פעולות קלט פלט (I\O Streams)

הקדמה

שפת התכנות Java מספקת מחלקות רבות אשר משמשות לביצוע פעולות קלט ופלט. רוב המחלקות הן מחלקות שהאובייקטים שנוצרים מהן מייצגים streams.

תרגום המילה stream לעברית הוא זרם/נחל, ואכן, האובייקטים שמהווים streams מתארים זרימה של bytes אשר מהווה, למעשה, זרימה של נתונים. ניתן גם לראות ב-stream מקור של bytes, או לחילופין, יעד של bytes, אשר סדר ה-bytes שנשלח אליו או מגיע ממנו בעל משמעות. כך, למשל, תכנית שמקבלת קלט של נתונים מן המקלדת מבצעת זאת באמצעות strean (זרם) של בתים (byte) מהמקלדת. כיוון שקשה לדמיין במחשבותינו אובייקט שמייצג זרימה של נתונים ניתן לראות באובייקטים הללו ייצוג לצינורות שדרכם הנתונים עוברים.

את ה-streams שמיוצגים על ידי המחלקות ששייכות ל-java.io ניתן לחלק לשתי קבוצות:

<u>input streams</u>

אלה הם מקורות שניתן לקלוט/לקרוא מהם בתים (byte). למקורות אלה לא ניתן לכתוב/לשלוח בתים (byte) בחזרה.

<u>output streams</u>

אלה הם יעדים שניתן לשלוח/לכתוב אליהם בתים (byte). מיעדים אלה לא ניתן לקרוא/לקבל בתים (byte) חזרה.

חלוקה נוספת שניתן לבצע לקבוצת ה-streams שמיוצגים באמצעות אובייקטים שיוצרים מהמחלקות שייכות ל-package java.io : היא לשתי הקבוצות הבאות:

node streams

אלה הם מקורות/יעדים של bytes שמקושרים למקור מסוים כגון קובץ בדיסק, שטח זיכרון, חיבור תקשורת…

filter streams

אלה הם מקורות/יעדים של בתים שמקושרים ל input streams/output streams קיימים אחרים. בדרך זו, כאשר מנסים לקרוא מהם נתונים, הנתונים מגיעים, למעשה, מ-input stream אחר, וכאשר מנסים לכתוב אליהם נתונים, הנתונים נשלחים, למעשה, אל output stream אחר.

כפי שתיווכח/י בהמשך, נושא ה-I\O Streams טופל ב-Java על ידי יצירת מחלקות,כמעט לכל סוג של stream. מסיבה זו, קיים ב-Java ג'ונגל של streams מסוגים שונים. ההכרות עם סוגי המחלקות השונות בפרק זה תיעשה באמצעות המחלקות העיקריות והשימושיות יותר.

היררכית מחלקות ה-I/O ב Java

ב-Java קיימות מחלקות רבות שמשמשות בתור streams , גם לצורך ביצוע פעולות לקלט וגם לצורך ביצוע פעולות פלט. יש המתארים מחלקות אלה כג'ונגל אחד גדול. בתחילה, לפני שמכירים את המחלקות, הן אכן נותנות את הרגשת ה-ג'ונגל. בחלק זה, אנסה להקנות מעט מושג על היררכית מחלקות ה-streams השונות. בתחילה אציג את שתי ההיררכיות הבולטות ביותר, ובהמשך אציג היררכיות נוספות.

InputStream

+ FileInputSteram
+ PipeInputStream
+ StringBufferInputStream
+ ByteArrayInputSteam
+ SequencedInputStream
+ ObjectInputStream
+ SequenceInputStream
+ FilterInputStream
    + DataInputStream
    + BufferedInputStream
    + LineNumberInputStream
    + PushBackInputStream

OutputStream

+ FileOutputStream

+ PipedOutputStream

+ ByteArrayOutputStream

+ ObjectOutputStream

+ FilterOutputStream

+ DataOutputStream

+ BufferdOutputStream

+ PrintStream

המחלקות המופיעות בהיררכיות שמתוארות למעלה הן המחלקות העיקריות. מחלקות נוספות מתווספות מעת לעת בגירסאות החדשות של השפה. קיימות היררכיות נוספות של מחלקות. הבנה של אופן פעולת המחלקות בשתי היררכיות אלה מהווה בסיס להבנת המחלקות האחרות (לרבות ההיררכיות האחרות) שאינן מוזכרים בספר זה.

המחלקות האבסטרקטיות InputStream ו- OutputStream

המחלקה האבסטרקטית InputStream מהווה את מחלקת האב בהיררכיית המחלקות שמייצגות inputstream של bytes. במלים אחרות, מחלקה זו משמשת כמחלקת הבסיס לכל המחלקות שמייצגות זרימה של bytes אל תוך התכנית (קלט). מחלקה זו יורשת ישירות מהמחלקה Object.

Object

|

InputStream

המתודות העיקריות שמוגדרות במחלקה זו הן:

```
public abstract int read() throws IOException
```

מתודה זו קוראת את ה-byte הבא בתור מה-inputstream. הערך שהיא מחזירה הוא ערך מטיפוס int בטווח 0 עד 255 שמייצג את ערך ה-byte שנקרא. אם אין עוד אף byte שניתן לקרוא (תהליך הקריאה הגיע אל סוף ה-stream) אז מוחזר הערך: 1-. אם אין נתון שניתן לקריאה ותהליך הקריאה עדיין לא הגיע אל סוף ה-stream אז המתודה תחכה עד להופעתו, אלא אם כן ייזרק exception.

```
public int read(byte[] b) throws IOException
```

מתודה זו קוראת מערך של bytes מה- input stream אל תוך מערך ה-bytes, שהיא מקבלת כארגומנט. המתודה מחזירה את מספר ה-bytes שנקראו או –1 במידה שתהליך הקריאה הגיע אל סוף ה-stream. אם אין נתונים שניתן

לקריאה ותהליך הקריאה עדיין לא הגיע אל סופו של ה-stream אז המתודה תמתין להופעתם. אם ה-stream הגיע לסוף הקובץ אז המתודה תפסיק את פעולתה ותחזיר –1 . אם ייזרק מתוכה exception אז היא תיפסק. מספר הבתים שבפועל נקראים לא שווה בהכרח ל- b.length (יוסבר בהסבר של המתודה הבאה). מתודה זו פועלת באמצעות הפעלת המתודה:

```
public int read(byte[] b, int off, int len) throws IOEception
```

