technische universität
dortmund

Fakultät für Informatik
Lehrstuhl 4

Egor Kudrjaschow, Henning Brümmer, Andreas Blume
Nils Dunker, Felix Homa, Benedikt Maus, Dennis Ziebart, Bastian Korte

Wintersemester 2019/20

Rechnernetze und verteilte Systeme

Programmieraufgabe

Ausgabe: 09. Dezember 2019 **Abgabe:** 13. Januar 2020 **Besprechung:** 27. Januar – 31. Januar 2019

Inhalt

1 Vorwort

Da in den Vorjahren häufig Fragen gestellt wurden, die explizit in der Aufgabe erklärt wurden, hier die Bitte, die Aufgabe **vollständig** zu lesen.

Es empfiehlt sich eine vollständige Entwicklungsumgebung zu nutzen. Das heißt, kein BlueJ, sondern Eclipse oder IntelliJ IDEA. siehe Entwicklungsumgebungen auf dem Hilfsblatt.

Auf dem Hilfsblatt gibt es auch weitere Hilfestellungen zum Lösen der Aufgabe.

Falls dennoch Fragen zur Aufgabe bestehen:

- Moodle Diskussionsforum (Regeln im Forum beachten)
- HelpDesk

Egor Kudrjaschow, Henning Brümmer, Andreas Blume
Nils Dunker, Felix Homa, Benedikt Maus, Dennis Ziebart, Bastian Korte

2 Anforderungen

Die Abgabe muss

- in Java 11 oder aktueller[1] gelöst werden.
- als ZIP-Archiv erfolgen (NICHT RAR!).
- der Ordnerstruktur wie in der Vorlage entsprechen.
- über das Moodle[2] erfolgen.
- mit JavaDoc dokumentiert sein. (siehe JavaDoc im Hilfsdokument; sonst bis zu 25% Punktabzug, der Export ist nicht notwendig, die Dokumentation im Code ist ausreichend.)

Die Abgabe darf

- in Gruppen von bis zu 3 Personen durchgeführt werden.
- Dafür die gruppe.txt bearbeiten mit Zeilen nach folgendem Schema:

  ```
  Vorname; Nachname; Matrikelnummer
  ```

Mit 0 Punkten wird werden Abgaben bewertet, die

- nicht Java 11 oder aktueller gelöst sind.
- keine Quelltext haben (.java-Dateien fehlen)
- nicht kompilierbar sind. (Besuchen Sie ggf. den HelpDesk frühzeitig)
- zusätzliche Programm-Bibliotheken abseits der Java-Standard-Bibliothek[3] nutzen.[4]
- Plagiate sind! (Original und Kopie)[5]

Zu beachten:

Die Main-Klasse (**edu.udo.cs.rvs.SSDPMain**) darf NICHT bearbeitet werden. Änderungen in dieser Klasse werden während dem Testen automatisch verworfen.

Es gibt ein „Beispiel-Gerät“ (ein SSDP-Peer) im Moodle, mit dem Sie feststellen können, ob Ihr Client Anfragen empfängt und sendet.

Zu starten ist das „Beispiel-Gerät“ mit „java -jar ssdpPeer.jar“.

1 https://www.oracle.com/technetwork/java/javase/downloads/index.html
2 https://moodle.tu-dortmund.de/course/view.php?id=17038
3 https://docs.oracle.com/en/java/javase/11/docs/api/ (üblicherweise wird nur Inhalt des Moduls **java.base** benötigt)
4 Apache Commons, Log4J. usw. sind also nicht erlaubt.
5 Es ist uns nicht möglich festzustellen, welche der Abgaben das Original und welche eine Kopie ist.

Egor Kudrjaschow, Henning Brümmer, Andreas Blume
Nils Dunker, Felix Homa, Benedikt Maus, Dennis Ziebart, Bastian Korte

3 Aufgabenstellung

Im Rahmen dieser Aufgabe soll ein SSDP[6]-Peer (Simple Service Discovery Protocol) programmiert werden. Dieser soll nach lokalen Geräte suchen und eine Liste der bekannten Geräte verwalten. Die Implementierung muss nicht exakt der nachfolgenden Vorschläge entsprechen, jedoch die selbe Funktionalität mit den selben Befehlen haben!

Hierfür empfiehlt es sich 3 Threads zu nutzen:

- Der Listen-Thread, um Datagramm zu empfangen
- Der Worker-Thread, um die Empfangenen Datagramme zu verarbeiten
- Der User-Thread, um Eingaben des Nutzers zu lesen und entsprechende Aktionen durchzuführen

3.1 Listen-Thread

Zusammengefasst soll dieser **Thread** ein **MulticastSocket** auf Port 1900 öffnen, der Multicast-Gruppe „239.255.255.250“ beitreten und bis zum Programmende endlos Datagramme empfangen und dem Worker-Thread zur Verfügung stellen.

Ein MulticastSocket wird wie ein DatagramSocket geöffnet. Um einer Multicast-Gruppe beizutreten bietet ein MulticastSocket die Methode **joinGroup(InetAddress)** an. Eine **InetAddress** wird nicht wie normale Objekte initialisiert, stattdessen stellt die InetAddress-Klasse einige statische Methoden zum initialisieren eines solchen Objektes bereits. Für eine gegebene Adresse bietet sich hier die **getByName(String)**-Methode an.

