מעבדה מספר 5 – OCSF ו - EventBus

מטרת המעבדה היא לעשות שימוש בתוכנת OCSF ליצירת אפליקציה פשוטה של Client-Server העושה שימוש בEventBus לצורך שליחת הודעות ללקוח מסוים/לכל המנויים בהתאם לסוג הבקשה המתקבלת.
הקובץ המובא לפניכם כולל הסבר על מבנה התוכנה בה נעשה שימוש ובסופו משימות לביצוע. בנוסף בלינק הבא תמצאו פרויקט התחלתי ב־Github:

https://classroom.github.com/a/L4Oj_l-z

הפרויקט הנ"ל מכיל את קוד תוכנת ה־OCSF המאפשרת החלפת הודעות בין Client ל־Server תוך שימוש ב־OCSF וב-EventBus.

1. Frameworks

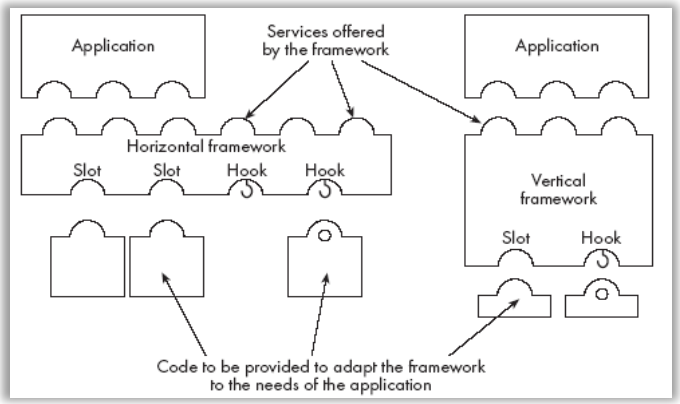
Framework היא תוכנה ברת שימוש חוזר אשר מממשת פתרון גנרי לבעיה כללית. היא מספקת שירותים שניתן ליישמם בתוכניות שונות. השימוש בה מבוסס על העיקרון שאפליקציות תוכנה הן שונות אבל יש בהן מן המשותף. במהותה ה־Framework אינה שלמה.

- היא משתמשת במחלקות ובמתודות שחסרות בה (**slots**).
- חלק מהפונקציונאליות היא אפשרית – המשתמש יכול להוסיפה (חלקים אלה נקראים **hooks** או **extension points**).
- מפתחים משתמשים בשירותים שה־framework מספקת (שירותים אלה נקראים Application Program Interface – API).

בגישה של תכנות מונחה עצמים ה־framework מורכבת מספריה של מחלקות.
ה־API מגדיר קבוצה של מתודות ציבוריות של המחלקות. חלק ממחלקות אלה תהיינה אבסטרקטיות ובנוסף הוא כולל ממשקים.

סוגים של frameworks

- Horizontal framework – מספקת שירותים כלליים שיכולים להיות שימושיים למספר גדול של אפליקציות.
- Vertical framework – היא יותר "שלמה" אבל עדיין כוללת מספר slots שיש להשלים כדי להתאימה לצרכים של אפליקציה מסוימת.



2. ארכיטקטורה של Client-Server

מערכת מבוזרת היא מערכת אשר:

החישובים בה מתבצעים בתוכניות נפרדות. על פי רוב מתבצעת על יחידות חומרה שונות שמשתפות פעולה כדי לבצע משימה של מערכת.

