Networking Template Code

Todd Matsuzaki

Daanyaal Syed

Protocol

- A set of standard rules used to communicate across a network
- Requests
  - Client → Server
- Responses
  - Server → Client

Event ID **(Short Int)**	**Message**
`RAND_INT`	`11203`

Event IDs

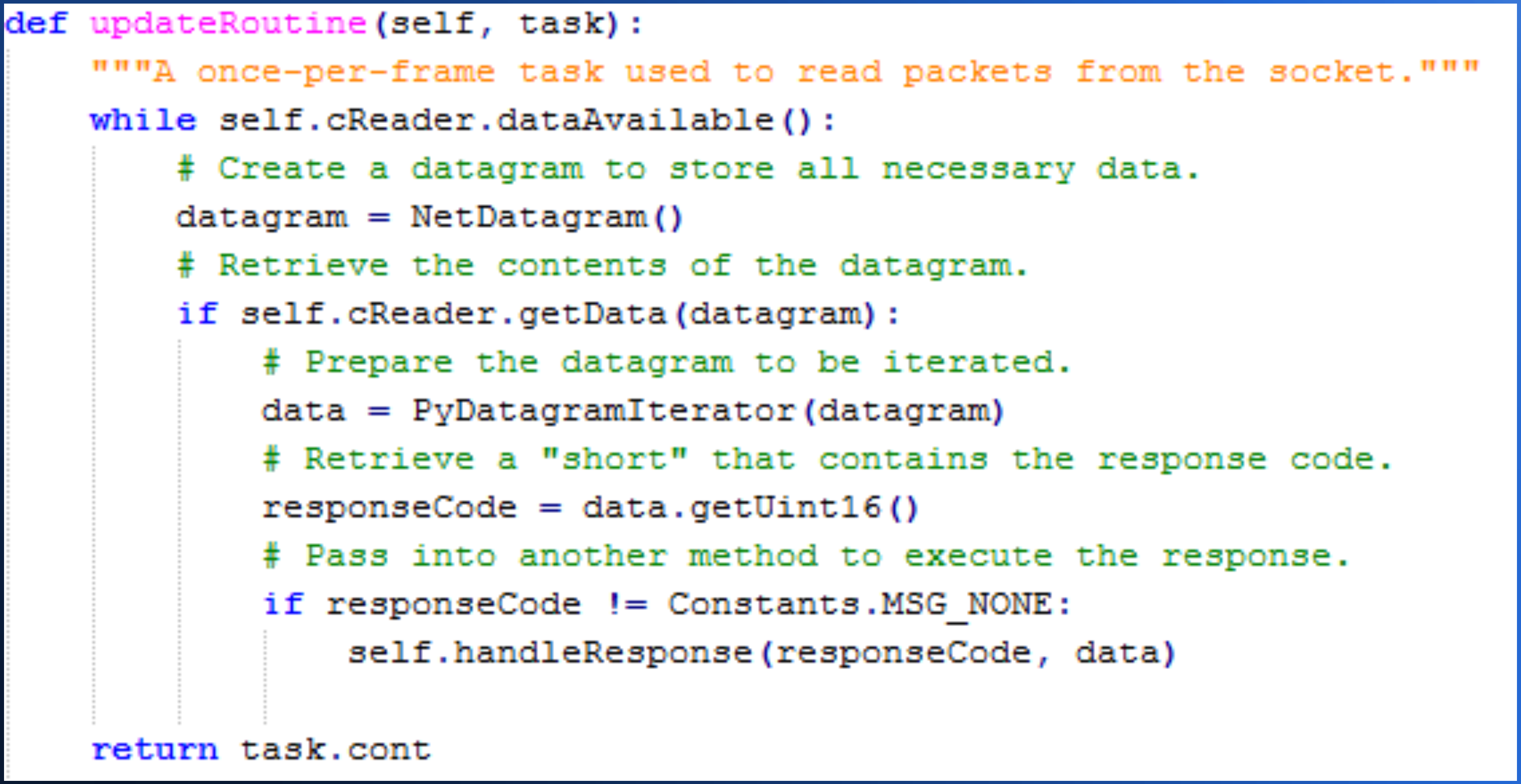
- Lets the recipient of a message know what type of message has been received
- Server and client use the same constants

RAND_INT	1
RAND_STRING	2
RAND_SHORT	3
RAND_FLOAT	4

ConnectionManager

- Sets up the Panda3D classes necessary for connecting and listening to a server
- Has methods for sending requests to and receiving responses from the server
- updateRoutine continuously listens to server for messages

