# 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 במסמך "ההגשה היבשה"
- 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 zip להגשה המכיל את:
  - כל קבצי הקוד והתיעוד לפי מסמך ההוראות להגשת תרגילי הבית במודל.
    - השורה הראשונה בתוך כל קובץ קוד java צריכה להיות

package homework2;

  - פתרון לשאלות ה"יבשות" בקובץ PDF. **על הקובץ להכיל את מספרי תעודות הזהות של שני הסטודנטים המגישים בלבד, ללא שמות.**

- 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).

יש להגיש את תרגיל בית 2 לפי ההנחיות הבאות בסדר הבא:

1. שני בני הזוג צריכים להצטרף לאותה קבוצה ב"בחירת קבוצות לתרגיל בית 2" במודל.

2. אחד מבני הזוג צריך להגיש את קובץ ההגשה ב״מטלת תרגיל בית 2״.

לא יהיו הארכות למועד הגשת התרגיל למעט בקשות מוצדקות כמו מילואים, אשפוז, וכו׳. אנא תכננו את הזמן שלכם בהתאם.

- על התוכנית לעבור הידור (קומפילציה). על תכנית שלא עוברת הידור יורדו 30 נקודות.

# תרגיל בית 2′

מועד ההגשה:
יום ד′ 1/8/2024

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.

היעזרו בהנחיות לתרגילי הבית המופיעות בתחילת התרגיל.

## הצגת הבעיה

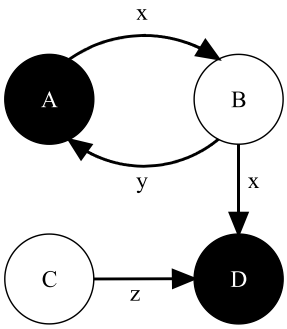
בתרגיל בית זה תיצרו הפשטה עבור גרף ותממשו אותה. בהמשך תיעזרו בהפשטה זו כדי לממש סימולטור של צינורות ומסננים, ולסיום תשתמשו בסימולטור זה לביצוע פעולות על מספרים שלמים.

*גרף מכוון* (*directed graph*) מורכב מאוסף של *צמתים* (*nodes*) שחלקם או כולם עשויים להיות מקושרים בעזרת *קשתות* (*edges*). לכל צומת בגרף יש אובייקט המתפקד כ-*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, כיווניות וצומת. כלומר, נדע האם הקשת בעלת התווית הנתונה נכנסת/ יוצאת מהצומת הנתונה.

*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 (55 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. 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).

   תכננו את הNode כך שיהיה עם המכנה המשותף הנמוך ביותר לכל הNodeים האפשריים בBipartiteGraph.

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:

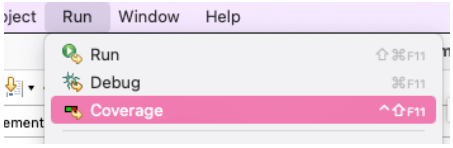
1. ב-Java Documentation ניתן למצוא את פירוט סיבוכיות החישוב של פעולות שונות של כל אחד מהמכלים הקיימים בשפת Java. ניתן להשתמש בנתונים אלה לשם הערכת סיבוכיות החישוב של פעולות על מבני נתונים שונים.
2. למרות שהפלט עבור הבדיקות של גרף ב-BipartiteGraphTestDriver מוגדר לעתים כרשימה של צמתים לפי סדר אלפביתי, 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.
3. ניתן להוסיף מחלקות נוספות על מנת לממש את ה ADT מחלקות אלו צריכות גם כן להכיל תיעוד. על פי הנדרש.

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

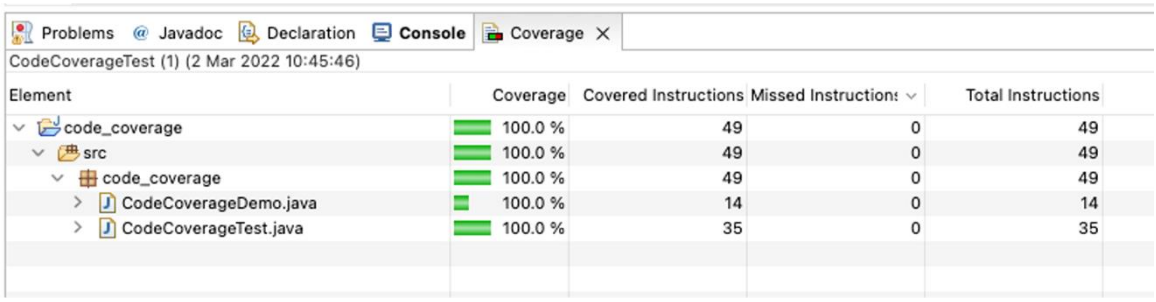
ג. בדיקת המימוש של BipartiteGraph תתבצע בעזרת המחלקה BipartiteGraphTestDriver שהמפרט שלה נתון בקובץ. ממשו מחלקה זו בהתאם למפרט הנתון.

