



Java lecturer 14

אז על מה נדבר היום...


- Packages
- Inner Classes
- Throws Exception

Packages

חבילות הם הדרך שלנו לרכז קטעי קוד בצורה היררכית ומסודרת כדי שיהיה לנו קל לגשת אליהם ממקומות שונים בקוד. ישנם חבילות שמגיעות ביחד עם קוד המקור של Java (תוכלו לראות אותם תחת התיקיה External Libraries) ישנם חבילות (ספריות) שאנחנו יכולים לייבא ממקומות אחסון שונים ברחבי האינטרנט בדרך כלל על ידי הכלי העיקרי שכרגע לא נכנס ל-איך הוא עובד, רק נזכור את השם שלו Maven. במשפט אחד: זה קובץ שנותנים לו את הכתובת של הקוד מקור והוא יודע לשמור את זה אצלינו בספריה כדי שנוכל להשתמש באותם קטעי קוד. עובדת בונים: מקור השם הוא בעברית "מבין" שבהטייה ליידיש זה "מייבין", ביטוי למישהו שהוא מבין בתחום.

Definition of maven noun from the Oxford Advanced Learner's Dictionary

maven *noun*

BrE /'meɪvn/ ; NAmE /'meɪvn/ 
(North American English)

★ an expert on something

+ Oxford Collocations Dictionary

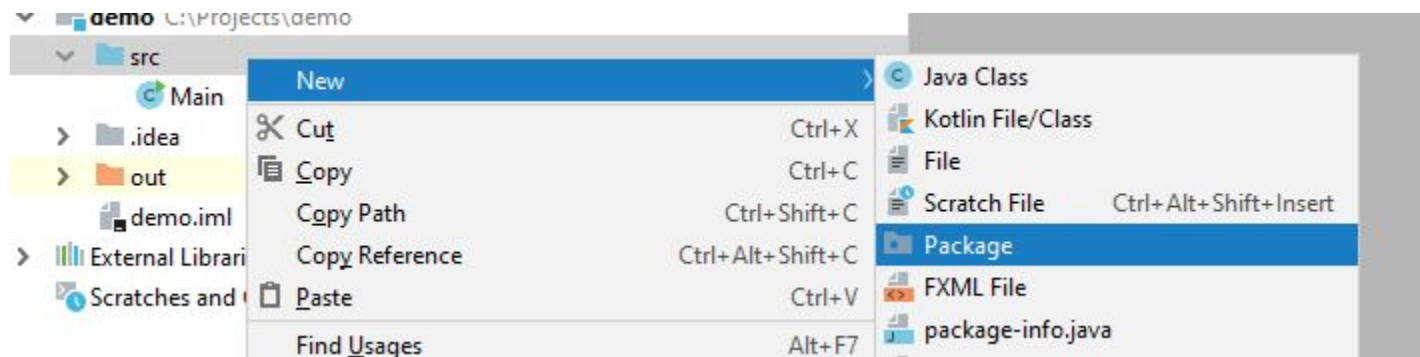
— Word Origin

1960s: Yiddish.

Packages

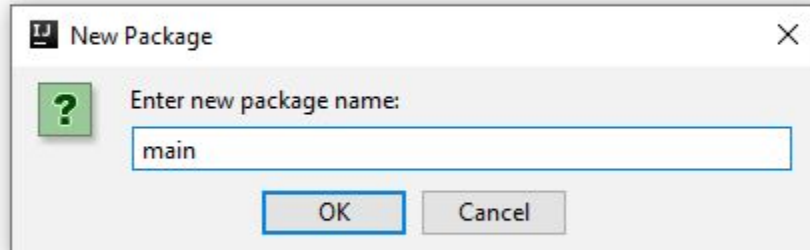
בנוסף למה שראינו, גם אנחנו יכולים לרכז קבצים בחבילות.
אז אם עד היום שמרנו את כל הקבצים שלנו בתיקייה אחת בשם src, הגיע הזמן לעשות קצת סדר ולנהל עוד קצת הרשאות.
לתיקיה אנחנו קוראים Package (חבילה)
הדרך ליצור חבילה היא פשוטה:

