

לפני שמתחילים

**מטרות**

* בסדנה זו נרחיב את הידע המקצועי שלנו בתחום התכנותי
* בסדנה זו נכיר פלטפורמות יעילות יותר ממה שהכרנו בעבר

**מבנה**

* נסביר מהו Maven ועל היתרון בשימוש בו לעומת מה שהיה בעבר
* פתיחת פרויקט ב-Maven

**לפני שמתחילים וודאו שעשיתם את הדברים הבאים:**

* התקנות
  + Java SDK
  + Maven
  + סביבת עבודה IntelliJ

מה היה לפני MAVEN ?

בתקופת האבן, כאשר רצינו לייבא ספריות לתוך הפרויקט שפיתחנו ב-,Java הצטרכנו לבצע import לספרייה או לייבא קובץ Jar ולקרוא ממנו - אך באסה, כשהספרייה/Jar עודכנו על ידי המפתח (בתקווה שגילינו את זה איכשהו) נשארנו עם הגרסה הקודמת עד לרגע שנחליט להרים את הכפפה ולעדכן בפרויקט שלנו את הספרייה/Jar העדכני ולהשתמש בו, תהליך זה **מייגע בכל פעם**, במיוחד בספריות שמתעדכנות באופן תדיר.

מה זה MAVEN ?

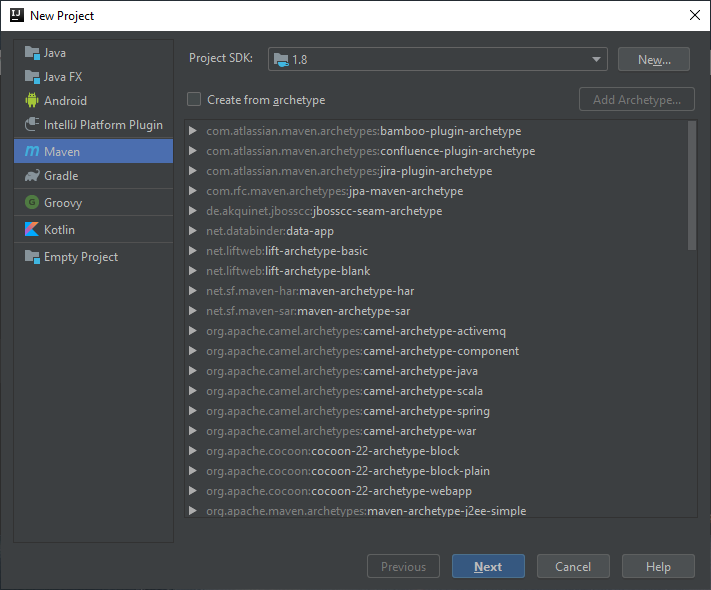
מתוך [What is Maven](https://maven.apache.org/what-is-maven.html):

Maven (מילה ביידיש שמשמעותה בתרגום חופשי – "אוגר\אספן של ידע" ,כבר נבין למה) הוא כלי לפתיחה וניהול של פרויקטים ב-Java שיצרה חברת Apache המאפשר התקנה, הסרה ועדכון של ספריות בקלי קלות, אין צורך בהוספת -Jarים.  
הכלי מאפשר לנהל ספריות ותוספים ב-Java **באופן אוטומטי**, ללא צורך במעקב מתמיד אחר הגרסאות החדשות של הספריות הללו וללא צורך בעדכון ידני של הקבצים בכל פעם שקיים אחד כזה (בעזרת התקנה מחדש של קבצי ה-Jar).  
בקצרה - Maven למעשה עוטף את פרויקט הJava שלנו ומאפשר ייבוא ספריות בצורה קלה שלא דורשת תחזוקה ברמה כמו פעם.

**בואו נתחיל!**

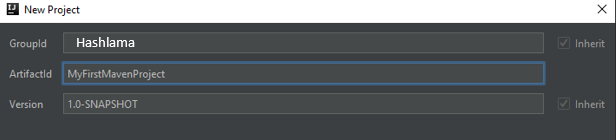
יש לקרוא את המסמך [הבא](https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html)

יצירת פרויקט MAVEN  
ראשית, ניצור פרויקט חדש דרך IntelliJ.  
נבחר ב-MAVEN (ולא ב-JAVA J)



נתקדם הלאה דלחלון הבא, ושם נמלא את פרטי הפרויקט.