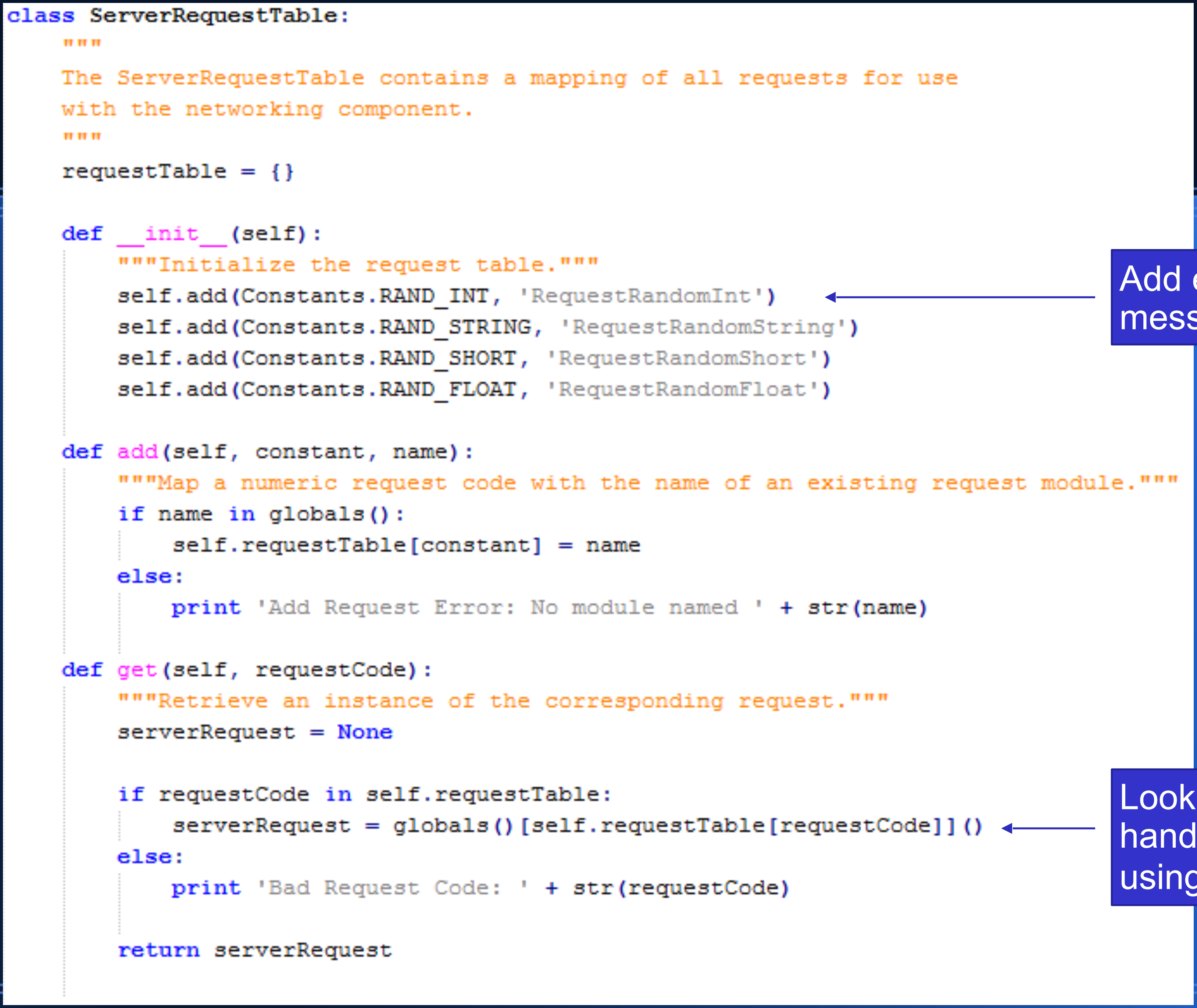
updateRoutine



```python
def updateRoutine(self, task):
    """A once-per-frame task used to read packets from the socket."""
    while self.cReader.dataAvailable():
        # Create a datagram to store all necessary data.
        datagram = NetDatagram()
        # Retrieve the contents of the datagram.
        if self.cReader.getData(datagram):
            # Prepare the datagram to be iterated.
            data = PyDatagramIterator(datagram)
            # Retrieve a "short" that contains the response code.
            responseCode = data.getUint16()
            # Pass into another method to execute the response.
            if responseCode != Constants.MSG_NONE:
                self.handleResponse(responseCode, data)

    return task.cont
```

RequestTable and ResponseTable

- Dictionaries that have an event ID for a key and the name of a message handling method for a value
  - Ex. `{ RAND_INT: requestRandomInt,`
    `RAND_STRING: requestRandomString}`
- Replaces if/else statements



```python
class ServerRequestTable:
    """
    The ServerRequestTable contains a mapping of all requests for use
    with the networking component.
    """
    requestTable = {}

    def __init__(self):
        """Initialize the request table."""
        self.add(Constants.RAND_INT, 'RequestRandomInt')
        self.add(Constants.RAND_STRING, 'RequestRandomString')
        self.add(Constants.RAND_SHORT, 'RequestRandomShort')
        self.add(Constants.RAND_FLOAT, 'RequestRandomFloat')

    def add(self, constant, name):
        """Map a numeric request code with the name of an existing request module."""
        if name in globals():
            self.requestTable[constant] = name
        else:
            print 'Add Request Error: No module named ' + str(name)

    def get(self, requestCode):
        """Retrieve an instance of the corresponding request."""
        serverRequest = None

        if requestCode in self.requestTable:
            serverRequest = globals()[self.requestTable[requestCode]]()
        else:
            print 'Bad Request Code: ' + str(requestCode)

        return serverRequest
```