באופן הבא:

```
read(b, 0, b.length)
```

```
public int read(byte[] b, int off, int len) throws IOEception
```

מתודה זו קוראת bytes כדי למלא מערך מטיפוס byte. מספר ה-bytes שהמתודה מתעתדת לקרוא הוא len. ה-bytes שנקראים מוכנסים למערך החל מתא מספר off. מספר ה-bytes שנקראים בפועל מוחזר על ידי המתודה. אם ה-stream הגיע לסוף הקובץ אז המתודה מחזירה –1. מספר הבתים שנקראו בפועל לא חייב להיות זהה ל-len, אך הוא לא יכול להיות גדול ממנו. גם מתודה זו תמתין בפעולתה עד להגעתם של נתונים שניתנים לקריאה אלא אם ייזרק לפני כן exception או אם היא תגיע לסוף הקובץ (במקרה כזה תחזיר –1). מתודה זו פועלת באמצעות הפעלה פשוטה של המתודה read. במחלקות שמוגדרות כמחלקות שיורשות ממחלקה זו מומלץ, מטעמי יעילות, להגדיר מחדש את המתודה הזו (לעשות overriding).

```
public void close() throws IOException
```

מתודה זו משמשת לסגירת ה-input streams, ולשחרור כל המשאבים ששימשו אותו. במחלקה InputStream מתודה זו חסרת כל תוכן, והיא לא עושה שום דבר. במחלקות שיורשות ממנה יש לכתוב אותה מחדש (override) כדי לטפל בשחרור/ניקוי המשאבים שמשמשים את ה-stream.

```
public int available() throws IOException
```

מתודה זו מחזירה את מספר הבתים שניתנים לקריאה באופן מיידי מה-stream מבלי שיהיה צריך לחכות. המספר המוחזר הוא גם מספר הבתים המקסימלי שעל קריאתם ניתן לדלג באמצעות הפעלת המתודה skip. המספר המוחזר הוא גם המספר שאם מספר הבתים שינסו לקרוא יהיה גדול ממנו אז המתודה תיכנס למצב של המתנה. בהגדרתה של המתודה הזו במחלקה InputStream היא מחזירה תמיד 0.

```
public long skip(long n) throws IOException
```

מתודה זו מקבלת את מספר הבתים שהקריאה הבאה מה-stream תדלג עליהם. המתודה מחזירה את מספר הבתים שעליהם הקריאה הבאה תדלג בפועל. מספר הבתים שעליהם הקריאה הבאה תדלג בפועל יכול להיות שונה מ-n בגלל מספר סיבות. העיקרית שבהן היא התקרבותו של הראש הקורא לסוף הקובץ. במקרה כזה, בהחלט ייתכן שמספר הבתים שנותרו לקריאה יהיה נמוך מ- n. מתודה זו, בגרסתה במחלקה InputStream, מפעילה את המתודה read בגירסה:

```
read(byte[] b)
```

ה-bytes נקראים אל מערך מטיפוס byte, ובכך מבוצעת פעולת הדילוג. מומלץ להגדיר מתודה זו מחדש במחלקות שיורשות ממחלקה זו.

```
public boolean markSupported()
```

מתודה זו מחזירה true אם ה-stream שעליו היא הופעלה תומך במתודות mark ו- reset ובכך תומך, למעשה, באפשרות לבצע את התהליך mark&reset. ביישומה של מתודה זו במחלקה InputStream מוחזר הערך false בכל מקרה. במחלקה שיורשת מ-InputStream יש לממש מתודה זו (יש לעשות overriding).

```
public void mark(int readlimit)
```

מתודה זו משמשת לסימון המיקום הנוכחי של ראש הקריאה מה-stream כך שניתן יהיה לחזור אליו באמצעות הפעלת

המתודה reset. הפרמטר readlimit הוא המספר המקסימלי של bytes שיוכלו להיקרא לפני שהסימון יפוג. במחלקה InputStream מתודה זו חסרת כל תוכן, ועל כן, במחלקות שיורשות מ-InputStream ושרוצים שהן יתמכו ב-mark&reset יש לכתוב אותה מחדש (overriding) כדי לצקת בה תוכן, ולאפשר בכך את ביצוען של פעולות ה-mark&reset.

```
public void reset() throws IOException
```

מתודה זו מחזירה את מיקום הראש הקורא ב-stream למיקום שסומן באמצעות המתודה mark, אשר הופעלה קודם לכן. מתודה זו נכתבה במחלקה זו כך שבאופן אוטומטי עם הפעלתה ניזרק IOException, כיוון שבברירת המחדל אפשרות ה- mark & reset לא נתמכת. במחלקות שיורשות ממחלקה זו ושרוצים שהן יתמכו בתהליך ה-mark&reset יש לכתוב את המתודה הזו מחדש (overriding) וזאת כדי לאפשר את ביצוע התהליך.

המחלקה האבסטרקטית OutputStream משמשת כמחלקת האב של כל המחלקות שמייצגות outputstream של bytes, ובמלים אחרות, מחלקה זו משמשת כמחלקת הבסיס לכל המחלקות שמייצגות זרימה של bytes אשר מהווה output. המחלקה יורשת ישירות מהמחלקה Object.

Object

|

OutputStream

המתודות העיקריות שמוגדרות במחלקה זו הן:

```
public abstract void write(int b) throws IOException
```

מתודה לכתיבת ערך מטיפוס byte אל ה-stream. הערך שנשלח אל המתודה יהיה מטיפוס byte למרות שהוא ייקלט אל תוך פרמטר מטיפוס int. מתוך הערך מטיפוס int שמיוצג על ידי הפרמטר b ייכתב אך ורק הערך שמיוצג על ידי 8 הביטים הראשונים. יתר 24 הביטים לא ייכתבו (המתודה תתעלם מהם). במחלקות שיורשות ממחלקה זה חייבים להגדיר את המתודה (זוהי מתודה אבסטרקטית).

```
public void write(byte []b) throws IOException
```

מתודה לכתיבת מערך של bytes אל ה-stream. המתודה פועלת באמצעות הפעלת המתודה:

```
public void write(byte []b, int off, int len)
```

