Brian Yoo | 140007707
Jeff Kang | 13900087
Jarrett Mead | 143008288

Overall summary:

The program will first run through its main on RUBTClient.java. As this main runs, it will first initialize important objects to help run threads, contact the peers and the tracker, and write the pieces to file. The desired peers from the list of peers from the torrent file will be extracts and a thread will be started for each peer we wish to download from. Each of these peer threads will first create and handshake and verify it against the one that the peer will send. Afterwards, the peer thread will send and interest message. If an un-choke message is reciprocated, then the peer thread will begin downloading. Upon requesting each piece, the peer thread will download each piece sent from the peer, and then send a have message for that piece after verifying it against the SHA-1 hash from the torrent file.

Simultaneously, a single thread (a listen thread) will listen for incoming connections and accept any that match the handshake we have. If a match occurs, then an individual upload thread will be created to manage the act of uploading to a seed. This thread will send a series of have messages to tell the seed what pieces we currently have. Afterwards, if the seed sends an interest message, we respond with an un-choke message, wait for requests messages, and then send piece messages back to the seed with the correct piece. A loop will read in requests messages from the seed until the seed decides to respond with a “not interested” message or any other message.

As the upload and download threads run simultaneously, another thread will run to update the tracker at the minimum interval that is obtains from the torrent file. Every interval, this thread (UpdateTrackerThread) will contact the tracker and update the “downloaded”, “uploaded”, “left” , and “event” information.

In order to trigger a proper quit for our program, type in q to the console at any time and hit enter. Upon quitting, each thread will end sequentially and then end the main file. Some concerns

Brief description of classes:

RUBTClient: Initializes objects and starts all necessary threads to upload and download from peers.
Torrent: Single object that holds most of the metadata from the torrent file.
ThreadHandler: Contains methods for the main to run. It helps set up IO streams and start/run all of the threads created in main.
Peer: Responsible for the methods of creating/verifying handshakes, opening TCP connections to respective peers.
PeerThread: Responsible for creating download threads per peer, sends interest messages and downloads each piece.
ListenThread: Creates a listening thread. It first creates a socket connection and listens for any incoming connection from peers. Then it tries to verify handshakes with any incoming connections and starts an

upload thread to start seeding with any valid peers.
UploadThread: Takes in a message and checks if it is an interested message. If so, then an un-choke message is reciprocated and the thread then waits for requests messages. Each request message will then be serviced as a piece message is created and send back to the seed.
Message: Contains methods to send interest messages, commit to downloading a file, and generating peer messages based on length prefix, message id, and payload.
Download: Object that holds information for the tracker (i.e. downloaded, left, event) and a synchronized methods to help the threads download simultaneously.
UpdateTrackerThread: Responsible for creating a thread that will run during the minimum interval value given from the torrent file. This thread will update the tracker for each passing interval of time.
UserInputThread: Responsible for creating a thread that will talk to all other threads and tell them to stop running if the user prompts to quit from the program.

Labor Division:
Brian Yoo had been responsible for creating the upload and download threads along with the updating tracker thread. He also established the Torrent class and its threads to help correctly create a URL and establish a tracker connection. Also, created the messages, verified and sent any messages from the peers and seeds.

Jeff helped in creating the threads for download, creating and verifying the handshakes for the upload. He also create the timer to print how long the download took and helped establish the quit, persistence, and ending each thread for the program.

Jarrett created and verified the handshakes for the download threads and created the Download class. He is also responsible for correctly downloading the last piece for the download threads, creating the synchronized methods that allowed the download threads to download simultaneously from the peers. Likewise, Jarrett had also helped to establish the quit, persistence, and terminating each thread for the program.