2019 / 06 / 30
כ"ד באב תשע"ח

האוניברסיטה העברית בירושלים

ביה"ס להנדסה ומדעי המחשב ע"ש רחל וסלים בנין

מבוא לתכנות מונחה עצמים (67125)

(מועד א' צהריים)

מרצה: יואב קן-תור

מתרגלים: זיו בן-אהרון, נגה רוטמן, ג'ונה הריס

תעודת זהות של הסטודנט/ית: _____

הוראות:

- משך המבחן הינו **3 שעות**.
- המבחן הינו עם חומר סגור. אין להשתמש בכל חומר עזר או אמצעי חישוב שהוא, לרבות מחשבון.
- למבחן שני חלקים:
  - חלק א' – כולל 6 שאלות, מהן עליכם לענות על **5**. כל שאלה שווה 10 נקודות.
  - חלק ב' – כולל 2 שאלות קוד גדולות, ועליכם לענות על שתיהן. כל שאלה שווה 25 נקודות.

שימו לב לנקודות הבאות:

- "Less is more" – רשמו תשובה קצרה ומדויקת, שלא תשאיר מקום לספק שאכן הבנת את החומר.
- ייתכן שהקוד המוגש יעבוד, אבל למבחן בתכנות מונחה עצמים זה לא מספיק. הקוד צריך להיות קריא ומתוכנן היטב. תעד את הקוד (אפשר בעברית) רק אם יש חשש שמישהו לא יבין אותו.
- זכור כי בעיצוב אין תשובה אחת נכונה, אבל בהחלט יש תשובות שהן לא נכונות. נקודות מלאות יתקבלו רק באם השתמשת באחת מהדרכים המתאימות ביותר.
- באם תענה על יותר שאלות מן הנדרש, רק הראשונות ייבדקו.
- אנו ממליצים לקרוא את כל המבחן מתחילתו עד סופו לפני תחילת הפתרון.
- המבחן כתוב בלשון זכר, אך מיועד לכלל סטודנטי הקורס במידה שווה.

בהצלחה!

חלק א'

- כולל 6 שאלות, מהן עליכם לענות על **5**. כל שאלה שווה 10 נקודות. הקפידו לנמק כאשר התבקשתם.
- חלק מן השאלות כוללות מספר סעיפים, אשר מופרדים לתיבות שונות במודל ומסומנים בהתאם (למשל, אם שאלה 1 כוללת שני סעיפים הם יסומנו כ"שאלה 1.1" ו"שאלה 1.2"). הקפידו לענות על כל חלקי השאלות.
- בשאלות הכוללות כתיבת קוד, הקפידו לכתוב קוד **יעיל ומובן**. רק מימושים כאלו יקבלו ניקוד מלא. ניתן להניח שהקלט הוא לא null.
- בשאלות בהם התבקשתם למלא את ה-modifiers, ניתן למלא מילה אחת או יותר ממילה אחת. **תיבה ריקה תחשב כטעות - הקפידו למלא modifier בכל מקום שניתן!**

דוגמא ל-modifier בעל מספר מילים: `private static int x`

לעומת modifier בעל מילה אחת: `private int x`

שאלה 1

בחרו את האפשרות הנכונה עבור כל היגד בפסקה הבאה:

א. ממשק (*interface*) **יכול\לא יכול** להכריז (*declare*) על מתודות protected.

ב. מחלקה אבסטרקטית (*abstract class*) **יכולה\לא יכולה** להכריז על מתודות protected.

ג. ממשק **יכול\לא יכול** להגדיר שדות שאינם final .

ד. מחלקה אבסטרקטית **יכול\לא יכול** להגדיר שדות שאינם final.

ה. מתודה סטטית (*static method*) **יכולה\לא יכולה** להיות אבסטרקטית (*abstract*).

שאלה 2

הסבירו בקצרה את ההבדל בין **Comparator** לבין **Comparable**, כיצד ניתן להשתמש בכל אחד על מנת למיין מערך (*array*) ותארו יתרון וחסרון של כל שיטה.

