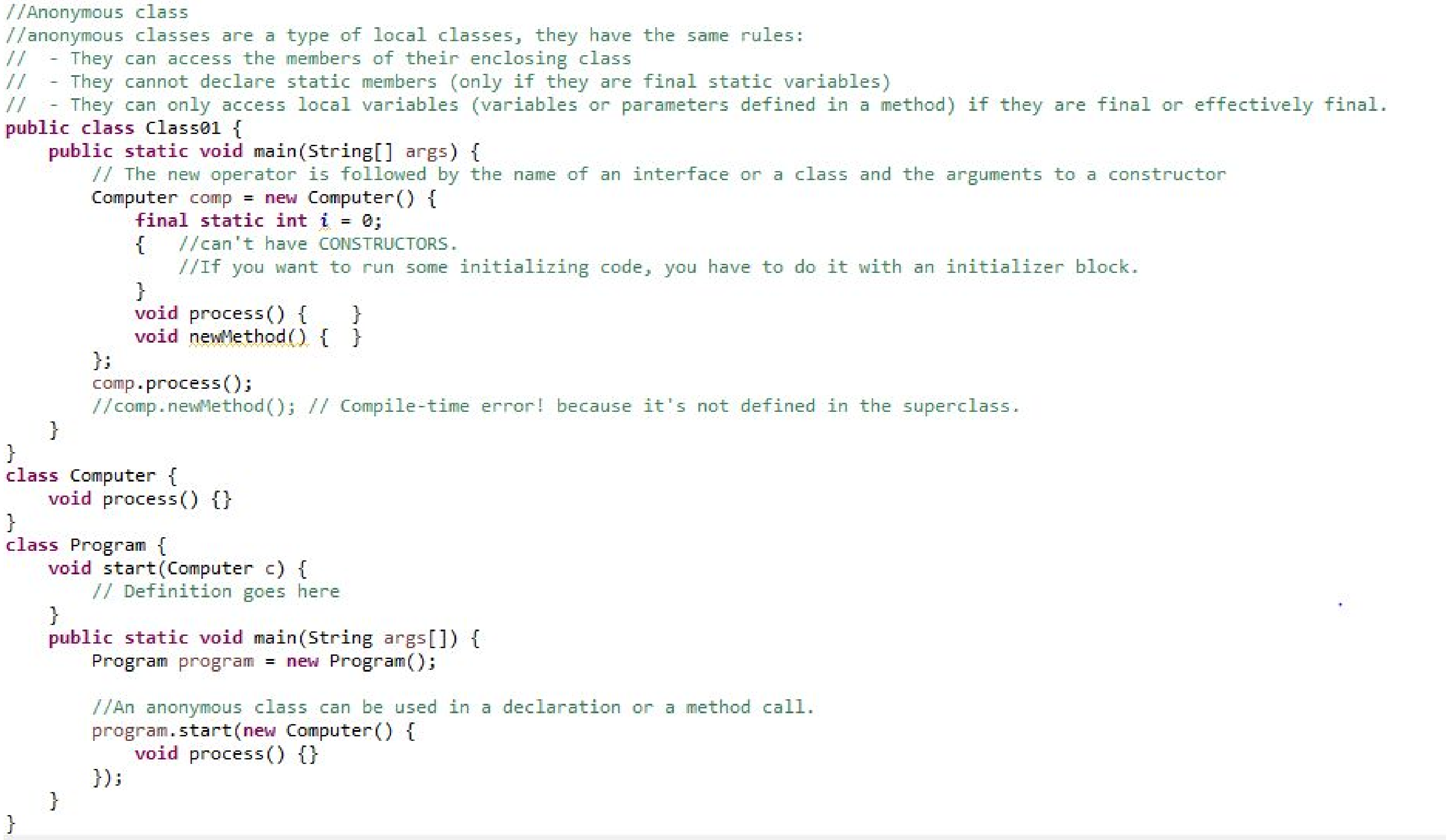


```java
//Local Class
public class Class01 {
    public static void main(String[] args) {
        Computer c = new Computer();
        c.process(0);
    }
}
```



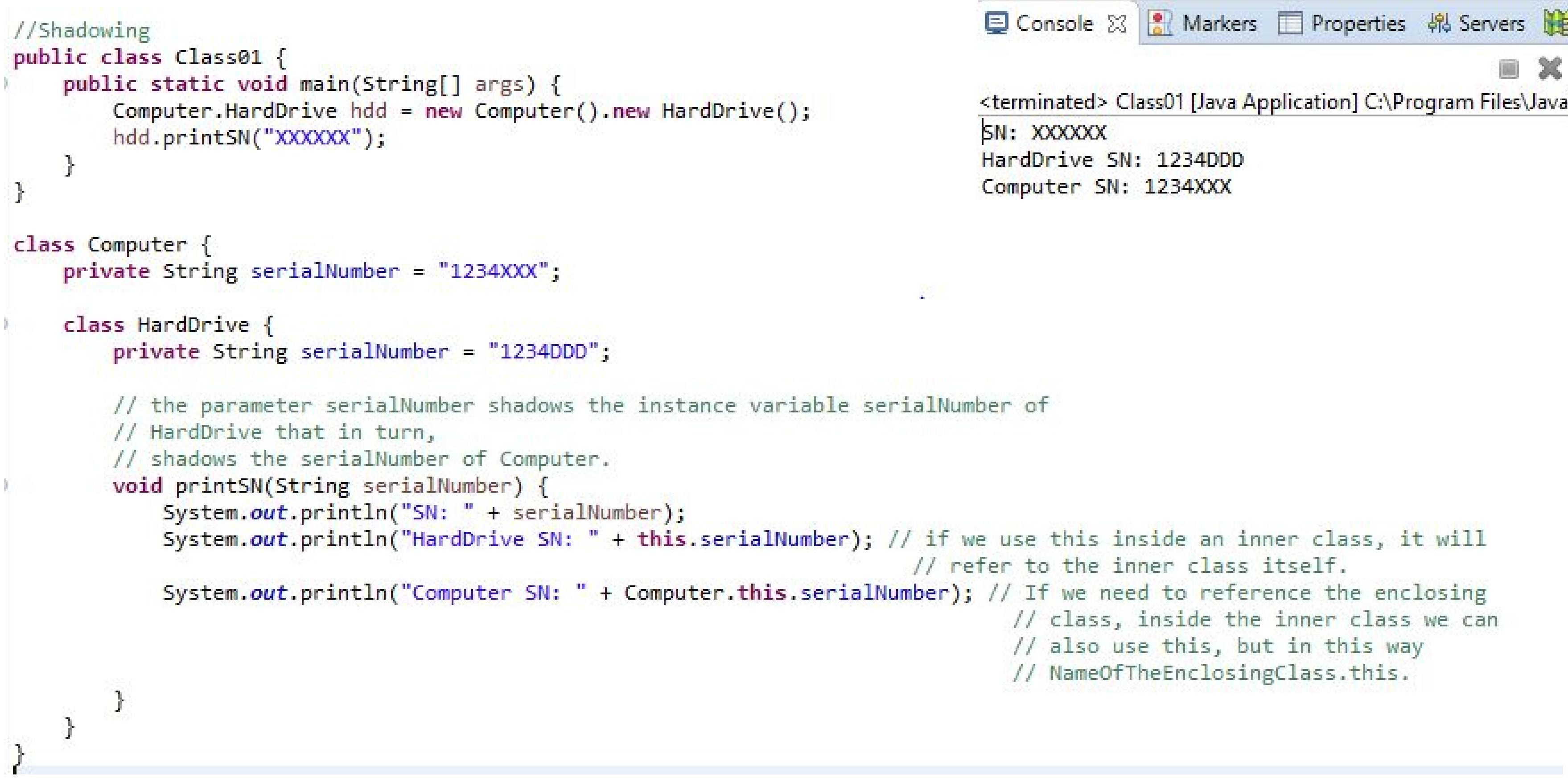
```java
class Computer {
    private String serialNumber = "1234XX";
    {
        serialNumber = "xxx";
    }
    void process(int n) { // if n doesn't change and it will be considered effectively final
        serialNumber = new String("3333XX");
        final String taskName = "Task #1";  // final
        int taskId = 1;                     // Effectively final
        StringBuffer taskName2 = new StringBuffer("Task #2");

        // Effectively final is only concerned with references, not objects or their content,
        // because at the end of the day, we are referencing the same object.
        taskName2.append("1"); // This is valid!

