LAB – DEFINING CHILD CLASSES

OBJECTIVE

In this lab, you will define a base class for a generic network device, and will define child classes for specific device types: IOS and IOS-XR. The main functionality for this lab will be to implement connect and get_interfaces methods for your child classes.

PART 1

Copy the required configuration for this lab from flash into the running-configuration of the CSR1000v and R2 routers.

STEP 1: COPY CONFIGURATION

In the CSR1kv router console, enter **privileged exec** mode and copy the additional configuration located in the **CONFIG3** file to the running-configuration using the commands:

```
CSR1kv> enable
CSR1kv# copy flash:CONFIG3 running-config
```

```
CSR1kv>enable
CSR1kv#copy flash:CONFIG3 running-config
Destination filename [running-config]?
75 bytes copied in 0.009 secs (8333 bytes/sec)
CSR1kv#
```

STEP 2: VERIFY CONFIGURATION

Verify the new running-configuration by using the command:

```
CSR1kv# show run | section line vty
```

NOTE:
To get the pipe (**|**) command, you may need to use the keys **Shift** + **#** rather than your local keyboard input.

```
CSR1kv#show run | section line vty
line vty 0 4
 logging synchronous
 login local
 transport input all
line vty 5 15
 logging synchronous
 login local
 transport input all
CSR1kv#
```

STEP 3: COPY CONFIGURATION

In the R2 router console, enter **privileged exec** mode and copy the additional configuration located in the **CONFIG7** file to the running-configuration using the commands:

```
R2> enable
R2# copy flash:CONFIG7 running-config
```

```
R2>enable
R2#copy flash:CONFIG7 running-config
Destination filename [running-config]?
64 bytes copied in 0.024 secs (2667 bytes/sec)
R2#
```

STEP 4: VERIFY CONFIGURATION

Verify the new running-configuration by using the command:

```
R2# show run | section line vty
```

NOTE:
To get the pipe (**|**) command, you may need to use the keys **Shift** + **#** rather than your local keyboard input.

```
R2#show run | section line vty
line vty 0 4
 logging synchronous
 login local
 transport input all
line vty 5 15
 logging synchronous
 login local
 transport input all
R2#
```

PART 2

Open a terminal and switch to the lab directory

STEP 1: OPEN A TERMINAL WINDOW

Double-click the Terminal icon on the desktop to open the terminal window for use in this lab.

STEP 2: CHANGE DIRECTORY

Change to the directory **labs/prne/** in the user home directory, which holds the files for the course labs.

```
~$ cd labs/prne/
```

PART 3

Open **Visual Studio Code**, create a new file and save it with a filename of **defining-child-classes.py**, ensuring to save the file in the **~/labs/prne/** directory.
This python application reads the device list from the file **devices-13.txt**.

STEP 1: DEFINE BASE CLASS

Define a base, network device class with the usual attributes for name, IP, username, and password. Your base class will have an empty connect method that sets the session attribute to **None**, and a **get_interfaces** method that returns a string **Base device, cannot get interfaces**.

```
import pexpect

# Class to hold information about a generic network device
class NetworkDevice():

    def __init__(self, name, ip, user='cisco', pw='cisco123!'):
        self.name = name
        self.ip_address = ip
        self.username = user
        self.password = pw

        self.interfaces = ''

    def connect(self):
        self.session = None

    def get_interfaces(self):
        self.interfaces = '--- Base Device, unknown get interfaces ---'
```

STEP 2: DEFINE CHILD CLASSES

Define child device-specific classes for IOS and IOS-XR device types.
For your IOS class, your connect method will actually connect to the device, storing the session as an attribute. Your get_interfaces method will do run the show int brief command on the device, and will store the output.
For your IOS-XR class, you will create 'stub' methods for both connect and get_interfaces. The connect method will set the session to None, and the get_interfaces method will return a string saying "Getting interface information a different way", to simulate the behaviour of different object types satisfying requests in a device-specific manner.

```
# Class to hold information about an IOS-XE network device
class NetworkDeviceIOS(NetworkDevice):

    # Initialize
    def __init__(self, name, ip, user='cisco', pw='cisco123!'):
        NetworkDevice.__init__(self, name, ip, user, pw)

    # Connect to device
    def connect(self):

        print('--- connecting IOS: telnet ' + self.ip_address)

        self.session = pexpect.spawn('telnet ' + self.ip_address,
                                     encoding='utf-8', timeout=20)
        result = self.session.expect(['Username:', pexpect.TIMEOUT, pexpect.EOF])

        print('--- attempting to: username: ' + self.username)

        # Successfully got username prompt, logging with username
        self.session.sendline(self.username)

        # Check for failure
        if result != 0:
            print('--- Timeout or unexpected reply from device')
            return 0

        result = self.session.expect(['Password:', pexpect.TIMEOUT, pexpect.EOF])
        # Check for failure
        if result != 0:
            print('--- Timeout or unexpected reply from device')
            return 0

        # Successfully got password prompt, logging in with password
        self.session.sendline(self.password)
        self.session.expect(['>', pexpect.TIMEOUT, pexpect.EOF])

        print('--- setting terminal length to 0')
        self.session.sendline('terminal length 0')
        self.session.expect(['>', pexpect.TIMEOUT, pexpect.EOF])

    # Get interfaces from device
    def get_interfaces(self):

        self.session.sendline('show interfaces summary')
        result = self.session.expect(['>', pexpect.TIMEOUT, pexpect.EOF])

        self.interfaces = self.session.before
```

```
# Class to hold information about an IOS-XR network device
class NetworkDeviceXR(NetworkDevice):

    # Initialize
    def __init__(self, name, ip, user='cisco', pw='cisco123!'):
        NetworkDevice.__init__(self, name, ip, user, pw)

    # Connect to device
    def connect(self):

        print('--- connecting XR: ssh ' + self.username +
              '@' + self.ip_address)

    # Must set terminal length to zero for long replies, no pauses
    def set_terminal(self):
        print('--- setting terminal length to 0')
        self.session.sendline('terminal length 0')
        self.session.expect(['>', pexpect.TIMEOUT, pexpect.EOF])

    # Get interfaces from device
    def get_interfaces(self):

        self.interfaces = '--- XR Device interface info ---'
```

