הנחיות לפתרון תרגילי הבית

- על הקוד המוגש להיות מתועד היטב ועליו לכלול:
  - מפרט, כפי שהודגם בתרגול.
  - תיעוד של כל מחלקה ומתודה ושל קטעי קוד רלוונטיים.
  - במידת הצורך, יש להוסיף תיעוד חיצוני.
- יש להפעיל את הכלי Javadoc כדי ליצור קבצי תיעוד בפורמט HTML ולצרף אותם לפתרון הממוחשב המוגש. כדי לגרום לקובצי ה-HTML להכיל את פסקאות המפרט שבהן אנו משתמשים, יש לציין זאת במפורש. ב-Eclipse, ניתן לבצע פעולה זו באופן הבא: 1. לבחור Export מתפריט File, לבחור Java->Javadoc וללחוץ על כפתור Next, 2. לבחור עבור Javadoc command את הקובץ javadoc.exe מתוך התיקייה bin הנמצאת בתיקייה שבה מותקן ה-Java SDK, 3. לבחור את הקבצים שלהם מעוניינים ליצור תיעוד וללחוץ פעמיים על כפתור Next, 4. להקיש ב-Extra Javadoc options את השורה הבאה וללחוץ על כפתור Finish:

-tag requires:a:"Requires:" -tag modifies:a:"Modifies:" -tag effects:a:"Effects:"

- התנהגות ברירת המחדל של פעולות assert היא disabled (הבדיקות לא מתבצעות). כדי לאפשר את הידור וביצוע פעולות assert, יש לבצע ב-Eclipse את הפעולות הבאות: 1. מתפריט Run לבחור Debug Configurations, 2. בחלון שנפתח, לעבור ללשונית Arguments, 3. בתיבת הטקסט VM arguments לכתוב ea–, 4. ללחוץ על כפתור Debug.

הנחיות להגשת תרגילי בית

- תרגילי הבית הם חובה.
- ההגשה בזוגות בלבד.
- עם סיום פתירת התרגיל, יש ליצור קובץ דחוס להגשה המכיל את:
  - כל קבצי הקוד והתיעוד.
  - פתרון לשאלות ה"יבשות" בקובץ Word או PDF. על הקובץ להכיל את שמות ומספרי תעודות הזהות **של שני הסטודנטים** המגישים.
- הגשת התרגיל היא **אלקטרונית בלבד**, דרך אתר הקורס **ע"י אחד מבני הזוג בלבד**.

הקובץ המוגש יקרא hw2_<id1>_<id2>.zip כאשר <id1> ו-<id2> הם מספרי הזהות של הסטודנטים המגישים. לדוגמא hw2_12345678_9876541.zip (כמובן יש להשתמש במספרי הזהות שלכם)

- תרגיל שיוגש באיחור וללא אישור מתאים (כגון, אישור מילואים), יורד ממנו ציון באופן אוטומטי לפי חישוב של 5 נקודות לכל יום איחור ועד 2 ימי איחור שלאחריהם לא תתאפשר ההגשה כלל.
- על התוכנית לעבור הידור (קומפילציה). על תכנית שלא עוברת הידור יורדו 30 נקודות.

מועד ההגשה:
יום ה', 28/12/17

המטרות של תרגיל בית זה הן להתנסות בתחומים הבאים:

- תכן של טיפוס נתונים מופשט (ADT), כתיבת מפרט עבורו, מימושו וכתיבת קוד המשתמש בו.
- כתיבת ה-abstraction function ו-representation invariant של טיפוס נתונים מופשט.
- ביצוע בדיקות יחידה בעזרת JUnit.

הצגת הבעיה

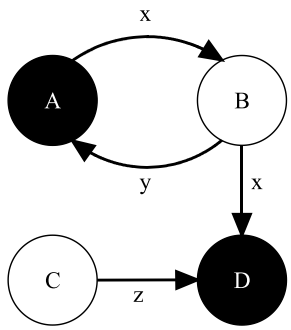
בתרגיל בית זה תיצרו הפשטה עבור גרף ותממשו אותה. בהמשך תיעזרו בהפשטה זו כדי לממש סימולטור של צינורות ומסננים, ולסיום תשתמשו בסימולטור זה לדמות ערוצי תשלום.

