

פתרון הבחינה בתוכנה 1 (2157 - 0368)
ובתוכנה 1א (2004 - 0368)

סיון טולדו, אוהד ברזילי, מיכל עוזרי-פלאטו, ליאור שפירא

סמסטר א' תשס"ז
מועד ב', 8 באוקטובר 2007

משך הבחינה שלוש שעות.

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. יש לצרף את טופס המבחן למחברת הבחינה. מחברת ללא טופס עזר תפסל. תשובות במחברת הבחינה לא תיבדקנה.

יש למלא מספר סידורי (מספר מחברת) ומספר ת"ז על כל דף של טופס הבחינה.

אסור השימוש בחומר עזר כלשהו, כולל מחשבוניו או כל מכשיר אחר פרט לעט.
בהצלחה!

לשימוש הבודקים:

40	4 ז	4 י	4 ה	10 ד	4 ג	10 ב	4 א	1
36					12 ג	12 ב	12 א	2
24		4 י	4 ה	4 ד	4 ג	4 ב	4 א	3
100	סה"כ							

שאלה 1, 40 נקודות

בשאלה זו נדון במחלקות שמייצגות מספרים רציונליים, כגון $1/3$ או $99/100$. תזכורת מתמטית קצרה: כדי לצמצם שברים צריך לחלק את המונה והמכנה בגורם המשותף הגדול ביותר (GCD באנגלית). למשל, הגורם המשותף הגדול ביותר של 42 ו-56 הוא 14, ולכן

$$\frac{42}{56} = \frac{3 \cdot 14}{4 \cdot 14} = \frac{3}{4}.$$

כדי לחבר או לחסר שברים, צריך למצוא את המכנה המשותף (LCM באנגלית). למשל, המכנה המשותף של 21 ו-6 הוא 42, ולכן

$$\frac{2}{21} + \frac{1}{6} = \frac{4}{42} + \frac{7}{42} = \frac{11}{42},$$

בשאלה הזו נניח שהמונה (numerator) והמכנה (denominator) מיוצגים תמיד על ידי שדות או משתנים מטיפוס int. בכל המחלקות שנדון בהן, המונה מיוצג על ידי שדה מופע בשם num והמכנה על ידי שדה מופע בשם denom.

א. המחלקות שמייצגות מספרים רציונליים ימומשו תוך שימוש במחלקת עזר MathExtra שמכילה שירותי מחלקה שמממשים אלגוריתם לחישוב מכנה משותף של שני שלמים (שם השירות lcm) ואלגוריתם לחישוב הגורם המשותף הגדול ביותר של שני שלמים (שם השירות gcd).

הגדר/הגדירי את המחלקה הזו, השאר/י את גוף השירותים הללו ריק (כלומר אין צורך לכתוב קוד שמממש את האלגוריתמים הללו).

```
public class MathExtra {
    public static int lcm(int a, int b) {
        return a * b / gcd(a, b);
    }

    public static int gcd(int a, int b) {
        if (b == 0)
            return a;
        return gcd(b, a % b);
    }
}
```

ב. במחלקה Rational1 שעליך להגדיר, משתמר הייצוג הוא $denom \neq 0$ (כלומר אסור למכנה להיות אפס). השלם/השלימי את הגדרת המחלקה. יש להוסיף קוד חסר ואת תנאי הקדם והאחר. על התנאים להיות עקביים עם מימושכם ולבטא הנחות נוספות שהנחתם (אם יש כאלה).

```
// an object of this class represents the rational
// number num/denom.
// representation invariant: denom != 0
```

```
public class Rational1 {
    private int denom;
    private int num;
```

```

// precondition: denom!=0
// postcondition:
public Rational1(int denom, int num) {
    this.denom = denom;
    this.num = num;
}

// precondition: r!=null
// postcondition:
public void add(Rational1 r) {
    int lcm = MathExtra.lcm(denom, r.denom);
    num = num * lcm / denom + r.num * lcm / r.denom;
    denom = lcm;
}

// precondition: r!=null && !r.equals(new Rational1(1,0))
// postcondition:
public void divideBy(Rational1 r) {
    Rational1 inverse = new Rational1(r.num, r.denom);
    multiply(inverse);
}

// precondition: r!=null
// postcondition:
public void multiply(Rational1 r) {
    num *= r.num;
    denom *= r.denom;
}

// precondition: r!=null
// postcondition:
public boolean equals(Rational1 r) {
    int thisGCD = MathExtra.gcd(denom, num);
    int otherGCD = MathExtra.gcd(r.denom, r.num);

    return (num/thisGCD == r.num/otherGCD)
        && (denom/thisGCD == r.denom/otherGCD);
}
}

