

# תרגול מס' 5: IO (קלט-פלט)

---

זרמי קלט וזרמי פלט (Input & Output Streams),  
קוראים וכותבים,  
והשימוש בהם לצורך עבודה עם קבצים

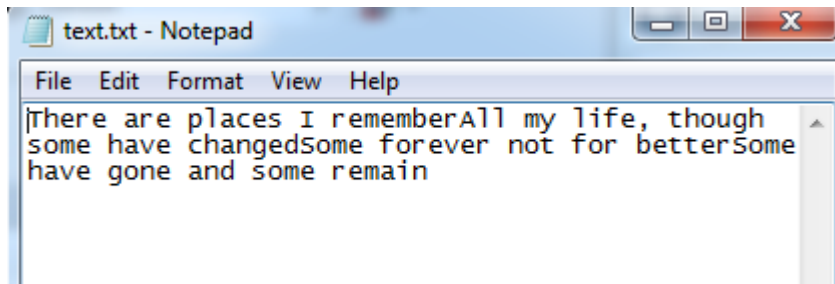
# Input/Output in Java

- פעולות קלט-פלט הן רכיב מרכזי בכל שפת תכנות, והן מאפשרות לתוכנית לקבל (קלט) או לשלוח (פלט) נתונים מ\אל משאב חיצוני.
- אפשרויות **פלט** בהם נתקלנו עד עתה:
  - הדפסה למסך באמצעות זרם הפלט הסטנדרטי `System.out`
- אפשרויות **קלט** בהם נתקלנו עד עתה:
  - קבלת ארגומנטים בשורת הפקודה (ניתנים בעת הרצת התוכנית).
  - קריאת נתונים המוקלדים ע"י המשתמש במסך הקונסול באמצעות זרם הקלט הסטנדרטי `System.in` (תוך שימוש ב-`Scanner`).
  - קריאת נתונים מקובץ טקסט (תוך שימוש ב-`Scanner`).
- היום נלמד מהם למעשה זרמי קלט\פלט, ונכיר מחלקות המאפשרות ביצוע פעולות קלט-פלט ברמות שונות של תחכום.

# המשימה

- במערכות הפעלה שונות נעשה שימוש בתווי בקרה שונים עבור ירידת שורה  
:(newline)
- ב-UNIX/Linux \n – (Line Feed)
- ב-Windows \r\n - (Carriage Return + Line Feed)
- יכולות להתעורר בעיות...

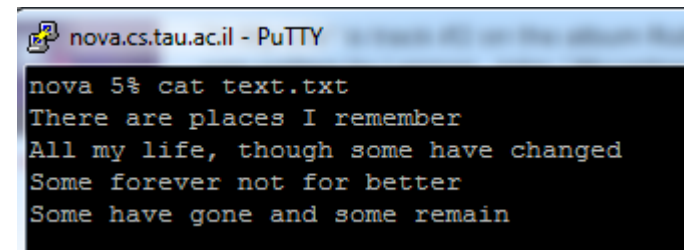
windows:



```
text.txt - Notepad
File Edit Format View Help
There are places I rememberAll my life, though
some have changedSome forever not for betterSome
have gone and some remain
```

דוגמא לקובץ  
linux שנכתב ב  
ונפתח ב  
windows

Linux:



```
nova.cs.tau.ac.il - PuTTY
nova 5% cat text.txt
There are places I remember
All my life, though some have changed
Some forever not for better
Some have gone and some remain
```

- נרצה לכתוב תכנית לתיקון קבצי טקסט
- בדוגמא – תיקון מ-UNIX ל-Windows

# תכנון פתרון

- ארגומנטים: קובץ קלט וקובץ פלט
- קריאה מקובץ הקלט
- כבר ראינו דוגמא עם Scanner,
- היום נראה דרכים אחרות
- החלפת ירידת השורה
- יצירת קובץ הפלט
- כתיבת הפלט
- לא בהכרח בסדר הזה...