גרף מכוון (*directed graph*) מורכב מאוסף של *צמתים* (*nodes*) שחלקם או כולם עשויים להיות מקושרים בעזרת *קשתות* (*edges*). לכל צומת בגרף יש *תווית* (*label*) ייחודית. לכל קשת בגרף יש כיוון, כלומר, עשוי להתקיים מצב בו קיימת קשת המקשרת בין הצומת A לצומת B, אך לא קיימת קשת המקשרת בין הצומת B לצומת A. בתרגיל זה נניח כי לא יכולה להיות יותר מקשת אחת המקשרת צומת מסוים לצומת אחר (אך יכולה, כמובן, להיות אחת בכיוון ההפוך).

בגרף *עם תוויות לקשתות* (*labeled edged*), *לכל* קשת משויך אובייקט המתפקד כתווית של הקשת. בתרגיל זה נניח כי כל הקשתות היוצאות מצומת הן בעלות תווית שונה וכי כל הקשתות הנכנסות לצומת הן בעלות תווית שונה. עם זאת, לשתי קשתות שונות שלא יוצאות מאותו צומת ושלא נכנסות לאותו צומת עשויה להיות תווית זהה. כלל זה מאפשר לנו לזהות קשת בגרף באופן חד-משמעי בעזרת התווית שלה ובעזרת אחד הצמתים שהיא מחברת.

גרף דו-צדדי (*bipartite graph*) הוא גרף מכוון שבו קיימים שני סוגים של צמתים – צמתים לבנים וצמתים שחורים. הקשתות בגרף כזה מחברות רק צמתים בעלי צבעים שונים – מצומת שחור לצומת לבן או מצומת לבן לצומת שחור. לא קיימות קשתות המחברות צומת שחור לצומת שחור או צומת לבן לצומת לבן.

דוגמה לגרף דו-צדדי עם תוויות לקשתות:



הבנים (*children*) של הצומת B הם הצמתים שאליהם יש קשת מ-B. בדוגמה הנ"ל, הבנים של B הם A ו-D. *האבות* (*parents*) של הצומת B הם הצמתים שיש קשת מהם ל-B. בדוגמה הנ"ל, האב היחיד של B הוא A.

שאלה 1 (60 נקודות)

בשאלה זו תעסקו בתכן טיפוס נתונים מופשט (ADT) עבור גרף דו-צדדי עם תוויות לקשתות, במימושו ובבדיקתו. אין כאן תשובה אחת "נכונה" שאתם אמורים לגלות. המטרה היא לגרום לכם לחשוב על הבעיה, לבחון חלופות שונות לפתרונה, ולנמק את הסיבות לבחירה באחת מחלופות אלה.

א.
עליכם להתחיל את התכן בכך שתחליטו על הפעולות שעל ההפשטה לכלול. לשם כך, ניתן להיעזר במפרט הנתון של המחלקה BipartiteGraphTestDriver המשתמש בהפשטה שתיצרו, אך יש לשים לב לנקודות הבאות:

1. המפרט הנתון משתמש במחרוזות כתוויות לקשתות ולצמתים. ההפשטה שאתם מתכננים צריכה להיות **כללית**, כלומר, להיות מסוגלת לקבל אובייקט מטיפוס כלשהו כתווית לקשת או לצומת. לשם פשטות, ניתן להניח כי אותו טיפוס מייצג גם תווית לקשת וגם תווית לצומת. בנוסף, ניתן להניח כי מופעים של טיפוס זה הם אובייקטים בלתי ניתנים לשינוי (immutable).
2. המתודות של המחלקה BipartiteGraphTestDriver הן בעלות תנאים מוקדמים שונים, למשל, הן מניחות שהמשתמש לא יוסיף לגרף את אותו צומת פעמיים. ההפשטה שעליכם לתכנן **לא** יכולה להניח תנאים מקדימים אלה.
3. על ההפשטה שתתכננו להיות כללית ולספק יותר פעולות מאשר הפעולות שמפעילה המחלקה BipartiteGraphTestDriver.
4. יש לתכנן את ההפשטה כך שניתן יהיה להשתמש בטיפוס כלשהו לצמתי הגרף. כלומר, צומת בגרף יהיה אובייקט חיצוני לגרף שעשוי להכיל פונקציונליות מעבר להיותו צומת בגרף (למשל, כל צומת יהיה אובייקט המייצג מחשב ברשת מחשבים).

