

## חריגים וטענות Exceptions and Assertions

- **חריגים (Exceptions)** מבטאים מצבים בלתי צפויים בריצת התוכנית כגון: ארגומנטים שאינם חוקיים, בעיות ברשת התקשורת, קובץ שאינו קיים וכדומה
- טענות (**Assertions**) – מבטאות הנחות שיש למתכנת בנקודה מסויימת בקוד. הטענות מבטאות על דרך החיוב
- בזמן ריצה, ניתן להסיר את הטענות מן הקוד לחלוטין ובכך לא להאט את ריצת התוכנית

## תכנות מתקדם בשפת Java חריגים ושגיאות (exceptions)

אוהד ברזילי  
אוניברסיטת תל אביב

## דוגמה

```
public class AddArguments {
    public static void main(String args[]) {
        int sum = 0;
        for (String arg : args) {
            sum += Integer.parseInt(arg);
        }
        System.out.println("Sum = " + sum);
    }
}
```

```
> java AddArguments 1 2 3 4
Sum = 10
```

```
> java AddArguments 1 two 3.0 4
Exception in thread "main" java.lang.NumberFormatException: For input string: "two"
at java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
at java.lang.Integer.parseInt(Integer.java:447)
at java.lang.Integer.parseInt(Integer.java:497)
at AddArguments.main(AddArguments.java:5)
```

## חריגים

- תנאים אשר עשויים להתקיים במהלך ריצת תוכנית תקינה נקראים *checked exceptions*
- תנאים אלו מיוצגים ע"י המחלקה `Exception`
- בעיות חמורות הנחשבות קטלניות (*fatal*), וכן מצבים המייצגים שגיאות בתוכנית (bugs) נקראים *unchecked exceptions*
- בעיות חמורות מיוצגות ע"י המחלקה `Error`
- שגיאות בתוכנית מיוצגות ע"י המחלקה `RuntimeException`
- תיעוד המחלקות האוטומטי (javadoc API) מתאר עבור כל מתודה את ה- *checked exceptions* שהיא עשויה לחולל

## גרעיניות

```
public class AddArguments3 {
    public static void main(String args[]) {
        int sum = 0;
        for (String arg : args) {
            try {
                sum += Integer.parseInt(arg);
            } catch (NumberFormatException nfe) {
                System.err.println("[ " + arg + " ] is not an integer"
                    + " and will not be included in the sum.");
            }
        }
        System.out.println("Sum = " + sum);
    }
}
```

```
> java AddArguments3 1 two 3.0 4
[two] is not an integer and will not be included in the sum.
[3.0] is not an integer and will not be included in the sum.
Sum = 5
```

## try-catch block

```
public class AddArguments {
    public static void main(String args[]) {
        try {
            int sum = 0;
            for (String arg : args) {
                sum += Integer.parseInt(arg);
            }
            System.out.println("Sum = " + sum);
        } catch (NumberFormatException nfe) {
            System.err.println("One of the command-line "
                + "arguments is not an integer.");
        }
    }
}
```

```
> java AddArguments2 1 two 3.0 4
One of the command-line arguments is not an integer.
```

## מחסנית הקריאות

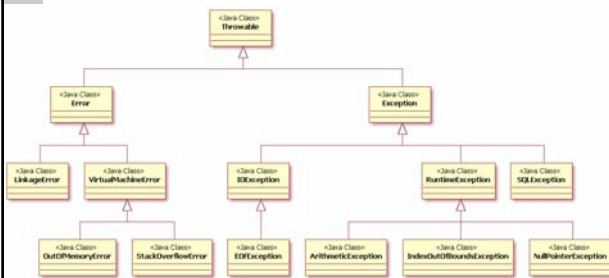
- אם חריג (exception) אינו מטופל ע"י בלוק catch במתודה כלשהי, הוא נזרק למתודה שקראה לאותה מתודה
- אם חריג היגיע אל המתודה main וגם שם הוא אינו מטופל התוכנית מסתיימת

## ריבוי בלוקי catch

- לבלוק try אחד עשויים להיות כמה בלוקים של catch השייכים לו, עבור סוגים שונים של שגיאות שעשויות לקרות:

```
try {  
    // code that might throw one or more exceptions  
} catch (MyException e1) {  
    // code to execute if a MyException exception is thrown  
} catch (MyOtherException e1) {  
    // code to execute if a MyOtherException exception is thrown  
} catch (Exception e3) {  
    // code to execute if any other exception is thrown  
}
```