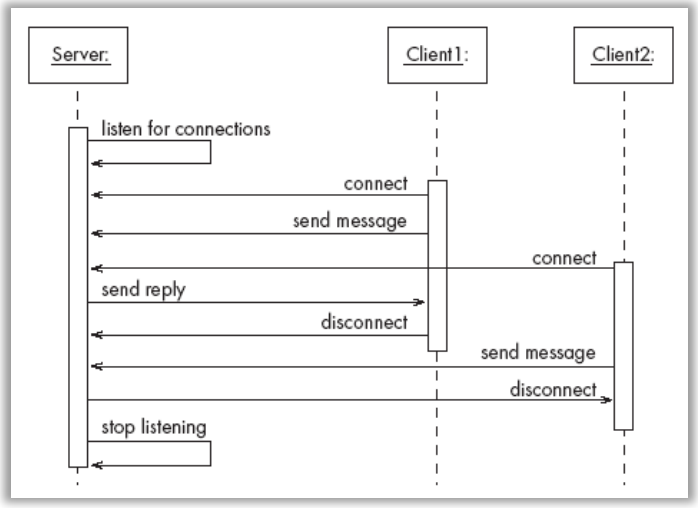
ה־Server היא תוכנית שירותים לתוכניות אחרות המחוברות אליה ע״י שימוש בערוץ תקשורת.

ה־Client היא תוכנית שנגשת ל־server (לאחד או יותר) כדי לקבל שירותים.

ה־server יכול לספק שירותים לכמה clients במקביל.

דוגמאות: רשת האינטרנט, Email, מערכות תקשורת, מערכת של database משותף למספר מחשבים.

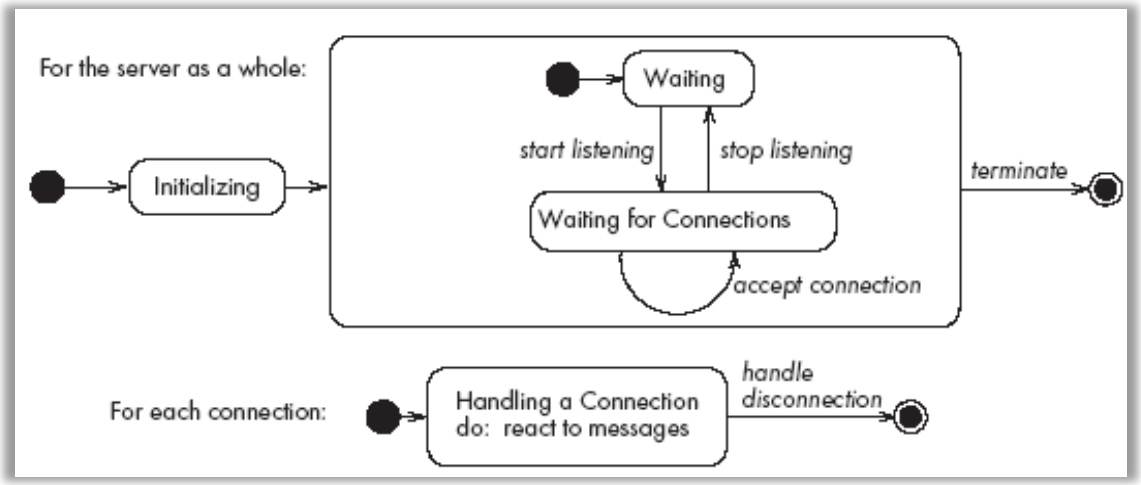
תקשורת של server עם שני clients



פעילויות של ה־server

1. אתחול עצמי.
2. להתחיל להקשיב ל־clients.
3. לטפל בסוגים שונים של אירועים הנוצרים על ידי ה־clients.
   - לטפל ב־connections.
   - להגיב להודעות.
   - לטפל ב־client disconnection
4. להפסיק להקשיב לעיתים.
5. לסיים בצורה מסודרת את הביצוע שלו.