כתבו מפרט עבור ההפשטה שבחרתם, כולל פסקאות @requires, @modifies ו-@effects. את המפרט יש לרשום בקובץ בשם BipartiteGraph.java. על קובץ זה להכיל גם מספר שורות סקירה כללית של המחלקה ושל מה שהיא מייצגת.

בנוסף, **יש לרשום תיעוד חיצוני** המסביר את השיקולים בבחירת הפעולות השונות של ההפשטה. הסבירו מדוע אוסף פעולות זה נראה לכם מספק לפתרון בעיות שונות על גרף דו-צדדי עם תוויות לקשתות.

ב.
ממשו את המפרט שיצרתם בסעיף א'. בעת המימוש יש לזכור כי סימולטור, שעשוי להיות בעל מבנה נתונים גדול למדי, יעשה שימוש במימוש זה. הסימולטור משתמש באופן תכוף בפעולה של מציאת רשימת הבנים של צומת מסוים ובפעולה של מציאת רשימת האבות של צומת מסוים. לכן, עליכם לרשום מימוש שיבצע את שתי פעולות אלה בזמן קבוע. גם הפעולות לבניית גרף צריכות להיות יעילות באופן סביר. עם זאת, ראשית עליכם לדאוג לתכן נכון ורק לאחר מכן לביצועים טובים.

יש לרשום abstraction function ו-representation invariant בתוך שורות הערה בקוד של BipartiteGraph. בנוסף, יש לממש מתודת checkRep() לבדיקת ה-representation invariant ולקרוא לה במקומות המתאימים בקוד.

הנחיות:

1. ב-Java Documentation ניתן למצוא את פירוט סיבוכיות החישוב של פעולות שונות של כל אחד מהמכלים הקיימים בשפת Java. ניתן להשתמש בנתונים אלה לשם הערכת סיבוכיות החישוב של פעולות על מבני נתונים שונים.
2. למרות שהפלט עבור הבדיקות של גרף ב-BipartiteGraphTestDriver מוגדר לעתים כרשימה של צמתים לפי סדר אלפביתי, אין זה אומר שהמימוש צריך להחזיר או להכיל צמתים לפי סדר זה. ניתן, לחילופין, למיין את רשימת הצמתים לפני הצגתה. לשם כך ניתן להשתמש במתודה sort() של המחלקה java.util.Collections.

יש לרשום תיעוד חיצוני המסביר את השיקולים שהובילו למימוש הנבחר בסעיף ב'.

ג.
הציעו **במילים** מימוש שונה מזה שבחרתם בסעיף ב' והשוו אותו לזה שבחרתם.

ד.
בדיקת המימוש של BipartiteGraph תתבצע בעזרת המחלקה BipartiteGraphTestDriver שהמפרט שלה נתון בקובץ. ממשו מחלקה זו בהתאם למפרט הנתון.

ה.
נתון שלד מימוש של המחלקה BipartiteGraphTest המשתמשת ב-JUnit ובמחלקה BipartiteGraphTestDriver לשם ביצוע בדיקות יחידה של BipartiteGraph. במחלקה זו נתונה בדיקה אחת לדוגמה. עליכם להוסיף בדיקות קופסה שחורה ל-BipartiteGraph. עליכם לוודא שכל הבדיקות עוברות בהצלחה.

יש לרשום תיעוד חיצוני המתאר את הבדיקות שבחרתם ולהסביר מדוע הן מספקות.

