SYSC3303A – Real-Time Concurrent Systems

TFTP

Group: 2

June 7, 2017

Table of Contents

[Group Members: 2](#_Toc484544124)

[Breakdown of Responsibilities: 3](#_Toc484544125)

[Iteration 1: 3](#_Toc484544126)

[Iteration 2: 3](#_Toc484544127)

[Iteration 3: 3](#_Toc484544128)

[Iteration 4: 4](#_Toc484544129)

[Iteration 5: 4](#_Toc484544130)

[Setup and Testing 4](#_Toc484544131)

[Code 5](#_Toc484544132)

[Diagrams 9](#_Toc484544133)

# Group Members:

1. Josh Campitelli (101010050)
2. Ahmed Khattab (100994398)
3. Dario Luzuriaga (100911067)
4. Ahmed Sakr (101018695)
5. Brian Zhang (101008207)

# Breakdown of Responsibilities:

## Iteration 1:

Josh Campitelli: Helped with the design, writing, and debugging of the project, specifically the handling of packets in the Client and Connection classes.

Ahmed Khattab: Worked towards the overall progress of the project including the shutting down of the server and created the UMC and UML diagrams.

Dario Luzuriaga: Helped work towards the completion of the project, the debugging, and the instructions.

Ahmed Sakr: Provided the base code for the project from assignment 1 and helped build and design the protocol and classes of the project.

Brian Zhang: Help build project modularity including constructing the packet class and debugged errors. Worked on logical problems and instructed team members on version control.

## Iteration 2:

Josh Campitelli: Contributed with the error handling in the client hub and connection class, helping with the interaction with the error simulations as well.

Ahmed Khattab: Introduced the timing diagrams and helped to develop the verbose mode in the server's side.

Dario Luzuriaga: Contributed to develop the interaction with the user in the error simulator, developed the UML diagrams and wrote documentation like README.txt.

Ahmed Sakr: Introduced the multi-thread capacities in the error simulator, helped to develop methods to add errors in packets and organize their block numbers.

Brian Zhang: Created error packets, helped with error handling on the client and connection classes.

## Iteration 3:

Josh Campitelli: Contributed handling errors 1 to 6, categorizing all I/O errors in connection.java, as well as improving the codes in client and server sides.

Ahmed Khattab: Introduced the timing diagrams, specified messages for each error to be displayed to the client's user and helped defining the interaction between Client.java and its user.

Dario Luzuriaga: Contributed to improving the Error Simulator making it easier for users to introduce errors in packets, helped to define new procedures of writing files, and contributed generating documents and UML diagrams.

Ahmed Sakr: Refactored the error simulator, the client, and some of the packet classes. Introduced new documentation across the whole codebase. Wrote many methods for the FileTransfer interface to allow for I/O operations to be completed. Fixed some pre-existing errors.

Brian Zhang: Helped define and integrate all I/O error packets, operated improvements in packet.java and increased the efficiency in handling packets in all instances.

## Iteration 4:

Josh Campitelli: Contributed refactoring the client and the connection classes, adding network errors and timeouts on the connection, client and server classes.

Ahmed Khattab: Introduced the timing diagrams, helped to create the errors for simulation as an upgrade of iteration 3, added more comments on programming code.

Dario Luzuriaga: Helped with the interface consistency, and contributed generating internal and external documents and UML diagrams.

Ahmed Sakr: Added network error simulation capability to the error simulator. Fixed all known errors from iteration 2 and iteration 3. Refactored a huge portion of the codebase to be more modular and consistent. Improved I/O methods.

Brian Zhang: Improved upon previous iterations using TA's feedbacks. Helped with network errors.

## Iteration 5:

Josh Campitelli: Bug fixes.

Ahmed Khattab: Timing diagrams.

Dario Luzuriaga: Bug fixes, helped with the documentation.

Ahmed Sakr: Implemented cross IP TFTP, refactored majority of codebase and fixed bugs.

Brian Zhang: Bug fixes, helped with the documentation.