Add each type of message to the table

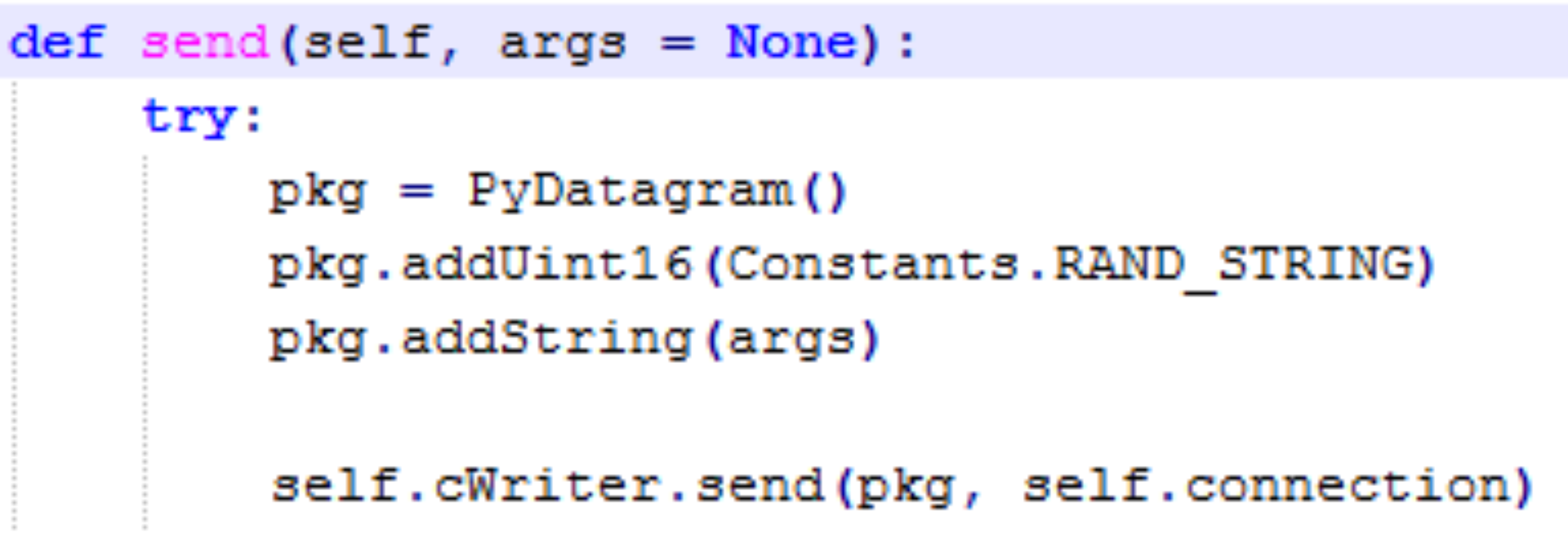
Look up the class that handles `requestCode` using the table

Requests/Responses

- Implement ServerRequest or ServerResponse
  - Provides a logging function
- Need to create a separate module for each type of message being sent between client and server
  - For most messages, this means separate request *and* response modules
    - Counterexample: Heartbeat

Requests/Responses

Example: RequestRandomString



```
def send(self, args = None):
    try:
        pkg = PyDatagram()
        pkg.addUint16(Constants.RAND_STRING)
        pkg.addString(args)

        self.cWriter.send(pkg, self.connection)
```

Steps to Add a New Request/ Reponse (Clientside)

1. Add request and response event IDs to Constants.py
2. Duplicate an existing request module, replacing the event ID (e.g. `Constants.RAND_INT`) and the message type (e.g. `addInt32`)
3. Modify ServerRequestTable
    1. Add the new module to the requestTable using `self.add()`
    2. Import the new module
4. Duplicate an existing response module, replacing the message type
5. Modify ServerResponseTable
    1. Add the new module to the responseTable using `self.add()`
    2. Import the new module

Steps to Add Networking Code to Your Own Modules

- Start a connection to the server with:

```
self.cManager = ConnectionManager()
self.startConnection()
```

- Send a message to the server using:

```
self.cManager.sendRequest(EVENT ID, MESSAGE)
```

Network Server Side

Responding to Requests from Clients

Overview

- Utilities
- GameRequest
- GameResponse
- GameRequestTable
- Request/Response codes
- Running Order

Utilities

- DataReader
  - Reads data from datagrams
  - Need to know the order of datagram or you'll read it wrong.
- GamePacket
  - Writes the datagram to be sent to the client.
  - Order must be right or the client will get bad information.

GameRequest

- GameRequest is an abstract class
- Make classes that extend this class to handle requests sent by the clients.
- Must @override parse() and doBusiness
- Creates a list of responses for the client.

GameRequest

- parse()
  - Use DataReader's static functions to read off the datagrams data and store them into variables.
- doBusiness()
  - Set variables for the response datagram and make any additional responses for clients to be put in their queues.

GameResponse

- GameResponse is an abstract class
- Make classes that extend this to create responses to be sent to client side.
- Have setters for the variables that may have to change.
- Must @override constructResponseInBytes()

