תרגיל בית מס. 4 – מפענח ל-JSON

בתרגיל זה נטפל בביטויים הנתונים בשפת JSON. אנו נגדיר את המחלקות הדרושות לתיאור ביטוי הנתון בשפה זו, נשתמש בהן ליצירת מופע המתאר בטוי נתון, ונאפשר שאילתות לגבי תוכנו של הביטוי.

1. מה זה JSON?


האיקון של JSON

JSON (קיצור של Java Script Object Notation - הגדרת עצמים באמצעות JavaScript[1]) היא שפה הרוכשת פופולריות בהתמדה, ומשמשת כתחליף ל-XML המיושנת. באמצעותה ניתן לתאר את התוכן של עצם (הסטטוס שלו) באמצעות טקסט קריא ומובן גם לבני אדם וגם למחשבים. (הסינטקס של JSON חלקי לזה של JavaScript ומכאן השם.)

בטוי JSON יכול להיות אחד משני דברים:

1. מערך (**array**) – משהו הדומה ל-List ב-Java: סדרה של ערכים, מופרדים ע"י פסיקים, הנתונים בין סוגריים מרובעים. לדוגמה, [0, 1, 3.14, "אינסוף"] הוא בטוי המתאר מערך עם ארבעה אברים: שני מספרים שלמים, ממשי אחד ומחרוזת. ערכים במערך יכולים להיות מכל סוג שהוא.
2. מערך אסוציאטיבי (**object**[2]) – משהו הדומה ל-Map ב-Java: סדרה של זוגות סדורים בפורמט <ערך>:<מפתח>, מופרדים ע"י פסיקים, הנתונים בין סוגריים מסולסלים. לדוגמה, {"me":"Tarzan","you":"Jane"} הוא ביטוי המתאר מפה בה המפתח "me" קשור לערך "Tarzan" והמפתח "you" לערך "Jane". המפתחות במערך אסוציאטיבי הם תמיד מחרוזות; הערכים יכולים להיות מכל סוג שהוא.

מהדוגמאות לעיל רואים שערכים יכולים להיות מחרוזות או מספרים (שלמים או ממשיים). אבל לערך מותר, באופן רקורסיבי, להיות גם מערך או מערך אסוציאטיבי. למשל,

```
{"שם":"אברהם אבינו",
 "גיל במותו":175,
 "נשים":["שרה", "הגר", "קטורה"],
 "צאצאים":{"שרה":["יצחק"],
           "הגר":["ישמעאל"],
           "קטורה":["יקשן", "מדן", "ישבק", "מדין", "שוח", "זמרן"]
          }
}
```

בדוגמה זו, הביטוי כולו הוא מערך אסוציאטיבי, המכיל 4 מפתחות: "שם", "גיל במותו", "נשים" ו-"צאצאים". שימו לב שכל המפתחות הם מחרוזות, אבל הערכים, לעומת זאת, מטיפוסים שונים: ל-"שם" מוצמדת מחרוזת, ל-"גיל במותו" מתאים מספר, ל-"נשים" – מערך של מחרוזות, ול-"צאצאים" מותאם מערך אסוציאטיבי שלם, שבו הערכים הם בעצמם מערכים של מחרוזות. (בחלק מהמערכים יש רק מחרוזת אחת, אבל הם עדיין מערכים.)

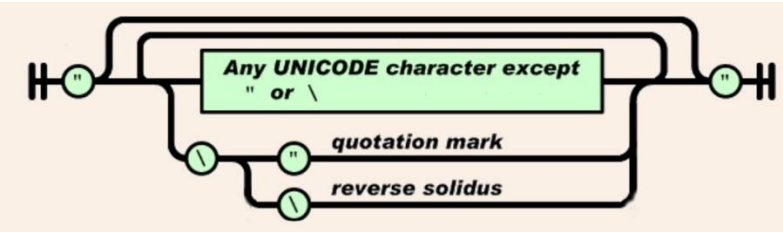
---

[1] ל-JavaScript אין שום קשר ל-Java, זו בחירה אומללה ומבלבלת של שם לשפה אחרת.

