תוכנה 1

תרגול מס' 6

ירושה, מחלקות אבסטרקטיות, IO

ירושה ממחלקות קיימות

- ראינו בהרצאה שתי דרכים לשימוש חוזר בקוד של מחלקה קיימת:
  - הכלה + ה
  - ירושה

- המחלקה היורשת יכולה **להוסיף** פונקציונאליות שלא היתה קיימת במחלקת הבסיס, או **לשנות** פונקציונאליות שקיבלה בירושה

דריסת שירותים

- המחלקה היורשת בדרך כלל מייצגת תת-משפחה של מחלקת הבסיס
- המחלקה היורשת יכולה לדרוס שירותים שהתקבלו בירושה
- כדי להשתמש בשירות המקורי (למשל מהשירות הדורס) ניתן לפנות לשירות המקורי בתחביר: super.methodName(…)

שימוש בשירות המקורי מתוך השירות הדורס

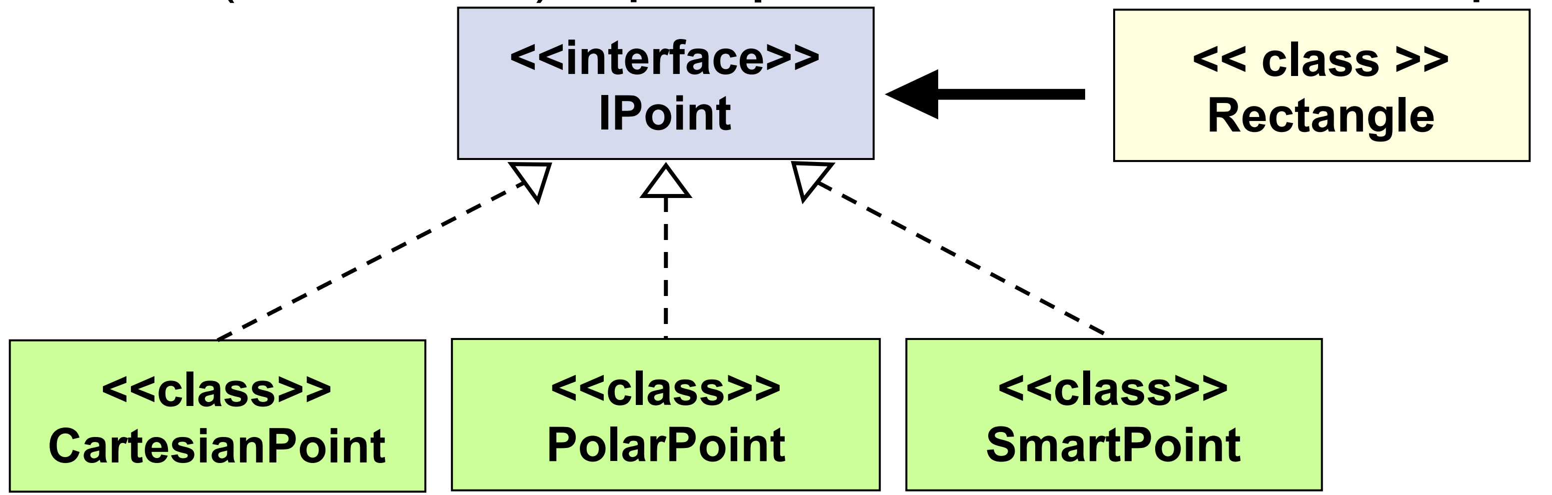
```
class B {
    protected int a;
    protected int b;
    public String toString(){
        return "a: " + this.a + " b: " + this.b;
    }
}

class C extends B{
    private int c;
    public String toString(){
        return super.toString() + " c: " + this.c;
    }
}
```

ניראות והורשה

- שדות ושירותים פרטיים (private) של מחלקת הבסיס אינם נגישים למחלקה היורשת
- כדי לאפשר גישה למחלקות יורשות יש להגדיר להם נראות protected
  - שימוש בירושה יעשה בזהירות מרבית, בפרט הרשאות גישה למימוש
  - נשתמש ב protected רק כאשר אנחנו מתכננים היררכיות ירושה שלמות ושולטים במחלקה היורשת
  - נכון ל-java21, מנשקים **לא מאפשרים** להגדיר מתודות כ-protected