## היררכיית שגיאות וחריגים (חלקית)



## בלוק finally

- קטע קוד המופיע בבלוק finally יתבצע בכל מקרה (בין אם קטע הקוד בבלוק ה try הצליח או נכשל)

```
try {  
    startFaucet();  
    waterLawn();  
} catch (BrokenPipeException e) {  
    logProblem(e);  
} finally {  
    stopFaucet();  
}
```

## הצהירי או טפלי

- אי הכרזה או טיפול גורר שגיאת קומפילציה

```
void f(int x) throws Exception {...}  
  
void g() {  
    f(1); // error  
}
```

**Compilation Error:** programmer must either catch the exception, or declare that "void g() throws Exception"

## הצהירי או טפלי

- קוד המכיל קריאה למתודה שעשויה לחולל (לזרוק) חריג נבדק (checked) צריך לנקוט אחת משתי הגישות: הכרזה על החריג הפוטנציאלי או טיפול בו

- טיפול ראיו, נעטוף את הבלוק הבעייתי בבלוק try-catch-finally

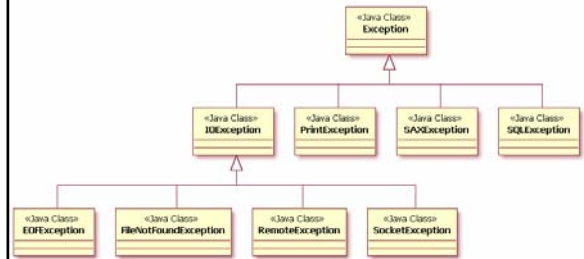
- הכרזה – נשתמש במילה השמורה throws כדי לציין את המתודה העוטפת כולה כ"בעייתית":  

```
void trouble() throws IOException { ... }  
void trouble() throws IOException, MyException { ... }
```

## unchecked exceptions

- על חריגים או שגיאות שהם unchecked אין חובה להצהיר בחתימת המתודה
- בחריגים או שגיאות שהם unchecked אין חובה לטפל בבלוק try-catch-finally
- למה?

## checked exceptions (רשימה חלקית)



## חריגים וירושה

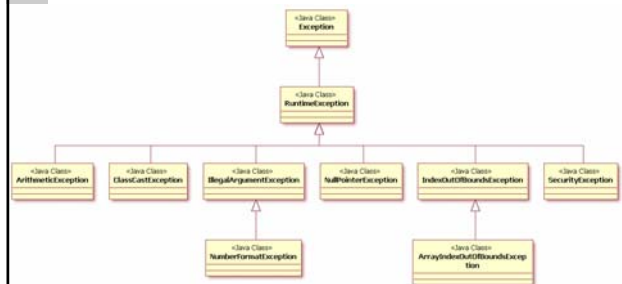
למתודה דורסת מותר לזרוק:

- אף חריג
- חריגים שזרקה המתודה הנדרסת
- חריגים היורשים מחריגים שזרקה המתודה הנדרסת

למתודה דורסת אסור לזרוק:

- חריגים שלא זרקה המתודה הנדרסת
- חריגים המהווים מחלקות בסיס לחריגים שזרקה המתודה הנדרסת

## unchecked exceptions (רשימה חלקית)



## ריבוי בלוקי catch

- אם יש כמה פסוקי catch מתאימים יתבצע הבלוק המתאים הראשון (לא בהכרח המתאים ביותר)

```
void readData() throws SQLException, IOException {  
    ...  
}  
  
void test() {  
    try {  
        readData();  
    }  
    catch (SQLException e) { ... }  
    catch (IOException e) { ... }  
}
```

Executed on SQLException

Executed on IOException

## חריגים וירושה - דוגמה

```
public class TestA {  
    public void methodA() throws IOException {  
        // do some file manipulation  
    }  
}  
  
public class TestB1 extends TestA {  
    public void methodA() throws EOFException { // OK  
        // do some file manipulation  
    }  
}  
  
public class TestB2 extends TestA {  
    public void methodA() throws Exception { // WRONG  
        // do some file manipulation  
    }  
}
```

## הגדרת חריגי משתמש

```
public class ServerTimeoutException extends Exception {
    private int port;
    public ServerTimeoutException(String message, int port) {
        super(message);
        this.port = port;
    }

    public int getPort() {
        return port;
    }
}
```