## נושאים נוספים

- בתרגול היום נזכיר את שני הנושאים הבאים בקצרה, ונפרט עליהם בשבועות הבאים:
  - היררכיית מחלקות ה-IO ב-Java
  - טיפול בשגיאות זמן ריצה (נפוצות במיוחד בעבודה עם IO)

# קלט ופלט בג'אווה

- משאבי מידע: קבצים, console, רשת, זיכרון, DB, תכניות אחרות...
- התכנית שלנו צריכה לדעת איך לתרגם את הביטים לעצמים \ טיפוסים פרימיטיביים ובחזרה



<http://docs.oracle.com/javase/tutorial/essential/io/index.html>

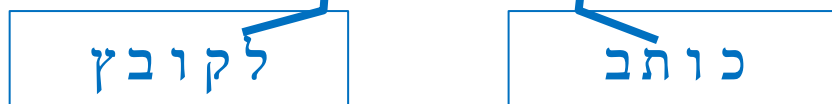
# זרמים (Streams)

- קבוצה של טיפוסים שיודעים לקרוא ולכתוב ממשאבים בצורה סדרתית

- קוראים \ כותבים בתים (bytes)
- הזרימה היא תמיד חד-כיוונית
- Input Streams – לקריאה
- Output Streams – לכתובה

FileOutputStream

- לדוגמא



# שימוש אופייני בזרמי קלט ופלט

- כל הזרמים נפתחים עם יצירתם
- `FileOutputStream` – אפילו יוצר קובץ חדש
- פתיחת זרם יכולה לגרום לזריקת חריג (Exception).
- יש לסגור את הזרמים בגמר השימוש כדי לאפשר שחרור משאבים.
- שימוש סטנדרטי (פסאודו קוד):

## קריאת נתונים מזרם קלט

```
Open input stream
While can read
    read unit
    do something
Close stream
```

## כתיבת נתונים לזרם פלט

```
Open output stream
While has data to write
    write unit
Close stream
```



# דוגמאות לזרמים שימושיים

זרמים	שימוש
FileInputStream FileOutputStream	קריאה\כתיבה של <b>בתים</b> <b>מקבצים</b> (Bytes)
BufferedInputStream BufferedOutputStream	קריאה\כתיבה של <b>בתים</b> תוך שימוש במאגר <b>מובנה</b>
FileReader, FileWriter	קריאה\כתיבה של <b>תווים</b> <b>מקבצים</b> :
DataInputStream DataOutputStream	קריאה\כתיבה של <b>טיפוסים</b> <b>פרימיטיביים ומחרוזות</b> (בדומה ל-Scanner):

<https://docs.oracle.com/javase/tutorial/essential/io/>

# FileInputStream

- שיטות שונות לקריאה:
  - המתודה `read()`:
    - קוראת `byte` אחד בכל פעם.
    - מחזירה ערך מטיפוס `int` שמכיל את ה `byte` הנקרא. במידה והגענו לסוף הזרם, חוזר הערך `-1`.
  - המתודה `read(byte[] b)`:
    - קוראת מספר `byte` כגודל המערך ומכניסה אותם לתוכו.
    - מחזירה ערך מטיפוס `int` אשר מציין את מספר הבתים שנקראו. במידה והגענו לסוף הזרם, חוזר הערך `-1`.