באופן הבא:

```
write(b,0,b.length)
```

```
public void write(byte b[], int off, int len) throws IOException
```

מתודה שמשמשת לכתיבת מערך של bytes אל ה-stream החל מה-byte שבמיקום off במערך. המתודה כותבת len יחידות של bytes.

```
public void close() throws IOException
```

מתודה זו משמשת לסגירת ה-output stream ולשחרור כל המשאבים אשר קשורים בו. במחלקה OutputStream מתודה זו ריקה מתוכן, ועל כן יש לכתוב אותה מחדש (overriding) במחלקות שיורשות מהמחלקה OutputStream.

```
public void flush() throws IOException
```

מתודה שמשמשת לכתיבת כל ה-bytes אשר אוחסנו ב-buffer באופן זמני אל ה-stream. קיימים streams שעושים שימוש ב-buffer לשם כתיבה של נתונים… במחלקה OutputStream מתודה זו ריקה מתוכן, ולכן יש לכתוב אותה מחדש (overriding) במחלקות שיורשות מהמחלקה OutputStream, ושעושות שימוש ב-buffer.

המחלקות FileInputStream ו- FileOutputStream

שתי המחלקות האבסטרקטיות שנסקרו קודם לכן, InputStream ו- OutputStream משמשות בסיס להגדרתן של מחלקות רבות אחרות אשר משמשות לביצוען של מגוון פעולות קלט ופלט. בסקירת מחלקות ה-I\O שהובאה בתחילת הפרק וודאי נוכחת לראות כי קיימות מחלקות שימושיות רבות. כיוון שאין אפשרות מעשית לסקור ולבדוק את כולן בפרק זה, בחרתי להתמקד בחשובות שבהן, ולהתחיל בחלק זה בסקירתן של המחלקות FileOutputStream ו-FileInputStream.

כל אחת משתי מחלקות אלה מתארת node stream אשר מקושר למקור/יעד של בתים אשר נחשב לקובץ. אובייקטים מטיפוס מחלקות אלה עושים שימוש בקבצים כמקור/יעד של bytes. בין ה-constructors של שתי המחלקות האלה ניתן גם למצוא constructors שמאפשרים לציין את שם הקובץ)בצירוף ה-path שלו(אשר אליו ה-stream, שמייצג האובייקט החדש שנוצר, יקושר.

המחלקה FileInputStream משמשת ליצירת אובייקטים אשר מתארים stream שדרכם זורמים נתונים מקובץ אל תוך התכנית. ה-constructors העיקריים במחלקה הם:

```
public FileInputStream (String name)
     throws FileNotFoundException, SecurityException
```

constructor שמשמש ליצירת אובייקט חדש מטיפוס FileInputStream אשר יקושר לקובץ ששמו נשלח כ-String אל ה-constructor. הקובץ חייב להיות קיים, וניתן לקריאה. הקשר לקובץ מיוצג על ידי אובייקט חדש שנוצר מהמחלקה FileDescriptor. אם קיים security manager אז מופעלת עליו המתודה checkRead ובתור ארגומנט נשלח אליה ה-reference לאובייקט מסוג String אשר מתאר את שם הקובץ שעימו עומד להיווצר הקשר. המתודה checkRead עלולה לזרוק SecurityException בהתאם. אם הקובץ ששמו name קיים, או שזהו שמה של ספריה ולא של קובץ או

שמסיבות כלשהן לא ניתן לפתוח אותו לצורך קריאה אז ייזרק FileNotFoundException.

```
public FileInputStream(File file)
      throws FileNotFoundException, SecurityException
```

constructor שמשמש ליצירת אובייקט חדש מטיפוס FileInputStream אשר יקושר לקובץ שמתואר על ידי אובייקט ה-File , שה-reference שלו נשלח אל constructor זה. תיאור הקשר לקובץ ייעשה באמצעות אובייקט חדש מטיפוס FileDescriptor. אם קיים security manager אז תופעל עליו המתודה checkRead ומה שיישלח אליה הוא ה-path שמתאר את הקובץ (יילקח מתוך אובייקט ה-File שכעת ה-reference שלו מוחזק בתוך אובייקט ה-FileInputStream החדש שנוצר). המתודה checkRead תזרוק SecurityException אם הגישה לקובץ אסורה. אם הקובץ לא קיים או שאובייקט ה-File מתאר ספריה ולא קובץ או שהקובץ לא ניתן לקריאה אז ייזרק FileNotFoundException.

```
public FileInputStream(FileDescriptor fdObj)
```

constructor ליצירת אובייקט חדש מטיפוס המחלקה FileInputStream אשר יקושר לקובץ שמתואר על ידי אובייקט מסוג FileDescriptor. אם קיים security manager אז תופעל עליו המתודה checkRead ומה שיישלח אליה הוא ה-reference לאובייקט ה-FileDescriptor שקיבל ה-constructor , ובדומה למתודות הקודמות , SecurityException ייזרק בהתאם.

ה-methods העיקריים שמוגדרים במחלקה הם:

```
public int read() throws IOException
```

מתודה שקוראת byte אחד מה- input stream שמייצג האובייקט. אם אין אף מידע זמין שניתן לקריאה אז המתודה מחכה עד שפעולת הקריאה מתאפשרת. המתודה מחזירה את מה שהיא קראה. אם המתודה הגיעה אל סוף הקובץ אז היא מחזירה 1-. מתודה זו תופסת את מקומה של המתודה read שהגיעה בהורשה מהמחלקה InputStream.

```
public int read(byte []b) throws IOException
```

מתודה זו קוראת bytes מה-input stream ישירות אל תוך המערך b. מספר ה-bytes שהיא קוראת איננו גדול מ-b.length. המתודה מחזירה את מספר הבתים שנקראו או 1- אם אין יותר מידע שניתן לקרוא כיוון שהיא הגיעה אל סוף הקובץ. מתודה זו באה במקומה של מתודה שהגיעה בהורשה מהמחלקה InputStream.

```
public int read(byte []b, int off, int len) throws IOException
```