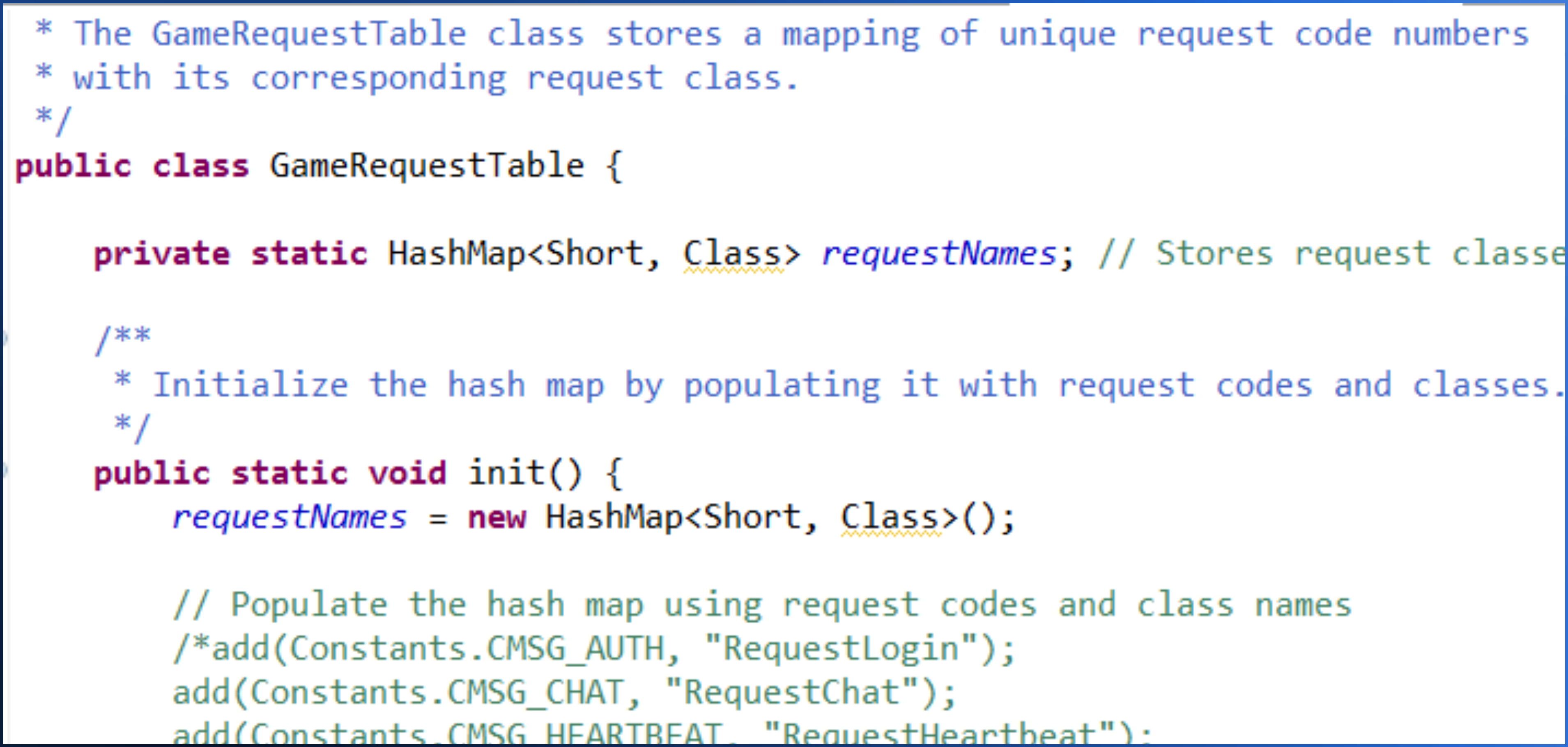
GameResponse

- constructResponseInBytes()
  - Start with instance of GamePacket
    - Constructor takes responseCode as a parameter
  - Use public methods to add additional data
    - addFloat()
    - addInt32()
    - Etc.
  - Returns instance of GamePacket.getBytes()

GameRequestTable

- This class allows the server to know which GameRequest subclass to use with regard to the request code.
- Whenever you create a new GameRequest subclass add its request code and the subclass name to the hash table.

GameRequestTable

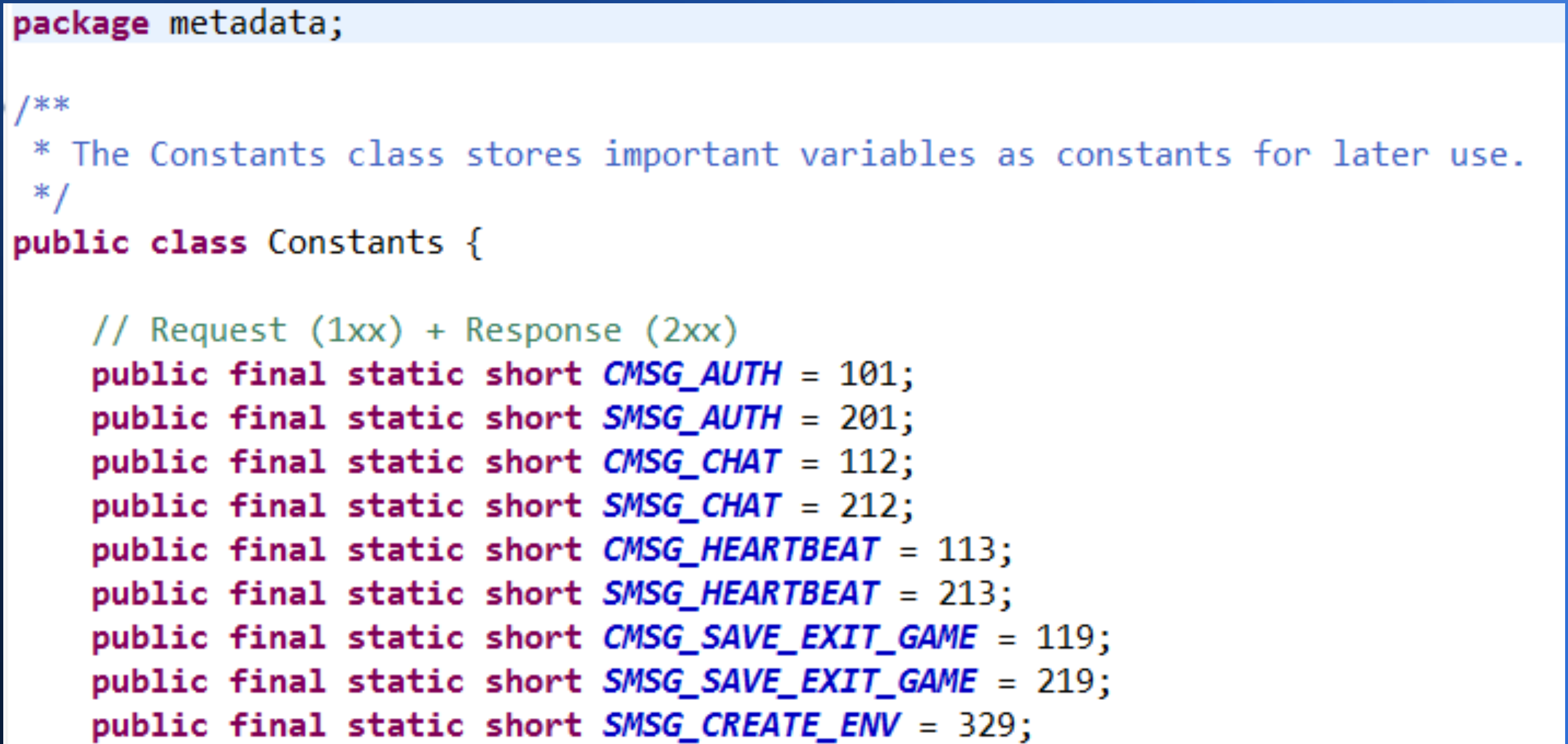


```
 * The GameRequestTable class stores a mapping of unique request code numbers
 * with its corresponding request class.
 */
public class GameRequestTable {

    private static HashMap<Short, Class> requestNames; // Stores request classe

    /**
     * Initialize the hash map by populating it with request codes and classes.
     */
    public static void init() {
        requestNames = new HashMap<Short, Class>();

        // Populate the hash map using request codes and class names
        /*add(Constants.CMSG_AUTH, "RequestLogin");
        add(Constants.CMSG_CHAT, "RequestChat");
        add(Constants.CMSG_HEARTBEAT, "RequestHeartbeat");
```

