טיפול בשגיאות (Exceptions)

הקדמה

הדרך המקובלת לטיפול בשגיאות בשפות אחרות מתבססת בדרך כלל על הערכים שמחזירה הפונקציה. קובעים ערכים מוסכמים כאינדיקציה לשגיאות השונות שעלולות להתרחש, כך שאם הפונקציה מחזירה את אחד מהם ניתן לזהות את התרחשותה של תקלה ולטפל בה בהתאם (הערך המוחזר – על פיו ניתן לדעת אם התרחשה תקלה, וכמו כן, מהי התקלה שהתרחשה). כך למשל, הפונקציה malloc ב-C שאמורה להקצות שטח זיכרון באופן דינמי ולהחזיר את הכתובת שלו. אם היא לא מצליחה בפעולת ההקצאה אז היא מחזירה NULL. כשדרך פעולה זו ידועה מראש, ניתן לכתוב את התכנית כך, שאם הפונקציה מחזירה NULL אז יבוצעו פעולות מסוימות (כגון: מתן הודעה למשתמש). הטיפול בשגיאות בדרך זו לוקה בחסר. ראשית, אין אחידות בין תכנית לתכנית, ובין מתכנת למתכנת. כך למשל, מתכנת A יכול לקבוע כי הפונקציה שכתב לקריאה מקובץ מחזירה –2 במקרה של כישלון, ומתכנת B יכול לקבוע כי הפונקציה שכתב לקריאה מקובץ מחזירה -1 במקרה של כישלון. שנית, בטיפול בשגיאות שמתבסס על הערכים המוחזרים מפונקציות, יש קושי בהעמסת אינפורמציה נוספת בנוסף לעצם התרחשותה של התקלה. כך למשל, פונקציה שמחזירה –1 במקרה שלא הצליחה בפעולתה, מעבר לכך שהיא לא הצליחה בפעולתה קשה יהיה לקבל אינפורמציה נוספת, כגון: תיאור מילולי ספציפי של הכשל שהתרחש. חיסרון נוסף קיים בכך שלעתים לא קיימת אפשרות לקבוע ערך מסוים כאינדיקטור להתרחשותה של שגיאה. לעתים טווח הערכים האפשריים שמתודה יכולה להחזיר כה גדול שלא קיימת האפשרות לקבוע ערך מסוים בתור הערך שהחזרתו תהווה אינדיקציה לתקלה. למרות חסרונות אלה, ניתן למצוא ב-JAVA מספר מתודות שנוקטות בדרך זו (לדוגמא: המתודה getParameter במחלקה applet אשר מחזירה null כאשר היא לא מוצאת ערך לפרמטר שאת שמו קיבלה).

ב-JAVA עיקר הטיפול בשגיאות/תקלות נעשה בדרך אחרת: הטיפול בשגיאות/תקלות ב-JAVA דומה לאופן הטיפול בשגיאות/תקלות ב C++. הטיפול בשגיאות/תקלות ב-JAVA כולל מכניזם של דיווח על התרחשותה של השגיאה/תקלה, ומכניזם שמטפל בה (משמע, מבצע פעולות מסוימות שיש לבצע בגלל התרחשות השגיאה/תקלה). גם המכניזם הראשון וגם השני מתבצעים באופן עצמאי, ומייד עם התרחשות השגיאה/תקלה.

עם היווצרות השגיאה/תקלה נוצר אובייקט מטיפוס מחלקה שיורשת מהמחלקה Throwable (או מטיפוס מחלקה אחרת שיורשת מ- Throwable). היווצרות האובייקט יכולה להתרחש בתוך מתודה של מחלקה שכבר נכתבה עבורנו, או באופן יזום בתוך מתודה שאנו הגדרנו. כיוון שרוב המחלקות, שמתארות אובייקט אשר נזרק מתוך תהליך בגלל מצב של תקלה/ שגיאה, יורשות מהמחלקה Exception (מחלקה אשר יורשת מהמחלקה Throwable) לאובייקט שנוצר ונזרק נקרא בשם exception. 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.

מהם סוגי השגיאות הקיימות ?

שגיאות בקלט מהמשתמש

לעתים מכניס המשתמש לתכנית נתונים לא תקינים: כתובת שלא קיימת ב-Internet, ערך מספרי עם נקודה עשרונית בעוד שהתכנית מצפה לקבל מספר שלם.

