פתרון בחינה בתוכנה 1

סמסטר ב', מועד א' ,תש"ע

27/06/2010

ליאור וולף, ליאור שפירא, רובי בוים, מתי שמרת

הוראות (נא לקרוא!)

- משך הבחינה **שלוש שעות** - חלקו את זמנכם ביעילות.
- יש לענות על כל השאלות.
- בשאלות שבהן יש צורך לנמק, תשובה ללא נימוק לא תזכה באף נקודה.
- יש לענות על כל השאלות בגוף הבחינה במקום המיועד לכך. המקום המיועד מספיק לתשובות מלאות (ויותר). יש לצרף את טופס המבחן למחברת הבחינה. מחברת ללא טופס עזר תפסל. תשובות במחברת הבחינה לא תיבדקנה. **במידת הצורך ניתן לכתוב בגב טופס הבחינה.**
- יש למלא מספר סידורי (מס' מחברת) ומספר ת.ז על כל דף של טופס הבחינה.
- ניתן להניח לאורך השאלה שכל החבילות הדרושות יובאו, ואין צורך לכתוב שורות import.
- במקומות בהם תתבקשו לכתוב מתודה (שירות), ניתן לכתוב גם מתודות עזר.
- אסור השימוש בחומר עזר כלשהוא, כולל מחשבונים או כל מכשיר אחר פרט לעט. בסוף הבחינה צורף לנוחותכם נספח ובו תיעוד מחלקות שימושיות.

לשימוש הבודקים:

שאלה	א	ב	ג	ד	ה	ו	ז	סה"כ
1								
2								
3								
ציון בחינה:								

בהצלחה!

שאלה 1 (30 נקודות)

מערכות המלצה כיום משתמשות באלגוריתמי Collaborative Filtering (סינון שיתופי). אלגוריתמים אלו מבוססים על ההנחה שמשתמשים שהסכימו בעבר יסכימו גם בעתיד – לדוגמה: כאשר תגלשו באתר AMAZON (אתר לקניית ספרים ועוד) תוכלו לצפות בהמלצות בסגנון " Customers who viewed this also viewed.." ("לקוחות שבחנו מוצר זה גם הסתכלו על...")

בסיס הנתונים של מערכות אלו הוא טבלה של דירוגים (מספר 'כוכבים' בין 1 ל 5) כאשר השורות מייצגות משתמשים והעמודות מוצרים. לדוגמה, $R_{u,i} = 3.2$ מייצג את העובדה שמשתמש u דירג את מוצר i בציון 3.2.

דוגמה:

User/Product	Umbrella	Camera	Teddy bear	…		
Moshe	5		2.3			
Danny		1.4				
Ruthy	4.9	4	0.3			
Naama	2					

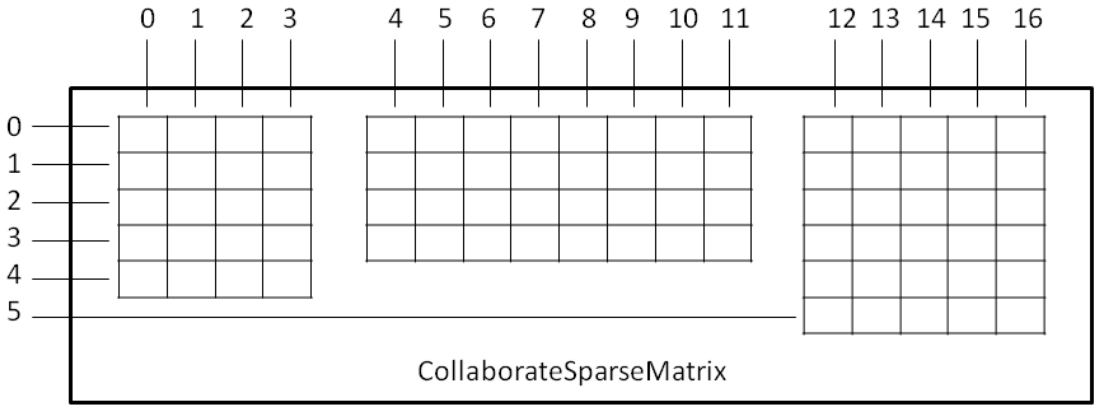
מטריצות אלו דלילות ביותר (פחות מ 1% בודד של הערכים ידועים) שכן משתמש בוחר לדרג רק חלק קטן מהמוצרים באתר. מימוש המטריצה ע"י מערך דו-מימדי אינו יעיל ואינו אפשרי בגלל מגבלות זיכרון (תחשבו כמה משתמשים יש לAMAZON למשל)

