

פתרון לבחינה בתוכנה 1

סמסטר א'+ב', מועד ב', תשע"ב

29/08/2012

אוהד ברזילי, דן הלפרין, ליאור וולף

ניר אטיאס, יעל אמסטרדמר, אלכסיי זגלסקי

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 כחווה (תנאי קדם, תנאי בתר) בתחביר המקובל, שייכתב בתחילת השרות.

לשימוש הבודקים בלבד:

| שאלה | א | ב | ג | ד | ה | ו | 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 מכאנית ובין אלקטרונית או בכל דרך אחרת כל חלק שהוא מטופס הבחינה.

שאלה 1 (15 נקודות)

ברשימה מעגלית של מספרים שלמים המספרים מסודרים ברצף (כמו ברשימה ליניארית) אלא שאין מספר ראשון או אחרון. למען הנוחות נתאר את הרשימה המעגלית כאילו הייתה ליניארית, למשל (1,2,4), אבל את אותה רשימה מעגלית מייצגות גם (2,4,1) וכן (4,2,1).

נתונה רשימה מעגלית באורך n (הרשימה נתונה כליניארית למען הנוחות). ברשימה איבר אחד לפחות. מצאו על פניה רצף שסכום איבריו מקסימאלי ואורכו (כלומר מספר האיברים בו) לכל היותר n ולכל הפחות 1. כתבו את המתודה `maxSubSum` המחשבת מהי נקודת ההתחלה של הרצף ברשימה הנתונה (האינדקס של האיבר הראשון ברשימת הקלט הוא 0), כמה איברים ברצף, ומה סכומם. אם לכמה רצפים אותו סכום (שהוא מקסימאלי), המתודה מחזירה רצף בעל אורך מקסימאלי מתוכם. אם קיימים מספר רצפים (מתוך בעלי הסכום המקסימאלי) שאורכם מקסימאלי, המתודה מחזירה רצף כלשהו מתוכם. החליטו בעצמכם והסבירו כיצד מועברים הקלט למתודה והפלט שלה.

דוגמאות קלט, פלט:

(2,4,6,8,10) -> (0,5,30)

(2,-1,3,4) -> (2,3,9)

(-1,1,-1,1,-1) -> (1,3,1)

(0,0,0,0) -> (2,4,0)

```
/**
 * @pre list != null
 * @pre !list.isEmpty()
 */
public static SubSum maxSubSum(List<Integer> list) {
    // find largest subsum, assuming the list is not empty
    SubSum maxSubSum = new SubSum(0, 1, list.get(0));

    //for all starting points
    for (int startIdx = 0; startIdx < list.size(); startIdx++) {
        //for all lengths up to n
        for (int subLength = 1; subLength <= list.size(); subLength++) {
            //compute the sum
            int currentSum = 0;
            for (int stepsFromStart = 0; stepsFromStart < subLength;
                 stepsFromStart++) {
                // the modulo is used for going over the list is a cyclic
                manner
                currentSum += list.get((startIdx + stepsFromStart) %
                                       list.size());
            }
            // update the result
            maxSubSum.updateIfBetter(startIdx, subLength, currentSum);
        }
    }
    return maxSubSum;
}
```

```
// record describing one run over the cyclic list
public class SubSum {
    private int start; // refers to the input linear list, starting from
                       zero
    private int length;
    private int sum;

    public SubSum(int start, int length, int sum) {
        this.start = start;
        this.length = length;
        this.sum = sum;
    }

    // new sub sum takes over if the sum of its elements is larger or
    // the sum is unchanged but made out of more elements
    public void updateIfBetter(int start, int length, int sum) {
        if ((sum > this.sum)
            || (sum == this.sum && length > this.length)) {
            this.start = start;
            this.length = length;
            this.sum = sum;
        }
    }

    public int getStart() {
        return start;
    }

    public int getLength() {
        return length;
    }

    public int getSum() {
        return sum;
    }
}
```

שאלה 2 (40 נקודות)

ברשת חברתית כלשהי מעוניינים לממש מערכת תוכנה לניהול אירועים (Event) של משתמשים (User). לאורך סעיפי השאלה נדון בכמה היבטים של מימוש מערכת זו.