מתודה זו קוראת עד ל- len בתים)bytes(של נתונים מה-inupt stream היישר אל תוך המערך שהיא מקבלת. אם המתודה נתקלת במצב שבו אין מידע זמין שניתן לקרוא אז היא ממתינה ומחכה עד אשר מופיעים נתונים שזמינים לקריאה. ה-bytes נכתבים אל המערך החל מתא מספר off. המתודה מחזירה את מספר הבתים הכולל שנקראו אל תוך המערך או 1- אם אין יותר מידע שניתן לקרוא כיוון שהיא הגיעה אל סוף הקובץ. מתודה זו באה במקומה של מתודה שהגיעה בהורשה.

```
public long skip(long n) throws IOException
```

מתודה זו משמשת לביצועו של דילוג בקריאת הנתונים מה-input stream. המתודה מקבלת את מספר הבתים שעליהם עליה לדלג, והיא מחזירה את מספר הבתים שעליהם היא דילגה בפועל. מספר הבתים שעליהם היא דילגה לא תמיד זהה למספר הבתים שעליהם היא התבקשה לדלג. לעתים הוא קטן יותר, ולעתים הוא אפילו שווה ל-0. מתודה זו באה במקומה של מתודה בשם זהה שהועברה בהורשה מהמחלקה InputStream.

```
public int available() throws IOException
```

מתודה זו מחזירה את מספר הבתים שניתן לקרוא מבלי לגרום לאחת ממתודות ה- read להיחסם. כל אחת ממתודות ה-read עלולה להיחסם אם אין נתונים שזמינים לקריאה באופן מיידי. במקרים כאלה כל אחת ממתודות ה-read פשוט נחסמת וממתינה עד להגעתם של נתונים זמינים לקריאה. מתודה זו באה במקומה של מתודה שהגיעה בהורשה

מהמחלקה InputStream.

```
public void close() throws IOException
```

מתודה לסגירת ה-stream שמייצג האובייקט, ולשחרור כל משאבי המערכת שבהם הוא השתמש. מתודה זו באה במקומה של מתודה שהגיעה בהורשה מהמחלקה InputStream.

```
public final FileDescriptor getFD() throws IOException
```

מתודה זו מחזירה את ה-reference לאובייקט ה-FileDescriptor אשר מתאר את הקובץ שמחובר לאובייקט.

```
protected void finalize() throws IOException
```

מתודה שבביצועה מובטח כי המתודה close תופעל כאשר אף משתנה לא יחזיק reference לאובייקט. מתודה זו באה במקומה של מתודה שמגיעה בהורשה מהמחלקה Object. לא מומלץ לסמוך על פעולתה של מתודה זו כיוון שהיא לא תמיד מופעלת.

המחלקה FileOutputStream משמשת לתיאור streams של bytes מהתכנית החוצה לקובץ. ה-constructors העיקריים שקיימים במחלקה:

```
public FileOutputStream(String name) throws FileNotFoundException
```

constructor ליצירת אובייקט FileOutputStream חדש שמקושר לקובץ ששמו name. הקשר לקובץ מיוצג באמצעות אובייקט חדש שנוצר מהמחלקה FileDescriptor. אם קיים security manager אז המתודה checkWrite מופעלת עליו, וזאת כאשר מה שנשלח אליה הוא שם הקובץ. המתודה checkWrite זורקת, אם יש צורך, SecurityException. אם השם שניתן הוא שם של ספריה ולא שם של קובץ, או שהקובץ לא קיים וגם לא ניתן לייצור אותו, או שהקובץ לא ניתן לפתיחה מכל סיבה שהיא אז נזרק FileNotFoundException.

```
public FileOutputStream(String name, boolean append)
      throws FileNotFoundException
```

constructor ליצירת אובייקט FileOutputStream חדש שמקושר לקובץ ששמו name. אם הארגומנט השני שנשלח אל ה-constructor הוא true אז ה-bytes ייכתבו אל סופו של הקובץ, ולא מתחילתו תוך שהם דורסים את ה-bytes הקיימים. הקשר לקובץ מיוצג באמצעות אובייקט חדש שנוצר מהמחלקה FileDescriptor. אם קיים security manager אז המתודה checkWrite מופעלת עליו, וזאת כאשר מה שנשלח אליה הוא שם הקובץ, name. המתודה checkWrite זורקת, אם יש צורך, SecurityException. אם השם שניתן הוא שם של ספריה ולא שם של קובץ, או שהקובץ לא קיים וגם לא ניתן לייצור אותו, או שהקובץ לא ניתן לפתיחה מכל סיבה שהיא אז נזרק FileNotFoundException.

```
public FileOutputStream (File file) throws IOException
```

constructor זה יוצר אובייקט FileOutputStream חדש אשר מקושר לקובץ שמיוצג על ידי אובייקט ה-File, אשר ה-reference שלו נשלח אל ה-constructor. הקשר לקובץ מיוצג על ידי אובייקט חדש שנוצר מהמחלקה FileDescriptor.

אם קיים security manager אז המתודה checkWrite מופעלת עליו, וזאת כאשר מה שנשלח אליה הוא ה-path שמייצג אובייקט ה-File. המתודה checkWrite זורקת, אם יש צורך, SecurityException. אם אובייקט ה-File שנשלח מייצג ספריה ולא קובץ, או שהקובץ לא קיים וגם לא ניתן לייצור אותו, או שהקובץ לא ניתן לפתיחה מכל סיבה שהיא אז נזרק FileNotFoundException.

```
public FileOutputStream(FileDescriptor fdObj)
```

constructor זה יוצר אובייקט FileOutputStream חדש אשר מקושר לקובץ שמיוצג על ידי אובייקט ה-FileDescriptor. אם קיים security manager אז המתודה checkWrite מופעלת עליו, וזאת כאשר מה שנשלח אליה הוא ה-reference לאובייקט ה-FileDescriptor (שנשלח אל ה-constructor). המתודה checkWrite זורקת, אם יש צורך, SecurityException. כדאי לשים לב לכך שכיוון שה constructor קיבל reference לאובייקט מטיפוס FileDescriptor אשר מתאר קשר לקובץ המשמעות היא שקיים קובץ שניתן לגשת אליו ולכתוב אליו. אם קשר לקובץ כזה לא היה קיים אז לא ניתן היה לייצור את אובייקט ה-FileDescriptor. זוהי הסיבה לכך ש constructor זה לא זורק FileNotFoundException.

