

האוניברסיטה העברית בירושלים
ביה"ס להנדסה ומדעי המחשב ע"ש רחל וסלים בנין

מבוא לתכנות מונחה עצמים (67125)

(מועד א' בוקר)

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.

יש לענות על כל שאלות המבחן במחברת הבחינה.

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.
 - 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 (כתבו את מספר המשפט ואת המילה הנכונה במחברת):

- מחלקה פנימית (*inner class*) יכולה לא יכולה להגדיר (*declare*) מתודות סטטיות.
 - מחלקה סטטית (*static class*) יכולה לא יכולה להגדיר מתודות שאינן סטטיות.
 - מחלקה פנימית יכולה לא יכולה לגשת לשדות פרטיים (*private*) של המחלקה החיצונית לה.
 - מחלקה פנימית יכולה לא יכולה לגשת לשדות שהוגדרו *protected* במחלקה החיצונית לה.
- הבהרה: אם למחלקה A קיימת מחלקה פנימית B אז נקרא ל A "המחלקה החיצונית של B"**

שאלה 2

תארו בקצרה מהי משפחת התבניות **Factory** (הנקראת גם תבנית העיצוב **Factory**), לאיזה עיקרון של תכנות מונחה עצמים היא מתקשרת ותארו בקצרה שימוש אפשרי בה.

טעויות נפוצות:

לא לציין כי פקטורי משמש עבור יצירת אובייקטים בעלי עץ הורשה חופף או הממשים את אותו ממשק

```
public class Test
{
    public static void main(String[] args)
    {
        String obj1 = new String("oop");
        String obj2 = new String("oop");

        if(obj1.hashCode() == obj2.hashCode())
            System.out.println("hashCode of object1 is equal to object2");
        if(obj1 == obj2)
            System.out.println("memory address of object1 is the same as object2");
        if(obj1.equals(obj2))
            System.out.println("value of object1 is equal to object2");
    }
}
```

- a. The code won't print anything
- b. hashCode of object1 is equal to object2
memory address of object1 is the same as object2
value of object1 is equal to object2
- c. memory address of object1 is the same as object2
value of object1 is equal to object2
- d. hashCode of object1 is equal to object2
value of object1 is equal to object2

שאלה 4

1) השלימו את ה modifiers המינימלים (הכי מגבילים) ע"מ שהקוד הבא יתקמפל (מלאו את ה-modifier הנכון במחברת הבחינה על פי המספור המודגש באדום):

```
package pack1;
public class A
{
    1. ____ int myInt;
    2. ____ A(int newInt) {
        myInt = newInt;
    }

    3. ____ int getInt() {
        return myInt;
    }
}

package pack2;
import pack1.A;

class B extends A
{
    4. ____ A myA;
    B() {
        super(2);
        myA = new A(2);
        myInt = this.getInt() + 1;
    }
    A getMyA()
    {
        return myA;
    }
}

package pack2;
public class C extends B{
    5. ____ B myB;
    6. ____ C() {
        myB = new B();
        this.myA = myB.getMyA();
    }

    public static void main(String[] args) {
        C c = new C();
    }
}
```

2) נמקו בקצרה את תשובתכם לסעיף הקודם

שאלה 5

כתבו מתודה המקבלת מערך (*array*) של מספרים שלמים (*int*) ומחזירה את מספר המספרים השונים במערך. על חתימת הפונקציה להיות בפורמט הבא:

```
static int distinctElements(int[] arr);
```

לדוגמה:

- עבור המערך [1,5,2,1,1,5] הפונקציה תחזיר 3 (1, 5 ו-2 הם המספרים היחידים במערך)
- עבור המערך [0,1,3,7] הפונקציה תחזיר 4

הערות:

- נתון כי כל המספרים הנתונים נמצאים בטווח 0-1000000
- על הקוד שלכם להיות יעיל במקום ובזמן

Common mistakes:

- Checking whether a Set contains an element before adding it. This check is unnecessary in a Set.
- Illegal Java syntax. In particular, a number of students created instances of interfaces (like Set) and many students created a Collection holding primitives.