```

הערות על הבדיקה:

- תנאי הקדם יכול להכיל רק ביטויים public
- בשאלה לא התבקשתם לציין מצב מופשט (abstract state)
- תלמידים אשר צמצמו את השבר לאחר כל פעולה היו צריכים לעשות זאת בשרות נפרד כדי להמנע משכפול קוד

ג. נניח שהספריה הסטנדרטית של ג'אוה מכילה את המחלקה Rational1 מהסעיף הקודם, ושקיבלת את קוד המקור שלה. כיצד היית מגדיר מחלקה משלך שתאפשר גם לחסר מספרים רציונליים? עני/ענה במילים, ללא קוד.

ניתן לרשת מהמחלקה Rational1 ולהוסיף מתודה הממשת חיסור כך: היא מכפילה את הארגומנט ב -1 ואז מחברת אותו.

הערות על הבדיקה:

- א. יש לשים לב שברושה אין גישה לשדות הפרטיים של מחלקת הבסיס
- ב. יש להמנע משכפול קוד מיותר

ד. עכשיו הגדירו מחלקה אחרת, Rational2, עם אותו משתמר ייצוג (denom!=0), אבל שזורת חריג כאשר המכנה מתאפס. החריג שנזרוק יהיה java.lang.ArithmeticException

הטקסט הבא מועתק מתוך התיעוד של הספרייה הסטנדרטית:

java.lang

Class ArithmeticException

[java.lang.Object](#)

└─ [java.lang.Throwable](#)

└─ [java.lang.Exception](#)

└─ [java.lang.RuntimeException](#)

└─ [java.lang.ArithmeticException](#)

All Implemented Interfaces:

[Serializable](#)

```
public class ArithmeticException
extends RuntimeException
```

Thrown when an exceptional arithmetic condition has occurred. For example, an integer "divide by zero" throws an instance of this class.

Since:

JDK1.0

הגדר/י את המחלקה, את תנאי הקדם והאחר שלה ואת החריגים שהיא זורקת.

```
// an object of this class represents the rational
// number num/denom.
// representation invariant: denom != 0

public class Rational2 {
    private int denom;
    private int num;

    // precondition: true
    // postcondition:
    // throws: (denom == 0) $implies throwing ArithmeticException
    public Rational2(int denom, int num) {
        if (denom == 0)
            throw new ArithmeticException();
        this.denom = denom;
        this.num = num;
    }

    // precondition: r!=null
    // postcondition:
    public void add(Rational2 r) {
        int lcm = MathExtra.lcm(denom, r.denom);
        num = num * lcm / denom + r.num * lcm / r.denom;
        denom = lcm;
    }
}
```

```

// precondition: r!=null
// postcondition:
// throws: r.equals(new Rational1(1,0) $implies throwing ArithmeticException
public void divideBy(Rational2 r) {
    if (r.equals(new Rational1(1,0)))
        throw new ArithmeticException();
    Rational2 inverse = new Rational2(r.num, r.denom);
    multiply(inverse);
}

// precondition: r!=null
// postcondition:
public void multiply(Rational2 r) {
    num *= r.num;
    denom *= r.denom;
}

// precondition: r!=null
// postcondition:
public boolean equals(Rational2 r) {
    int thisGCD = MathExtra.gcd(denom, num);
    int otherGCD = MathExtra.gcd(r.denom, r.num);

    return (num/thisGCD == r.num/otherGCD)
        && (denom/thisGCD == r.denom/otherGCD);
}
}

