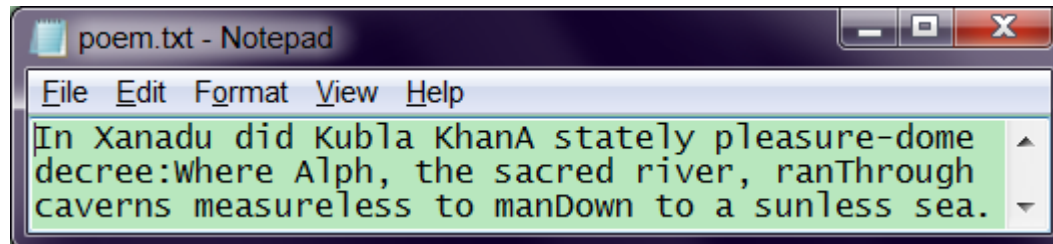


תרגול מס' 5: קלט-פלט

זרמים, קוראים וכותבים,
והשימוש בהם לצורך עבודה עם קבצים

המשימה

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



The screenshot shows a Notepad window titled "poem.txt - Notepad". The menu bar includes File, Edit, Format, View, and Help. The text in the window is:
In Xanadu did Kubla Khan
A stately pleasure-dome
decease:Where Alph, the sacred river,
ran through
caverns measureless to man
Down to a sunless sea.

- 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

IO!

■ יצירת קובץ הפלט

■ כתיבת הפלט

■ לא בהכרח בסדר הזה...

לא נדבר היום (כמעט) על

- היררכיית מחלקות ה IO ב-Java
- טיפול בשגיאות (רק קצת)

קלט ופלט בג'אווה

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



■ Tutorial מומלץ:

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

זרמים (Streams)

- קבוצה של טיפוסים שיודעים לקרוא ולכתוב ממשאבים בצורה סדרתית
 - קוראים \ כותבים **bytes**
 - הזרימה היא תמיד חד-כיוונית
 - Input Streams – לקריאה
 - Output Streams – לכתובה

FileOutputStream לדוגמא

ל ק ו ב ן כ ו ת ב

שימוש בזרמים

כל הזרמים נפתחים עם יצירתם ■

FileOutputStream – אפילו יוצר קובץ חדש ■

יכולה להיות שגיאה ■

שימוש סטנדרטי: ■

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

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

דוגמאות לזרמים שימושיים

■ קריאה\כתיבה לקבצים:

FileInputStream, FileOutputStream

■ BufferedInputStream, BufferedOutputStream

■ קריאה\כתיבה של טיפוסים פרימיטיביים ומחרוזות
(בדומה ל-Scanner):

DataInputStream, DataOutputStream

דוגמא 1 – שימוש ב- File IO Streams

```
public class ByteUnixToWindows {  
  
    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();  
    }  
}
```

ארגומנט: המסלול לקובץ

קוראים byte בכל פעם.
המתודה read מחזירה int כדי
לסמן את סוף הקובץ ב- -1

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

```
Problems @ Javadoc Declaration Console ✕  
<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.
```

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();  
    }  
}
```

עין לא: מתקנים את את newline

numRead יקבל את מס' בתים 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:**

■ `Characters` בג'אווה הם עם קידוד מסויים (UTF-16)

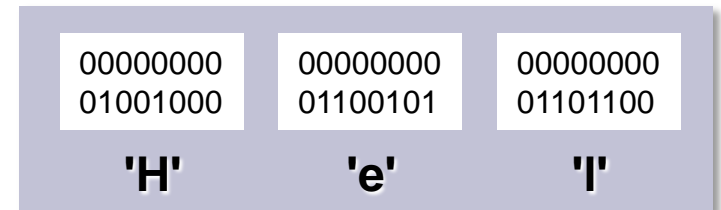
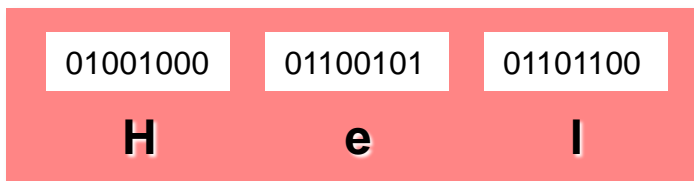
■ אבל בקבצי המחשב שלנו יש אולי קידוד אחר!

והפתרון...

■ 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

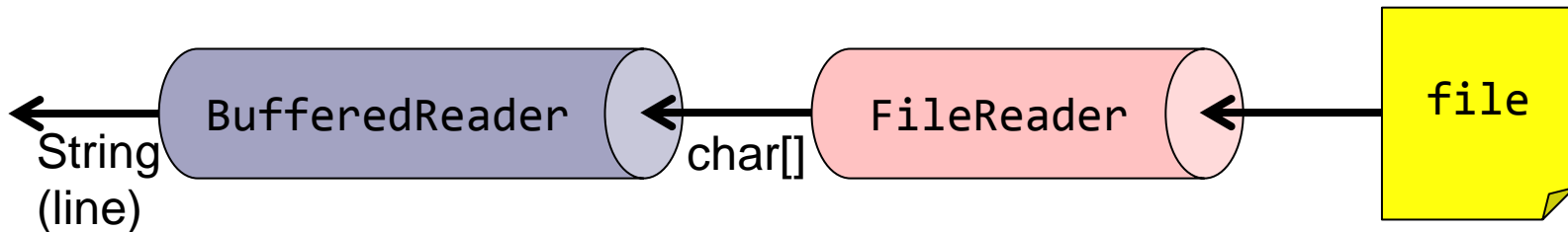
```
public class CharacterUnixToWindows {  
    public static void main(String[] args) throws IOException {  
        File fromFile = new File(args[0]);  
        FileReader fReader = new FileReader(fromFile);  
  