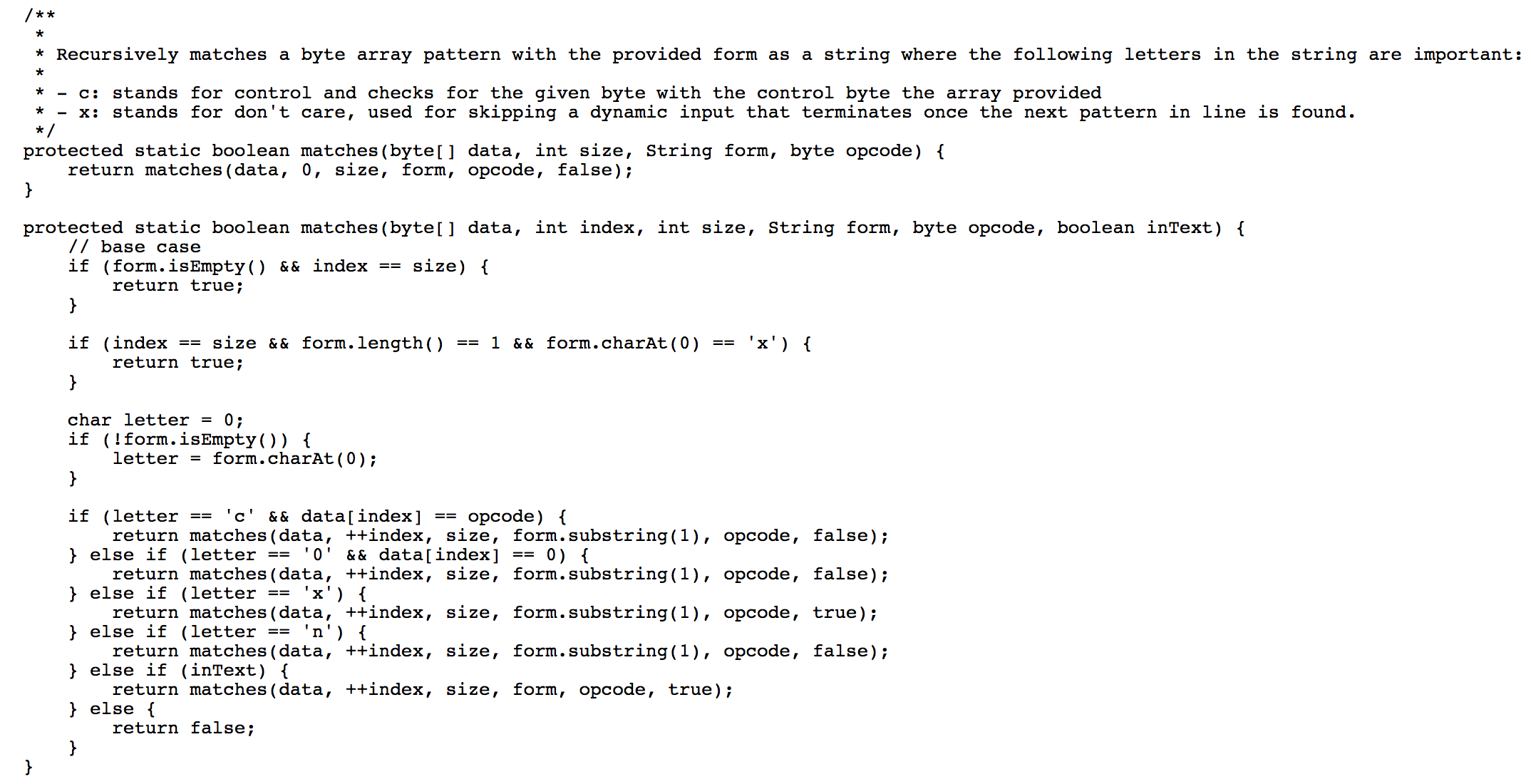
# Setup and Testing

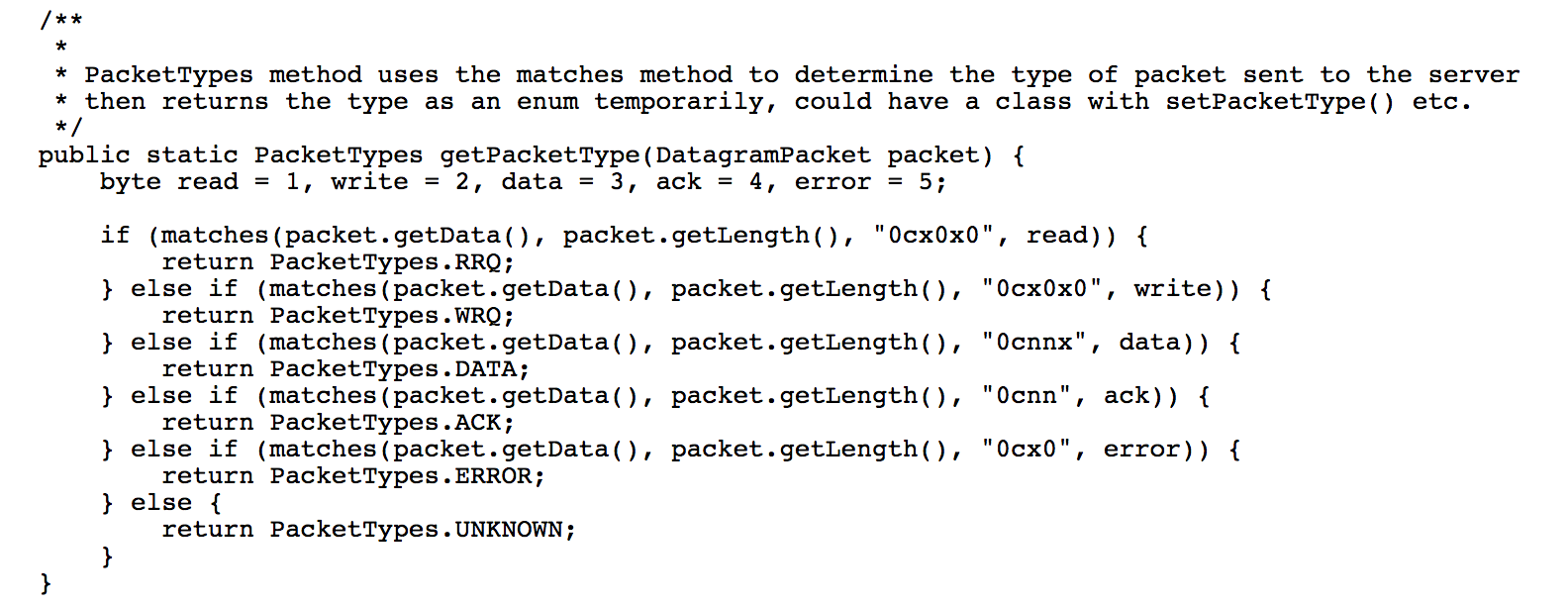
1. Create a new Java project in Eclipse.
2. Uncheck *use default location* and choose this project’s base folder as the new location.
3. After creating the new project, set up the run configuration for *com.tftp.Server* and *com.tftp.ErrorSimulator* on one computer and *com.tftp.Client* on the same and/or different computers.
   1. To set-up the run configurations, select *run configurations* in the dropdown beside the run button, right click *Java Application* and make a new configuration. Give the configuration a name and under *Main class* search and select one of the corresponding main classes: *com.tftp.Server*, *com.tftp.ErrorSimulator* or *com.tftp.Client*.
   2. *com.tftp.Server*, *com.tftp.ErrorSimulator* and *com.tftp.Client* are the Server, Error Simulator and Client respectively.
4. Run and follow the command line instructions for setting up the Server and ErrorSimulator first before running the Client.
   1. The default transfer test files are in data/client or data/server for Client and Server side respectively. They’re named after their size (ie. a 16-kilobyte file is named 16KB.txt)
   2. The Error Simulator operates on a mutable queue and *enqueues* and *dequeues* instructions when called upon. This way, you’re able to input and load multiple error simulations at once before needing to run a Client.
   3. To see the results of a simulated error, read the verbose output of either Client or Server and it will display how it responds to the error (ie. if the modified packet is a data packet with block number 2 and it’s an invalid opcode and the Client sends a write request, then the server will send *Packet type: ERROR*, *Data (as string): Undefined OpCode or Packet* and the Client will print out *Error Packet Received: Error Code: 04, Error Message: Undefined OpCode or Packet*).