המתודות העיקריות שמוגדרים במחלקה:

```
public void write (int b) throws IOException
```

מתודה זו כותבת את ה-byte שערכו b אל הקובץ שמקושר לאובייקט. מתודה זו באה במקום מתודה שהגיעה בהורשה מהמחלקה OutputStream.

```
public void write (byte[] b) throws IOException
```

מתודה זו כותבת b.length בתים (bytes) ממערך b אל הקובץ שמקושר לאובייקט. מתודה זו באה במקום מתודה שהגיעה בהורשה מהמחלקה OutputStream.

```
public void write (byte[] b, int off, int len) throws IOException
```

מתודה זו כותבת len בתים (bytes) מהמערך b החל ממיקום off אל הקובץ שמקושר לאובייקט. מתודה זו באה במקום מתודה שהגיעה בהורשה מהמחלקה OutputStream.

```
public void close() throws IOException
```

מתודה זו סוגרת את ה-FileOutputStream, ומשחררת את כל משאבי המערכת שבהם האובייקט השתמש. לאחר שמתודה זו הופעלה, לא ניתן עוד להשתמש באובייקט לכתיבת bytes לקובץ. מתודה זו באה במקומה של מתודה שהגיעה בהורשה מהמחלקה OutputStream.

```
public final FileDescriptor getFD() throws IOException
```

מתודה זו מחזירה את ה-reference לאובייקט ה-FileDescriptor אשר מתאר את הקובץ שמחובר לאובייקט.

```
protected void finalize() throws IOException
```

מתודה שבביצועה מובטח כי המתודה close תופעל. זאת יקרה כאשר אף משתנה לא יחזיק reference לאובייקט.
מתודה זו באה במקומה של מתודה שמגיעה בהורשה מהמחלקה Object.

```
import java.io.*;

public class FilesCopy
{
	public static void main(String[] args)
	{
		if (args.length != 2)
		{
			System.err.println("You should pass two filenames");
		}
		else
		{
			try
			{
				copy(args[0], args[1]);
			}
			catch (IOException e)
			{
				System.err.println(e);
			}
		}
	}

	public static void copy(String inputFile, String outputFile)
		throws IOException
	{

		FileInputStream input = null;
		FileOutputStream output = null;

		try
		{
			input = new FileInputStream(inputFile);
			output = new FileOutputStream(outputFile);
```

```
                IOUtils.streamCopy(input, output);
            }
            catch(IOException e)
            {
                e.printStackTrace();
            }
            finally
            {
                if(input!=null)
                {
                    try
                    {
                        input.close();
                    }
                    catch(IOException e)
                    {
                        e.printStackTrace();
                    }
                }
                if(output!=null)
                {
                    try
                    {
                        output.close();
                    }
                    catch(IOException e)
                    {
                        e.printStackTrace();
                    }
                }
            }
        }
}
```

```
import java.io.*;

public class IOUtils
{
      public static void streamCopy(
                        InputStream in, OutputStream out) throws IOException
      {
            synchronized(out)
            {
                  synchronized(in)
                  {
                        byte vec[] = new byte[256];
                        int numOfBytes = in.read(vec);
                        while(numOfBytes!=-1)
                        {
                              out.write(vec,0,numOfBytes);
                              numOfBytes = in.read(vec);
                        }
                  }
            }
      }
}
```

המחלקות ObjectInputStream ו- ObjectOutputStream

שתי מחלקות אלה נחשבות ל-filter streams. בתיאור הכללי שלהן, ניתן לומר שהן משמשות לכתיבת/קריאת אובייקטים (לרבות אלה שקשורים עימם – יוסבר בהמשך) וגם לכתיבת/ קריאת טיפוסים בסיסיים (פרימיטיביים) . כיוון ששתי המחלקות האלה מהוות filter streams הנתונים שנכתבים אליהם מועברים הלאה ל- stream אחר (לדוגמא: stream שיחובר לקובץ מסוים). כדי שניתן יהיה לכתוב אובייקט ל-ObjectOutputStream על המחלקה שלו (או אחת המחלקות שמעליו בהיררכית ההורשה) ליישם את Serializable. האובייקטים שנכתבים ל-ObjectOutputStream חייבים להיקרא בסדר שזהה לסדר כתיבתם.

בכתיבת אובייקט ל-ObjectOutputStream נכתבים גם כל האובייקטים האחרים אשר reference של כל אחד מהם מוחזק באובייקט שנכתב, ובלבד שאותו reference אינו מוחזק במשתנה static או במשתנה שמסומן במילה השמורה transient.

תנאי נוסף לכתיבתם של האובייקטים האחרים הוא שמחלקותיהם יישמו את Serializable. אם יעשה ניסיון לכתוב אובייקט שאחד ממשתניו מכיל reference לאובייקט שבמחלקתו לא מיושם Serializable אז תיזרק השגיאה NonSerializalizableException אלא אם אותו משתנה שמכיל את ה-reference יסומן במילה transient. בכתיבתו של אובייקט כלשהו לאובייקט מטיפוס ObjectOutputStream רק הערכים שבמשתני ה-instance variable (המשתנים שמופיעים בכל אובייקט) נכתבים. הערכים שבמשתני ה- static variabe (או בשם האחר: class variable) לא נכתבים. כמו כן, גם המתודות – אף אחת מהן לא נכתבת.

persistence – זהו המינוח המקצועי שמשמש לתיאור שמירתו של אובייקט למיקום קבוע בקובץ או בזיכרון במחשב הנוכחי או במחשב אחר. ניתן לומר על אובייקט כי הוא persistent capable כאשר ניתן לשמור אותו בקובץ או כאשר

ניתן לשלוח אותו לשמירה בזיכרון/בקובץ על ידי מחשב אחר. במלים אחרות ניתן לומר כי משמעות המונח persistence היא היכולת להעביר אובייקט דרך stream, כאשר אין זה משנה אם התוצאה הסופית היא שהאובייקט מועבר אל קובץ או אל הזיכרון של מחשב אחר.

ב-Serializable אין אף מתודה, והוא משמש לצורך סימון בלבד (אם אינך זוכר למה הכוונה כדאי שתחזור על הפרק שדו בנושא ההורשה). מחלקה שמיישמת interface זה – זאת יהווה אינדיקציה לכך שניתן לשמור אובייקטים מטיפוסה לקובץ/לזיכרון או לחילופין שניתן לשלוח אותם דרך חיבור תקשורת למחשב אחר.