- חריגי משתמש בדרך כלל רזים מאוד
- המתודה getMessage המוגדרת ב Exception משמשת לקבל מחרוזת המתארת את מהות השגיאה

## ריבוי בלוקי catch

```
void readData() throws SQLException, EOFException {
    ...
}

void test() {
    try {
        readData();
        Thread.sleep(100);
    }
    catch (SQLException e) { ... }
    catch (IOException e) { ... }
    catch (Throwable e) { ... }
}
```

Throws  
InterruptedException

Reminder:  
EOFException extends  
IOException

Executed on SQLException

Executed on EOFException

Executed on InterruptedException,  
NullPointerException ...

## טיפול בחריגי משתמש

- מתודה המשתמשת ב connectMe מהשקף הקודם עשויה להתמודד עם החריג ע"י החלפת שרת וניסיון חוזר בעזרת בלוק try-catch

```
public void findServer() {
    try {
        connectMe(defaultServer);
    } catch (ServerTimeoutException e) {
        System.out.println("Server timed out, trying alternative");
        try {
            connectMe(alternativeServer);
        } catch (ServerTimeoutException e1) {
            System.out.println("Error: " + e1.getMessage() +
                " connecting to port " + e1.getPort());
        }
    }
}
```

## זריקת חריג

- מתודה עשויה לזרוק חריגים סטנדרטים או חריגים שהוגדרו ע"י משתמש

```
public void connectMe(String serverName) throws ServerTimeoutException {
    boolean successful;
    int portToConnect = 80;
    successful = open(serverName, portToConnect);
    if (!successful) {
        throw new ServerTimeoutException("Could not connect", portToConnect);
    }
}
```

## טענות (assertions)

- תחביר:

```
assert <boolean_expression>;
assert <boolean_expression> : <detail_expression>;
```

- אם הביטוי boolean\_expression משתערך ל false התוכנית זורקת AssertionError

- הביטוי detail\_expression הופך למחרוזת לתיאור מהות השגיאה

## הדפסת מחסנית הקריאות

- אחת המתודות השימושיות של המחלקה Throwable היא המתודה printStackTrace המדפיסה את שרשרת הקריאות שהובילה לחריג:

```
public static void main(String[] args) {
    try {
        riskyMethod();
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
    System.out.println("Continuing main ...");
}
```

## שמורה פנימית

```
if (x > 0) {
    // do this
} else {
    // do that
}
```

אבל אם ידוע ש x אינו יכול להיות שלילי, עדיף:

```
if (x > 0) {
    // do this
} else {
    assert ( x == 0 );
    // do that, unless x is negative
}
```

## דוגמאות שימוש

■ טענות מבטאות הנחות שיש למתכנת על הלוגיקה הפנימית בקטע קוד מסוים

■ לדוגמא:

- שמורה פנימית
- שמורת מבני בקרה (control flow invariant)
- שמורת מחלקה ותנאי בתר

## שמורת מחלקה ותנאי בתר

■ כלים אשר אופקים טענות Design by Contract עשויים לחולל קוד ש"ישתל" בגוף התוכנית במקום המתאים ( code instrumentation)

■ לדוגמא, הקוד הבא:

```
/** @post getValue() == newValue, "value is updated" */
public void setValue(int newValue) {
    val = newValue;
}
```

■ יהפוך ל:

```
public void setValue(int newValue) {
    val = newValue;
    assert(getValue() == newValue : "value is updated");
}
```

## שמורת מבני בקרה

```
switch (suit) {
    case Suit.CLUBS: // ...
        break;
    case Suit.DIAMONDS: // ...
        break;
    case Suit.HEARTS: // ...
        break;
    case Suit.SPADES: // ...
        break;
    default: assert false : "Unknown playing card suit";
        break;
}
```

## טענות וביצועים

■ כבירת מחדל אכיפת הטענות אינה מאפשרת יש לאפשר זאת במפורש:

```
> java -enableassertions MyProgram
```

1 א

```
> java -ea MyProgram
```

■ ניתן לשלוט באכיפת טענות עבור מחלקה מסוימת, חבילה או היררכיית חבילות. הפרטים המלאים:

<http://java.sun.com/j2se/1.5.0/docs/guide/language/assert.html>