

מנגנוני שפת Java

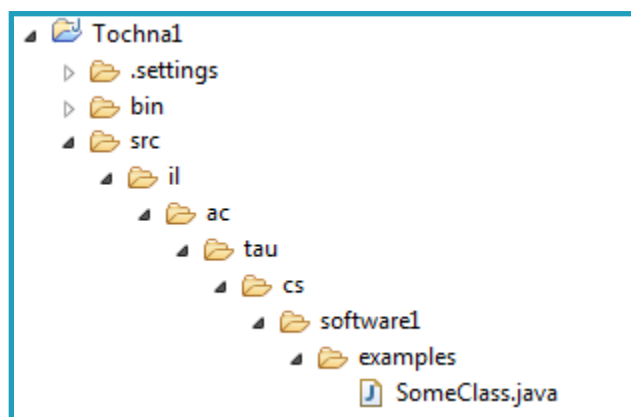
מה נכלל בעבודה עצמית זו?

- ✓ חבילות ומרחב השמות
- ✓ משפטי import
- ✓ classpath/jar/Javadoc

חלק א' - חבילות ומרחב השמות

עד כה ראינו תוכניות ב Java שכולן נכתבו בתיקה אחת ישירות מתחת ל src ב eclipse. אם יש לנו פרויקט גדול שמכיל הרבה קוד, הגיוני שנרצה לסדר אותו לפי מבנה תיקיות היררכי, בדיוק כמו שאנו מסדרים קבצים במחשב.

התיקיות בפרויקט נקראות חבילות (package) ויכולות להכיל תיקיות אחרות. השם המלא (fully qualified name) של מחלקה מכיל את היררכיית החבילות שלה, מהחיצונית עד הפנימית, מופרדות בנקודה. התבוננו במבנה ההיררכי הבא:



מבנה זה מייצג מבנה סטנדרטי של פרויקט גדול בארגון. כל אחת מחבילות הביניים (למשל cs) יכולה להכיל חבילות ומחלקות נוספות.

המחלקה SomeClass נמצאת בחבילה שאינה הדיפולטית, ולכן:

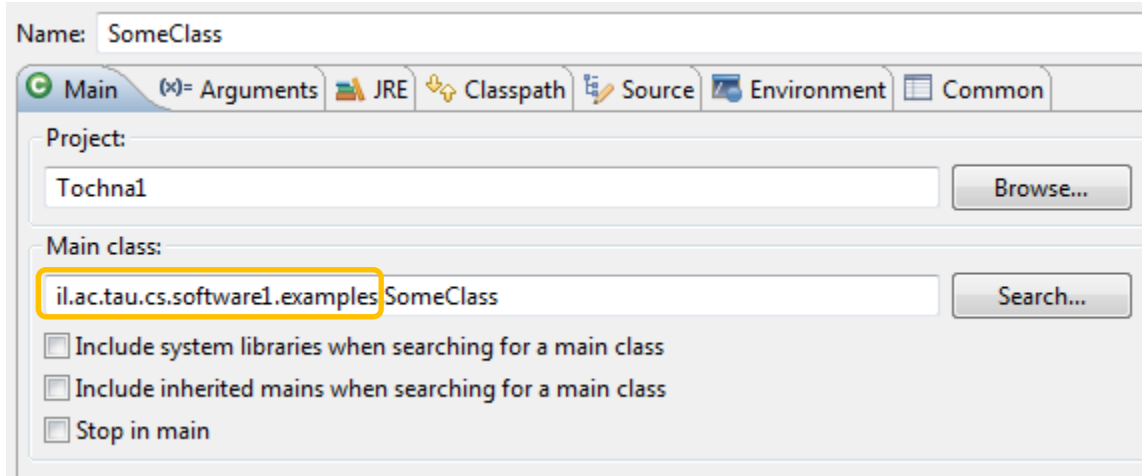
1. כשנממש את המחלקה, נצטרך להצהיר על החבילה של המחלקה, וההצהרה צריכה להיות זהה למבנה ההיררכי האמיתי. ב eclipse אפשר לחולל את ההצהרה הזו בצורה אוטומטית. שימו לב שההצהרה על ה package נמצאת מחוץ למחלקה, והיא השורה הראשונה בקובץ. עבור SomeClass נצהיר על החבילה באופן הבא:

```
package il.ac.tau.cs.software1.examples;

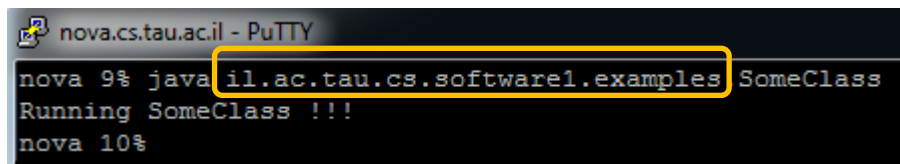
public class SomeClass{
    public static void main(String[] args){
        System.out.println("Running SomeClass !!!");
    }
}
```