SparseMatrix הינו ממשק לייצוג מטריצות אלו בצורה יעילה. הקוד נתון להלן:

```
public interface SparseMatrix {
	// returns the number of columns
	public int getNumColumns();

	// returns the number of rows
	public int getNumRows();

	// return the rating of user u to item i
	public double getRating(int u, int i);

	// returns an iterator over the indexes of the known values
	// of column i
	public Iterator<Integer> getColIndexes(int i);

	// returns an iterator over the indexes of the known values
	// of row u
	public Iterator<Integer> getRowIndexes(int u);
}
```

בשאלה זו נרצה לממש את המחקר החדש של אחד מחברי סגל הקורס. הרעיון הוא לשפר את ההמלצות של חברה ספציפית בעזרת מידע מחברת אחרות. לדוגמה, בגלל שאותה לקוחה רוכשת מוצרים גם ב BLOCKBUSTER וגם ב TOYSRUS, נוכל להמליץ לה לקנות בובת R2D2 (דמות מסרטי מלחמת הכוכבים) ב TOYSRUS בעקבות הידע שהיא אהבה את סרטי STARWARS מ BLOCKBUSTER.

מטרה:
עליכם לממש מחלקה חדשה בשם CollaborateSparseMatrix שממשת את SparseMatrix על ידי שרשור של כמה מטריצות קיימות למטריצה בודדת. המטריצות מתארות את בסיס הנתונים של מספר חברות שונות בעלות מוצרים שונים אך בעלות משתמשים דומים. לכן 'נשרשר' את המטריצות בשורה ולא בעמודה, כפי שמראה הדוגמה הבאה:



הערות חשובות:

- הפונקציה getRating מומשה עבורכם. כלומר, אינכם צריכים להתייחס לשאלה מה הפונקציה תחזיר עבור ערכים שלא נתון דירוג עבורם,(לדוגמה עבור getRating(5,0).
- עבור השורות והעמודות, המספור של כל מטריצת מתחיל מ 0. לכן במטריצה החדשה, עבור כל המוצרים המטריצות למעט הראשונה, תצטרכו למספר מחדש את המוצרים (לפי סדר המטריצות והמוצרים). לדוגמה, המוצר השני במטריצה השניה, שקודם יוצג בספרה 1, ייצוג כעת בספרה 5 (שכן במטריצה הראשונה יש 4 מוצרים).
- שורות המטריצה מייצגות את אותם המשתמשים (ולכן ניתנות ל'שירשור'), אך מטריצה מסוימת יכולה להכיל יותר שורות (משתמשים) מאחרות. מספר השורות הכולל של המטריצה החדשה יהיה כמספר המקסימלי מבין המטריצות המוכלות.
- שימו לב שהפונקציות getColIndexes ו getRowIndexes מחזירות איטרטורים של אינדקסים ולא של דירוגים (עבור הערכים הידועים בלבד).

בסעיפים הבאים תתבקשו להשלים את הגדרת המחלקה CollaborateSparseMatrix. שימו לב שלמחלקה שתי פונקציות עזר פרטיות (את השנייה אתם תממשו):

- **getMatrixOffset(int matrix)**: מחזירה את ההפרש שיש להוסיף לאינדקס המוצרים למטריצה ספציפית. לדוגמה, getMatrixOffset(1) תחזיר 4 בדוגמה הנוכחית.
- **getHoldingMatrix(int item)**: מחזירה את אינדקס המטריצה שמחזיקה את המוצר הנתון. לדוגמה getHoldingMatrix(5) תחזיר 1 בדוגמה הנוכחית.

המתודה getRating(int u, int i) מדגימה שימוש בשתי המתודות הנ"ל.

```
public double getRating(int u, int i) {
    int m = getHoldingMatrix(i);
    return matrices.get(m).getRating(u, i - getMatrixOffset(m));
}
```

חלק א' (2 נקודות)

השלימו את הגדרת המחלקה

```
public class CollaborateSparseMatrix implements SparseMatrix
{
    private List<SparseMatrix> matrices;

    public CollaborateSparseMatrix(List<SparseMatrix> matrices) {
        this.matrices = matrices;
    }

    private int getMatrixOffset(int matrix) {
        int offset = 0;
        for (int i=0; i < matrix; i++)
            offset += matrices.get(i).getNumColumns();
        return offset;
    }
    ...
}
```

חלק ב' (4 נקודות)

