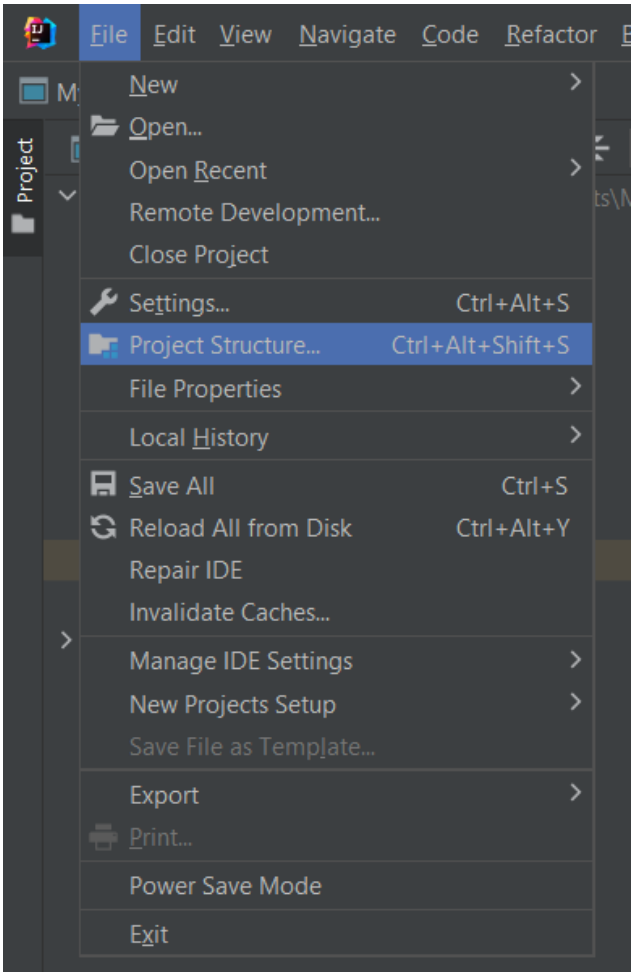
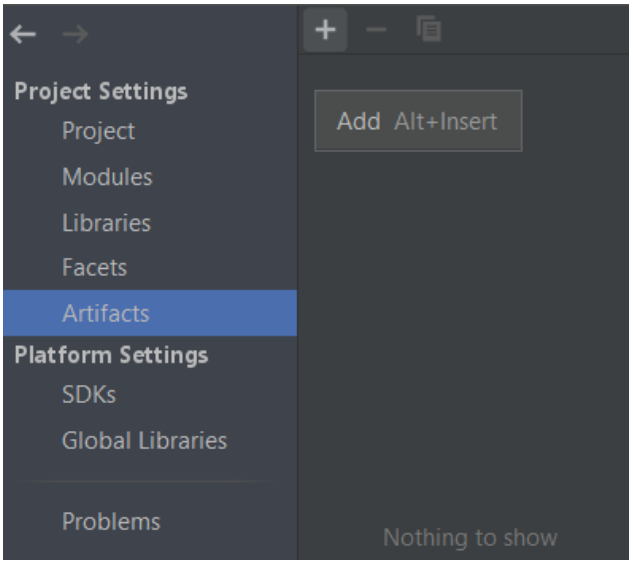
איך יוצרים JAR בIntelliJ?

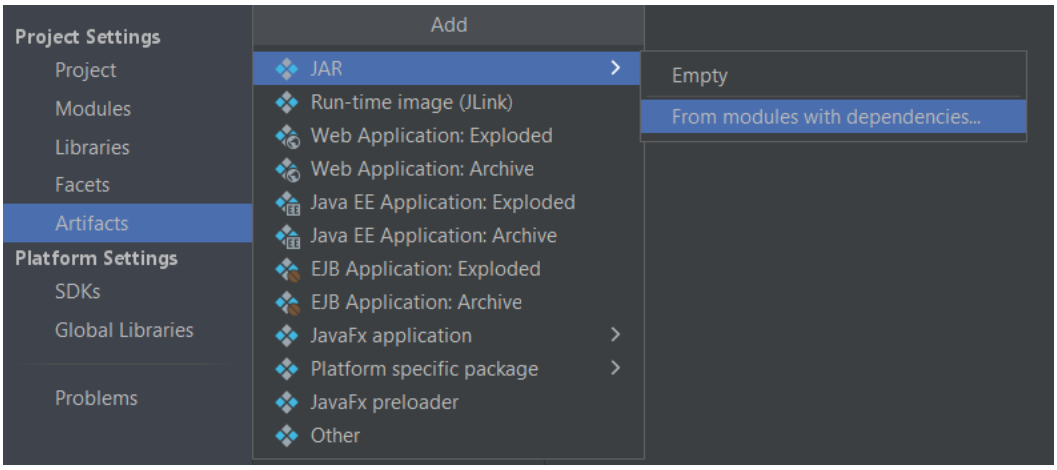
דרך ראשונה –

1. לך ל-project structure :