המחלקה ObjectOutputStream כוללת את ה-constructors העיקריים הבאים:

```
public ObjectOutputStream(OutputStream out) throws IOException
```

constructor זה יוצר אובייקט חדש מטיפוס ObjectOutputStream אשר דרכו ניתן לכתוב אל ה-output stream שה-reference אליו נקלט בתוך הפרמטר out.

```
public ObjectOutputStream() throws IOException, SecurityException
```

constructor זה מאפשר להגדיר מחלקה חדשה שיורשת מהמחלקה ObjectOutputStream. כיוון ש-constructor זה לא מקבל ערכים בעת הפעלתו הוא מאפשר, למעשה, להגדיר מחלקה חדשה מבלי לאלץ את השימוש במשתנים שמוגדרים במחלקה ObjectOutputStream.

המתודות העיקריות שמוגדרות במחלקה:

```
public final void writeObject(Object obj) throws IOException
```

מתודה זו משמשת לכתיבת אובייקט מסוים לאובייקט ה-ObjectOutputStream שעליו היא הופעלה. יחד עם האובייקט שכותבים נכתבים גם כל האובייקטים האחרים ש-references אליהם נמצאים באובייקט שכותבים (ובלבד שהם לא מאוחסנים במשתנים מסוג static או transient). בנוגע לאובייקטים האחרים חל אותו כלל. אובייקטים נוספים שה-references אליהם מאוחסנים בתוך כל אחד מהם גם ייכתבו.

```
public void write(int data) throws IOException
```

מתודה לכתיבתו של byte אחד בודד. המתודה תמתין ("תיחסם"-"blocked") עד אשר ה-byte ייכתב. מתודה זו באה במקומה של מתודה שהגיעה בהורשה מהמחלקה OutputStream.

```
public void write(byte []b) throws IOException
```

מתודה לכתיבתו של מערך של bytes. מתודה זו תיחסם ("blocked"), משמע תמתין בפעולתה, עד אשר מערך הבתים ייכתב. מתודה זו באה במקומה של מתודה שהגיעה בהורשה מהמחלקה OutputStream.

```
public void write(byte []b, int off, int len) throws IOException
```

מתודה לכתיבת תת מערך מתוך מערך של בתים (bytes). המתודה כותבת את len הבתים שמופיעים במערך b, החל מה-byte שנמצא במקום off. מתודה זו באה במקומה של מתודה שהגיעה בהורשה מהמחלקה OutputStream.

```
public void flush() throws IOException
```

מתודה זו גורמת לכתיבת כל ה-bytes שנאספו ב-buffer ושטרם נשלחו בפועל. מתודה זו באה במקומה של מתודה שהגיעה בהורשה מהמחלקה OutputStream.

```
public void close() throws IOException
```

מתודה לסגירת ה-stream שמיוצג על ידי אובייקט ה-ObjectOutputStream . בתום השימוש באובייקט ה-ObjectOutputStream שיוצרים חייבים להפעיל עליו מתודה זו כדי לשחרר את כל המשאבים שנעשה בהם שימוש. מתודה זו באה במקומה של מתודה שהגיעה בהורשה מהמחלקה OutputStream.

```
public void writeBoolean(boolean data) throws IOException
```

מתודה זו משמשת לכתיבת ערך מטיפוס boolean , הערך data , לאובייקט ה-ObjectOutputStream שעליו היא הופעלה.

```
public void writeByte(int data) throws IOException
```

מתודה זו משמשת לכתיבת ערך מטיפוס int אל אובייקט ה-ObjectOutputStream שעליו היא הופעלה.

```
public void writeShort(int data) throws IOException
```

מתודה זו משמשת לכתיבת ערך מטיפוס short (הערך שמאוחסן ב-16 הביטים הראשונים בערך ה-int שמאוחסן בפרמטר data).

```
public void writeChar(int data) throws IOException
```

מתודה זו משמשת לכתיבת ערך מטיפוס char לאובייקט ה-ObjetOutputStream שעליו היא הופעלה.

```
public void writeInt(long data) throws IOException
```

מתודה זו משמשת לכתיבת ערך מטיפוס int אל אובייקט ה-ObjectOutputStream אשר עליו היא הופעלה.

```
public void writeFloat(long float) throws IOException
```

מתודה זו משמשת לכתיבת ערך מטיפוס float אל אובייקט ה-ObjectOutputStream אשר עליו היא הופעלה.

```
public void writeDouble(double data) throws IOException
```

מתודה זו משמשת לכתיבת ערך מטיפוס double אל אובייקט ה-ObjectOutputStream אשר עליו היא הופעלה.

```
public void writeUTF(String data) throws IOException
```

מתודה זו משמשת לכתיבת מחרוזת של תווים בפורמט UTF.

המחלקה ObjectInputStream מתארת stream שדרכו זורמים bytes אשר מתארים אובייקט/ים אל תוך התוכנית. אובייקט מטיפוס מחלקה זו משמש לקריאת אובייקטים וערכים מטיפוסים בסיסיים אשר עברו serialization. הם יכולים להגיע מקובץ או ממקור אחר כדוגמת מקור אשר שולח אותם באמצעות כתיבתם לאובייקט מטיפוס ObjectOutputStream.

רק אובייקטים מטיפוס מחלקות שבהגדרתן מיושם java.io.Serializable או java.io.Externalizable יכולים להיקרא מתוך אובייקט מטיפוס ObjectInputStream . קריאתו של אובייקט מתוך אובייקט ה-ObjectInputStream נעשית באמצעות המתודה readObject. מתודה זו מחזירה reference לאובייקט שנקרא. כיוון שה-reference שמוחזר הוא reference מטיפוס Object.יש לבצע casting ל-reference שהגיע כך שיהיה reference מטיפוס המחלקה המתאימה. גם מערכים וגם מחרוזות תווים נחשבים ב-Java לאובייקטים , ולכן יש לבצע casting מתאים גם כאשר מדובר בקריאתם. ערכים מטיפוס בסיסי (פרימיטיבי) ניתן לקרוא באמצעות אחת המתודות המתאימות לטיפוס הערך שקוראים.

ה-consturctors העיקריים:

```
public ObjectInputStream(InputStream in)
      throw IOException, StreamCorruptedException
```

constructor זה משמש ליצירת אובייקט חדש אשר יקושר ל-input stream שה-reference שלו נשלח אל ה-constructor.

```
public ObjectInputStream() throws IOException, SecurityException
```

constructor זה הוגדר כדי לאפשר את הגדרתה של מחלקה שיורשת ממחלקה זו.