Request/Response Codes

- Constants Class holds all the request and response codes. Whenever you create a new subclass you should add its request or response code to the Constants.

Request/Response Codes



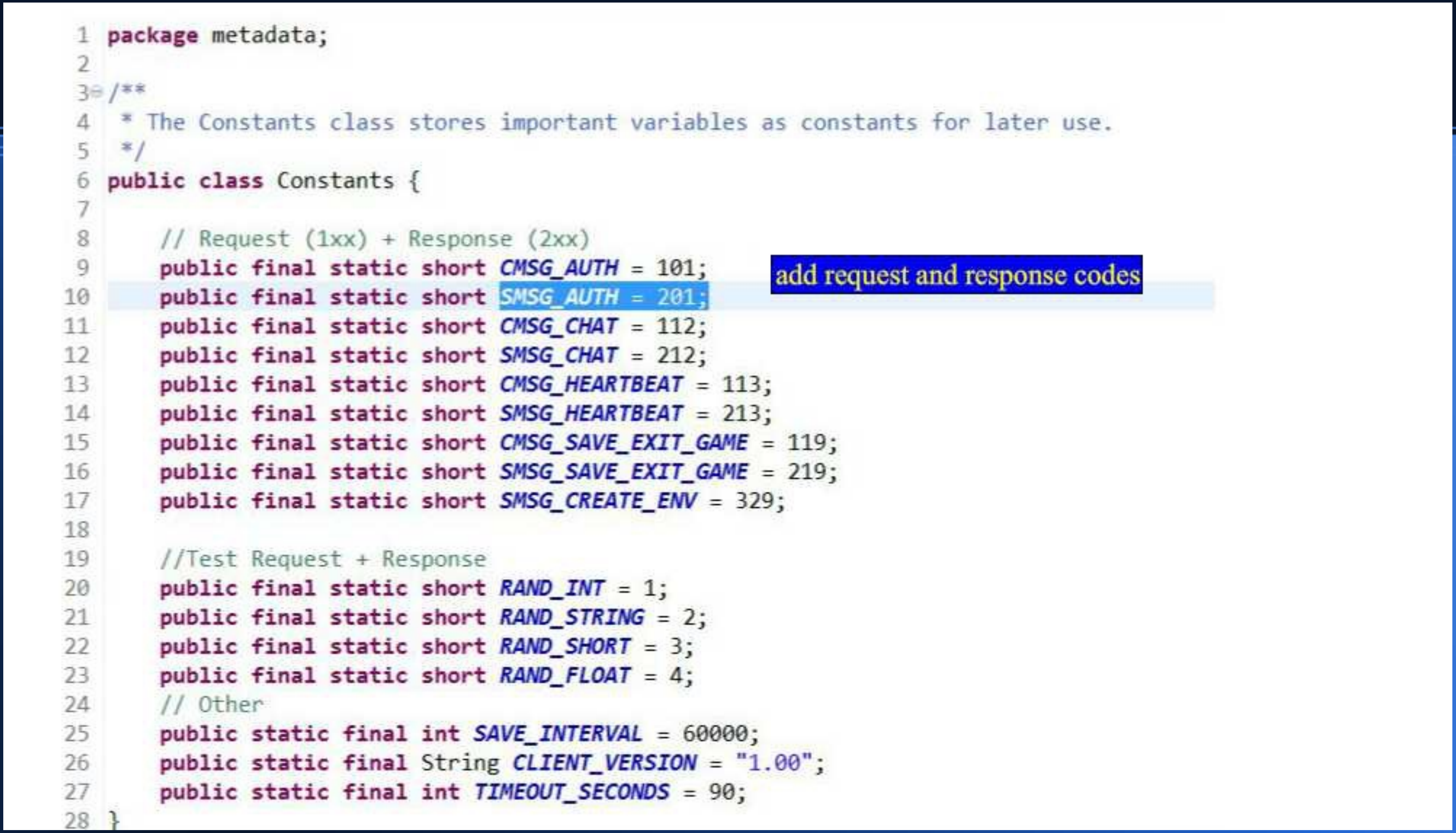
```
package metadata;

/**
 * The Constants class stores important variables as constants for later use.
 */
public class Constants {

    // Request (1xx) + Response (2xx)
    public final static short CMSG_AUTH = 101;
    public final static short SMSG_AUTH = 201;
    public final static short CMSG_CHAT = 112;
    public final static short SMSG_CHAT = 212;
    public final static short CMSG_HEARTBEAT = 113;
    public final static short SMSG_HEARTBEAT = 213;
    public final static short CMSG_SAVE_EXIT_GAME = 119;
    public final static short SMSG_SAVE_EXIT_GAME = 219;
    public final static short SMSG_CREATE_ENV = 329;
```