מדוע התיקיה src אינה חלק ממבנה החבילות? זה תלוי בקונפיגורציה של ה-eclipse. כברירת מחדל, src היא תיקיה שמכילה קבצי sources, אך ניתן להגדיר ב-eclipse תיקיות נוספות שיכילו קבצי מקור, ואז "ההתחלה" היא מתוך תיקיות אלה.

2. כשנריץ את התוכנית, עלינו להשתמש בשם המלא שלה. אם נעשה run דרך ה-eclipse ואז נסתכל על הקונפיגורציה, נגלה שזה מה שה-eclipse עושה:



כמובן שאותו הדבר צריך לקרות בהרצה ב command line:



כעת, צרו את התוכנית MainClass בתוך החבילה הדיפולטית והעתיקו לתוך הקובץ את הקוד הבא:

```
package il.ac.tau.sc.software1.examples;
public class MainClass {
    public static void main(String[] args){
        System.out.println("it works!");
    }
}
```

בגלל חוסר ההתאמה בין המיקום הפיזי לבין הצהרה על החבילה, ה-eclipse מציע שתי אפשרויות תיקון. בחרו בזו שמתאימה את מבנה החבילות וודאו שאתם מקבלים את מבנה התיקיות ההיררכי שנגזר משם החבילה.

חלק ב' – משפטי import

צרו את שלושת המחלקות הבאות בפרויקט שלכם:

```
package pack1;

public class A {
    public static void func(){
        System.out.println("pack1.A");
    }
}
```

```
package pack1.pack2;

public class A {
    public static void func(){
        System.out.println("pack1.pack2.A");
    }
}
```

```
package pack1.pack3;

public class B {
    public static void main(String[] args){
        A.func();
    }
}
```

קודם כל, נשים לב שאין שום בעיה לייצר שתי מחלקות עם אותו השם (A) בפרויקט שלנו, זאת כיוון ש A אינו השם המלא של המחלקה, והשם המלא כולל גם את החבילה. באותה החבילה לא ניתן לייצר שתי מחלקות עם אותו השם.

שנית, נשים לב ש B לא מתקמפלת כיוון ש B לא יודעת מי זו A. האם זה קורה בגלל שיש שתי מחלקות בשם A? נבדוק את זה. החליפו את שם המחלקה A אשר נמצאת ב pack1.pack2 לשם C באמצעות refactor. שימו לב שה eclipse מחפש שימושים אפשריים של המחלקה ויכול להציע לכם לתקן גם את הקוד של B. אל תאפשרו לו.

מה קורה אחרי שהחלפנו את שם המחלקה להיות C? כעת יש רק מחלקה A אחת, אבל הקוד של B עדין לא מתקמפל. מדוע?

B מכירה רק את המחלקות שנמצאות איתה באותה החבילה, ומעבר לזה, היא צריכה לקבל שמות מלאים. יש שתי דרכים להעביר את השם המלא:

1. ציון מפורשות: אם אנחנו רוצים להשתמש ב A שנמצאת ב pack1 עלינו לכתוב:

```
pack1.A.func();
```

בתוך גוף הפונקציה.

כעת, גם הקומפיילר יודע בדיוק לאיזו מחלקה מתכוונים.

צורת הכתיבה הזו נכונה אך לא שימושית במיוחד. בפרויקטים שבהם יש היררכיית מחלקות עמוקה כתיב זה מסרביל מאד את הקוד.

2. שימוש במשפטי import: זו הדרך המקובלת לגרום למחלקה מחבילה אחת להכיר מחלקות מחבילה אחרת.