המתודות העיקריות שמוגדרות במחלקה זו:

```
public final Object readObject() throws OptionalDataException,
      ClassNotFoundException, IOException
```

מתודה זו משמשת לקריאת אובייקט מתוך אובייקט ה-ObjectInputStream שעליו היא הופעלה. המתודה מחזירה reference מטיפוס Object לאובייקט שקראה. יש לעשות casting בהתאם לאובייקט שבו מדובר.

```
public int read() throws IOException
```

מתודה זו קוראת מה-stream שעליו היא הופעלה byte אחד, ומחזירה ערך מטיפוס int אשר 8 הביטים הראשונים שלו מייצגים את ה-byte שנקרא. מתודה זו באה במקומה של מתודה שהגיעה בהורשה מהמחלקה InputStream.

```
public int read(byte []b, int off, int len) throws IOException
```

מתודה זו קוראת אל תוך מערך של bytes (מערך שה-reference אליו מאוחסן במשתנה b) len בתים (bytes) מתוך אובייקט ה-ObjectInputStream אשר עליו היא הופעלה. הבתים שהיא קוראת נשמרים אל תוך המערך b החל מהמקום שמספר האינדקס שלו הוא off. המתודה מחזירה את מספר הבתים שהיא קראה או 1- אם הגיעה אל סופו של ה-stream.

```
public int available() throws IOException
```

מתודה זו מחזירה את מספר הבתים שניתן לקרוא מבלי לחכות לבתים נוספים שיגיעו.

```
public void close() throws IOException
```

מתודה זו גורמת לסגירתו של אובייקט ה-ObjectInputStream אשר עליו היא הופעלה. בפעולתה ישוחררו כל משאבי המערכת אשר שימשו את האובייקט.

```
public boolean readBoolean() throws IOException
```

מתודה לקריאת ערך מטיפוס boolean מתוך אובייקט ה-ObjectInputStream אשר עליו היא הופעלה.

```
public byte readByte() throws IOException
```

מתודה לקריאת ערך מטיפוס byte מתוך אובייקט ה-ObjectInputStream אשר עליו היא הופעלה.

```
public short readShort() throws IOException
```

מתודה לקריאתו של ערך מטיפוס short מתוך אובייקט ה-ObjectInputStream אשר עליו היא הופעלה.

```
public int readUnsignedShort() throws IOException
```

מתודה לקריאתו של ערך מטיפוס short מתוך אובייקט ה-ObjectInputStream אשר עליו היא הופעלה. הערך שנקרא מקבל התייחסות כערך אי שלילי (יש התעלמות מביט הסימן בקריאתו).

```
public char readChar() throws IOException
```

מתודה זו משמשת לקריאת ערך מטיפוס char בגודל 16 ביטים מתוך אובייקט ה-ObjectInputStream אשר עליו היא הופעלה.

```
public int readInt() throws IOException
```

מתודה זו משמשת לקריאתו של ערך מטיפוס int מתוך אובייקט ה-ObjectInputStream אשר עליו היא הופעלה.

```
public long readLong() throws IOException
```

מתודה לקריאתו של ערך מטיפוס long מתוך אובייקט ה-ObjectInputStream אשר עליו היא הופעלה.

```
public float readFloat() throws IOException
```

מתודה לקריאתו של ערך מטיפוס float מתוך אובייקט ה-ObjectInputStream אשר עליו היא הופעלה.

```
public double readDouble() throws IOException
```

מתודה לקריאתו של ערך מטיפוס double מתוך אובייקט ה-ObjectInputStream אשר עליו היא הופעלה.

```
public String readUTF() throws IOException
```

מתודה זו משמשת לקריאתה של מחרוזת מתוך אובייקט ה-ObjectInputStream, וזאת כאשר היא מגיעה בפורמט UTF.

הדוגמא הבאה כוללת שלוש מחלקות. המחלקה Person מתארת אדם. ממחלקה זו יווצר אובייקט לתיאור אדם. המחלקה SavePersonToFile פועלת כ-stand alone application אשר מקבל בשורת הפקודה ערכים שמתארים אדם, יוצר אובייקט מטיפוס Person בהתאם ושומר אותו לקובץ ששמו זהה למספר תעודת הזהות. המחלקה GetPerson פועלת כ-stand alone application אשר מקבלת בשורת הפקודה מספר תעודת זהות, ניגש לקובץ ששמו זהה למספר תעודת הזהות שהתקבל, קוראת את האובייקט ששמור בתוכו ומדפיסה את פרטיו למסך.

המחלקה Person:

```
import java.io.Serializable;

public class Person implements Serializable
{
      private String privateName;
```

```
private String familyName;
private int id;
private String telephone;

public static long serialVersionUID = 123123123;

public Person(String privateName, String familyName, int id,
       String telephone)
{
    super();
    this.privateName = privateName;
    this.familyName = familyName;
    this.id = id;
    this.telephone = telephone;
}

public String getPrivateName()
{
     return privateName;
}

public void setPrivateName(String privateName)
{
     this.privateName = privateName;
}

public String getFamilyName()
{
     return familyName;
}

public void setFamilyName(String familyName)
{
```

```
		this.familyName = familyName;
	}

	public int getId()
	{
		return id;
	}

	public void setId(int id)
	{
		this.id = id;
	}

	public String getTelephone()
	{
		return telephone;
	}

	public void setTelephone(String telephone)
	{
		this.telephone = telephone;
	}

	public String toString()
	{
		return this.privateName + " " + this.familyName +
		 " " + this.telephone + " " + this.id;
	}
}
```

המחלקה SavePersonToFile:

```
import java.io.*;

public class SavePersonToFile
{
      public static void main(String[] args)
      {
            if (args.length == 4)
            {
                  Person person = new Person(args[0],
                        args[1],Integer.parseInt(args[2]),args[3]);
                  FileOutputStream fos = null;
                  ObjectOutputStream oos = null;
                  try
                  {
                        fos = new FileOutputStream(args[2]);
                        oos = new ObjectOutputStream(fos);
                        oos.writeObject(person);

                  }
                  catch(IOException e)
                  {
                        e.printStackTrace();
                  }
                  finally
                  {
                        if(fos!=null) try{fos.close();}catch(IOException e){}
                        if(oos!=null) try{oos.close();}catch(IOException e){}
                  }
            }
            else
```

```
		{
			System.out.println("utility requires four arguments!");
		}
	}
}
```