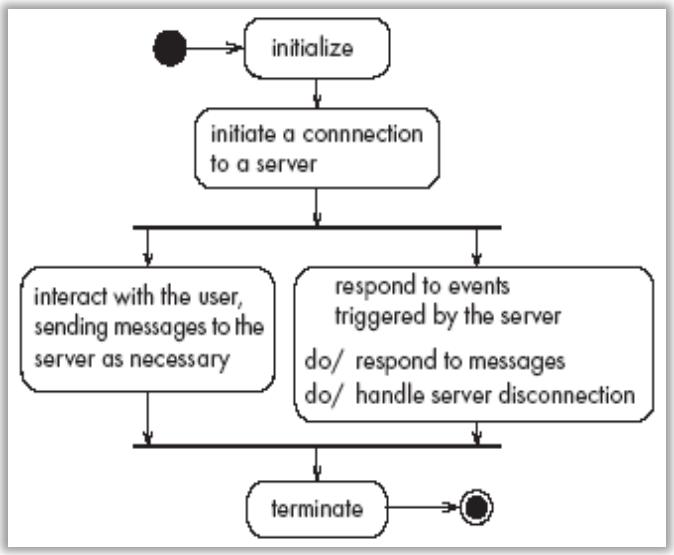
להלן תרשים פעילות של server :



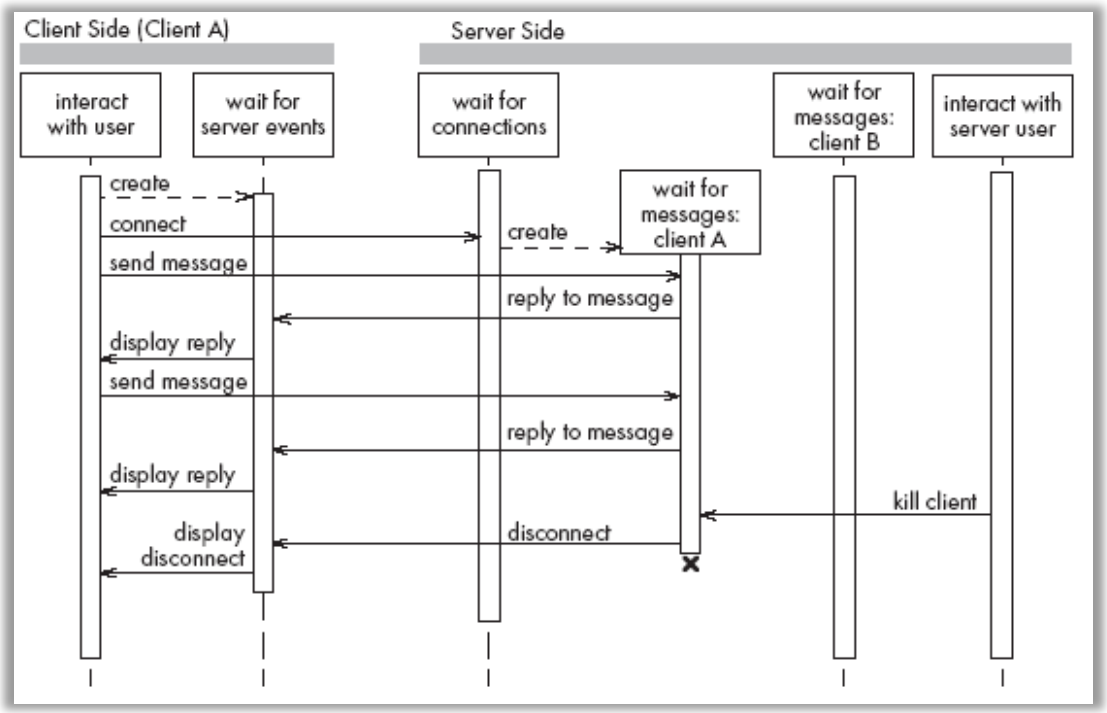
<u>פעילויות של ה־client</u>

1. אתחול עצמי.
2. להתחיל connection.
3. לשלוח הודעות.
4. לטפל בסוגים שונים של אירועים הנוצרים על ידי ה־server :
   - להגיב להודעות.
   - לטפל ב־server disconnection
5. לסיים בצורה מסודרת את הביצוע שלו.

