BGN: 2191A

P4

Q3

a.

```
import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.io.IOException;
import java.net.Socket;
import java.nio.file.Path;
import java.nio.file.Paths;

class Main {

    public static final int PORT = 1991;

    public static void main(String[] args) throws IOException
{
        if (args.length < 1) {
            System.err.println("Usage: server [directory]");
            System.exit(1);
        }
        String directory = args[0];
        ServerSocket socket = new ServerSocket(PORT);
        while (true) {
            Socket client = socket.accept();
            BufferedReader in = new BufferedReader(
                new InputStreamReader(
                    client.getInputStream()
                )
            );

            String req = in.readLine();
            if (req == null) {
                client.close();
            }

            Path filePath = Paths.get(directory, req);
            BufferedInputStream bis = null;
            OutputStream os = null;

            try {
                File file = new File(filePath.toString());
                byte[] buffer = new byte[(int) file.length()];
                FileInputStream fis = new
FileInputStream(file);
                bis = new BufferedInputStream(fis);
                bis.read(buffer, 0, buffer.length);
                os = client.getOutputStream();
                os.write(buffer, 0, buffer.length);
                os.flush();

            } finally {
                if (bis != null)
                    bis.close();
                if (os != null)
                    os.close();
                client.close();
            }
        }
    }
}
```

b. A disadvantage is that the server can only handle one client connection at a time. If there are many clients who want to connect, or one of the clients requests a large file which takes a long time to send, the other clients might have to wait a long time for their requests to be processed. A mitigation to this is to use multithreading, spinning up a different thread to handle each incoming client connection.

c. When the client receives data from stdin, instead of sending the string directly to the server, it stores the request data in an instance of the following class:

   class Request implements Serializable {

```
    private static final long serialVersionUID = 1L;

    public String filename;
    /* potentially other data… */
}
```

The client sends this object to the server which reconstructs the object and locates the requested file.

When the server reads the contents of the file into the buffer, instead of directly sending the contents of this buffer to the client, it first stores this buffer (and the filename) into an instance of the following class:

```
class SerializableFile implements Serializable {

    private static final long serialVersionUID = 1L;

    public String filename;
    public byte[] data;
    /* potentially other metadata… */
}
```

This object is then sent back to the client who reconstructs it and processes the file as desired.

An advantage of this approach is that the system is now much more extensible. It is easy for the client to send additional data with its request and the server with its response.

A major disadvantage of this approach is security. The classes sent between the client and the server could contain malicious code which might be executed by the receiver, expecting it to be legitimate. This could lead to an arbitrary code execution vulnerability.

d. This service could be modified by having the client include additional data in their request. For example, an "Action" enum value ("GET", "POST", "DELETE", etc.) which the server could act upon.

   A security fix would be to ensure on the server-side that the requested file path cannot climb out of the directory specified on the command line, e.g., by filtering for ".." and by disallowing symlinks to be created.

   Another security feature would be to use public-key cryptography to communicate between the client and the server, so no eavesdropper could read the files being transmitted.

Also, the server should be able to send serialized objects encoding error messages to the client (e.g., “file does not exist”, or “failed to write file”).