Common Mistakes:

- Not answering all parts of the question
- Saying that Comparator is an abstract class
- Not to mention that there could be many Comparators that implement different orders between objects, but only one order can be defined using Comparable - which is the natural order between those objects
- Not specify how objects of classes that implement Comparable are sorted (simply calling list.sort() or Collections.sort(list)), or specify that they cannot be sorted that way
- Not specify how sorting is done with Comparators

Common mistakes:

- Not answering all parts of the question
- Saying that Comparator is an abstract class
- Not to mention that there could be many Comparators that implement different orders between objects, but only one order can be defined using Comparable - which is the natural order between those objects
- Not specify how objects of classes that implement Comparable are sorted (simply calling list.sort() or Collections.sort(list)), or specify that they cannot be sorted that way
- Not specify how sorting is done with Comparators

שאלה 3

א) מהי תוצאת הקוד הבא:

```
public class Complex {
    private double r, i;
    Complex(double r, double i) {
        this.r = r;
        this.i = i;
    }
    Complex add(Complex other) {
        return new Complex(this.r + other.r, this.i + other.i);
    }
    public String toString() {
        return this.r + " + " + this.i + "i";
    }
    public static void main(String[] args) {
        Complex c1 = new Complex();
        Complex c2 = c1;
        c1 = new Complex(2, 5);
        Complex c3 = c1.add(c2);
        System.out.println(c3);
    }
}
```

1. $5.0i + 2.0$
2. $10.0i + 4.0$
3. שגיאת קומפילציה
4. שגיאת זמן ריצה

ב) נמקו בקצרה את תשובתכם לסעיף הקודם.

Common mistakes:

- Thinking that there was a default constructor (because another constructor was defined, the default constructor was not generated automatically).

- Thinking that if we assigned c2 = c1, then assigning c1 to a new object will affect c2.
- Thinking that the default value of a double is null.

שאלה 4

א) השלימו את ה modifiers בקוד הבא על מנת שיודפס:

```
42
58
```

```
class OuterClass {
    1._____ int y = 2;
    2._____ firstNested nested = new firstNested();
    static class firstNested {
        3._____  int a = 4;
        secondNested inner = new secondNested();

        4._____  class secondNested {
            void add(int num) {
                a += num;
                y += a + num;
            }
            void print_all() {
                System.out.print(a);
                System.out.print(y);
                System.out.println();
            }
        }
    }
    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        outer.nested.inner.print_all();
        outer.nested.inner.add(1);
        outer.nested.inner.print_all();
    }
}
```

ב) נמקו בקצרה את תשובתכם לסעיף הקודם.

Common mistakes:

- Not defining the int member of OuterClass to be static.

שאלה 5

ממשו מתודה שמסמלצת טורניר בין זוגות של מתמודדים כאשר כל מנצח עובר שלב עד שנשאר אחד.

קלט: מערך (*array*) של אובייקטים מסוג `Challenger`.

פלט: המנצח של הטורניר (אובייקט מסוג `Challenger`).

האובייקטים המדמים את המתחרים (`Challenger`) **מומשו** (לא צריך לממש) והדבר היחיד שידוע עליהם זה שיש להם את המתודה:

```
public boolean winsAgainst(Challenger other);
```

המחזירה "true" רק אם האובייקט מנצח את האחר.

על חתימת המתודה שתממשו להיות:

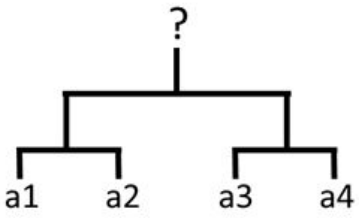
```
public static Challenger tournament(Challenger[] challengers);
```

דוגמא:

עבור שני מתמודדים `a` ו-`b` נסמן `a > b` אם `a` מנצח את `b`. אזי, עבור הקלט

```
Challenger[] challengers = {a1, a2, a3, a4};
```

כאשר `a4 > a1 > a2 > a3`, הטורניר שצריך להתקיים הוא (על פי סדר המשתתפים במערך):



