

## תרגול מס' 3: המתרגם

שימוש במחלקות קיימות  
מחזוריות, קבצים, וקבלת קלט מהמשתמש

## שלבי הפיתוח - חזרה קצרה



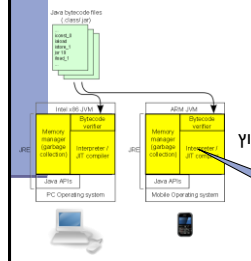
- ישנם שני שלבים נפרדים:
  - שלב פיתוח התוכנית
  - בשלב זה אנו משתמשים במהדר (קומפיילר) כדי להמיר קבצי .java. (קבצי טקסט הקריא למתכנת), לקבצי .class. שנועדו עבור המפרש (אינטרפרטר).

קומפילציה

## שלבי הפיתוח - חזרה קצרה

- ישנם שני שלבים נפרדים:
  - שלב הרצת התוכנית
  - בשלב זה אנו משתמשים במפרש כדי להריץ את קבצי ה-class שייצרנו.
  - ב-Java אותו קובץ class יכול לרוץ בסביבות שונות אם קיים עבורן מפרש.

הרצה

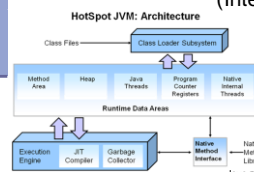


## שלבי הפיתוח - חזרה קצרה

- ה-JDK (Java Development Kit) נדרש לתהליך הפיתוח
  - קומפיילר
- ה-JRE (Java Runtime Environment) נדרש להרצת תוכניות
  - JVM (Java Virtual Machine)
  - הספריות הסטנדרטיות

## חזרה קצרה

- ה-JVM הוא "מכונה" המריצה תוכניות Java
  - יודע לטעון תוכניות
  - יודע לוודא את תקינות הקבצים הנטענים
  - מכיל את המפרש (Interpreter)



## המתרגם

## שאלות

- האם כבר יש שירות תרגום שאנחנו יכולים להשתמש בו?
- כיצד קוראים מקבצים?
- מה הפורמט של הקלט?
- נצטרך להחליט

## המתרגם

- משימה:
- תכנית המתרגמת קטעי טקסט לשפה אחרת
- הקלט: קובץ המכיל את קטעי הטקסט וכן את השפה אליה רוצים לתרגם



## API – Application Programming Interface

- ממשק המאפשר לאפליקציה לתקשר עם תוכנה אחרת
- עבור ג'אווה קיימים כלים רבים הזמינים ברשת כקוד פתוח
- בתרגול זה נשתמש ב-API לתרגום כללי בשם Translate, במציאות, קיימים ברשת כלים שונים של Google, Microsoft ועוד

## הפשטה

- כצעד ראשון נפתור בעיה הרבה יותר פשוטה
- תכנית שמתרגמת את המילה "Hello" מאנגלית לצרפתית
- יש: שימוש בשירות תרגום
- אין: קלט, טקסט, עבודה עם קבצים, פורמט

## אינטראקציה עם המשתמש

- נתחיל להתקדם עקב בצד אגודל אל היעד שלנו
- קלט מהמשתמש יינתן בשורת הפקודה
- פרמטר ראשון: המילה לתרגום
- פרמטר שני: שפת המקור
- פרמטר שלישי: שפת היעד

## שלב א'

```
public class TranslatorEngine1 {  
  
    public static void main(String[] args) throws Exception {  
  
        String TranslatedText = Translate.execute("Hello",  
            Language.ENGLISH,  
            Language.FRENCH);  
  
        System.out.println(TranslatedText);  
  
    }  
}
```

## שלב ב'

```
public class TranslatorEngine2 {  
  
    public static void main(String[] args) throws Exception {  
        String TranslatedText = Translate.execute(args[0],  
            Language.fromString(args[1]),  
            Language.fromString(args[2]));  
        System.out.println(TranslatedText);  
    }  
}
```

## קריאת קלט

- נקרא קלט מהמשתמש (console)
  - עדיין מילה אחת
  - אין שימוש בקבצים
- נשתמש במחלקה Scanner
  - מה הפורמט של הקלט?