צד הלקוח

- בהרצאה ראינו את המנשק IPoint, והצגנו 3 מימושים שונים עבורו
- ראינו כי **לקוחות** התלויים במנשק IPoint בלבד, ואינם מכירים את המחלקות המממשות, יהיו **אדישים** לשינויים עתידים בקוד הספק
- שימוש **במנשקים** חוסך **שכפול בקוד לקוח,** בכך שאותו קטע קוד עובד בצורה נכונה עם מגוון ספקים (פולימורפיזם)



המנשק IPoint

```
public interface IPoint {
    /** returns the x coordinate of the current point*/
    public double getX();

    /** returns the y coordinate of the current point*/
    public double getY();

    /** returns the distance between the current point and (0,0) */
    public double rho();

    /** returns the angle between the current point and the abscissa */
    public double theta();

    /** move the current point by dx and dy */
    public void translate(double dx, double dy);

    /** rotate the current point by angle degrees with respect to (0,0) */
    public void rotate(double angle);

    …
}
```

צד הספק

- לעומת זאת, **מנגנון ההורשה** חוסך **שכפול קוד בצד הספק**
- ע"י הורשה מקבלת מחלקה את קטע הקוד בירושה במקום לחזור עליו. שני הספקים חולקים אותו הקוד
- פתרון אלטרנטיבי הוא להשתמש במתודות דיפולטיות
- ננסה לזהות את שכפול הקוד בין 3 מימושי המנשק IPoint ולרכז קטעים משותפים אלה במחלקת בסיס משותפת ממנה ירשו שלושת המימושים.

Sealed Classes

- מ-Java 17, נוספה המילה השמורה sealed, שמאפשרת להגדיר עבור מנשק\מחלקה אילו מנשקים\מחלקות יכולים להרחיב\לממש אותה
- מאפשר שליטה עדינה יותר על מבנה הירושה, לעומת השימוש ב-final
- כל מחלקה\מנשק שהוא חלק מהיררכיית ירושה כזו חייב להיות final, sealed או non-sealed

```
public abstract sealed class Shape
    permits Rectangle, Square {...}

public sealed class Rectangle extends Shape
    permits TransparentRectangle{...}

public final class TransparentRectangle extends Rectangle {...}

public non-sealed class Square extends Shape {...}
```

ניתן להרחבה ללא -non sealed

Sealed Classes

- דוגמא יותר מעניינת להרחבה שהיא non-sealed:
  - במקרה הזה, שימושי לsecurity (לפחות למראית עין)

```
public sealed interface Command
    permits LoginCommand, LogoutCommand, …, UserPluginCommand
{...}

public final class LoginCommand extends Command {...}
…
public non-sealed abstract class UserPluginCommand extends
Command {...}
```

מחלקות מופשטות Abstract Classes

- מחלקה מופשטת מוגדרת ע"י המלה השמורה **abstract**
- לא ניתן ליצור מופע של מחלקה מופשטת (בדומה למנשק)
- יכולה לממש מנשק מבלי לממש את כל השירותים המוגדרים בו
- זהו מנגנון המועיל להימנע משכפול קוד במחלקות יורשות

מחלקות מופשטות - דוגמא

```
public abstract class A {
    public void f() {
        System.out.println("A.f!!");

    }

    abstract public void g();
}

A a = new A();   X

public class B extends A {
    public void g() {
        System.out.println("B.g!!");
    }
}

A a = new B();
```

CartesianPoint	PolarPoint
`private double x;` `private double y;`	`private double r;` `private double theta;`

`public CartesianPoint(double x, double y) {`
`    this.x = x;`
`    this.y = y;`
`}`
	`public PolarPoint(double r, double theta) {`
	`    this.r = r;`
	`    this.theta = theta;`
	`}`