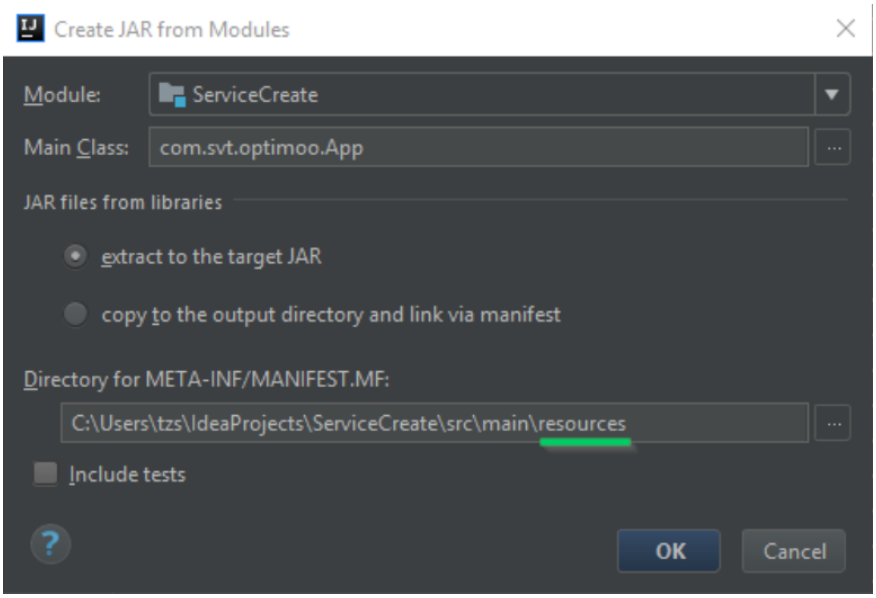


2. יצר Artifact חדש:



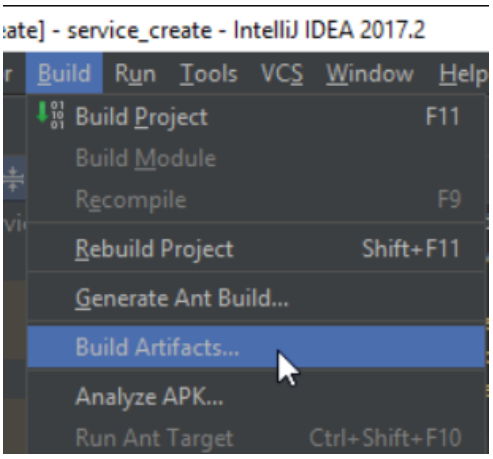


3. בחר את המחלקה הראשית (main class) ושנה את ה-manifest folder:

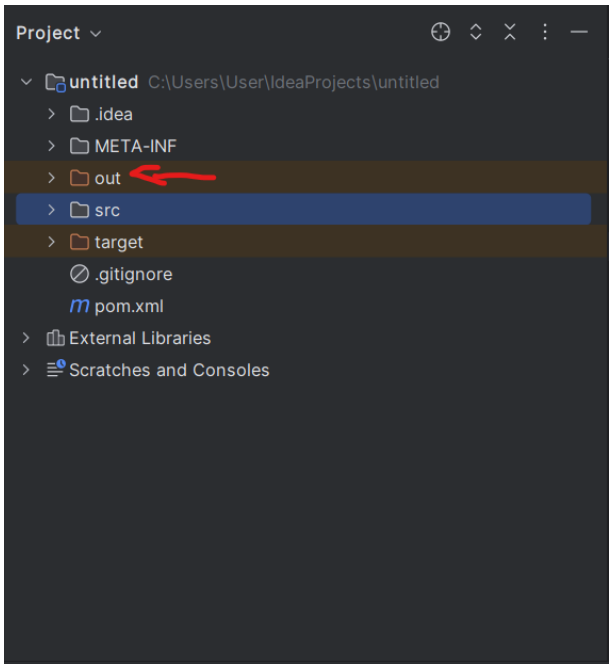


4. בחר את התלויות (dependencies) שאתה רוצה שיארזו לקובץ JAR.