* GroupID - אלמנט המייצג את המזהה של הקבוצה\ארגון שבנו את הפרויקט (לדוגמה, Even/Everest). סוג של namespace שאמור להיות globally unique. בד"כ כתובת אתר האינטרנט של החברה.
* ArtifactId - שם הפרויקט שלנו, שם ייחודי
* Version – ל qualifier בשם “SNAPSHOT” (אותיות גדולות) – יש משמעות מיוחדת, והיא משמשת לציין גרסה שעדיין בפיתוח שמשתנה כל הזמן. במקום לבדוק אם הגרסה התעדכנה בעזרת מספר הגרסה, הוא יבדוק את התאריך, כך שהמפתח לא נדרש לשנות כל רגע מספר גרסה בכדי שחבריו יקבלו גרסה עדכנית בכל build



מעולה ! פתחנו פרויקט JAVA עם ! MAVEN JJJ

נפתח לנו פרויקט עם כל מיני תיקיות. לפני שנדבר על מבנה הפרויקט, ניתן לראות שקיים קובץ שנקרא Pom.xml.

**אז מה זה בכלל ה-Pom.xml הזה ?**

קובץ זה מהווה את ליבת הקונפיגורציה של Maven, ופה גם נגדיר את התלויות שבהן נשתמש. כלומר, ייבוא הספריות שהזכרנו מקודם, יתנהלו מפה!

[קראו](https://maven.apache.org/guides/introduction/introduction-to-the-pom.html) כאן על הקובץ POM

**ואיפה אני בכלל מוצא תלויות להוסיף לפרויקט שלי ?**בקישור הבא [MavenRepository](https://mvnrepository.com/) - נבצע חיפוש של מה שאנחנו רוצים להוסיף, נעתיק את התלות המוצעת ונדביק בקובץ pom.xml.

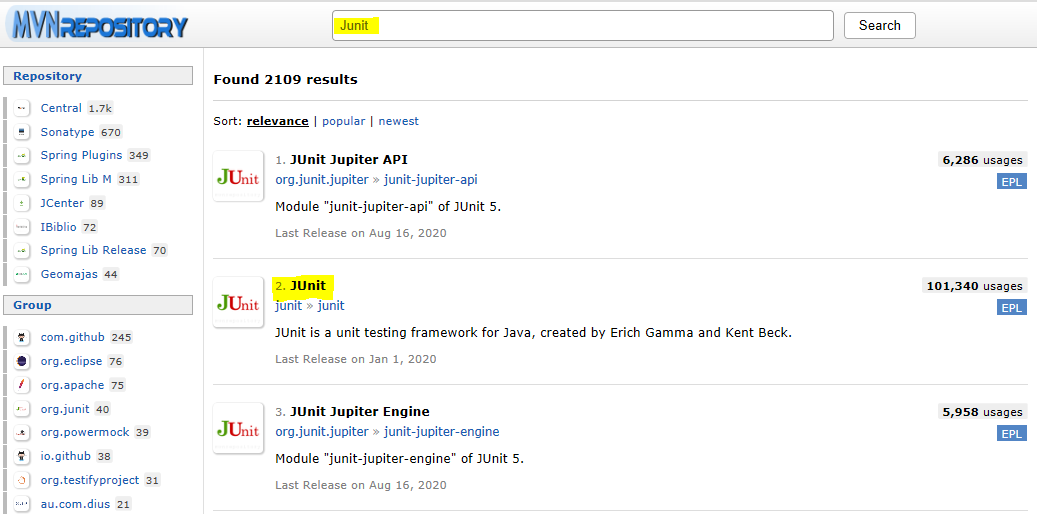
**ואיפה שמים הכל אתם שואלים?**

את התלויות נגדיר בתוך תגית ה- dependencies (תלויות) שלנו (יש להוסיף אותן באופן עצמאי):

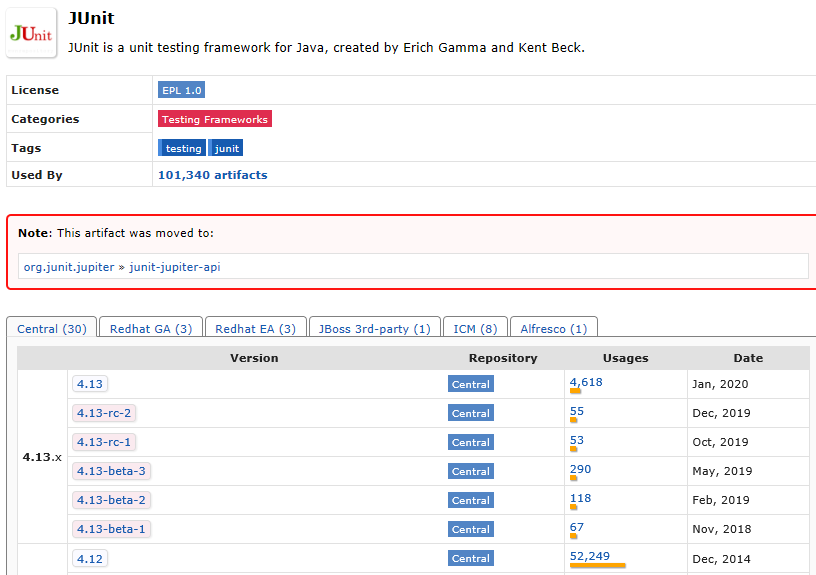
<dependencies>  
 *<!-- here all our dependencies will be !-->*  
</dependencies>