`public double getX()  { return x;}`	`public double getX() {  return r * Math.cos(theta); }`
`public double getY()  { return y;}`	`public double getY() {  return r * Math.sin(theta); }`
`public double rho()  { return Math.sqrt(x*x + y*y); }`	`public double rho() { return r;}`
`public double theta()  { return Math.atan2(y,x);}`	`public double theta() { return theta;  }`

קשה לראות דמיון בין מימושי המתודות במקרה זה.
כל 4 המתודות בסיסיות ויש להן קשר הדוק לייצוג שנבחר לשדות

CartesianPoint

```
public double distance(IPoint other) {
    return Math.sqrt((x-other.getX()) * (x-other.getX()) +
                     (y-other.getY())*(y-other.getY()));
}
```

PolarPoint

```
public double distance(IPoint other) {
    double deltaX = getX()-other.getX();
    double deltaY = getY()-other.getY();

    return Math.sqrt(deltaX * deltaX +
                     deltaY * deltaY);
}
```

הקוד דומה אבל לא זהה, נראה מה ניתן לעשות...

ננסה לשכתב את CartesianPoint ע"י הוספת משתני העזר deltaX ו- deltaY

CartesianPoint

```
public double distance(IPoint other) {
	double deltaX = x-other.getX();
	double deltaY = y-other.getY();

	return Math.sqrt(deltaX * deltaX +
				(deltaY * deltaY );
}
```

PolarPoint

```
public double distance(IPoint other) {
	double deltaX = getX()-other.getX();
	double deltaY = getY()-other.getY();

	return Math.sqrt(deltaX * deltaX +
				deltaY * deltaY);
}
```

נשאר הבדל אחד:
נחליף את x להיות getX() –
במאזן ביצועים לעומת כלליות נעדיף תמיד את הכלליות

CartesianPoint

```
public double distance(IPoint other) {
      double deltaX = getX()-other.getX();
      double deltaY = getY()-other.getY();

      return Math.sqrt(deltaX * deltaX +
                        deltaY * deltaY );
}
```

PolarPoint

```
public double distance(IPoint other) {
      double deltaX = getX()-other.getX();
      double deltaY = getY()-other.getY();

      return Math.sqrt(deltaX * deltaX +
                        deltaY * deltaY );
}
```

שתי המתודות זהות לחלוטין!
עתה ניתן להעביר את המתודה למחלקה AbstPoint ולמחוק אותה מהמחלקות CartesianPoint ו- PolarPoint

CartesianPoint

```
public String toString(){
	return "(x=" + x + ", y=" + y +
		", r=" + rho() + ", theta=" + theta() + ")";
}
```

PolarPoint

```
public String toString() {
	return "(x=" + getX() + ", y=" + getY() +
		", r=" + r + ", theta=" + theta + ")";
}
```

תהליך דומה ניתן גם לבצע עבור toString

מימוש המחלקה האבסטרקטית

```
public abstract class AbstractPoint implements IPoint{
    public double distance(IPoint other) {
        double deltaX = getX()-other.getX();
        double deltaY = getY()-other.getY();

        return Math.sqrt(deltaX * deltaX  +   deltaY *
        deltaY );
    }

    public String toString() {
        return "(x=" + getX() + ", y=" + getY() +
               ", r=" + rho() + ", theta=" + theta() +
        ")";
    }
}
```

ירושה מהמחלקה האבסטרקטית

```
public class PolarPoint extends AbstractPoint{
    private double r;
    private double theta;

    public PolarPoint(double r, double theta) {
        this.r = r;
        this.theta = theta;
    }

    @Override
    public double getX() {
        return r * Math.cos(theta);
    }

    @Override
    public void rotate(double angle) {
        theta += angle;
    }
     …
}
```