בתוך המחלקה B קרבו את העכבר ל A שמסומנת בקו אדום. ה eclipse בעצם מציע תיקונים אפשריים לבעיה שקיימת, ושני התיקונים הראשונים שהוא מציע הוא ביצוע import-ים. בחרו ב import מתוך pack1 והתבוננו בקוד. לקוד נוספה השורה הבאה:

```
import pack1.A;
```

כעת הקומפיילר יודע לאיזו מחלקה A אנו מתכוונים. מספר הערות על משפטי import:

- אם נרצה "לייבא" מספר מחלקות מאותה החבילה ניתן להשתמש ב *. למשל, אם נרצה לייבא את כל המחלקות שנמצאות ישירות בתוך pack1 נרשום `import pack1.*`.
- ביצוע `import pack1.*` לא מייבא מחלקות שנמצאות בתיקיות הנמצאות בתוך pack1.
- ה import לא שותל קוד במחלקה שבה הוא כתוב. הוא נועד לפשט את הקוד ולאפשר להשתמש בשמות "הקצרים" של כל מחלקה בגוף הקוד.
- כדאי לתת ל eclipse להוסיף את ה import-ים הנדרשים ולא לכתוב אותם בעצמנו.

ומה יקרה אם נרצה לעשות שימוש בשתי המחלקות A שקיימות בפרויקט בתוך B? אם ננסה לעשות import לשתייהן, זה לא יצליח, כיוון שה import מאפשר לנו להשתמש בשם "המקוצר" ובמקרה זה מדובר בשני שמות מקוצרים זהים. במקרה כזה, כדאי לעשות import רק למחלקה שבה נעשה יותר שימוש ועבור המחלקה השניה נכתוב את השם המלא.

אפשרות import נוספת (שאינה נפוצה מאוד) היא static import שמאפשרת "לייבא" מתודה אחת ולא מחלקה (בדומה לתחביר שקיים גם ב python). העתיקו את הקוד הבא לתוך המחלקה B והריצו אותו:

```
package pack1.pack3;

import static pack1.pack2.A.func;
import pack1.A;

public class B {
    public static void main(String[] args){
        A.func();
        func();
    }
}
```

ניתן לראות שהמחלקה pack1.A מיובאת במלואה, וכל שימוש ב A יקושר אליה. מתוך המחלקה pack1.pack2.A יבאנו רק את func, ולכן ניתן להשתמש בפונקציה זו ללא ציון שם המחלקה. ומה יקרה אם גם ב B יש פונקציה בשם func? תיווצר בעיית עמימות (Ambiguity) ולכן זה לא יתאפשר.

חלק ג' – [classpath/jar/Javadoc](#)

מהם קבצי Javadoc, וכיצד הם מסייעים לנו בתור מפתחים?

קובץ jar

קובץ שמבוסס על דחיסה הדומה לדחיסת קובץ zip שמכיל בתוכו בעיקר קבצי class, כלומר קבצי ג'אוה לאחר שעברו קומפילציה. קבצים אלו משמשים להעברה של קוד "ארוז" בין מפתחים לשימוש חוזר בקוד. מכיוון שהקבצים הארוזים הם קבצי class ולא קבצי java, הם אינם מיועדים להמשך פיתוח ושינוי ע"י מפתח אחר. אנו בתור מפתחים יכולים להשתמש בjar חיצוני שסופק לנו (כפי שתעשו במדריך זה) או לארוז קוד משלנו בjar.

Javadoc

פורמט יצירת תיעוד קוד של ג'אווה. הפורמט מאפשר ליצר מחלקות ומתודות להוסיף תיעוד שמתאר את המחלקות והמתודות בצורה קלה יחסית. הפורמט מאפשר למשתמש עתידי במחלקות ובמתודות המתועדות גישה נוחה לאותו תיעוד, בצורה שאותה נראה בהמשך. תיעוד בפורמט של Javadoc ניתן לארוז בקבצי jar.

בחלק זה נשתמש בקוד שנכתב במקום אחר וסופק לנו כקובץ jar ובנוסף נקבל גם קובץ jar שני שמכיל את התיעוד של המחלקות בפורמט של Javadoc. השלבים שנבצע הם:

1. הורדה של קובץ jar שמכיל קוד חיצוני, והוספה שלו לפרויקט שלנו.
2. ניסיון לכתוב קוד חדש משלנו שעושה שימוש בפונקציה ספציפית שנמצאת בjar החדש שהוספנו (עם כישלון מתוכנן מראש).
3. הורדה והוספה של קובץ jar נוסף של javadoc שמכיל את התיעוד הרלוונטי לקוד שהורדנו בשלב 1.
4. שינוי של הקוד שכתבנו בשלב 2, בהתאם לתיעוד שקיבלנו בjavadoc.