המחלקה GetPersonFromFile:

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class GetPersonFromFile
{
	public static void main(String[] args)
	{
		if (args.length == 1)
		{
			FileInputStream fis = null;
			ObjectInputStream ois = null;
			try
			{
				fis = new FileInputStream(args[0]);
				ois = new ObjectInputStream(fis);
				Person person = (Person)(ois.readObject());
				System.out.println(person);

			}
			catch(ClassNotFoundException e)
			{
				e.printStackTrace();
			}
			catch(IOException e)
```

```
                {
                        e.printStackTrace();
                }
                finally
                {
                        if(fis!=null) try{fis.close();}catch(IOException e){}
                        if(ois!=null) try{ois.close();}catch(IOException e){}
                }
        }
        else
        {
                System.out.println("This utility requires one argument only!");
        }

    }
}
```

המחלקות Reader ו- Writer

שתי מחלקות אסטרקטיות שעומדות בראש של שתי היררכיות של מחלקות אשר מתארות streams לקלט ולפלט. בעוד שהמתודות שניתן להפעיל על אובייקטים מטיפוס InputStream ו-OutputStream מניחות שהנתונים שזורמים אל/מ התכנית הם אוסף של בתים בלבד, המתודות שניתן להפעיל על אובייקטים מטיפוס Reader ו-Writer מניחות שהבתים שזורמים מייצגים chars.

אובייקטים שנוצרים מהמחלקות שיורשות מ-Writer ומ-Reader מהווים streams אשר מתארים זרימה של chars. משתמשים בהם לצורך יצירת streams שדרכם עוברים טקסטים בפורמט Unicode.

כדי לחבר streams שבהם זורמים טקסטים בפורמט אחר מ-Unicode עם streams מסוג Reader ו-Writer יש להשתמש ב-InputStreamReader וב-OutputStreamWriter.

המחלקה File

אובייקט מטיפוס מחלקה זו יכול לייצג או קובץ או ספריה. אלה לא חייבים להיות קבצים ו/או ספריות שכבר קיימים/קיימות. אובייקט מטיפוס מחלקה זו יכול גם לייצג קבצים/ספריות שטרם נוצרו. מחלקה זו מספקת מתודות שמשמשות להשגת אינפורמציה על קבצים/ספריות שאובייקטים מטיפוסה מייצגים.

אובייקט מטיפוס מחלקה זו יכול לפרט את שמות הקבצים אשר בספריה מסוימת, לבדוק אם קובץ מסוים קיים, לבדוק את סוגו של קובץ, ליצור ספריות חדשות, לשנות את שמם של קבצים, למחוק קבצים קיימים וכדומה. . .

מחלקה זו לא תומכת בפעולות קלט/פלט לקבצים. פעולות קלט/פלט לקבצים מתבצעות באמצעות אובייקטים שיוצרים מהמחלקות:

RandomAccessFile

FileReader

FileWriter

FileInputStream

FileOutputStream

הקבועים שמוגדרים במחלקה File:

```
public static final char separatorChar
```

זהו התו שמשמש להפרדה בין שמות ספריות או בין שם ספריה לשם של קובץ. אם במערכת קיימת מחרוזת תווים שמשמשת להפרדה האמורה אז משתנה זה מכיל את התו הראשון מתוך המחרוזת

```
public static final String separator
```

זהו ייצוג במחרוזת תווים של סימן ההפרדה בין שמות של ספריות לשמות של ספריות/קבצים.

```
public static final char pathSeparatorChar
```

קבוע זה מכיל את התו הראשון מתוך רצף התווים שמשמש במחשב בתור סימן מפריד ברשימה של pathים. ב-windows זהו הסימן ; וביוניקס זהו הסימן : .

```
public static final String pathSeparator
```

זהו ייצוג במחרוזת תווים של סימן ההפרדה שמשמש במחשב להפרדה בין pathים שמופיעים ברשימה של pathים.

ה-constructors העיקריים שקיימים במחלקה:

```
public File (String pathname)
```

ליצירת אובייקט שמתאר את הקובץ/הספריה שתיאורו נשלח כמחרוזת תווים אל ה-constructor. התיאור בצורה של מחרוזת תווים כולל גם את השם של הקובץ/הספריה, וגם ה-path שלו/ה.

```
public File (String pathname, String child)
```

ליצירת אובייקט שמתאר את הקובץ/הספריה ששמו/ששמה child ושה-path אליו הוא pathname. אם pathname שווה ל-null אז פעולתו זהה לפעולת ה-constructor הקודם.

```
public File(File parent, String child)
```

ליצירת אובייקט חדש שמתאר את הקובץ/הספריה ששמו/שמה child ושנמצא תחת הספריה שמתוארת על ידי אובייקט מטיפוס File אשר ה-reference אליו נשלח כארגומנט ראשון אל ה constructor.

המתודות העיקריות שקיימות במחלקה:

```
public String getName()
```

מתודה זו מחזירה reference מטיפוס String אשר מתאר את השם של הקובץ/הספריה שמתארת אובייקט ה-File.

```
public String getParent()
```

מתודה זו מחזירה reference לאובייקט מטיפוס String אשר מתאר את שמה של ספריית האב לספריה/לקובץ אשר מתאר האובייקט.

```
public boolean exists()
```

מתודה זו מחזירה true או false בהתאם לשאלה האם הקובץ/הספריה שמתאר האובייקט קיים או לא.

```
public boolean isDirectory()
```

מתודה זו מחזירה true או false בהתאם לשאלה האם האובייקט מתאר ספריה או לא.

```
public boolean isFile()
```

מתודה זו מחזירה true או false בהתאם לשאלה האם האובייקט מתאר קובץ או לא.

```
public String[] list()
```

מתודה זו מחזירה reference למערך מטיפוס String אשר מכיל references ל-Strings אשר מתארים את שמות תתי הספריות ואת שמות הקבצים שנמצאים תחת הספריה שמיוצגת על ידי אובייקט ה-File אשר עליו היא הופעלה. אם האובייקט שממנו המתודה הופעלה מייצג קובץ אז המתודה מחזירה null.