## המחלקה Scanner

- סורק טקסט פשוט
- "שובר" את הקלט לרכיביו השונים (מילה, מספר וכדומה)
- בעת יצירה מקבל כפרמטר מהיכן לקרוא את הקלט

```
Scanner s = new Scanner(System.in);  
int anInt = s.nextInt();  
float aFloat = s.nextFloat();  
String aString = s.next();  
String aLine = s.nextLine();
```

<http://docs.oracle.com/javase/7/docs/api/index.html?java/util/Scanner.html>

## המחלקה Scanner

- המחלקה Scanner מייצגת משאב מערכת (קובץ שנפתח לקריאה)
- משאבים כאלו יש להחזיר למערכת מהר ככל הניתן.
  - ע"י שימוש מפורש בשירות close()
  - ע"י שימוש בבלוק try() שנלמד בהמשך הקורס

## פורמט הקלט

- מהו הפרוטוקול המשותף שחולקים האפליקציה והמשתמש לצורך התקשורת ביניהם
  - איזה מידע דרוש
  - כיצד הוא מקודד (מספר, מחרוזת, ...)
  - מה סדר הפרמטרים

נבחר: <word> <source-lang> <target-lang>  
לדוגמא,  
הקלט: hello English French  
הפלט: bonjour

## דוגמא

```
Scanner s = new Scanner(System.in);  
System.out.println("enter line:");  
while (s.hasNext())  
    System.out.println(s.next());  
s.close();
```

קרא כ- standard input

קרא את ה-Token הבא

## קבצים

- במקום לקרוא את שורת הקלט מהמשתמש נקרא אותה מקובץ
  - קובץ מיוצג ע"י:
    - המחלקה `java.io.File`
    - המחלקה `java.nio.file.Path`
  - נאתחל את האובייקט עם המסלול (path) לקובץ
- ```
Path p = Paths.get("C:\\Software1\\example.txt");
Path p = Paths.get("C:", "Software1", "example.txt");
```

## שלב ג'

```
public class TranslatorEngine3 {
    public static void main(String[] args) throws Exception {
        Scanner s = new Scanner(System.in);
        String[] fragments = s.nextLine().split(" ");
        String TranslatedText = Translate.execute(fragments[0],
            Language.fromString(fragments[1]),
            Language.fromString(fragments[2]));
        System.out.println(TranslatedText);
        s.close();
    }
}
```

## תלות בסביבה

- ג'אווה היא שפת תכנות חוצת סביבות, אבל מערכת הקבצים תלויה בסביבה!
  - למשל, המפריד בסביבת Unix הוא / (slash)  
`/usr/local/software1/example.txt`
  - ובסביבת Windows הוא \ (backslash)  
`C:\Software1\example.txt`
  - אך היא תומכת גם ב- / כמפריד.
- **נשתדל להימנע משימוש מפורש במפריד!**
  - לדוגמא: ניתן להשתמש בפונקציות `resolveXXX()` של `Path`

## מסלול (Path) לקובץ

- מסלול יחסי – `Relative path`
  - `Paths.get("subfolder", "example.txt")`
  - ב- eclipse המיקום הנוכחי במהלך ריצה הוא ה- Project root
- מסלול מלא – `Absolute path`
  - `Paths.get("C:", "Software1", "example.txt")`

## קלטים מרובים

- מספר שורות קלט מקובץ
  - נקרא מספר קלטים עד לסוף הקובץ
- שימוש במתודות `hasNextLine()` ו-`nextLine()`

## שלב ד'

```
public class TranslatorEngine4 {
    private static final String FILE_NAME = "example.txt";

    public static void main(String[] args) throws Exception {
        Scanner s = new Scanner(Paths.get(FILE_NAME));
        String[] fragments = s.nextLine().split(" ");
        String TranslatedText = Translate.execute(fragments[0],
            Language.fromString(fragments[1]),
            Language.fromString(fragments[2]));
        System.out.println(TranslatedText);
        s.close();
    }
}
```

## פיסקה

- פיסקה ולא רק מילה אחת
- מה יהיה הפורמט החדש?

■ נבחר:

<source-lang>#<target-lang>#<paragraph>

## שלב ה'

```
public class TranslatorEngine5 {  
    private static final String FILE_NAME = "...";  
    public static void main(String[] args) throws Exception {  
        Scanner s = new Scanner(Paths.get(FILE_NAME));  
        while (s.hasNextLine()) {  
            String[] fragments = s.nextLine().split(" ");  
            System.out.println(Translate.execute(fragments[0],  
                Language.fromString(fragments[1]),  
                Language.fromString(fragments[2])));  
        }  
        s.close();  
    }  
}
```

## קריאת פיסקה מהקובץ

- פיסקה יכולה להכיל מספר שורות (נוותר בינתיים על קלטים מרובים).
- נרצה לקרוא ולצרף אותן למחרוזת אחת.
- ניתן להשתמש באופרטור +, שיוצר בכל פעם מחרוזת חדשה
- אנו נשתמש במחלקה `StringBuilder`