1. לחצן ימני בעכבר על התיקייה שבה אנחנו רוצים לייצר את החבילה.
2. New
3. Package



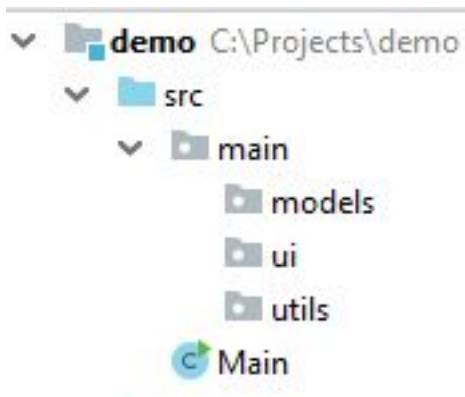
Packages

בחלון שיפתח לנו נכתוב את שם החבילה.
מוסכמה: שמות של חבילות יתחילו באות קטנה.
דבר ראשון בתחילת כל פרוייקט נצטרך ליצור חבילה ראשית שתכיל בעצמה את שאר החבילות.
לחבילה הראשית נקרא...



Packages

אחרי שייצרנו חבילה ראשית, נוכל לייצר עוד תתי חבילות עבור נושאים שקשורים אחד לשני.
למשל חבילה עבור כל הקבצים שמטפלים בתצוגה.
או למשל חבילה עבור כל המודלים שייצרנו
ובתוך החבילה של המודלים אפשר לייצר עוד חבילה שמכילה לדוגמא את כל החיות וכן הלאה.
בדרך כלל את ה Main שמריץ את התוכנית נשאיר בתיקייה הראשית של ה - main
ועכשיו הספריות שלנו יהיו קצת יותר מסודרים והם יראו לדוגמא כך:



Packages

שימו לב עכשיו שלאחר שייצרנו את הקבצים בחבילות נוספה לנו שורה חדשה בכל קובץ.
לדוגמא המחלקה Main תראה כך:

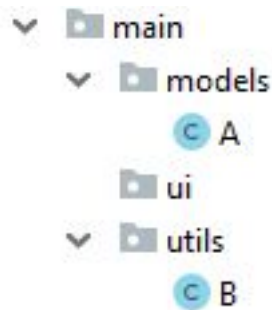
```
package main;
```

```
public class Main {  
  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

זה אומר שמעכשיו שנרצה לקרוא למחלקה מתוך חבילה אחרת נצטרך לייבא אותה קודם על ידי השימוש ב `import` דרך קריאה למחלקה לפי החבילה שבו היא נמצאת.
נראה דוגמא בשקף הבא:

Packages

לצורך הדוגמא יצרנו 2 מחלקות A ו-B בשני חבילות שונות.



אם נרצה לקרוא למחלקה A בתוך מחלקה B, הקוד שלנו יצטרך להיראות כך:

```
package main.utils;
```

```
import main.models.A;
```

```
public class B extends A {  
}
```


Packages

אז אחרי שראינו איך מייצרים חבילות ומייבאים מחלקות בין חבילות, הגיע הזמן להזכיר גם הרשאות. אם עד היום הכרנו את ההרשאות גישה הבאות:

- **public** שנותן הרשאות גישה לכל מי שיוצר את המופע
 - **private** שלא נותן הרשאות גישה
 - **protected** שנותן הרשאות גישה רק בירושה.
- כעת הגיע הזמן להכיר גם את:
-

הרשאת הגישה הדיפולטיבית היא רק למחלקות שנמצאים באותה החבילה בלבד וזה לא משנה גם עם ירשנו. עם המחלקה נמצאית האותה החבילה יש לה הרשאות ואם לא אז אין לה הרשאות.

Inner Classes

עד היום ראינו שכדי לייצר מחלקה אנחנו יוצרים קובץ חדש.
Java מאפשרת לנו ליצור גם מחלקה בתוך מחלקה כמו שאנחנו יוצרים משתנים של המחלקה.
לדוגמא:

```
public class Person {  
    public String name = "Moshe";  
  
    public class Student {  
        public int score = 100;  
    }  
}
```

הדרך ליצירת מופע מהמחלקה הזאת היא באמצעות קריאה למופע האבא:

```
public class Main {  
  
    public static void main(String[] args) {  
        Person person = new Person();  
        Person.Student student = person.new Student();  
        System.out.println(person.name + " " + student.score);  
    }  
}
```

Inner Classes

החידוש במחלקה פנימית בנוסף לסדר לוגי...

שמעכשיו ניתן ליצור גם מחלקות שמוגדרות כ **private** או **protected** לדוגמא: במקרה שנרצה ליצור מחלקת עזר מצד אחד אבל לא נרצה שאף אחד יוכל ליצור ממנה מופע ולממש את הפעולות שלה או שנרצה לתת את האפשרות הזאת רק למחלקות יורשות בלבד עכשיו אם נשנה את המחלקה Student ל **private** נקבל ב main שגיאה

```
public class Person {  
    public String name = "Moshe";  
  
    private class Student {  
        public int score = 100;  
    }  
}  
  
public class Main {  
  
    public static void main(String[] args) {  
        Person person = new Person();  
        Person.Student student = person.new Student();  
        System.out.println(person.name + " " + student.score);  
    }  
}
```

Throws Exception

(גלגול חריגות) Throws Exception

לצורך ההבנה של throw נחזור קצת אחורה, נסתכל על קטע קוד ונסה להבין מה בעייתי בו.