Nachdem das MulticastSocket der Gruppe beigetreten ist, können Pakete wie bei einem DatagramSocket über die **receive**-Methode empfangen werden. Die empfangenen Datagramme sollten so schnell wie möglich in eine Liste (Empfehlung: **LinkedList**) aufgenommen werde, die der Worker-Thread abfragen kann. Zu beachten ist, dass der Zugriff auf geteilte Objekte (wie z.B. eben genannter Liste) synchronisiert werden muss (siehe **Threadsynchronisierung** im Hilfsdokument).

Der Thread soll sich zum Programmende beenden und dabei das MulticastSocket automatisch schließen.

3.2 Worker-Thread

Zusammengefasst soll dieser **Thread** die empfangenen **Datagramme** aus dem Listen-Thread verarbeiten und dem User-Thread mitteilen, welche Geräte gerade Dienste im Netzwerk anbieten.

Hierfür muss der Thread sowohl das Objekt des User-Threads als auch das Objekt des Listen-Threads kennen.

6 Angelehnt an: https://tools.ietf.org/html/draft-cai-ssdp-v1-03

Egor Kudrjaschow, Henning Brümmer, Andreas Blume
Nils Dunker, Felix Homa, Benedikt Maus, Dennis Ziebart, Bastian Korte

Auch der Worker-Thread soll bis zum Programmende in Endlosschleife laufen.

Als erstes muss geprüft werden, ob überhaupt Datagramme zu abarbeiten vorliegen. Wenn keine Datagramme vorliegen sollte der Thread einige Millisekunden schlafen um den Prozessor nicht mit unnötig vielen Prüfungen zu überlasten. Hierfür bietet sich die statische Methode **Thread.sleep(int)** an. Der Parameter gibt hierbei die Zeit in ms die gewartet werden soll. Ein Wert von 10ms ist üblicherweise ausreichend.

Wenn nun Datagramme vorliegen sollte man sich immer das älteste nehmen und aus der Liste entfernen (Threadsynchronierung!). Danach kann man die Daten des Datagramms auswerten.

Da SSDP-Pakete auf dem HTTPU-Standard (HTTP over UDP) aufbauen und dies ein Zeilenbasierter Standard ist, ist es zu empfehlen die Paketdaten mit Hilfe eines **BufferedReaders** einzulesen (siehe **BufferedReader** und **Datagramdaten als BufferedReader** im Hilfsdokument).

Ein Paket ist so aufgebaut, dass die erste Zeile den Typ des Pakets angibt. Danach eine beliebige Anzahl an Key-Value-Paaren, für die gilt: „Key: Value“. Nach dem Doppelpunkt folgt ein Leerzeichen, danach der Wert, welcher weitere Doppelpunkte enthalten kann, die dann Teil des Wertes sind. Jede Zeile enthält genau ein Paar, bis dass Ende des Pakets durch eine Leerzeile markiert wird. (**Hinweis: "example string".split(":", 2)** ist ggf. hilfreich um Key und Value von einander zu trennen. Rückgabe-Typ ist String[])

Ein Gerät, dass SSDP-Dienste anbietet kann auf zwei Arten antworten:

Entweder mit einer gerichteten Antwort (Unicast):

```
HTTP/1.1 200 OK                                 // Anfrage war OK
S: uuid:12345678-1234-1234-1234-123456789abc    // uuid des ANFRAGENDEN-Gerätes
ST: ge:fridge                                   // Der Service-Type
USN: uuid:abcde01-7dec-11d0-a765-00a0c91e6bf6   // uuid des ANTWORTENDEN-Gerätes
„leere Zeile“                  // Die leere Zeile schließt die Antwort ab
```

Entweder mit einer ungerichteten Antwort (Multicast):

```
NOTIFY * HTTP/1.1                                       // Mitteilung an alle ...
HOST: 239.255.255.250:1900                              // ... der SSDP-Gruppe
LOCATION: http://192.168.1.254:51777/rootDesc.xml       // irrelevant
SERVER: Debian/wheezy UPnP/1.1 MiniUPnPd/2.1            // irrelevant
NT: upnp:rootdevice                                     // Service-Type
USN: uuid:b3c152d8-blaa-5736-a8c3f32db7a42abdfe:upnp:rootdevice // UUID
NTS: ssdp:alive
„leere Zeile“
```

ACHTUNG: Dies sind nicht die einzigen Paketarten die Sie empfangen werden, jedoch die einzigen Paketarten die uns interessieren. Pakete die nicht in eines der beiden Schemata passen, sollen verworfen werden. Sie werden beispielsweise ihre eigene Suchanfrage empfangen. Die Kommentare sind ebenfalls nicht Teil des Pakets.