הנחיות:

1. במידה והפרויקט ב-Eclipse לא מוצא את JUnit, יש לגרום לו לעשות זאת באופן הבא: א. לבחור בתפריט Project -> Properties, ב. לבחור בעץ בצד שמאל Java Build Path, ג. לבחור בצד ימין את הלשונית Libraries, ד. ללחוץ על כפתור Add External JARs, ה. לעבור לתיקיה plugins בתוך התיקייה שממנה מופעל ה-Eclipse ושם להיכנס לתוך התיקייה org.junit_4.1.2.0 (או כל גרסה דומה אחרת של JUnit), ו. לבחור את הקובץ junit.jar, ז. ללחוץ על כפתור OK.
2. ניתן להוסיף מחלקות נוספות על מנת לממש את ה ADT מחלקות אלו צריכות גם כן להכיל תיעוד. על פי הנדרש.
3. במידה והמימוש שלכם זורק חריגות, ניתן להוסיף אותן להצהרות של הפונקציות ב BipartiteGraphTestDriver.
4. הרצת קובץ בדיקה המשתמש ב-JUnit מתבצעת ע"י לחיצה על החץ שבכפתור Run ובחירת האפשרות Run As -> JUnit Test (אפשר לבצע אותה פעולה גם מתפריט Run).
5. המחלקה BipartiteGraphTestDriver מניחה הנחות על תנאים מקדימים לביצוע המתודות שאנו לא מרשים ל-BipartiteGraph להניח. לכן, כדי לבדוק מקרי קצה שאינם עומדים בהנחות אלה, יש להוסיף ל-BipartiteGraphTest גם בדיקות שלא משתמשות ב-BipartiteGraphTestDriver .

להגשה ממוחשבת: המחלקות BipartiteGraph, BipartiteGraphTestDriver ו-BipartiteGraphTest, כולל תיעוד מתאים וגם representation invariant ו-abstraction function ל-BipartiteGraph.
להגשה "יבשה": יש לרשום תיעוד חיצוני המסביר את השיקולים והחלופות בסעיפים א', ב', ג' וה', כפי שנדרש בכל סעיף.

שאלה 2 (15 נקודות)

בשאלה זו תשתמשו בהפשטה שיצרתם בשאלה 1 כדי לממש סימולטור כללי של צינורות ומסננים. במערכת כזו קיימים שלושה סוגי אובייקטים:

- *אובייקטי עבודה* המסתובבים במערכת.
- *צינורות* המעבירים את אובייקטי העבודה ממקום למקום.
- *מסננים* המעבדים את אובייקטי העבודה שמגיעים לצינורות הקלט שלהם ושולחים אובייקטי עבודה לצינורות הפלט שלהם.

מערכת מסוג זה יכולה לייצג בעיות מתחומים שונים, למשל, קו ייצור למכוניות. בקו ייצור למכוניות אובייקטי העבודה הם מכוניות בשלבי ייצור שונים, הצינורות הם מסועים בקו הייצור והמסננים הם תחנות עבודה שבהן מתבצעים שלבים שונים בייצור המכונית.

הטופולוגיה של מערכת צינורות ומסננים מיוצגת בעזרת גרף דו-צדדי עם תוויות לקשתות, שבו צינורות הם צמתים שחורים ומסננים הם צמתים לבנים. קשת מצינור למסנן מסמנת שהמסנן מקבל קלט דרך הצינור, ואילו קשת ממסנן לצינור מסמנת שהמסנן שולח פלט דרך הצינור. העובדה שהגרף הוא דו-צדדי מובילה לכך שצינורות

מחוברים רק למסננים ושמסננים מחוברים רק לצינורות. התווית של כל קשת מאפשרת למסנן להבדיל בין צינורות הקלט ובין צינורות הפלט שלו, אם זה חיוני לפעולת המסנן.

לצינור עשויה (אך לא חייבת) להיות קיבולת, שהיא המספר המרבי של אובייקטי עבודה שהוא יכול להחזיק. במידה ומספק אובייקטים נכנסים לצינור במקביל, הצינור עלול להגיע למצב בו הוא מלא ולסרב לקבל קלט נוסף כל עוד לא רוקן לפחות אובייקט אחד מיציאתו.

