

פתרון הבחינה בתוכנה 1

סמסטר א', מועד א', תשע"א

6/2/2011

אוהד ברזילי, דן הלפרין, רובי בוים, מתי שמרת

הוראות (נא לקרוא!)

- משך הבחינה **שלוש שעות**, חלקו את זמנכם ביעילות.
- אסור השימוש בחומר עזר כלשהו, כולל מחשבוניו או כל מכשיר אחר פרט לעט. בסוף הבחינה צורף לנוחותכם נספח ובו תיעוד מחלקות שימושיות.
- 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, שייכתב בתחילת השרות.

לשימוש הבודקים בלבד:

שאלה	א	ב	ג	ד	ה	סה"כ
1						
2						
3						
4						

בהצלחה!

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

שאלה 1 (20 נקודות)

א. (15 נקודות) ממשו את השרות `maxProfit` אשר בהינתן מערך של מחירי מנייה לאורך תקופה כלשהי (באגורות) יחזיר את הרווח הגדול ביותר (באגורות) שניתן היה להשיג מקניית המנייה בזמן כלשהו ומכירתה בזמן אחר, מאוחר יותר. הרווח מחושב ע"י חיסור המחיר בעת המכירה מהמחיר בזמן הקניה.

לדוגמא, המערך `[13, 11, 12, 12, 14]` מתאר מנייה אשר ביום הראשון נסחרה ב-13 אגורות, ביום השני ב-11, בימים השלישי והרביעי ב-12, וביום החמישי ב-14 אגורות. קניה של המנייה ביום הראשון ומכירתה ביום האחרון תניב רווח של אגורה אחת. קנייה של המנייה ביום השני ומכירתה בחמישי תניב רווח של 3 אגורות – שהוא גם הרווח המקסימלי במקרה זה.

להלן כמה דוגמאות ריצה של הפונקציה:

`maxProfit ([101,103, 106, 105, 105, 107, 106]) → 6`

`maxProfit ([50, 49, 47, 46, 41]) → -1`

ניתן להגדיר מבני עזר או שרותים חדשים לצורך המימוש. בסעיף זה אין התייחסות לסיבוכיות זמן הריצה של האלגוריתם. אם יש לכם הנחות לגבי הקלט של הפונקציה ציינו אותן במפורש בתחביר פורמלי ככל האפשר ע"י שימוש בטענות עיצוב בעזרת חוזה, בראש הפונקציה:

```

/ **
 * @pre: prices != null
 * @pre: prices.length >= 2
 */

public static int maxProfit (int [] prices) {
    int max = prices[1] - prices[0];
    for (int i = 0; i < prices.length; i++) {
        for (int j = i+1; j < prices.length; j++) {
            if (prices[j] - prices[i] > max){
                max = prices[j] - prices[i];
            }
        }
    }
    return max;
}

```

שגיאות נפוצות:

- אתחול `max` ל-0: הורדו 3 נקודות
- החזרת 0 או `prices[0]` כערך שגיאה: הורדו 2 נקודות
- אתחול `j` ל-`i` או ל-0: הורדו 2 נקודות
- תנאי קדם חסר: הורדו 2 נקודות על כל תנאי חסר
- ציון תנאי הקדם (הסגוי!) `!prices[i]` הורדה נקודה
- שגיאה אלגוריתמית: הורדו 11 נקודות

ב. (5 נקודות) נסחו במדויק את תנאי האחר של הפונקציה שמימשתם תוך שימוש בכמתי לכל (\forall) וקיים (\exists) :

קיום: $\exists i, j s. t. 0 \leq i < j < prices.length. \$ret = prices[j] - prices[i]$

מקסימליות: $\forall i, j s. t. 0 \leq i < j < prices.length. \$ret \geq prices[j] - prices[i]$

שגיאות נפוצות:

- השמטת תכונת הקיום או המקסימליות: הורדו 2 נקודות
- שימוש באי שוויון חזק בתנאי המקסימליות: הורדה נקודה
- אי התייחסות לערך המוחזר ($\$ret$): הורדה נקודה
- שימוש בערך מוחלט של הפרש: הורדו 2 נקודות
- אי הקפדה על תחום ערכי i, j : הורדו 2 נקודות
- אי ציון $i > j$ או שימוש ב \geq : הורדה נקודה

שאלה 2 (40 נקודות)