שאלה 6

נתונה המחלקה Point המייצגת נקודה במרחב דו-ממדי. המחלקה אינה ניתנת לשינוי.

```
public class Point {
    private int x, y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public double norm() {
        return Math.sqrt(x * x + y * y);
    }
    public Point add(Point other) {
        return new Point(this.x + other.x, this.y + other.y);
    }
}
```

כתבו מתודה המקבלת רשימה (*List*) של נקודות ומספר שלם *k* ומחזירה רשימה עם *k* הנקודות הקרובות ביותר לראשית הצירים מתוך הנקודות הנתונות. לפונקציה החתימה הבאה:

```
public static List<Point> getKClosestToOrigin(List<Point> points, int k)
```

(שימו לב – יש דוגמא בעמוד הבא)

norm	point
1.0	(1,0)
4.24	(-3,3)
1.41	(-1,1)
5.0	(5,0)
2.0	(0,2)

דוגמא:

אנו מייצגים נקודה בצורה (x,y) . כאשר הפונקציה נקראת עם $k=3$ ורשימת הנקודות:
 $\{(1,0), (-3,3), (-1,1), (5,0), (0,2)\}$

ערך ההחזרה של הפונקציה יהיה רשימה באורך 3 עם הנקודות

$\{(1,0), (-1,1), (0,2)\}$

כי אלו הן 3 הנקודות בעלות הנורמה הקטנה ביותר (כפי שניתן לראות בטבלה)

הנחות והוראות נוספות:

- ניתן להניח שרשימת הנקודות הנתונה היא באורך לפחות k
- ניתן להניח כי אין שתי נקודות באותו המרחק מהראשית
- הנכם רשאים לכתוב מחלקות/מתודות עזר נוספות אם יש צורך בכך

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 (לדוגמה הפונקציה sort של Collections).

שאלה 1

הנביא יונה מאס בניהול מספר ארנקים שונים למעקב אחר המטבעות השונים שלו (ראה המשך). לשם כך החליט להטיל עליכם את המשימה לממש עבורו ארנק אחד שיבצע מעקב זה.

בעולם יש שתי קטגוריות של מטבעות:

1. מטבע פיאת (FiatCurrency)
2. מטבע וירטואלי (CryptoCurrency)

מספר דוגמאות למטבעות הם:

- מטבע פיאת בשם "שקל" שערכו בדולרים הוא 0.28
- מטבע פיאת בשם "דולר" שערכו בדולרים הוא 1
- מטבע וירטואלי בשם "ביטקוין" שערכו בדולרים הוא 10000

כמובן שיכולים להיות מטבעות נוספים.

הגדרות והסברים נוספים:

1. שימו לב כי כל מטבע משתייך לאחת מהקטגוריות ולאחת בלבד
2. לכל מטבע יש שם באנגלית
3. לכל מטבע יש ערך בדולרים
4. כל שני מטבעות עם שם זהה מייצגים את אותו המטבע. בפרט, אין שני מטבעות מקטגוריות שונות בעלי אותו שם
5. שם של מטבע חייב להיקבע בזמן יצירתו (בהמשך השאלה הסבר נוסף לגבי מי יוצר מטבעות)
6. ערך של מטבע יכול להיקבע בזמן יצירתו או לאחר מכן. אם הערך אינו נקבע בזמן היצירה יקבל המטבע ערך דיפולטיבי אפס
7. כל מטבע חשוף לאינפלציה (inflation)
 - o הפעלת inflation על מטבע פיאת תגרום לירידת ערכו ל-90 אחוז מערכו המקורי (כלומר ערכו יוכפל ב-0.9)
 - o הפעלת inflation על מטבע וירטואלי תגרום לירידת ערכו ל-80 אחוז מערכו המקורי (כלומר ערכו יוכפל ב-0.8)
8. כל ארנק יכול להכיל מספר מטבעות שונים (מספר מטבעות פיאת שונים ומספר מטבעות וירטואלים שונים). לכל מטבע, הארנק שומר את הכמות ממטבע זה הנמצאת בארנק

(א) ממשו שתי מחלקות המייצגות את **FiatCurrency** ו- **CryptoCurrency**. באפשרותכם לממש טיפוסים (type) נוספים שיעזרו לכם במימוש המחלקות הנ"ל. זכרו כי כל מטבע תומך ב-4 פעולות:

הפעלת אינפלציה על המטבע (ראה הגדרות לתיאור התוצאה של פעולה זו)	<code>void inflation()</code>
מחזיר את שם המטבע	<code>String getName()</code>
מחזיר את ערך המטבע בדולרים	<code>double getUSDValue()</code>
משנה את ערך המטבע למספר הנתון	<code>void setUSDValue(double usdValue)</code>

הסבירו את החלטות המימוש שלכם בקצרה.