Die Informationen in den die uns interessieren sind die UUID und der angebotene Dienst-Typ. In der Unicast-Antwort sind diese in den USN und den ST zu finden. In der Multicast-Antwort sind diese in USN und **N**T zu finden. Zusätzlich gibt die Multicast-Antwort an, ob sich das Gerät an- oder abmeldet. Dies ist in NTS zu finden. Die möglichen Werte sind „ssdp:alive“ zum anmelden und „ssdp:byebye“ zum abmelden. Beim Abmelden ist der Dienst-Typ irrelevant. Einige Geräte bieten mehrere Dienst-Typen an, diese Geräte senden für jeden Dienst-Type ein separates Paket.

Wenn die Informationen gelesen sind, sollen sie an den User-Thread übergeben werden (Threadsynchronisierung!). Geräte die sich abmelden, sollen aus der Liste der bekannten Geräte entfernt werden!

Ebenfalls zu beachten ist, dass Geräte sich ankündigen können ohne, dass vorher eine Suchanfrage versendet wurde. Diese sind dann jedoch ausschließlich Multicast.

3.3 User-Thread

Zusammengefasst soll dieser **Thread** die Nutzereingaben lesen, verarbeiten und entsprechende Aktionen durchführen. Entsprechend verwaltet dieser Thread die anderen beiden Threads.

Auch der User-Thread soll bis zum Programmende in einer Endlosschleife laufen. Die Nutzereingabe sollen über **System.in** gelesen werden und die Ausgabe über **System.out** erfolgen (siehe **Command Line Interface** im Hilfsdokument). Auch hier gilt: Wenn gerade kein Befehl vorliegt sollte der Thread einige Millisekunden warten um den Prozessor nicht unnötig zu beanspruchen (10ms sind auch hier ein guter Wert).

Es sollen 4 Befehle implementiert werden. Befehle sind in der Eingabe zeilenweise getrennt.

Befehl 1: EXIT

Wenn der „EXIT“-Befehl vom Nutzer gesendet wird, so soll das Programm sich beenden.

Befehl 2: CLEAR

Wenn der „CLEAR“-Befehl vom Nutzer gesendet wird, so sollen alle Geräte vergessen werden. (Threadsynchronisierung!)

Befehl 3: LIST

Wenn der „LIST“-Befehl vom Nutzer gesendet wird, so sollen alle bekannten Geräte wie folgt gelistet werden (Threadsynchronisierung!). Jede Zeile soll genau 1 Gerät enthalten. D.h. eine UUID und ein oder mehr Dienst-Typen. Beispiel:

```
abcdef01-7dec-11d0-a765-a0a0c91e6bf6  -  ge:fridge, ge:ice-dispenser
17bb6bfd-3bde-4810-b89c-c39c2a1183ac  -  urn:service:WANPPPConnection
75728365-4352-1457-aabc-8bcafe174bca  -  ge:coffee-pot
abc642b3-73ad-bc4a-5b8e-58efabdc729c  -  ge:tea-pot
```

Befehl 4: SCAN

Wenn der „SCAN“-Befehl vom Nutzer gesendet wird, so soll über das MulticastSocket des Listen-Threads eine Suchanfrage versendet werden. Dafür ist ein DatagramPacket zu instanzieren mit den Paket-Daten:

```
M-SEARCH * HTTP/1.1            // Suche nach Geräten im Format HTTP/1.1
S: uuid:12345678-1234-1234-1234-123456789abc     // UUID des Anfragenden
HOST: 239.255.255.250:1900     // In der SSDP-Gruppe
MAN: "ssdp:discover"           // Anfrage-Typ: Geräte finden
ST: ge:fridge                  // Was für ein Geräte-Typ?
„leere Zeile“                  // Die leere Zeile schließt die Such-Anfrage ab
```

Die Kommentare sind nicht Teil des Pakets. Die UUID muss mit einer echten UUID ersetzt werden. Hierfür kann die UUID[7]-Klasse genutzt werden, die die Methode **randomUUID()** bereitstellt.

Die Zieladresse des Pakets ist die Multicast-Gruppe (also „239.255.255.250“) und der Zielport 1900. Zum versenden stellt das MulticastSocket die Methode **send(DatagramPacket)** bereit.

3.4 Fehlerbehandlungsrichtlinien

Das Programm soll Fehler/**Exceptions** sinnvoll behandeln. D.h. wenn beispielsweise ein „defektes“ Paket empfangen wird, so soll die Anwendung nicht abstürzten. Sie darf in diesem Fall eine Fehlermeldung ausgeben, sollte danach aber ohne Einschränkungen weiterlaufen.

Fehler die zum Programmende führen dürfen:

- Das MulticastSocket kann nicht gebunden werden.
- die **receive(DatagramPacket)**-Methode schlägt fehl.
- die **send(DatagramPacket)**-Method schlägt fehl.

Fehler die nicht zum Programmende führen dürfen:

- Fehlerhafte Daten in einem empfangenen Paket
- „Dumme“ Eingaben des Nutzers
- nicht abgefangene **Exceptions.** z.B.
  - **ConcurrentModificationException** (bei keiner/falscher Threadsynchronisierung)
  - **NullPointerException** (Ein Objekt wird benutzt, obwohl es **null** ist)

7 https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/UUID.html