שלב 1: `a1` נגד `a2` --> המנצח הוא `a1` וגם `a3` נגד `a4` --> המנצח הוא `a4`

שלב 2: `a1` נגד `a4` --> המנצח הוא `a4`

תוצאה: `a4` ניצח בטורניר ולכן הוא יוחזר

*(**שימו לב** - יכול להיות מצב בו הסדר שבו המתחרים מתמודדים אחד מול השני משנה את התוצאה. כלומר, יכול להיות מצב בו `a>b>c` וגם `c>a`)*

הנחות מקלות:

- הניחו כי מספר המתחרים הוא חזקה של 2
- הניחו כי לא יכול להיות תיקו בין שני מתחרים

Common mistakes:

Using redundant data structures or recursion the best solution is a single function with a single data structure.

שאלה 6

עבור מערך (*array*) מספרים, נרצה למצוא את התת-מערך הארוך ביותר של מספרים עוקבים עם רווח זהה וחיובי ביניהם. כתבו מתודה המקבלת מערך של מספרים ומדפיסה את אורך התת-מערך הגדול ביותר ואת הרווח בין המספרים באותו תת-מערך בפורמט. לדוגמא, אם קיים תת-מערך כנ"ל באורך **7** עם רווחים של **2** בין כל המספרים בתת מערך, יודפס:

```
7,2
```

אם לא קיימת תת-מערך כמתואר, על המתודה להדפיס "0,0". החתימה של המתודה שתממשו חייבת להיות:

```
public static void biggestAscend(int[] nums);
```

דוגמאות:

- קלט: [0,1,2] פלט: 3,1 (התת-מחרוזת בדוגמא היא בעצם כל המחרוזת ולכן היא באורך 3, והרווח בין הספרות הוא 1)
- קלט: [10,0,1,2,4,6,8] פלט: 4,2
- קלט: [10,9,8,7,0] פלט: 0,0

Common mistakes:

- Many students miscalculated the length of the subarray. Note that when there are n equal difference values in a row, the length of the subarray is n+1.
- When using a Map to store the lengths of the subarrays, it was important to check whether or not the difference value of the subarray was already in the map. Many students did not check this, and either ignored the previous value in the map, or increased it even it referred to a different subarray. This is a problem in cases like 1 2 3 5 6, where the same difference value appears more than once, but they're not all in a row.
- Hard to read code. We were less strict regarding style guidelines in the exam than we were in exercises. However, many style guidelines exist to make code easier to read. A number of students used uninformative or misleading variable names that made their code very hard to understand. Similarly, many students did not document non-trivial code that was very hard to follow with no explanation.
- Illegal Java syntax. In particular, a number of students treated the array like an ArrayList and many students created a Collection holding primitives.

חלק ב'

- קראו בעיון את שתי השאלות הבאות. קראו כל שאלה עד הסוף לפני תחילת הפתרון.
- לשאלות יכולות להיות מספר סעיפים ותתי-סעיפים.
- הקפידו לנמק את העיצוב שלכם עבור כל סעיף. תשובה ללא נימוק לא תקבל את מלוא הניקוד.
- כאשר אתם כותבים נימוק הקפידו לפרט אילו עקרונות ותבניות עיצוב (design patterns) היו בשימוש.
- הקפידו על עיצוב יעיל. העיצוב ייבחן על בסיס:
  1. שימוש בכלי המתאים ביותר מהכלים שנלמדו בקורס.
  2. שימוש במינימום ההכרחי של מחלקות וממשקים.
- בכל פעם שהוגדרה מתודה למימושכם, אם חתימתה חסרה, עליכם להשלים את ה-modifiers לפי שיקולכם.
- ניתן להשתמש בקוד Java של מחלקות שלמדנו עליהן (לדוגמא הפונקציה sort של Collections).

שאלה 1

בשאלה זו נבנה מערכת פשוטה למידול חנויות. חנות מורכבת, בין היתר, ממוצרים (**Product**) וממוכר (**Seller**). כפי שידוע, מחיר מוצר מסוים יכול להשתנות בין חנות לחנות ואינו שווה בהכרח לערך המוצר בפועל. כך גם במערכת שלנו, לכל מוצר יש ערך גולמי וכל מוכר קובע מחיר למוצר כלשהו לפי פונקציית תמחור (שיכולה להיות תלויה למשל בערכו הגולמי של המוצר). למעשה, פונקציית תמחור היא פונקציה ממוצרים (**Product**) למספרים ממשיים.

