Aufgabe 1a)

Es wurde kein Packet definiert, da dies durch die geringe Komplexität des Programms nicht erforderlich ist.

Die Parameterübergabe erfolgt im Muster „-x Wert“, um eine beliebige Reihenfolge zu ermöglichen. Dabei können theoretisch auch Parameter doppelt vorkommen, die sich dann überschreiben. Für die Hilfe wurde auch die UNIX-Langform zugelassen.

```
for(int i=0; args.length>i; i++)
{
	if(args[i].equals("--help") || args[i].equals("-h"))
	{
		// ...
	}
	else if( ...
```

Die Datenübertragung beschränkt sich auf den reinen Text „Hello World“, die Verbindung wird danach geschlossen, der Server sucht in einer Endlosschleife nach weiteren Clients und der Client beendet sich.

Aufgabe 1b)

Die Ausführenden Klassen sind im Paket alpv_ws1415.ub1.webradio.webradio.server bzw. alpv_ws1415.ub1.webradio.webradio.client aufgeteilt, je nach funktioneller Zuordnung. Desweiteren wurde in alpv_ws1415.ub1.webradio.ui eine neue Klasse Log angelegt, die die Aufrufe von System.out.println() kapselt und so eine Steuerung des Ausgabeumfangs zulässt, was für die Entwicklung sehr hilfreich war.

Server-Seite

Der Server besteht aus 8 Klassen (darunter Player.Streamer und ClientHandler.ClientSender), von denen 4 im Betrieb als eigener Thread laufen (ConnectionServer, ConsoleListener, Player.Streamer und ClientSender, letzterer pro Sendeauftrag ein Thread).

ConnectionServer und ConnectionServerTCP

Der ConnectionServer ist für das Annehmen und Aufbauen von Verbindungen zuständig, er läuft in einem eigenen Thread. Er startet außerdem den Player und die ServerUI (hier ConsoleListener). ConnectionServer ist abstract und hat Ableitungen für die einzelnen Verbindungsprotokolle, zur Zeit ist ConnectionServerTCP implementiert.

Der ConnectionServer wartet auf neue Verbindungsanfragen von Clients, erstellt einen Socket und reicht diesen gekapselt in einem ClientHandler an den Player weiter, der die restliche Kommunikation übernimmt.

```
Socket client = socket.accept();

// Erzeuge neuen ClientHandler
ClientHandler ch = new ClientHandlerTCP(client);
player.addClientHandler(ch);
```

ClientHandler, ClientHanlderTCP und ClientHandlerTCP.ClientSender

Der ClientHandler hat ebenfalls Ableitungen für verschiedene Verbindungsprotokolle, hier ClientHandlerTCP. Er ist die Schicht zwischen dem Socket und dem Player, der nur die Daten bereitstellt. Später übernimmt er auch das Entgegennehmen von Chat-Nachrichten, um diese an den Chat-Server weitergeben zu können.

Der protokollspezifische ClientHandler erzeugt für jede Übertragung einen neuen ClientSender, der die Datenpackete überträgt und als seperater Thread läuft, um den aufrufenden Prozess nicht zu blockieren, falls die Verbindung stockt.

```
Thread t = new Thread(new ClientSender(audioData, this));
t.start();
```

Wird im ClientSender ein Fehler festgestellt, wird die Verbindung des ClientHandlers beendet.

```
catch(IOException e)
{
	context.close();
}
```

Player und Player.Streamer

Der Player hat 2 Zuständigkeiten: Zum einen verwaltet er die verbundenen Clients, zum anderen kümmert er sich darum, dass die Audiodateien gelesen und in passenden Teilen gesendet werden. Die letzte Aufgabe wird zu großen Teilen von seiner Unterklasse Player.Streamer übernommen, die in einem eigenen Thread läuft. Sie kommunizieren durch ihre gemeinsamen Ressourcen. Der Streamer sendet dabei immer eine festgelegte Zeitspanne an AudioDaten, damit Server und Clients in etwa synchron bleiben. Diese berechnet sich aus den zu sendenden Sekunden *2 (da Stereo), die in Bytes umgerechnet werden. Dazu wird ein kleiner Overhead addiert, um Verbindungslücken auszugleichen, auch wenn dadurch der Stream etwas schneller ist als die abspielenden Clients.

```
bufferSize = (int) (sendBufferMillis / 1000 * audioFormat.getSampleRate() *
audioFormat.getSampleSizeInBits() / 8) + bufferOverhang;
```

Damit die verschiedenen geteilten Ressourcen nicht durch gleichzeitigen Zugriff aus 2 Threads inkonsistent werden und Iteratoren failen, wurden kritische Code-Abschnitte synchronisiert und passende Lock-Objekte angelegt.

Beim Schließen schließt der Player auch alle verknüpften ClientHandler und beendet den Streamer-Thread. Das Schließen blockiert, bis der Streamer beendet ist.

Playlist und PlaylistItem

Um die Wiedergabeliste zu organisieren, gibt es eine Playlist, die PlaylistItems, bestehend aus Dateipfad und Formatinformationen, speichern kann. Die Playlist ist eine Queue, die gespielte Tracks wieder ans Ende anhängt und so beliebig lang Musik liefert.

```
PlaylistItem song = playlist.remove();
playlist.add(song);
```

Sie kann außerdem neue Stücke aufnehmen.

ConsoleListener

Der ConsoleListener läuft in einem eigenen Thread und übernimmt die Kommunikation auf stdin und stdout. Über Befehle kann der Server beendet werden oder ein neues Lied hinzugefügt werden.

Client-Seite

RadioClient und RadioClientTCP

RadioClientTCP ist eine Variante von RadioClient. Sie kann sich mit einem Server verbinden und Musik empfangen, bis die Musik abbricht. Ist die Verbindung unterbrochen, läuft der Client weiter, um sich per Connect neu verbinden zu können.

Bis zur Übertragung von Metadaten versucht die Klasse, das AudioFormat aus der Datei direkt auszulesen - das wird bei der nächsten Aufgabe korrigiert.

```
File testFile = new File("test.wav");
AudioFileFormat aff = AudioSystem.getAudioFileFormat(testFile);
AudioPlayer player = new AudioPlayer(aff.getFormat());
```