שלב 1

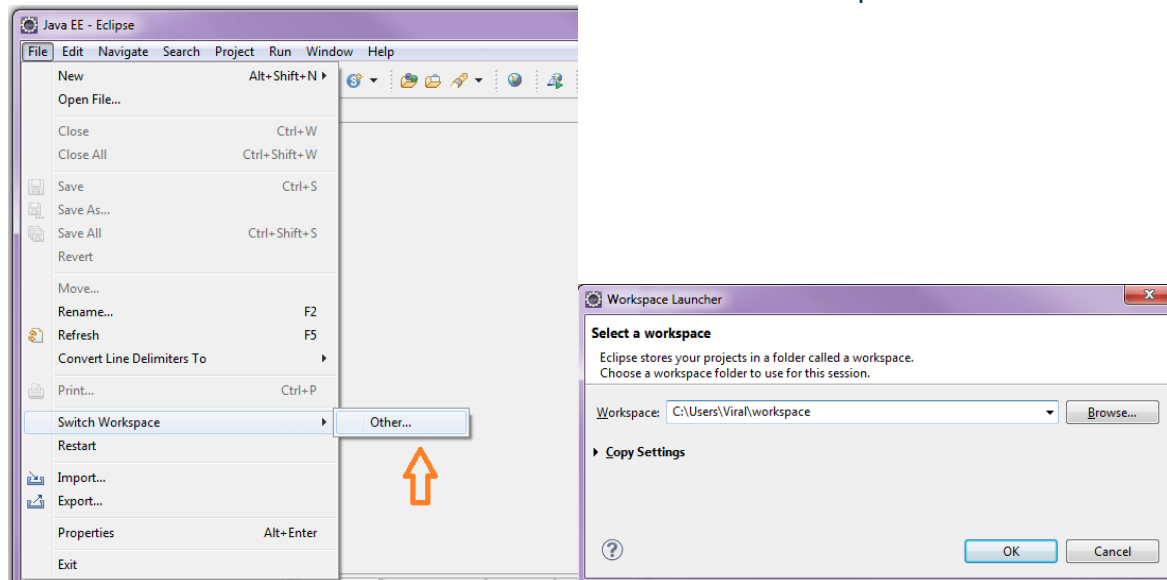
הורידו את קובץ ה jar של ה class-ים מהקישור הבא:

<http://courses.cs.tau.ac.il/software1/2021a/lectures/tutorials/resources/Tutorial3classes.jar>

כדאי, אך לא חובה, להעתיק את ה jar-ים שבהם משתמשים בפרויקט לתוך הפרויקט, רצוי בתיקה מיוחדת (lib או resources), ואנו אכן נעשה זאת במדריך זה.

צרו פרויקט ג'אווה חדש בשם Tutorial_3, גשו לתיקייה של הפרויקט בworkspace שהגדרתם לעclipse וצרו תיקייה חדשה בשם 'resources' והעבירו את קובץ jarn שהורדתם לתיקייה זו.

על מנת למצוא איפה נמצא workspace שלכם בג'אווה התבוננו בתמונות הבאות:



הוספת ה jar לפרויקט ב eclipse:

1. מתוך תצוגת ה package explorer, סמנו את ה jar המבוקש, לחצו קליק ימני, ובחרו באופציה add to build path
2. לחלופין, ניתן להוסיף בשיטה הבאה גם jar-ים שאינם נמצאים בתצוגה של הפרויקט:

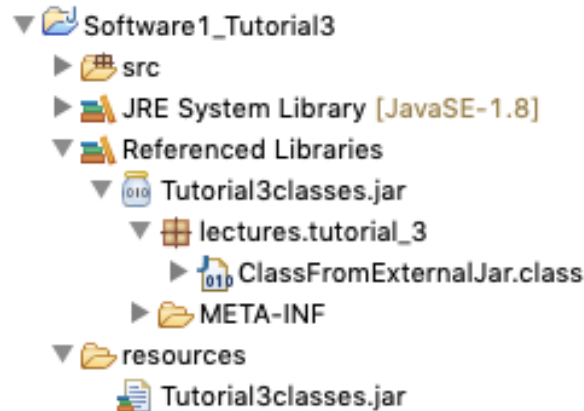
מקש ימני על הפרויקט שלנו -> בחירה ב 'Build Path' <- Configure Build Path
בחלון שנפתח לבחור בטאב Libraries <- Add External JARs <- לבחור את קובץ jar שלכם
וללחוץ על Apply.
ההוספה עשויה להיראות קצת שונה בגרסאות eclipse שונות מבחינת שמות התפריטים
והלחצנים אך העקרונות זהים.

ישנן 2 דרכים נוספות להוספה jar לפרויקט שלנו שלא דרך eclipse (אין צורך לבצע אם הוספתם את קובץ jar).

3. הוספת ה jar בהרצה משורת הפקודה (הדוגמה מתייחסת להרצת תוכנית בשם MainClass):
Java -cp jar1.jar MainClass
שימו לב שלהוספת מספר jar-ים יש תחביר קצת שונה בין מערכת הפעלת אחת לאחרת (ספציפית התו המפריד בין שמות ה jar-ים שונה).

4. אפשרות אחרונה היא לעדכן את משתנה הסביבה CLASSPATH שקיים בכל מערכת ההפעלה, והוא למעשה מכיל מיקומים שבהם ה JVP יחפש את המחלקות שבהם התוכנית משתמשת. ניתן לקנפג אותו ישירות מתוך מערכת ההפעלה, או באמצעות שתי השיטות המקומיות שציינו קודם. למעשה, הדגל -cp הוא קיצור של הדגל -classpath שמאתחל את משתנה הסביבה הזה בריצה המסוימת של תוכנית ה java.

לאחר הוספת ה jar לפרויקט ב eclipse, זו אמורה להיות תצוגת הפרויקט:



שימו לב שתחת referenced libraries אנחנו רואים את מבנה החבילות של הקוד בתוך ה jar גם אם הקוד עצמו לא נגיש לנו. אתם מוזמנים, אגב, ללחוץ על קובץ ה class ולנסות להבין מה תוכנו.

שלב 2

נעת נכתוב תוכנית אשר תשתמש בקוד שהורדנו, מבלי לראות את הקוד עצמו. את התוכנית עצמה נכתוב בחבילה הדיפולטית (אם כי זה לא משנה). הוסיפו מחלקה חדשה לפרויקט תחת src בשם 'Main Class', והעתיקו לשם את הקוד המופיע בהמשך. שימו לב לא לשכוח את פקודת import שלמעלה, היא זאת שמייבאת את הקוד מקובץ jar שהוספנו. אם אתם מקבלים שגיאה בפקודה זו ככל הנראה לא ייבאתם בהצלחה את קובץ ה jar.

```
import lectures.tutorial_3.ClassFromExternalJar;

public class MainClass {
    public static void main(String[] args){
        ClassFromExternalJar.mathFunc(0);
    }
}
```

שימו לב שה import שלנו כולל את הנתיב המלא למחלקה ClassFromExternalJar. הריצו את התוכנית. מה קיבלתם? נסו להריץ אותה מספר פעמים (רמז: הקוד מכיל אקראיות ולכן אתם עשויים לקבל התנהגות שונה בהרצות שונות!).

האם יכול להיות שאנחנו לא משתמשים בקוד כמו שצריך? כשמקריבים את העכבר לפונקציה mathFunc אנחנו לא מקבלים שום הסברים על אופן פעולת הפונקציה. לכן כאשר אנחנו מקבלים קוד חיצוני חשוב לקבל אותו עם תיעוד נגיש, וכאן נכנס לפעולה javadoc.

שלב 3

אנחנו נרצה לשלב תיעוד של הקוד בקובץ jar שקיבלנו בתוך ה eclipse. הורידו את ה jar שמכיל את קבצי התיעוד מהקישור הבאה:

<http://courses.cs.tau.ac.il/software1/2021a/lectures/tutorials/resources/Tutorial3doc.jar>

והעתיקו אותה לתיקיית ה jar-ים משלב 1.