IO

המחלקה Scanner

- סורק טקסט פשוט אשר יודע לחלץ טיפוסים פרימיטיביים ומחרוזות.
- import java.util.Scanner;
- "שובר" את הקלט לרכיביו השונים (מילה, מספר וכדומה).
- בעת היצירה מקבל כפרמטר מהיכן לקרוא את הקלט.
  - בפרט, יכול לאפשר לנו לקרוא קלט מהמשתמש.

```
Scanner scanner = new Scanner("1 1.4 the long\nand winding road.");
int anInt = scanner.nextInt();       //1
float aFloat = scanner.nextFloat(); //1.4
String aString = scanner.next();    //the
String aLine = scanner.nextLine();  // long
String bLine = scanner.nextLine();  //and winding road.
```

https://docs.oracle.com/javase/8/docs/api/index.html?java/util/Scanner.html

שימוש ב delimiter ב Scanner

```java
String input = "1 fish 2 fish red fish blue fish ";
Scanner s = new Scanner(input).useDelimiter(" fish ");
while (s.hasNext())
	System.out.println(s.next());
s.close();
```

Output:
1
2
red
blue

ה delimiter הדיפולטי מבצע הפרדה על תוים לבנים (רווחים, ירידות שורה, טאבים ועוד).

המתרגם

- משימה:
  - תכנית המתרגמת קטעי טקסט לשפה אחרת
  - הקלט: קובץ המכיל את קטעי הטקסט וכן את השפה אליה רוצים לתרגם.
- שאלות:

  - האם כבר יש שירות תרגום שאנחנו יכולים להשתמש בו?
  - כיצד קוראים מקבצים?
  - מה הפורמט של הקלט?

פתרון צעד אחר צעד

- כצעד ראשון נפתור בעיה הרבה יותר פשוטה.

- תכנית שמתרגמת את המילה “Hello” מאנגלית לצרפתית
  - יש: שימוש בשירות תרגום.
  - אין: קלט, טקסט, עבודה עם קבצים, פורמט.

API -
Application Programming Interface

- ממשק המאפשר לאפליקציה לתקשר עם תוכנה אחרת.
- בג'אווה קיימים כלים רבים הזמינים ברשת בקוד פתוח.
- בתרגול זה נשתמש ב-API תרגום כללי Translate.
  - ברשת קיימים כלים שונים של Google, Microsoft ועוד.

שלב א'

```
public class TranslatorEngine1 {

  public static void main(String[] args) {

              Bonjour
  String TranslatedText = Translate.execute("Hello", "English",
    "French");

    System.out.println(TranslatedText);
  }
}
```

מתודה סטטית, שנקראת מן המחלקה (Translate). ניתן להניח שקיים מימוש של Translate בפרוייקט שלנו

שלב ב'- אינטראקציה עם המשתמש

- קלט מהמשתמש יינתן בשורת הפקודה.
  - פרמטר ראשון: המילה לתרגום.
  - פרמטר שני: שפת המקור.
  - פרמטר שלישי: שפת היעד.

```
public class TranslatorEngine2 {

  public static void main(String[] args) {
    String TranslatedText =
                    Translate.execute(args[0], args[1], args[2]);
     System.out.println(TranslatedText);
  }
}
```

קלט אינטרקטיבי

- מה אם נרצה להעביר קלט במהלך ריצת התוכנית?

```
>java TranslatorEngine
Enter your input:
Hello English French
Your translation is: Bonjour
```

שימוש ב Scanner לצורך קריאת קלט מהמשתמש

- נשתמש ב Scanner על מנת לקרוא את הקלט מהמשתמש.
  - בתור התחלה, נקרא מה-console (הקלט הסטנדרטי של התכנית) - system.*in*
  - האובייקט system.*in* הוא מטיפוס InputStream עליו נדבר בהמשך הקורס.

```
Scanner scanner = new Scanner(System.in);
int anInt = scanner.nextInt();
…
```

דוגמא

קרא מ- standard input

```
Scanner s = new Scanner(System.in);
System.out.println("enter line:");
while (s.hasNext())
     System.out.println(s.next());

s.close();
```

קרא את ה- Token הבא

מתי הקוד הזה יעצור?