# Code

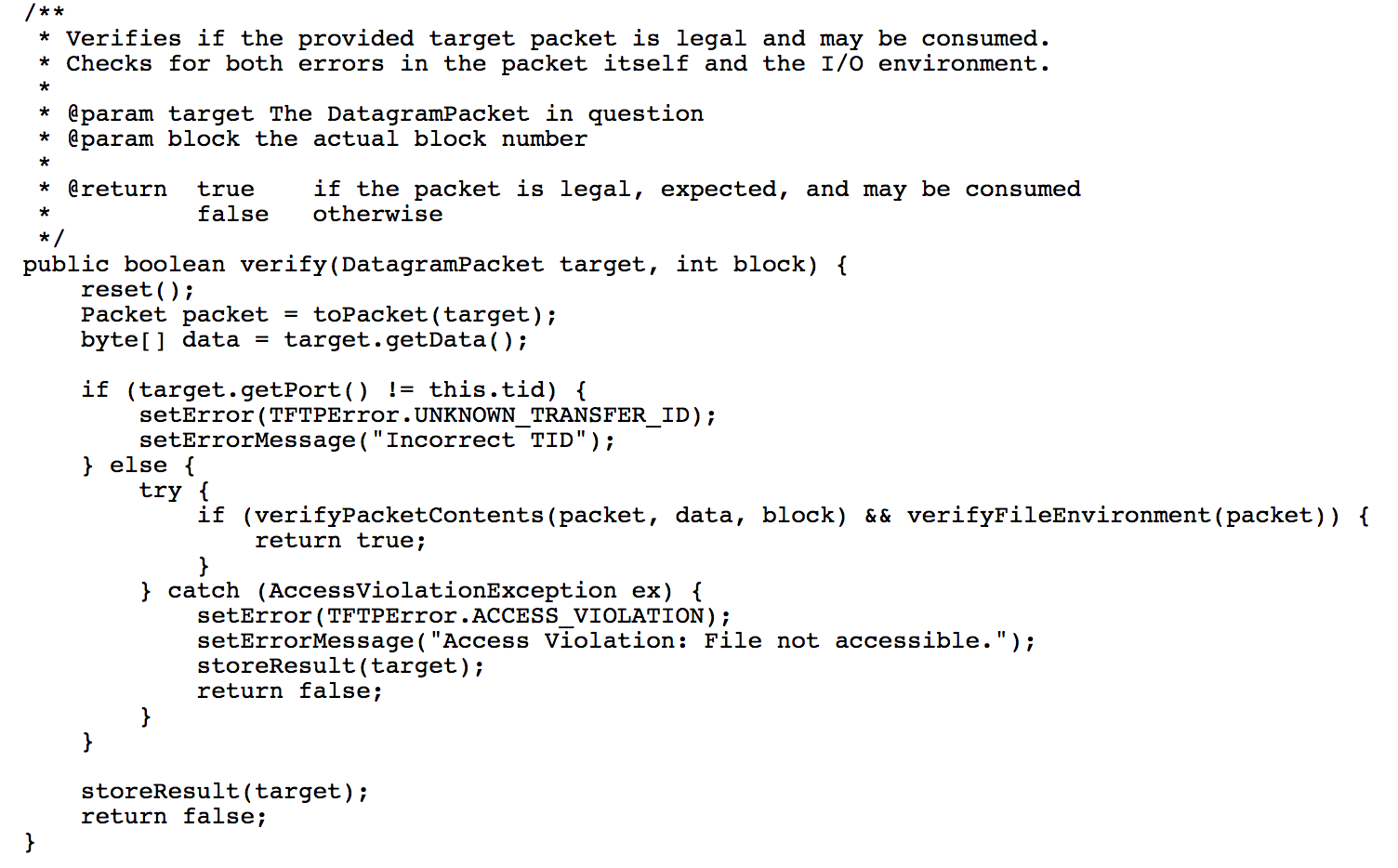
*To see the API for the codebase, extract javadoc.zip and open javadoc/index.html however a few methods may be missing.*

