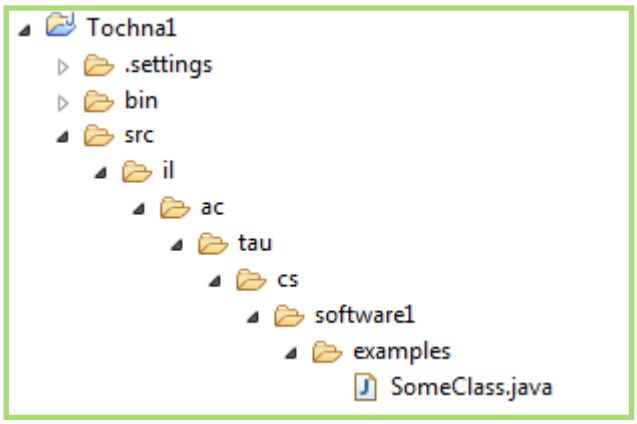
עבודה עצמית לשבוע 3:

חלק א׳ - חבילות ומרחב השמות

עד כה ראינו תוכניות ב Java כולם נכתבות בתיקיה אחת ישירות מתחת ל src ב eclipse. אם יש לנו פרוייקט גדול שמכיל הרבה קוד, הגיוני שנרצה לסדר אותו לפי מבנה תיקיות היררכי, בדיוק כמו שאנחנו מסדרים קבצים במחשב.

התיקיות בפרוייקט נקראות חבילות – package ויכולות להכיל תיקיות אחרות. השם המלא (fully qualified name) של מחלקה מכיל את היררכיות החבילות שלה, מהחיצונית עד הפנימית, והן מופרדות בנקודה.

התבוננו במבנה ההיררכי הבא:



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

המחלקה SomeClass נמצאת בחבילה שאינה הדיפולטית, וזה ישפיע על כמה היבטים:

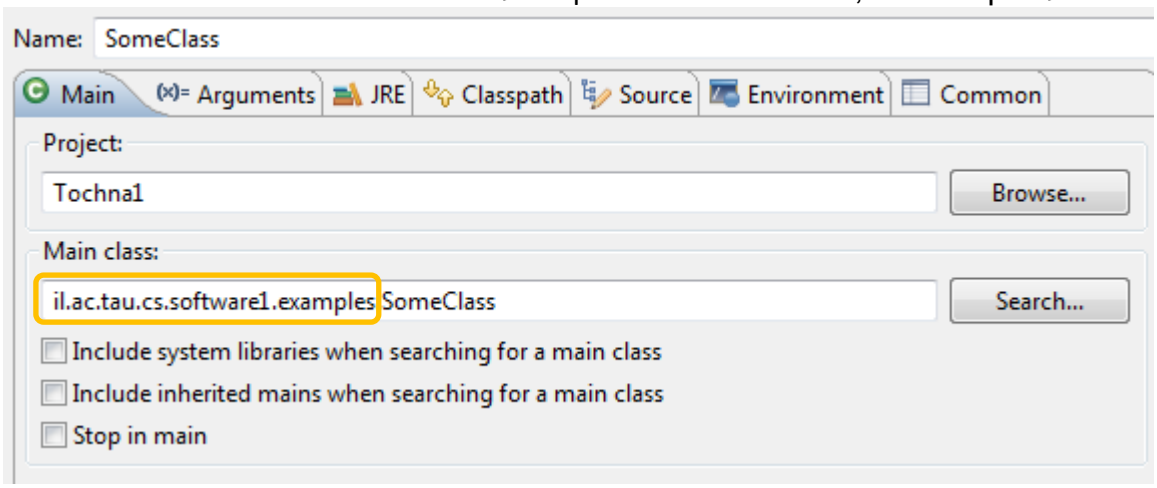
1. כשנממש את המחלקה, נצטרך להצהיר על החבילה של המחלקה, וההצהרה צריכה להיות זהה למבנה ההיררכי האמיתי. ב eclipse אפשר לחולל את ההצהרה הזו בצורה אוטומית. עבור SomeClass נצהיר על החבילה באופן הבא.

שימו לב שההצהרה על ה package נמצאת מחוץ למחלקה, והיא השורה הראשונה בקובץ.

