

פתרון הבחינה בתוכנה 1

סמסטר א', מועד א', תש"ע
07/02/2010

דן הלפרין, אוהד ברזילי, אסף זריצקי, 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. יש לצרף את טופס המבחן למחברת הבחינה. מחברת ללא טופס עזר תיפסל. 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.
- ניתן להוסיף הנחות לגבי אופן השימוש בשירותים המופיעים בבחינה, ובלבד שאין הן סותרות את תנאי השאלה. יש לתעד הנחות אלו כחוזה (תנאי קדם, תנאי בתר) בתחביר המקובל, שיכתב בתחילת השרות.
- אסור השימוש בחומר עזר כלשהו, כולל מחשבוניו או כל מכשיר אחר פרט לעט. בסוף הבחינה צורך לנוחותכם נספח ובו תיעוד מחלקות שימושיות.

לשימוש הבודקים בלבד:

שאלה	א	ב	ג	ד	ה	סה"כ
1	5	4	6	10	10	35
2	10	10	15			35
3	5	5	5			15
4	15					15

100

בהצלחה!

כל הזכויות שמורות ©
מבלי לפגוע באמור לעיל, אין להעתיק, לצלם, להקליט, לשדר, לאחסן במאגר מידע, בכל דרך שהיא, בין מכונית ובין אלקטרונית או בכל דרך אחרת כל חלק שהוא מטופס הבחינה.

שאלה 1 (35 נקודות)

בקצה רמת גן יש מפעל המייצר קופסאות שוקולד (בונבונירות). בכל קופסת שוקולד ניתן למצוא מגוון פרליני שוקולד מסוגים שונים, בצורות ובגדלים שונים. מבנה מודל של קופסת השוקולד (ChocolateBox) המכילה פרליני שוקולד (Praline)

נתונים המנשקים והמחלקות הבאים:

- המנשק Shape בו מוגדר שירות יחיד `getArea()` המחזיר את שיטחה (בסמ"ר) של צורה.

```
public interface Shape {
    double getArea();
}
```

- המחלקה האבסטרקטית Praline מייצגת פרלין שוקולד. כל פרלין הוא צורה, והשרות `getArea` מתייחס לשטח בסיסו. בנוסף יש לו משקל וכן מצוין אחוז (מוצקי) הקקאו ממשקל הפרלין.

```
public abstract class Praline implements Shape {
    private double weight;
    private double cocoaPrecentage;

    public Praline(double weight, double cocoaPrecentage) {
        this.weight = weight;
        this.cocoaPrecentage = cocoaPrecentage;
    }

    public double getWeight() {
        return weight;
    }

    public double getCocoaPercentage() {
        return cocoaPrecentage;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null) return false;
        if (!(obj instanceof Praline)) return false;
        Praline other = (Praline) obj;
        return cocoaPrecentage == other.cocoaPrecentage &&
            weight == other.weight;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        long temp;
        temp = Double.doubleToLongBits(cocoaPrecentage);
        result = prime * result + (int) (temp ^ (temp >>> 32));
        temp = Double.doubleToLongBits(weight);
        result = prime * result + (int) (temp ^ (temp >>> 32));
        return result;
    }
}
```

א. (5 נקודות) הגדירו את המחלקה SquarePraline (שאינה מופשטת) שיורשת מהמחלקה Praline ומגדירה פרלין שצורת בסיסו ריבועית. המחלקה מגדירה בנאי המקבל את אורך צלע הריבוע וכן נתונים אחרים הדרושים למימוש תקין של המחלקה.

```
public class SquareParline extends Praline {
    private double length;

    public SquareParline(double weight, double cocoaPrecentage,
        double length) {
        super(weight, cocoaPrecentage);
        this.length = length;
    }

    @Override
    public double getArea() {
        return length * length;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = super.hashCode();
        long temp;
        temp = Double.doubleToLongBits(length);
        result = prime * result + (int) (temp ^ (temp >>> 32));
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (!super.equals(obj))
            return false;
        if (!(obj instanceof SquareParline))
            return false;
        SquareParline other = (SquareParline) obj;
        if (Double.doubleToLongBits(length) !=
            Double.doubleToLongBits(other.length))
            return false;
        return true;
    }
}
```