ד. 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.

ה. בדקו את אחוז הקוד שנבדק במהלך הרצת טסטים.
קיימת תמיכה ב- Eclipse. על מנת לבדוק את ה- code coverage יש להריץ את הבדיקות ל-BipartiteGraph בעזרת הכפתור  או בעזרת לשונית Run ← Coverage




יש להוסיף צילום מסך של אחוז הקוד שנבדק, לדוגמא,




ניקוד הסעיף יהיה בהתאם לאחוז ה- coverage שקיבלתם (עבור הקבצים הרלוונטיים לתרגיל זה בלבד)

| coverage | ניקוד |
|---|---|
| מעל 80% | 10 |

| | |
|---|---|
| 70%-79% | 7 |
| 60%-69% | 6 |
| 50%-59% | 5 |
| 30%-49% | 3 |
| מתחת ל- 30% | 0 |

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.

<u>להגשה ממוחשבת</u>: המחלקות BipartiteGraph, BipartiteGraphTestDriver ו-BipartiteGraphTest, כולל תיעוד מתאים וגם representation invariant ו-abstraction function ל-BipartiteGraph.
במידה והוספתם מחלקות נוספות יש לצרף גם את קבצי מחלקות אלו.
<u>להגשה ״יבשה״</u>: יש לרשום תיעוד חיצוני המסביר את השיקולים והחלופות בסעיפים א׳, ב׳ כפי שנרש בכל סעיף. כמו כן, יש לצרף את צילום המסך לבדיקת ה- code coverage.

<u>שאלה 2</u> (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 מיוצגים ע״י צמתים שחורים ומסננים מיוצגים ע״י 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.

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 יתקדם 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.

סימולציה של צינור היא העברת אובייקט עבודה למסנן.
סימולציה של מסנן היא העברת אובייקט עבודה לצינור.

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.

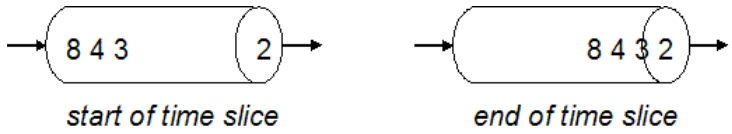
3. המחלקה Simulator לא צריכה לממש את הממשק Simulatable.
4. ניתן להוסיף פעולות אל BipartiteGraph במידת הצורך אך יש לספק מפרט מתאים לפעולות אלה.
5. ניתן להוסיף מחלקות נוספות על מנת לממש את Simulator מחלקות אלו צריכות גם כן להכיל תיעוד. על פי הנדרש.

<u>להגשה ממוחשבת</u>: המחלקה Simulator.
במידה והוספתם מחלקות נוספות יש לצרף גם את קבצי מחלקות אלו.

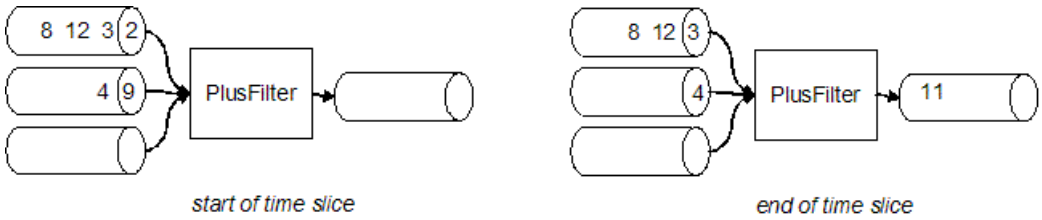
שאלה 3 (30 נקודות)

בשאלה זו תבדקו את הסימולטור שבניתם עבור פעולות חשבוניות על מספרים שלמים.
אובייקטי העבודה יהיו מספרים שלמים והמסננים יהיו פונקציות הפועלות על מספרים אלה.