א. (10 נקודות) מחסנית שלמים מקסימלית (MaxStackOfInts) הוא מנשק דומה למחסנית, אשר מאפשר לברר בכל שלב גם מהו הערך המקסימלי השמור במחסנית.

```
public interface MaxStackOfInts {  
  
    // Pushes an integer onto the top of this stack.  
    public void push (int i);  
  
    // Removes the integer at the top of this stack.  
    public void pop ();  
  
    // Looks at the integer at the top of this stack without  
    // removing it from the stack.  
    public int top ();  
  
    // Tests if this stack is empty.  
    public boolean isEmpty();  
  
    // Returns the maximum element of the given stack.  
    public int max();  
}
```

ממשו מחלקה המממשת את המנשק (ניתן להיעזר במחלקות המתועדות בנספח). אם ברצונכם להוסיף הנחות נוספות על השימוש במחלקה ציינו אותן כהערות מעל מימוש המחלקה או השרותים בתחביר פורמאלי ככל הניתן.

ראו מקום לפתרון בעמוד הבא

```
public class SimpleMaxStackofInts implements MaxStackOfInts {
    private List<Integer> impl = new ArrayList<Integer>();

    @Override
    public void push(int i) {
        impl.add(0, i);
    }

    // @pre: !isEmpty()
    @Override
    public void pop() {
        impl.remove(0);
    }

    // @pre: !isEmpty()
    @Override
    public int top() {
        return impl.get(0);
    }

    @Override
    public boolean isEmpty() {
        return impl.isEmpty();
    }

    // @pre: !isEmpty()
    @Override
    public int max() {
        return Collections.max(impl);
    }
}
```

שגיאות נפוצות:

- אתחול משתנה MAX עם 0 או 1
- List<int>
- אי עדכון MAX לאחר POP
- הכנסה איברים לסוף הרשימה והוצאה מהתחלתה

ב. (5 נקודות) לביל יש רעיון להכליל את המנשק `MaxStackOfInts` כדי שיטפל בעוד טיפוסים (ולא רק במספרים שלמים). ביל מציע את המנשק `MaxStack<T extends Comparable<T>>`

ממשו את כל השינויים שיש לבצע במחלקה מסעיף א' בשביל לתמוך בהכללת הטיפוסים. בתשובתכם יש להתייחס הן לשינויים בחתימות השרותים והן לשינויים הנדרשים במימוש.

```
public class SimpleMaxStack<T extends Comparable<T>>
    implements MaxStack<T> {
    private List<T> impl = new ArrayList<T>();

    @Override
    public void push(T element) {
        impl.add(0, element);
    }

    @Override
    public void pop() {
        impl.remove(0);
    }

    @Override
    public T top() {
        return impl.get(0);
    }

    @Override
    public boolean isEmpty() {
        return impl.isEmpty();
    }

    @Override
    public T max() {
        return Collections.max(impl);
    }
}
```

שגיאות נפוצות:

- כותרת מחלקה `implements MaxStack<T extends Comparable<T>>`
- שימוש ב-`<T extends Comparable<T>>` בטיפוס המשתנה בחתימות הפונקציות
- שימוש ב-`<` במקום ב-`compareTo`

ג. (5 נקודות) אומה לא מבינה מדוע ביל בחר חתימה כל כך מסובכת למנשק MaxStack ומציעה

את החתימה הבאה: `MaxStack<T extends Comparable>`

ביל טוען שהשימוש בחתימה שהציעה אומה מסוכן ואם המחלקה שמימשתם בסעיף ב' תממש מנשק זה (במקום את המנשק שהציע ביל) הדבר עלול לאפשר שימושים לא רצויים.

הניחו כי המחלקה שמימשתם בסעיף ב' מממשת את המנשק של אומה ולא את המנשק של ביל, וכי תבו קטע קוד המשתמש במחלקה ומדגים את בעיית בטיחות הטיפוסים שאליה התייחס ביל.

התקבלו דוגמאות שהראו כי צד שימוש במנשק המוצע פגם בעקרון בטיחות הטיפוסים. לדוגמא:

```
MaxStack<Comparable> stack = new MaxStack<Comparable>();
stack.push("hello");
stack.push(new Integer(5));
int max = stack.max(); // can't compare strings with integers
```

שגיאות נפוצות:

- הגדרת מחסנית של Number למרות שלא מוחש Comparable
- הגדרת מחסנית של Object למרות שלא מוחש Comparable

ד. (10 נקודות) ביל מעוניין להשתמש במחסנית מסעיף ב' גם לטיפוסים שאינם מממשים את הממשק `Comparable<T>`. אומה מציעה לו לשנות את חתימת הממשק `MaxStack` להיות `MaxStack<T>`.