(ב) בסעיף זה אתם נדרשים לממש את המחלקה **Wallet** אשר מייצגת ארנק. המחלקה צריכה לממש את כל המתודות הבאות:

בנאי היוצר ארנק חדש ריק	<code>Wallet()</code>
הוספה של <code>amount</code> מטבעות מסוג <code>coin</code> לארנק	<code>_____ addCoin(_____ coin, double amount)</code>
מחזיר את שווי כל המטבעות בארנק <u>בדולרים</u>	<code>double getWalletBalance()</code>

הסבירו את החלטות המימוש שלכם בקצרה.

הבהרות:

- שימו לב כי חלק מטיפוסי הפרמטרים וערכי ההחזרה חסרים ב API. עליכם להשלים אותם בצורה המתאימה ביותר לנתוני השאלה ובהתאם למימושכם
- ארנק מכיל אוסף של מטבעות, וכמות מכל מטבע שנמצא בארנק
- על הארנק לתמוך בכל סוגי המטבעות
- הנכם רשאים להוסיף מתודות עזר נוספות שאינן מופיעות ב-API
- במתודה `addCoin`, אם המטבע הנתון כבר מופיע בארנק, הכמות הנתונה מתווספת על הכמות הקיימת שלו בארנק
- השווי בדולרים של מטבע `c` כלשהו שנמצא בארנק הוא הערך של המטבע בדולרים כפול הכמות ממטבע זה הנמצאת בארנק

ג) בשני הסעיפים, נמקו על עיצובכם בנוסף למימוש הנדרש.

1.ג) הרגולטור החליט שהבנק הוא האחראי הבלעדי על הנפקת (יצירת) מטבעות. כך יוכל הבנק לשלוט בערך המטבע ובתזמוני האינפלציה. כעת, כל מי שרוצה מופע של מטבע צריך לקבל אותו מהבנק. לשם כך הבנק הגדיר מתודה סטטית בשם `getCoin` (ראו חתימה מטה) אשר תחזיר מטבעות ע"פ דרישה. בנוסף, הבנק החליט שאם נוצר מופע של מטבע מסוים, אין טעם ליצור מופע נוסף שלו בעתיד, שכן מופע זה ייצג את אותו המטבע בדיוק.

ממשו את המתודה `getCoin` במחלקה **Bank** אשר מקבלת שם של מטבע רצוי ואת הטיפוס שלו ומחזירה מופע של המטבע המתאים. הבנק הגדיר `enum` שמייצג סוג מטבע.

```
public enum CoinType {FIAT, CRYPTO}
```

חתימת המתודה `getCoin` היא:

```
public static _____ getCoin(String name, CoinType type)
```

* שימו לב, גם כאן ערך ההחזרה חסר ועליכם להשלים אותו בהתאם למימושכם

זכרו כי על פי החלטת הבנק, המתודה צריכה לממש את ההתנהגות הבאה:

בהינתן שם של מטבע `name` וטיפוס `type`:

- אם בעבר נוצר מופע של מטבע עם שם `name`, יוחזר מופע זה
- אחרת, ניצור מופע של מטבע מטיפוס `type` ושם `name` עם ערך דיפולטיבי 0 ונחזיר אותו. בנוסף, נשמור מופע זה כדי שנוכל להחזירו בקריאות הבאות

עליכם לחשוב איך בדיוק לשמור מופע של מטבע שנוצר לשימוש בקריאות הבאות. הנכם רשאים להוסיף קוד נוסף במחלקה **Bank** מחוץ למתודה `getCoin`.

הקוד הבא מדגים את ההתנהגות הרצויה:

```
_____ btc1 = Bank.getCoin("BTC", CoinType.CRYPTO);  
_____ btc2 = Bank.getCoin("BTC", CoinType.CRYPTO);  
System.out.println(btc1 == btc2); // output is true
```

2.ג) כפי שהוזכר, הבנק אחראי גם על אינפלציית המטבעות. ממשו מתודה נוספת עבור המחלקה **Bank בשם `inflation` שאחראית על הפעלת אינפלציה על כל אחד מהמטבעות הקיימים בעולם (כל המטבעות שנוצרו אי פעם על ידי הבנק).**

חתימת המתודה:

```
static void inflation()
```

הנחות והנחיות נוספות:

- הפונקציה צריכה להפעיל את פעולת `inflation` על כל מטבע בהתאם לקטגוריה אליה משתייך המטבע (הגדרת פעולת אינפלציה לכל קטגוריה מופיעה בתחילת השאלה)
- פעולת אינפלציה אמורה להשפיע על כל המטבעות שנמצאים בארנקים (**Wallet**) של אנשים
- ניתן להניח כי יצירת מטבעות מתבצעת רק בבנק, במתודה `getCoin` שכתבתם בסעיף **1.ג)**

Common mistakes:

- When you create a Hash-based data structure, the type being hashed should override the equals and hashCode methods of Object. E.g a HashMap with keys of special type Currency
- Generic type can't be instantiated with primitive type, e.g. **HashMap<Currency, double>**
- A sub-class does not inherit the constructors of its super-class. Therefore, if it needs a special constructor, the class itself must implement it. You can avoid implementing a constructor only if the super-class has a zero-arguments constructor
- Static method can't access non-static data members. Many students used a non-static data member for the Currencies saving mechanism in Bank.getCoin
- Abstract methods must be declared using the abstract modifier
- Holding two data structures - one for each type of Currency - is a bad implementation. It hurts the single-choice principle. A better approach is to use a common type for all currencies and have a data structure of that type instead
- Final members defined in a super-class cannot be assigned in a sub-class
- Primitives can't be compared to null. In fact, primitives can't ever have a null value
- The wallet balance should've been calculated in the getWalletBalance method. If the balance is calculated when a coin is added, the inflation action doesn't affect the wallet's balance, though it should
- A double value can't be assigned to an integer variable (unless we do it explicitly and knowingly using casting)

שאלה 2

חברת מכשירי תקשורת מייצרת מכשירים למגוון לקוחות. המכשירים מתקשרים ביניהם **רק** באמצעות שרת (Server) המנתב את ההודעות למכשירים המתאימים. נתון כי לאובייקט **CommServer** יש את ה-API הבא:

מתודה היוצרת חיבור בין שני מכשירי קשר	<code>public boolean addConnection(CommDevice d1, CommDevice d2)</code>
מתודה המנתקת חיבור בין שני מכשירי קשר. אם הקשר לא קיים, לא קורה כלום	<code>public void removeConnection(CommDevice d1, CommDevice d2)</code>
מתודה ששולחת את ההודעה msg מהמכשיר d למכשיר אליו הוא מחובר. אם ההודעה נשלחה מוחזר true ואחרת מוחזר false	<code>public boolean sendMessage(CommDevice d, String msg)</code>

מהנדסי החברה כבר ממשו את מחלקת **CommServer** (עם ה-API הנתון) אבל צריכים עזרה במימוש מכשירי הקשר.

החליטו בחברה שמכשירי קשר יכולים לתקשר רק עם מכשיר קשר אחד בכל פעם, וכי אפשר ליצור קשר בין שני מכשירים **רק** כאשר שניהם לא מקושרים למכשיר אחר. לאחר יצירת קשר, המכשירים יכולים לשלוח איזה מספר הודעות שהם צריכים עד לניתוק אותו הקשר (לאחר מכן יהיה צריך לחדש את הקשר). כדי לאפשר ל Server לענות על הדרישות הללו הוחלט שכל מכשיר קשר חייב לעקוב אחר ה-API הבא:

מתודה המבקשת מה-Server להתחבר למכשיר קשר אחר. אם יצירת הקשר מצליחה, מוחזר true ואחרת false. לא ניתן ליצור קשר בין מכשיר לעצמו!	<code>public boolean connect(CommDevice d)</code>
מתודה המבקשת מה-Server לנתק את הקשר. אם המכשיר כבר לא מקושר, לא קורה כלום	<code>public void removeConnection()</code>
מתודה המאפשרת יצירת קשר חדש עם המכשיר d. המתודה נקראת על ידי ה-Server כשהמכשיר d מבקש להתחבר למכשיר, מחזירה true אם ניתן ליצור קשר ואחרת false	<code>public boolean acceptConnection(CommDevice d)</code>
מתודה המקבלת הודעה מה-Server. כאשר ה-Server מתבקש להעביר הודעה למכשיר, הוא קורא למתודה זו עם ההודעה שקיבל	<code>public void receiveMessage(String msg)</code>
מתודה שמבקשת מה-Server לשלוח הודעה למכשיר אליו מקושרים באותו הרגע. המתודה מחזירה true אם ההודעה נשלחה (כלומר, אם עדיין יש קשר עם מכשיר) ו false אחרת	<code>public boolean sendMessage(String msg)</code>
מתודה המדפיסה את ההודעה האחרונה שהתקבלה	<code>public void printMessage()</code>

מנהלי צוותי הפיתוח בחברה רוצים להיות מסוגלים לייצר מספר מכשירי קשר עם התכונות שפורטו בקלות.

