Program 2: The Chargen++ Protocol

Dr. William Kreahling

October 12, 2016

1 Chargen Client

1.1 Overview

The purpose of this assignment is to give you some experience implementing application-layer protocols using the services provided by the transport layer. After completing this assignment, you will have experience reading and understanding requests for comments (RFCs), and you will understand how to implement TCP and UDP clients using the standard Java API.

Your task is to develop an application that implements the client-side of the chargen protocol (RFC 864). The application will support both the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) versions of the protocol. For the TCP version of this assignment your TCP server must use the modified RFC 864 located on the class web page. Your TCP client must support both protocols!

1.2 Client Design

To receive full credit on this assignment, your program must not only function correctly, but must also be designed well and written in Java. This section describes the structure you should follow to implement your programs.

Figure 1 includes a UML class diagram representing the relationships between the classes and interfaces that are part of the application you will develop. The following subsections describe these classes and interfaces.

1.2.1 Class `ChargenClientDriver`

`ChargenClientDriver` is the entry point of your application. It includes one method: `main()`. The program must accept either two, three, or four parameters. The first parameter must be either the string "TCP" or the string "UDP" (case should not matter). The program will use this parameter to determine which client implementation to use. The second parameter must be either the IP address or host name of the server to which to connect. The third, optional, parameter is the port number to which to connect. If this parameter is not supplied, the default value must be 19 (the "well-known" chargen port). The chargen server on Polaris is running, and your client must be able to connect to it. The fourth, optional, parameter is a flag to send to your version of the Chargen server. The flags are defined in the RFC listed on the main web page. In the absence of any flag, the client and server use default behaviors. *You may make changes to this design with consultation of the instructor.*

A note on subtleties: *The Chargen server running on polaris follows the original RFC (864) So when your TCP client connects to any Chargen server using port 19 your TCP client should use the original rules as specified in RFC 864. When your TCP client connects to a Chargen server on a port other than 19, you must use the protocol defined on the class web page. Your TCP server will use the protocol defined on the class web page because you are not* ***allowed*** *to use port 19.*

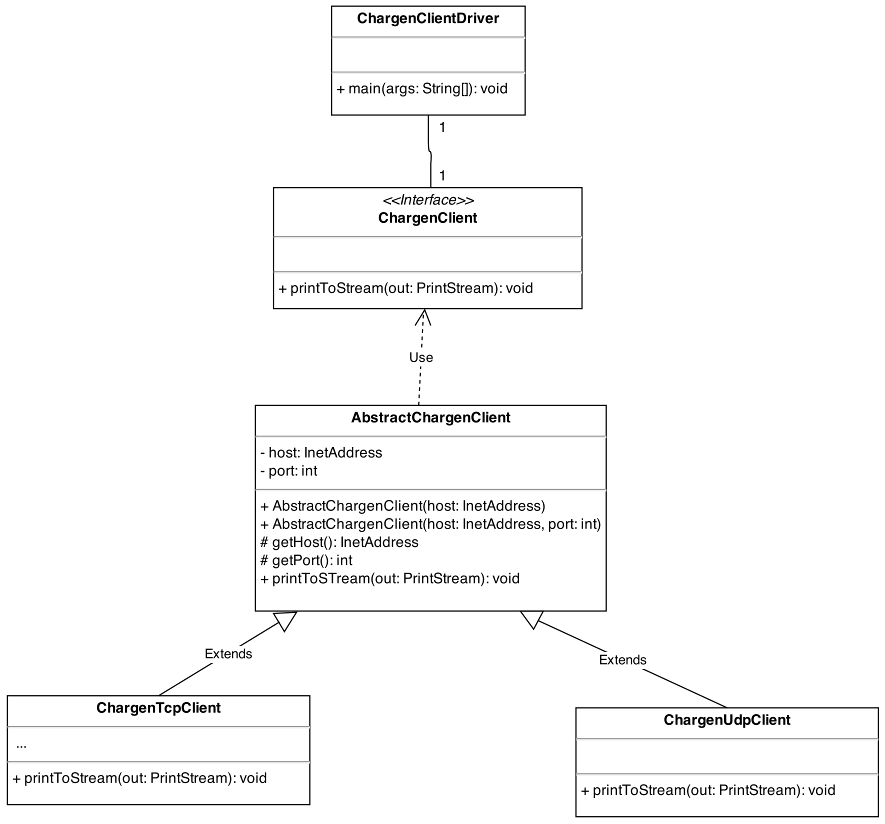
ChargenClientDriver
+ main(args: String[]): void
1
1
<<Interface>>
ChargenClient
+ printToStream(out: PrintStream): void
Use
AbstractChargenClient
- host: InetAddress
- port: int
+ AbstractChargenClient(host: InetAddress)
+ AbstractChargenClient(host: InetAddress, port: int)
# getHost(): InetAddress
# getPort(): int
+ printToSTream(out: PrintStream): void
Extends
Extends
ChargenTcpClient
...
+ printToStream(out: PrintStream): void
ChargenUdpClient
+ printToStream(out: PrintStream): void