ממשו את כל השינויים שיש לבצע במחלקה שמימשתם בסעיף ב' בשביל לתמוך בחתימה החדשה שהציעה אומה. במידת הצורך, ניתן להגדיר שדות, שרותים, בנאים או מבני עזר חדשים לצורך המימוש.

```
public class SimpleMaxStack<T> implements MaxStack<T> {
    private List<T> impl;
    private Comparator<? super T> comparator;

    public SimpleMaxStack() {
        impl = new ArrayList<T>();
    }

    public SimpleMaxStack(Comparator<? super T> comparator) {
        this.comparator = comparator;
    }

    // push, pop, top and isEmpty did not change

    @Override
    public T max() {
        T candidate = impl.get(0);
        for (T element : impl) {
            if (compare(element, candidate) > 0) {
                candidate = element;
            }
        }
        return candidate;
    }

    @SuppressWarnings("unchecked")
    private int compare(Object o1, Object o2) {
        return comparator == null ?
            ((Comparable<? super T>)o1).compareTo((T)o2) :
            comparator.compare((T)o1, (T)o2);
    }
}
```

שגיאות נפוצות:

- אי מתן `Comparator` - במקום, בדיקה האם המחלקה מממשת את `Comparable` ואם לא שימוש בערך אחר (כגון `toString`, זריקת פריג)
- אי החזרת `MAX`
- העברת פונקציית `compareTo` בצורה לא נכונה

ה. (10 נקודות) ברשת החברתית Facebook שוקלים מחדש את מימוש תכונת ה- notifications (עדכונים). בכל פעם שמתמש לוחץ על לשונית ה- notifications בעמוד הפרופיל שלו, יוצגו לו כל העדכונים האחרונים שהוזנו עבורו, מהעדכני ביותר ועד לישן ביותר. עדכונים שכבר הוצגו למשתמש לא יוצגו בפעם הבאה כשהמשתמש ילחץ על לשונית ה- notifications. כמו כן, בכל פעם שיוצגו העדכונים, המערכת תקיף במסגרת צהובה את העדכון שעשוי להיות המעניין ביותר עבור המשתמש, על פי מספר ה-like של אותו עדכון.

```
public interface Notification {
    public int getNumOfLikes();
    public void postToNotificationPanel(boolean isHighlighted);
}
```

בשאלה זו אנו נממש רכיב המממש את המנשק NotificationManager אשר יהיה אחראי על מימוש הלוגיקה שפורטה לעיל.

- כדי להוסיף עדכונים לרכיב ה- NotificationManager יש להפעיל עליו את השרות addNotification עם העדכון החדש.
- בכל פעם שמתמש לוחץ על לשונית ה- notifications בעמוד הפרופיל שלו המערכת קוראת לשרות showNotificationTab אשר אחראי לפרסם את העדכונים המתאימים בחלון שנפתח (ע"י הפעלה של postToNotificationPanel על העדכונים המתאימים). כדי להקיף עדכון במסגרת צהובה יש להעביר לשרות את הערך true.

```
public interface NotificationManager {
    public void showNotificationTab();
    public void addNotification(Notification notification);
}
```

ממשו מחלקה המממשת את המנשק NotificationManager. אם ברצונכם להוסיף הנחות נוספות על השימוש במחלקה ציינו אותן כהערות מעל מימוש המחלקה או השרותים בתחביר פורמאלי ככל הניתן. אין צורך לממש את המנשק Notification.

מקום נוסף בעמוד הבא

```
public class SimpleNotificationManager implements
    NotificationManager {

    private SimpleMaxStack<Notification> notifications;

    public SimpleNotificationManager() {
        notifications = new SimpleMaxStack<Notification>(
            new Comparator<Notification>() {
                @Override
                public int compare(Notification n1,
                    Notification n2) {
                    return n1.getNumOfLikes() -
                        n2.getNumOfLikes();
                }
            });
    }

    @Override
    public void showNotificationTab() {
        while (!notifications.isEmpty()) {
            notifications.top().postToNotificationPanel(
                notifications.max() == notifications.top());
            notifications.pop();
        }
    }

    @Override
    public void addNotification(Notification notification) {
        notifications.push(notification);
    }
}
```

שגיאות נפוצות:

- לא מסופק ה Comparator הנחוץ
- הודעות לא נשלחות בסדר הנכון (סדר קבלה)
- **Highlighting** שגוי

שאלה 3 (25 נקודות)