דוגמא: נניח שיש שרת אחד S ומכשירי הקשר A ו-B רוצים לתקשר

```
>> <A asks S to connect to B>
>> <S asks if B accepts>
>> <B isn't connected to anyone, so accepts the connection>
>> <S connects between A and B>
>> A: Hi B, it's me!
>> <S sends message to B, returning true>
>> B: Bye
<< <S sends message to A, returning true>
>> <B asks S to disconnect from A>
>> <S disconnects A and B from each other>
>> A: B?
>> <S sees that A is not connected and returns false>
>> A: :(
>> <S sees that A is not connected and returns false>
>> <A asks S to disconnect from B (so it will be free for other
connections)>
>> <S does nothing>
```

שימו לב לכאורה, שני השלבים האחרונים מיותרים, אבל הוחלט שהשרת לא חייב לדווח שמכשיר ניתק קשר למכשיר השני

הערות:

- אין צורך לשמור היסטוריה של הודעות, רק את ההודעה האחרונה שהתקבלה
- שימו לב ש **CommServer** קורא ישירות ל-`receiveMessage` ול-`acceptConnection` בתזמונים הרלוונטיים
- שימו לב, השרת לא מדווח למכשיר קשר אם ניתקו אותו

(א) ממשו את מכשיר הקשר הבסיסי ביותר על פי ה-API הנתון (מכשיר הקשר בעל התכונות הבסיסיות בלבד).
הסבירו בקצרה איך המימוש שלכם מאפשר יצירת מגוון מכשירי קשר עם התכונות הנדרשות.

(ב) החברה קיבלה תלונות מלקוחותיה על כך שהיא לא מאפשר הצפנה בתקשורת בין המכשירים שלה. המהנדסים הראשיים של החברה הבחינו שמדובר בפגיעה בפרטיות הלקוחות שלה והחליטו להוסיף תוסף המאפשרת הצפנת הודעות הלקוחות על פי אלגוריתם הצפנה לבחירתם. הוחלט כי לתוסף החדש הבנאי הבא:

```
public EncoderDevice (CommDevice d, HashMap<Character, Character> encoder,
    HashMap<Character, Character> decoder);
```

עזרו למהנדסים להוציא מוצר חדש המאפשר הוספת הצפנה למכשיר קשר שהלקוח כבר קנה (בלי לשלוח לו מחלקת מכשיר קשר חדשה!). על המכשיר החדש להיות מסוגל להצפין הודעות.

האם שיטת ההצפנה שהציעו מהנדסי החברה (בעזרת HashMap) עונה על עקרונות העיצוב שנלמדו בקורס? ספציפית, האם זוהי השיטה הכי כללית/גנרית? אם לא, הציעו עיצוב (design) מוצלח יותר. נמקו בקצרה.

הערות:

- כאשר המכשיר שולח הודעה, עליו להצפין אותה על ידי החלפת כל תו בהודעה בתו המתאים לו ב-encoder (הנתון באתחול)
- כאשר המכשיר מקבל הודעה, עליו לפענח כל הודעה באותו אופן על ידי decoder
- הניחו שכל לקוח יודע שלכל תו צריך להיות תו מקביל גם ב-encoder וגם ב-decoder

(ג) החברה החליטה שמעבר לתוסף ההצפנה, יש להוסיף את האופציה לאימות הקשר בין מכשירים בעת יצירת החיבור ביניהם. הוחלט לממש תוסף נוסף שיתאים לכלל מכשירי החברה, לפיו כאשר מכשיר א' יוצר קשר עם מכשיר ב' על שני המכשירים לשלוח קוד אימות (בצורת הודעה) למכשיר השני. בקבלת ההודעה, כל מכשיר מוודא שהקוד שקיבל הוא אכן הקוד השמור ליצירת קשר עמו, ואם הקוד שגוי הקשר מיד מתנתק.

התוסף החדש צריך לממש את הבנאי הבא:

```
public AuthenticatorDevice (CommDevice d, String sendKey, String
    receiveKey);
```

עזרו למהנדסים לממש את התוסף החדש.