המנשק `DateRange` מבטא את רצף הזמן בו מתקיים האירוע (אינטרוול) עם זמן התחלה (`getStartDate`) וזמן סיום (`getEndDate`). השרות `isOverlap` בודק האם יש חפיפה בין הזמנים של הרצף המבטא על ידי העצם הנוכחי (`this`) ובין זה המבטא על ידי הארגומנט `other`.

```
import java.util.Date;

public interface DateRange {
    Date getStartDate();
    Date getEndDate();
    boolean isOverlap(DateRange other);
}
```

(10 נקודות) השלימו את קוד המחלקה המקובעת `SimpleDateRange` (כלומר, לא ניתן לשנות את העצם לאחר יצירתו) המממשת את המנשק `DateRange`. לנוחותכם צרפנו חלק משותף המחלקה `Date` בנספח שבסוף הבחינה:

```
import java.util.Date;

public class SimpleDateRange implements DateRange {
    private final Date startDate;
    private final Date endDate;

    public SimpleDateRange(Date startDate, Date endDate) {
        super();
        this.startDate = startDate;
        this.endDate = endDate;
    }

    @Override
    public Date getStartDate() {
        return startDate;
    }

    @Override
    public Date getEndDate() {
        return endDate;
    }

    @Override
    public boolean isOverlap(DateRange other) {
        SimpleDateRange temp = new SimpleDateRange(
            other.getStartDate(), other.getEndDate());
        return isWithinRange(other.getStartDate())
            || temp.isWithinRange(startDate);
    }

    private boolean isWithinRange(Date testDate) {
        return testDate.after(startDate) && testDate.before(endDate);
    }
}
```

לצורך מימוש מערכת האירועים נעזר בנימין בשני מנשקי עזר: User ו- Event המייצגים משתמש ואירוע (בהתאמה):

```
public interface User {
    public String getId();
    public String getName();
}
```

```
public interface Event {
    public String getId();
    public DateRange getDateRange();
}
```

השרות `getId` (בשני המנשקים) מחזיר מזהה ייחודי, אשר משמש לשמירת פרטי המשתמש או האירוע במסד נתונים. השרות `getName` מחזיר את שם המשתמש, והשרות `getDateRange` מחזיר את הזמן שבו מתקיים האירוע.

המנשק `EventManager` (למטה) משמש לצורך ניהול הקשרים בין אירועים ומשתמשים. השרות `getUsers` מחזיר את קבוצת כל המשתמשים אשר עשו `attending` לאירוע `e` שהועבר כארגומנט. באופן דומה, השרות `getEvents` מחזיר את קבוצת כל האירועים שלהם עשה המשתמש `u`, שהועבר כארגומנט, `attending`. השרות `attending` יוצר קשר בין המשתמש `u` והאירוע `e`, כלומר הוא מוסיף את המשתמש `u` לקבוצת המשתמשים של האירוע `e` (מוסיף באופן לוגי – ניתן לממש זאת בכל דרך שתבחרו), ומוסיף (שוב, באופן לוגי) את האירוע `e` לרשימת האירועים של המשתמש `u`. הטיפוסים `User` ו- `Event` אינם "מכירים" זה את זה, והקשר בינם מנוהל על ידי `EventManager`.

שימו לב כי משתמש יכול לנכוח בכמה אירועים, וכי באירוע נוכחים יותר ממשתמש אחד (בדרך כלל...). כמו כן, בשאלה זו אין צורך לממש את המנשקים `User` ו- `Event`, וניתן להניח כי הארגומנטים המועברים לפונקציות הן של מחלקות אשר מומשו כהלכה. בפרט ניתן להניח כי השירותים `hashCode` ו- `equals` ממומשים בצורה תקינה.

```
import java.util.Set;

public interface EventManager {
    public void attending(User u, Event e);
    public Set<User> getUsers(Event e);
    public Set<Event> getEvents(User u);
}
```

א. (10 נקודות) עזרו לבנימין לממש את המחלקה SimpleEventManager לפי ההגדרות לעיל:

```
public class SimpleEventManager implements EventManager {
    private Map<User, Set<Event>> events = new HashMap<User, Set<Event>>();

    @Override
    public void attending(User u, Event e) {
        Set<Event> eventSet = events.get(u);
        if (eventSet == null)
            eventSet = new HashSet<Event>();
        eventSet.add(e);
        events.put(u, eventSet);
    }

    @Override
    public Set<User> getUsers(Event e) {
        Set<User> result = new HashSet<User>();
        for (User user : events.keySet()) {
            if (events.get(user).contains(e))
                result.add(user);
        }
        return result;
    }

    @Override
    public Set<Event> getEvents(User u) {
        Set<Event> result = events.get(u);
        return result != null ? result : new HashSet<Event>();
    }
}
```

לקראת שחרור גרסה 2 של התוכנה, הבחינו המתכנתים בתופעה מוזרה – משתמשים רבים מדווחים על נוכחות באירועים אבל לא מגיעים בפועל. אחת הסיבות לכך היא חפיפה בין זמנים של אירועים שונים שלהם עשה/עשתה המשתמש/ת attending.