        File toFile = new File(args[1]);  
        FileWriter fWriter = new FileWriter(toFile);  
  
        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
בכיוון ההפוך!
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

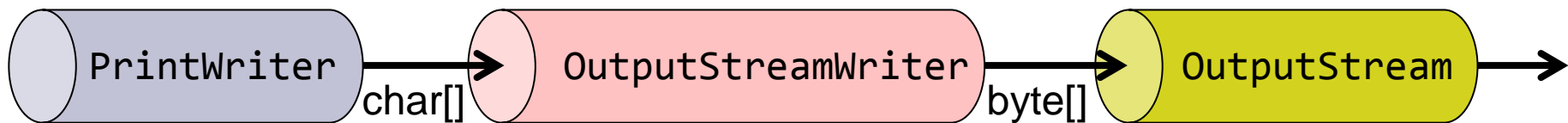
■ רוצים להדפיס ל-OutputStream נתון.

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

■ PrintWriter מאפשר הדפסה בדומה ל-System.out

תומך במתודות printf ו-format

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



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();  
    }  
}
```

readLine() "שובר"
שורות" לפי כל הסוגים
האפשריים של newline

בכתיבה נוסף את
ירידת השורה הרצויה

close() סוגר גם את
כל הזרמים הנעטפים

אבל... אין דרך פשוטה יותר?

■ קריאה וכתובה לקבצים הן פעולות סטנדרטיות

■ דין: אולי צריך `BufferedReader` ו-
`BufferedWriter`?

■ היינו רוצים לקרוא את כל הקובץ בפקודה אחת

■ החל מ-Java SE 7.0 יש דרך לעשות זאת!

המחלקה `java.nio.file.Files`

<http://docs.oracle.com/javase/7/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`

Files - דוגמאות

■ **copy** – העתקת קבצים

■ ובדומה **delete, move**

■ **isWritable, isReadable, isDirectory**

exists, isExecutable – 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 ב-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());  
    }  
}
```

טבלת זרמים שימושיים – בתים

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

טיפול בשגיאות זמן ריצה

ישנן שתי דרכים להתמודדות עם שגיאות זמן ריצה (חריגות) בג'אווה:

1. חילחול מעלה של החריגה אל המתודה הקוראת (זריקת אחריות הלאה)
על המתודה להצהיר על זריקת חריגה בחתימה שלה. לדוגמא:

```
public static String analyzefile(String filename)  
    throws IOException {...}
```

2. שימוש במנגנון try-catch לתפיסת החריגה וטיפול בה.

טיפול בשגיאות זמן ריצה

דוגמא לשימוש ב-try-catch בתוכנית IO טיפוסית

```
public static void main(String[] args){
    BufferedReader reader= null;

    try {
        reader= new BufferedReader(new FileReader("Untitled.txt"));
        String s;
        while((s = reader.readLine()) != null)
            System.out.println(s);
    }
    catch(FileNotFoundException ex) {
        //handle the FileNotFoundException
    }
    catch(IOException ex) {
        //handle the IOException
    }
    finally {
        if (reader!= null)
            reader.close();
    }
}
```

■ 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