Figure 1: Chargen Client UML Diagram

1.2.2 Interface `ChargenClient`

ChargenClient is the interface to a chargen client implementation. The interface includes one method: printToStream() which takes a PrintStream as a parameter. Classes implementing this interface will provide an implementation of this method that prints the received character sequence to the supplied stream.

1.2.3 Class `AbstractChargenClient`

AbstractChargenClient is an abstract class that implements the ChargenClient interface. This class includes attributes that are common to all Chargen client implementations, as well as protected methods that enable subclasses to access those attributes.

1.2.4 Class `ChargenTcpClient`

ChargenTcpClient is a class that extends AbstractChargenClient and provides a concrete implementation of the printToStream() method using TCP. Data received from the remote host is printed to the specified PrintStream.

1.2.5 Class `ChargenUdpClient`

ChargenTcpClient is a class that extends AbstractChargenClient and provides a concrete implementation of the printToStream() method using UDP. Data received from the remote host is printed to the specified PrintStream.

2 Chargen Server

2.1 Server Design

You must follow the following design for the server portion of the assignment.

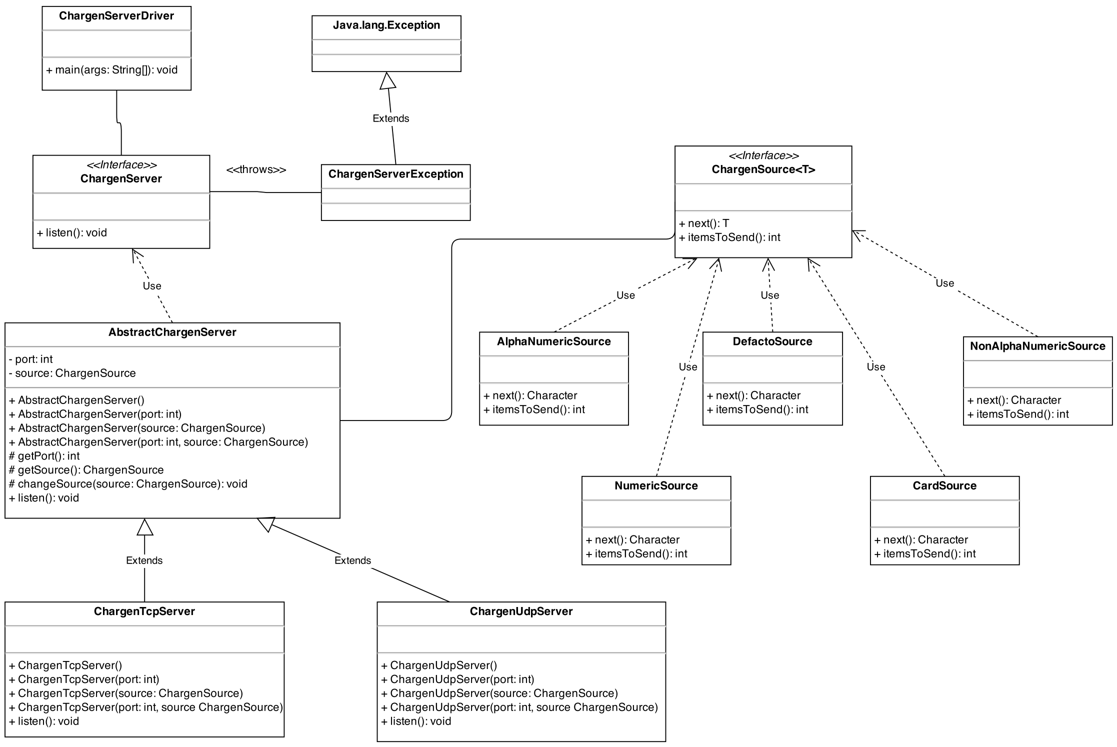

Figure 2: Chargen Server UML Diagram

Figure 2 includes a UML class diagram representing the relationships between the classes and interfaces that are part of the application you will develop. The following subsections describe these classes and interfaces.

2.1.1 Class `ChargenServerDriver`

`ChargenServerDriver` provides the entry point of your application. It includes one method: `main()`. The program must accept either one or two parameters. The first parameter must be either the string "TCP" or the string "UDP". The program will use this parameter to determine which server implementation to use. The second, optional, parameter is the port number on which the server with listen for incoming connections. If this parameter is not supplied, the default value must be 19 (the "well-known" chargen port).