מתודות שימושיות נוספות:

[https://docs.oracle.com/javase/8/docs/api/java/io/FileInputStream.html#read\(\)](https://docs.oracle.com/javase/8/docs/api/java/io/FileInputStream.html#read())

# דוגמא 1 – שימוש ב- File IO Streams

```
public class ByteUnixToWindows {
```

ארגומנט: המסלול לקובץ

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

```
        File fromFile = new File(args[0]);
```

```
        FileInputStream fis = new FileInputStream(fromFile);
```

```
        int readByte;
```

```
        while ((readByte = fis.read()) != -1) {
```

```
            System.out.write(readByte);
```

```
        }
```

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

```
        fis.close();
```

```
    }
```

```
}
```

קוראים byte בכל פעם.  
המתודה read מחזירה את ה byte שהיא קראה, כשמגיעים לסוף הקלט חוזר הערך -1.

```
Problems @ Javadoc Declaration Console x
<terminated> ByteUnixToWindows [Java Application] C:\jav
In Xanadu did Kubla Khan
A stately pleasure-dome decree:
Where Alph, the sacred river, ran
Through caverns measureless to man
Down to a sunless sea.
```

כרגע רק כותבים ל-console,  
לא תיקנו את הבעיה!

# הפתרון לא יעיל!

- נרצה לקרוא הרבה בתים **בבת אחת**
- נוסיף כתיבה לקובץ תוך שימוש ב-`FileOutputStream`
- נקבל כארגומנט שני את המסלול לקובץ הפלט

## דוגמא 2 – מערך בתים

```

public class ByteArrayUnixToWindows {

    public static void main(String[] args) throws IOException {
        File fromFile = new File(args[0]);
        FileInputStream fis = new FileInputStream(fromFile);

        File toFile = new File(args[1]);
        FileOutputStream fos = new FileOutputStream(toFile);

        byte[] readBytes = new byte[1000];
        int numRead;
        while ((numRead = fis.read(readBytes)) != -1) {
            fos.write(readBytes, 0, numRead);
        }

        fis.close();
        fos.close();
    }
}

```

**קן:** כותבים לקובץ  
**עדין לא:** מתקנים את `newline`

**numRead** שווה למס' הבתים שקראנו בפועל, לכן זה גם מס' הבתים שנכתוב

## עבודה עם טקסט

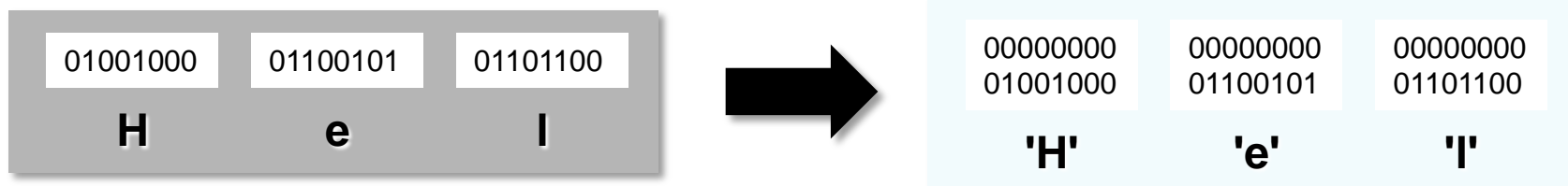
- הקלט והפלט שלנו הם קבצי טקסט
- תיקון newline עם bytes – אפשרי, אבל לא נוח!
- היינו רוצים לעבוד עם **מחרוזות ו-characters**

# Reader & Writer

- מחלקות שקוראות וכותבות רצפים של **characters** ממשאבים.
- לדוגמא: `FileReader`, `FileWriter`
- **בעיה פוטנציאלית:**
- `Characters` בג'אווה הם עם קידוד מסויים (UTF-16)
- מה אם לקבצים שלנו יש קידוד אחר?