ב. (4 נקודות) המחלקה `ChocolateBox` מייצגת קופסת שוקולד. הפרלינים השונים בקופסה מסודרים בשורות כשבכל השורות מספר שווה של פרלינים. בחרו את הייצוג הפנימי של קופסת השוקולדים והשלימו את הבנאי.
 הערה: קיראו את השאלה עד סופה לפני שתענו על סעיף זה. בחירת הייצוג הפנימי המתאים עשויה להקל עליכם בסעיפים הבאים.

```
public class ChocolateBox {

    private int rows;
    private int columns;

    private Praline[][] pralines;

    public ChocolateBox(int rows, int columns) {

        this.rows = rows;
        this.columns = columns;
        this.pralines = new Praline[rows][columns];

    }

    // more methods...
```

ג. (6 נקודות) קופסת שוקולד חייבת לעמוד בסטנדרטים מסוימים של איכות לשם כך ממשו מתודת `isValid` בשם המחלקה `ChocolateBox` המוודאת ש:
 1. שטח הפרלינים (כולל) בקופסה נמצא בתחום ערכים מסוים המוגדר על ידי הגבולות העליון והתחתון (כולל) של התחום.
 2. שיעור הקקאו הכולל בקופסא גבוה מאחוז סף נתון.

(מקום נוסף בעמוד הבא)

```
public boolean isValid(double cocoaThreshold,
    double areaLowerBound, double areaUpperBound) {
    double area = 0;
    double cocoaWeight = 0, totalWeight = 0;

    for (Praline[] row : pralines) {
        for (Praline p : row) {
            area += p.getArea();
            cocoaWeight += p.getCocoaPercentage() * p.getWeight();
            totalWeight += p.getWeight();
        }
    }
    boolean isValidArea = area >= areaLowerBound &&
        area <= areaUpperBound;
    boolean isValidCocoaPercentage =
        cocoaWeight / totalWeight >= percentageThreshold;

    return isValidCocoaPercentage && isValidArea;
}
```

ד. (10 נקודות) קופסת שוקולד היא **k-מגוונת** (k varied) אם לא קיימת תת-מטריצה בגודל $k \times k$ שבה כל הפרלינים זהים בכל מאפניהם. **תת-מטריצה** היא בחירה של k עמודות ו- k שורות כלשהן (לא בהכרח ברצף).

לצורך מימוש השרות isVaried נממש תחילה מתודת עזר סטטית generateKSubsets המייצרת את כל תת הקבוצות בגודל k של שלמים בתחום $[first..last]$.

```
/**
 * @pre first <= last
 * @pre k >= 0
 * @pre last - first + 1 > k
 */
public static Set<Set<Integer>> generateKSubsets(
    int first, int last, int k) {
    Set<Set<Integer>> result = new HashSet<Set<Integer>>();

    if (k == 0) {
        result.add(new HashSet<Integer>());
        return result;
    }
    if (k < last - first + 1) {
        result.addAll(generateSubsets(first + 1, last, k));
    }
    for (Set<Integer> set :
        generateSubsets(first + 1, last, k - 1)) {
        set.add(first);
        result.add(set);
    }
    return result;
}
```

ה. (10 נקודות) בעזרת המתודה הסטטית מהסעיף הקודם ממשו את מתודת המופע isVaried במחלקה ChocolateBox, הבודקת האם הקופסה היא k-מגוונת.

```
public boolean isVaried(int k) {
    Set<Set<Integer>> rowsSubsets =
        generateSubsets(0, rows - 1, k);
    Set<Set<Integer>> colsSubsets =
        generateSubsets(0, columns - 1, k);

    for (Set<Integer> r : rowsSubsets) {
        for (Set<Integer> c : colsSubsets) {
            if (!isVaried(r, c)) {
                return false;
            }
        }
    }

    return true;
}

private boolean isVaried(Set<Integer> rows, Set<Integer> cols) {
    Praline p =
        pralines[rows.iterator().next()][cols.iterator().next()];

    for (int i : rows) {
        for (int j : cols) {
            if (!pralines[i][j].equals(p))
                return true;
        }
    }
    return false;
}
```

שאלה 2 (35 נקודות)

שאלה זו עוסקת בתכנון ובהגדרת טיפוס נתונים עבור פולינום (polynomial – רב איבר) שמקדמיו ממשיים. בשאלה נדון בטיפוס מקובע (immutable). טיפוס מקובע הוא טיפוס שלא ניתן לשנות את המופעים שלו לאחר שנוצרו.