למסנן יש storage buffer בו יכולים להישמר אובייקטי עבודה במידת הצורך. יתכן שחלק מאובייקטי העבודה ימשיכו מיד לצינור המוצא, בעוד שאחרים ישמרו ב buffer לעיבוד מאוחר יותר.

הסימולטור מנוהל בסבבים (rounds) בשיטת round-robin. כל צינור וכל מסנן מקבל הוראה לבצע סימולציה של עצמו למשך סבב אחד. לאחר שכל הצינורות והמסננים סיימו, הסימולטור מתקדם לסבב הבא. בכל סבב, מתבצעת קודם כל סימולציה של כל הצינורות ולאחר מכן סימולציה של כל המסננים. מכיוון שצינורות לא מחוברים זה לזה, סדר הצינורות בשיטה זו אינו חשוב.

לכן, פסאודו-קוד של תהליך הסימולציה הוא:

```
for each pipe p
        simulate p for one round
for each filter f
        simulate f for one round
increment simulator round
```

תכננו וממשו מחלקה בשם Simulator שתבצע סימולציה של מערכת צינורות ומסננים. על המפרט של המחלקה לכלול פסקאות @requires, @modifies ו-@effects, וכן מספר שורות סקירה כללית של המחלקה ושל מה שהיא מייצגת.

הנחיות:

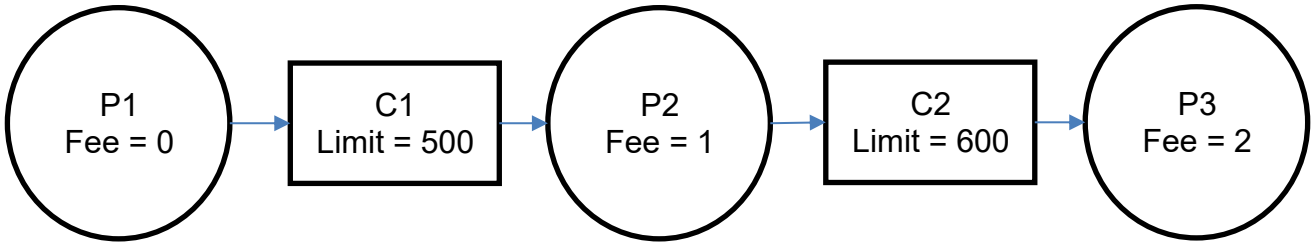
1. על המחלקה לספק פעולה בשם simulate() שתבצע סימולציה של סבב אחד במערכת, תוך שימוש בפסאודו-קוד הנ"ל.
2. על הסימולטור לתמוך בביצוע סימולציה על BipartiteGraph עם צמתים כלשהם. ההנחה היחידה על הצינורות והמסננים היא שהם מממשים את הממשק הנתון Simulatable.

להגשה ממוחשבת: המחלקה Simulator.

שאלה 3 (25 נקודות)

בשאלה זו תבדקו את הסימולטור שבניתם עבור מערכת המדמה ערוצי תשלום.
אובייקטי העבודה יהיו העברות המסננים יהיו שחקנים אשר רוצים להעביר תשלומים מאחד לשני.

נרצה לבנות מערכת שבה שחקים רוצים להעביר כספים מאחד לשני. שני שחקנים יכולים להיות מחוברים אחד לשני ישירות דרך ערוץ תשלום, או להיות מחוברים על ידי מספר שחקנים בדרך.



