EECS 293

Software Craftsmanship

2016 Fall Semester

Programming Assignment 3

Due at the beginning of your discussion session on

September 19-23, 2016

Reading

Read Chapters 16, 19.5, and 19.6 in Code Complete, and Item 55 in Effective Java.

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. In particular, you should modify your code to avoid methods with high McCabe's complexity.

Devices

Modify the `getConnectors` method in the `Device` interface and `AbstractDevice` class so that it returns a copy of the `List<Connector>` of this device.

More Messages

Define a public final `StringMessage` class that implements `Message`, that contains a private final `String`, and that has the following methods:

- `public StringMessage(String string)` initialize this message with the given string. If the string is null, the message should contain an empty non-null string. The constructor does not throw any exception.
- `public String getString()` returns the underlying string.

- `public Boolean equals(Object anObject)` compares this string message to the specified object, and returns true if and only if the argument is not null, is a `StringMessage` object, and if the underlying strings are equal.

Furthermore, the `StringMessage` delegates to the underlying string the following public methods: `length`, `charAt`, `contains(CharSequence)`, `endsWith`, `startsWith(String)`, `indexOf` (all methods), `lastIndexOf` (all methods), `isEmpty`, and `hashCode`.

Peripherals

It is time to add peripherals to UXB! Similar to hubs, abstract peripherals extend abstract devices. An

`AbstractPeripheral<T extends AbstractPeripheral.Builder<T>>`

has a public static abstract nested builder that extends the `AbstractDevice`'s and that overrides the `validate` method to check not only the version but also that all connectors are of type peripheral. The builder also has a public constructor whose function is to invoke that of its parent. The `AbstractPeripheral` has a protected constructor `AbstractPeripheral(Builder<T> builder)`, which simply invokes the `AbstractDevice` constructor.

Printers

An abstract printer extends an abstract peripheral and overrides `getDeviceClass` to return `PRINTER`. Its builder extends the `AbstractPeripheral`'s. Both abstract printers and their builders have constructors that invoke those of their parents.

A `SisterPrinter` is a concrete printer that extends `AbstractPrinter`. Its builder extends the `AbstractPrinter`'s and has the methods:

- `public SisterPrinter build()`validates the builder and returns a new `SisterPrinter`.
- `protected Builder getThis()` returns this builder.

Create also a printer called `CannonPrinter` (with two n's!) that is similar to the `SisterPrinter` (so far).

Video Devices

A video device, such as a Webcam, is capable of capturing a continuous video stream or a snapshot. Create a class `AbstractVideo` that capture the general concept of a video peripheral. The implementation of `AbstractVideo` should be conceptually similar to `AbstractPrinter`. Then, create the concrete class `GoAmateur` that extends `AbstractVideo` and is otherwise similar to a `SisterPrinter`.

Communication

UXB devices are only helpful if they can communicate with each other. The following methods will support device communication.

First, in the `Message` interface, add a method

```
void reach(Device device, Connector connector)
```

which signifies that the `Message` has reached the given device coming from the given connector. Correspondingly, a connector implements a method

```
public void recv(Message message)
```

which makes sure that the message reaches the connector's device. Furthermore, add the related new methods in the `Device` interface:

```
void recv(StringMessage message, Connector connector)
void recv(BinaryMessage message, Connector connector)
```

These methods signify the arrival of a message at the given connector in the device. A `StringMessage` and a `BinaryMessage` implement the reach method by invoking `recv` on the given device with this message as one of its arguments. So far, these methods achieve nothing but a lot of back and forth invocations. Here is their resolution: `Hub`, `SisterPrinter`, `CannonPrinter`, and `GoAmateur` implement the method `void recv(StringMessage message, Connector connector)` as follows:

- `Hub` logs the informational message: "recv not yet supported".
- `SisterPrinter` logs an informational message "Sister printer has printed the string: " followed by the message string and the printer serial number (converted to a String).
- `CannonPrinter` logs an informational message "Cannon printer has printed the string: " followed by the message string and the UXB version number.

- `GoAmateur` logs an error message: “GoAmateur does not understand string messages: ” followed by the message string and the connector index.

They implement the method `void recv(BinaryMessage message, Connector connector)` as follows:

- `Hub` logs the informational message: “recv not yet supported”.
- `SisterPrinter` logs an informational message “Sister printer has printed the binary message: “ followed by the sum of the message value and the product code (or just the message value if the product code is not present).
- `CannonPrinter` logs an informational message “Cannon printer has printed the binary message: “ followed by the product of the message value and the serial number (or just the message value if the serial number is not present).
- `GoAmateur` logs an informational message: “GoAmateur is not yet active: ” followed by the message value.

In all these methods, if either argument is null, a `NullPointerException` is throw and if the connector does not belong to this device, an `IllegalArgumentException` is thrown.

Broadcast

Create a test that, given a `List<Device>` and a `List<Message>`, delivers all messages to all devices on their zero connector (if any). The device list should contain at least one device of each type, and the message list should contain at least one binary and at least one string message.

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.

Discussion Guidelines

Although the discussion can range through the whole reading assignment, the emphasis will be on:

- Functions that exceed McCabe's complexity of 4 (if any)
- Non-structured programming constructs and break statements (if any)

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.

Grading Guidelines

Advance Warning: starting with Programming Assignment 4, an automatic C (or less) is triggered by any routine with complexity greater than 4 or by any substantially repeated piece of code.