לאורית יש רעיון. היא מציעה לממש מחלקה חדשה CleverEventManager היורשת מהמחלקה SimpleEventManager, ולשנות את החוזה של השרות attending באופן הבא: המערכת תרשום את המשתמש לאירוע רק אם אין למשתמש אירוע אחר, מתנגש, באותן שעות. אם יש כזה, השרות לא יעשה דבר.

ב. (5 נקודות) ממשו את שרות העזר isLegal של המחלקה CleverEventManager הבודק התנגשות בין האירוע החדש e ובין האירועים הקיימים של המשתמש u:

```
public class CleverEventManager extends SimpleEventManager {  
  
    public boolean isLegal(User u, Event e) {  
        for (Event event : getEvents(u)) {  
            if (event.getDateRange().isOverlap(e.getDateRange()))  
                return false;  
        }  
        return true;  
    }  
  
    ...  
}
```

ג. (5 נקודות) נסחו בצורה פורמאלית ככל הניתן את החוזה של השרות attending של המחלקה CleverEventManager כפי שהציעה אורית (אין צורך לממש את השרות attending):

```
public class CleverEventManager extends SimpleEventManager {  
    /**  
     * @pre u != null  
     * @pre e != null  
     * @post (!exists Event e2 s.t. (!e.equals(e2) &&  
     *     $prev(getEvents(u).contains(e2)) &&  
     *     e.getDateRange().isOverlap(e2.getDateRange()))) $implies  
     *     getEvents(u).contains(e)  
     * @post (exists Event e2 s.t. (!e.equals(e2) &&  
     *     $prev(getEvents(u).contains(e2)) &&  
     *     e.getDateRange().isOverlap(e2.getDateRange()))) $implies  
     *     !getEvents(u).contains(e)  
     */  
    @Override  
    public void attending(User u, Event e) {...}  
    ...  
}
```

ד. (5 נקודות) האם החוזה החדש עומד בכללי עיצוב על פי חוזה? נמקו את תשובתכם.

לא, מכיוון שמחלקה יורשת לא יכולה לשנות את תנאי האחר באופן שעבור קלטים מסוימים (אירועים עברם יש התנגשות) לא יתקיים תנאי האחר של מחלקת האב (האירוע יקושר למשתמש ע"י ה-EventManager).

גלעד יש רעיון חלופי לגבי מימוש השרות attending במחלקה CleverEventManager. הוא מציע שאם יש למשתמש אירוע אחר, מתנגש, באותן שעות, השרות attending יזרוק את החרג :EventCollisionException

```
public class EventCollisionException extends Exception {  
    public EventCollisionException (String message) {  
        super(message);  
    }  
}
```

ה. (5 נקודות) האם ההצעה של גלעד תקינה? נמקו את תשובתכם.

לא. הדריסה של attending ב-CleverEvent Manager לא יכולה לזרוק חריג מסוג זה משום שבמתודה הנדרסת ב-SimpleEventManager לא הצהרנו על זריקת חריג. לכל היותר ניתן לזרוק חריג היורש מ- RuntimeException.

שאלה 3 (30 נקודות)

א. (16 נקודות) המנשק `Iterator<T>` מגדיר הילוך על אוסף איברים. המתכנת בנימין זיהה שבמקרים רבים בקוד הוא מעוניין לטפל בתת-קבוצה של איברי האוסף ולכן הפעולה הראשונה בלולאה היא פעולת התניה (`if`).

בנימין מעוניין לממש רעיון זה במחלקה ייעודית ומופשטת, `SubsetIterator<T>`, אשר מקבלת איטרטור סטנדרטי כקלט לבנאי ועוברת רק על תת קבוצה של איברים שמחזיר האיטרטור. כדי לזהות אילו מהאיברים שייכים לקבוצה, תוגדר המתודה האבסטרקטית `isInSubset`.

