EECS 293
Software Craftsmanship
2015 Fall Semester

Programming Assignment 2

Due at the beginning of your discussion session on

September 12-16, 2016

Reading

- Chapter 14 in Code Complete
- Items 48 and 50 in Effective Java
- Java's BigInteger documentation (introduction only)
- Section 19.1 in Code Complete (excluding "Forming Boolean Expression Positively", "Guidelines for Comparison to 0", "In C++, consider ...")
- Section 15.1 ("Plain if-then Statements" only) in Code Complete
- Sections 17.1, 19.4 in Code Complete
- Section 17.3 ("Rewrite with a status variable" only) in Code Complete

Programming

The UXB (Universal eXtreme Bus) is a new (fictitious) standard to connect peripherals to computers. It was originally planned as a replacement for multiple existing standards, such as USB or HDMI. In the planning stage, a new feature was introduced: a single UXB device can have multiple ports so that the same device can be connected to multiple computers. For example, a printer can be plugged into multiple nearby computers, all of which would be able to print. In addition, a Webcam can be connected to several computers, enabling multiple users to access the same video feed. Furthermore, multiple ports introduce redundancy and consequently better fault tolerance. For example, if a peripheral is

connected twice to a computer, then the device can continue to operate even if a cable or an UXB hub fails. Although multiple device ports seemed a great idea, they caused no end of aggravation to the UXB programmer (you!).

Package

You should organize your project in a package. The package name is up to you: it could range from the simple (‘uxb) to the detailed (‘edu.cwru.<cwruid>.uxb).

Connectors

A UXB device has multiple *connectors*. Each connector is a physical plug that enables the user to insert a cable for connecting the device to a computer or to a UXB hub. Connectors are of two types: *computer-side* and *peripheral-side*. A computer-side connector is an outlet that is installed in a computer and a peripheral-side connector is a plug that is installed in a peripheral device. A UXB *cable* always runs from a peripheral to a computer but never from a computer to a computer or from a peripheral to a peripheral. A computer can have multiple computer-side connectors so that multiple peripherals can be added. A peripheral can have multiple peripheral-side connectors so that the peripheral can be added to multiple computers. A UXB hub can have multiple computer-side and peripheral-side connectors.

Define a public final class `Connector`. A `Connector` contains a public enumerated type `Type` that takes the values `COMPUTER` and `PERIPHERAL`. A `Connector` has the private variables:

- `final int index`, which represent the plug number in the connector’s device,
- `final Type type`, which is the type of this connector,
- `Optional<Connector> peer`, which is the other connector if any to which this connector is plugged.

The Connector has the following public methods:

- `Connector(int index, Type type)` creates a new connector with the given index and type, and no peer.

- `int getIndex()`, `Type getType()`, `Optional<Connector> getPeer()` return the index, type, and peer of this connector.

There is no method yet for setting the peer, but this and other additional methods will be added in the rest of the project.

Messages

UXB cables carries messages between devices. Define a public interface `Message` that has no methods (not yet). Different types of messages are possible, but in this assignment, you will only implement a numeric message. Define a public final `BinaryMessage` class that implements `Message`, that contains a private final `BigInteger`, and that has the following methods:

- `public BinaryMessage(BigInteger value)` initialize this message with a copy of the given value. If the value is null, the message should contain zero. The constructor does not throw any exception.
- `public BigInteger getValue()` returns the underlying value.
- `public boolean equals(Object anObject)` compares this message to the specified object, and returns true if and only if the argument is not null, is a `BinaryMessage` object, and if the underlying integers are equal.

Device Class

UXB devices are grouped in *classes*. The concept of device class similar in traditional USB. UXB classes are: audio, communication, human-interface, physical-interface (e.g., force feedback joysticks), image, printer, mass storage, video, audio-video, virtual reality, and hub. Create a public enumerated type `DeviceClass` with values `UNSPECIFIED`, `AUDIO`, `COMM`, `HID`, `PID`, `IMAGE`, `PRINTER`, `STORAGE`, `VIDEO`, `AV`, `VR`, and `HUB`.