## הפתרון...

- בד"כ Java פותרת את הבעיה בעצמה!
- קידוד ברירת מחדל מוגדר עבור מערכת ההפעלה
- Java מתרגמת אותו ל-characters שלה



- לעתים ניתן להגדיר מה הקידוד הדרוש

```
new InputStreamReader(is, Charset.forName("UTF-8"));
```



# דוגמא 3 – Reader & Writer

```

public class CharacterUnixToWindows {

    public static void main(String[] args) throws IOException {
        File fromFile = new File(args[0]);
        FileReader fReader = new FileReader(fromFile);

        File toFile = new File(args[1]);
        FileWriter fWriter = new FileWriter(toFile);

        char[] charRead = new char[1000];
        int numRead;
        while ((numRead = fReader.read(charRead)) != -1) {
            String string = new String(charRead, 0, numRead);
            String windowsString = string.replaceAll("\n", "\r\n");
            fWriter.write(windowsString);
        }

        fReader.close();
        fWriter.close();
    }
}

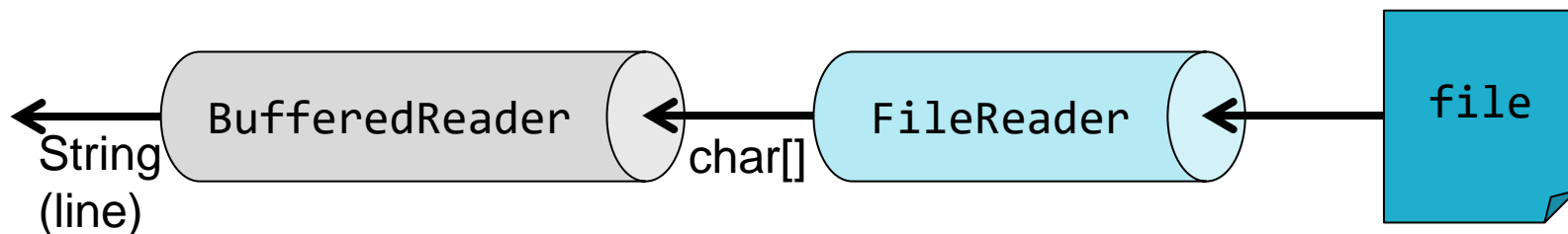
```

הפתרון לא היה עובד  
בהמרה הפוכה  
!(linux ל windows)  
למה?

# Stream Wrappers

- קיימים זרמים אשר "עוטפים" זרמים אחרים ומוסיפים להם פונקציונליות
- לדוגמא, רוצים לקרוא מקובץ (FileReader) אבל שורה בכל פעם (BufferedReader)
- כשניצור את הקורא השני, נעביר לו את הראשון כארגומנט.

```
new BufferedReader(new FileReader(file))
```



# איך זה עובד?

- אנחנו נעבוד עם הזרם העוטף החיצוני ביותר (BufferedReader בדוגמא)
  - נשלח לו מהקוד בקשות קריאה או כתיבה
- כל זרם עוטף מחליט מתי לשלוח בקשת קריאה\כתיבה לזרם הנעטף על-ידו
  - ומבצע עיבוד על המידע לפני שהוא מעביר אותו הלאה
- עלינו רק לדאוג לחבר את הזרמים בצורה נכונה

# Stream Wrappers – דוגמא נוספת

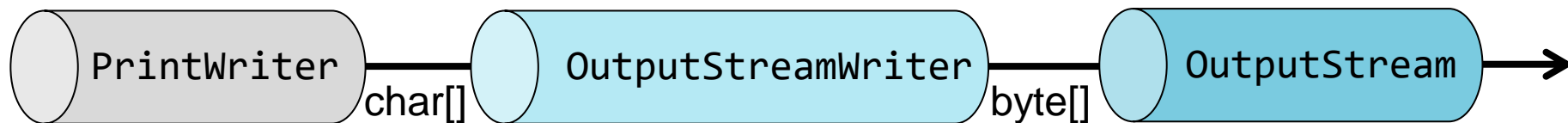
- רוצים להדפיס ל-OutputStream נתון.