```

הערות על הבדיקה:

- בפונקציה שהצהירה שהיא זורקת חריג צריך שתהיה קיימת אפשרות ריצה שבה הפונקציה אכן זורקת את החריג (למשל פעולות חיבור וכפל על רציונלים תקינים אינן יכולות לזרוק חריגים)
- תנאי הקדם צריך להכיל את כל המקרים שבהם הקלט חוקי או מוביל לזריקת חריג
- עבור זריקת חריג היה צריך לציין באילו מקרים הוא נזרק

ה. אם נגדיר ממשק (interface) בשם Rational ששתי המחלקות הקודמות יממשו אותו, האם צריך להכריז על זריקת חריגים בממשק? נמקו תשובתכם.

מבחינת קומפילציה שתי האפשרויות חוקיות. להכרזה על חריגים בממשק יש יתרון בכך שלקוח של Rational2 אינו מופתע מזריקה של חריג. להכרזה על חריגים בממשק יש חסרון בכך שלקוח של Rational1 אינו מבין מה פשר החריגים בחתימת השרות. ולממשק Ratioan1 הדבר יוצר בעיה בהתמודדות עם תנאי קדם חלש מדי.

שתי האפשרויות קיבלו ניקוד מלא אם נמקו כהלכה.

1. מה צריך להיות החוזה (תנאי הקדם האחר וזריקת החריגים) של מנשק כזה?

לזריקת חריגים יש השפעה על תנאי הקדם – היא מחלישה אותו: מקרי קצה שלא טופלו קודם כלל, כעת מזרק עבורם חריג.

מכיוון שלמחלקה מותר רק להחליש את תנאי הקדם של המנשק שאותו היא מממשת נעדיף מנשק ללא חריגי זמן ריצה (runtime exceptions) המאפשרים לנו לנסח תנאי קדם מחמירים בחוזה (כפי שנוסחו ב Rational1).

נשים לב כי חריגים אלו יכולים להיזרק רק בשרות dividedBy (ובבנאי אשר אינו חלק מהמנשק). הצהרה על חריג זה בשרותים אחרים "just in case" היא שגויה

2. נניח שהיינו מגדירים מחלקה בשם Rational3 שהקוד שלה היה זהה* לשל Rational2 אבל שזורקת חריג שהגדרנו בעצמנו, מהטיפוס:

```
public class RationalDivideByZero  
    extends java.lang.Exception {...}
```

(*כלומר, המימוש של שירותים ב-Rational3 יהיה זהה למימוש ב-Rational2, אבל כמובן ששירותים שזורקים חריג יצהירו שהם עלולים לזרוק אותו).

איך צריך להגדיר את החוזה (תנאי הקדם האחר וזריקת החריגים) של מנשק שהמחלקות Rational1 ו-Rational3 מממשות (אם היה כזה)? נמקו תשובתכם.

בשונה מהסעיף הקודם, כעת מבחינת קומפילציה יש הגבלות: מתודה מממשת לא יכולה לזרוק checked exception אם הוא לא הוגדר במפורש בחתימת השרות במנשק.

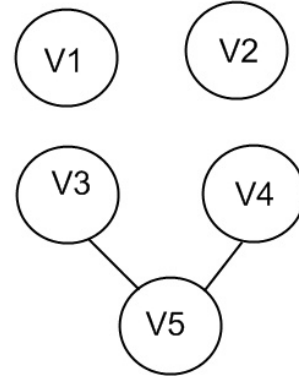
כדי להגדיר מנשק עיקבי העונה על כל תנאי השאלה יש להגדיר במנשק המשותף את השרות dividedBy הזורק חריג ולהגדיר את תנאי הקדם של השרות dividedBy על פי Rational1 (הארגומנט אינו שווה לאפס).

יש לציין שהגדרה זו היא מאולצת, שכן ע"פ ההגדרה במנשק לא ברור באילו סיטואציות ייזרק החריג.

