Exceptions handling

טיפול בשגיאות ובחריגות

שגיאות

במהלך ריצת התכנית יכולות להיות שגיאות מסוגים שונים למשל:

- שגיאות של חריגה בזיכרון.
- שגיאה בהמרה.
- שגיאות מתמטיות.
- שגיאות קלט פלט.
- שגיאות בהרשאות.
- ועוד ועוד ועוד...

טיפול אפשרי בשגיאות

כאשר מתגלה שגיאה/חריגה ניתן:

- סיום התכנית – זוהי תגובה קיצונית מידי.
- החזרת ערך שגיאה(ערך חריג) – לא תמיד קיים ערך כזה.
- סימון שגיאה ע"י דגל גלובלי.
- קריאה לפונקציה **ייעודית** לטיפול בשגיאה.

מנגנון ה try - catch של ג׳אווה

- בג׳אווה קיים מנגנון מיוחד לטיפול בשגיאות באמצעות ״זריקת״ השגיאה במקום אחד ו״תפיסתה״ במקום אחר.
- כאשר פונקציה נתקלת בשגיאה מסוימת היא יכולה לזרוק אותה.
- הפונקציה הקוראת יכולה לתפוס את השגיאה או להמשיך לזרוק אותה הלאה.
- אם הפונקציה הראשית main תזרוק את השגיאה את התכנית תקרוס.

מנגנון ה try - catch (המשך)

- כל השגיאות יורשות ממחלקה אחת הנקראת Exception אשר יורשת ממחלקת Throwable.
- נוכל לרשת ממנה בעצמנו כדי לייצר שגיאה חדשה או לייצר instance שלה או של מחלקה היורשת ממנה.
- כאשר נרצה לזרוק שגיאה נשתמש במלה השמורה throw כאשר נרצה לתפוס שגיאה נייצר בלוק try - catch

try - catch example

```
public class StamException extends Exception {...} // מייצרים שגיאה מסוג חדש

public static void stam()  throws StamException{
        throw new StamException()
}

public static void main(String[] args) {
        try {
                stam();
        }catch(StamException ex) {
                System.out.println("Can't preform the action");
        }
}
```

כל פונקציה שזורקת שגיאה חייבת להכריז על כך

הפונקציה divide זורקת את השגיאה ואנו יכולים להמשיך לזרוק אותה (כאן זה היה מפיל את התכנית) ולהכריז על כך או לתפוס אותה ואז נקבל instance של השגיאה ונוכל לגשת לפרטים שלה.

ירושה ופולימורפיזם ב exceptions

כאשר נתפוס שגיאות נקפיד לתפוס ראשית את השגיאה הספציפית ביותר כלומר היורשת ורק לאחר מכן את המורישה. ובדוגמה שלנו :

```
public static void main(String[] args) {
    try {
        stam();
    }catch(StamException ex) {
        System.out.println("Can't do stam");
    }catch(Exception ex) {
        System.out.println("General error:  " + ex.getMessage())
    }
}
```

אם סדר התפיסה היה הפוך,
כלל לא היינו מגיעים לשגיאת
החלוקה, הכל היה "נתפס"
בתור instace של Exception

בלוק finally

- במקרים רבים בין אם נזרקה שגיאה או לא, נרצה לשחרר משאבים, לסיים תהליכים ולשמור ערכים מסוימים.

- לצורך כך קיימת המלה השמורה finally המשובצת בסיום בלוק ה try וה catch ואשר הבלוק שיבוא אחריה יתבצע בכל מקרה.

```
try {
...
} catch(AException ax){
….
} catch(BException bx){
…
} finally {
… // קטע הקוד הזה יתבצע בכל מקרה
}
```

Runtime Exceptions

נשים לב שישנן פונקציות אשר זורקות שגיאות שלא תפסנו או זרקנו הלאה:

- Integer.parseInt() אשר זורקת NumberFormatException
- כאשר ניגשנו למערך מבלי לתפוס את
  ArrayIndexOutOfBoundsException שיכול להיזרק
- ביצענו המרה של אובייקטים מבלי לתפוס את ClassCastException
- ניגשנו לשדות של אובייקטים מבלי לתפוס את NullPointerException

שאלה - RuntimeExceptions

- נשאלת השאלה כיצד זה יתכן?
- מדוע לא חויבנו על ידי המערכת לתפוס את השגיאות הללו כפי שחויבנו לתפוס כל שגיאה אחרת אשר יורשת מ Exception?
- נשים לב שכל השגיאות הללו יורשת ממחלקה אחת בשם RuntimeException אשר יורשת מ Exception.
- נשאלת שוב השאלה: מה מיוחד בה? מה משותף לכל השגיאות הללו?

תשובה - RuntimeExceptions

- מה שמשותף לכולם היא העובדה שהם יכולים להיזרק במהלך הפעולה הרגילה של ה JVM
- הם נקראים unchecked exceptions אשר לא צריך להכריז על זריקתם או לתפוס אותם וזאת בניגוד ל checked exceptions אשר אלו כל שאר המחלקות היורשות מ Execptions ושלא יורשות מ RuntimeException כדוגמת FileNotFoundExcption, IOException, ו InterruptedException.

תשובה (המשך)

- הרעיון כאן שכל השגיאות "הנורמליות" יכולות להימנע על ידי תכנות נכון! בין אם זה בדיקת גודל המערך או האובייקט לפני גישה אליהם, בדיקת קלט (שלא קיבלנו אפס או שהמחרוזת מכילה רק ספרות) ועוד. אנו יכולים להימנע מהן על ידי תכנות נכון!
- שגיאות אחרות כגון IOException יכולות להתרחש ללא תלות בתכנות שלנו כגון חוסר קישור לאינטרנט או קובץ פגום וכו' ולכן אנו מחויבים על ידי הג'אווה לתפוס שגיאות כאלו.

מסקנה

ג׳אווה יוצאת מתוך הנחה שאנחנו לא ״טיפשים״ ונותנת לנו חופש ביידיים

הבה לא נאכזב אותה!

בהצלחה!!!