        // Local classes can only be used inside the method or block that defines them
        // CANNOT be declared with an access level
        // local class can be declared as abstract or final (but not at the same time).
        class Core {}

        // The local class has to be used BELOW its definition. Otherwise, the compiler won't be able to find it.
        Core core = new Core();

        class Processor {
            // can access the members of the enclosing class,
            // but they cannot declare static members (only static final attributes), just like inner classes.
            Processor() {
                System.out.println("Processor #1 of computer " + serialNumber);
                System.out.println("Processor " + n + " processing " + taskName + " id " + taskId);
                System.out.println(taskName2);
            }
            void method1() {}
        }

        Processor p1 = new Processor();
    }
}
```

ANONYMOUS Class

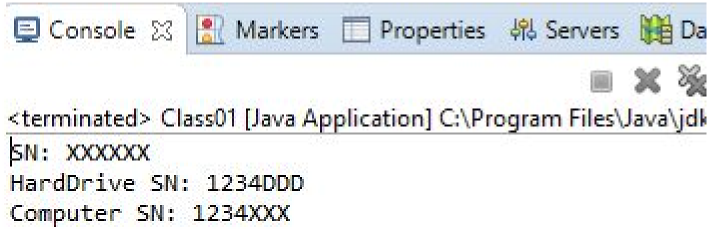


```java
//Anonymous class
//anonymous classes are a type of local classes, they have the same rules:
//  - They can access the members of their enclosing class
//  - They cannot declare static members (only if they are final static variables)
//  - They can only access local variables (variables or parameters defined in a method) if they are final or effectively final.
public class Class01 {
    public static void main(String[] args) {
        // The new operator is followed by the name of an interface or a class and the arguments to a constructor
        Computer comp = new Computer() {
            final static int i = 0;
            {   //can't have CONSTRUCTORS.
                //If you want to run some initializing code, you have to do it with an initializer block.
            }
            void process() {    }
            void newMethod() {  }
        };
        comp.process();
        //comp.newMethod(); // Compile-time error! because it's not defined in the superclass.
    }
}
class Computer {
    void process() {}
}
class Program {
    void start(Computer c) {
        // Definition goes here
    }
    public static void main(String args[]) {
        Program program = new Program();

        //An anonymous class can be used in a declaration or a method call.
        program.start(new Computer() {
            void process() {}
        });
    }
}
```

Shadowing



```
//Shadowing
public class Class01 {
    public static void main(String[] args) {
        Computer.HardDrive hdd = new Computer().new HardDrive();
        hdd.printSN("XXXXXX");
    }
}

class Computer {
    private String serialNumber = "1234XXX";

    class HardDrive {
        private String serialNumber = "1234DDD";

        // the parameter serialNumber shadows the instance variable serialNumber of
        // HardDrive that in turn,
        // shadows the serialNumber of Computer.
        void printSN(String serialNumber) {
            System.out.println("SN: " + serialNumber);
            System.out.println("HardDrive SN: " + this.serialNumber); // if we use this inside an inner class, it will
                                                                       // refer to the inner class itself.
            System.out.println("Computer SN: " + Computer.this.serialNumber); // If we need to reference the enclosing
                                                                               // class, inside the inner class we can
                                                                               // also use this, but in this way
                                                                               // NameOfTheEnclosingClass.this.

        }
    }
}
```



```
<terminated> Class01 [Java Application] C:\Program Files\Java\jdk
SN: XXXXXX
HardDrive SN: 1234DDD
Computer SN: 1234XXX
```