השלימו את מימוש המחלקה:

```
abstract class SubsetIterator<T> implements Iterator<T> {
```

```
    Iterator<T> iterator;
```

```
    T currentItem;
```

```
    public SubsetIterator(Iterator<T> iterator) {
```

```
        this.iterator = iterator;
```

```
        this.currentItem = getFirstItem(iterator);
```

```
    }
```

```
    protected T findNextItem() {
```

```
        while(iterator.hasNext()) {
```

```
            T item = iterator.next();
```

```
            if(isInSubset(item))
```

```
                return item;
```

```
        }
```

```
        return null;
```

```
    }
```

```
    protected T getFirstItem(Iterator<T> iterator) {
```

```
        return findNextItem();
```

```
    }
```

```
    protected abstract boolean isInSubset(T item);
```

```
    protected T getCurrentItem() {
```

```
        return currentItem;
```

```
    }
```

```
    public boolean hasNext() {
```

```
        return currentItem != null;
```

```
    }
```

```

public T next() {
    T retval = currentItem;
    currentItem = findNextItem();

    return retval;
}

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

```

ב. (7 נקודות) השלימו את מימוש המחלקה MonotoneIncIterator אשר מחזירה תת-סדרה של איברים בסדר עולה ממש.

```

public class MonotoneIncIterator<T extends Comparable<T>> extends
SubsetIterator<T> {
    MonotoneIncIterator(Iterator<T> iterator) {
        super(iterator);
    }

    @Override
    protected boolean isInSubset(T item) {
        return getCurrentItem().compareTo(item) < 0;
    }

    @Override
    protected T getFirstItem(Iterator<T> iterator) {
        if(iterator.hasNext())
            return iterator.next();

        return null;
    }
}
}

```

ג. (7 נקודות) ממשו את מחלקה MonotoneIterator המחזירה תת-סדרה של איברים בסדר עולה או יורד. הבנאי מקבל פרמטר נוסף מסוג enum שלו שני ערכים INCREASING ו- DECREASING הקובעים את מונטוניות הסדרה מוזרת. יש להגדיר גם את טיפוס ה-enum וגם את המחלקה.

יש להימנע משכפול קוד.

```
public enum Direction {
    INCREASING,
    DECREASING
}

public class MonotoneIterator<T extends Comparable<T>> extends
SubsetIterator<T> {
    private Direction direction;

    public MonotoneIterator(Iterator<T> iterator, Direction direction) {
        super(iterator);
        this.direction = direction;
    }

    @Override
    protected T getFirstItem(Iterator<T> iterator) {
        if(iterator.hasNext())
            return iterator.next();

        return null;
    }

    @Override
    protected boolean isInSubset(T item) {
        return direction == Direction.INCREASING
            ? getCurrentItem().compareTo(item) < 0
            : getCurrentItem().compareTo(item) > 0;
    }
}
```

שאלה 4 (15 נקודות)

הסעיפים בשאלה זו מתייחסים לארבע המחלקות הבאות:

```
public class A {
    public A() {
        System.out.println("A1");
    }

    public A(int a) {
        System.out.println(a);
    }

    public int foo(int a) {
        return 0;
    }
}

public class AA extends A {
    public AA(int aa) {
        System.out.println(aa);
    }
}

public class B extends A {
    public B(int b) {
        super(b);
    }

    public int foo(Integer b) {
        b = new Integer(b + 5);
        return b;
    }

    public int foo(int b) {
        return b;
    }
}

public class C {

    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 עוברת קומפילציה או מכיוון שקיימת שגיאה בזמן ריצה (זריקת חריג), ציינו מה השגיאה והסבירו מה גרם לה.

סעיף א' (3 נק')

```
A a = new A();  
System.out.println(a.foo(5));
```

```
solution: A1  
          0
```

סעיף ב' (3 נק')

```
A a = new AA();  
System.out.println(a.foo(5));
```

```
solution: Compilation Error (the constructor AA() is undefined)
```

סעיף ג' (3 נק')

```
A a = new AA(5);  
System.out.println(a.foo(7));
```

```
solution: A1  
          5  
          0
```

סעיף ד' (3 נק')

```
B b = new B(5);  
A a = (A) b;  
System.out.println(a.foo(2));
```

```
solution: 5  
          2
```

סעיף ה' (3 נק')

```
Integer i = new Integer(3);  
A a = new B(i);  
System.out.println(a.foo(i));
```

```
solution: 3  
          3
```