לצורך מימוש הסעיפים בשאלה, נתונים לכם שני ממשקים: **PricingPolicy** ו- **Valuable**:

1. **Valuable**: ממשק המייצג אובייקט בעל ערך כספי (כמובן שלא כל האובייקטים בעולם הם בעלי ערך כספי, כמו למשל רעות)
2. **PricingPolicy**: ממשק המייצג שיטת תמחור לאובייקט בעל ערך (**Valuable**)

```
@FunctionalInterface
public interface Valuable {
    /**
     * @return the raw value of the object
     */
    double rawValue();
}
@FunctionalInterface
public interface PricingPolicy {
    /**
     * @return the price of the valuable v according to the policy
     */
    double price(Valuable c);
}
```

לא ניתן לשנות את הממשקים הנתונים!

בסעיפי השאלה אתם נדרשים לממש מחלקות שונות. אם לא צוינו במפורש חתימות של מחלקות/מתודות עליכם לקבוע אותם באופן המתאים ביותר להתנהגות הרצויה כפי שמוגדר בשאלה.

א) את המוצרים השונים בחנות נייצג בעזרת המחלקה **Product**. כל מוצר מאופיין על ידי שני פרמטרים:

- שם (מחרוזת)
- ערך גולמי (מספר ממשי)

על המחלקה **Product** להכיל בנאי המקבל שני ערכים אלו ובונה מוצר בהתאם:

```
Product(String name, double value)
```

<u>שימו לב:</u> ה-API של המחלקה **Product** ניתן לכם באופן חלקי. עליכם לחשוב מהי החתימה המלאה של המחלקה, האם נדרשות עוד מתודות ב- API ולממש אותן.

את המוכר נייצג בעזרת המחלקה **Seller**. לכל מוכר יש שיטת תמחור שונה (**PricingPolicy**) שנקבעת עבורו ברגע יצירתו. המחלקה **Seller** מממשת בנאי אחד ומתודה נוספת אחת:

`Seller(PricingPolicy p)`	בנאי היוצר מוכר עם שיטת תמחור *p*
`double priceOfProduct(Product prod)`	פונקציה המחזירה את המחיר של המוצר הנתון, ע"פ שיטת התמחור של המוכר

ממשו את שתי המחלקות **Product** ו- **Seller**. נמקו והסבירו את מימושכם.

ב) קים באג אין חזר לצפון קוריאה והחליט להקים שוק קפיטליסטי. כהתחלה הקים בסטה בשוק המקומי, והחליט לנקוט בשיטת תמחור שתשבור את השלטון - תמחור כל מוצר בשקל אחד פחות ממחירו הזול ביותר בשוק (כאשר כל המחירים נבחרו על ידי השלטון).

קים באג אין מבין בעגבניות והתקוממויות, אבל לא מבין בג'אווה. ממשו עבורו פונקציה סטטית המקבלת מערך של שיטות תמחור ומחזירה שיטת תמחור חדשה, הנותנת לכל מוצר את מחירו הנמוך ביותר מבין כל השיטות, פחות שקל אחד. חתימת המתודה הינה:

```
static PricingPolicy competitivePolicy(PricingPolicy[] policies)
```

נמקו בקצרה את מימושכם.

