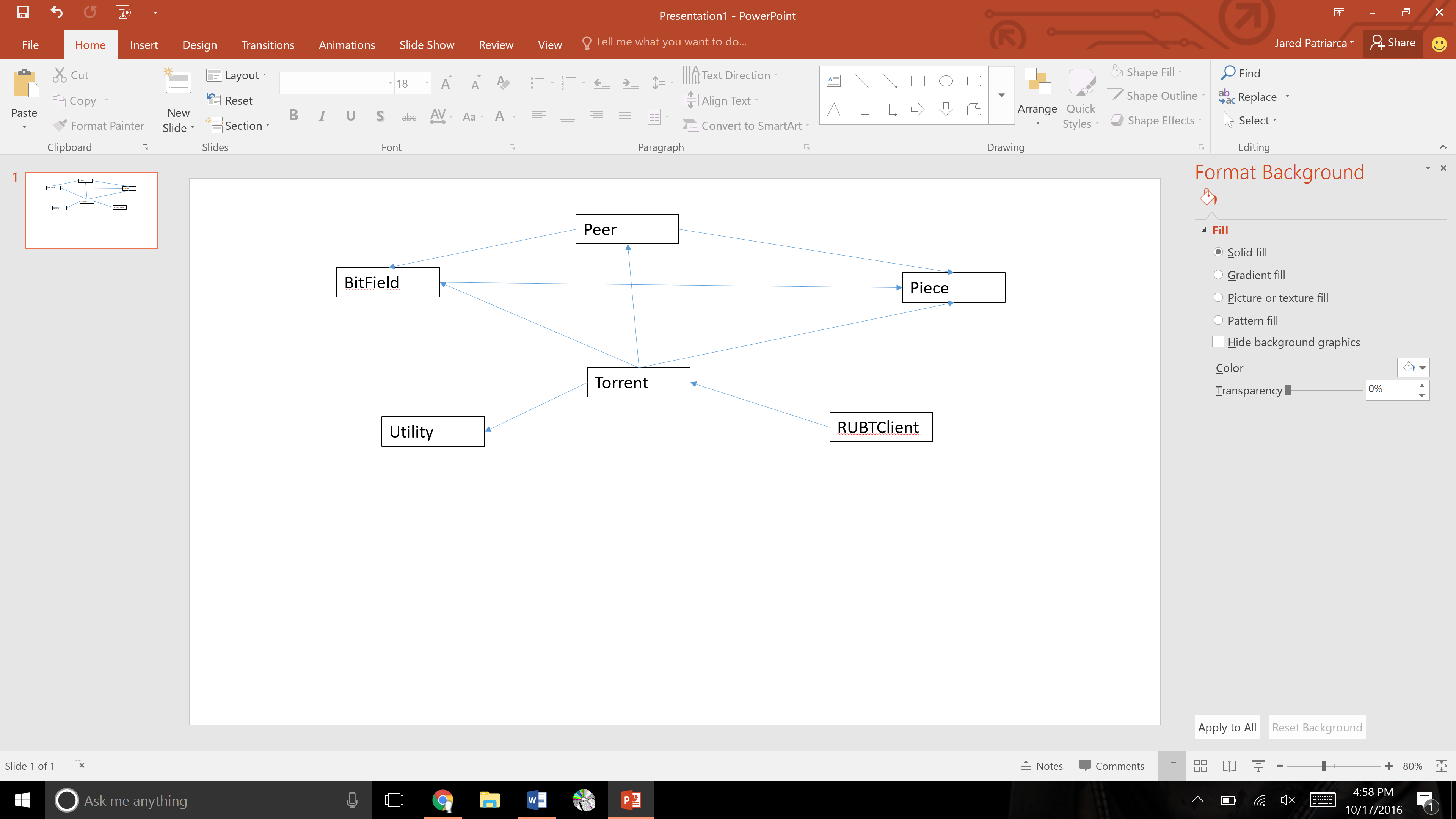
RUBTClient Project - Part 1

Jared Patriarca

Paul Warner



**Overview -**

Our program is started in the RUBTClient where is reads in the arguments, tests them, then calls to create a peerid and passes the information along to the Torrent class to read the file and create the torrent object. This class passes the info to the TorrentInfo class to decode the information then uses the Utility class to escape all the chars before contacting the tracker. It create a bitfield object and an array of piece objects for later use. Next the RUBTClient calls fetchtracker data where we contact the tracker using http and get a list of peers that we can download from. From the list we read the hashmap and find the RU peer. Next the RUBTClient calls beginDownload and we start communication with the peer using TCP/IP. Next the Peer object is called to send the handshake and verify the connection once this is down we call the beginCommunication method that send the unchoke and interested signal to download all the pieces using a requestPiece method. Once the pieces are download we go back to the torrent class to save the pieces to our file.

**Control Classes:**

* BitField.java – This is our bitfield object where we store a value to tell us what part of the file we have and what part is still missing. We have created methods to read a byte and set a bit value at a certain offset to a given value and to search for any values that mean we still need to download that piece of the file.
* Peer.java – This class sets up a peer object that stores all the info about said peer. Then the handshake method is called and we start the communication with the peer by sending it the header, hash, and id. Once this is started we began to communicate that we want to unchoke and are interested in what the peer has. Once this happens we start the download of all the pieces by requesting pieces.
* Piece.java – This is our object file that we store information about each piece we download. We can store the data, length, and infohash and also compare the hash with the correct value from the torrent file.
* RUBTClient.java – This is our main class where we check the input values, open a serversocket for our peer (port 6881 through 6889, whatever one works first) and generate a peerid. We then send the read information to our torrent class to begin the download.
* Torrent.java – This class handles reading in the torrent file and sending it to the torrentinfo class to return to us an object that we can work with. Then we figure out how many pieces and of what size and from where we need to download from (the tracker’s info). Then we contact the given tracker by using http protocol and download a list of peers that are available. All this is also using the ben coder to read and decode most of the data. Now that we have found the peer with “RU” we start the download using TCP/IP and a handshake using the bittorrent protocol given to us in the description. We create our bitfield and an arraylist for all the pieces we need. Next the RUBTClient calls beginDownload and we send the peer the handshake and start downloading the pieces. Once the piece is downloaded we can call write file and it will write the current piece’s date to the file part by part.
* Utility.java – this class is used to escape the chars is a byte buffer string so that they can be passed to the peer using the correct protocol. This is done by adding % and 0 where there is only 1 char not 2.

**Resources Classes:**

* TorrentInfo.java - Provided by Robert Moore II. The TorrentInfo uses the Bencoder2 class to unBencode the torrent information in the .torrent file and return a object that we can get the torrent info from.
* Bencoder2.java - Provided by Robert Moore II. Used to un-Bencode byte arrays.
* BencodingException.java - Provided by Robert Moore II. Exception thrown by Bencoder2.java.
* ToolKit.java - Provided by Robert Moore II. This file allows us the ablility to read in byte buffers and print them out.