ממשו את המתודה getHoldingMatrix כפי שמוסבר לעיל.

```
private int getHoldingMatrix(int item) {
    if (item < 0 || item >= getNumColumns()) {
        throw new IllegalArgumentException();
    }
    int i = 0;
    while ((i < matrices.size() - 1) &&
           (getMatrixOffset(i+1) < item)) {
        i++;
    }
    return i;
}
```

חלק ג' (4 נקודות)

ממשו את המתודה getNumColumns

```
public int getNumColumns() {
    return getMatrixOffset(matrices.size()-1) +
           matrices.get(matrices.size()-1).getNumColumns();
}
```

חלק ד' (4 נקודות)

ממשו את המתודה getNumRows

```
public int getNumRows() {
    return Collections.max(matrices,
        new Comparator<SparseMatrix>() {
            public int compare(SparseMatrix m1, SparseMatrix m2) {
                return m1.getNumRows() - m2.getNumRows();
            }
        }).getNumRows();
}
```

חלק ה' (4 נקודות)

ממשו את המתודה getColIndexes

```
public Iterator<Integer> getColIndexes(int i) {
    int m = getHoldingMatrix(i);
    return matrices.get(m).getColIndexes(i - getMatrixOffset(m));
}
```

חלק ו' (12 נקודות)

סיימו לממש את המתודה getRowIndexes

```
טעויות נפוצות (חלקים ה'-ו'):

- בדיקה האם דירוג קיים על ידי פונקצית getRating במקום getColIndexes/getRowIndexes
- אי הוספת ה OFFSET המתאים של האינדקס המוחזר
- ההנחה שבכל מטריצה שu קטן ממספר השורות אכן קיימים דירוגים עבור u. במילים אחרות, אי
  הבדיקה ש hasNext יחזיר true כאשר מקדמים את האיטרטור של המטריצות.

דוגמה לפתרון אפשרי בעמוד הבא..
```

```
public Iterator<Integer> getRowIndexes(final int u) {
    return new Iterator<Integer>() {
        private int currentMatrix = 0;
        private Iterator<Integer> currentIter = init();

        public boolean hasNext() {
             return currentIter.hasNext();
        }

        public Integer next() {
            if (!hasNext()) {
                 throw new NoSuchElementException();
            }
            Integer result = currentIter.next()+
                                getMatrixOffset(currentMatrix);
            advance();
            return result;
        }

        public void remove() {
            throw new UnsupportedOperationException();
        }

        private Iterator<Integer> init() {
            currentIter = matrices.get(0).getRowIndexes(u);
            advance();
            return currentIter;
        }

        private void advance() {
            while ((!currentIter.hasNext()) &&
                    (currentMatrix < matrices.size()-1)) {
                currentIter =
                   matrices.get(++currentMatrix).getRowIndexes(u);
            }
        }
    };
}
```

שאלה 2 (50 נקודות)

בשאלה זו נעסוק בגרף של פילטרים. גרף של פילטרים הוא ייצוג של סדר מסויים שבו זורם מידע במערכת מאובייקט לאובייקט כאשר כל פעם הוא עובר מודיפיקציה. לגרפים כאלו שימושים נרחבים בתחומים כמו מודלים תעשייתיים, פיענוח וידאו דחוס במחשב, ועוד ועוד. למשל וידאו יכול להקרא מקובץ, החלק של האודיו עובר למפענח אודיו (למשל מפענח mp3) והחלק של הוידאו עובר למפענח של וידאו. אנחנו נעסוק בפילטרים של מחרוזות.

בגרף של פילטרים שאנחנו נממש ישנם שלושה סוגי שחקנים: מקור, מקבל, ופילטר.

מקור (source):

יודע לשדר מחרוזות. שידור של מחרוזת מתבצעת ע"י קריאה למתודה Broadcast. מתודה זו קוראת למתודה *receiveString* של עצמים מסוג Receiver שנרשמו להאזין (מילה אחרת לקבל) את המחרוזות היוצאות ע"י קריאה למתודה *addReceiver*.

```
public interface Source{
    public void addReceiver(Receiver R);
    public void broadcast(String S);
}
```

מקבל (receiver):

יודע לקבל מחרוזות דרך המתודה receiveString

```
public interface Receiver{
    public void receiveString(String S);
}
```

פילטר: פילטר הוא מקור אשר משמש גם בתפקיד של מקבל (Receiver). כלומר, הוא מממש גם את Source וגם Receiver

דרך הפעולה של רוב הפילטרים היא לקבל מחרוזת דרך הפקודה receiveString, לעבד אותה, ולשדר (או לא לשדר) מחרוזת של תוצאה לפי תנאי מסוים.