שלב ג' – שימוש בסיסי ב-Scanner

נבחר את פורמט הקלט:

<word> <source-lang> <target-lang>

לדוגמא,

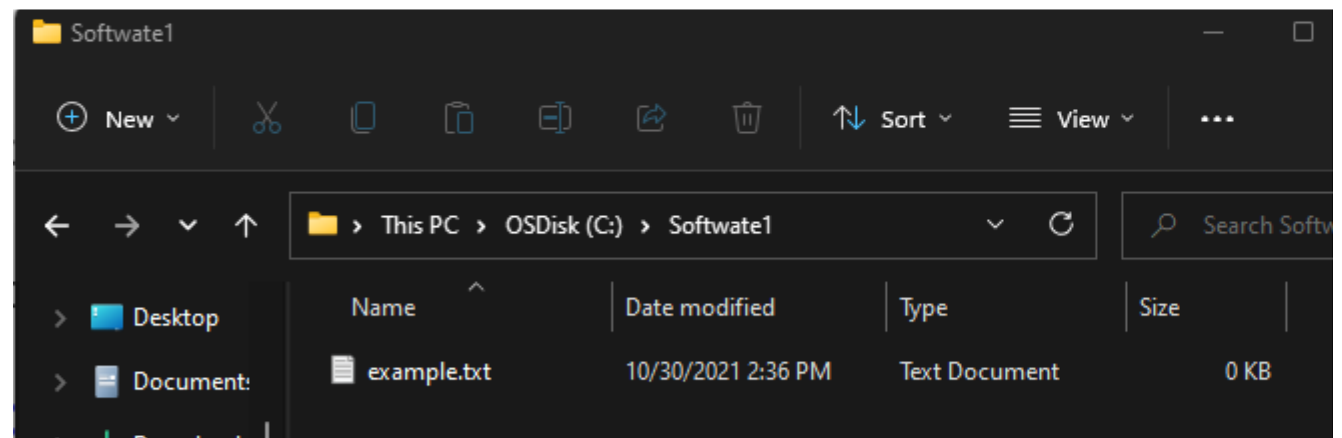
- הקלט: hello English French
- הפלט: bonjour

```
public class TranslatorEngine3 {

  public static void main(String[] args) {
      Scanner s = new Scanner(System.in);
      String[] fragments = s.nextLine().split(" ");
      String TranslatedText =
      Translate.execute(fragments[0], fragments[1], fragments[2]);
      System.out.println(TranslatedText);
      s.close();
  }
}
```

קבצים

- במקום לקרוא את שורת הקלט מהמשתמש נקרא אותה מקובץ.
- קובץ מיוצג ע"י המחלקה java.io.File
- נאתחל את האובייקט עם המסלול (path) לקובץ.

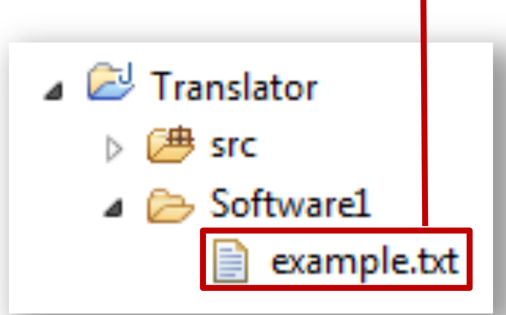
```
String filePath = "C:\\Software1\\example.txt”;
File exampleFile = new File(filePath);
```



מסלול (Path) לקובץ

- מסלול יחסי – Relative path

```
new File("Software1\\example.txt")
```



- ב- eclipse המיקום ה"נוכחי" במהלך ריצה הוא ה-Project root
- דרך טובה לבדוק את המיקום הנוכחי של הפרוייקט הוא לייצר קובץ במיקום היחסי, ואז לבדוק היכן הוא נוצר.