<u>הערות</u>:

- הינכם רשאים לממש מחלקות/מתודות עזר נוספות אם יש צורך בכך
- ניתן להניח כי המערך הנתון כארגומנט מכיל לפחות איבר אחד

ג) בסעיף זה נניח כי כבר מומשה עבורנו מחלקה המייצגת חנות:

```
class Store {
   /* ... data members ... */
   /* default constructor */
   Store() { /* initialization code */ }

   /* return the market cap of the store */
   double getStoreMarketCap() { /* return value code */ }
   /* ... more methods ... */
}
```

נוסף על מחלקה זו, נתונים לנו עוד ממשק ומחלקה ממומשת:

- **Business** ממשק המייצג עסק
- **StockExchange** מחלקה המייצגת בורסה, שמכילה אוסף של עסקים (**Business**)

```
public interface Business {
   /* return the market cap of the business */
   double getBusinessMarketCap();
}

class StockExchange {
   /* ... data members ... */
   /* default constructor */
   StockExchange() { /* initialization code */ }
   /* Add the given business to this stock exchange */
   void addBusiness(Business business) { /* add business code */ }
   /* ... more methods ... */
}
```

במימוש הנוכחי הנתון לנו, ניתן להכניס רק עסקים לבורסה (על ידי המתודה *addBusiness*). אנו נדרשים לבצע שינויים כך שגם חנויות (**Store**) יוכלו להיכנס בבורסה.

1. הסבירו מדוע לא ניתן להוסיף חנות (**Store**) לבורסה על ידי *addBusiness*

2. הציעו פתרון לבעיה שתיארתם בסעיף **ג.1** וממשו אותו, **כאשר אין באפשרותכם לשנות את המחלקה Store**

3. כתבו קוד קצר אשר יוצר בורסה וחנות ומוסיף את החנות לבורסה (תוך שימוש בפתרון שמימשתם)

הבהרה: כאשר אנו חושבים על חנות בתור עסק, השווי של החנות (market cap) הוא בדיוק השווי שלה בתור עסק

Part B Q1:

- 1: Please note: complicating a design unnecessarily is not a requirement for a good grade, nor for being a good programmer. Simplicity is the ultimate sophistication.
    - Neither Seller not Product should be abstract, as the question did not indicate that a Product or Seller cannot be created as Objects.
    - Seller should not implement PricingPolicy, it should **have** a PricingPolicy ("has-a"), which should be used to implement priceOfProduct.
    - API violations: either not implementing rawValue, or implementing priceOfProduct by sending the PricingPolicy the raw value of a Product instead of the Product itself.
- 2: This is where the majority of the problems manifested, mostly around the question of "where can I get a product so that I can compare policies?". Some of the most common mistakes were:
    - Using the PricingPolicy price method without a product. Since this method does not exist, this won't work.
    - Creating your own product, then iterating the array of policies using it to determine the "cheapest" policy. As a policy can be non-linear, this won't produce the correct answer.
    - Iterating through the array of policies using a product which is never recieved (so the code cannot compile).
    - Implementing the competativePolicy static method within the Product class, presumably in order to use "this" (the instance Product) when comparing the policies. This answer does not work for several reasons, perhaps the most important of them being that as a static method, "this" has no meaning.
- 3:
    - You could not change either Store, StockExchange or Business.
    - Extending StockExchange does not comply with the requirements of the question, i.e. adding a Store to the StockExchange.

שאלה 2

בספינת החלל USS Discovery יש מחסור חמור באנשי צוות לאחר היתקלות קשה עם משחתת קלינגונית. על הקפטנית נגה יוניקורן-בורנהאם לאייש במהירות את מחלקת ההנדסה על מנת לאפשר לדיסקאברי להמשיך במסעה ברשת המיציליאלית כדי להימלט מהאיום הקלינגוני המיידי.

מעבורת של צוערים (Cadets) נשלחה מהתחנה לבי עמוק 7 על מנת לסייע בתגבור הכוחות.

כל צוער הוא מופע של המחלקה הבאה:

```java
package USSCandidateSelection;
public class Cadet {
   private final String name;
   private int age;
   private int height;
   private List<Cadet> classMates = new ArrayList<>();
   private List<String> skills = new ArrayList<>();
   public Cadet(String name, int height, int age) {
       this.name   = name;
       this.height = height;
       this.age = age;
   }

   public void addClassMates(List<Cadet> classMates) {
      this.classMates.addAll(classMates);
   }
   public void addSkills(List<String> skills) {
       this.skills.addAll(skills);
   }
   public String getName() { return this.name; }
   public int getHeight() { return this.height; }
   public int getAge(){ return this.age; }
   public List<Cadet> getClassMates() {
       return new ArrayList<>(classMates);
   }
   public List<String> getSkills() { return new ArrayList<>(skills); }

}
```