המחלקה IntPipe תייצג צינור המסוגל להעביר מספרים שלמים. ניתן לממש צינור כזה בעזרת שתי רשימות – אחת עבור מספרים שזה עתה נכנסו לצינור ואחת עבור מספרים מוכנים ליציאה. כאשר צינור מטיפוס IntPipe מבצע איטרציה של סימולציה, עליו לקחת את כל המספרים ברשימת הקלט ולהעביר אותם לסוף רשימת הפלט, לדוגמה:



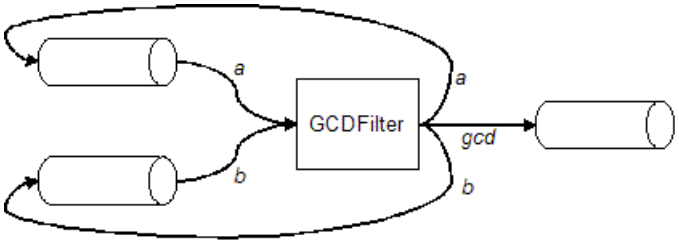

המחלקה PlusFilter תייצג מסנן בעל אפס או יותר צינורות קלט ובעל צינור פלט אחד. מסנן מטיפוס זה מחשב את סכום המספרים השלמים בצינורות הקלט שלו. כאשר מסנן מטיפוס PlusFilter מבצע איטרציה אחת של סימולציה, עליו לקחת מספר שלם מכל אחד מצינורות הקלט שלו, לחשב את הסכום של מספרים אלה, ולשלוח את התוצאה לצינור הפלט שלו. יש להתייחס לצינור קלט ריק כאילו הוא מכיל את המספר אפס. PlusFilter שלא מחוברים אליו צינורות קלט, יפלוט תמיד אפס. דוגמה:




המחלקה GCDFilter תייצג מסנן למציאת המחלק המשותף הגדול ביותר
(Greatest Common Divisor) בעזרת האלגוריתם של אוקלידס.
לקריאה נוספת:

https://en.wikipedia.org/wiki/Euclidean_algorithm

המחלקה GCDFilter תייצג מסנן בעל שני צינורות קלט המחוברים למסנן בקשתות בעלות התוויות a ו-b ובעל שלושה צינורות פלט המחוברים למסנן בקשתות בעלות התוויות a, b ו-gcd כפי שניתן לראות בתרשים הבא:



צינור הפלט המחובר אל המסנן בעזרת קשת בעלת התווית gcd, יכיל את תוצאת פעולת האלגוריתם.
בכל איטרציה של סימולציה, מסנן ה- GCDFilter מחשב צעד אחד באלגוריתם של אוקלידס למציאת המחלק המשותף הגדול ביותר (Greatest Common Divisor), כפי שהוא מתואר ב-pseudo code הבא:

```
a = integer from input pipe connected through edge a
b = integer from input pipe connected through edge b
if b = 0 then
        // computation is finished – a is the GCD
        send a to output pipe connected through edge gcd
else if a < b then
        // swap a and b so a >= b
        send b to output pipe connected through edge a
        send a to output pipe connected through edge b
else
        // let (a,b) = (b, remainder of a/b)
        send b to output pipe connected through edge a
        send a % b to output pipe connected through edge b
end if
```

בכל סיבוב של סימולציה, יתקדם האלגוריתם צעד אחד. כל זוג מספרים יסתובב במערכת עד ש-b יקבל את הערך אפס. בסיבוב זה, המסנן יפלוט את המחלק המשותף הגדול ביותר לצינור הפלט המחובר אליו בעזרת הקשת בעלת התווית gcd.

א. תכננו וממשו את המחלקות IntPipe, PlusFilter ו-GCDFilter. 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.

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

הנחיות:

1. ניתן להוסיף מחלקות נוספות, מחלקות אלו צריכות גם כן להכיל תיעוד. על פי הנדרש.

<u>להגשה ממוחשבת</u>: המחלקות IntPipe, PlusFilter, GCDFilter, SimulatorTestDriver ו-SimulatorTest, כולל תיעוד מתאים.
במידה והוספתם מחלקות נוספות יש לצרף גם את קבצי מחלקות אלו.