להלן תרשים פעילות של client:



חוטים במערכת server-client:



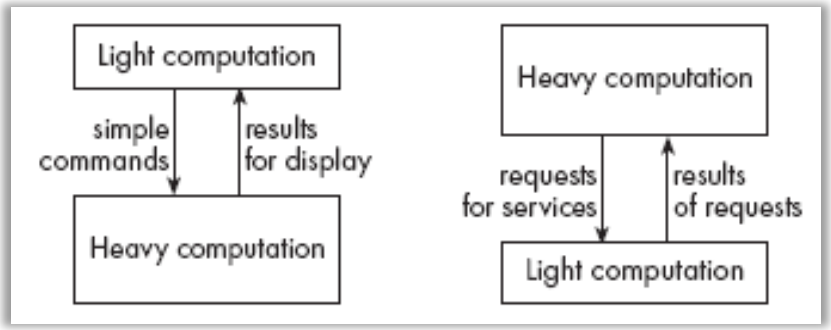
"Thin client" לעומת "fat client"

מערכות Thin-client(a)

- ה־client קטן ככל האפשר.
- רוב העיבוד נעשה ע״י ה־server.
- ניתן ״להוריד״ את תוכנת ה־client בקלות באמצעות הרשת.

מערכות Fat-client(b)

- ה־client מבצע חלק גדול מעיבוד המערכת.
- ה־server יכול לטפל ביותר clients כיוון שהעיבוד שנעשה על ידיו מצומצם יותר.



3. **<u>פרוטוקולי תקשורת</u>**

ההודעות המועברות בין ה־clients ל־server יוצרות שפה. גם ההודעות מה־server ל־clients יוצרות שפה.
כאשר הם "מדברים" הם משתמשים בשפות אלה. שתי השפות מגדירות **פרוטוקול תקשורת.**

4. **<u>משימות שיש לבצע בפיתוח מערכת Client-Server:</u>**

1. יש לתכנן את המשימות שעל ה־server וה־client לבצע בתחילת העבודה.
2. יש לתכנן את ביזור עבודת המערכת בין ה־client וה־server.
3. יש לתכנן את ההודעות.
4. יש לתכנן את המנגנונים ל:
   א. אתחול
   ב. טיפול ב־connections
   ג. שליחה וקבלת הודעות
   ד. סיום

5. **<u>טכנולוגיות נדרשות למערכות Client – Server</u>**

Internet Protocol (IP)

- מאפשר משלוח הודעות ממחשב אחד לשני.
- הודעות ארוכות מפוצלות לקטעים קטנים.

Transmission Control Protocol (TCP)

- מטפל ב־connections בין שני מחשבים.
- מחשבים יכולים להחליף הרבה הודעות IP באמצעות ה־connection.
- מבטיח שהודעות התקבלו בצורה משביעת רצון.

ל־host יש כתובת IP ושם.

- מספר servers יכולים להתבצע על אותו host.
- כל server מזוהה ע"י מספר port (0 – 65535).
- כדי להתחיל תקשורת עם server, ה־client צריך לדעת אם מספר ה־port ואת שם ה־host.

6. **<u>כינון connection ב־Java</u>**

החבילה java.net package מאפשרת יצירת TCP/IP connection בין שתי אפליקציות.
The java.net package
לפני הקמת ה־connection, ה־server חייב להתחיל להקשיב לאחד מה־ports:

```
ServerSocket serverSocket = new ServerSocket(port);
Socket clientSocket = serverSocket.accept();
```

ליצירת connection בין ה־client ל־server

```
Socket clientSocket = new Socket(host, port);
```

7. **<u>החלפת מידע ב־Java</u>**

כל תוכנית משתמשת במופעים של המחלקות (שנמצאות בחבילה java.io):

InputStream – לקבלת הודעות מתוכנית אחרת.