שאלה 2, 36 נקודות

בשאלה זו נממש מחלקה אבסטרקטית של גרפים לא מכוונים (להלן גרף). גרף הוא מבנה שיש לו קבוצת קודקודים V וקבוצת קשתות E . לצורך השאלה נניח שכל קודקוד הוא מופע של מחלקה בשם `Vertex` (ראו למטה), וכל קשת היא זוג לא סדור `Pair` של קודקודים.

בדוגמא זו:
 $V = \{ V1, V2, V3, V4, V5 \}$
 $E = \{ \{V3, V5\}, \{V5, V4\} \}$



המחלקה לייצוג קודקוד מוגדרת בפשטות:

```
public class Vertex {}
```

- א. הגדירו מחלקת עזר גנרית (generic class) בשם `Pair` אשר מתאימה לייצג זוגות לא סדורים של עצמים מאותו טיפוס. על המחלקה לכלול:
1. בנאי המקבל שני עצמים מאותו טיפוס
 2. מתודה `toArray` המחזירה מערך בן שני אברים המכיל את אברי הזוג
 3. מתודת `equals` להשוואה לזוג לא סדור אחר (השוואה מבוססת תוכן)

```
public class Pair<T> {
    public Pair(T e1, T e2){
        this.e1 = e1;
        this.e2 = e2;
    }
    public T[] toArray(){
        return (T[]) new Object[]{e1, e2};
    }
    @Override
    public boolean equals(Object o) {
        if (!(o instanceof Pair))
            return false;
        Pair other = (Pair)o;
        return e1.equals(other.e1) && e2.equals(other.e2) ||
            e1.equals(other.e2) && e2.equals(other.e1);
    }
    private T e1;
    private T e2;
}
```

הערות על הבדיקה וטעויות שכיחות:

- לא ניתן לייצר מערך מטיפוס `T` ע"י `new T[2]`
- על ההשוואה להשתמש ב `equals` של האיבר
- ההשוואה היא ללא חשיבות לסדר
- המחלקה `Pair` גנרית ואינה תלויה ב- `Vertex`
- אם המימוש משתמש במערך כשדה, על השרות `toArray` לא לחשוף אותו

- ב. הגדירו מחלקה אבסטרקטית של גרף, Graph, על המחלקה לכלול:
1. מתודה אבסטרקטית vertices אשר מחזירה את אוסף כל הקודקודים בגרף.
 2. מתודה אבסטרקטית neighbors אשר בהינתן קודקוד קיים מחזירה את אוסף כל הקודקודים בגרף אשר יש קשת בינם ובין הקודקוד הנתון.
 3. מתודה connected שאינה אבסטרקטית אשר מקבלת כארגומנט קודקוד ומחזירה את אוסף הקודקודים אשר יש מסלול מקודקוד זה אליהם. על מתודה זו להיות יעילה במובן שאסור לה לקרוא למתודה neighbors של קודקוד מסוים יותר מפעם אחת.

לנוחותכם, להלן תיעוד הממשק Set:

java.util

Interface Set<E>

All Superinterfaces:

[Collection<E>](#), [Iterable<E>](#)

Some Implementing Classes:

[HashSet](#), [LinkedHashSet](#), [TreeSet](#)

```
public interface Set<E>
    extends Collection<E>
```

A collection that contains no duplicate elements...

Method Summary	
boolean	add (E o) Adds the specified element to this set if it is not already present (optional operation).
boolean	addAll (Collection<? extends E> c) Adds all of the elements in the specified collection to this set if they're not already present (optional operation).
void	clear () Removes all of the elements from this set (optional operation).
boolean	contains (Object o) Returns true if this set contains the specified element.
boolean	containsAll (Collection<?> c) Returns true if this set contains all of the elements of the specified collection.
boolean	equals (Object o) Compares the specified object with this set for equality.
int	hashCode () Returns the hash code value for this set.
boolean	isEmpty () Returns true if this set contains no elements.
Iterator<E>	iterator () Returns an iterator over the elements in this set.
boolean	remove (Object o) Removes the specified element from this set if it is present (optional operation).

boolean	<code>removeAll(Collection<?> c)</code> Removes from this set all of its elements that are contained in the specified collection (optional operation).
boolean	<code>retainAll(Collection<?> c)</code> Retains only the elements in this set that are contained in the specified collection (optional operation).
int	<code>size()</code> Returns the number of elements in this set (its cardinality).
<code>Object[]</code>	<code>toArray()</code> Returns an array containing all of the elements in this set.
<code><T> T[]</code>	<code>toArray(T[] a)</code> Returns an array containing all of the elements in this set; the runtime type of the returned array is that of the specified array.