- **פולינום** הוא ביטוי מהצורה: $c_n x^n + c_{n-1} x^{n-1} + \dots + c_2 x^2 + c_1 x + c_0$
- $c_n, c_{n-1}, \dots, c_2, c_1, c_0$ הם מספרים ממשיים, והם נקראים **המקדמים** של הפולינום (coefficients).
- המעריך הגדול ביותר בפולינום (של איבר שהמקדם שלו אינו אפס) נקרא **הדרגה** (degree) של הפולינום.

לדוגמא: הביטוי $2x^3 - 5x^2 + 13$ הוא פולינום שדרגתו 3 ומקדמיו הם: 2, -5, 0, 13

נתון המנשק IPolynomial ובו שירותים המתוארים בצורה מילולית:

```

/** פולינום מקובע (immutable) שמקדמיו ממשיים */
public interface IPolynomial {

    /** השרות מחזיר את דרגת הפולינום */
    public int degree ( );

    /** השרות מחזיר את המקדם של האיבר עם מעריך d */
    public double coeff (int d);

    /** השרות מחזיר את הסכום של הפולינום הנוכחי ושל הארגומנט q */
    public IPolynomial add (IPolynomial q);

    /** השרות מחזיר את המכפלה של הפולינום הנוכחי ושל הארגומנט q */
    public IPolynomial mul (IPolynomial q);

    /** מחזיר את תוצאת חיסור q מהפולינום הנוכחי */
    public IPolynomial sub (IPolynomial q);

}
    
```

להלן תזכורת מתמטית:

סכום של שני פולינומים הוא פולינום שהמקדמים שלו הינם סכומים של המקדמים המתאימים (לפי חזקות) של שני הפולינומים הנתונים. למשל, עבור $a(x) = 6x^3 + 3x^2 + 8$ ו-
 $b(x) = 2x^2 + 4x + 1$ נקבל: $a(x) + b(x) = 6x^3 + 5x^2 + 4x + 9$.

מכפלה של שני פולינומים הינה פולינום, אשר מתקבל כתוצאה של "פתיחת סוגריים" רגילה בביטוי:

$$a(x) \cdot b(x) = (a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0) \cdot (b_m x^m + b_{m-1} x^{m-1} + \dots + b_1 x + b_0)$$

לדוגמה, עבור $a(x) = 3x^2 + x + 2$ ו- $b(x) = x^3 + 4x$ נקבל:

$$a(x) \cdot b(x) = (3x^2 + x + 2) \cdot (x^3 + 4x) = 3x^{2+3} + 3 \cdot 4x^{2+1} + x^{1+3} + 4x^{1+1} + 2x^3 + 8x =$$

$$= 3x^5 + 12x^3 + x^4 + 4x^2 + 2x^3 + 8x = 3x^5 + x^4 + 14x^3 + 4x^2 + 8x$$

א. (10 נקודות) לאומה הסתומה אין מושג ירוק איך לממש את המנשק הנתון, אולם ביל הדביל טוען שאם היא תגדיר במחלקה מופשטת (abstract class) שרות מפעל מופשט (abstract factory method) בשם createPolynomial, היא תוכל בעזרתו לממש כמעט את כל שרותי המנשק, פרט ל-degree-ול-coeff.

עזרו לאומה לממש את המחלקה AbstPolynomial כפי שהציע ביל, הימנעו משכפול קוד ככל הניתן:

```
public abstract class AbstPolynomial implements IPolynomial {

    abstract public int degree();

    abstract public double coeff(int d);

    abstract protected IPolynomial createPolynomial(double... coeffs);

    @Override public IPolynomial add(IPolynomial q) {
        double[] result = new double[Math.max(degree(), q.degree()) + 1];
        for (int i = 0; i < result.length; i++) {
            result[i] = coeff(i) + q.coeff(i);
        }
        return createPolynomial(result);
    }

    @Override public IPolynomial mul(IPolynomial q) {
        double[] result = new double[degree() + q.degree() + 1];
        for (int i = 0; i <= degree(); i++)
            for (int j = 0; j <= q.degree(); j++)
                result[i + j] += coeff(i) * q.coeff(j);
        return createPolynomial(result);
    }

    @Override public IPolynomial sub(IPolynomial q) {
        return add(q.mul(createPolynomial(-1)));
    }
}
```

טעויות חוזרות:

- קריאה ל q.mul(-1) - מקבל IPolynomial כארגומנט, לא int
- הקצאת גודל מערך העזר להיות degree(), במקום degree()+1
- שכפול קוד במימוש sub
- שגיאה במימוש אלגוריתם הכפל: result[i + j]= במקום result[i + j] +=

ב. (10 נקודות) אומה הסתומה עדיין לא יודעת איך לממש את המחלקה, אולם לביל הדביל יש רעיון: הוא מצא בספר המתמטיקה שלו מכיתה ח' הגדרה ולפיה פולינום הוא סכום של מונומים (חד איברים). למשל: הפולינום $p(x) = 6x^3 - 3x^2 + 8$ הוא סכום של המונומים m_1, m_2, m_3 כך ש: $m_1(x) = 6x^3, m_2(x) = -3x^2, m_3(x) = 8$.

לאחר חיפוש קצר באינטרנט הוא אפילו מצא את המנשק הבא:

```
/** חד איבר עם מקדם ממשי */
public interface IMonom {

    public double getCoeff();

    public int getDegree();

}
```

ביל מסביר לאומה שהיא תוכל לממש פולינום כרצף של מונומים ע"י שימוש בטיפוס `Iterable<IMonom>` (אין הכרח שהמונומים יופיעו בסדר כלשהו) ואף ניתן לה קובץ שלד לשם כך.

עזרו לאומה להשלים את מימוש השרותים `coeff` ו-`degree` של המחלקה המופשטת `AbstMonomialPolynomial`. (מקום נוסף בעמוד הבא):

```
public abstract class AbstMonomialPolynomial extends AbstPolynomial {

    abstract public IPolynomial createPolynomial(double... coeffs);

    protected Iterable<IMonom> monoms;

    @Override public double coeff(int d) {

        for (IMonom monom : monoms) {
            if (monom.getDegree() == d)
                return monom.getCoeff();
        }
        return 0;
    }

}
```

טעויות חוזרות:

- בלבול בין `Iterable` ו-`Iterator` (בדרך כלל התבטא בקריאה ל `monoms.next()`)
- אי החזרת ערך ברירת מחדל (מחוץ ללולאה) הוא החזרת ערך שגוי -1) אינו יכול להיות מוחזר כערך שגיאאה(!)

```
@Override public int degree() {  
  
    int maxDegree = 0;  
    for (IMonom monom : monoms) {  
        if (monom.getDegree() > maxDegree)  
            maxDegree = monom.getDegree();  
    }  
    return maxDegree;  
}
```

ג. (15 נקודות) עיזרו לאומה הסתומה ולביל הדביל לממש את המחלקה הקונקרטית `MonomialPolynomial`. אם לצורך מימוש המחלקה יש צורך בשרותי עזר נוספים, שדות, בנאים או טיפוסים עזר חדשים ממשו גם אותם.

```
public class MonomialPolynomial extends AbstMonomialPolynomial {

    public MonomialPolynomial(double... coeffs) {
        List<IMonom> monoms = new ArrayList<IMonom>();
        for (int i = 0; i < coeffs.length; i++) {
            // zero coeffs might affect the degree
            if(coeffs[i] != 0)
                monoms.add(new Monom(coeffs[i], i));
        }
        super.monoms = monoms;
    }

    @Override
    public IPolynomial createPolynomial(double... coeffs) {
        return new MonomialPolynomial(coeffs);
    }

    /** @inv: getCoeff() != 0 */
    private static class Monom implements IMonom {
        private double coeff;
        private int degree;

        /** @pre: coeff != 0 */
        public Monom(double coeff, int degree) {
            this.coeff = coeff;
            this.degree = degree;
        }

        @Override
        public double getCoeff() {
            return coeff;
        }

        @Override
        public int getDegree() {
            return degree;
        }
    }
}
```

טעויות חוזרות:

- המחלקה הפנימית אינה `static` - אין חובה לממש את `IMonom` כמחלקה פנימית אבל אם היא מומשה ככזו עליה להיות `static` בנאי
- בנאי
 - אי הגדרת בנאי
 - הגדרת בנאי פרטי יחיד
 - אי עדכון השדה `super.monoms` בבנאי
 - השמה של מערך לתוך `super.monoms` - מערך אינו `Iterable`
 - `super.monoms.add` - `Iterable` אין `add`
- `createPolynomial`
 - משנה את העצם הנוכחי (`this`)
 - אינו מחזיר עצם חדש
 - מחזיר טיפוס שאינו `IPolynomial`
- `new Set / new IMonom` - לא ניתן לייצר מופעים מחמשק