מסלול (Path) לקובץ

- מסלול מלא – Absolute path

```
new File("C:\\Software1\\example.txt")
```

- יתרון – ניתן להריץ את התוכנית מכל מקום והיא תמיד תוכל למצוא את הקובץ.
- חסרון - הרבה פעמים הקובץ ממוקם יחסית לתוכנית, ואז אם היא מועתקת, גם הקובץ מועתק והקוד לא ירוץ.
  - טעות נפוצה בתרגילי הבית: הגשת קוד שמכיל מסלול מלא לקובץ:

```
new File("C:\\users\\lenadank\\software1\\ex4\\my_file.txt")
```

מסלול (Path) לקובץ

- כיצד נדאג שהתכנית תתאים לכל מערכת הפעלה? (Windows, Linux...)
  - **פתרון א':**

```
new File("Software1/example.txt")
```

  - **פתרון ב':**

```
new File("Software1" + File.separator + "example.txt")
```

  - **פתרון ג':** נקבל את המסלול כקלט מהמשתמש.

שלב ד' – Scanner וקריאה מקובץ

המסלול לקובץ יהיה (שדה) קבוע של המחלקה

```
public class TranslatorEngine4 {

  private static final String FILE_NAME = "Software1" + File.separator +
       "example.txt";

  public static void main(String[] args) throws Exception {

    Scanner s = new Scanner(new File(FILE_NAME));
    String[] fragments = s.nextLine().split(" ");
    String TranslatedText = Translate.execute(fragments[0],
       fragments[1], fragments[2]);
    System.out.println(TranslatedText);

    s.close();
  }
}
```

זריקת חריגים – הצהרת throws

- בעת חיבור ה-Scanner לקובץ עלולה להיזרק שגיאה (חריג, Exception) מסוג `FileNotFoundException`
  - במקרה שהקובץ ממנו ניסינו לקרוא לא קיים, ריצת המתודה תעצור
  - החריג מכיל הסבר על מקור השגיאה
- שתי אפשרויות להתמודדות: טפלו או זרקו הלאה
  - נדבר על טיפול בחריגים ועוד בהמשך הקורס.
  - כרגע נטפל בחריג באופן הבא:
    - נצהיר על זריקת חריג בחתימת המתודה באמצעות המילה השמורה `throws`.
    - החריג עליו נצהיר יהיה חריג מטיפוס Exception, שהוא החריג הכללי ביותר שיש. כלומר, המתודה שלנו מצהירה שהיא יכולה לזרוק חריג, ומי שקורא לה צריך להיות מודע לזה ולטפל בזה במידת הצורך.

שלב ה' – קלטים מרובים

- מספר שורות קלט מקובץ.
  - נקרא מספר קלטים עד לסוף הקובץ, שימוש ב- hasNextLine

```
public class TranslatorEngine5 {
  private static final String FILE_NAME = "Software1/example5.txt";

  public static void main(String[] args) throws Exception {
    Scanner s = new Scanner(new File(FILE_NAME));
     while (s.hasNextLine()) {
        String[] fragments = s.nextLine().split(" ");
        System.out.println(Translate.execute(fragments[0],
         fragments[1], fragments[2]));
     }
     s.close();
  }
}
```

נרחיב את המחלקה שלנו לטיפול בפיסקאות

- נרצה לקרוא פיסקה, להמיר לשורה אחת, ולתרגם.
- צריך להגדיר את פורמט הקלט מחדש.

נגדיר:

```
<source-lang>#<target-lang>#<paragraph>
```

- למשל:

```
English#French#Hello world!
This program works.
Bye.
```

שלב ו' – תרגום פסקה

```java
public class TranslatorEngine6 {
  private static final String FILE_NAME = "Software1/ example6.txt“;

  public static void main(String[] args) throws Exception {
    Scanner s = new Scanner(new File(FILE_NAME));
    s.useDelimiter("#");
    String srcLanguage = s.next();
    String destLanguage = s.next();
    s.skip("#");
    String text = "";
    while (s.hasNextLine()) {
       text += s.nextLine() + ' ';
    }
    System.out.println(Translate.execute(text, srcLanguage, destLanguage));
    s.close();
  }
}
```

```
English#French#Hello world!
This program works.
Bye.
```

לאן עכשיו?