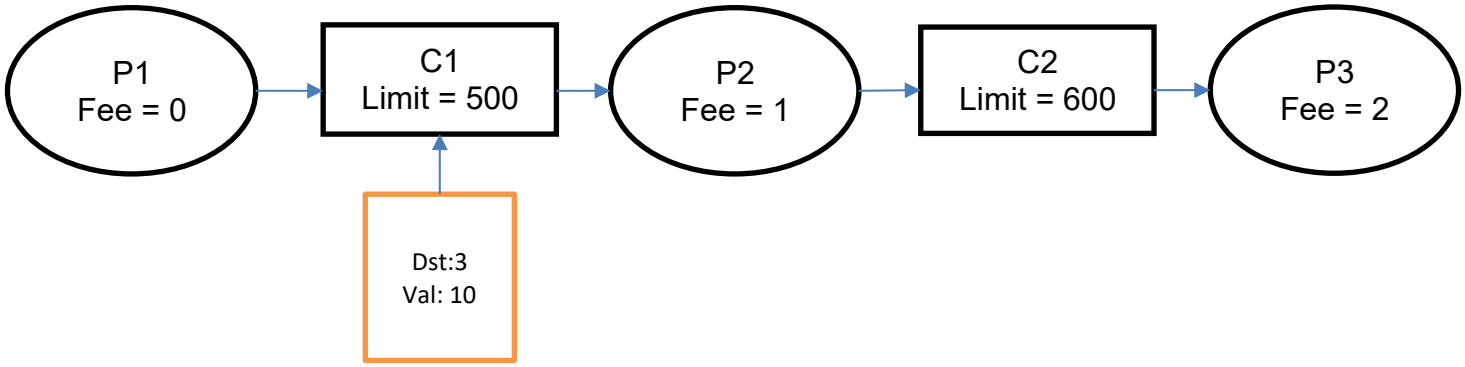
בדוגמא, שחקן 1 יכול להעביר תשלומים לשחקן 3, כאשר שחקן 2 הוא המתווך.

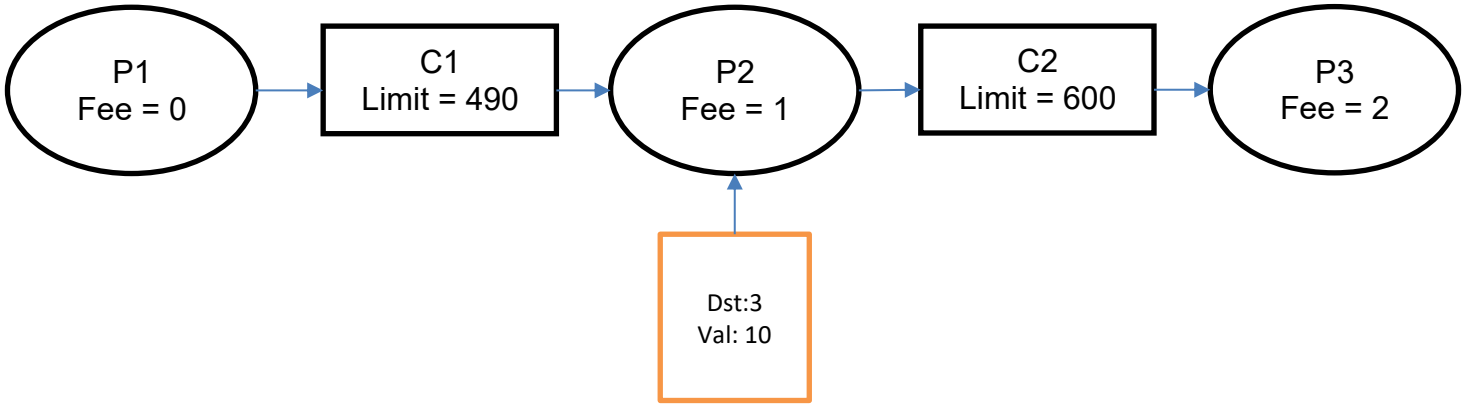
העברה (Transaction) הוא אובייקט immutable המכיל סכום לתשלום ויעד. היעד הוא אחד השחקנים המשתתפים בערוץ התשלום. מימוש המחלקה נתון כחלק מקבצי התרגיל.