א) קפטן יוניקורן-בורנהאם הטילה עליכם למצוא את המועמדים האופטימליים למחלקת ההנדסה. הדרישות מאנשי הנדסה:

- על איש הנדסה להיות מוכשר בתחומים הבאים (כלומר, על כולם להמצא ברשימת ה-*skills* שלו):

1. Warp Drives // = “WD”
2. Energy Pattern Rematerialization // = “EPR”
3. Dilithium Refining // = “DR”
4. Antimatter Reactors // = “AR”

(לנוחיותכם, השתמשו בקוד המקוצר הכתוב ליד כל אחד מהמקצועות)

- בנוסף, מאחר ובליבת מנוע העיוות חלק מהמכשירים נמצאים בגובה רם, על הצוערים להיות בגובה של לפחות 165 ס"מ

עליכם להוסיף לחבילה **USSCandidateSelection** טיפוס פומבי כלשהו בשם **CadetSelection** ובו שיטה עם החתימה הבאה:

```
public List<Cadet> getCadets(List<Cadet> candidates)
```

השיטה מקבלת את רשימת הצוערים ומחזירה רשימה ובה רק הצוערים שמתאימים למחלקת ההנדסה.

ענו ונמקו בקצרה (משפט או שניים) האם העיצוב שלכם מקיים או לא מקיים כל אחד מהעקרונות הבאים:

1. Composability
2. Decomposability
3. Open-Closed

שימו לב כי הפתרון שלכם לא מוכרח לקיים את כל העקרונות הנ"ל על מנת לקבל את מלוא הנקודות.

(שימו לב – המשך השאלה נמצא בעמוד הבא!)

ב) לנוחיותכם, הצוערים שנבחרו מועברים לספינה בטלפורטציה באמצעות פונקציית הקסם *teleportCadets*. על מנת להראות לקפטנית כיצד להשתמש בחבילה שלכם, השלימו את מחלקת ה-**MainClass** הבאה כדי שהקפטנית תוכל בקלות למצוא את אנשי ההנדסה המוצלחים ביותר לספינתה:

```
import java.util.*;

import USSCandidateSelection.*;


public class MainClass {

public static void main(String[] args) {

        List<Cadet> cadets = teleportCadets();

        //… complete function

}

private static List<Cadet> getMyFutureEngineers(List<Cadet> candidates) { ... }
```

- יש להשלים את תוכן הפונקציה הפרטית *getMyFutureEngineers* כך שתחזיר את חמשת המועמדים המתאימים ביותר
- לשם כך יש למיין את המועמדים מהמבוגר לצעיר (שכן הצוערים המבוגרים יותר בעלי יותר ניסיון)
- יש להשלים את תוכן פונקציית ה-main כך שבסיומה יודפסו הצוערים הנבחרים

הערות:

- אתם לא יכולים לשנות את חתימת השיטות, אבל רשאים להרחיב את ה-API של הטיפוסים כרצונכם, עם שמירה על עקרונות האנקפסולציה ו-minimal API
- אתם רשאים להגדיר בחבילה מחלקות וממשקים, פומביים או לא-פומביים, לפי שיקול דעתכם

(שימו לב – המשך השאלה נמצא בעמוד הבא!)

ג) מחלקת ההנדסה אוכלסה וספינת החלל דיסקאברי הצליחה להמלט מאזור העימות! כל הכבוד.

הקפטנית הייתה כל כך מרוצה מהפתרון שלכם שהיא רוצה כעת שתעזרו לה לאכלס גם את מחלקת האבטחה, על מנת לתגבר את הכוחות למקרה של עימות חדש. הדרישות מאנשי אבטחה:

- על איש אבטחה להיות מוכשר בתחומים הבאים:

1. Phaser Weapons // = PW
2. Bat'leth Dueling // = BD
3. Vulcan Death Grip // = VDG

- בנוסף, על מנת לאפשר ללוחמים לעבור בחללים הצרים שבספינה בעודם רודפים אחרי פולשים, לא ניתן לאפשר לצוערים מעל גובה 190 ס"מ להצטרף לשורות מחלקת האבטחה.

