**Course:** CS352, Internet Technology, Summer 2014

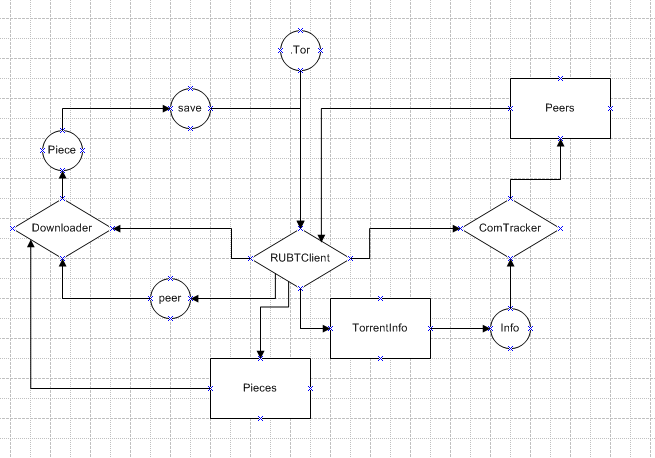
**Instructor:** Bernhard Firner

**Project:** BitTorrent project 1

**Group: “**TheException-all-ists” **-** Cody Goodman, Conrado Uraga**,** Chris Wargo

**Note:** This project is designed to be run on Java1.7 or higher. Earlier versions of the JRE are not supported by this code.

**Project Description and Design Overview**



Diamonds: Main controlling classes

Rectangles: Classes mainly used as data structs

Circles: Instances of the classes/arguments

save and .tor: The command line arguments

**Classes**

**RUBTClient**

This is the BitTorrent client class that contains the main method and directs the operation of the overall process. It accepts command-line arguments for the name of the .torrent file to be

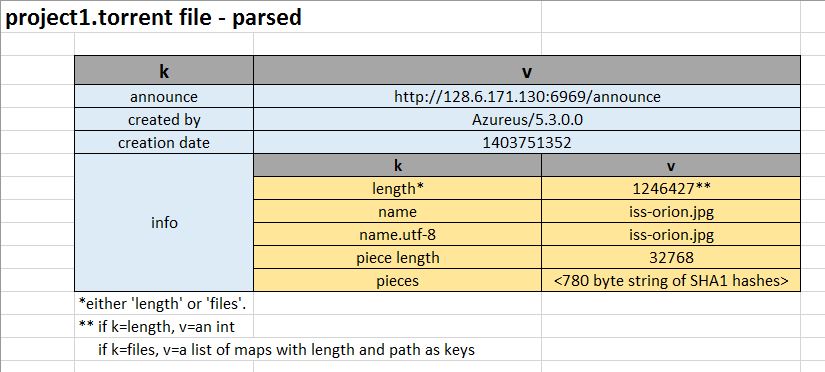
loaded and the name of the file to save the data to. It manages the various steps required to obtain the peer-to-peer BitTorrent download.

**TorrentInfo**

This is a data-structure class written by Robert More II, a TA from a previous semester. The group added a toString() method to help during early testing phases. This class accepts the torrent as a byte array and proceeds to parse and decode the byte array into useful information for the program. This data is stored in various fields within an object created from this class. This class is written to only handle a single file torrent download and will have to be modified during the second part of the assignment. Prior to the distribution of this class, the group wrote its own TorrentInfo class with the ability to recognize multi-file torrents, but we chose to utilize the distributed class for part 1, assuming it was well tested. In addition to only handling single-file torrents, this class does not recognize and parse the following optional torrent key/attributes:

* announce-list
* creation date
* comment
* created by
* encoding
* private (in info dictionary)
* md5sum (in info dictionary)

The project1.torrent file distributed for this assignment contains the following data when parsed. It can be seen that some optional keys are present in the file.



**Bencoder2**

This is another class written by Robert More II and distributed on Sakai. It handles the task of finding k,v tuples in the byte array representation of the Torrent meta-file. Using parsing rules laid out in the BitTorrent 1.0 protocol specification, it both encodes and decodes the various data types found in torrent files; Dictionary, List, Integer, String. The TorrentInfo class calls on Bencoder2 in order to get the data stored in the metafile so that it can be stored as typed attibutes in a TorrentInfo object.

**BencodingException**

This is another class written by Robert More II, which serves as an exception class for the Bencoder2 and TorrentInfo classes.

**CommunicationTracker**

The communication tracker is a class designed to control communications between the client and bit torrent tracker using HTTP protocol. It accepts a TorrentInfo object as an argument, and proceeds to handle the communications involved with that specific torrent download. The primary method in the class is communicateWithTracker(). This method takes the basic tracker URL found in the torrent file and appends it with parameters using standard CGI methods (i.e. a '?' after the announce URL, followed by 'param=value' sequences separated by '&'). ISO-8859-1 encoding is also applied to the info\_hash parameter to ensure proper escaping. This parameterized connection is then made with the tracker using the standard HTTP GET request. The response code and response from the tracker are then stored and the connection is closed. The response from the tracker is then decoded, and the following data are extracted and stored:

* failure reason (if present)
* interval
* complete
* incomplete
* peers (list of peer maps including ID and port)

**MessageHandler**

This class is tasked with deciding which piece to download next and handling the messages received from the peer. An instance of this class represents a unique communication between the client and a specific peer in the P2P swarm. Message handler objects are created in separate threads in the RUBTClient main class. Threading is used in order to handle downloads from multiple peers (and later uploads/seeds).

**Peer**

This class is tasked with managing the connection with a peer and sending/receiving messages between that peer. Handling of messages should take place elsewhere

Stopped here for now. Need to understand how Peer and Piece work. Don’t completely understand yet why there is both a MessageHandler and Peer class. Both seem to handle client/peer communications and data. Going back to study the references a bit more.

**Piece**

TODO

**BtUtils**

This class was written to provide utilities such as constants and commonly called methods as a convenience for developing the a BitTorrent client. Many of the commonly used flag/parameter values for the BitTorrent protocol are stored here with descriptive human-readable constant names, e.g. UNCHOKE\_ID = 1;

**BtException**

An exception that can occur while the Piece class is writing the payload to the resulting file.

**Issues to be addressed in part 2 of the project**

1. The Bencoder class given to us can only decode single-file torrents. It will need to be modified to decode multi-file torrents if it sees a ‘files’ key in the torrent’s info dictionary.

From our last chat:

the client decodes the .tor file and creates and instance of TorrentInfo which is passed to CommunicationTracke

the ComTreacker gets the peers from the tracker and returns them to the rubt client

the client then sends the list of pieces and a peer to download from to a downloader

\*\*Messagehandler

which handles the messages and passes any piece messages to the piece obj to be saved to the file