**לדוגמה, הוספת התלות Junit תראה כך:**

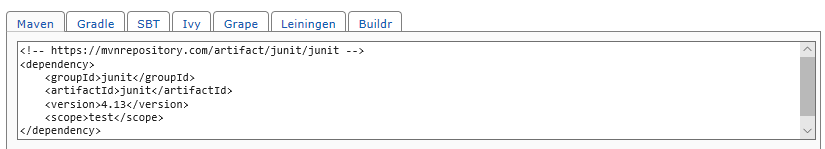
ראשית, נחפש את הגרסה הרצויה של Junit במאגר ה - [MavenRepository](https://mvnrepository.com/) שלנו:



נבחר את הספרייה שאנו רוצים לקשר לפרויקט, ונגיע לטבלה המציגה את הגרסאות של הספרייה שנבחרה:



לאחר שנכנס לגרסה שאנו רוצים, נמצא את כל הפרמטרים החשובים לנו בתיבת הטקסט תחת הקטגוריה של Maven:



נעתיק את הפרמטרים האלו לפרויקט שלנו (זוכרים? בקובץ ה – Pom.xml):

<dependencies>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.12</version>  
 <scope>test</scope>  
 </dependency>  
</dependencies>

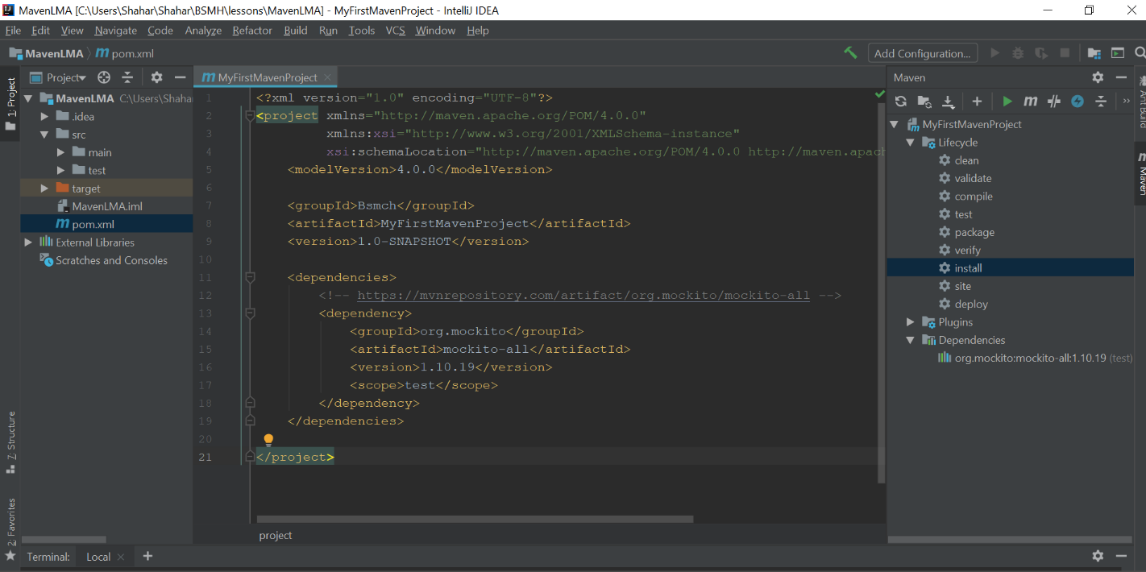
כעת, כל מה שנותר לעשות הוא לסנכרן את התלויות שהוספנו עם הפרויקט.

איך עושים זאת?