Running Order

- Start GameServer
- When connection is made creates instance of GameClient
- GameClient listens to connection for datagrams
- After receiving datagram takes request code and matches it to GameRequest subclass
- GameRequest object gets the data stream then uses parse()

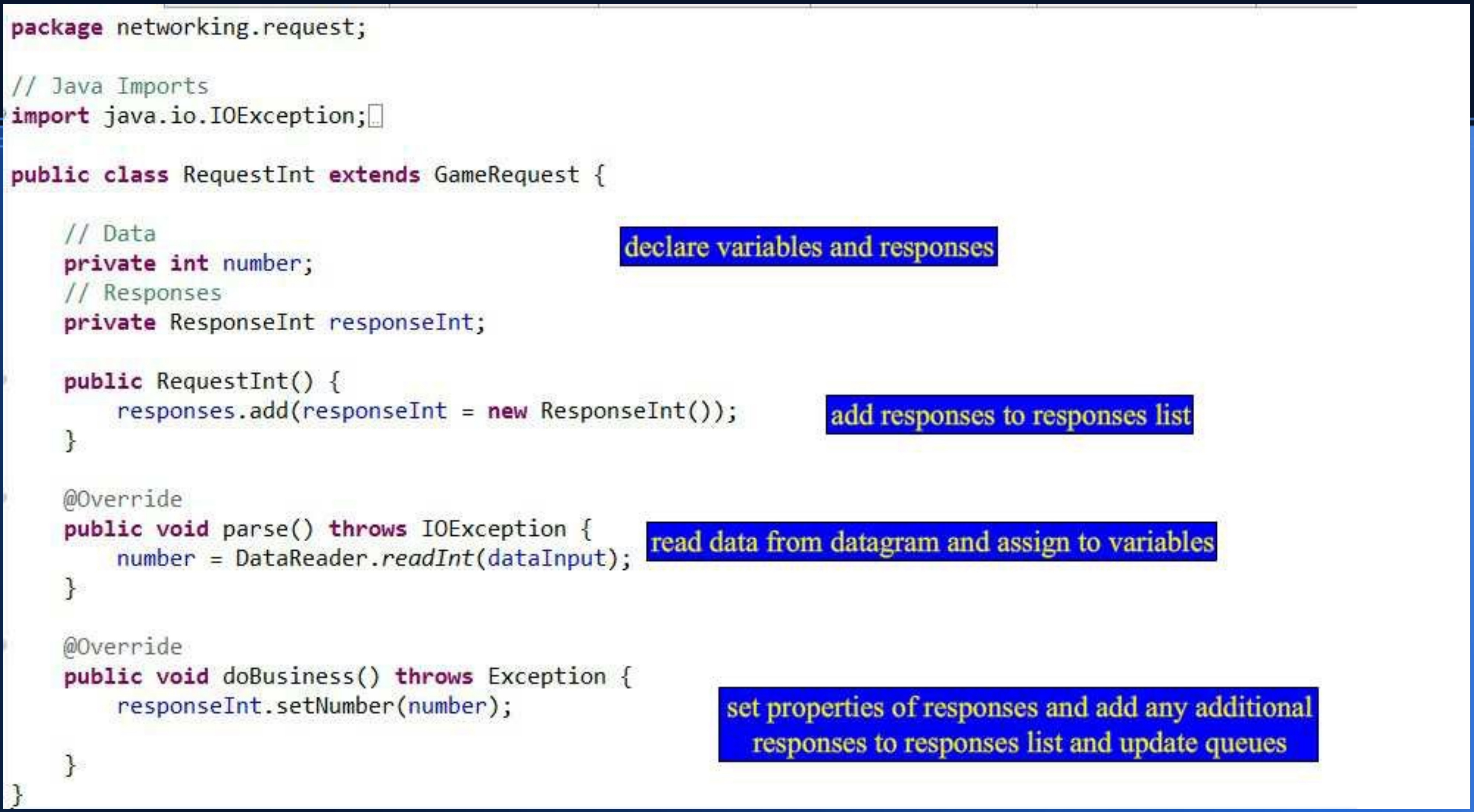
Running Order

- After data is parsed GameClient runs GameRequest's doBusiness method
- GameResponses are created and put into either the response list of the GameRequest or the update queue of other GameClients.



```java
package metadata;

/**
 * The Constants class stores important variables as constants for later use.
 */
public class Constants {

    // Request (1xx) + Response (2xx)
    public final static short CMSG_AUTH = 101;
    public final static short SMSG_AUTH = 201;
    public final static short CMSG_CHAT = 112;
    public final static short SMSG_CHAT = 212;
    public final static short CMSG_HEARTBEAT = 113;
    public final static short SMSG_HEARTBEAT = 213;
    public final static short CMSG_SAVE_EXIT_GAME = 119;
    public final static short SMSG_SAVE_EXIT_GAME = 219;
    public final static short SMSG_CREATE_ENV = 329;

    //Test Request + Response
    public final static short RAND_INT = 1;
    public final static short RAND_STRING = 2;
    public final static short RAND_SHORT = 3;
    public final static short RAND_FLOAT = 4;
    // Other
    public static final int SAVE_INTERVAL = 60000;