תקלות חומרה

לעתים רכיבי החומרה שעימם התכנית עובדת לא פועלים באופן תקין מסיבות שונות. דוגמאות: נייר שנגמר במדפסת, עכבר שהתנתק וכדומה... במקרים כאלה רכיב החומרה לא יכול לבצע את הפעולה שתכנית מצפה שהוא יעשה. לעתים התכנית נתקלת בבעיות לא צפויות שמקורן במגבלות של החומרה. דוגמאות: דיסק שהתמלא כך שלא ניתן לכתוב אליו, בעיות זיכרון...

שגיאות בפעולתן של מתודות בתכנית

אלה הם מצבי תקלה שהוגדרו על ידי המתכנת או שהוגדרו במחלקות שקיימות כבר כחלק מהשפה. הכוונה בקבוצת שגיאות אלה היא לכל אותם מצבים שבהם המתודה לא פעלה באופן שבו אנו מצפים שתפעל. דוגמאות: ניסיון גישה למקום במערך שמעבר לטווח האינדקסים שלו, ניסיון להוציא ממחסנית (STACK) איבר כאשר המחסנית ריקה, ניסיון לבצע פעולת חישוב מתמטית והתקלות בשגיאה (ניסיון לחשב שורש של מספר שלילי למשל) וכדומה.

כאשר מתרחשת שגיאה/תקלה אופן הטיפול המקובל הוא אחד מהשניים: התכנית תמשיך לפעול, ותאפשר למשתמש

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

כעת אסקור בקצרה את הפקודות ואת המלים השמורות שאציג בהרחבה בהמשכו של הפרק. בתום הסקירה תופיע דוגמא פשוטה לאופן השימוש במלים השמורות האלה.

throw

זוהי פקודה שיש לרשום בצירוף reference לאובייקט חדש 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 בכותרת 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 מתכנתים אחרים יוכלו 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- reference 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 reference לאובייקט מטיפוס Exception (או מחלקה אחרת שיורשת מ- Exception), ובלבד שה- exception שייזרק יהיה מטיפוס המחלקה שמופיעה בסוגריים של ה-catch (או שמטיפוס מחלקה שיורשת ממנה). לשורה שמתחילה במילה catch נקרא בשם משפט ה-catch.

finally

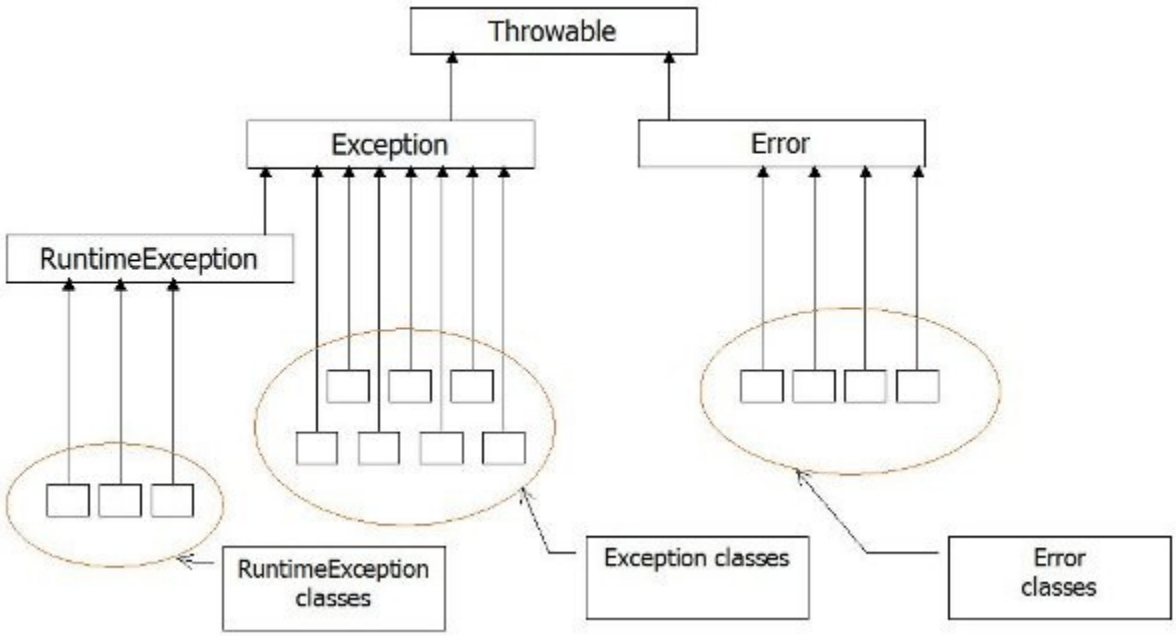
מילה שמורה אשר תופיע אחרי בלוק ה- catch (ואם יש יותר מ- catch אחד, אז אחרי כולם), ומייד אחריה בלוק של פקודות אשר יתבצעו בכל מקרה. בין אם נזרק exception או לא. בבלוק ה- finally נמקם, בעיקר, פקודות לסגירת משאבים של התכנית.

