CS 352 - Summer 2013 - Bittorrent Project 0 - Write Up
==========

Authors:

Isaac Yochelson
Robert Schomburg
Fernando Geraci

Introduction:

The purpose of this document is to explain the general implementation flow of the program. All the implemetation details were documented in javadocs and are available under this folder as index.html.

This bittorrent client implementation heavily leverages the native strengths of java in the fields of portablility and multi-threading. The portability comes from Java's system of partial compilation to intermediary byte code, which can be contained in secure jar files for each package, and then interpreted by any JVM properly implementing version 7 of Java. The multi-threading is being used to provide scalability in our project. Because

Implementation:

The program was originally engineered the way it looks know, a multipackage, multithreaded OOP design.

The general flow of the program is the following:

START
> RUBTClient - main method. - MAIN THREAD
> Bittorrent
> Connect to Peer - PEER THREAD (one per peer)
> Creates a Listener thread - LISTENER THREAD (one per peer independant to peer thread)
* sends and validates HANDSHAKE
* send Bitfield
* recevies Bitfield
* Waits for getting unchoked from Peer
* Send pieces request
* Start receiving pieces
> Analyze each receive piece
> Once piece is complete, verify SHA-1
* Mark that Piece is complete
> Notify Peers
* Once all pieces have been downloaded and verified
> Notify Tracker
> Write File
> Gracefully close all connections
* Terminate Client
TERMINATE

For each peer connection, there will be a 3 level multithreaded process. The main thread will the actual Bittorrent client, then, for each peer connection there will be two threads for each peer, a sending thread which will act as a server, handling incoming file requests, and a listener thread

which will handle incoming communications from the particular peer. These threads of execution reside in the run methods of the Peer and PeerListener classes respectively. The current limit on the number of concurrent peer connections is artificial, in that we are specifically testing each peer in the peer
list to see if its IP address matches 128.6.171.3:6916 prior to opening a connection to it. Once that constraint is removed, the next level of restriction will be that we are selecting TCP ports from an interval allowing only 10 simultaneous connections. Both of these constraints can be relaxed in the future.

The program was though in a MVC design, having a Utils package for public static utility methods, Model package for backbone implementations and an Exceptions package for custom built exceptions. The model consists of object representations of peers, as well as necessary state and file information we maintain in the classes of the bt.Model package. The view consists of the text we output through the system standard output channel. At the moment, the controller is limited to command line arguments, but there will be a command-line interface controller implemented in the future.

Classes:

---

base:
RUBTClient is the controller in our system. Its function is to create the model and view, then provide for user control through text input. (The control is not yet implemented.

bt.Model:
BitTorrent is the top level of our model. This class is instantiated to house all state information related to a particular .torrent file. This class has its own thread, where it determines what messages to send to the peers in the network. The BitTorrent handles communications with the tracker and saves the file to disk.

Peer is the next level of our model. This class is instantiated once for each peer that we are in contact with for each instance of BitTorrent. Peer handles serving requested file pieces to the peer it represents. It also handles handshaking and bitfields. It contains the methods for receiving incoming messages, but those are called by the PeerListener. Each Peer object runs on its own thread of execution.

PeerListener is part of the model, and every instance of PeerListener is linked to an object of type Peer, which is its parent. Peer Listener receives messages from a particular peer relating to a particular .torrent file. A PeerListener object operates on its own thread of execution.

Request is a simple class which is used in a manner similar to a struct in C++. A request is created as an immutable data container related to a single request message.

Server is not yet implemented. Server will be a server to handle incoming requests for communication from peers.

bt.Exception
BencodingException is an exception thrown by methods in Bencoder2.

bt.Utils
Bencoder2 and TorrentInfo were provided for our use in this project. They are used for decoding binary bencoded messages during our interaction with the tracker.

CommandParser is part of the controller in our system. It parses text input to determine the actions the user has requested.

Utilities is a class which is never instantiated. It contains public static methods for use by other classes in the system.

Execution Notes:

---

1. The program could be imported and ran from the Eclipse IDE.
2. The program could be uncompressed and run via terminal from the bin/ in the program's root folder.

The command line Usage is:

```
%> java RUBTClient <input.torrent> <output filename>
```

example:

```
%> java RUBTClient cs352.png.torrent cs352.png
```