2.1.2 Interface `ChargenServer`

`ChargenServer` is the interface to a chargen server implementation. The interface includes a method: `listen()`. Classes implementing this interface will provide an implementation of this method. `listen()` will listen for requests from clients;

2.1.3 Class `AbstractChargenServer`

`AbstractChargenServer` is an abstract class that implements the `ChargenServer` interface. This class includes attributes that are common to all Chargen server implementations, as well as protected methods that enable subclasses to access those attributes. For all constructors, if no ChargenCharacterSource is supplied, DefactoChargenCharacterSource should be used by default. getCharacterSource() will return a ChargenSource – an object used to generated the sequences produced by the server. changeSource(ChargenSource) will accept a new ChargenSource that will change the one used to generate data.

2.1.4 Class **ChargenTcpServer**

ChargenTcpServer is a class that extends AbstractChargenServer and provides a concrete implementation of the listen() method using TCP. The method repeatedly waits for a client to connect and responds appropriately. This will be a *serial* server.

2.1.5 Class **ChargenUdpServer**

ChargenTcpServer is a class that extends AbstractChargenServer and provides a concrete implementation of the listen() method using UDP. The method repeatedly waits for a datagram from a client and responds appropriately. This will be a *serial* server.

2.1.6 Class **ChargenServerException**

The exceptions generated by the various concrete implementations of the ChargenServer interface's listen() method vary across implementations. For example, a TCP implementation might throw exceptions related to TCP socket errors; a UDP implementation might throw other exceptions. We do not want users of our interface to have to explicitly catch all these exceptions. (Remember, an interface presents an *abstract* view of the component.) If a user is using only the UDP implementation, why make them catch TCP-related exceptions? Instead, to provide an abstract interface, we introduce ChargenServerException. Classes that provide concrete implementations of listen() will catch the protocol-specific exceptions, wrap those exceptions in a ChargenServerException, and then throw that exception. This approach passes along all the error information in the wrapped object, but keeps the interface "clean".

2.1.7 Interface **ChargenSource**

The Chargen RFC does not specify the format of the character sequence produced by a server application. Therefore, servers may choose to vary the character sequence. The ChargenSource provides the abstract interface to a component that produces some sequence for chargen. I have broadened the scope of the Chargen server to return a sequence of Objects instead of characters.

2.1.8 Class **DefactoChargenSource**

The DefactoChargenSource class implements the ChargenSource interface and produces the defacto standard character sequence produced by chargen servers.

2.1.9 Class **ChargenSource**

The Alpha NumericChargenSource class implements the ChargenSource interface and produces a character sequence of alphanumeric characters, produced by chargen servers. NumericChargenSource class implements the ChargenSource interface and produces a character sequence of numeric character NonAlphaNumericChargenSource class implements the ChargenSource interface and produces a character sequence of non-alphanumeric characters. CardSource interface and produces a sequence of Card objects. Card must be comprised of two enumerations once for the suite and one for the value. You must send a randomly chosen card from a standard deck of fifty-two cards. You will not send a duplicate card until fifty-two cards have been sent. Cards should be sent back in random order.

2.1.10 More on`CardSource`

Card Source must model a playing card in a standard deck of cards. It must be comprised of two enumerations (The Card class and both enumerations should be defined in package common). The enumerations must contain getter and setter methods (as appropriate), as well as a toString methods that nicely format the enumeration. You must use correct naming conventions for enumerations.

3 Details

You must use packages for your program. There will be a client package, a common package, and a server package. All files needed by both the client and the server **must** be kept in a separate directory/package. There will be no duplication of files.

4 Suggestions

The following classes in the standard Java API will could you complete this assignment:

- java.net.ServerSocket
- java.net.InetAddress
- java.net.Socket
- java.net.DatagramPacket
- java.net.DatagramSocket

Remember: **Unless you are in main(), do not catch exceptions if you cannot recover from them! Let the method throw the exception.** The only "exception" here is when you catch a low-level exception in order to "wrap" it with a higher-level exception that you in turn throw.

5 Hand-In Instructions

This assignment is due by 11:59 PM on Wednesday, November 2. Submit only the Java source files. You must submit the assignment using the *handin* command on agora. Handin works as follows:

```
handin.<course#>.<section#>  <assignment#>  <files>
```

6 Notes on Collaboration

I encourage you to work in teams of two for this assignment. **Please do not consult *anyone* other than me, or your teammate, on *any* aspect of this assignment**.