EECS 293
Software Craftsmanship
2016 Fall Semester

Programming Assignment 4

Due at the beginning of your discussion session on

September 26-30, 2016

Reading

Read Chapter 5 and Section 19.6 in Code Complete.

Grading Guidelines

Starting with this assignment, an automatic C (or less) is triggered by:

- Any routine with complexity greater than 4, or by
- Any piece of code that is essentially repeated.

Programming

First, make all the changes discussed in your discussion section. Additionally, you should refactor your code to make sure that it adopts the principles covered in the reading assignments.

Message Communication (Ruby and Scala only)

Message communication is based on the method chain: peer.recv, message.reach, device.recv. The invocation chain is needed in Java (why?) but tortuous and non-idiomatic in other language. Propose and implement a simpler alternative for message communication.

Connections

Devices and connectors are only useful if you can plug in a cable between them. It is time to connect them!

Connection Exception

The following methods and classes support device interconnections. First, define a public `ConnectionException` that extends the standard `Exception` class. A `ConnectionException` will be used to signal problems when attempting to interconnect devices. The `ConnectionException` contains a public enumerated type `ErrorCode` that can take the values `CONNECTOR_BUSY`, `CONNECTOR_MISMATCH`, and `CONNECTION_CYCLE`. It also has private final variables that hold a `Connector` and an `ErrorCode`, and their getters. Finally, it defines a public constructor that sets the connector and error code.

Java also requires that an `Exception` should define a `serialVersionUID`. The project will not use the serial version UID, but you can set it to the value 293.

Connector

Add to the connector the method:

```
public setPeer(Connector peer) throws ConnectionException
```

that sets the peer of this connector. If the peer is null, it throws a `NullPointerException`. If the connector already has a peer, it throws a `ConnectionException` with the appropriate error code. If the new peer and the connector have the same type, it throws a `ConnectionException` with the appropriate error code.

Devices

Add to the `Device` interface and to the `AbstractDevice` class a method `public Set<Device> peerDevices()` that returns the devices to which this device is connected directly through one of its connectors.

UXB traversal and Cycles

Since a UXB device can have multiple connectors, it would be helpful to know all the other devices to which a device is connected, either directly or through various hubs. However, UXB connectivity presents all sorts of issues. For example, a novel feature of UXB is that a hub can have multiple peripheral connectors. As a result, a UXB hub can be connected back to itself (!). You can also create a loop in good old USB by connecting a hub back into one of its input ports.

However, a USB hub has a single upstream connector, so you would not be able to power up this contraption. In UXB, hubs can have multiple peripheral connectors, and a spare upstream connector can be ultimately be used to power up a loop. Furthermore, you could create long cycles, in which hub 1 plugs into hub 2, which plugs to hub 3, etc., which then plugs back to hub 1.

Devices

To get a handle on these issues, you should define the following two methods on the `Device` interface and implement them in the `AbstractDevice` class:

- `public Set<Device> reachableDevices()` returns all devices that are reachable either directly (the `peerDevices`) or indirectly from this device
- `public boolean isReachable(Device device)` returns true if the argument is connected directly or indirectly to this device, false otherwise.

A note on run time. You could easily implement `isReachable` as:

```
public boolean isReachable(Device device) {
      return reachableDevices().contains(device);
}
```

However, this implementation would force you to find all reachable device first and then answer the question later. In reality, there can be instances when the target device can be found very early on, in which case there is no point continuing the exploration of the remaining reachable devices. You are expected to create a faster implementation of `isReachable` that returns as soon as the target device is found without exploring all other reachable devices.

In support of `reachableDevices` and `isReachable`, you can define private and *public* auxiliary functions on `Device`.

Connectors

Add to connectors:

`public boolean isReachable(Device device)` returns true if the given device is reachable from this connector's device.

Then, modify `setPeer` so that it throws a `ConnectionException` with an appropriate error code if this cabling would create a loop.

General Considerations

These classes may contain as many auxiliary private methods as you see fit, and additional helper classes may be defined.

You should write JUnit tests to make sure that your primary methods work as intended. However, we will revisit testing later on in the course, so extensive testing is not yet recommended. Similarly, your code should have a reasonable number of comments, but documentation is going to be the topic of a future assignment. As a general guideline at this stage of the course, comments and tests should be similar to those accepted in EECS 132.

Submission

Bring a copy to discussion to display on a projector. Additionally, submit an electronic copy of your program to Blackboard. In addition to your code, include a README file explaining how to compile and run the code. The code should be handed in a zip, tar.bz2, or tar.gz archive. Archives in 7z cannot be accepted.