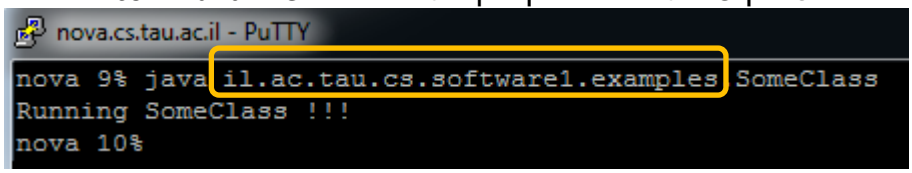
```
package il.ac.tau.cs.software1.examples;

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. ניתן לקנפג את כל תיקיות הקבצי המקור (sources) ואז ה״התחלה״ היא מתוך תיקיות אלה. באופן דיפולטי, src היא תיקיה שמכילה קבצי sources, וניתן להוסיף עוד תיקיות.

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 PackageExample 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 בפרוייקט 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) בפרוייקט שלנו, זאת כיוון ש A אינו השם המלא של המחלקה, והשם המלא כולל את 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 זאת A. האם זה קורה בגלל שיש שתי מחלקות בשם A? נבדוק את זה. החליפו את שם המחלקה Aאשר 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.
2. צורת הכתיבה שהוצגה באופציה 1 היא נכונה אך לא שימושית במיוחד. בפרוייקטים שבהם יש היררכיית מחלקות עמוקה זה יותר קוד מאוד מסורבל. הדרך המקובלת לגרום למחלקה מחבילה אחת להכיר מחלקות מחבילה אחרת היא באמצעות משפטי import.

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 אנחנו מתכוונים.
מספר הערות על משפטי import:

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

ומה יקרה אם נרצה לעשות שימוש בשתי המחלקות A שקיימות בפרוייקט בתוך B?
אם ננסה לעשות import לשתיהן, זה לא יצליח, כיוון שה import מאפשר לנו להשתמש בשם "המקוצר" ובמקרה זה מדובר בשני שמות מקוצרים זהים. זה אומר שעבור המחלקה שבה נעשה שימוש רק יותר נעשה import 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 ולכן זה לא יתאפשר.

חלק ג' – classpath/jar/javadoc

בחלק זה נשתמש בקוד שנכתב במקום אחר וסופק לנו כקובץ jar שמכיל רק קבצים מקומפלים (קבצי class). בנוסף, נקבל גם קובץ jar שמכיל את התיעוד של המחלקות בפורמט של Javadoc (נסביר בהמשך).
הורידו את קובץ ה jar של ה class-ים מהקישורית הבאה:
https://courses.cs.tau.ac.il/software1/2223b/lectures/tutorials/resources/Tutorial3classes.jar

הוספת ה jar לפרוייקט ב eclipse:

1. כדאי, אך לא חובה, להעתיק את ה jar-ים שבהם משתמשים בפרוייקט לתוך הפרוייקט, רצוי בתיקיה מיוחדת (lib או resources).
2. מתוך תצוגת ה package explorer, סמנו את ה jar המבוקש, ליחצו קליק ימני, ובחרו באופציה add to build path
3. לחילופין, ניתן לבחור באופציה configure build path ולהוסיף את ה jar תחת libraries. בשיטה הזו ניתן להוסיף גם jar-ים שאינם נמצאים בתצוגה של הפרוייקט.

הוספת ה jar בהרצה משורת הפקודה (הדוגמא 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 שונים).

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 המסויימת של תוכנית ה java.

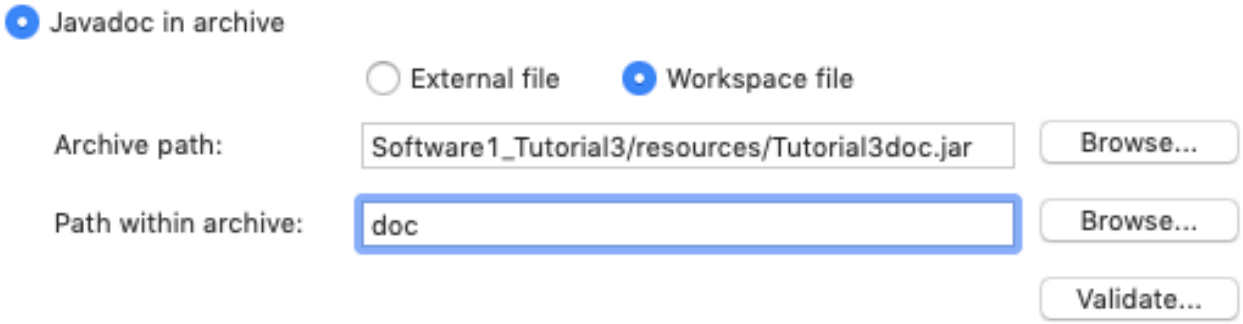
לאחר הוספת ה jar לפרוייקט ב eclipse, זו אמורה להיות תצוגת הפרוייקט:

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.

כעת נכתוב תוכנית אשר תשתמש בקוד שהורדנו, מבלי לראות את הקוד עצמו. את התוכנית עצמה נכתוב בחבילה הדיפולטית (אם כי זה לא משנה).