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

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

סוגי השגיאות הקיימים

שגיאות שמתרחשות בזמן ריצת התכנית מתוארות באמצעות אובייקטים שנוצרים ממחלקות שכבר הוגדרו (מהוות חלק מהמחלקות של השפה) או ממחלקות חדשות שהגדרנו כמחלקות שיורשות מאותן מחלקות קיימות. את המחלקות הקיימות ניתן לסווג לשלוש קבוצות עיקריות.



הקבוצה הראשונה (Error Classes) כוללת את כל המחלקות שיורשות באופן ישיר או עקיף מהמחלקה Error. אובייקטים שנוצרים ממחלקות אלה מתארים בדרך כלל תקלות ובעיות יסודיות שהדרך להתגבר עליהן איננה באמצעות מתן טיפול בתקלה בזמן ריצה כי אם באמצעות בדיקת הסביבה שבה אנו מריצים לרבות ה-JVM ואופן הפעלתו. בדרך כלל מדובר בבעיות שנובעות מבעיה בהפעלת ה-JVM או הפלטפורמה בה אנו משתמשים. כך למשל, בעיות זיכרון ובעיות שנובעות מבאג ב-JVM יתוארו באמצעות אובייקטים שיווצרו ממחלקות אלה.

הקבוצה השניה (RuntimeException Classes) כוללת את כל המחלקות שיורשות באופן ישיר או עקיף מהמחלקה RuntimeException. אובייקטים שיווצרו ממחלקות אלה מתארים תקלות שמקורן בדרך כלל בשגיאות בקוד. כך למשל, חריגה מגבולות של מערך תתואר באמצעות אובייקט שיווצר מהמחלקה ArrayIndexOutOfBoundsException, מחלקה ששייכת לקבוצה זו. כך גם ניסיון לקרוא להפעלת מתודה באמצעות משתנה שמכיל null במקום להכיל reference לאובייקט אשר יגרום להיווצרות תקלה שתתואר באמצעות אובייקט שיווצר מהמחלקה NullPointerException, מחלקה שגם היא שייכת לקבוצה זו. בדרך כלל אנו נימנע מניסיון לטפל באמצעות try&catch בתקלות מסוג RunttimeException ונעדיף למצוא את הטעות בקוד ולתקן אותה.

הקבוצה השלישית (Exception Classes) כוללת את כל המחלקות שיורשות באופן ישיר או עקיף מהמחלקה Exception מבלי לרשת מהמחלקה RuntimeException. אובייקטים שיווצרו ממחלקות אלה מתארים תקלות שמקורן בדרך כלל בנסיבות שלא בשליטתנו. כך למשל המחלקה SQLException שאובייקט שנוצר ממנה מתאר תקלה שמתרחשת בבסיס הנתונים שעימו אנו עובדים. דוגמא נוספת היא המחלקה IOException שאובייקט שנוצר ממנה מתאר תקלה בקלט/פלט של התכנית אשר נובעת מהסביבה בה היא רצה (למשל: כבל תקשורת שהתנתק). תקלות שמתוארות באמצעות אובייקטים שנוצרים ממחלקות ששייכות לקבוצה זו הן התקלות שבהן אנו בדרך כלל נרצה לטפל בגוף התכנית באמצעות המנגנון try & catch. אלו תקלות שעבורן נרצה להוסיף לקוד המקור שלנו קוד שיתבצע במידה שהן תתרחשנה כך שבתגובה להתרחשותן התכנית תוכל, למשל, להציג הודעה למשתמש.

הפקודה throw

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

throw *reference to Exception instance*

לדוגמא:

```
if (a < 100)
{
      throw new MyException();
}
```

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. בהמשך נציג הסבר מפורט יותר.

המילה השמורה throws

אם בתוך מתודה מתבצעת פעולת זריקה של exception (בין אם באמצעות הפקודה throw ובין אם מתוך מתודה אחרת שהופעלה) אז ניתן להוסיף לכותרת של המתודה שאנו מגדירים את המילה השמורה throws בצירוף ה-type של ה-exception שעלול להיזרק. בדרך זו אנו מיידעים מתכנתים אחרים שרוצים להשתמש במתודה שלנו בכך שמהפעלתה עלול להיזרק exception מסוג מסויים. ניתן גם לפרט יותר מסוג אחד של exception אחרי המילה throws (יש להפריד ביניהם באמצעות פסיקים). בהמשך נראה שבמצבים מסויימים חייבים להשתמש במילה השמורה throws.