- `OutputStreamWriter` מאפשר לנו לעטוף Stream ב-`Writer` (וגם לבחור את הקידוד, כפי שראינו עם `InputStreamReader`)

- `PrintWriter` מאפשר הדפסה בדומה ל-`System.out`

```
new PrintWriter(new OutputStreamWriter(givenOutputStream))
```

תומך במתודות `printf` ו-`format`



# דוגמא 4 – Buffered

```

public class BufferedUnixToWindows {

    public static void main(String[] args) throws IOException {
        File fromFile = new File(args[0]);
        BufferedReader bufferedReader = new BufferedReader(new
            FileReader(fromFile));

        File toFile = new File(args[1]);
        BufferedWriter bufferedWriter = new BufferedWriter(new
            FileWriter(toFile));

        String line;
        while ((line = bufferedReader.readLine()) != null) {
            bufferedWriter.write(line + "\r\n");
        }
        bufferedReader.close();
        bufferedWriter.close();
    }
}

```

כל הזרמים הנעטפים  
close() סוגר גם את

בכתיבה נוסף  
את ירידת  
השורה הרצויה

# סיכום הטכניקות שהוצגו בדוגמאות 1-4

דוגמא	מחלקות IO רלבנטיות	תיאור
1	FileInputStream FileOutputStream	קריאה וכתיבה של <b>בית</b> בודד מקובץ. הבית מוחזר כ-int המאחסן מספר בטווח 0-255 או -1 במקרה של הגעה לסוף הקובץ.
2	FileInputStream FileOutputStream	קריאה של הרבה <b>בתים</b> בבת אחת לתוך מערך מסוג byte[] (שמשמש כמעין מאגר ידני המשפר ביצועים בהשוואה לקריאה של בית בודד בכל פעם). כתיבה של מערך בתים לקובץ.
3	FileReader FileWriter	קריאה של הרבה <b>תווים</b> בבת אחת לתוך מערך מסוג char[]. הקידוד מבתים לתווים נעשה אוטומטית ע"י ה-Reader. המרה ממערך תווים למחרוזת, עיבודה, וכתיבתה לקובץ (המרה ממחרוזת לבתים נעשית ע"י ה-Writer).
4	BufferedReader BufferedWriter	קריאה של תווים תוך שימוש <b>במאגר מובנה</b> המשפר ביצועים. קריאה של <b>שורות</b> טקסט (זיהוי \n או \r\n) מהקובץ כמחרוזות איתן קל לעבוד. כתיבה של <b>מחרוזות</b> תוך שימוש <b>במאגר מובנה</b> .

# בחירת זרם לפי סוג הקובץ

סוג הזרם (כתיבה)	סוג הקובץ
<code>new FileReader(fileName)</code>	קובץ טקסט קצר
<code>new BufferedReader(new FileReader(fileName))</code>	קובץ טקסט ארוך
<code>new FileInputStream(fileName)</code>	קובץ בינארי קצר
<code>new BufferedInputStream(new FileInputStream(fileName))</code>	קובץ בינארי ארוך

סוג הזרם (קריא)	סוג הקובץ
<code>new FileWriter(fileName)</code>	קובץ טקסט קצר
<code>new BufferedWriter(new FileWriter(fileName))</code>	קובץ טקסט ארוך
<code>new FileOutputStream(fileName)</code>	קובץ בינארי קצר
<code>new BufferedOutputStream(new FileOutputStream(fileName))</code>	קובץ בינארי ארוך

# איפה נכנס ה Scanner לתמונה?

- בתרגול שעבר ראינו דוגמת שימוש ב Scanner לקריאה מקבצי טקסט.
- מה עדיף – Scanner או FileReader יחד עם BufferedReader?
  - ל Scanner יש Buffer, אבל הוא קטן יותר מה Buffer של ה-BufferedReader. גודל ה-Buffer הוא רלוונטי כאשר נעבוד עם קבצים גדולים ונרצה לחסוך גישות לדיסק (עדיף 10 קריאות של 4096 בתים מאשר 4096 קריאות של 10 בתים (המספרים הם שרירותיים)!
- Scanner מאפשר פעולות עיבוד מתוחכמות על קובץ הטקסט אותו אנו קוראים, ומפרק את הקלט לטוקנים. בעוד שה-BufferedReader מחזיר מחרוזות בלבד, ה-Scanner יכול לחלץ טיפוסים פרימיטיביים כמו boolean, int וכו'. שימושי כאשר אנחנו רוצים לבצע המרות תוך כדי הקריאה מהקובץ.