```
public static int divide(int num1, int num2) {  
    return num1 / num2;  
}
```

הבעיה היא שיכול להיות שהמשתמש הכניס את הספרה 0 למשל.
ובמקרה כזה התוכנה שלנו תקרוס.

איך היינו מונעים זאת?

באמצעות try ו catch

```
public static void readAndDivide() {  
    Scanner s = new Scanner(System.in);  
    System.out.print("Enter 2 numbers: ");  
    int num1 = s.nextInt();  
    int num2 = s.nextInt();  
    try {  
        System.out.println("divide result: " + divide(num1, num2));  
    } catch (ArithmeticException e) {  
        System.out.println(e.getMessage());  
    }  
}  
  
public static void main(String[] args) {  
    readAndDivide();  
}
```

Throws Exception

הבעייתיות בקוד שכתבנו היא שאנחנו מנענו מהתוכנה לקרוס אבל מי שישתמש בפונקציה לא ידע שיכולה להיות לו שגיאה. ויכול להיות שגם אם הוא ידע שיכולה להיות בעיה, הוא ירצה לטפל בה באופן שונה ממה שאנחנו ממשנו. לכן בשגיאות שיכולות להיות קריטיות למי שמשתמש בפונקציה, אז לפעמים מומלץ לגלגל חריגות כאלה מתחום האחריות של המתודה לתחום האחריות למי שקורא לה. הדרך לעשות זאת היא באמצעות המילה **throws** ולאחריה את סוג החריגה המצופה. לדוגמה במקרה שלנו אנחנו נסיף למתודה של החלוקה את החריגה האריתמטית:

```
public static int divide(int num1, int num2) throws ArithmeticException {  
    return num1 / num2;  
}
```

ומי שיקרא למתודה הזאת מהיום והלאה, יהיה חייב לממש **try** - **catch**.

```
public static void readAndDivide() {  
    ...  
    try {  
        System.out.println("divide result: " + divide(num1, num2));  
    } catch (ArithmeticException e) {  
        System.out.println(e.getMessage());  
    }  
}
```

Throws Exception

ברוב המקרים לא נרצה ששיטות ידפיסו את הודעת החריגה, אבל כן ידווחו עליה ל- main ושהוא יטפל בבעיה.
לכן ניתן גם לגלגל אחריות מפונקציה לפונקציה.
לדוגמא במקרה שלנו:

```
public static int divide(int num1, int num2) throws ArithmeticException {  
    return num1 / num2;  
}
```

```
public static void readAndDivide() throws ArithmeticException{  
    ...  
    System.out.println("divide result: " + divide(num1, num2));  
}
```

```
public static void main(String[] args) {  
    try {  
        readAndDivide();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

Throws Exception

```
public class Clock {  
    private int hours, minutes;  
    public Clock(int hours, int minutes) {  
        setHours(hours);  
        setMinutes(minutes);  
    }  
    public void setMinutes(int minutes) {  
        if (minutes < 0 || minutes >= 60)  
            System.out.println("Minutes have to be between 0-59");  
        else  
            this.minutes = minutes;  
    }  
    public boolean setHours(int hours) {  
        if (hours < 0 || hours >= 24)  
            return false;  
        else {  
            this.hours = hours;  
            return true;  
        }  
    }  
}
```

אז אחרי שראינו דרך יפה ל- "לא לקחת אחריות", איך זה עוזר לנו ביום יום?
אז נראה איך טיפלנו בנתונים שגויים עד היום.
לצורך הדוגמא ניצור מחלקה שמייצגת שעון.
בבנאי יש בעיה משום שהוא אינו יכול להחזיר ערך בשביל
להחזיר אינדיקציה האם הערכים שקיבל תקינים או לא.
אז ניצור פונקציה של set שהיא תבדוק את הערכים ותדפיס
הודעת שגיאה במקרה שהם אינם תקינים.
אופציה שניה:
פונקציה שתחזיר boolean .
מכיוון שאנחנו לא רוצים שהמחלקה תתעסק עם הודעות
למשתמש
לכן נחזיר boolean כאינדיקציה לכך אם הערך שהוכנס
תקין, והפונקציה גם תבצע את ההשמה אם הוא תקין