השתמשו במקום זה כדי לענות על סעיף ב':

```
public abstract class Graph {

    public abstract Set<Vertex> vertices();
    public abstract Set<Vertex> neighbours(Vertex v);

    public Set<Vertex> connected(Vertex v){

        Set<Vertex> visited = new HashSet<Vertex>();
        LinkedList<Vertex> visiting = new LinkedList<Vertex>();

        visited.add(v);
        visiting.add(v);

        while(!visiting.isEmpty()){

            Vertex curr = visiting.remove();
            for (Vertex vertex : neighbours(curr)) {
                if(!visited.contains(vertex)){
                    visited.add(vertex);
                    visiting.add(vertex);
                }
            }
        }
        return visited;
    }
}
```

הערות על הבדיקה וטעויות שכיחות:

- לא ניתן לשנות מבנה של Iterable תוך כדי שעובר עליו האיטרטור – מצב זה אינו מוגדר
- במימושים אשר החזיקו מצב פנימי של הגרף, אסור לשרות `connected` לשנות את מבנה זה
- פתרון שכיח נוסף היה פתרון ברקורסיה: במקרה זה יש לשים לב כי המשתנים המקומיים אינם נגישים בקריאה הרקורסיבית

ג. ממשו במלואה את המחלקה MatrixGraph שהיא הרחבה של Graph. המחלקה מקבלת בבנאי שלה מטריצה ריבועית, משולשית עליונה עם אלכסון אפס, כמערך דו מימדי של int (אין צורך לבדוק זאת). הגרף המתואר ע"י המטריצה הוא הגרף בו יש קודקוד המשויך לכל שורה במטריצה, וקשת לכל איבר במטריצה אשר שונה מאפס. כך, למשל, אם בשורה i ובעמודה j ישנו ערך שונה מאפס אזי יש קשת (i,j) בגרף.

לדוגמא, המטריצה הזו:

0	3	0	-5
0	0	0	19
0	0	0	0
0	0	0	0

וגם זו:

0	23	0	54
0	0	0	1092
0	0	0	0
0	0	0	0

הן שתיים מהמטריצות המשויכות לגרף הבא:

$V = \{ V1, V2, V3, V4 \}$
 $E = \{ \{V1, V2\}, \{V1, V4\}, \{V2, V4\} \}$

(שימו לב, עליכם לאתחל את הקודקודים בעצמכם, שמות המופעים הינם שרירותיים)

מימוש א' (ללא שימוש ב Pair):

```
public class MatrixGraph extends Graph {
    public MatrixGraph(int[][] mat) {
        this.mat = mat;
        vertices = new Vertex[mat.length];
        for (int i = 0; i < mat.length; i++) {
            vertices[i] = new Vertex();
        }
    }
    @Override
    public Set<Vertex> neighbours(Vertex v) {
        Set<Vertex> result = new HashSet<Vertex>();
        int index = -1;
        // find index of v argument
        for (int i = 0; i < vertices.length; i++) {
            if (vertices[i] == v){
                index = i;
                break;
            }
        }
        for (int i = 0; i < vertices.length; i++) {
            if (mat[index][i] != 0)
                result.add(vertices[i]);
        }
        return result;
    }
}
```

```

@Override
public Set<Vertex> vertices() {
    Set<Vertex> result = new HashSet<Vertex>();

    for (int i = 0; i < vertices.length; i++)
        result.add(vertices[i]);

    return result;
}

private int[][] mat;
private Vertex[] vertices;
}

