Java Test #2

Checking for message validity

Introduction

In this test you will write some Java source code that you will return to Smart Technology Solutions Limited for examination. By taking part in this test, you agree to waive any intellectual property rights to the solution you submit for examination.

This document describes the background to the test and the requirements for successful completion.

Overview of test

You have to write code to check whether a message, represented as a string of bytes, conforms to a protocol definition. Messages follow a simple protocol with some guard bytes and a checksum. Full details of this protocol are given below.

Submission

When completed, email

- your Java source files (including any supporting and test files)
- a compiled .jar file containing the class(es) implementing the `Validator` interface

to jobs@stspayments.com or to your agent/recruiter, indicating how much time you spent. Do not include publicly-available libraries if you use any, but do indicate what libraries and versions are required and where they might be obtained.

Details

1. You are provided with one Java interface, `Validator`, which you must implement with source code of your own. We have provided you with one example implementation: `ValidatorExample`. You may edit this example, or provide your own implementation of `Validator`.

   Important: your implementation class must provide a public no-argument constructor.

   You are also supplied with a `main` application class, `RunExample`, which instantiates the `ValidatorExample` class and tries it with a test byte array. You may modify this class and write any other classes you need to test your implementation of `Validator`.

2. The `Validator` interface has a single method that accepts a byte array (`byte[]`) containing a message and returns `true` if the message is valid according to this specification, or `false` if not.

   A valid message is one that starts with an STX byte, has correctly-escaped data, and ends with an ETX byte and an LRC with the correct value. The definitions of STX, ETX, LRC and escaping are given below.

3. You are provided with one byte array of test data in the `RunExample` class. You should create other test data byte arrays, and write whatever supporting code you see fit to ensure that your solution is able to handle all types of valid and invalid message.

Message protocol

The following describes the message protocol. It is the sort of protocol often used to interconnect devices using an RS232 serial connection. To send some data, the data is first escaped as necessary (see below), Start-of-Text (STX) and End-of-Text (ETX) bytes are added at start and end, respectively, and a Longitudinal Redundancy Check (LRC) is calculated and appended.

A message is constructed as STX <Data> ETX LRC, where:

Field	**Value**	**Description**
STX	0x02	Start-of-Text: Signifies the start of data.
ETX	0x03	End-of-Text: Signifies the end of data.
DLE	0x10	Data Link Escape: Used in front of any 0x02, 0x03 or 0x10 data bytes to distinguish them from STX, ETX or DLE.
<Data>	sequence of bytes	The data is made up of a stream of zero or more bytes. Any data values which are 0x02, 0x03 or 0x10 are escaped to avoid confusion with STX, ETX or DLE, by prepending them with a DLE: • Data value 0x02 becomes 0x10 0x02 • Data value 0x03 becomes 0x10 0x03 • Data value 0x10 becomes 0x10 0x10 All other data is sent as-is.

LRC	single byte	Longitudinal Redundancy Check: the LRC is calculated as the exclusive-or of all the bytes of the message, excluding any DLEs, excluding the STX, but including the ETX.
		The Java operator for exclusive-or is ^. To calculate the LRC on the message STX, 0x01, DLE, 0x02, 0x25, ETX, you could write:
		`byte lrc = 0;`
		`lrc ^= (byte) 0x01;`
		`lrc ^= (byte) 0x02;`
		`lrc ^= (byte) 0x25;`
		`lrc ^= (byte) 0x03;`

Example

If data is 0x02, 0x0A, 0x10, 0x07 0x08, the whole message would be:

```
0x02 0x10 0x02 0x0A 0x10 0x10 0x07 0x08 0x03 0x14
STX  DLE            DLE                 ETX  LRC
           *    *         *    *    *    *
```

Bytes marked with an asterisk ('*') are included in the LRC calculation.

Notice