OutputStream – לשלוח הודעות לתוכנית אחרת.

```
output = clientSocket.getOutputStream();
input = clientSocket.getInputStream();
```

<u>משלוח וקבלת הודעות</u>

without any filters (raw bytes)

```
output.write(msg);
msg = input.read();
```

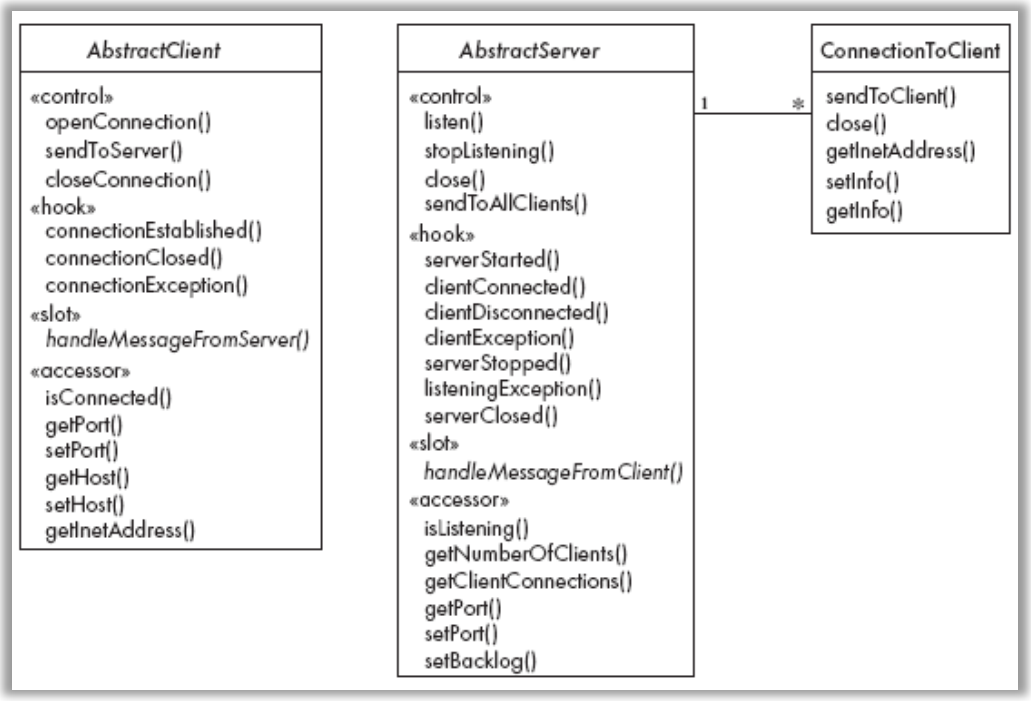
- **or using** DataInputStream / DataOutputStream **filters**

```
output.writeDouble(msg);
msg = input.readDouble();
```

- **or using** ObjectInputStream / ObjectOutputStream **filters**

```
output.writeObject(msg);
msg = input.readObject();
```

8. **<u>The Object Client-Server Framework (OCSF)</u>**



שימוש ב־OCSF

- אין משנים את שלושת המחלקות של ה־OCSF.
- יוצרים תת מחלקות המרחיבות את המחלקות AbstractClient ו־AbstractServer.
- קוראים למתודות הציבוריות של ה־framework.
- דורסים מתודות slot ו־hook (שתוכננו מיוחד לפעולת הדריסה).

צד ה־Client

מורכב ממחלקה בודדת בשם AbstractClient. חובה להרחיבה.

כל תת מחלקה צריכה לספק מימוש למתודה handleMessageFromServer שמטפלת בקבלת הודעה מה־server.

המחלקה מממשת את הממשק Runnable – כלומר יש בה מימוש למתודה run

הממשק הציבורי של המחלקה AbstractClient

Controlling methods:

- openConnection
- closeConnection
- sendToServer

Accessing methods:

- isConnected
- getHost
- setHost
- getPort
- setPort
- getInetAddress

מתודות שניתן לדרוס:

- connectionEstablished
- connectionClosed

מתודות שחובה לממש:

- handleMessageFromServer

הנחיות לשימוש במחלקה AbstractClient

יש ליצור תת מחלקה של AbstractClient

יש לממש את המתודה handleMessageFromServer

יש לכתוב קוד ש:

- יוצר מופע של המחלקה המרחיבה את AbstractClient.

- קורא למתודה openConnection.
- שולח הודעות ל־server באמצעות המתודה sendToServer.

