**System Design and Error Handling**

\*Use exceptions because they require handling, whereas error codes don’t have to be handled.

\* Should use using statements around bridge to automatically to call Dispose

\*Mimic the File class in C#

\*Remove factory classes. The device programming service should be directly responsible for returning device and bridge information and also programming the device. Can also make static Bridge and Device classes to make opening bridges much easier

\*Exceptions can be handled differently in user code depending on the application type and custom error messages can be reported. Example: If an IOException occurs when we try and open a bridge device because it isn’t connected, then in our command line tool we can report a message telling the user to run a command to see valid bridges. On the other hand, in our Api we can throw a different error telling the user to select a valid bridge device from the list. Here, we threw the same exception in both applications but reported a different error to the end user based on the application type.

\*Make exception names and messages a bit more specific

**Bridge Classes Methods/Exceptions:**

* void Open()
  + NotFoundException/FileNotFoundException – The device was not found.
  + BridgeIOException/IOException - An I/O error occurred while opening the device. It may be opened by another thread or process.
  + AlreadyInitializedException/InvalidOperationException – The bridge is already initialized.
* void Open(string type, string serialNbr)
  + NotFoundException/FileNotFoundException – The device was not found.
  + BridgeIOException/IOException - An I/O error occurred while opening the device. It may be opened by another thread or process.
  + AlreadyInitializedException/InvalidOperationException – The bridge is already initialized.
* static IEnumerable<BridgeInfo> GetAllConnectedBridgeInfo()
* void ConfigureAsSPIBridge(int busId, int frequ, bool msbFirst = true, int chipSelectLine = -1);
  + AlreadyInitializedException - The SPI device is already initialized.
  + BridgeIOException/IOException - An I/O error occurred while opening the SPI device. It may be opened by another thread or process.
* void ConfigureAsI2CBridge(int busId, int devId);
  + AlreadyInitializedException - The I2C device is already initialized.
  + BridgeIOException/IOException - An I/O error occurred while opening the I2C device. It may be opened by another thread or process.
* void WriteToBridge(byte dataToWrite);
  + NotInitializedException/InvalidOperationException – SPI or I2C device used for communication is not initialized
  + BridgeIOException – Cannot read or write. The underlying device may have been disposed, closed or disconnected.
* void WriteToBridge(byte[] dataToWrite);
  + NotInitializedException/InvalidOperationException – SPI or I2C device used for communication is not initialized
  + BridgeIOException – Cannot read or write. The underlying device may have been disposed, closed or disconnected.
* byte[] ReadFromBridge(int numBytesToRead);
  + NotInitializedException/InvalidOperationException – SPI or I2C device used for communication is not initialized
  + BridgeIOException – Cannot read or write. The underlying device may have been disposed, closed or disconnected.
* byte ReadFromBridge();
  + NotInitializedException/InvalidOperationException – SPI or I2C device used for communication is not initialized
  + BridgeIOException – Cannot read or write. The underlying device may have been disposed, closed or disconnected.
* void WriteToBridgeOnPin(int pinNumber, bool signal = true);
  + BridgeIOException – Cannot read or write. The underlying device may have been disposed, closed or disconnected.
* bool ReadFromBridgeOnPin(int pinNumber);
  + BridgeIOException – Cannot read or write. The underlying device may have been disposed, closed or disconnected.
* void Reset();
* void Dispose();

**Device Classes Methods/Exceptions**

* Constructor(IBridge bridge)
  + ArgumentNullException – bridge is null
* void Wipe();
  + Exceptions from bridge device will bubble up and can be handled in user code
* void SetProgrammingMode(ProgrammingMode mode);
* void Program(string filePath);
  + Exceptions from bridge device and file handling service will bubble up and can be handled in user code
* void Program(FileInfo fileInfo);
  + Exceptions from bridge device and file handling service will bubble up and can be handled in user code

**Static Device Class Methods/Exceptions:**

* static string[] SupportedDeviceTypes
* static IDevice Open(string type, IBridge bridge)
  + Exceptions from device and file handling service will bubble up and can be handled in user code
  + NotFoundException/FileNotFoundException – The device was not found.

**Static Bridge Class Methods/Exceptions (Remove factories)**

* string[] SupportedBridgeTypes
* static IBridge Open()
  + Exceptions from bridge device will bubble up and can be handled in user code
  + NotFoundException/FileNotFoundException – The device was not found.
* static IBridge Open(string type, string serialNbr)
  + Exceptions from bridge device will bubble up and can be handled in user code
  + NotFoundException/FileNotFoundException – The device was not found.
* static IEnumerable<BridgeInfo> GetAllConnectedBridgeInfo()
  + Will return an empty list or not. In user code, if the list is empty we can report an error message if desirable.

**Device Programming Service Methods/Exceptions**

* string[] SupportedDeviceTypes { get; }
* string[] SupportedBridgeTypes { get; }
* IEnumerable<BridgeAndDeviceInfo> GetAllConnectedBridgeAndDeviceInfo();
* IEnumerable<BridgeInfo> GetAllConnectedBridgeInfo();
* void ProgramDevice(string bridgetype, string bridgeSerialNbr, string deviceType, programFilePath);
  + Exceptions from bridge device, programmable device, and file handling service will bubble up and can be handled in user code
* void ProgramDevice(string deviceType, programFilePath);
  + Exceptions from bridge device, programmable device, and file handling service will bubble up and can be handled in user code