- טיפול בשגיאות
  - פורמט לא תקין, כשלון בזיהוי השפות או בתרגום
  - ניתן לבדוק בקוד או להגדיר בחוזה
- הרחבת התכנית
  - תרגום מספר קבצים
  - מספר פסקאות בקובץ יחיד
  - זיהוי אוטומטי של שפת הקלט
- ...

Beyond Scanner

- **StringBuilder** – מייצגת מחרוזות ניתנת לשנוי.
- **FileReader/ FileWriter** – קריאה/ כתיבה של תווים מקבצים.
- **BufferedReader** – עוטף את FileReader ומתחזק מאגר מובנה על מנת לצמצם קריאות מהדיסק.
- **BufferedWriter** – עוטף את FileWriter ומתחזק מאגר מובנה על מנת לצמצם כתיבות מהדיסק.

ועוד במדריך I/O ללימוד עצמי...

המחלקה StringBuilder

- מייצגת מחרוזות ניתנת לשנוי (mutable)
- מאפשרת לבצע שינוי במחרוזת קיימת מבלי ליצור עצמים חדשים
- שירותים חשובים: append ו- insert

```
StringBuilder sb = new StringBuilder("abc");
sb.append("d");
```

למה לא לשרשר מחרוזות באמצעות חיבור מחרוזות?

FileReader ו- FileWriter

- קריאה/ כתיבה של תווים מקבצים.

```
FileReader fReader = new FileReader(new File(FILE_IN));
FileWriter fWriter = new FileWriter(new File(FILE_OUT));
char[] charRead = new char[1000];
int numRead;
while ((numRead = fReader.read(charRead)) != -1){
        fWriter.write(charRead, 0, numRead);
}
fReader.close();
fWriter.close();
```

BufferedReader ו-BufferedWriter

- קריאה/ כתיבה תוך שימוש ממאגר מובנה.

```
FileReader fReader = new FileReader(new File(FILE_IN));
FileWriter fWriter = new FileWriter(new File(FILE_OUT));
BufferedReader buffReader = new BufferedReader(fReader);
BufferedReader buffWriter = new BufferedWriter(fWriter);

String line;
while ((line = buffReader.readLine() != null){
      buffWriter.write(line + "\n");
}
buffReader.close();
buffWriter.close();
```

סוגרים גם את fReader/ fWriter

שאלה מבחינה
(מועד א', סמסטר א', תשע"ו)

<u>שאלה 3 (4 נק')</u>

השאלה הבאה מתייחסת לקוד הבא. מה יודפס?

