0. **Allgemeine Information**

   Clients sollen in der Lage sein, beliebige Webseiten abzufragen und der Server soll dazu als simpler Web-Proxy dienen und angefragte Webseiten entsprechend cachen können, sodass mehrere Anfragen derselben Webseite nur 1x übers Internet geladen werden müssen und danach aus dem Cache des Servers ausgeliefert werden können. Sie schreiben dafür nur die Server-Anwendung, als Client können Sie zum Testen Telnet bzw. Putty verwenden. Der Server soll letztlich auch in der Lage sein durch Multi-Threading mehrere Clients parallel zu bedienen. Es gibt gesamt 100 Punkte.

1. **Generelle Anforderung**

   Stellen Sie bei dem hier entwickelten Projekt sicher, dass jegliche geöffnete Ressourcen auf jeden Fall nach deren Verwendung geschlossen werden. Offen verbleibende Ressourcen werden jeweils mit 3 Punkten in Abzug gebracht.

2. **Custom Exceptions [2 Punkte]**

   Erstellen Sie ein Package namens *org.campus02.web* und implementieren Sie zwei benutzerdefinierte Exceptions: UrlLoaderException und *CacheMissException* je als checked *Exception*. Beide sollen die Möglichkeit bieten, eine Fehlermeldung sowie eine weitere *Exception* (als „Ursache“) für das Exception-Chaining entgegenzunehmen.

3. **Klasse WebPage [3 Punkte]**

   Erstellen Sie eine Klasse WebPage mit folgenden privaten Attributen:

   - String url
   - String content
   - int size

   Erstellen Sie einen Konstruktor, der die Strings url und content übernimmt und entsprechend den Attributen zuweist. Das Attribut size soll mit der Länge des Strings content initialisiert werden. Erstellen Sie getter-Methoden für die Attribute und erstellen Sie eine toString-Methode, wo alle Attribute ausgegeben werden. Ebenso sollten Sie die Methoden equals und hashCode implementieren, wobei diese nur das Attribut url berücksichtigen dürfen.

4. **Klasse UrlLoader [10 Punkte]**

   Erstellen Sie eine Klasse UrlLoader mit einer öffentlichen statischen Methode

   ```
   public static WebPage loadWebPage(String url) throws UrlLoaderException { … }
   ```

   Diese Methode soll versuchen, den Seiteninhalt übergebene url mittels der aus der Vorlesung bekannten URL Klasse zu laden und als Objekt vom Typ WebPage zurückzuliefern. D.h. die url sowie den übers Netzwerk geladenen content übergeben Sie an den Konstruktor der WebPage Klasse. Wie in der Signatur oben ersichtlich sollen jegliche Exceptions die dabei auftreten können, als UrlLoaderException weitergeworfen werden und die ursprüngliche Exception mittels Chaining Mechanismus aufrechterhalten können.

5. **Klasse PageCache [20 Punkte]**

Erstellen Sie eine Klasse PageCache, mit privatem Attribut cache vom Typ HashMap<String,WebPage>. Diese Map speichert als Schlüssel URLs und als Wert dazugehörige WebPage Objekte. Schreiben Sie die folgenden Methoden:

[1 Punkt] getter-Methode für das Attribut cache

[3 Punkte] Methode, um vom cache zu lesen:

```
public WebPage readFromCache(String url) throws CacheMissException{…}
```

Falls die url im cache enthalten ist, das dazugehörige WebPage Objekt zurückliefern, ansonsten eine CacheMissException mit Fehlermeldung ausgeben.

[2 Punkte] Methode, um in den cache zu schreiben:

```
public void writeToCache(WebPage webPage) {…}
```

soll die übergebene webPage bezogen auf ihre url in den cache schreiben.

[14 Punkte] eine Methode um den cache bezogen auf eine Datei mit URLs aufwärmen d.h. alle in der Datei enthaltenen URLs sollen mittels UrlLoader Klasse geladen und in den cache geschrieben werden.

```
public void warmUp(String pathToUrls) {…}
```

Die Methode soll potentiell auftretende UrlLoaderExceptions abfangen und lediglich auf die Konsole schreiben, dass beim WarmUp ein Fehler aufgetreten ist. Implementieren Sie diese Methode so, dass jedenfalls versucht wird alle URLs aus der Datei zu laden, auch wenn unterwegs wo Exceptions auftreten könnten. Sie finden im Moodle eine Datei demo_urls.txt welche Sie zum Testen verwenden können.

6. **Klasse WebProxy [20 Punkte]**

Erstellen Sie eine Klasse WebProxy mit 3 privaten Attributen:

- PageCache cache
- int numCacheHits
- int numCacheMisses

[1 Punkt] Schreiben Sie einen Konstruktor ohne Parameter welcher intern einen neuen PageCache erstellt und dem entsprechenden Attribut zuweist.

[1 Punkt] Schreiben Sie zweiten Konstruktor (Overloading), welcher ein PageCache Objekt entgegennimmt und dem entsprechenden Attribut zuweist.