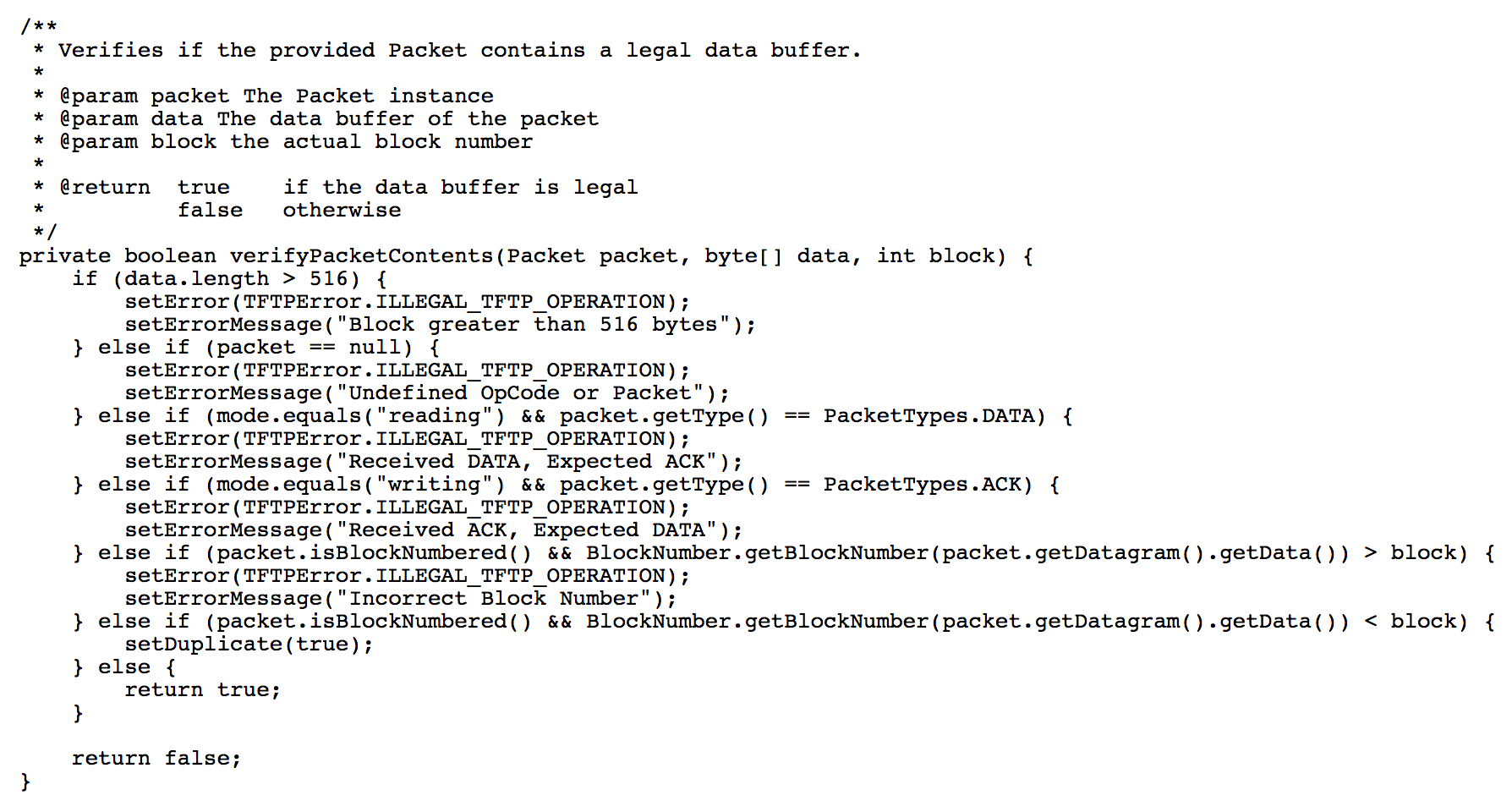
To determine what type of packet is being sent, we used two methods from the **Packet** class; *matches()* for returning true given a certain byte array pattern and opcode, and *getPacketType(DatagramPacket packet)* for providing the byte array pattern based on TFTP DatagramPacket data regulations and the opcode returning an enumerable named *PacketTypes*.

