שעור 4. תכנות מונחה עצמים

טיפול בחריגים

הדרך המקובלת לטיפול בשגיאות מתבססת בדרך כלל על הערכים שמחזירה הפונקציה. לדוגמה פונקציה find(...) המחפשת איבר בתוך מערך מחזירה 1- אם האיבר לא נמצא.
הטיפול בשגיאות בדרך זו לוקה בחסר. ראשית, אין אחידות בין תכנית לתכנית. שנית, בטיפול בשגיאות שמתבסס על הערכים המוחזרים מפונקציות, יש קושי בהעמסת אינפורמציה נוספת בנוסף לעצם התרחשותה של התקלה. חיסרון נוסף קיים בכך שלעתים לא קיימת אפשרות לקבוע ערך מסוים כסימן להתרחשותה של שגיאה. למרות חסרונות אלה, ניתן למצוא ב-**JAVA** מספר מתודות שנוקטות בדרך זו, לדוגמה פונקציה **indexOf(String str)** של מחלקת String אשר מחזירה 1- כאשר היא לא מוצאת **str** כתת-מחרוזת.

הטיפול בשגיאות/תקלות ב-**JAVA** כולל מנגנון של דיווח על התרחשותה של השגיאה/תקלה, ומנגנון שמטפל בה (משמע, מבצע פעולות מסוימות שיש לבצע בגלל התרחשות השגיאה/תקלה). גם המנגנון הראשון וגם השני מתבצעים באופן עצמאי, ומייד עם התרחשות השגיאה/תקלה.

אנו כבר מכירים את התופעה זו, לדוגמה כאשר מנסים לחלק למספר השווה לאפס:

```
public static void main(String[] args) {
      int x=2, y=0, z=0;
      z = z + x/y;
}
```

מקבלים הודעת שגיאה:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
      at SimpleExamples.main(SimpleExamples.java:6)
```

דוגמה אחרת לשימוש בחריגים: מנסם להגיע לאיבר שמחוץ למערך:

```
public static void main(String[] args) {
      int a[] = new int[5];
      for(int i=0; i<=a.length; i++){
            a[i] = i;
      }
}
```

מקבלים:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
      at SimpleExamples.main(SimpleExamples.java:9)
```

עם היווצרות השגיאה/תקלה נוצר אובייקט מטיפוס Exception מחלקה Exception שיורשת מהמחלקה Throwable (או מטיפוס מחלקה אחרת שיורשת מ- Throwable כמו כן, בהסברים שיופיעו בהמשך, בכל מקום שבו תהיה התייחסות למחלקה Exception, ההתייחסות תכלול גם את המחלקות אשר יורשות מ-Exception. אלה הן המחלקות שמתארות מצבי שגיאה/תקלה שבהם נעסוק.
הנה סוגי השגיאות הקיימות:

- שגיאות בקלט מהמשתמש, לדוגמה: כתובת שלא קיימת באינטרנט.
- תקלות החומרה, לדוגמה: נייר שנגמר במדפסת, דיסקט מקולקל.
- מגבלות פיזיות, לדוגמה: דיסקט שהתמלא ולא ניתן לכתוב אליו.
- שגיאות בפעולתן של מתודות בתכנית: ניסיון גישה למקום במערך שמעבר לטווח האינדקסים שלו, ניסיון להוציא מווקטור (Vector) איבר כאשר הווקטור ריק:

```
public static void main(String[] args) {
      Vector<String> vec  = new Vector<String>();
      vec.elementAt(0);
}
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0 >= 0
      at java.util.Vector.elementAt(Unknown Source)
      at SimpleExamples.main(SimpleExamples.java:14)
```

כאשר מתרחשת שגיאה/תקלה אופן הטיפול בה יהיה אחד מהשניים:

1. התכנית תמשיך לפעול, ותאפשר למשתמש לנסות לבצע שוב את אותה פעולה או לבצע פעולות אחרות.
2. התכנית תסתיים תוך שהיא שומרת את הנתונים העדכניים שיש בה ותוך מתן חיווי מתאים למשתמש.

