# BitTorrent Programming Assignment Phase 1

**Group members**
- Joseph Devita - jd761
- Eonseon Kim - ek530
- Daoun Oh - do184

**Overview**
**RU**BTClient (PHASE ONE) forms the foundation of standard BitTorrent protocol. First, RUBTClient decodes a torrent meta-data file. The results of this decryption are used too construct a HTTP GET request which is sent to the tracker associated with the metadata. The contents of what is returned from the GET is extracted and used to establish communication with peers.

Peer communication is accomplished via the extracted data from the tracker response. A key peer {R,U,B,1,1} is searched for and extracted. For this phase of the project, this is the only peer we attempt to leech a file from. The pieces are decoded and written to an output-file, which is determined via a command-line argument.

We unfortunately did not properly anticipate the amount of small issues we would run into debugging the communications after connecting with the peer. Originally we had tried to structure the program via MVC standards, but when it became clear the debugging was taking longer than anticipated, it was left on the back burner until phase2. The application, in it's current state doesn't download the file properly.

**Class Overview**

**Tracker implements TrackerConnection extends Thread**
Tracker aims to encapsulate all the data associated with the tracker. It is responsible for the initial communication with the tracker. It also houses the logic which decodes the tracker response message and the current download/upload count.

- Stores decoded torrent meta info (field - torrentInfo)
- Binds a socket to communicate w/the tracker (function - setListeningPort(int, int)
- Constructs a URL to send a GET request to the tracker (function - constructURL(byte[], int)
- Sends a GET request to the tracker and returns a byte[] with appropriate data (function - sendHTTPGET(URL)
- Stores current uploaded (for future part of assignment) and downloaded sums (fields - int, int)
- Extracts peers from tracker response (function - update(string, TorrentManager)

**TorrentManager implements TorrentProtocol extends Thread**
TorrentManager handles creation of the file and interfaces with peer(s) and the tracker when appropriate. Logic related to message constructor invocation, SHA hashing, and store message in a FIFO manner.

- Initializes the Tracker *(constructor - via hook)*
- receives peer information from tracker — logic of f()
- starts connections w/peer(s) (only one actually starts in this version) - *(function - configure() )*
- Stores messages in an eventQueue - *(function - addToQueue() )*

- Pops message from a peer via EventQueue - *(function - leech() )*
    - takes appropriate action based on message encoding - *(constructor - Message -> subclass)*
- Check/update *(function(s) - updateFile(Piece, ByteBuffer)*
- create a random peer Id *(function - genPeerId() )*
- SHA1 hashing - *(function - verifySHA1())*
- Parse and create new messages *( function - run() )*

**Peer implements PeerConnection extends Thread**

Peer encapsulates logic and resources associated with communicating to a peer. Raw input and output streams are initialized and created in this class. Synchronous communication(s) are also managed in this class. This class also houses an inner class which send KEEPALIVE message via a timer object.

- Initializes a peer object, launch new thread *(constructor - peer(byte[], int, string, Tracker, TorrentManager)*
- Handshaking with a peer (function - handshake*(byte[], byte[])*
- Binds a socket for a peer connection *(function - connect() )*
- Reads an input stream and decodes new messages *(function(s) connect() & run() )*
- Gets request message(s) from stream *(function - getRequest() )*
- Gets a piece(s) and checks if entire piece *(function - appendAndVerify() )*
- Sends keepAlive message *(inner class - Connection(Peer) )*
- Choke flag *(field - boolean[])*
- interest flag *(field - boolean[])*
- bitfield for message *(field - boolean[])*
- Aggregation of inner class to send keep alive message *(field - Connection)*
- Socket on which to communicate with for this peer *(field - Socket)*
- A reference to the control object *(field - TorrentManager)*

**Abstract Message**

Represents a template which all other message objects are based off (see UML diagram). All 8 type of messages extend this class, which houses a factory method and encoding logic. An abstract signature for adding payload is also declared which all sub-classes implement.

