9. Lab I – State

9.1. Purpose

We wish to implement a network device capable of operating under various states. The network device will contain a state object and delegate client requests to it.

To support a flexible application, an interface is used as a base for all the state objects – `INetworkDevice` We will implement 3 states extending the `AbstractDeviceState` `(implementing INetworkDevice)` as different states: `EnabledState`, `DisabledState`, `SuspendedState` and a `NetworkDevice` implementation class.

Each State object can behave as a different state.

Lab Scenario (<u>WHAT</u> to do)	**Examine the supplied code :** `INetworkDevice` interface defines the basic functionality for a network device machine. (enable, disable, transmit, receive, suspend, resume) **Implement 3 State objects representing:** Enabled, Suspended, Disabled states , each should extend `AbstractDeviceState` and provide partial functionality so that: • A suspended network device will only support resume and disable • A disabled device will only support enable • An enabled device will support disable, receive, transmit and suspend **Implement an `NetworkDevice` class.** Implement the state transition in the network device - define the state variable , constants and set method
Implementation Steps (<u>HOW</u> to do it)	**Code the AbstractDeviceState class** - each state should have a pointer to the network machine it belongs to - it implements `INetworkDevice` - it throws an unsupported operation exception for all methods because the states sub-class will override methods they are interested in implementing. - Code EnabledState - extend AbstractDeviceState - implement only the relevant methods - i.e an enabled state cant be enabled but it can be disabled Code DisabledState - extend AbstractDeviceState - implement only the relevant methods - i.e a disabled state cant transmit but it can be enabled Code SuspendedState - extend AbstractDeviceState - implement only the relevant methods , i.e a suspended state cant transmit but it can be resumed

	Code NetworkDevice - define a state member variable - implement INetworkDevice all inherited methods should be delegated to the `AbstractDeviceState` member variable - define a setState method define constants reflecting the possible states

9.2. Review of State

A single interface is supplied :

1. `INetworkDevice` interface serving as base for concrete states

Five classes to create

1. AbstractNetworkState implements INetworkState and its methods. All methods will throw an operation not supported exception.

2. EnabledState implements AbstractNetworkState and its methods (only ones relevant to an enabled state)

3. DisabledState implements AbstractNetworkState and its methods (only ones relevant to a disabled state)

4. SuspendedState implements AbstractNetworkState and its methods (only ones relevant to a suspended state)

5. `NetworkDevice` will be the manager and state machine. It will hold an individual state object (extending AbstractDeviceState) and delegate functionality to it

9.2.1. Create a new class – AbstractDeviceState

1. create the class `AbstractDeviceState`
2. create a `protected NetworkDevice networkDevice` member variable representing a reference to the embedding state machine for future callbacks
3. create the constructor taking a NetworkDevice parameter and saving it
4. implement all the methods inherited from `INetworkDevice` as throwing an **UnsupportedOperationException.** Future state classes extending this class will override these methods and provide a more proper implementation for relevant methods

9.2.2. Create a new class – EnabledState

This class represents an enabled state and so will only override 4 methods
disable, suspend, transmit and receive. Enable and resume should not be implemented.

1. extend AbstractDeviceState
2. write the constructor - receives a `NetworkDevice` and passes to super()
3. override disable()

```
networkDevice.setState(networkDevice.DISABLED_STATE);
```

4. override suspend()

```
networkDevice.setState(networkDevice.SUSPENDED_STATE);
```

5. override transmit()

```
System.out.println("Transmitting");
```

6. override receive()

```
System.out.println("receiving");
```

9.2.3. Create a new class – DisabledState

This class represents a disabled state and so will only override 1 method
enable. Other methods inherited from AbstractDeviceState should not be overridden

1. extend AbstractDeviceState
2. write the constructor - receives a `NetworkDevice` and passes to super()
3. override enable()

```
networkDevice.setState(networkDevice.ENABLED_STATE);
```

9.2.4. Create a new class – SuspendedState

This class represents a suspended state and so will only override 2 methods : Resume and disable. Other methods inherited from AbstractDeviceState should not be overridden

1. extend AbstractDeviceState
2. write the constructor - receives a `NetworkDevice` and passes to super()
3. override resume()

   ```
   networkDevice.setState(networkDevice.ENABLED_STATE);
   ```

4. override disable()

   ```
   networkDevice.setState(networkDevice.DISABLED_STATE);
   ```

9.2.5. Create a new class – NetworkDevice

This class represents the state manager. Each network device contains a state object and delegates method calls to it.

1. implement INetworkDevice
2. code a state member variable

   ```
   private AbstractDeviceState currentState;
   ```

3. code public constants reflecting possible states

   ```
   public final AbstractDeviceState DISABLED_STATE =
                                       new DisabledState(this);
   public final AbstractDeviceState ENABLED_STATE =
                                       new EnabledState(this);
   public final AbstractDeviceState SUSPENDED_STATE =
                                       new SuspendedState(this);
   ```

4. code the constructor and set a default initial state value

```
public NetworkDevice() {
    super();
    currentState = DISABLED_STATE;
}
```

5. delegate all methods inherited from INetworkDevice to the current state \

```
   public void enable() { currentState.enable(); }

   public void disable() { currentState.disable(); }

   public void transmit() {currentState.transmit();}
   public void receive() { currentState.receive();    }
   public void suspend() { currentState.suspend();    }
   public void resume() {currentState.resume();}
```

6. define a `setState(AbstractDeviceState newState)` method

```
  public void setState(AbstractDeviceState newState) {
      currentState = newState;
  }
```

7. define a main method testing the code

```
public static void main(String[] args) {
    try {
          NetworkDevice device = new NetworkDevice();
          device.enable();
          device.transmit();
          device.suspend();
          device.receive();
          device.resume();
          device.disable();
          System.out.println("Done");
    } catch (Exception e) {
          e.printStackTrace();
    }
}
```

Note: The solution for this exercise is available in the 'solutions' directory.