כעת אסקור בקצרה את הפקודות ואת המלים השמורות:

throw

זוהי פקודה שיש לרשום בצירוף מצביע לאובייקט מטיפוס Exception (או מחלקה אחרת שיורשת מהמחלקה Exception). ביצועה של פקודה זו נעשה בדרך כלל כחלק ממשפט תנאי, ובעצם התרחשותה מתרחשת, למעשה, פעולת הדיווח על השגיאה/התקלה (פעולת הזריקה של השגיאה/תקלה). כדי להקל בהסברים, ולעשותם דומים ככל האפשר להסברים באנגלית, אשתמש מעתה ואילך במילה "לזרוק" במקום במילה "לדווח".

throws

זוהי מילה שמורה שיכולה להופיע בתוך כותרתה של מתודה. מילה שמורה זו מופיעה בצירוף שם המחלקה שמתארת את השגיאה/תקלה (זו יכולה להיות המחלקה Exception או מחלקה אחרת שיורשת מהמחלקה Exception). בעצם הופעתה של מילה זו בכותרת של מתודה ניתן לדעת שמתוך המתודה עלול להיזרק exception מטיפוס המחלקה ששמה הופיע אחרי המילה throws.

try

זוהי מילה שמורה שמופיעה בצירוף בלוק שבתוכו פקודה אחת או יותר. בבלוק ה- try (הבלוק שאחרי המילה try) ממקמים את שורות הקוד שמתוכן עלול להיזרק ה- exception (בין אם מתוך מתודה שהקריאה להפעלתה ממוקמת בתוך אותו בלוק, ובין אם על ידי הפקודה throw שמופעלת בתוך אותו בלוק).

catch

מילה שמורה זו – מופיעה בצירוף סוגריים בדומה לכותרת של פונקציה. בסוגריים מופיע שם של פרמטר בצירוף שם הטיפוס שלו, שהוא המחלקה Exception (או מחלקה אחרת שיורשת ממנה). פרמטר זה קולט לתוכו את ה- מצביע של אובייקט ה- Exception (או מחלקה אחרת שיורשת מ- Exception) שנזרק מתוך בלוק ה- try שהוסבר קודם לכן. אחרי המילה catch והסוגריים כפי שהוסבר, יופיע בלוק אשר יכלול את הפקודות אשר יתבצעו במידה שמתוך בלוק ה- try ייזרק מצביע לאובייקט מטיפוס Exception (או מחלקה אחרת שיורשת מ- Exception), ובלבד שה- exception שייזרק יהיה מטיפוס המחלקה שמופיעה בסוגריים (או שמטיפוס מחלקה שיורשת מהמחלקה שמופיעה בסוגריים). לשורה שמתחילה במילה catch נקרא בשם משפט ה-catch.

finally

מילה שמורה אשר תופיע אחרי בלוק ה- catch (ואם יש יותר מ- catch אחד, אז אחרי כולם), ומיד אחריה בלוק של פקודות אשר יתבצעו בכל מקרה. בין אם נזרק exception או לא. בבלוק ה- finally נמקם, בעיקר, פקודות לסגירת משאבים של התכנית.

ניתן לתאר את הפקודות והמשפטים שהוסברו עד כה באופן הסכימטי הבא:

```
try{
        פקודות שונות שכוללות בין היתר את הפעלתן של מתודות שעלולות לזרוק
        exceptions או לחילופין פקודה/פקודות throw לזריקת exception\s
}
catch (Exception e){
        פקודות שיתבצעו אם ייזרק Exception
}
finally{
        פקודות שיתבצעו בכל מקרה
}
```

דוגמה:

```
int arr[] = {1,2,3,4,5};
try{
```

```
        for (int i=0; i<=arr.length; i++)  arr[i] = i;
}
catch (ArrayIndexOutOfBoundsException e){
        System.out.println("array index out of bounds");
        System.err.println(e);
 }
```

הפקודה **throw** מופיעה בצירוף מצביע לאובייקט מטיפוס Exception (או מחלקה אחרת שיורשת ממנה). סכימת הפקודה היא:

throw *reference to Exception instance*

כאשר מתבצעת הפקודה **throw** אז המתודה שמתוכה נזרק ה- exception מופסקת בדיוק בשורה שבה מופיעה הפקודה **throw**. הביצוע מופסק באופן מיידי, והוא עובר אל ה- **catch** המתאים.