ברשת בתי הקפה "שוקו-שוקו" החליטו למחשב את מערכת הזמנת הקפה. לצורך כך פיתחה להם אומה את המנשק הבא:

```
public interface Espresso {
    enum Drink { LATTE, MOCHA, AMERICANO, CAPPUCCINO, ESPRESSO }
    enum Milk { ONE_PERCENT, THREE_PERCENT, CREAM, SOY }

    Drink getDrink();
    Milk getMilk();
}
```

א. (5 נקודות) ממשו את המנשק Espresso

```
public class SimpleEspresso implements Espresso {
    private Drink drink;
    private Milk milk;

    public SimpleEspresso(Drink drink, Milk milk) {
        this.drink = drink;
        this.milk = milk;
    }

    @Override
    public Drink getDrink() {
        return drink;
    }

    @Override
    public Milk getMilk() {
        return milk;
    }
}
```

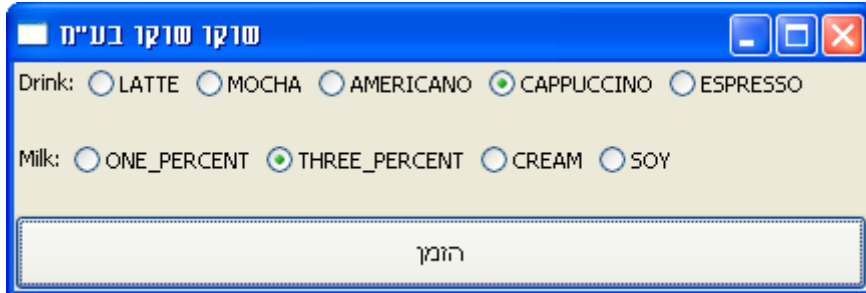
שגיאות נפוצות:

- משתנים לא פרטיים

ב. (10 נקודת) ביל מימש ממשק משתמש גרפי (GUI) עבור מערכת ההזמנות. המלצר/ית מזין/ה למערכת את סוג הקפה ואת סוג החלב ולוחץ/ת על "הזמן" (ראו איור 1). בעת לחיצה על "הזמן" יופעל השרות הסטטי

```
public static void order(Espresso e)
```

של המחלקה BarUnit.



איור 1 : מסך הזמנת קפה

ביל הדפיס את קוד המחלקה (בעמוד הבא) שמימש כדי להראות לאומה, אך בדרך עזר במטבחון שם הניח את כוס הקפה על התדפיס. השלימו את קוד המחלקה (במקומות המסומנים בכתמי קפה). בסוף המחלקה הושאר מקום להגדרת מבני עזר או שרותים חדשים במידת הצורך.

```

public class DrinksPanel {
    private Button milks[];
    private Button drinks[];
    private Button submit;

    private Shell createShell(Display display) {
        Shell shell = new Shell(display);
        FillLayout fillLayout = new FillLayout();
        fillLayout.type = SWT.VERTICAL;
        shell.setLayout(fillLayout);
        shell.setText("מ'בט שוקו שוקו");

        drinks = new Button[Espresso.Drink.values().length];

        Composite drinksPanel = new Composite(shell, SWT.NULL);
        drinksPanel.setLayout(new RowLayout());

        Label drinkLabel = new Label(drinksPanel, SWT.NULL);
        drinkLabel.setText("Drink: ");

        for (int i = 0; i < Espresso.Drink.values().length; i++) {
            drinks[i] = new Button(drinksPanel, SWT.RADIO);
            drinks[i].setText(Espresso.Drink.values()[i].name());
        }

        milks = new Button[Espresso.Milk.values().length];

        Composite milksPanel = new Composite(shell, SWT.NULL);
        milksPanel.setLayout(new RowLayout());

        Label milkLabel = new Label(milksPanel, SWT.NULL);
        milkLabel.setText("Milk: ");

        for (int i = 0; i < Espresso.Milk.values().length; i++) {
            milks[i] = new Button(milksPanel, SWT.RADIO);
            milks[i].setText(Espresso.Milk.values()[i].name());
        }

        submit = new Button(shell, SWT.PUSH);
        submit.setText("הזמן");

        submit.addListener(new SelectionAdapter() {
            @Override
            public void widgetSelected(SelectionEvent e) {
                BarUnit.order(new SimpleEspresso(getDrink(),
                                                    getMilk()));
            }
        });
    }

    return shell;
}

```

שגיאות נפוצות:

- התעלמות מהעובדה ש-order היא מתודה סטטית
- שימוש ב SelectionEvent כאילו היה חטיפוס Espresso

```
private Espresso.Drink getDrink() {  
    for (Button drink : drinks) {  
        if (drink.getSelection()) {
```

```
            return Espresso.Drink.valueOf(drink.getText());
```