למשל פילטר אחד יכול לקבל מחרוזת A ולשדר מחרוזת A.

פילטר אחר יוגדר לקבל מחרוזת A ולשדר את המחרוזת "קצר" אם האורך של A קטן מ 50 תווים או לא לשדר כלום אם האורך ארוך או שווה ל-50 תווים. ראו דוגמאות נוספות בסעיפים ב' ו-ג' למטה.

גרף של פילטרים נבנה ע"י כך שרושמים פילטרים להאזין למקורות (זכרו פילטר הוא גם מקור). כלומר פילטר מסוים יכול להרשם להאזין לשני פילטרים אחרים, שאחד מהם רשום לפילטר רביעי שרשום להאזין למקור מסויים.

סעיף א'

ממשו את המחלקה SystemInSource המממשת את המנשק Source אשר קוראת שורות קלט מ-System.in, ומשדרת אותן. הבנאי של המקור ריק, והוא מתחיל לקבל קלט מהשתמש/ת לאחר שמתבצעת קריאה למתודה בשם public void gogo()

לנוחיותכם מצורף בנספח תיעוד המחלקה Scanner

```
public class SystemInSource implements Source {
	private Scanner scanner;
	private Set<Receiver> receivers;

	public SystemInSource() {
		receivers = new HashSet<Receiver>();
		scanner = new Scanner(System.in);
	}

	@Override
	public void addReceiver(Receiver listener) {
		receivers.add(listener);
	}

	@Override
	public void broadcast(String str) {
		for (Receiver r : receivers) {
			r.receiveString(str);
		}
	}

	public void gogo() {
		while (scanner.hasNextLine()) {
			broadcast(scanner.nextLine());
		}
		scanner.close();
	}
}
```

סעיף ב'

ממשו פילטר אבסטרקטי (מופשט) אשר מממש פונקציונאליות רבה ככל האפשר משני המנשקים. בנו את הפילטר באופן כזה שפילטרים שיורשים מפילטר זה אינם צריכים לממש מחדש את המתודות שמומשו במחלקה האבסטרקטית. קראו לפילטר זה **AbsFilter.**

שימו לב: אין צורך לממש מחדש מתודות שמומשו בסעיף א'. פשוט רשמו את החתימה של המתודה במקום המתאים ללא מימוש.

רמז: בסעיף ג' למשל יש דוגמא לפילטר. חשבו כיצד תממשו את הפילטר האבסטרקטי בסעיף ב' כך שמימוש סעיף ג' הוא פשוט וסובב רק סביב ה-business logic.

```
public abstract class AbsFilter implements Receiver, Source {
      private Set<Receiver> receivers;

      public AbsFilter() {
            receivers = new HashSet<Receiver>();
      }

      @Override
      public void addReceiver(Receiver r) {
          // כמו בסעיף קודם
      }

      @Override
      public void broadcast(String str) {
          // כמו בסעיף קודם
      }
}
   מימוש receiveString במחלקה זו הוא טעות שכן פונקציה זו מממשת את ה
 "business logic" של כל אחת ממחלקות הפילטר הקונקרטיות. אין לה מימוש
      משותף לכל המחלקות היורשות ולכן אין סיבה לממש אותה במחלקת העל.
```

סעיף ג'

גרפר (Greper) הוא פילטר אשר מקבל בבנאי שלו מחרוזת pattern, ופועל באופן הבא: כשהוא מקבל (דרך receiveString) מחרוזת str אז:

במקרה בו pattern הוא תת מחרוזת של str, הוא משדר את str . אחרת, הוא לא משדר

ממשו את המחלקה **GreperFilter**

```
class GreperFilter extends AbsFilter  {

      private String pattern;

      public GreperFilter(String pattern) {
          this.pattern = pattern;
      }

      @Override
      public void receiveString(String str) {
          if (str.contains(pattern)) {
              broadcast(str);
          }
      }
}
```

סעיף ד'

ספרן (**StringCounterFilter**) הוא פילטר אשר משדר, בכל פעם שהוא מקבל מחרוזת (דרך receiveString) את המחרוזת המכילה את מספר המחרוזות שקיבל עד עתה. ממשו מחלקה זו.

```
public class StringCounter extends AbsFilter {
	private int count;

	@Override
	public void receiveString(String str) {
		broadcast(Integer.toString(++count));
	}
}
```