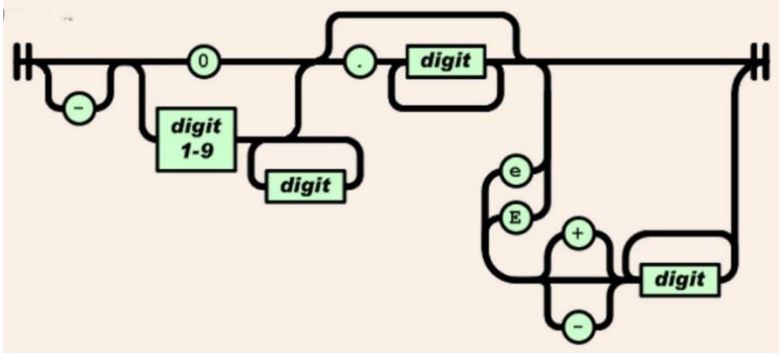
[2] סליחה על השימוש הנלוז במלה "object", אבל כך נקרא המערך האסוציאטיבי בדוקומנטציה של JSON.

הסינטקס של JSON נתון בדיאגרמות הבאות:

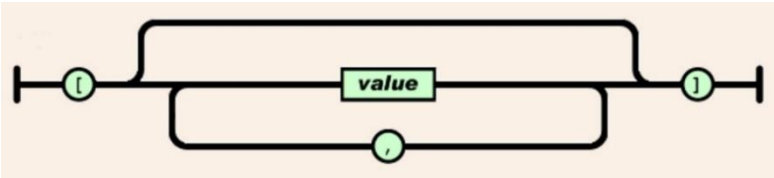
מחרוזת (string):



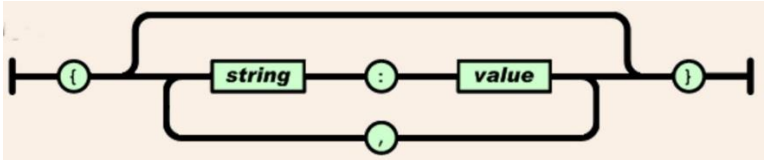
מספר (number):



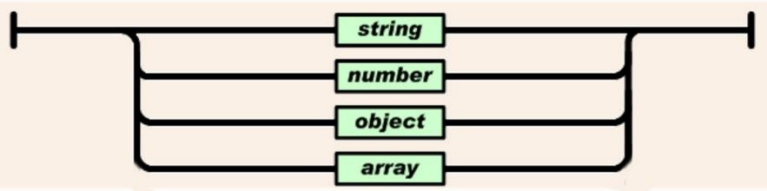
מערך (array):



מערך אסוציאטיבי (object):



ולבסוף, ערך (value):



את הדיאגרמות קוראים כך: מתחילים בצד שמאל, ובכל צומת מותר לבחור בכל דרך אפשרית. כל סדרה של בחירות תיצור בטוי JSON חוקי; אוסף כל הבחירות האפשריות יוצר את כל בטויי ה-JSON האפשריים.

לדוגמא, נתבונן בדיאגרמה של מערך (array): בקריאה מצד שמאל אנו פוגשים ב-[, מה שאומר שכל עצם מסוג מערך חייב להתחיל בסימן '['. אחר כך אנו פוגשים בפיצול: אפשר להמשיך ישר או לעלות למעלה. עליה למעלה תביא אותנו לסימן] ומשם לסוף הדיאגרמה, מה שאומר ש-']' ואחריו '[' הוא מערך תקין (אם כי ריק מתוכן). אם, לעומת זאת, נמשיך ישר, נפגוש ב-value, מה שאומר שאברי המערך רשאים להיות כל ערך שהוא (כמובהר בדיאגרמה של value בהמשך). אחרי ה-value יש שוב פיצול: אפשר להמשיך ישר אל ה-] וסיום המערך, או לרדת למטה, שם נפגוש-, משם אין ברירה אלא להמשיך לעוד מופע של-value. מכאן שעצמים במערך מופרדים על ידי פסיק: ','. יתר הדיאגרמות פועלות באופן דומה.