Devices

A UXB device is a UXB-enabled computer, peripheral, or hub.

Interface

Create a public interface `Device` with the following methods:

- `Optional<Integer> getProductCode()` returns the product code of this device. If the product code is unknown, return an empty optional.
- `Optional<BigInteger> getSerialNumber()` returns the serial number of this device. If the serial number is unknown, return an empty optional.
- `Integer getVersion()` returns the UXB version that this device supports.
- `DeviceClass getDeviceClass()` returns the class of this UXB device.
- `Integer getConnectorCount()` returns the number of connectors that this device has.
- `List<Connector.Type> getConnectors()` returns the type of each connector in this device.
- `Connector getConnector(int index)` returns the connector of this device at the given index.

In future assignments, additional methods will be defined on the `Device` interface.

Connectors (again)

Add to the connector one more private final variable of type `Device`, which represents the device to which the connector belongs. The constructor correspondingly becomes `Connector(Device device, int index, Type type)` Also, add the public method `Device getDevice()` to return the device to which this connector belongs.

Abstract Device

A prototypical UXB device is the

```
AbstractDevice<T extends AbstractDevice.Builder<T>>
```

It implements the `Device` interface and contains private final variables for the product code, serial number, version, and connector lists. It implements, as per the interface: `getProductCode()`, `getSerialNumber()`, `getVersion()`, and `getConnectors()`, and `getConnector(int index)`.

Builder

The `AbstractDevice<T>` contains a public static abstract nested class `Builder<T>`. The builder has private variables for version,

product code, and serial number. It also has a list of connector types. The builder has public methods:

- `Builder(Integer version)` creates a new builder with the given UXB version, no connectors, empty product code, and empty serial number.
- `T productCode(Integer productCode)` sets the product code to the given value. If the `productCode` is null, set it to an empty optional. Return `getThis()` (see below).
- `T serialNumber(BigInteger serialNumber)` sets the serial number to the given value. If the serial number is null, set it to an empty optional. Return `getThis()`.
- `T connectors(List<Connector.Type> connectors)` set the connector types to a copy of the given value . If argument is null, the device will have no connectors. Return `getThis()`.

The builder has protected methods:

- `abstract T getThis()`.
- `List<Connector.Type> getConnectors()` returns a copy of the connector types.
- `void validate()`throws an `NullPointerException` if and only if the version number is null. The exception should contain a clear explanatory message.

An abstract device implements:

```
protected AbstractDevice(Builder<T> builder)
```

to initialize an abstract device from the given builder. The constructor must not throw any exception.

Hubs

A `Hub` is a concrete device that extends an `AbstractDevice<Hub.Builder>`.

Builder

A `Hub` contains a public static nested class `Builder` that extends `AbstractDevice.Builder<Builder>` and the following public methods:

- `Builder(Integer version)` creates a new builder with the given UXB version, no connectors, and with empty product code and serial number.
- `Hub build()`initializes the hub with the builder's version, product code, serial number, and connector list. If the version is null, or the hub has no computer connector, or the hub has no peripheral connector, it throws an `IllegalStateException` with a clear explanatory message.

The builder has protected methods:

- `Builder getThis()`returns this builder.
- `void validate()`throws an `IllegalStateException` if and only if the version number is null, or if the hub has no computer connector, or if the hub has no peripheral connector. The exception must contain a clear explanatory message.

The hub has a private constructor:

`private Hub(Builder builder)` initializes the hub from the given builder. The constructor must not throw any exception.

The hub overrides the public method `getDeviceClass()` to always return `HUB`.

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. Additionally, comments should only be applied to the code sections that you feel are not self-documenting.

Blackboard Resources

The Course Document page contains a folder on useful Java features, such as enumerated types and (under Java 8) the optional class.

Discussion Guidelines

The class discussion will focus on:

- High-level code organization
- The design and documentation of the implementation
- Straight-line code, conditional code

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.