# חריגות - Exceptions

## הגדרה

ב-Java קיים מנגנון של חריגות. מנגנון זה מאפשר לנו להחליט מה לעשות כאשר מישהו משתמש בתוכנית שלנו בצורה לא תקינה. נוכל לעצור את התוכנית ולזרוק למשתמש הודעה שמסבירה לו מדוע התוכנית לא יכולה לבצע מה שהוא מנסה לעשות, או שנוכל להבין מה הטעות שלו ולטפל בה בלי שהוא ישים לב. באותו אופן כאשר אנו משתמשים בקוד המובנה ב-Java או בספריות חיצוניות אנו עלולים ליצור שגיאות/חריגות. במקרה זה מישהו הגדיר מה לעשות עבור כל מצב כזה.

## סוגי חריגות

כאשר מתרחשת שגיאה/חריגה נוצר אובייקט מתאים המייצג שגיאה/חריגה זו. ישנה היררכיה של ממשקים ומחלקות שמטרתם ליצור אובייקטים אלו. בראש ההיררכיה נמצא ממשק Throwable המייצג אובייקט שניתן "לזרוק" אותו למשתמש ולסיים את התוכנית. הממשקים והמחלקות היורשות מ-Throwable מטרתם לייצג שגיאות וחריגות ספציפיות כדי שהמשתמש יוכל להבין מהי הבעיה. סוגי הבעיות הן:

1. **Errors** - אלו הן שגיאות שאינן תלויות במתכנת או במשתמש, כמו לדוגמא חוסר מקום בזיכרון או הצפה שלו. שגיאות אלו מתגלות בזמן ריצה. המתכנת אינו צריך לטפל בשגיאות אלו כי בדרך כלל אין לו מה לעשות נגדן.
2. **Checked Exceptions** - אלו הן חריגות שנבדקות בזמן הקומפילציה עוד לפני שהתוכנית מתחילה לרוץ. דוגמא לפקודות שיכולות לזרוק חריגות מסוג זה הן פקודות קלט ופלט על קבצים. כאשר הקומפיילר מזהה שפקודות מסוג זה קיימות בתוכנית, הוא בודק שהמתכנת אכן טיפל במקרים בהם יכולים להיזרק חריגות מסוג זה. לא ניתן להתעלם מהם, במידה והמתכנת לא טיפל בחריגות אלו התוכנית לא תקומפל ותיזרק חריגה מתאימה.
3. **Unchecked Exception** - אלו חריגות שאינן נבדקות בזמן קומפילציה אלא מתגלות בזמן ריצה, ולכן נקראות "חריגות זמן ריצה". חריגות אלו מתחלקות לשני סוגים: חריגות מתכנת וחריגות משתמש. חריגות מתכנת אלו הן חריגות שהמתכנת לא שם לב אליהן כמו: חריגות חישוביות, גישה לערך null, או גישה לתא לא מוגדר במערך. חריגות משתמש הן בדרך כלל חריגות קלט לא תקין.

ניתן גם ליצור חריגה על ידי הגדרת מחלקה שמממשת את ממשק Exception. לכל אובייקט מסוג Throwable יש שיטות שמוגדרות עליו, כמו הדפסה של מסלול החריגה, מחרוזת מייצגת את השגיאה וכו'. ניתן לקרוא עוד בקישור: <https://docs.oracle.com/javase/7/docs/api/java/lang/Throwable.html>.

## עקרון הכרז או טפל

כאשר תופסים חריגה ניתן לטפל בה בשני דרכים:

1. **להכריז** - לעצור את התוכנית ולזרוק את אובייקט החריגה המציין מה הייתה הבעיה, באיזה קובץ, ובאיזה שורה. דרך זו בעצם אומרת - זיהיתי שיש כאן בעיה אך זה לא האחריות שלי לטפל בה, אלא אני רק מודיע עליה, ומי שמקבל את ההודעה יחליט מה לעשות עם מידע זה.
2. **לטפל** - לא לעצור את התוכנית, אלא לטפל בחריגה. הטיפול בחריגה יכול להיות לנסות לבצע אותה פעולה שוב, או לבצע פעולות אחרות שמונעות את הבעיה. דרך זו אומרת - זיהיתי את הבעיה ואני אחראי לטפל בה כך שלא תפגע בתקינות של התכנית.

## זריקת חריגה

### throw

כאשר רוצים לזרוק חריגה נשתמש בפקודה throw Exception(), המשתמשת במילה השמורה throw ולאחריה מצביע לאובייקט מסוג Exception או מחלקה שיורשת מ-Exception. ניתן להוסיף כפרמטר גם מחרוזת String שמסבירה בפירוט על החריגה.

throw new ArithmeticException("can't divide by zero");

ביצועה של פקודה זו נעשה בדרך כלל כחלק ממשפט if, ובעצם הפעלתה נזרקת למעשה החריגה.