2.מטרת התרגיל

בתרגיל זה נכתוב תכנית הקוראת ביטוי JSON תקין ובונה פיזית בזכרון את העצם שהבטוי מתאר. התכנית תאפשר הדפסה של תכנו של העצם, וגם תשאול ערכים ספציפיים בו. למשל, לגבי הדוגמה של אברהם אבינו שניתנה לעיל, יהיה ניתן לשאול מהו הערך של האבר השלישי ברשימת בניה של "קטורה" בערך הקשור במפתח "צאצאים":

```
String son3 = abraham.get( "צאצאים" ).get( "קטורה" ).get( 2 );
```

(בהנחה ש-`abraham` הוא reference לעצם שנוצר מהבטוי שניתן בדוגמה, אזי `son3` יכיל את הערך "`ישבק`".)

החלק המרכזי של "הבנת" הקלט נקרא parsing. בהשאלה מעולם התכנות, תהליך ה-parsing של שורה בתכנית מחשב, למשל, כרוך בזהוי סוג הפקודה (אם זה, למשל, while או if), מהם הפרמטרים (מהו התנאי של ה-while או ה-if), מהן הפקודות שיש לבצע אם התנאי מתקיים, וכו'. בדומה, בכל שלב בפענוח של בטוי JSON יש לברר באיזה מבנה נמצאים (מערך או מערך אסוציאטיבי), אילו אברים יש במבנה, וכו'.

אחת הטכניקות לבנית מפענחים (parsers) נקראת Recursive Descent. טכניקה זו מתאימה רק לסוגי דקדוק מסויימים, אך למזלנו, הדקדוק של JSON (זה שנתון בעמוד הקודם) אכן מתאים לה. בטכניקה זו יש לבנות מתודה לכל סוג של עצם בדקדוק (לדוגמה מספר, מחרוזת, מערך וכו'), שתפקידה להכיר ולקבל רק עצמים מסוג זה. כל מתודה כזו קוראת מהקלט את כל התוים ששייכים לעצם שאותו היא מזהה. היתרון בשיטה נובע מהאפשרות לקרוא למתודה המזהה עצם בכל מקום בו הדבר נדרש.

לדוגמה, ערך (Value) יכול, לפי הדקדוק, להיות רק אחד מארבעה דברים. לפיכך, המתודה המזהה ערך צריכה רק להחליט איזה מארבעת דברים אלה נמצא לפניה בקלט, ואז לקרוא למתודה המתאימה. העובדה שאחד מדברים אלה יכול להיות מערך, ושמערכים, לפי הדקדוק, יכולים להכיל ערכים בפני עצמם, אינה מסבכת את המפענח: המתודה של ערך יכולה לקרוא למתודה של מערך, שמצידה יכולה לקרוא שוב למתודה של ערך, אם נחוץ. (זו הסיבה לשמה של הטכניקה: היא עושה שימוש מסיבי ביכולת של המחשב להפעיל מתודות באופן רקורסיבי.)

באתר נתונה מחלקה בשם CharScanner המאפשרת קריאה של הקלט הסטנדרטי תו אחר תו (יכולת שנעדרת, משום מה, מהתכונות של ה-Scanner הסטנדרטי.) יתר על כן, מחלקה זו מרחיבה את המנשק הרגיל של Iterator עם מתודה נוספת הקרויה peek(): מתודה המאפשרת לברר מהו התו הבא בקלט מבלי ממש לקרוא אותו. כלומר, אם peek() מודיעה שהתו הבא הוא '[', הקריאה הבאה ל-next() תחזיר את התו הזה בדיוק. (מכאן שניתן לקרוא ל-peek() כמה פעמים מבלי לקדם את הקלט.)