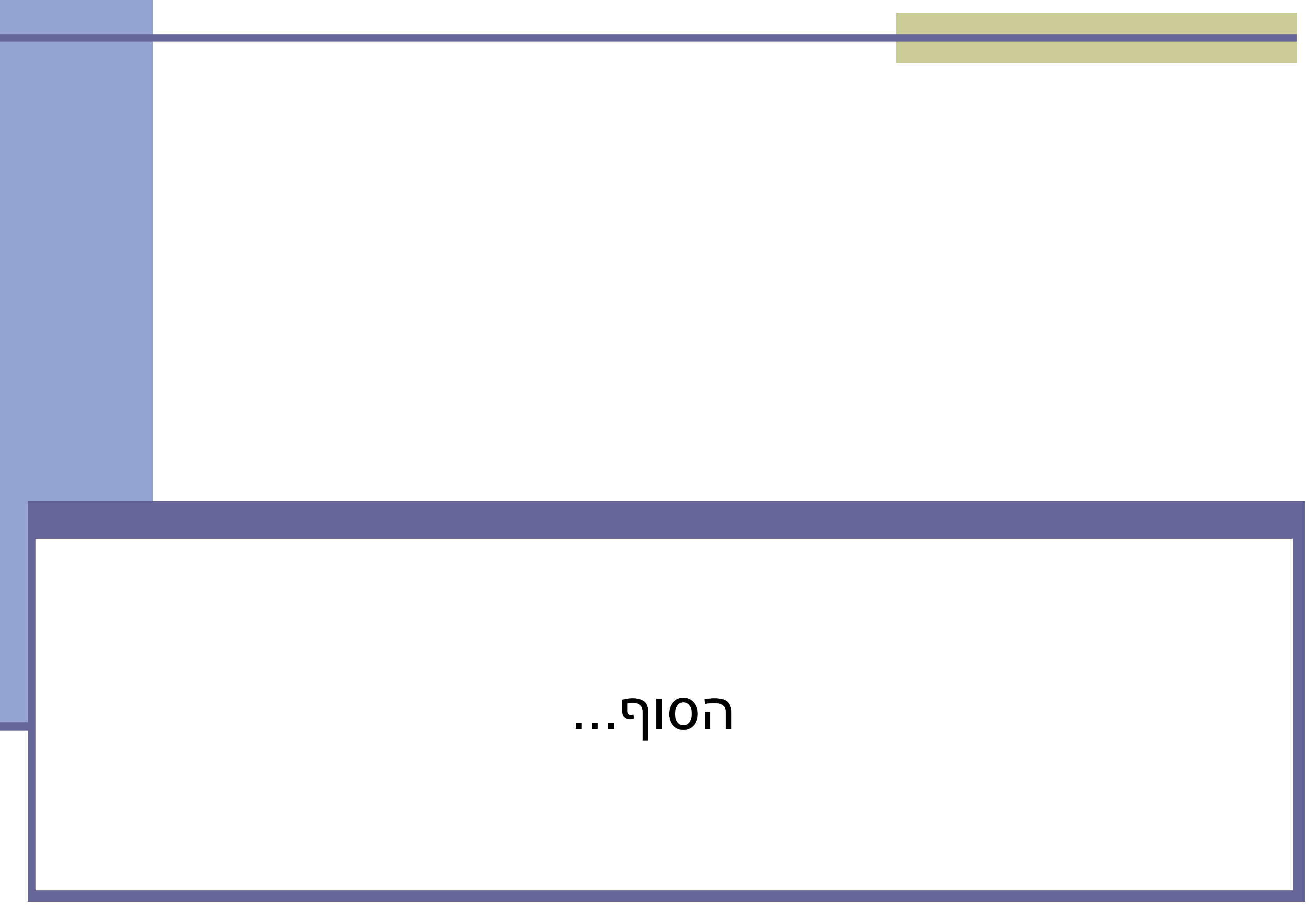
```
String input = " hey 1 hey 2 hey red hey blue ";
Scanner s = new Scanner(input).useDelimiter(" hey ");
while (s.hasNext())
            System.out.println(s.nextLine());
s.close();
Scanner s2 = new Scanner(input).useDelimiter(" hey ");
System.out.println(s2.nextInt());
System.out.println(s2.next());
s2.close();
```

```
 hey 1 hey 2 hey red hey blue
1
2
```

שאלה מבחינה (מועד ב', סמסטר ב', תשע"ח)

<u>שאלה 6</u>:
להלן מספר טענות הקשורות למחלקה java.util.Scanner.
<u>טענה 1</u>: ניתן להשתמש ב Scanner רק במקרים של קריאה מקובץ או במקרים של קריאה מהקלט שהוקלד ע"י המשתמש (System.in).
<u>טענה 2</u>: באמצעות scanner ניתן לקרוא טיפוסים פרימיטיביים כמו int בצורה ישירה, ללא המרה מפורשת ממחרוזת.
<u>טענה 3</u>: הפלט של המתודה nextLine לא תלוי ב delimiter שיכול להיקבע ע"י שימוש במתודה useDelimiter.
בחר/י בתשובה הטובה ביותר.

א. כל הטענות לא נכונות.
ב. רק טענה 1 נכונה.
ג. רק טענה 2 נכונה.
ד. רק טענה 3 נכונה.
ה. רק טענות 1+2 נכונות.
ו. רק טענות 1+3 נכונות.
ז. רק טענות 2+3 נכונות.
ח. כל הטענות נכונות.


הסוף...