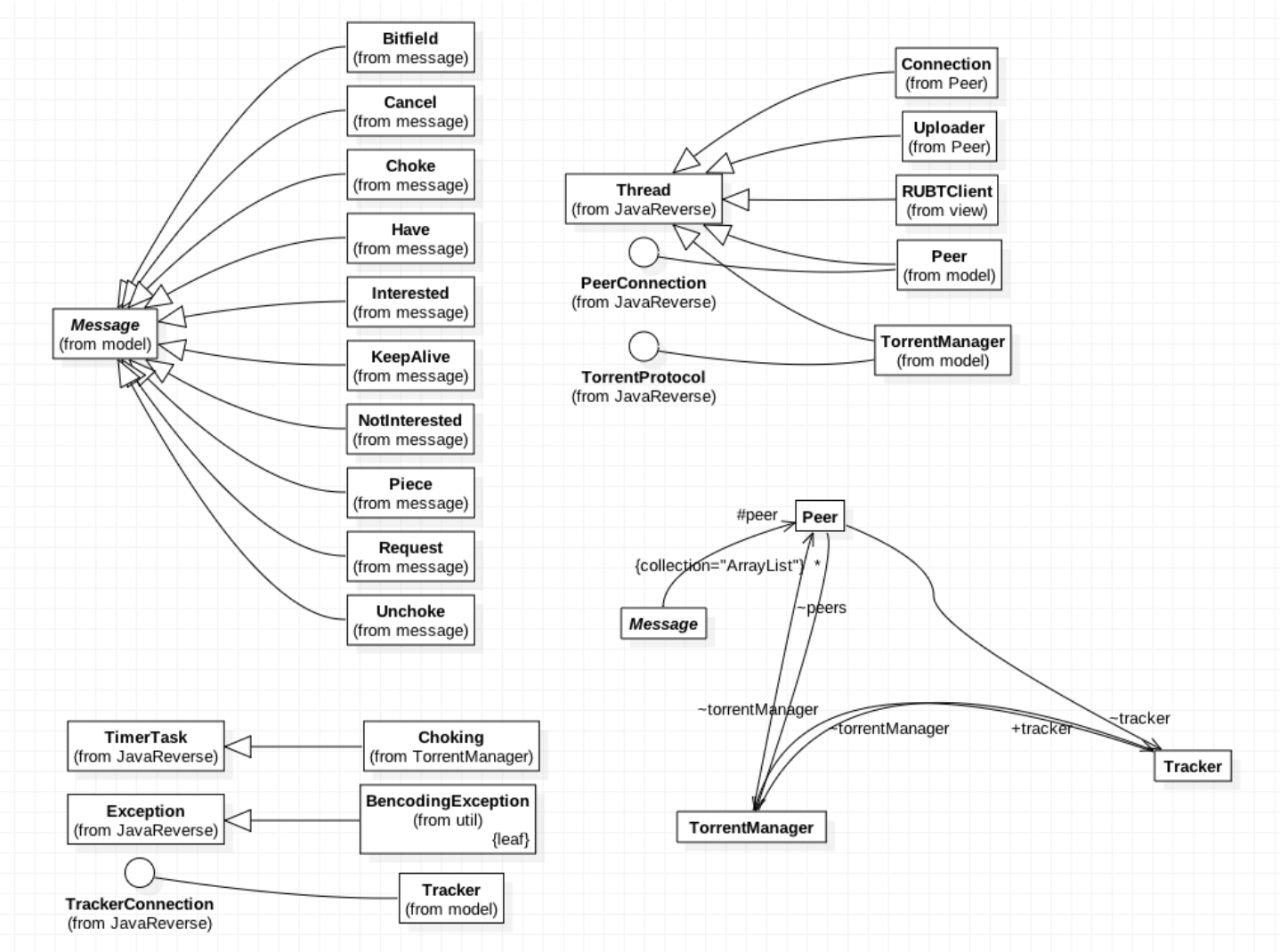
- Encodes a message for output stream *(function - encode (Message, OutputStream)*
- Create appropriate message subclass *(function -MessageFactory(InputStream, Peer)*
- add payload to a message *(abstract function - addPayload(DataoutputStream)*
- static references of message key *(field - byte(s)*
- mapping : byte id -> message name *(field - string[])*

**RUBTClient extends Thread**

This class houses main. Execution starts here by initializing the TorrentManager

- Reads command line arguments and initializes the TorrentManager (constructor - TorrentManager)

## UML Diagram


Bitfield
(from message)
Cancel
(from message)
Choke
(from message)
Have
(from message)
Interested
(from message)
Message
(from model)
KeepAlive
(from message)
NotInterested
(from message)
Piece
(from message)
Request
(from message)
Unchoke
(from message)
Connection
(from Peer)
Uploader
(from Peer)
Thread
(from JavaReverse)
RUBTClient
(from view)
Peer
(from model)
PeerConnection
(from JavaReverse)
TorrentManager
(from model)
TorrentProtocol
(from JavaReverse)
#peer
Peer
{collection="ArrayList"}
*
Message
~peers
~torrentManager
~torrentManager
+tracker
~tracker
Tracker
TorrentManager
TimerTask
(from JavaReverse)
Choking
(from TorrentManager)
Exception
(from JavaReverse)
BencodingException
(from util)
{leaf}
TrackerConnection
(from JavaReverse)
Tracker
(from model)