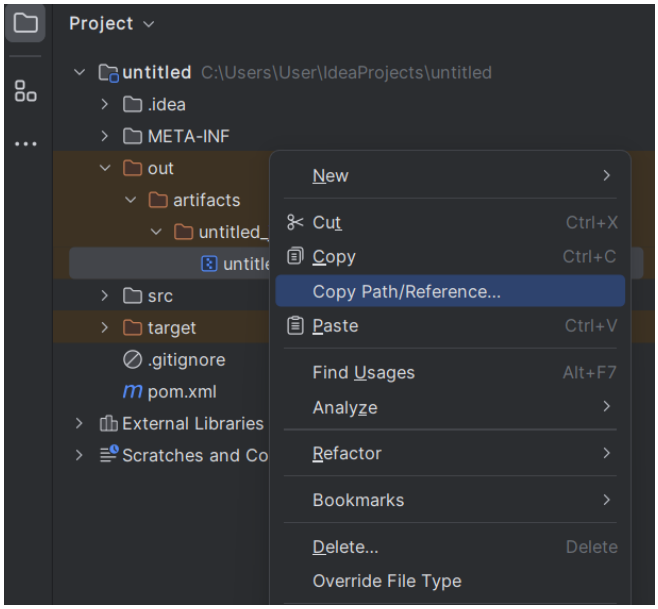
5. כדי לייצר את ה-Artifact לך לבחר build artifacts ובחר "rebuild". תיקיית out תיווצר עם קובץ ה-jar והתלויות שלו.



6. נוצר לכם תיקייה חדשה בשם OUT כאן:



7. לפתוח אותה ולהעתיק את ה PATH (אחרי הלחיצה בוחרים את ה absolute path) כך:



8. כדי לוודא שהקובץ JAR עובד כמו שצריך פותחים את ה CMD וכתובים כך:

```
C:\Users\User>java -jar C:\Users\User\IdeaProjects\untitled\out\artifacts\untitled_jar\untitled.jar
```

דרך שניה –

יש שתי דרכים לעשות זאת, האחת בעזרת maven-assembly-plugin והשנייה בעזרת maven-shade-plugin. הצורה הראשונה מומלצת לפרויקטים ללא JavaFX והשנייה מומלצת לפרויקטים שמשתמשים ב־JavaFX.

1. בעזרת maven-assembly-plugin

עלינו להוסיף לאלמנט project->build->plugins (או ליצור אחד) תת אלמנט חדש, כך:

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>single</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <archive>
      <manifest>
        <mainClass>com.example.hellolab.App</mainClass>
      </manifest>
    </archive>
    <descriptorRefs>
      <descriptorRef>jar-with-dependencies</descriptorRef>
    </descriptorRefs>
  </configuration>
</plugin>
```

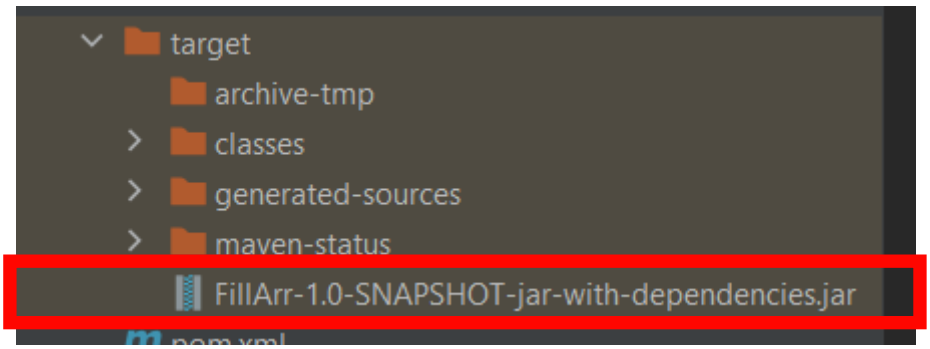
נשמור את הקובץ ונסנכרן אותו עם Maven.

כעת, נלחץ לחיצה ימנית על שם הפרויקט ונבחר ב־...Run as->Maven build. בחלון שנפתח, ב־Goals, נכתוב את השורה הבאה:

```
clean compile assembly:single
```

בעצם, אנחנו אומרים ל־Maven לבצע שלושה דברים בזה אחר זה: לנקות שאריות מבניות קודמות של הקוד, לקמפל את הקוד ולהכין לנו את ה־JAR בעזרת maven-assembly-plugin.

לאחר מכן, בתוך התיקייה target שבפרויקט, נראה קובץ JAR עם הסיומת jar-with-dependencies-:



ונוכל להריצו משורת הפקודה עם java -jar (שימו לב לעשות זאת בתיקייה הנכונה).

בעזרת maven-shade-plugin

היתרון של השימוש בשיטה זו הוא שאפשר לכלול את כל הנדרש להרצה, כולל ספריות שכתובות ב־C (native libraries) ומשתמשים בהן מתוך Java - למשל, בתוך JavaFX יש שימוש נרחב בספריות כאלה, על מנת ליצור ממשק משתמש מותאם למערכת ההפעלה. כמו בדרך הראשונה, נתחיל עם pom.xml. נוסיף לו ב־project אלמנט של dependencies (אם לא קיים עדיין) עם התוכן הבא (או נוסיף את התגיות לאלמנט שכבר קיים):