**



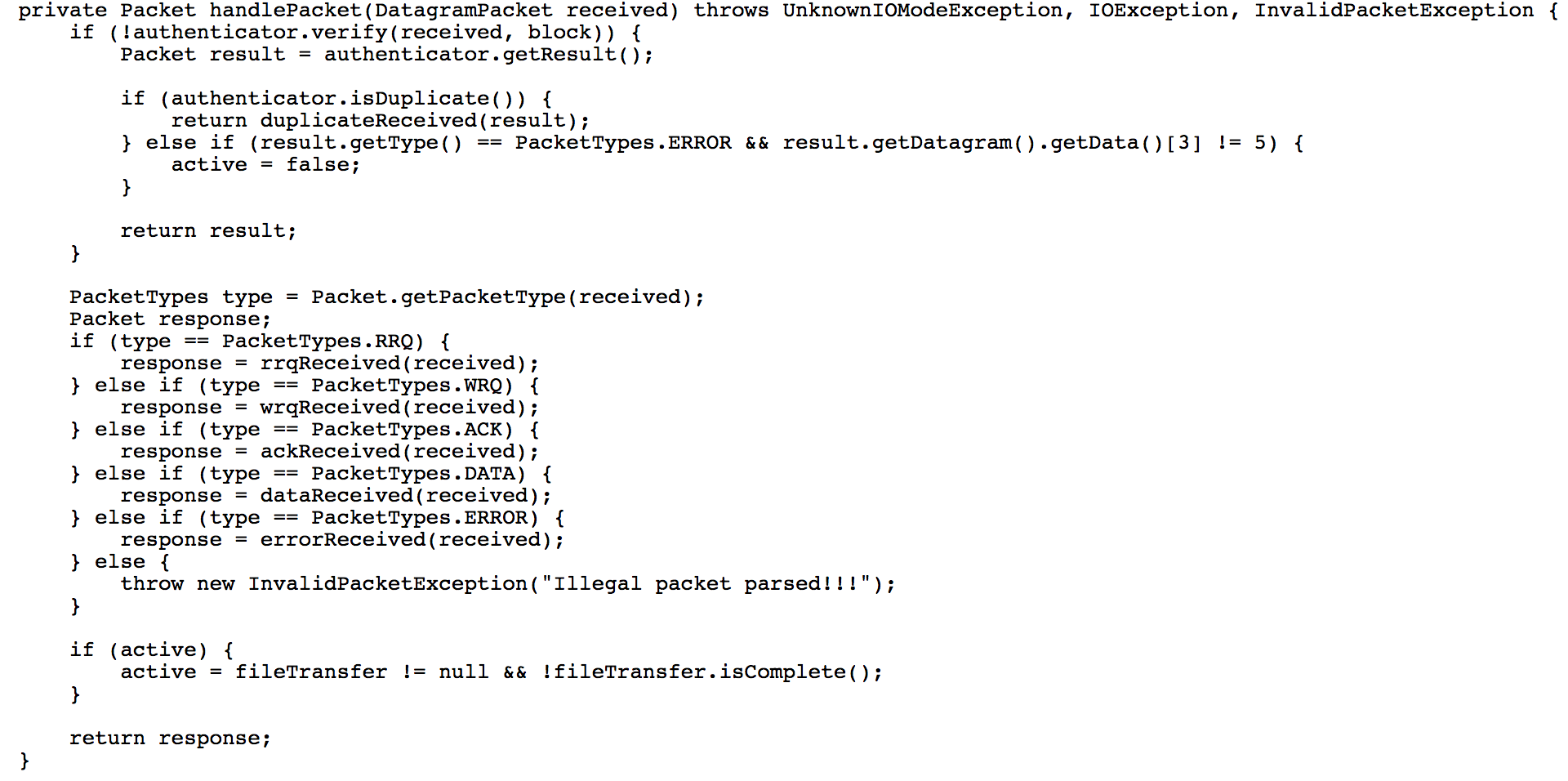
To verify if the DatagramPacket is a valid packet and not an erroneous one, we used the following three boolean methods from the **Authenticator** class; *verify(DatagramPacket target, int block)*, *verifyPacketContents(Packet packet, byte[] data, int block)* and *verifyFileEnvironment(Packet packet)*.