    public static final String CLIENT_VERSION = "1.00";
    public static final int TIMEOUT_SECONDS = 90;
}
```

add request and response codes



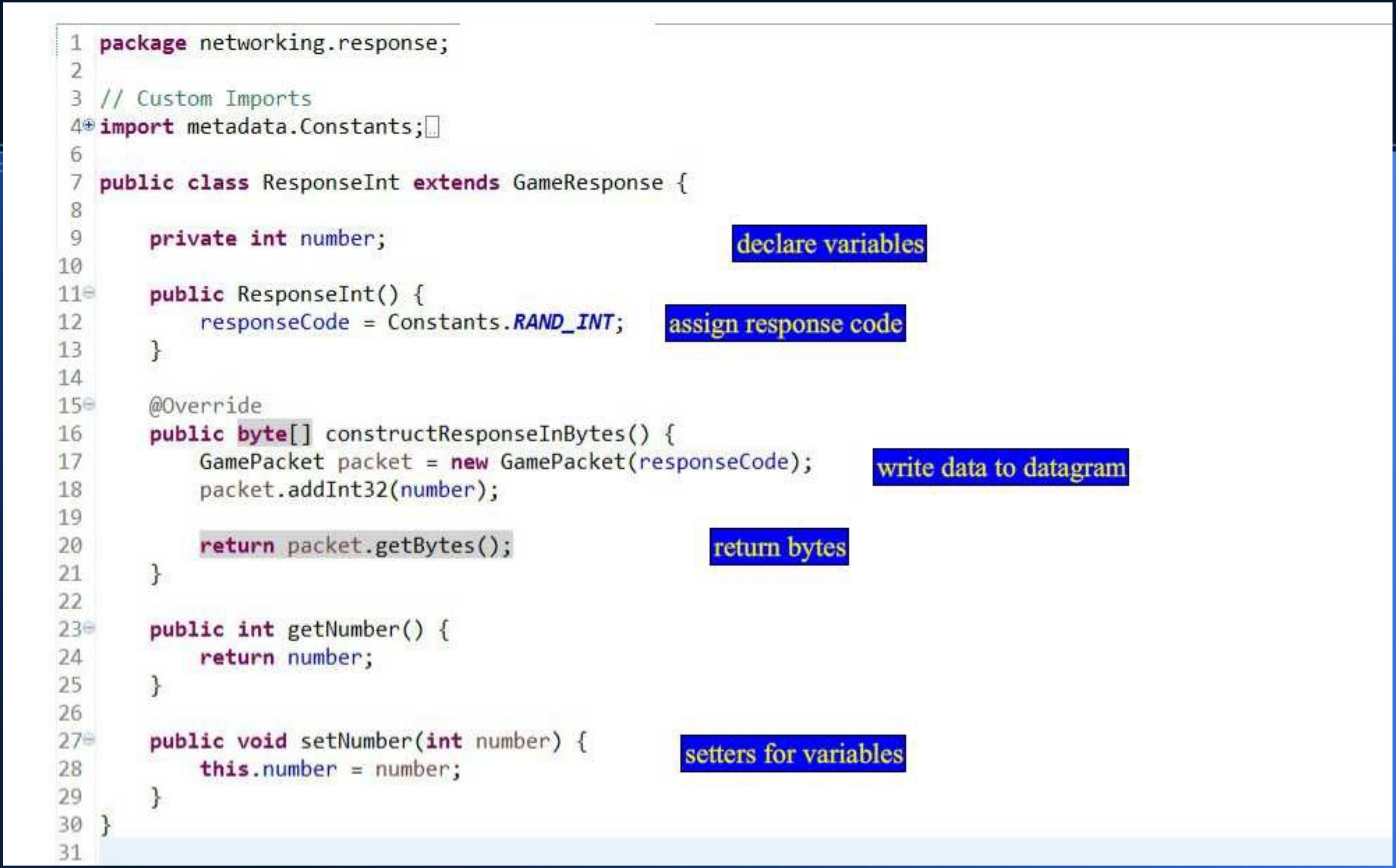
```java
package networking.request;

// Java Imports
import java.io.IOException;

public class RequestInt extends GameRequest {

    // Data
    private int number;
    // Responses
    private ResponseInt responseInt;

    public RequestInt() {
        responses.add(responseInt = new ResponseInt());
    }

    @Override
    public void parse() throws IOException {
        number = DataReader.readInt(dataInput);
    }

    @Override
    public void doBusiness() throws Exception {
        responseInt.setNumber(number);

    }
}
```

declare variables and responses

add responses to responses list

read data from datagram and assign to variables

set properties of responses and add any additional responses to responses list and update queues



```java
package networking.response;

// Custom Imports
import metadata.Constants;□

public class ResponseInt extends GameResponse {

    private int number;

    public ResponseInt() {
        responseCode = Constants.RAND_INT;
    }

    @Override
    public byte[] constructResponseInBytes() {
        GamePacket packet = new GamePacket(responseCode);
        packet.addInt32(number);

        return packet.getBytes();
    }

    public int getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }
}