```
<dependency>
  <groupId>org.openjfx</groupId>
  <artifactId>javafx-base</artifactId>
  <version>14</version>
</dependency>
<dependency>
  <groupId>org.openjfx</groupId>
  <artifactId>javafx-controls</artifactId>
  <version>14</version>
</dependency>
<dependency>
  <groupId>org.openjfx</groupId>
  <artifactId>javafx-fxml</artifactId>
  <version>14</version>
</dependency>
<dependency>
  <groupId>org.openjfx</groupId>
  <artifactId>javafx-graphics</artifactId>
  <version>14</version>
  <classifier>win</classifier>
</dependency>
<dependency>
  <groupId>org.openjfx</groupId>
  <artifactId>javafx-graphics</artifactId>
  <version>14</version>
  <classifier>linux</classifier>
</dependency>
<dependency>
  <groupId>org.openjfx</groupId>
  <artifactId>javafx-graphics</artifactId>
  <version>14</version>
  <classifier>mac</classifier>
</dependency>
```

בצורה כזו, אנחנו אומרים ל־maven שהפרויקט שלנו משתמש בספריות של JavaFX, וכן מציינים שעליו לכלול את הגרסה המתאימה של javafx-graphics (שמכילה קוד ספציפי למערכות הפעלה שונות) לכל מערכות ההפעלה.

לאחר מכן, ל־project->build->plugins נוסיף את ה־plugins הבאים:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.8.0</version>
  <configuration>
    <release>13</release>
  </configuration>
</plugin>
<plugin>
  <groupId>org.openjfx</groupId>
  <artifactId>javafx-maven-plugin</artifactId>
  <version>0.0.3</version>
  <configuration>
    <mainClass>org.cshaifasweng.winter.Main</mainClass>
  </configuration>
</plugin>
```

יש להחליף את org.cshaifasweng.winter.Main בנתיב המלא למחלקה עם ה־Main
(המשך בעמוד הבא מפאת גודל האלמנט)

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-shade-plugin</artifactId>
  <version>3.2.1</version>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>shade</goal>
      </goals>
      <configuration>
        <transformers>
          <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
            <mainClass>org.cshaifasweng.winter.Main</mainClass>
          </transformer>
        </transformers>
      </configuration>
    </execution>
  </executions>
</plugin>
```

יש להחליף את org.cshaifasweng.winter.Main לנתיב המלא למחלקה הראשית בהתאם.

כאן מגדירים את תהליך הבנייה של הפרויקט. לצורך עבודה טובה יותר עם JavaFX ישנו הפלאגין השני (אם כי יתכן שאינו חובה) ולבסוף יש את ההוספה של maven-shade-plugin. מגדירים לו להשתמש ב־ ManifestResourceTransformer על מנת שיכלול משאבים נוספים של הפרויקט (למשל, קובצי fxml) וכמובן מגדירים שוב את שם המחלקה שבה ישנו ה־Main.

חשוב: עקב בעיה ב־JavaFX, אם יוצרים JAR לפרויקט עם ממשק גרפי, עלינו ליצור קובץ נפרד עם Main ובו לקרוא ל־main של המחלקה שיורשת מ־Application. שימו לב להגדיר את Maven אליו ולא אל הקובץ המקורי.

לאחר מכן, ניצור את ה־JAR ב־IntelliJ: נלחץ לחיצה ימנית על שם הפרויקט ונבחר ב־...Run as->Maven Build וב־ goal נכתוב clean package. נלחץ על Run וה־JAR יווצר לנו בתיקייה target. שימו לב שיווצר אחר שמתחיל ב־ original - אין צורך להשתמש בו וניתן למחוק אותו. הקובץ שמתחיל בשם הפרויקט שלנו הוא הקובץ הנכון ונוכל להריצו בעזרת java -jar <JAR name>.

הוספת קובצי המקור

```
<resources>
  <resource>
    <directory>src/main/java</directory>
  </resource>
  <resource>
    <directory>src/main/resources</directory>
  </resource>
</resources>
```

על מנת להוסיף את קובצי המקור (.java) לתוך ה־JAR, עליכם להוסיף מספר שורות בתוך האלמנט build:
בעצם, אנחנו אומרים ל־Maven לשמור בתוך ה־JAR שתי תיקיות חשובות: src/main/java־ו src/main/resources־,
כולל כל הקבצים שבתוכן. במעבדה האחרונה נרצה לשמור גם את קובצי הבדיקות - ניתן להוסיפם בצורה דומה.