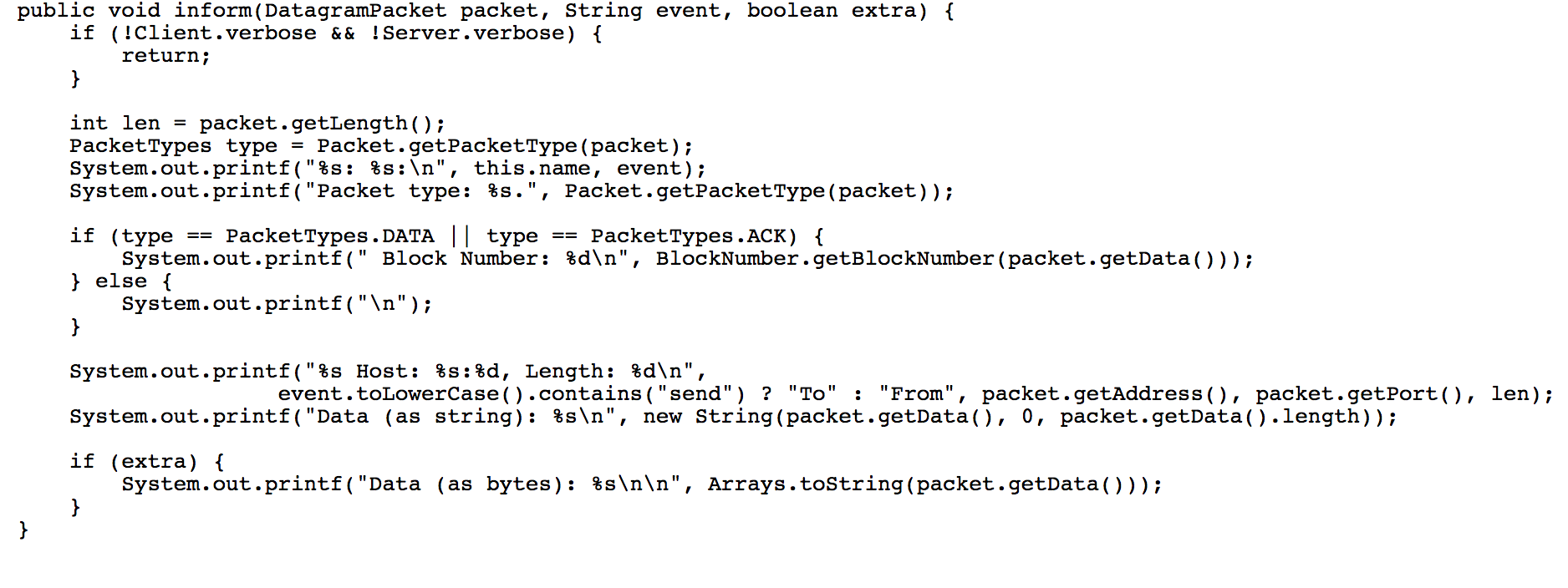




To handle and return the appropriate packet, we used a *handlePacket(DatagramPacket received)* method in the **Connection** class. By determining what *PacketTypes* enumerable the Packet is, we assign the appropriate response to the *Packet* variable *response*.



To print all the information in console we used multiple *printf* statements in  *inform(DatagramPacket packet, String event, boolean extra)*.



# Diagrams