Goals & Phases

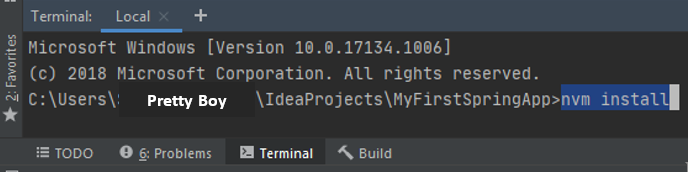
יש לקרוא באתר [זה](https://www.baeldung.com/maven-goals-phases) עד "4. Maven Goal" כולל (אין צורך לקרוא את Maven Plugin ומעבר).

עכשיו כשהבנו מה זה, נבצע mvn install באמצעות intelijj:



וכמו שקראנו, כל השלבים (הphases) שלפני יתבצעו - ונקבל את כל הקבצים שהיינו צריכים.

נעשה זאת באמצעות הפקודה בטרמינל שלנו >> mvn install :



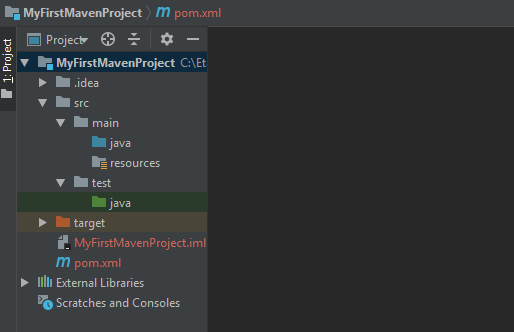
איזה כיף! עכשיו הספריות שלנו מנוהלות **באופן אוטומטי**.

אחרי שהבנו את הדבר העיקרי שמולו נעבוד בפרויקט מסוג Maven, בואו נדבר על מבנה התיקיות שנפתח לנו.

# מבנה התיקיות

אז הבנו איך ליצור פרויקט ואיך Maven עובד, אבל עד עכשיו לא ראינו אפילו קובץ אחד של Java. בואו נבין איפה זה נכנס בפרויקט שלנו:

מתוך [Maven Directory Structure](https://www.baeldung.com/maven-directory-structure):

└───maven-project

    ├───pom.xml

    ├───README.txt

    ├───NOTICE.txt

    ├───LICENSE.txt

    └───src

        ├───**main**

        │   ├───**java**

        │   ├───filters

        │   └───webapp

        ├───**test**

        │   ├───**java**

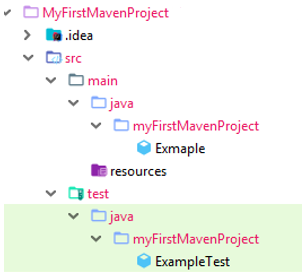
        │   ├───resources

        │   └───filters

        ├───it

        ├───site

        └───assembly



אם נסתכל על המבנה שנוצר לנו,

ניתן לראות עץ הכולל שני חלקים עיקריים ->

MAIN – בעץ זה יישמרו קבצי המקור שלנו, תחת תיקיית java – יאללה לכתוב קוד!

TEST – בעץ זה יישמרו קבצי הבדיקות שלנו, תחת תיקיית java.

כמה הסברים:

* תיקיית src נועדה לקוד, ותיקיית target לתוצרי קומפילציה (קבצי class., למשל).
* תיקיית ה src/main נועדה לקוד שמיועד לדילוור, ותחתיה יש תיקיה לכל שפת תכנות. יכולה להיות, למשל, תיקיה ל java, ל scala ול javaScript. ספריית ה src/test נועדה לקוד של בדיקות יחידה / אינטגרציה. בפנים יש תיקיות של ה java packages – כמו בכל פרויקט java רגיל.
* תיקיית src/main/resources/ נועדה לקבצים אחרים של הפרויקט שאינם קוד, למשל קבצי הקונפיגורציה של Spring Framework.
* את תיקיית ה target מחלקים לתוצרים של קוד שהולך לדילוור (classes) ותיקיית הקוד שלא הולך לדלוור (test-classes).
* pom.xml הוא קובץ הקונפיגורציה של Maven, המקבילה של build.xml של Ant.
* קובץ ה pom.xml (קיצור של Project Object Model) הוא קובץ XML, דקלרטיבי, שמגדיר את ה Strategy של תהליך ה build שלנו.

**רוצים לחקור עוד?**

<https://maven.apache.org/guides/getting-started/>

**חשוב !**  
שימו לב שלצורך מימוש Spring (לא לדאוג, תלמדו עליו בשלב הבא) בפרויקט שלנו, נוסיף package כשם הפרויקט עם אות קטנה תחת תיקיית Java ורק שם נפתח את המחלקות שלנו.