<u>המילה השמורה throws</u>

אם בתוך מתודה מתבצעת פעולת זריקה של exception (בין אם באמצעות הפקודה **throw** ובין אם מתוך מתודה אחרת שהופעלה) אז חייבים לבצע את אחת משתי הפעולות הבאות (בהמשך נראה שאם מדובר ב-exception מטיפוס מחלקה שיורשת מ- **RuntimeException** אז לא חייבים):

1. בתוך המתודה ה- **exception** יטופל באמצעות הבלוקים **try & catch**.
2. המתודה האמורה – מוסיפים לכותרתה את המילה **throws** בצירוף שם טיפוס ה- **exception** שנזרק מתוכה. זאת מהווה אינדיקציה לכך שמתוך המתודה עלול להיזרק **exception** אל המקום שממנו היא הופעלה. לדוגמא:

```
public double getValue() throws ArithmeticException() {
        if (num<0)
                throw new ArithmeticException();
        else
                return Math.sqrt(num);
}
```

משמעותה של תוספת זו היא שה- **exception** שנזרק בתוך המתודה יועבר הלאה אל המקום שממנו מתודה זו הופעלה.

ניתן גם להצהיר על כוונתה של מתודה לזרוק **exceptions** ממספר סוגים. במקרה כזה יש לרשום בשורת הכותרת של המתודה את כל שמות טיפוסי ה- **exceptions** שיכולים להיזרק עם פסיקים מפרידים. אין כל מגבלה למספר סוגי ה- exceptions שיכולים להיזרק.

אם מבצעים **overriding** למתודה שמגיעה בהורשה לא ניתן לקבוע במתודה החדשה באמצעות **throws** כי היא מסוגלת לזרוק **exceptions** שלא צוינו כניתנים לזריקה על ידי המתודה המקורית. ניתן לציין פחות exceptions מכפי שצוין במתודה המקורית, אך לא יותר.

כאשר מציינים שמתודה מסוימת יכולה לזרוק exceptions מסוגים מסוימים (באמצעות הוספת המילה throws ושמות הטיפוסים של אותם exceptions לשורת הכותרת שלה), ניתן יהיה לזרוק מהמתודה גם Exceptions מטיפוס המחלקות שיורשות מהמחלקות ששמן צוין בשורת הכותרת של המתודה (אחרי המילה throws).

לדוגמא:

```
public double getValue() throws Exception{
        .
        .
        .
}
```

מתודה שבכותרתה יצוין שהיא מסוגלת לזרוק exception מטיפוס המחלקה Exception, תהיה מסוגלת, למעשה, לזרוק כל exception שהוא מטיפוס מחלקה אשר יורשת מהמחלקה Exception.

אם מתוך המתודה נזרק **RuntimeException** (או **exception** מטיפוס מחלקה אחרת שיורשת מהמחלקה **RuntimeException**) אז לא חייבים לתפוס אותו באמצעות בלוק **try & catch** וגם לא חייבים לציין בכותרת של המתודה - באמצעות **throws** – שהוא עלול להיזרק ממנה, כמו בדוגמה הראשונה:

```
int x=2, y=0, z=0;
z = z + x/y;
```

משפט ה- try & catch ובלוק ה- finally

משפט ה- **catch** & **try** כולל בלוק **try** ובלוק אחד (או יותר) של **catch**. בבלוק ה- **try** ימוקמו הקריאות להפעלת המתודות שמתוכן עלול להיזרק **exception** (בין אם באמצעות **throw** ובין אם מתוך מתודה אחרת שמופעלת בתוכן). במשפט ה- **catch** חייב להיות פרמטר אחד מטיפוס המחלקה **Exception** או מחלקה אחרת אשר יורשת ממנה. זו יכולה להיות מחלקה ששייכת לקבוצת המחלקות שכבר קיימות בשפה, וזו גם יכולה להיות מחלקה חדשה שהוגדרה על ידי המתכנת.

אחרי בלוק ה- **catch** (או הבלוקים) ניתן למקם בלוק **finally**. הפקודות שבתוך בלוק ה- **finally** מתבצעות בכל מקרה. בין אם מתוך בלוק ה- try נזרק **exception** ובין אם לאו.