התכונה של הדקדוק של JSON שעושה אותו מתאים לשיטת ה-Recursive Descent היא זו: התו הראשון בכל עצם מזהה את העצם חד ערכית[3]. מערך מתחיל ב-'[', מערך אסוציטיבי ב-'{', מחרוזת ב-'"' ומספר ב-'-' או ספרה. (שימו לב שכל סימן אחר בראש עצם אינו חוקי!) מכאן החשיבות של המתודה peek(): אפשר להשתמש בה כדי לברר מהו העצם הבא בקלט, ולפי זה לבחור לאיזו מתודה לקרוא, וזאת מבלי לפגוע בקלט שעל מתודה זו "לאכול" לפני שהיא חוזרת.

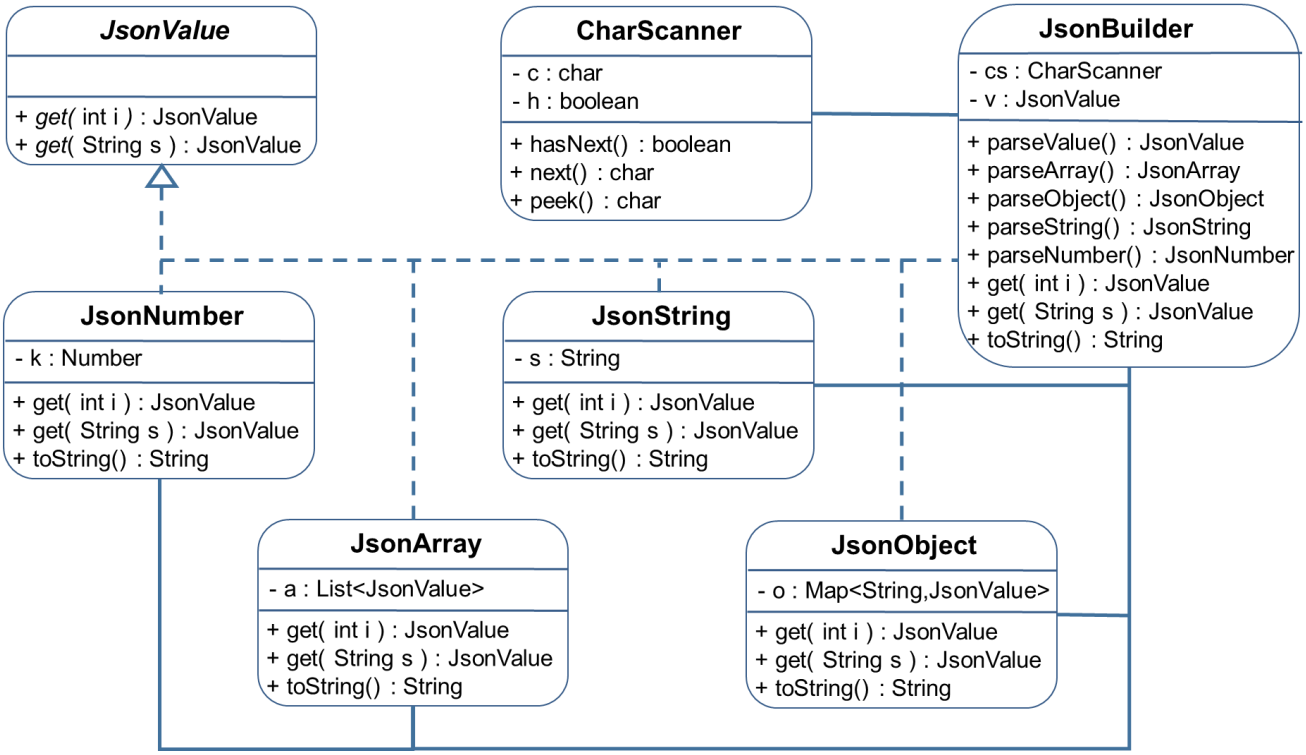
אחרי שנבנו כל המתודות הדרושות, לא קשה להרחיב אותן כך, שבנוסף לזהוי של הקלט כביטוי חוקי של JSON, גם תיצורנה את האלמנטים המקבילים בזכרון של התכנית באופן בו הם