```

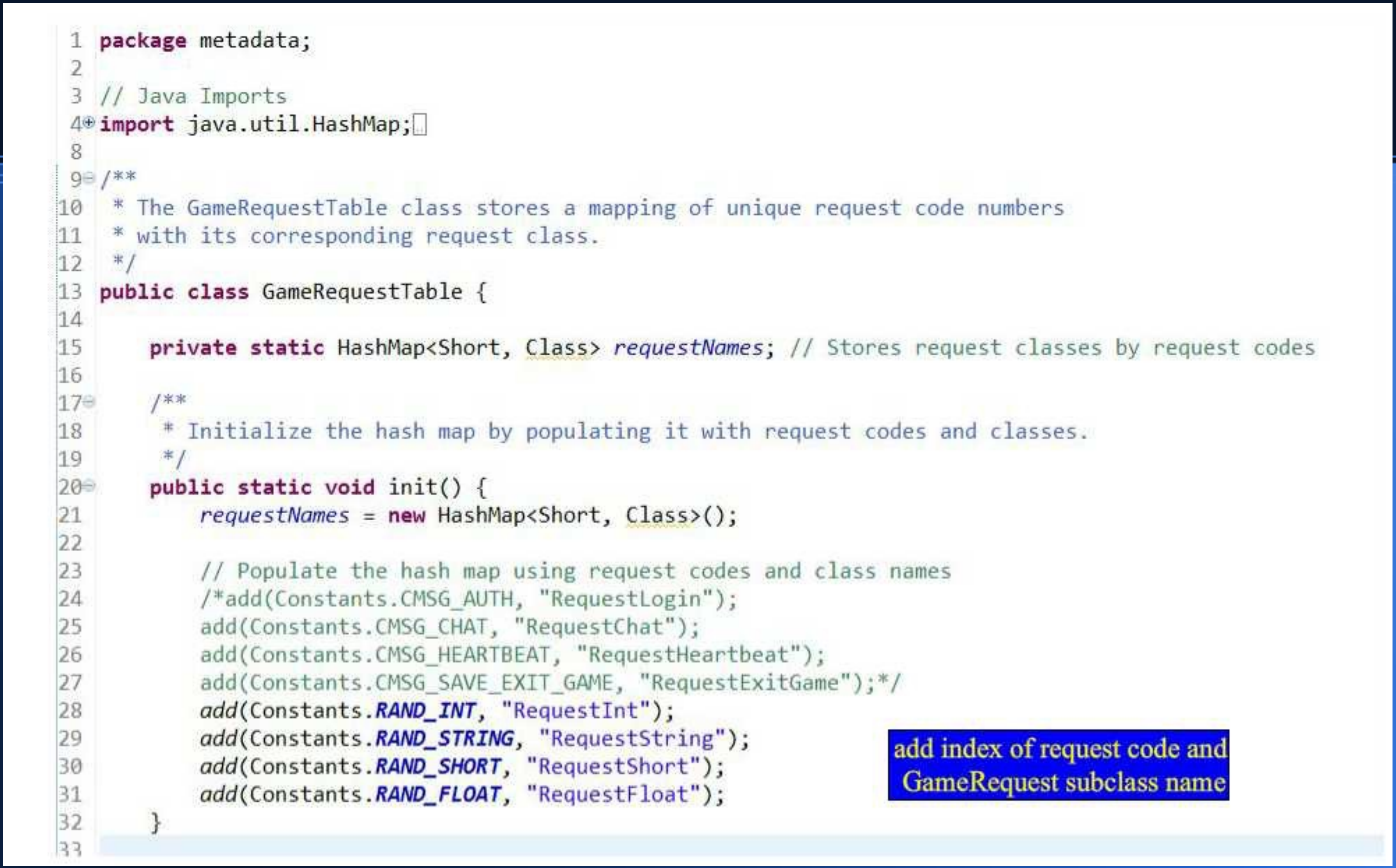
declare variables

assign response code

write data to datagram

return bytes

setters for variables



```java
package metadata;

// Java Imports
import java.util.HashMap;

/**
 * The GameRequestTable class stores a mapping of unique request code numbers
 * with its corresponding request class.
 */
public class GameRequestTable {

    private static HashMap<Short, Class> requestNames; // Stores request classes by request codes

    /**
     * Initialize the hash map by populating it with request codes and classes.
     */
    public static void init() {
        requestNames = new HashMap<Short, Class>();

        // Populate the hash map using request codes and class names
        /*add(Constants.CMSG_AUTH, "RequestLogin");
        add(Constants.CMSG_CHAT, "RequestChat");
        add(Constants.CMSG_HEARTBEAT, "RequestHeartbeat");
        add(Constants.CMSG_SAVE_EXIT_GAME, "RequestExitGame");*/
        add(Constants.RAND_INT, "RequestInt");
        add(Constants.RAND_STRING, "RequestString");
        add(Constants.RAND_SHORT, "RequestShort");
        add(Constants.RAND_FLOAT, "RequestFloat");
    }

```

add index of request code and GameRequest subclass name

Happy Coding!

- When creating a GameRequest subclass
  - Add request code to Constants class
  - Add request code and subclass name to GameRequestTable
- When creating a GameResponse subclass
  - Add response code to Constants class