הסבירו בקצרה כיצד ניתן להשתמש בשני התוספים במקביל וגם בנפרד. למשל, איך מתייחסים למקרה בו לקוח A משתמש רק במנגנון האימות, לעומת לקוח B שהחליט שהוא רוצה גם אימות וגם הצפנה. האם הצורה שבה מימשתם את הקוד הינה קלה להרחבה (extensible; כלומר קל ליצור תוספים נוספים)?

הערות:

- כפי שתואר, כשמכשיר A יוצר קשר עם מכשיר B, ההודעה הראשונה ש B מצפה לקבל היא sendKey וההודעה הראשונה שמכשיר A מצפה לקבל היא receiveKey

דוגמא לתהליך יצירת הקשר:

נניח שהקוד הוא שהמכשיר שיוצר את הקשר צריך לשלוח הוא **itsme** וקוד האימות שמקבל הקשר צריך לשלוח הוא **iknow**. כאשר שני המכשירים מכירים באותם הקודים, תהליך יצירת הקשר יהיה

```
>> <d1 connects to d2>
>> d1: itsme           // d1 sends the correct sendKey
>> <d2 verifies code>   // d2 checks that the sendKey is correct
>> d2: iknow           // d2 sends the correct receiveKey
>> <d1 verifies code>   // d1 checks that the receiveKey is correct
>> d1: //some message  // regular communication can follow
```

אם d2 לא מכיר את הקוד הזה, נקבל

```
>> <d1 connects to d2>
>> d1: itsme           // d1 sends the correct sendKey
>> <d2 verifies code>   // d2 checks that the sendKey is correct
>> <d2 disconnects from d1> // d2 doesn't recognize the sendKey
```

לבסוף, אם d2 מתחזה

```
>> <d1 connects to d2>
>> d1: itsme           // d1 sends the correct sendKey
>> <d2 verifies code>   // d2 checks that the sendKey is correct
>> d2: Hi!             // d2 doesn't send the correct receiveKey
>> <d1 verifies code>   // d1 checks that the receiveKey is correct
>> <d1 disconnects from d2> // d1 doesn't get the correct receiveKey
```

Common Mistakes:

- Not creating an instantiable class for the simple CommDevice (i.e. only creating an abstract class called CommDevice but not extending a SimpleDevice from it)
- Trying to use a foreach loop on a String type during encryption/decryption. Since a String is not iterable doing this will result in a compilation error, but even worse, does not make much sense
- Many people forgot to answer or misunderstood the design question in section B
- Again, in the design question of section B, a lot of students answered that the best design to use is Strategy, but we weren't looking for the name of the best design, just what you would do (if someone called it Strategy, but wrote the answer we gave above, they got all the points, of course). The problem with saying that they should use Strategy is that in Strategy we are still limiting the encoding algorithms to a family of algorithms that the engineers wrote, but a customer might decide that he wants a completely different type of encoding than what we might expect (for instance, adding a random symbol after each char of the message, i.e. "a message" -> "af 1mge4s«s\$afg`e~" or adding a random symbol after every 2 chars, etc.)
- In the same question, making the hashmaps generic or even just using one hashmap are, in the same way that the above answer is incorrect, wrong answers
- A lot of people didn't write an explanation for their implementation, which is a shame because if you would have written an explanation but didn't have time to implement, you would still get some points (sure, a small amount compared to what you would have gotten otherwise, but still you would get *something*)
- I'm glad to say that most students understood that the correct design pattern is to use a decorator for the encoding/authentication. However, some students forgot to extend the normal CommDevice (and then, when using the CommServer, you would get a compilation error).
- There were many different opinions on how the CommServer should work (i.e. if it should be passed, created in CommDevice, etc.). The best answer would be to pass it in the constructor (and many did this), but we will also receive answers where the server was instantiated in the constructor of CommDevice. However, making the server final/static is wrong! Especially if you are inheriting from a class, since you are limiting the design of all of the children of CommDevice (what if we want a device that moves between different servers?)
- In the authentication process, some assumed that messages are sent in parallel to the "acceptConnection" requests... this is not possible with the API that you were given for CommServer
- Finally, many students didn't have time to implement the authentication device... in many CS exams (and especially in OOP) time is a big constraint. Not getting to all of the questions is okay, however (for future exams) try to read all of the questions in the exam before you start answering and then plan your time in advance (this is counter intuitive). I don't know if it would have helped here, but it did help me in my exams.

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

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