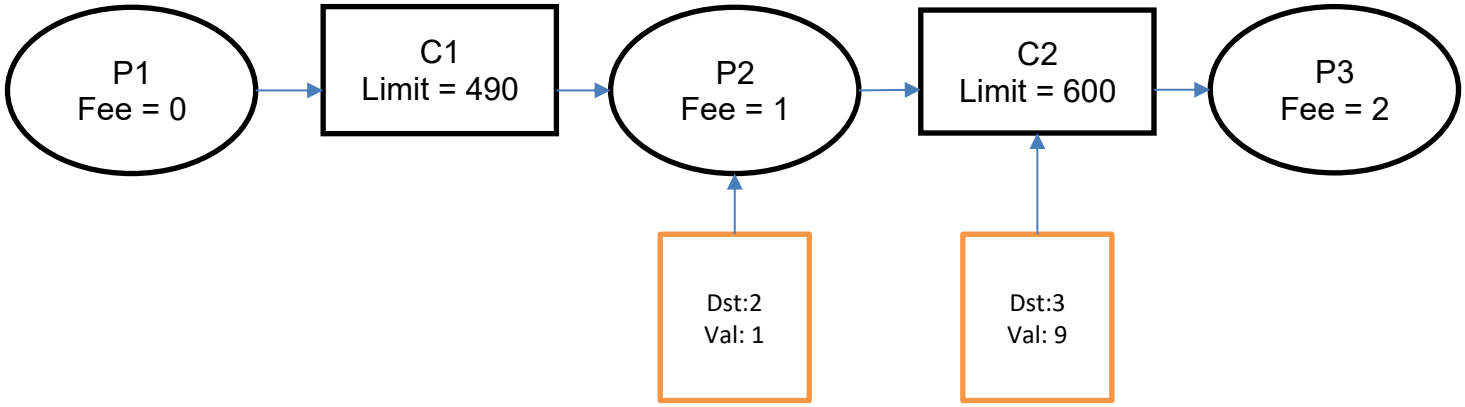
המחלקה Channel תממש צינור שיהווה ערוץ להעברת תשלומים. ל Channel יש קיבולת אינסופית (כלומר אין הגבלה על מספר התשלומים שנמצאים בו בכל רגע) , אבל יש מגבלה על הכמות הכוללת שיכולה לעבור דרך אותו צינור. במידה והסכום הכולל של ההעברות בערוץ מגיע ל limit לא ניתן יותר להעביר תשלומים דרך ערוץ זה .

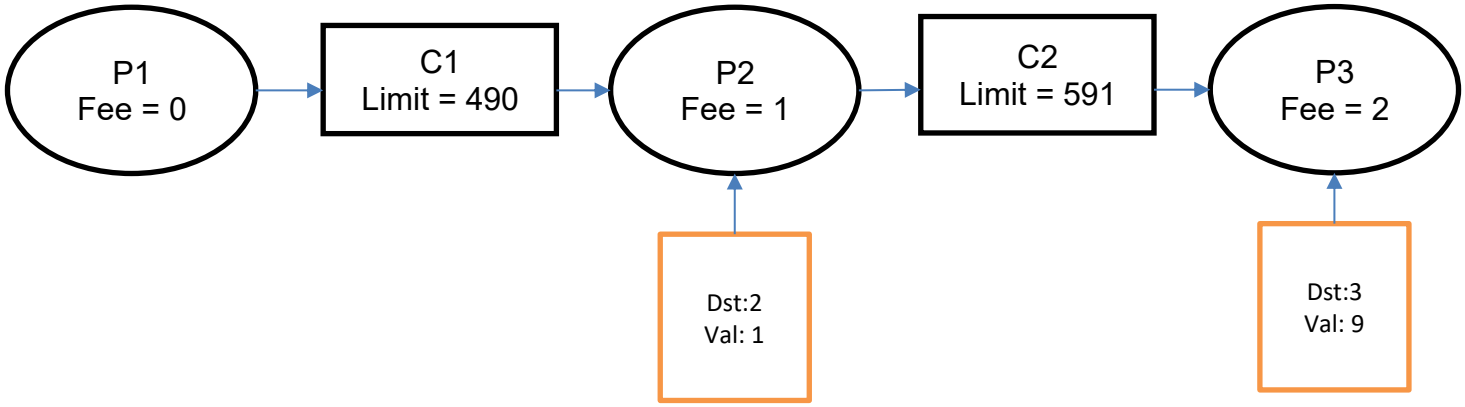
המחלקה Participant תממש מסנן שיהווה שחקן במערכת העברות התשלומים. לכל שחקן יש עמלה (fee) שאותה הוא גובה במידה והוא משמש מתווך בהעברת תשלום בין שחקנים אחרים. כאשר שחקן מקבל העברה, יעד התשלום יכול להיות השחקן עצמו, או שחקן אחר במערכת. במידה וההעברה מיועדת אליו, הוא ישמור אותה ב storage buffer שלו. אחרת, ייצור שתי העברות חדשות. אחת שמיועדת אליו וסכומה יהיה כסכום העמלה (אותה הוא ישמור ב buffer) והעברה נוספת שיעדה יהיה יעד ההעברה המקורית וסכומה יהיה סכום ההעברה המקורית לאחר הפחתת העמלה.