STEP 3: CREATE FUNCTION TO READ INFORMATION FROM FILE

Create a function that reads the devices from the file **devices-13.txt**, creating and returning the list of device objects, created as appropriate for each IOS, IOS-XR, and generic device in the file.

```
# ============================================================================
def read_devices_info(devices_file):

    devices_list = []

    file = open(devices_file, 'r')
    for line in file:

        device_info = line.strip().split(',')

        # Create a device object with this data
        if device_info[1] == 'ios':

            device = NetworkDeviceIOS(device_info[0], device_info[2],
                                      device_info[3], device_info[4])

        elif device_info[1] == 'ios-xr':

            device = NetworkDeviceXR(device_info[0], device_info[2],
                                     device_info[3], device_info[4])

        else:
            device = NetworkDevice(device_info[0], device_info[2],
                                   device_info[3], device_info[4])

        devices_list.append(device)

    return devices_list
```

STEP 4: CREATE FUNCTION TO DISPLAY INFORMATION

Create a function to display the device information (name, IP, username, password), and then display the interface information.

NOTE:
Your **interfaces** for this exercise are only the complete string returned by the device via the show interface summary command. You do not need to parse that output, since this challenge is about OOP and not concerned with details of parsing CLI output.

```
# ====================================================================
def print_device_info(device):

    print('------------------------------------------------------------')
    print('    Device Name:      ', device.name)
    print('    Device IP:        ', device.ip_address)
    print('    Device username:  ', device.username,)
    print('    Device password:  ', device.password)

    print('')
    print('    Interfaces')
    print('')

    print(device.interfaces)
    print('------------------------------------------------------------\n\n')
```

STEP 5: CREATE MAIN CODE

Your **main** code should read the devices file using your **read** function, and for each device object connect, then perform a get_interfaces, then print the device data using your **print** function.

```
# Main program: connect to device, show interface, display
devices_list = read_devices_info('devices-13.txt')

for device in devices_list:

    print('==== Device ================================================')

    session = device.connect()  # Connect to device
    device.get_interfaces()  # Get interface info for this device
    print_device_info(device)  # Print interface info for this device
```

STEP 6: SAVE, RUN AND VERIFY APPLICATION

Save you application and then run it from the terminal rather than from within visual studio code.

```
~/labs/prne$ python3 defining-child-classes.py
```

The output from your application will be displayed in your terminal window, verify that it is comparable to below.

```
devasc@labvm:~/labs/prne$ python3 defining-child-classes.py
==== Device ================================================
--- connecting IOS: telnet 192.168.56.101
--- attempting to: username: cisco
--- setting terminal length to 0
------------------------------------------------------
    Device Name:       CSR1kv
    Device IP:         192.168.56.101
    Device username:   cisco
    Device password:   cisco123!

    Interfaces


CSR1kv#terminal length 0
CSR1kv#show interfaces summary

 *: interface is up
 IHQ: pkts in input hold queue     IQD: pkts dropped from input queue
 OHQ: pkts in output hold queue    OQD: pkts dropped from output queue
 RXBS: rx rate (bits/sec)          RXPS: rx rate (pkts/sec)
 TXBS: tx rate (bits/sec)          TXPS: tx rate (pkts/sec)
 TRTL: throttle count

  Interface                  IHQ       IQD       OHQ       OQD      RXBS      RXPS      TXBS      TXPS      TRTL
-------------------------------------------------------------------------------------------------------------------
* GigabitEthernet1             0         0         0         0         0         0         0         0         0
  GigabitEthernet2             0         0         0         0         0         0         0         0         0
CSR1kv#
------------------------------------------------------


==== Device ================================================
------------------------------------------------------
    Device Name:       base-01
    Device IP:         192.168.56.130
    Device username:   unknown
    Device password:   unknown

    Interfaces

--- Base Device, does not know how to get interfaces ---
------------------------------------------------------
```

PART 4 (OPTIONAL BUT HIGHLY RECOMMENDED)

As this lab is completed in NETLAB+ and your code files will be erased when the reservation ends, it is advisable to save your files in GitHub under your repository for this course.