[6 Punkte] Schreiben Sie eine Methode namens fetch um eine URL abzufragen:

```
public WebPage fetch(String url) throws UrlLoaderException {…}
```

Die Methode soll zunächst versuchen die übergebene url im cache zu finden und von dort ein WebPage Objekt mittels readFromCache-Methode (siehe PageCache) zu lesen und zurückzuliefern. Wenn die Seite im cache gefunden wird, soll auch das Attribut numCacheHits um 1 erhöht werden. Falls Sie nicht gefunden wird muss die CacheMissException gefangen und der UrlLoader verwendet werden, um die noch nicht gecachte Seite abzurufen und als WebPage Objekt zurückliefern. Ebenso muss die Seite dann mittels writeToCache-Methode (siehe PageCache) in den cache geschrieben werden und das Attribut numCacheMisses um 1 erhöht werden.

[1 Punkt] Schreiben Sie eine Methode statsHits, welche einen String zurückliefert „stats hits: XY“, wobei XY der aktuelle Wert von numCacheHits sein soll.

[1 Punkt] Schreiben Sie eine Methode statsMisses, welche einen String zurückliefert „stats misses: YZ“, wobei YZ der aktuelle Wert von numCacheMisses sein soll.

[10 Punkte] Schreiben Sie eine Methode writePageCacheToFile:

```
public boolean writePageCacheToFile(String pathToFile) {…}
```

Diese Methode soll einen Pfad entgegennehmen und alle Einträge (Key + Value), die sich in der HashMap des PageCache Objektes befinden, Zeile für Zeile in eine Datei schreiben können, wobei eine Zeile wie folgt aussehen soll:

url;content

d.h. z.b.

https://www.orf.at;<html>....</html>

Die Methode soll bei erfolgreichem Schreiben true liefern. Falls irgendwelche Exceptions dabei auftreten müssen diese gefangen, der StackTrace auf der Konsole ausgegeben werden und die Methode false zurückliefern.

7. **Klasse ClientHandler [25 Punkte]**

Schreiben Sie eine Klasse ClientHandler welche das Runnable Interface implementiert. Die Klasse benötigt zwei private Attribute, welche über den Konstruktor entgegengenommen und initialisiert werden sollen:

- Socket client
- WebProxy proxy

In der Run Methode ist folgender Kommunikationsablauf basierend auf der Socket-Verbindung zum Client zu implementieren, wobei Client beliebig viele Kommandos nacheinander senden können:

- Wenn der Client nur den Text „bye“ eingibt will er aufhören und die Kommunikation soll entsprechend beendet werden. Es ist keine Antwort an den Client erforderlich

Jedes weitere Kommando besteht aus 2 Teilen, welche mit „ “ (Leerzeichen / Space) voneinander getrennt sind:

- Sofern ein Text gesendet wurde, der nicht aus 2 Teilen besteht, soll ein Fehlertext an den Client geschickt werden → error: command invalid
- Kommando **„fetch <url>“**: Wenn der 1. Teil „fetch“ ist, soll die danach folgende <url> (2. Teil) mittel WebProxy Objekt geladen werden. Sie sollen den String des content Attributs an den Client zurückschicken. Falls eine Exception auftritt ist diese zu fangen und ein Fehlertext an den Client zu senden → error: loading the url failed
- Kommando **„stats <hits | misses>“**: Wenn der 1. Teil „stats“ ist, soll je nach 2. Teil des Kommandos entweder die Anzahl an cache hits bzw. cache misses an den Client geschickt werden, welche mittels der jeweiligen Methoden des WebProxy Objekts abgefragt werden können.
    - stats **hits**: ist der 2. Teil „hits“ dann die cache hits zum Client senden
    - stats **misses:** ist der 2. Teil „misses“ dann die cache misses zum Client senden
    - **bei unbekanntem 2. Teil** ist ein Fehlertext an den Client zu senden → error: invalid command

8. **Klasse Server [15 Punkte]**

   Schreiben Sie eine Klasse Server mit einer main-Methode. Implementieren Sie darin eine Server-Anwendung die Clients einen WebProxy Dienst unter dem Port 5678 bereitstellt:

   Erstellen Sie zu Beginn ein PageCache Object und verwenden Sie die warmUp-Methode mit der demo_urls.txt Datei um den cache initial zu laden. Erstellen Sie ein WebProxy Objekt und übergeben Sie den geladenen cache. Binden Sie den Server Socket an das zuvor genannte Port und warten Sie auf Client Verbindungen. Sobald sich ein Client verbindet erstellen Sie ein neues ClientHandler Objekt und rufen Sie direkt dir run-Methode davon auf. Dieser Server kann nur einzelne Clients nacheinander bedienen. Sie können mittels Putty od. Telnet Ihre Server Anwendung ausprobieren. Es ist kein separater Client zu implementieren.

9. **Klasse ServerMT für Multithreading [5 Punkte]**

   Kopieren Sie Server Klasse und geben Sie Ihr den neuen Namen ServerMT. Ändern Sie diese Klasse in dem Sie hier den ClientHandler, welcher Runnable implementiert, gemeinsam mit der Thread Klasse verwenden, um pro eingehender Client Verbindung einen eigenen Thread abzuspalten, damit mehrere Clients parallel bedient werden können.