# המחלקה `java.nio.file.Files`

<http://docs.oracle.com/javase/8/docs/api/index.html?java/nio/file/Files.html>

- מכילה שירותים שימושיים לעבודה עם קבצים
- עובדת עם עצמים מסוג `java.nio.file.Path` שמתאימים למסלולי קבצים (בדומה ל-`java.io.File`).
- המחלקה המשלימה `java.nio.file.Paths` מכילה שירותים שימושיים עבור מסלולי קבצים.
- `Paths.get("examples", "example.txt")` יחזיר אובייקט מסוג `Path` שמתאים למסלול הקובץ היחסי `examples/example.txt`

# המחלקה Files - דוגמאות

- **copy** – העתקת קבצים
- **delete, move** ובדומה
- **isExecutable, isWritable, isReadable, isDirectory**
- **exists** – מחזירות פרטים שונים לגבי ה-Path
- **readAllBytes** – קריאת כל הקובץ בבת אחת.
- אין צורך לפתוח ולסגור זרמים
- מתאים רק לקבצים קטנים יחסית!

# דוגמא 5 – שימוש במחלקה Files

```
public class FilesUnixToWindows {  
  
    public static void main(String[] args) throws IOException {  
        String fromFile = args[0];  
        String toFile = args[1];  
  
        byte[] allBytes = Files.readAllBytes(Paths.get(fromFile));  
        String string = new String(allBytes);  
        String windowsString = string.replaceAll("\n", "\r\n");  
        Files.write(Paths.get(toFile), windowsString.getBytes());  
    }  
}
```

# לסיכום

- ראינו דרכים שונות לעבודה עם קלט ופלט
  - זרמים, קוראים וכותבים, Files ,Scanner
  - בעיקר עבודה עם קבצים, אבל לא רק!
- נשתמש בהם לפי הצורך
  - האם יש צורך בעוד זרמים?
  - שיקולי יעילות ומודולריות לעומת נוחות

# טבלת זרמים שימושיים – בתים

Output streams		Input streams	
כתיבה לקובץ	<b>FileOutputStream</b>	קריאה מקובץ	<b>FileInputStream</b>
(עוטף) כנ"ל לכתיבה	BufferedOutputStream	(עוטף) קריאה יותר יעילה דרך buffer	BufferedInputStream
(עוטף) כנ"ל לכתיבה	DataOutputStream	(עוטף) קריאת טיפוסים פרימיטיביים	DataInputStream
		(עוטף) מאפשר "החזרה" של חלק מהבתים ל-Stream הנעטף	PushbackInputStream
		עוטף רצף של Streams: קורא מאחד, אח"כ מהשני וכו'	SequenceInputStream

# טבלת זרמים שימושיים - תווים

Writers		Readers	
כתיבה לקובץ	<b>FileWriter</b>	קריאה מקובץ	<b>FileReader</b>
כנ"ל לכתיבה	StringWriter	קריאה ממחרוזת	StringReader
(עוטף) כנ"ל לכתיבה	<b>BufferedWriter</b>	(עוטף) קריאה יותר יעילה דרך <code>buffer</code> , מאפשר קריאת שורה	<b>BufferedReader</b>
כנ"ל לכתיבה	OutputStreamWriter	עוטף <code>Stream</code> . מאפשר בחירת קידוד	InputStreamReader
(עוטף) פעולות הדפסה שונות ( <code>println</code> למשל)	<b>PrintWriter</b>		
		(עוטף) מאפשר לדעת כמה שורות קראנו ע"י <code>getLineNumber</code>	LineNumberReader