שאלה 3 (15 נקודות)

אומה מראה לביל קטע קוד שמצאה בשיעורי הבית בקורס Scheme ותרגמה לשפת Java:

```
public static int fibonacci(int n) {
    if (n == 0)
        return 0;
    if (n == 1)
        return 1;
    return fibonacci(n-1) + fibonacci(n-2);
}
```

אומה טוענת כי הקוד מחשב את האיבר ה-n בסדרת פיבונאצ'י: 0,1,1,2,3,5,8,13,...

א. (5 נקודות) ביל מסביר לאומה שהיא שכחה לציין את תנאי הקדם (precondition) של הפונקציה. האם תוכלו לסייע לאומה בניסוח תנאי הקדם עבור מימוש זה? השתמשו בתחביר פורמאלי חוקי במידת האפשר:

```
/** @pre: n >= 0
 * @pre: n < MAX_VALUE
 */
בשל המימוש הרקורסיבי של הפונקציה יש לה סיבוכיות זמן ריצה וזיכרון אקספוננציאליים, ולא ניתן להעביר ערכי n גדולים. הערך המדויק של MAX_VALUE אמנם תלוי בהגדרות ה-JVM והמכונה אולם הוא קטן (בהרבה מ-MAX_INT)
```

ב. (5 נקודות) ביל ממליץ לאומה לשכתב את הקוד כך שלא יהיו לשרות תנאי קדם כלל (precondition: true) אולם יהיו לו תנאי צד. ביל ממליץ לעשות שימוש בחריג לא נבדק (unchecked exception) מטיפוס IllegalArgumentException שימסך חריגים אשר עשויים להפתיע את לקוחות הפונקציה וייתן חיווי טוב יותר על אופי התקלות שקרו. עיזרו לאומה לשכתב את מימוש הפונקציה כפי שהמליץ לה ביל:

```
public static int fibonacci(int n) {
    if (n < 0)
        throw new IllegalArgumentException("n must not be negative");

    try {
        if (n == 0)
            return 0;
        if (n == 1)
            return 1;
        return fibonacci(n - 1) + fibonacci(n - 2);
    } catch (StackOverflowError e) {
        throw new IllegalArgumentException("n is too big");
    }
}
```

ג. (5 נקודות) אומה לא מרוצה מהמימוש הקודם ומחליטה לשכתב את מימוש הפונקציה כך שלא יהיו לה תנאי קדם או תנאי צד כלל (אינה זורקת חריגים, precondition: true). עיזרו לאומה לשכתב את הפונקציה והשלימו את החוזה שלה (יש יותר מפתרון אחד אפשרי). השתמשו בתחביר פורמאלי חוקי במידת האפשר:

```
/**
 * @pre: true
 *
 * @post: n<0 $implies $ret=-1
 *        n>=0 returns the n'th Fibonacci number
 */
public static int fibonacci(int n) {
    public static int fibonacci(int n) {
        if (n < 0)
            return -1;

        if (n == 0)
            return 0;

        if (n == 1)
            return 1;

        int prev = 1;
        int curr = 1;

        for (int i = 2; i < n; i++) {
            curr += prev;
            prev = curr - prev;
        }

        return curr;
    }
}
```

פתרון חלופי:

```
/**
 * @pre: true
 *
 * @post: n<0 || n too big $implies $ret=-1
 *        else: returns the n'th Fibonacci number
 */
public static int fibonacci(int n) {
    if (n < 0)
        return -1;

    try {
        return fibonacci_helper(n);
    } catch (Exception e) {
        return -1;
    }
}

public static int fibonacci_helper(int n) {
    if (n == 0)
        return 0;
    if (n == 1)
        return 1;
    return fibonacci_helper(n-1) + fibonacci_helper(n-2);
}
```

מפתח ניקוד לשאלה 3

סעיף א:

ניקוד	טעות
-1	חסר תנאי עבור $n == 0$
-1 לכל תנאי מיותר	תנאים לא רלוונטיים (לדוגמא: $n != null$, n שלם)
בנוסף +1 למי שהתייחס	חסר חסם עליון על n