## Scanner – Set Delimiter Example

```
String input = "1 fish 2 fish red fish blue fish ";  
Scanner s =  
    new Scanner(input).useDelimiter(" fish ");  
while (s.hasNext())  
    System.out.println(s.next());  
s.close();
```

## שלב ו'

```
public class TranslatorEngine6 {  
    private static final String FILE_NAME = "...";  
    public static void main(String[] args) throws Exception {  
        Scanner s = new Scanner(Paths.get(FILE_NAME));  
        s.useDelimiter("#");  
        Language from = Language.fromString(s.next());  
        Language to = Language.fromString(s.next());  
        s.skip("#");  
        StringBuilder text = new StringBuilder();  
        while (s.hasNextLine()) {  
            text.append(s.nextLine());  
            text.append(' ');  
        }  
        System.out.println(Translate.execute(text.toString(), from, to));  
        s.close();  
    }  
}
```

## המחלקה `StringBuilder`

- מייצגת מחרוזת ניתנת לשנוי (mutable)
- מאפשרת לבצע שינוי במחרוזת קיימת מבלי ליצור עצמים חדשים עם כל שינוי
- שירותים חשובים: `append` ו-`insert`

```
StringBuilder sb = new StringBuilder("abc");  
sb.append("d");
```

## חזרה קצרה על מחרוזות

תכנות מתקדם בשפת Java  
אנדרסיסית חל אביב

35

## לאן עכשיו?

- טיפול בשגיאות
  - פורמט לא תקין, כשלאן בזיהוי השפות או בתרגום
  - ניתן לבדוק בקוד או להגדיר בחוזה
- הרחבת התכנית
  - תרגום מספר קבצים
  - מספר פסקאות בקובץ יחיד
  - זיהוי אוטומטי של שפת הקלט
- ...

תכנות מתקדם בשפת Java  
אנדרסיסית חל אביב

32

## מחרוזות – פונקציות בדיקה

| Method                              | Description                                                                    |
|-------------------------------------|--------------------------------------------------------------------------------|
| <code>equals (str)</code>           | whether two strings contain the same characters                                |
| <code>equalsIgnoreCase (str)</code> | whether two strings contain the same characters, ignoring upper vs. lower case |
| <code>startsWith (str)</code>       | whether one contains other's characters at start                               |
| <code>endsWith (str)</code>         | whether one contains other's characters at end                                 |
| <code>contains (str)</code>         | whether the given string is found within this one                              |

כדי להשוות שתי מחרוזות מבחינת תוכן יש להשתמש בפונקציה `equals()` ולא באופרטור `==` שבדוק אם מדובר באותו אובייקט

37

## מחרוזות

- מחרוזות הן אובייקטים המכילים רצף של תווים.

`String s = "Hello";`

| index     | 0 | 1 | 2 | 3 | 4 |
|-----------|---|---|---|---|---|
| character | H | e | l | l | o |

- כל אלמנט במחרוזת הוא מסוג `char`.
- האינדקס של התו הראשון הוא 0.
- אורך המחרוזת מוחזר ע"י הפונקציה `length()`
- שרשרת מחרוזות נעשה ע"י האופרטור `+`

`String s2 = s + " World" + 5 // "Hello World5"`

36

## מחרוזות – פיצול לחלקים

| Method name                          | Description                                                                                  |
|--------------------------------------|----------------------------------------------------------------------------------------------|
| <code>split (DelimiterString)</code> | Splits the string into tokens using the given delimiter string. Returns an array of Strings. |

```
String str= "Another useful example";
String[] tokens = str.split(" ");
//tokens = {"Another","useful","example"}
```

39

## מחרוזות – פונקציות שימושיות

| Method name                                                                      | Description                                                                                                                                                        |
|----------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>indexOf (str)</code>                                                       | index where the start of the given string appears in this string (-1 if not found)                                                                                 |
| <code>substring (index1, index2)</code><br>or<br><code>substring (index1)</code> | the characters in this string from <code>index1</code> (inclusive) to <code>index2</code> (exclusive); if <code>index2</code> is omitted, grabs till end of string |
| <code>toLowerCase ()</code>                                                      | a new string with all lowercase letters                                                                                                                            |
| <code>toUpperCase ()</code>                                                      | a new string with all uppercase letters                                                                                                                            |

הימוש של הפונקציות לעיבוד מחרוזות יחזיר תמיד מחרוזת חדשה ולא יבצע שינויים על המחרוזת המקורית שעליה נקראה הפונקציה (Strings are immutable in Java).

38