### throws

כאמור לעיל ישנם חריגות שאם הן נזרקות מחייבות אותנו לטפל בהם. אם מתוך שיטה כלשהי נזרקה חריגה מסוג זה, בין אם מתוך פקודה throw או שהופעלה שיטה אחרת שזורקת חריגה, אזי נוכל לטפל בחריגה בשתי דרכים. דרך אחת לשפל בחריגה כמו שנלמד בהמשך באמצעות בלוקים של try ו-catch. דרך שנייה להגדיר שחריגות מסוימות שיתפסו בשיטה זו מיד ייזרקו הלאה למי שהפעיל את השיטה. נוכל לעשות זאת על ידי הוספה בסוף הכותרת של השיטה (לאחר הפרמטרים) את המילה השמורה throws ולאחריה מצביע לאובייקט מסוג Exception או מחלקה שיורשת מ-Exception. לאחר פעולה זו אם תיתפס בשיטה כל חריגה מהסוג שהגדרנו בכותרת, או חריגה שיורשת מהסוג שהגדרנו בכותרת, אזי חריגה זו תיזרק אוטומטית הלאה. ניתן להגדיר מספר חריגות שהשיטה זורקת על ידי הוספת פסיק בין שמות החריגות.

public void f(int a) throws ArithmeticException, NullPointerException{…)

היתרונות של דרך זו הוא שכך הקוד קריא וברור יותר, וגם ניתן לתעד שהשיטה זורקת חריגה ב-javadoc כך שזה יהיה ברור יותר למשתמשים אחרים במחלקה. שיטה שבכותרתה יצוין שזורקת חריגה מסוג האב Exception תהיה מסוגלת לזרוק כל חריגה אפשרית

אם מבצעים overriding לשיטה שמגיעה בהורשה, לא ניתן לקבוע throws בשיטה החדשה שתזרוק שגיאות שלא הוגדרו בשיטה המקורית, אך ניתן להגדיר שזורקת חלק מהחריגות שהוגדרו בשיטה המקורית.

## טיפול בחריגה

טיפול בחריגה יעשה באמצעות המילים השמורות try, catch, ו-finally.

### try

זוהי מילה שמורה שלאחריה פותחים בלוק שבתוכו נמקם את שורות הקוד שיכולות לעורר ולזרוק חריגה. החריגה יכולה להיזרק או מתוך פונקציה שהקריאה להפעלתה ממוקמת בתוך אותו בלוק, או על ידי הפקודה throw שמופעלת בתוך אותו בלוק. המשמעות של פעולה זו היא שהקוד בתוך בלוק זה הוא מוגן. מערכת ההפעלה תפעיל קוד זה, וכאשר תזהה כי יש חריגה, מיד תעבור לבלוק של catch המתאים שיתמודד עם חריגה זו. לאחר שנזרקת חריגה וטיפלנו בה לא ניתן לחזור למקום שממנו נזרקה החריגה בבלוק ה-try.

try{

// Critical code

}

### catch

לאחר בלוק ה-try נכתוב בלוק catch שתופס חריגה מסוימת שנזרקה מבלוק ה-try. ניתן לכתוב כמה בלוקים של catch עבור כל סוג חריגה שייזרק מבלוק ה-try, כך שכל בלוק תופס חריגה שונה.

catch זוהי מילה שמורה שמקבלת פרמטר מסוג Exception או מחלקה שיורשת מ-Exception, בדומה לפרמטר שמכניסים לפונקציה. סוג החריגה בפרמטר הוא סוג החריגה שיתפוס בלוק ה-catch הנוכחי. לאחר כתיבת הפרמטר נפתח בלוק ובו נכתוב מה אנחנו רוצים שהתוכנית תעשה כדי להתמודד עם החריגה שנזרקה מבלוק ה-try. ניתן לבצע פעולות על אובייקט החריגה באמצעות השם שנתנו לפרמטר.

catch(FileNotFoundException ex) {…)

ניתן גם להגדיר שבלוק אחד יתפוס ויטפל בכמה סוגי חריגות באמצעות סימן ' | '.

catch(IOException|FileNotFoundException ex) {…)

אחרי שבלוק catch מתבצע, התכנית ממשיכה להתבצע מהנקודה שאחרי בלוק ה-catch (ואם יש בלוק finally אז אחריו), לא ניתן לחזור אל המקום שבו נוצרה החריגה בבלוק ה-try ולהמשיך משם. כאשר בתוך בלוק ה-catch מופיעה אחת מהפקודות: return, throw, break, continue, התכנית תבצע תחילה את בלוק ה-finally (אם קיים) אך לא תמשיך מהנקודה שאחריו.

אם חריגה שנזרקה לא מטופלת באף בלוק catch, אזי ב-default handler שקיים ב-Java יטפל בה, בדרך כלל התוכנית תסתיים עם הודעת שגיאה על המסך.

### finally

מילה שמורה אשר תופיע לאחר כל בלוקי ה-catch, ומיד אחריה בלוק של פקודות אשר יתבצעו בכל מקרה, בין אם נזרקה חריגה מבלוק ה-try או לא. השימוש העיקרי של בלוק זה הוא לבצע פקודות שמטרתן סגירת המשאבים של התכנית.

public static void main(String args[]) {

int a[] = new int[2];

**try** {

System.out.println("Access element three :" + a[3]);

} **catch** (ArrayIndexOutOfBoundsException e) {

System.out.println("Exception thrown :" + e);

} **finally** {

a[0] = 6;

System.out.println("First element value: " + a[0]);

System.out.println("The finally statement is executed");

}

}

## שימוש במשאבים

### Try with resources

# בדיקות - Junit