סעיף ב:

ניקוד	טעות
בנוסף +1 למי שטיפל	חסר טיפול ב-Stack overflow
-1	הצהרה על השירות כ- throws Exception (במקום IllegalArgumentException, או ללא throws)
-1 או -2 כתלות בטעות	מימוש שגוי של IllegalArgumentException (חריג קיים)
-4 או -5 כתלות בטעות	השירות לא זורק אף פעם חריג (מקרה פרטי: זורק ותופס בעצמו)
-2	זריקת חריגה על $n == 0$
-1	try-catch מיותר (בדו"כ על הקריאה הרקורסיבית)
-2	חסר טיפול במקרה ש- $n < 0$
-1	טעות משמעותית בתחביר בזריקת חריג

סעיף ג:

בשאלה זו החוזה "שווה" 2 נקודות והמימוש 3 נקודות.

ניקוד	טעות
-2	החזרת null
-1	try-catch מיותר
-2	שימוש בחריגים
-1	אין התייחסות לטווח בחוזה (בתנאי הבתר)
-1	החזרת 0 על $n < 0$ (זהה למספר פיבונאצ'י של 0)
-1	חסרה התייחסות בחוזה לטיפול בקלט תקין
-1	העתקה 1-1 של המימוש לחוזה
-1	קליטת קלט מהמשתמש לתיקון קלט של מספר שלילי לשירות
-2	השירות אינו מחזיר ערך עבור קלט לא חוקי

שאלה 4 (15 נקודות)

בזמן חיפושיו באינטרנט בשאלה 2 נתקל ביל במחלקות חדשות המטפלות בקלט/פלט ובפרט במחלקה Path. לדאבונו הוא גילה כי מחלקות אלו ייכנסו לסטנדרט רק בגירסה הבאה של Java. ביל אינו רוצה לחכות כדי להשתמש ב-API של מחלקות אלו, ולפיכך הוא מממש מחלקות המגדירות בדיוק את אותם שירותים פומביים, הממומשים בעזרת המחלקות שכבר קיימות בשפה.

תחילה הגדיר ביל את CopyOption:

```
public enum CopyOption {
    ATOMIC_MOVE,
    COPY_ATTRIBUTES,
    REPLACE_EXISTING;
}
```

לאחר מכן פנה לממש את המחלקה Path מיועדת להחליף את המחלקה File הקיימת כיום. ביל מחליט לממש את Path בעזרת File.

ביל מימש את השירות copyTo המעתיקה קובץ למקום אחר על הדיסק, אבל נתקע באמצע המימוש של פונקציות נעזר שלה. עיזרו לו להשלים את המימוש של פונקציות העזר copyToFile:

הפונקציה מקבלת שני פרמטרים: target המציין את המיקום החדש של הקובץ ו-options שהוא מערך של אפשרויות פעולה. בשלב ראשון החליט ביל לממש רק את האפשרות REPLACE_EXISTING. בהעדרה של אפשרות זו במערך, השירות לא יכתוב על קובץ שכבר קיים על הדיסק.

הקוד שביל כבר מימש:

```
public class Path {
    private File file;

    public Path(String pathname) {
        file = new File(pathname);
    }

    public void copyTo(Path target, CopyOption... options)
        throws IOException {
        if (file.isDirectory()) {
            copyToDir(target, options);
        } else {
            copyToFile(target, options);
        }
    }

    private void copyToDir(Path target, CopyOption... options)
        throws IOException {
        ...
    }

    private void copyToFile(Path target, CopyOption... options)
        throws IOException {
        ...
    }
}
```

השלימו את המתודה copyToFile המעתיקה קובץ (לא ספרייה) למקום חדש בדיסק.

```
private void copyToFile(Path target, CopyOption... options)
throws IOException {
    // No need to copy a file to itself
    if (this == target)
        return;

    // should we replace an existing file?

    if (!Arrays.asList(options).
        contains(CopyOption.REPLACE_EXISTING) &&
        target.file.exists()) {
        return;
    }

    FileInputStream in = null;
    FileOutputStream out = null;
    try {

        in = new FileInputStream(this.file);
        out = new FileOutputStream(target.file);

        int c;
        while ((c = in.read()) != -1) {
            out.write(c);
        }

    } finally {
        closeIgnoreException(in);
        closeIgnoreException(out);
    }
}
```

ושוב, בהצלחה!