עדכנו את תכניתכם המקורית (פונקציית ה-main וכן כל טיפוס אחר שמימשתם אם יש בכך צורך), כך שתאפשר לאתר גם מועמדים למחלקת האבטחה. לשם כך:

- הוסיפו למחלקה **MainClass** את הפונקציה הסטטית:

```
private static List<Cadet> getMyFutureSecurity(List<Cadet>
                    candidates) { ... }
```

- ממשו אותה כך שתחזיר את 5 המועמדים הטובים ביותר. על מנת להיות איש ביטחון אפקטיבי, על המועמדים להיות גם בעלי יכולות חברתיות טובות. לשם כך - עליכם למיין את המועמדים לפי מספר החברים שיש לו במחזור שלו באקדמיה. המתודה *getClassMates* של כל צוער מחזירה מספר זה
- עדכנו את פונקציית ה-main כך שתדפיס בסופה גם את חמשת הצוערים המתאימים ביותר למחלקת האבטחה

ענו ונמקו בקצרה (משפט או שניים) האם העיצוב שלכם מקיים או לא מקיים כל אחד מהעקרונות הבאים:

1. Composability
2. Decomposability
3. Open-Closed

שימו לב כי הפתרון שלכם לא מוכרח לקיים את כל העקרונות הנ"ל על מנת לקבל את מלוא הנקודות.

<u>Common mistakes</u>

- (Probably due to bad lecture presentation terminology) - Confusing [de]composability of a **system** with that of a **method/function** and incorrectly answering the design

description part of the question. Decomposability does not mean breaking a **method** into parts, but breaking the **A problem** into separate subproblems to be solved independently.

- General lack of understanding of the 3 basic principles of modularity. Specifically - using multiple methods instead of one big one is not “composability” and breaking down a solution into multiple methods is also not decomposability. Units must be autonomous - i.e. - classes or interfaces.
- Thinking that having several very specific methods satisfies Open-Closed principle. Not understanding that if *any* code change is needed to change or add functionality (in the classes performing the tasks) it breaks the “closed” part of the principle. Writing new classes or methods outside the main program that can use old parts *as-is* is actual idea.
- Thinking that implementing comparable allows for using a List’s sort method without a comparator.
- Implementing comparable in a parametrized way (using a global flag) which breaks the contract dictating “natural order”
- Some people implemented comparator methods writing expressions like cadet1.getHeight().compareTo(cadet2.getHeight()). This doesn’t work, as the API clearly states that the height method returns an “int” primitive. Primitives are not objects. This is not considered syntax error but a basic misunderstanding of the difference between primitives and objects.

אובייקטים נפוצים:

Class HashMap<K,V>	
V get(Object key)	Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key
Set<K> keySet()	Returns a Set view of the keys contained in this map
V put(K key, V value)	Associates the specified value with the specified key in this map. If the map previously contained a mapping for the key, the old value is replaced.
V remove(Object key)	Removes the mapping for the specified key from this map if present

Collection<V> values()	Returns a Collection of the values contained in this map

Class HashSet<E>	
boolean add(E e)	Adds the specified element to this set if it is not already present
boolean contains(Object o)	Returns true if this set contains the specified element
boolean remove(Object o)	Removes the specified element from this set if it is present
Iterator<E> iterator()	Returns an iterator over the elements in this set

Class LinkedList<E>	
boolean add(E e)	Appends the specified element to the end of this list
void clear()	Removes all of the elements from this list
E poll()	Retrieves and removes the head (first element) of this list
E get(int index)	Returns the element at the specified position in this list

<u>הרשימה נועדה לעזור ואינה מחייבת</u>. ייתכן שקיימות פונקציות או סוגי אובייקטים שלא נמצאים ברשימה אבל כן צריך להשתמש בהם על מנת לפתור את המבחן