לדוגמא:

```
public int getValue() throws MyException
{
            if (num<20)
                throw new MyException();
            else
                return 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.

לדוגמא:

```
public int getSize() throws MyException, InternetException
{
            . . .
}
```

אין כל מגבלה למספר סוגי ה- exceptions שיכולים להיזרק. אם מבצעים overriding למתודה שמגיעה בהורשה אז כאשר מדובר ב-exceptions מהסוג השלישי (מחלקות שיורשות באופן ישיר או עקיף מהמחלקה Exception מבלי לרשת מהמחלקה RunttimeException) לא ניתן לקבוע במתודה החדשה באמצעות throws כי היא מסוגלת לזרוק exceptions שלא צוינו כניתנים לזריקה על ידי המתודה המקורית או שאינם יורשים מהסוג/ים שמצויין/נים בה.

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 void display() 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.

משפט ה- try & catch

משפט ה- try & catch 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).

try

{

פקודות שבפעולתן עלולה להיווצר תקלה

}

אחרי בלוק ה- try ימוקם בלוק catch אחד או יותר.

```
catch (Exception e)
{
```

פקודות שיבוצעו אם הייתה תקלה שנתפסה

```
}
```

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.

finally

{

פקודות שתתבצענה בכל מקרה

}

משפט ה- catch

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.

הפרמטר שבמשפט ה-catch קולט לתוכו reference לאובייקט שמתאר שגיאה/תקלה ושנזרק מתוך בלוק ה-try. אובייקט זה יכול להיות מטיפוס שזהה לטיפוס הפרמטר או מטיפוס של מחלקה שיורשת מהמחלקה שהפרמטר מטיפוסה.

ה- reference לאובייקט ה-exception מגיע אל משפט ה-catch לאחר שהוא נזרק מבלוק ה- try. כאשר יש יותר מבלוק catch אחד, אז ה- exception שנזרק ייתפס על ידי בלוק ה-catch הראשון שיכול לתפוס אותו (כלומר, על ידי משפט ה-catch הראשון שהפרמטר שלו יכול לקלוט לתוכו את ה-reference של ה- exception שנזרק), ובלוק catch זה הוא גם בלוק ה-catch היחידי שיפעל מבין כל בלוקי ה-catch.

אם אין אף משפט catch מתאים שיכול לתפוס את ה-exception, אז אובייקט ה-exception יועבר אל המקום שממנו הופעלה המתודה. במקרה כזה, אובייקט ה-exception מועבר רק לאחר שבלוק ה- finally מתבצע.

אחרי שמשפט catch מסוים מתבצע, התכנית ממשיכה להתבצע מהנקודה שאחרי בלוק ה- try & catch 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 מהפקודה 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 try & catch, 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. כך לדוגמא, המחלקה IOException מהווה בסיס למחלקות רבות אחרות, אשר מתארות (כל אחת מהן) שגיאות/תקלות שקשורות בקלט/פלט של נתונים. כתיבת משפט catch לתפיסת exceptions מטיפוס IOException, ישמש, למעשה, לתפיסת exceptions שמטיפוס כל אחת מהמחלקות שיורשות מ- IOException.

בתוך משפט ה-catch ניתן למקם פקודת throw ובכך להמשיך את הטיפול בשגיאה/בתקלה גם במקום אחר.

לדוגמא:

```
try
{
            myServer.show();
}
catch(IOException e)
{
            System.out.println("IOException has occurred !");
            throw e;
}
```

בלוק ה-finally

כאופציה, ניתן למקם אחרי בלוק ה-catch (או הבלוקים) את בלוק ה-finally. בלוק זה יכלול את שורות הקוד שחייבות להתבצע בכל מקרה: בין אם נזרק exception ובין אם לא. אם בלוק ה-catch תופס את אובייקט ה-exception שנזרק, אז תחילה יפעלו הפקודות שבתוך אותו בלוק, ורק אחר כך יופעלו הפקודות שבתוך בלוק ה-finally. גם אם אובייקט ה-exception שנזרק לא נתפס על ידי אף בלוק catch גם אז: בלוק ה-finally יתבצע. בלוק ה-finally מבוצע תמיד. בין אם נזרק exception ובין אם לא. גם אם בתוך בלוק ה-catch תבוצענה אחת הפקודות: throw, return, continue או break בלוק ה-finally יתבצע. המקרה היחידי שבו בלוק ה-finally לא יתבצע הוא כאשר מתוך בלוק ה-try או מתוך בלוק ה-catch תופעל המתודה: System.exit(0). מתודה זו תגרום לסיומה של התכנית מבלי שבלוק ה-finally יתבצע. נהוג לכלול בתוך בלוק ה-finally את כל הפעולות שהכרחי לבצע בכל מקרה, בין אם נזרק exception ובין אם לא, כגון: סגירת קבצים, סגירת אפיקי תקשורת וכדומה.