שגיאות נפוצות:

- מחזיר את drink
- ניסיון לייצר Espresso מחברות בעזרת new

```
        }  
    }  
    return null;  
}
```

```
private Espresso.Milk getMilk() {
```

```
    for (Button milk : milks) {  
        if (milk.getSelection()) {  
            return Espresso.Milk.valueOf(milk.getText());
```

```
        }  
    }  
    return null;  
}
```

ג. (10 נקודות) להלן קוד השרות calculatePrice של המחלקה של BarUnit:

```
private double calculatePrice(Espresso e) {
    double price = 0;

    switch (e.getDrink()) {
        case LATTE:
            price += 10;
            break;
        case MOCHA:
            price += 12;
            break;
        case AMERICANO:
            price += 9;
            break;
        case CAPPUCCINO:
            price += 11.5;
            break;
        case ESPRESSO:
            price += 8;
            break;
    }

    switch (e.getMilk()) {
        case ONE_PERCENT:
        case THREE_PERCENT:
            break;
        case CREAM:
            price += 2;
            break;
        case SOY:
            price += 5;
            break;
    }

    return price;
}
```

אומה זכרה שבקורס "תוכנה 1" היא למדה ש switch הוא סממן לבאג בעיצוב המערכת והחליטה להמיר אותו בפולימורפיזם אמיתי על ידי שכתוב המנשק Espresso והמחלקה הממשת אותו, כך שקוד השרות calculatePrice יהיה:

```
private double calculatePrice(Espresso e) {
    return e.getPrice();
}
```

להלן קוד השרות getPrice במחלקה החדשה (שאר השרותים הושמטו):

```
public class SimpleEspresso implements Espresso {
    ...

    @Override
    public double getPrice() {
        return getDrink().getPrice() + getMilk().getPrice();
    }
}
```

עיצור לאומה לשכתב את המנשק Espresso ואת הטיפוסים Milk ו-Drink כך שיתאימו לקוד שלעיל. ניתן להגדיר מבני עזר או שרותים חדשים לצורך המימוש.

```
interface Priceable {
    double getPrice();
}

public interface Espresso extends Priceable {
    enum Drink implements Priceable {
        LATTE(10), MOCHA(12), AMERICANO(9), CAPPUCCINO(11.5),
        ESPRESSO(8);

        private double price;

        private Drink(double price) {
            this.price = price;
        }

        @Override
        public double getPrice() {
            return price;
        }
    }

    enum Milk implements Priceable {
        ONE_PERCENT(0), THREE_PERCENT(0), CREAM(2), SOY(5);

        private double price;

        private Milk(double price) {
            this.price = price;
        }

        @Override
        public double getPrice() {
            return price;
        }
    }

    Drink getDrink();
    Milk getMilk();
}
```


שאלה 4 (15 נקודות)

הסעיפים בשאלה זו מתייחסים לשלוש המחלקות הבאות:

```
public class A {
    int i = 5;
    public A() {
        foo();
    }
    private void foo() {
        System.out.println(i);
    }
}

class B extends A {
    int i = 6;
}

class C extends B {
    int i = 7;
    public void foo() {
        System.out.print(super.i);
    }
    public static void main(String[] args) {
        *****
    }
}
```

בכל אחד מהסעיפים הבאים מוחלפת שורת הכוכביות בקטע קוד. הינכם מתבקשים לציין מהו הפלט של התכנית עבור כל מקרה. במידה ולדעתכם אין פלט לתכנית מכיוון שאינה עובר קומפילציה או מכיוון שקיימת בעיה בזמן הריצה (זריקת חריג) הסבירו מה הבעיה, פתרון ללא הסבר לא יזכה בנקודות.

סעיף א' (3 נקודות)

```
A b = new B();
```

תשובה:

סעיף ב' (3 נקודות)

```
C c = new C();
System.out.print(c.i);
```

תשובה:

5
7

סעיף ג' (3 נקודות)

```
C c = new C();
System.out.print(((B)c).i);
```

תשובה:

5
6

סעיף ד' (3 נקודות)

```
C c = new C();
c.foo();
```

תשובה:

5
6

סעיף ה' (3 נקודות)

שימו לב, קטע הקוד בסעיף זה זהה לזה שבסעיף ד' ובנוסף החלפנו את מימוש המתודה foo שבמחלקה C.

```
public void foo() {
    System.out.print(super.super.i);
}
```

תשובה:

שגיאת קומפילציה: super.super