```java
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.
האם יכול להיות שאנחנו לא משתמשים בקוד כמו שצריך? כשמקרבים את העכבר לפונקציה mathFunc אנחנו לא מקבלים שום הסברים על אופן פעולת הפונקציה, אבל בעצם יש לנו קובץ עם כל התיעוד, ונוכל לשלב את התיעוד בתוך ה eclipse.
הורידו את ה jar שמכיל את קבצי התיעוד מהקישורית הבאה:
https://courses.cs.tau.ac.il/software1/2223b/lectures/tutorials/resources/Tutorial3doc.jar
והעתיקו אותה לתיקיית ה jar-ים.
להוספת הקישור בין קובץ ה jar-ים שמכיל את הקבצים המקומפלים לבין התיעוד, עקבו אחרי המדריך הבא:
https://stackoverflow.com/questions/9870448/how-to-attach-source-or-javadoc-in-eclipse-for-any-jar-file-e-g-javafx
שימו לב שכל 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.

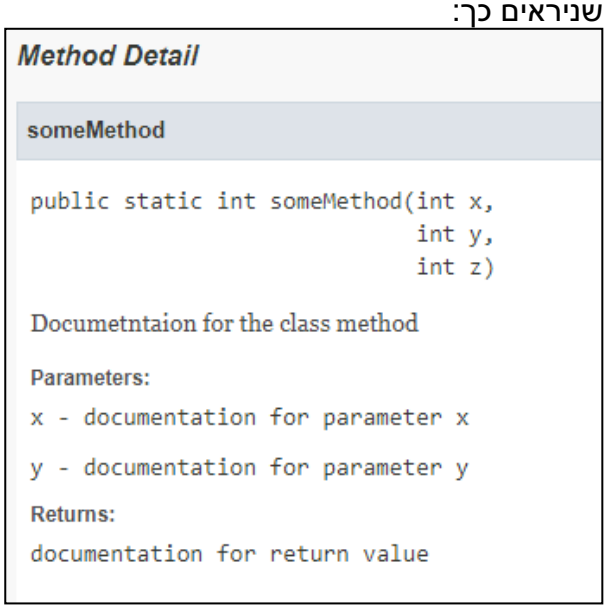
כעת,קרבו שוב את העכבר לשם הפונקציה. אתם אמורים להיות מסוגלים לראות את התיעוד. התיעוד עמום בכוונה, אך הוא מציג שני אלמנטים מרכזיים: אופי הקלט המותר ואופי הפלט. שימו לב שהפלט המובטח יתקבל רק עבור קלטים "חוקיים" – כלומר, אלה שמקיימים את ההגבלות שציינו. מה יקרה עבור קלטים לא חוקיים? אתם מוזמנים לבדוק מספר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

כיצד מייצרים את התיעוד שיופיע ב Javadoc מתוך הקוד שלכם? ניתן לחולל אותו אוטומטית באמצעות ה eclipse (Project-> generate JavaDoc) או בשורת הפקודה באמצעות הפקודה javadoc.
את התיעוד יש להוסיף באופן הבא:

```java
/** 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 /*.

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



זהו בעצם אותו הפורמט שאנחנו רואים בתיעוד ה API של Java באתר של Oracle. לדוגמא, התיעוד של המחלקה String:
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html
התיעוד של כלל הספריות:
https://docs.oracle.com/en/java/javase/17/docs/api/index.html