```
Try
{
      ...
}
catch(IO Exception e)
{
      ...
}
finally
{
      ...
}
```

הכלל "טפל או הצהר"

עבור כל שורת קוד שעלול להיזרק ממנה exception מהסוג השלישי (exception שהוא אובייקט שנוצר מאחת המחלקות שיורשות מהמחלקה Exception מבלי לרשת מהמחלקה RuntimeException) אנו חייבים לבחור לעשות את אחת משתי הפעולות הבאות:

<u>אפשרות ראשונה</u>

למקם בלוק try & catch סביב שורת הקוד האמורה כדי לתפוס את ה-exception האמור. בכך, למעשה, אנו בוחרים לטפל ב-exception שעלול להיזרק בעצמנו.

<u>אפשרות שניה</u>

להוסיף לכותרת של המתודה שכעת אנו מגדירים את המילה השמורה throws בתוספת סוג ה-exception האמור. בכך, למעשה, אנו בוחרים לא לטפל ב-exception שעלול להיזרק בעצמנו ולהעביר את האחריות למי שכותב את הקוד אשר קורא להפעלת המתודה שכעת אנחנו מגדירים.

קטע קוד שלא מציית לכלל האמור לא עובר קומפילציה.

דוגמא מסכמת

בדוגמא לעיל, אנו מגדירים מחלקה חדשה לתיאור תקלה ייחודית לתכנית שאנו כותבים. המחלקה החדשה יורשת ישירות מהמחלקה Exception ובכך כל אובייקט שיווצר ממנה כדי לתאר את התרחשותה של התקלה הייחודית האמורה יהיה חייב להיתפס בבלוק try&catch מתאים (בין אם זה יהיה ב-try&catch שאנו נכתוב או ב-try&catch שמתכנת אחר יכתוב). כיוון שמחלקה החדשה יורשת ישירות מהמחלקה Exception היא שייכת לקבוצה השלישית של תקלות שהכלל "טפל או הצהר" חל עליה.

```
public class MyException extends Exception
{
      MyException(String msg)
      {
            super(msg);
      }
}
```

כדאי לשים לב לשימוש ב-`super` בתוך ה-`constructor` שאנו מגדירים במחלקה החדשה `MyException` על מנת לגרום להודעת הטקסט להגיע אל תוך המשתנה `message` שבתוך האובייקט החדש.

קטע הקוד הבא עושה שימוש במחלקה `MyException` כדי לתאר תקלה שנגרמת כתוצאה מכך שערך שלילי נשלח למתודה. יש לשים לב לכך שהדוגמא הבאה נכתבה לצרכים לימודיים בלבד. כאשר מפתחים קוד יש להימנע משימוש במנגנון הטיפול ב-`exceptions` באמצעות בלוק `try & catch` ולהעדיף בדיקות שימנעו את היווצרות התקלה (במקרה זה לדוגמא ניתן היה לבדוק את הערך הנשלח למתודה ובמידה שהוא שלילי לא לקרוא להפעלתה).

```
public class MyExceptionHandlingDemo
{
      public static void main(String args[])
      {
            int val = 123;
            try
            {
                  aa(val);
            }
            catch(MyException e)
            {
                  e.printStackTrace();
            }
            finally
            {
                  System.out.println("finally always works");
            }
            val = -123;
            try
            {
```

```
            aa(val);
      }
      catch(MyException e)
      {
            e.printStackTrace();
      }
      finally
      {
            System.out.println("finally always works");
      }
}

static void aa(int num) throws MyException
{
      System.out.println("aa start");
      bb(num);
      System.out.println("aa end");
}

static void bb(int val) throws MyException
```

```
	{
		System.out.println("bb start");
		cc(val);
		System.out.println("bb end");
	}

	static void cc(int number) throws MyException
	{
		System.out.println("cc start");
		if (number<0)
		{
			throw new MyException("negative number");
		}
		else
		{
			System.out.println(number + " log is " + Math.log(number));
		}
		System.out.println("cc end");
	}
}
```