דוגמא להעברת תשלום בין שחקן 1 לשחקן 3 :


P1
Fee = 0
C1
Limit = 500
P2
Fee = 1
C2
Limit = 600
P3
Fee = 2
Dst:3
Val: 10


P1
Fee = 0
C1
Limit = 490
P2
Fee = 1
C2
Limit = 600
P3
Fee = 2
Dst:3
Val: 10


P1
Fee = 0
C1
Limit = 490
P2
Fee = 1
C2
Limit = 600
P3
Fee = 2
Dst:2
Val: 1
Dst:3
Val: 9


P1
Fee = 0
C1
Limit = 490
P2
Fee = 1
C2
Limit = 591
P3
Fee = 2
Dst:2
Val: 1
Dst:3
Val: 9

הנחות:

- לצורך הפשטות, נניח שהגרף הוא גרף עץ חד כיווני. כלומר שחקן שמקבל העברה יצטרך או לשמור אותה או להעביר אותה לערוץ היחיד שיוצא ממנו (יתכן וקיימים מספר ערוצים שנכנסים אליו)
- נניח שהשחקנים יודעים שבמידה והם רוצים לבצע העברה, קיים מסלול בו היא יכולה להגיע אל היעד הסופי שלה.
- במידה ושחקן "נתקע" עם ההעברה מכיוון שהערוץ היוצא ממנו הגיע ל limit הוא ישמור אותה אצלו.
- על מנת לבצע העברה, מונחת העברה על ערוץ התשלום היוצא מהשחקן. כך אתם יכולים לדמות התחלה של ההעברה בסימולטור.
- מטרת התרגיל היא לבצע שימוש במבנה הנתונים אותו ממשתם בשאלות הקודמות ולא להוות סימולטור מושלם של החיים האמיתיים. לכן אין צורך לדאוג לגבי התנהגות לא צודקת של שחקנים או מקרים בהם העברות לא מגיעות ליעדן.

א.

תכננו וממשו את המחלקות Participant ו Channel. כדי שמחלקות אלה יהיו שימושיות בסימולטור שבניתם, עליהן לממש את הממשק הנתון Simulatable. ניתן להניח כי מחלקות אלה פועלות עם תוויות מטיפוס String בלבד.

כתבו מפרט עבור מחלקות אלה, כולל פסקאות @requires, @modifies ו-@effects. יש לרשום גם מספר שורות סקירה כללית של כל המחלקה ושל מה שהיא מייצגת.

ב.

בדיקת המימוש של הסימולטור תתבצע בעזרת המחלקה SimulatorTestDriver שהמפרט שלה נתון. ממשו מחלקה זו בהתאם למפרט הנתון. במידה והסימולטור שלכם זורק חריגות, ניתן להוסיף אותן להצהרות של הפונקציות ב SimulatorTestDriver. (אבל לא לשנות את הממשק Simulatable).

ג.

עליכם לרשום מחלקה בשם SimulatorTest שתשתמש ב-JUnit ובמחלקה SimulatorTestDriver לשם ביצוע בדיקות קופסה שחורה של המחלקה Simulator. עליכם לוודא שכל הבדיקות עוברת בהצלחה.

להגשה ממוחשבת: המחלקות Participant, Channel, SimulatorTestDriver ו-SimulatorTest, כולל תיעוד מתאים.