```

מימוש חלופי (עם שימוש ב Pair):

```

public class MatrixGraph extends Graph {

    public MatrixGraph(int[][] mat) {
        vertices = new Vertex[mat.length];
        edges = new HashSet<Pair<Vertex>>();

        for (int i = 0; i < mat.length; i++) {
            vertices[i] = new Vertex();
        }

        for (int i = 0; i < mat.length; i++) {
            for (int j = 0; j < mat.length; j++) {
                if(mat[i][j]!=0)
                    edges.add(new Pair<Vertex>(vertices[i],vertices[j]));
            }
        }
    }

    @Override
    public Set<Vertex> neighbours(Vertex v) {
        Set<Vertex> result = new HashSet<Vertex>();
        for (Pair<Vertex> p : edges) {
            Object[]arr = p.toArray();    // ugly! ugly! ugly!
            Vertex v0 = (Vertex) arr[0]; // ugly! ugly! ugly!
            Vertex v1 = (Vertex) arr[1]; // ugly! ugly! ugly!
            if(v.equals(v0))
                result.add(v1);
            if(v.equals(v1))
                result.add(v0);
        }
        return result;
    }

    @Override
    public Set<Vertex> vertices() {
        Set<Vertex> result = new HashSet<Vertex>();
        for (int i = 0; i < vertices.length; i++)
            result.add(vertices[i]);
        return result;
    }

    private Vertex[] vertices;
    private Set<Pair<Vertex>> edges;
}

```

הערות על הבדיקה וטעויות שכיחות:

- אין להחזיר את השדה vertices – הדבר יוצר דליפה של הייצוג הפנימי
- לא היה צורך לשכתב את המחלקה Vertex – שרות ברירת המחדל equals מספיק

• שאלה 3, 24 נקודות

נתונות המחלקות הבאות:

```
public class Base {
    protected int i = 0;
    public Base(int i) { this.i = i; }
    public Base(Base b) { this(b.i); }
    public Base foo() { return new Base(this); }
}

public class Sub extends Base {
    public Sub(int i) { super(i); }
    public Sub(Sub s) { super(s.i * 2); }
    public Base foo() { return this; }
    public boolean equals(Object o) { return ((Sub) o).i == this.i; }

    public static void main(String[] args) {
        Base b1 = new Base(1);
        Base b2 = b1;
        Base b3 = b2.foo();
        Base b4 = new Sub(1);
        Base b5 = b4.foo();

        // HERE
    }
}
```

עבור כל אחת משורות הקוד הבאות בדקו האם ניתן להגדירה בפונקציית ה-main של המחלקה Sub במקום ההערה // HERE. סמנו את האפשרות המתאימה: האם תהיה שגיאת קומפילציה (אם כן, ציינו מה השגיאה), או שתהיה תעופה בזמן ריצה (אם כן, ציינו מאיזה סיבה) או שהקוד ירוץ בצורה תקינה (אם כן, ציינו מה יהיה פלט ההדפסה). נמקו בקצרה.

א.

System.out.println(b1 == b3);

שגיאת קומפילציה (מהי?) / תעופה בזמן ריצה (מה הסיבה) / קוד תקין (מה מודפס)

ב.

System.out.println(b1.equals(b3));

שגיאת קומפילציה (מהי?) / תעופה בזמן ריצה (מה הסיבה) / קוד תקין (מה מודפס)

ג.

```
System.out.println(b1.equals(b4));
```

שגיאת קומפילציה (מהי?) / תעופה בזמן ריצה (מה הסיבה) / קוד תקין (מה מודפס)

ד.

```
System.out.println(b4.equals(b1));
```

שגיאת קומפילציה (מהי?) / תעופה בזמן ריצה (מה הסיבה) / קוד תקין (מה מודפס)

ה.

```
System.out.println(b5 == b4);
```

שגיאת קומפילציה (מהי?) / תעופה בזמן ריצה (מה הסיבה) / קוד תקין (מה מודפס)

ו.

```
System.out.println(b5.equals(b4));
```

שגיאת קומפילציה (מהי?) / תעופה בזמן ריצה (מה הסיבה) / קוד תקין (מה מודפס)

ושוב, בהצלחה!