אחרי שמשפט **catch** מסוים מתבצע, התכנית ממשיכה להתבצע מהנקודה שאחרי בלוק ה- **catch** & **try** ואם קיים גם בלוק **finally** אז אחריו. כאשר בתוך בלוק ה-**catch** מופיעה אחת מהפקודות: **return**, **throw**, **break** או **continue** התכנית תבצע, תחילה, את בלוק ה-**finally** (אם קיים) אך לא תמשיך מהנקודה שאחריו.

לאחר שה-exception נזרק וטופל על ידי בלוק ה-catch המתאים, לא ניתן לחזור אל המקום שבו נוצרה הטעות, ולהמשיך משם.

אם exception שנזרק לא מטופל באף בלוק **catch** & **try**, לא במתודה שבתוכה הוא נזרק, ולא במתודה האחרת שממנה המתודה האמורה הופעלה, וגם לא באף מתודה אחרת אז ה-**default handler** שקיים ב- **JAVA** יטפל בו, ואז התכנית גם, בדרך כלל, תסתיים עם הודעת שגיאה על המסך.

מומלץ לתפוס במשפט **catch** אחד קבוצה של **exceptions** ולתת לכל אחד מהם את אותו הטיפול. כך ניתן לחסוך בקוד.

דוגמה לחריגים של משתמש:

```
public class PointException extends Exception{
      public PointException(String error) {
            super(error);
      }
}
import java.util.*;
/**
* this class represent a 2d point in the plane. <br>
* supports several operations on points in the plane.
*/
public final class Point {
// ******** private data members ********
      private double _x, _y;  // we "mark" data members using _.
// ********* constructors ********
   public Point (double x1, double y1){
      _x = x1;
      _y = y1;
   }
 /** copy constructor **/
   public Point (Point p){
      _x = p.x();
      _y = p.y();
   }
   // ********** public methodes *********
   public double x() {return _x;}
   public double y() {return _y;}
   /** @return a String contains the Point data*/
   public String toString()  {
      return "[" + _x + "," + _y+"]";
   }
//   this is a safe constructor:
   public Point(String str) throws PointException, NumberFormatException{
      if(str == null) throw new PointException(" PointException: Point is null!!!");
      StringTokenizer st = new StringTokenizer(str," [,]");
      int ct = st.countTokens();
      if(ct != 2) throw new PointException
            (" PointException: "+str+" Point has "+ct+" coordinates instead of 2  ");

      _x = (new Double(st.nextToken()).doubleValue());
      _y = (new Double(st.nextToken()).doubleValue());
}
}// class Point


public class TestPoint {
      public static void main(String[] args) {
            Point p1 = new Point(1.5, 2);
            Point p2;
            System.out.println("p1="+p1.toString());
            String param = "[1,3.5]";
            String param1 = "[a,3.5]";
            String param2 = "[10,3.5, 7]";
            String param3 = null;
            try {
                  p2 = new Point(param);
                  p2 = new Point(param1);
                  p2 = new Point(param2);
                  p2 = new Point(param3);
                  System.out.println("p2="+p2.toString());
            }
            catch (PointException ep)
            {
```

```
                System.err.println(ep);
                System.exit(-1);  // exit the java program!
            }
            catch (NumberFormatException e) {
                System.err.println(" the coordinate is not a number!! "+e);
                System.exit(-2);  // exit the java program!
            }
            // *** if any other 'unknown' exception happens.. ***
            catch (Exception e) {
                System.err.println("** unknown error !! **"+e);
                System.exit(-2);  // exit the java program!
            }
        }
}
```

עקרון "הכרז או טפל"

ב-JAVA לא ניתן להתעלם מקוד הזורק חריגה. לדוגמה, אם המתודה f1() קוראת למתודה f2(), וזו האחרונה מכריזה על אפשרות זריקת חריגה מסוג E1, אזי f1() חייבת לבצע אחת מהשניים:

1. (Declare) הכרזת על זריקת חריגה E1 על ידי משפט throws בכותרת המתודה.
2. (Handle) טיפול בחריגה E1 על ידי בלוק try ו-catch מתאימים.

<u>הערה</u>: קריאת למתודה העלולה לזרוק חריגה ללא ביצוע אחד מהשניים הוא שגיאת קומפילציה ב-JAVA.

Inheritance structure of the main exceptions: