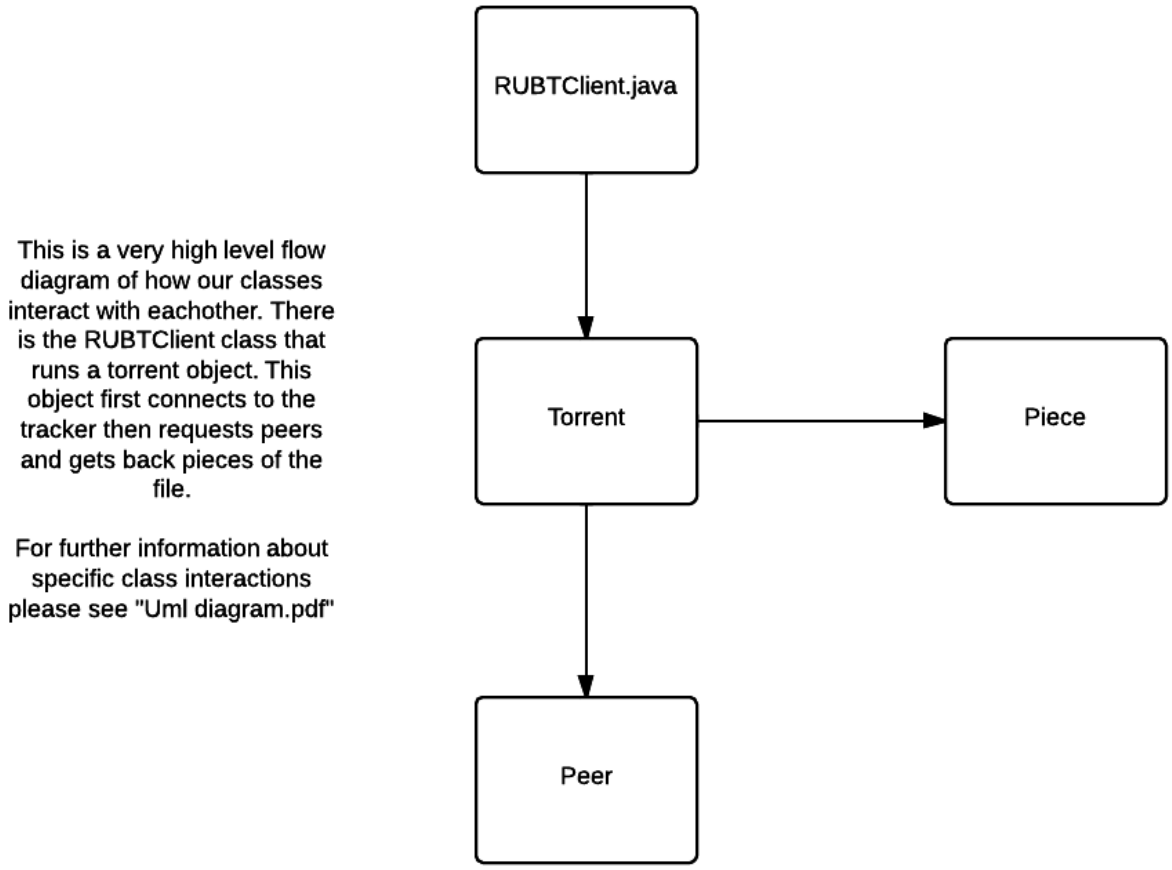
Kaushal Parikh
Edward Zaneski
William Langford

For this part of the assignment we had to implement concurrent uploads to our client that only downloaded in the previous part. We did this all in Peer.java, and handled all the different messages in the case/break statements. We also had to implement simultaneous downloads. The program is run and architectured similarly to our last submission, so please see the write up below for more high level overviews of how it works.

How it works:
There is a very high level diagram below. The RUBTClient class starts a torrent thread, this torrent thread first uses the Tracker class to communicate with the tracker to get information about peers. Then it connects to peers via the Peers class. Then we download chunks of the file and re-assemble the file on our end. See the diagram below, and the “Uml diagram.pdf” also included in the folder for more information:



RUBTClient.java:
This file contains our main function as well as spawns our main thread. It is responsible for taking in command line arguments and parsing the torrent file using the TorrentInformation class

provided by the specification.

Torrent.java:
Torrent manages a single torrent download. It communicates with the tracker and spawns multiple Peer objects, each representing a peer in the swarm. For this project, it only accepts peers whose Id starts with “RUBT” and are from 128.6.171.130. All Peer objects notify their owning Torrent when a Piece is finished downloading and the Torrent object writes that data to file. This is done to prevent simultaneous writes to the same file.

Peer.java:
Peer represents a connection to a peer that is responsible for downloading pieces. It performs all of its processing in its own thread. It uses a private subclass PeerSocketRunnable to perform its network activity. PeerSocketRunnable runs in its own thread and is responsible for the TCP connection with its Peer. Since TCP data is received as a byte stream, it reassembles the data into discrete messages. These messages are then passed on to its owning Peer object. The class is intended to be modified and expanded to handle the Sockets for multiple Peers in a single thread.

BencodeWrapper:
Wrapper around bencode2 class that does most of the tedious typecasting from bytebuffers to strings for us.

Piece.java
Piece is a data object that stores information on a piece of the file. Piece serves as a temporary storage area while Peer downloads the data for the piece. It has the concept of slices, which are segments of the Piece that are the proper size for transmission over the network.