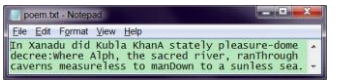
תרגול מס' 5: קלט-פלט

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...



- 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...

IO!

לא נדבר היום (כמעט) על

- טיפול בשגיאות
- היררכיית מחלקות הIO ב-Java

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 – לכתיבה
- לדוגמא

```
FileOutputStream
```

לקובץ כותב

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();
    }
}
```

numRead יקבל את מס' בתים שקראנו בפועל, לכן זה גם מס' הבתים שנכתוב

כן: כותבים לקובץ
עדין לא: מתקנים את newline

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

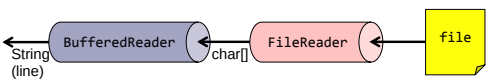
```
public class CharacterUnixToWindows {

    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))
```



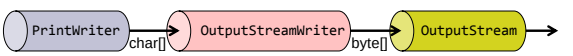
איך זה עובד?

- 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

דוגמא נוספת – Stream Wrappers

- רוצים להדפיס לOutputStream 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))
```



אבל... אין דרך פשוטה יותר?

- קריאה וכתיבה לקבצים הן פעולות סטנדרטיות
  - **דיון**: אולי צריך BufferedFileReader ו-BufferedFileWriter?
- היינו רוצים לקרוא את כל הקובץ בפקודה אחת
- החל מ-Java SE 7.0 יש דרך לעשות זאת!

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 נוסיף את ירידת השורה הרצויה

close() סוגר גם את כל הזרמים הנעטפים

Files - דוגמאות

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

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 למסולי 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

טבלת זרמים שימושיים

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

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 - המשך

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

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