Throws Exception

```
public static void main(String[] args) {  
    Scanner s = new Scanner(System.in);  
    Clock c = new Clock();  
    c.setMinutes(70);  
    System.out.print("Enter hours: ");  
    int hours = s.nextInt();  
    while (!c.setHours(hours)) {  
        System.out.print("Enter hours again: ");  
        hours = s.nextInt();  
    }  
    c = new Clock(20, 80);  
}
```

איך היה נראה ה- main שלנו?!
בעיה ראשונה:
המתודה setMinutes מחזירה void,
ולכן במקרה של ערך שגוי, לכותב ה- main אין דרך לדעת
שנכנס ערך שגוי,
ולכן לא מתאפשר, למשל לקלוט את הערך מחדש
בשונה מהפונקציה setHours
הבעיה העיקרית:
הקונסטרקטור שקורא ל- set שמחזיר לו boolean.
מאחר ולא מתפקידו לבקש מהמשתמש
לקלוט ערך חדש.
לכן ב- main, שוב, אין לנו אינדיקציה שנכנס ערך שגוי.

איך נפתור את הבעיה?
באמצעות זריקת שגיאה מותאמת אישית

Throws Exception

```
public class Clock {
```

מהיום נוכל לכתוב את המחלקה שלנו כך:

```
    public Clock(int hours, int minutes) throws Exception {  
        setHours(hours);  
        setMinutes(minutes);  
    }
```

```
    public void setMinutes(int minutes) throws Exception {  
        if (minutes < 0 || minutes >= 60) {  
            throw new Exception("Minutes have to be between 0-59");  
        } else  
            this.minutes = minutes;  
    }
```

כעת באמצעות חריגה מותאמת אישית ניתן להפסיק ביצוע מתודה ולגלגל את הטיפול בחריגה למי שמשתמש בקוד

```
    public void setHours(int hours) throws Exception {  
        if (hours < 0 || hours >= 24) {  
            throw new Exception("Hours have to be between 0-23");  
        } else  
            this.hours = hours;  
    }  
}
```

Throws Exception

```
public static void main(String[] args) {  
    Scanner s = new Scanner(System.in);  
    Clock c;  
    boolean isValidInput = false;  
    while (!isValidInput) {  
        System.out.print("Enter hours and minutes: ");  
        try {  
            c = new Clock(s.nextInt(), s.nextInt());  
            System.out.println("The time is " + c);  
            isValidInput = true;  
        } catch (Exception e) {  
            System.out.println(e.getMessage());  
        }  
    }  
}
```

עכשיו ה main שלנו יראה כך:

Output:

```
Enter hours and minutes: 20 90  
Minutes have to be between 0-59  
Enter hours and minutes: 29 90  
Hours have to be between 0-23  
Enter hours and minutes: 23 54  
The time is 23:54
```

הגדירו את המחלקה Classroom שנתונה הם מערך ציוני סטודנטים ושם המורה שם המורה ומספר הסטודנטים בכיתה ישלחו כפרמטר לבנאי. לא ניתן לשנות את מספר הסטודנטים בכיתה לאחר יצירת האובייקט. ניתן להוסיף למחלקה תכונות נוספות לפי הצורך. הגדירו את השיטה setGrades המקבלת כפרמטר מערך מספרים ושמה את ערכיו בתוך המערך שבמחלקה. שימו לב: יתכן והמערך שהתקבל כפרמטר יותר גדול ממספר התלמידים בכיתה, ואז נזרוק Exception עם הודעה מתאימה. במידה וגודל המערך שהתקבל כפרמטר יותר קטן ממספר התלמידים בכיתה נשאיר 0 במערך הציונים עבור שאר הסטודנטים. יש לוודא שכל הציונים המועברים בתחום בין 0 ל-100, אחרת יש לזרוק Exception עם הודעה מתאימה. כתבו מתודה המקבלת אינדקס של סטודנט מסוים ומחזירה את הציון שלו. במידה והתקבל אינדקס שאינו בטווח, או עבור סטודנט שטרם הוזן עבורו ציון, יש לזרוק Exception עם הודעה מתאימה לכל אחד מהמקרים. כתבו תוכנית המייצרת אובייקט מטיפוס Classroom ומפעילה כל אחת מהמתודות. שימו לב לתפוס את כל השגיאות האפשריות. כמו כן, לא ניתן להניח שהקלט המתקבל תקין.