סעיף ה'

נגדן (Negator) הוא פילטר אשר מקבל בבנאי שלו פילטר F מטיפוס AbsFilter שלא רשום עדיין לקבלת הודעות ושלא נרשמו אצלו עדיין לקבלת הודעות (רכיב קשירות מגודל אחד בגרף של הפילטרים). הנגדן מקבל מחרוזת A ומתנהג כך:

אם F מוגדר לשדר פלט כלשהוא במידה והוא מקבל את A, הנגדן לא משדר. אם F לא משדר פלט כתוצאה מקבלת מחרוזת A, אזי הנגדן משדר את המחרוזת A.

מותר לנגדן להניח שאף אובייקט אחר לא ישלח מחרוזות ל-F (כלומר לא יתבצעו קריאות לreceiveString של F מחוץ הנגדן) ושאף אובייקט אחר לא ירשם להאזין לפלט של F.

ממשו את הנגדן במחלקה **NegatorFilter**

```
public class NegatorFilter extends AbsFilter {
	AbsFilter filter;
	boolean receivedFromFilter;

	public Negator(AbsFilter f) {
		filter = f;
		filter.addReceiver(this);
	}

	@Override
	public void receiveString(String str) {
		if (receivedFromFilter) {
			receivedFromFilter = false;  // reset the flag
		} else { // received from an external source
			receivedFromFilter = true;
			filter.receiveString(str);
			if (receivedFromFilter) {
				broadcast(str);
			}
			receivedFromFilter = false;
		}
	}
}
```

סעיף ו'

שרשרת פילטרים היא פילטר אשר מתקבל משרשור של פילטרים. הבנאי שלה ריק והיא מממשת מתודה בחתימה public void addFilter(AbsFilter F)

אם לא התבצעה קריאה למתודה addFilter אזי שרשרת הפילטרים לא פולטת. אם התבצעה קריאה אחת שכזו עם הפילטר F1 אז השרשרת מתנהגת כמו הפילטר F1.
התבצעה קריאה למתודה addFilter פעמיים, פעם אחת עם F1, ופעם שנייה עם F2 אזי הפלט עם קבלת מחרוזת A יהיה הפלט של F2 מופעל על הפלט של F1 כאשר F1 מופעלת על A, וכן הלאה.

ניתן להניח שהפילטרים שבשרשרת לא ירשמו לקבלת פלט מאף מקור אחר.

שם מחלקת השרשרת היא **FilterChain**.

```
public class FilterChain extends AbsFilter {
      private List<AbsFilter> filters;
      private Receiver lastReceiver;

      public FilterChain() {
          filters = new ArrayList<AbsFilter>();
      }

      public void addFilter(AbsFilter filter) {
          lastReceiver = new Receiver() {
              @Override
              public void receiveString(String str) {
                  if (this == lastReceiver) {
                      FilterChain.this.broadcast(str);
                  }
              }
          };
          if (!filters.isEmpty()) {
              filters.get(filters.size() - 1).addReceiver(filter);
          }
          filters.add(filter);
      }

      @Override
      public void receiveString(String str) {
          if (!filters.isEmpty()) {
              filters.get(0).receiveString(str);
          }
      }
}
```

סעיף ז'

המחלקה StringWriter ממשת את המנשק Reciever ומדפיסה את המחרוזות שהיא מקבלת למסך.
ממשו ת וכנית קצרה אשר מקבלת מחרוזות מSystem.in וממדפיסה כמה מחרוזות היא קיבלה בכל שלב שלא מכילות את המחרוזת "IloveSW1".

למשל באותה הרצה עבור קלט מהמשתמש ==> פלט למסך
Polymorphism ==> 1
Polymorphism ==> 2
Polymorphismmmm ==> 3
IloveSW1 ==> 3
IloveSW1 ==> 3
המלך דוד ==> 4
IloveSW1 ==> 4

```
public class ILoveSW1 public class ILoveSW1 {

    public static void main(String[] args) {
        StringWriter out = new StringWriter();
        StringCounter counter = new StringCounter();
        NegatorFilter negator =
                  new NegatorFilter(new GreperFilter("ILoveSW1"));
        SystemInSource in = new SystemInSource();

        in.addReceiver(negator);
        negator.addReceiver(counter);
        counter.addReceiver(out);

        in.gogo();
    }
}
```