להוספת הקישור בין קובץ ה jar-ים שמכיל את הקבצים המקומפלים לבין התיעוד, עקבו אחרי המדריך הבא, רק שימו לב להבהרה לשלב האחרון, שמופיע בהמשך:

<https://stackoverflow.com/questions/9870448/how-to-attach-source-or-javadoc-in-eclipse-for-any-jar-file-e-g-javafx>

הבהרה לשלב האחרון: כל התיעוד נמצא בתוך ה jar בתוך התיקיה doc, ולכן בשלב האחרון עליכם לציין את הנתיב בתוך ה jar בצורה הבאה:

לחצו על הכפתור validate על מנת לוודא שה jar שלכם מכיל תיעוד Javadoc תקין.

כעת, קרבו שוב את העכבר לשם הפונקציה. אתם אמורים להיות מסוגלים לראות את התיעוד. התיעוד עמום בכוונה, אך הוא מציג שני אלמנטים מרכזיים: אופי הקלט המותר ואופי הפלט. שימו לב שהפלט המובטח יתקבל רק עבור קלטים "חוקיים" – כלומר, אלה שמקיימים את ההגבלות שצוינו. מה יקרה עבור קלטים לא חוקיים? אתם מוזמנים לבדוק מספר קלטים לא חוקיים בעצמכם. הכלל אומר שכותבת הפונקציה לא מתחייבת לשום דבר בהינתן קלט לא חוקי.

הריצו את הפונקציה עם מספר קלטים חוקיים וודאו שהיא מקיימת את מה שמובטח בתיעוד.

ראינו כיצד ניתן לעשות שימוש בקבצי jar חיצוני ולהוסיף את התיעוד שלו באמצעות Javadoc.

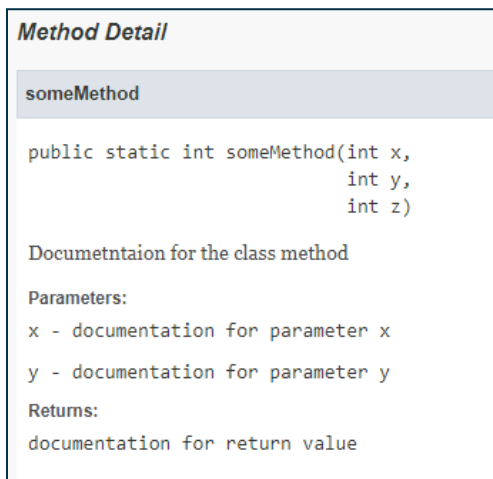
כיצד מייצרים את התיעוד שיופיע ב Javadoc מתוך הקוד שלכם שתכתבו בעצמכם?
את התיעוד יש להוסיף באופן הבא:

```
/** Documentation for the package */  
package somePackage;  
  
/** Documentation for the class  
 * @author your name here  
 */  
public class SomeClass {  
  
    /** Documentation for the class variable */  
    public static int someVariable;  
  
    /** Documentation for the class method  
     * @param x documentation for parameter x  
     * @param y documentation for parameter y  
     * @return  
     * documentation for return value  
     */  
    public static int someMethod(int x, int y, int z) {  
        // this comment would NOT be included in the documentation  
        return 0;  
    }  
}
```

כלומר, ניתן לכתוב תיעוד על החבילה, על המחלקה ועל רכיבים שונים במחלקה (שדות, פונקציות). התיעוד יתחיל בסימן /** ויסתיים ב /.*.

כעת אחרי שיש לנו תיעוד בפורמט של Javadoc, ניתן לחולל אותו אוטומטית באמצעות ה eclipse (Project-> generate JavaDoc) או בשורת הפקודה באמצעות הפקודה javadoc.

מעבר לכך שניתן לצפות בתיעוד בתוך ה eclipse כפי שראינו קודם, נוצרים דפי html עבור כל מחלקה שנראים כך:



```
Method Detail  
  
someMethod  
  
public static int someMethod(int x,  
                             int y,  
                             int z)  
  
Documetntaion for the class method  
  
Parameters:  
x - documentation for parameter x  
y - documentation for parameter y  
  
Returns:  
documentation for return value
```

זהו בעצם אותו הפורמט שאנחנו רואים בתיעוד ה API של Java באתר של Oracle. לדוגמה, התיעוד של המחלקה String:

<https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/String.html>

התיעוד של כלל הספריות:

<https://docs.oracle.com/en/java/javase/21/docs/api/>