[3] דקדוק בעל תכונה זו קרוי "דקדוק LL1": השפה נקראת משמאל לימין (זה ה-L הראשון, שאומר Left), ודי בסימן אחד מצד שמאל (זה ה-L1) לזהות את טיב הביטוי. בשפה מסוג "LR0" יש לקרוא את הביטוי עד סופו (0 תוים מקצהו הימני – R זה Right) כדי לזהותו בברור. C ו-Java הן דקדוק מהסוג האחרון.

מיוצגים ב-Java: מחרוזת בצורת String, מספר בצורת Number (מחלקה המייצגת גם Integer וגם Double), מערך בצורת List ומערך אסוציאטיבי בצורת Map.

התוצר הסופי של התכנית הוא ערך (Value) המיוצג באמצעות המבנים המתאימים ב-Java.

להלן דיאגרמת UML שעשויה לסייע לכם בבנית המפענח:



יש להגדיר גם שתי מחלקות של חריגות:

1. JsonSyntaxException: חריגה שכל אחת מהמתודות parse… עשויה לזרוק במקרה שהיא נתקלת בקלט לא צפוי. שגיאה כזו מציינת טעות בסינטקס של קלט ה-JSON.
2. JsonQueryException: חריגה שכל אחת מהמתודות get… יכולה לזרוק במקרה שהשאילתא אינה נכונה. למשל, יש נסיון לקבל אבר שמספרו גדול מאורך המערך אליו השאילתא מתייחסת, או נסיון לקבל אבר מתוך מערך אסוציאטיבי על פי מקומו ולא על פי המפתח שלו, וכדומה.

את התרגיל יש להגיש בזוגות עד 22 במאי 2021. יש לכלול בהגשה את כל קבצי ה-Java וכן קובצי טקסט שיצרתם לצורך בדיקה, וצילומי מסך של הבדיקה. את קבצי ההגשה יש לקבץ לקובץ יחיד (ZIP או RAR) ששמו `Ex3_xxxxxxxxx_yyyyyyyyy.zip`, שבו x* ו-y* הם מספרי הזהות של המגישים. את הקובץ הזה יש להעלות לאתר הקורס.

שאלות, הבהרות קושיות ומענות – לפורום הקורס. אין זמן להתחבט בשאלות – מה שלא מובן יש לשאול ולברר, ומהר...

להלן תכנית main():

```
public static void main( String[] args )
{
   JsonBuilder avraham = null;
   try
   {
      avraham = new JsonBuilder( new File( args[0] ));

      System.out.println( avraham );
      System.out.println( avraham.get( "issue" ).get( "Ketura" ).get( 2 ) );
   }
   catch( JsonSyntaxException e )
   {
      e.printStackTrace();
   }
   catch( JsonQueryException e )
   {
      e.printStackTrace();
   }
   catch (FileNotFoundException e)
   {
      e.printStackTrace();
   }
}
```

להלן קובץ קלט אפשרי (מצוי באתר בשם JasonInput):

```
{ "name"  : "Avraham",
  "age"   : 175,
  "wives" : ["Hagar", "Sarah", "Ketura"],
  "issue" : {"Hagar" : ["Yishmael"],
             "Sarah" : ["Yitskhak"],
             "Ketura" : ["Zimran", "Jokshan", "Medan", "Midian",
"Ishbak", "Shuah"]
            }
}
```

ולהלן הפלט שקלט זה אמור להפיק:

```
{wives:[Hagar, Sarah, Ketura]
issue:{Sarah:[Yitskhak] Hagar:[Yishmael]
       Ketura:[Zimran, Jokshan, Medan, Midian, Ishbak, Shuah]}
age:175
name:Avraham}

Medan
```

בהצלחה!