מניסוח השאלה היה ניתן להבין כי יש להדפיס לאחר **כל** קריאה של מחרוזת.
התקבלו גם תשובות שלא עשו שימוש בפילטרים והדפיסו לאחר כל קלט,
כדוגמת הפתרון למטה.

```
public class ILoveSW1 {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int counter = 0;

        while (scanner.hasNextLine()) {
            if (!scanner.nextLine().contains("ILoveSW1")) {
                counter++;
            }
            System.out.println(counter);
        }
        scanner.close();
    }

}
```

שאלה 3 (20 נקודות)

בכל סעיף מהסעיפים הבאים יש לענות האם:

(א) התכנית עוברת קומפילציה (הידור) או לאו. אם לא, יש לנמק. תשובה ללא נימוק לא תזכה בנקודות.
(ב) האם התכנית עפה בזמן ריצה? אם כן איפה ומדוע.
(ג) אם התכנית רצה כמו שצריך, מהו הפלט המודפס?

סעיף א'

```
//A.java
01 public class A {
02     public A() {
03             System.out.println("a1");
04     }
05     public A(int x) {
06             System.out.println("a2");
07     }
08}
//B.java
09 public class B extends A {
10     public B() {
11             super(5);
12             System.out.println("b1");
13     }
14     public B(int x) {
15             this();
16             System.out.println("b2");
17     }
18     public B(int x, int y) {
19             super(x);
20             System.out.println("b3");
21     }
22}
//Testing.java - main
…
23             B b = new B(5,5);
…
```

התשובה שלכם:

התכנית תדפיס:

a2
b3

ביצוע יצירת האובייקט בשורה 23 תגרום לקריאה לבנאי של B בשורה 18, קריאה זו תגרום לביצוע הבנאי של A בשורה 5 (יודפס a2) ולאחר מכן יתבצע הגוף של הבנאי של B (יודפס b3)

סעיף ב'

```
//A.java
01 public class A {
02    public int x = 5;
03
04    public A() {
05          x = 10;
06          System.out.println(this.x);
07    }
08
09}
//B.java
10 public class B extends A {
11    public int x = 7;
12
13    public B() {
14          System.out.println(((A)this).x);
15          System.out.println(x);
16    }

17    public String toString() {
18          return ""+x;
19    }
}
//Testing.java – main
…
20          B b = new B();
…
```

התשובה שלכם:

התכנית תדפיס:

10
10
7

הבנאי של מחלקה B יגרום להפעלת הבנאי של A לפני תחילת פעולתן. בבנאי זה יושם הערך 10 במשתנה x אשר יודפס מייד לאחר מכן.
כאשר חוזרים לבנאי של B, מודפס המשתנה x עם הסבה של הטיפוס הסטטי לA ולכן יודפס שוב הערך 10. כאשר מדפיסים פשוט את x, הטיפוס הסטטי הוא כשל this ולכן B ולכן מודפס הערך 7

חלק ג'

```
//A.java
01 public class A {
02     public A() {
03     }
04     public A(A a, B b) {
05           System.out.println(a + " " + b);
06     }
07     public A(B a, A b) {
08           System.out.println(a + " " + b);
09     }
10}
//B.java
11 public class B extends A {
12     public B() {
13     }
14     public B(B b) {
15           super(b, b);
16     }
17     public B(A a) {
18           super((B)a, a);
19     }
20}
//Testing.java – main
…
21     B b = new B(new B());
22     System.out.println(b);
…
```

התשובה שלכם:

שגיאת קומפילציה בשורה 15. הקומפיילר אינו יכול להכריע לאיזה בנאי של A לקרוא (שניהם מתאימים באותה המידה). קיבלנו גם תשובה כי הדבר גורר שגיאת זמן ריצה (עקב ambiguity) מאחר ובגרסאות קודמות של אקליפס כך היה (וכך גם בקומפילציה ידנית של הקוד).

חלק ד'

```
01:
02: class Test
03:     {
04:         static void show()
05:         {
06:             System.out.println("Show method in Test class");
07:         }
08:  }
09: public class Main extends Test
10:    {
11:        static void show()
12:        {
13:            System.out.println("Show method in Main class");
14:        }
15:        public static void main(String[] args)
16:        {
17:             Test t = new Test();
18:             t.show();
19:             Main m = new Main();
20:             m.show();
21:
22:             t = m;
23:             t.show();
24:
25:             m = t;
26:             m.show();
27:        }
28: }
```

התשובה שלכם:

שגיאת קומפילציה בשורה 25 – לא ניתן להמיר מהטיפוס Test לטיפוס Main

ושוב בהצלחה!!