יש לממש את המתודות connectionClosed ו־connectionException.

משתני מופע:

- Socket השומר את כל המידע על הקשר עם ה־Server.
- שני Streams : ObjectOutputStream ו־ObjectInputStream.
- חוט אשר רץ ומבצע את מתודת ה־run של ה־AbstractClient.
- שני משתנים אשר מאחסנים את ה־host וה־port של ה־server.

צד ה־Server

כולל שתי מחלקות :

- **AbstractServer** – החוט אשר מקשיב ל־connections חדשים.
- **ConnectionToClient** –חוטים אשר מטפלים ב־connectios ל־clients.

הממשק הציבורי של המחלקה AbstractServer

Controlling methods:

- listen
- stopListening
- close
- sendToAllClients

Accessing methods:

- isListening
- getClientConnections
- getPort
- setPort
- setBacklog

מתודות שניתן לדרוס:

- serverStarted
- clientConnected
- clientDisconnected
- clientException
- serverStopped
- listeningException
- serverClosed

מתודות שחובה לממש:

- handleMessageFromClient

<u>הממשק הציבורי של המחלקה ConnectionToClient</u>

Controlling methods:

- sendToClient
- close

Accessing methods:

- getInetAddress
- setInfo
- getInfo

<u>הנחיות לשימוש במחלקות AbstractServer ו־ConnectionToClient</u>

יש ליצור תת מחלקה של AbstractServer.

יש לממש את המתודה handleMessageFromClient.

יש לכתוב קוד ש:

- יוצר מופע של המחלקה המרחיבה את ה־AbstractServer .
- קורא למתודה listen.
- שולח הודעות ל־clients בעזרת המתודות getClientConnections ו־sendToClient או sendToAllClients.

ניתן לממש מתודות נוספות.

<u>המימוש הפנימי של AbstractServer ו־ConnectionToClient</u>

- המתודות setInfo ו־getInfo עושות שימוש במחלקה **HashMap**.
- מתודות רבות בחלק של ה־server הן synchronized.
- אוסף המופעים של ConnectionToClient מאוחסן במחלקה מיוחדת הנקראת ThreadGroup.
- ה־server צריך לעצור את ההקשבה כל 500ms כדי לראות אם המתודה stopListening נקראה. אם לא, יש לחדש את ההקשבה מיד.

9. **דפוס תכן (Design pattern) –**

פתרון עקרוני לבעיית תכן מונחה עצמים החוזרת על עצמה. הפתרון מוצג כתבנית שיטתית הכוללת ארגון כללי של עצמים ומחלקות לפתרון הבעיה. התבנית הינה כללית מאוד ודורשת התאמה עבור פתרון ספציפי לבעיה שלנו. ישנם 3 סוגים של דפוסי תכן :

- דפוסי מבנה – דפוס תכן המשמש להגדרת מבנים מסובכים.
  דוגמאות : Adapter, Decorator.
- דפוסי יצירה – דפוס תכן המשמש להגדרה של יצירת אובייקטים בזמן ריצה.
  דוגמאות : Singleton, Abstract Factory
- דפוסי התנהגות – דפוס תכן המשמש להקצאת התנהגות מסוימת לאובייקטים בזמן ריצה. העברת המיקוד מזרימת הבקרה אל אופן התקשורת בין האובייקטים.
  דוגמאות : Iterator, Observer

10. **דפוס התכן Observer –**

- דפוס התנהגות
- תבנית להגדרת תלות בין אובייקט אחד למספר אובייקטים התלויים בו כך שכאשר האובייקט משנה את מצבו, כל האובייקטים התלויים בו מעודכנים אוטומטית.
- דוגמה – מכירה פומבית, כאשר ניתן לרשום כמה משתתפים שנרצה ולכל אחד מהם תהיה האפשרות להציע מחיר על המוצר שמוצע למכירה. ברגע שקונה מסוים יציע מחיר – כל שאר הקונים יקבלו עדכון על כך בזמן אמת.

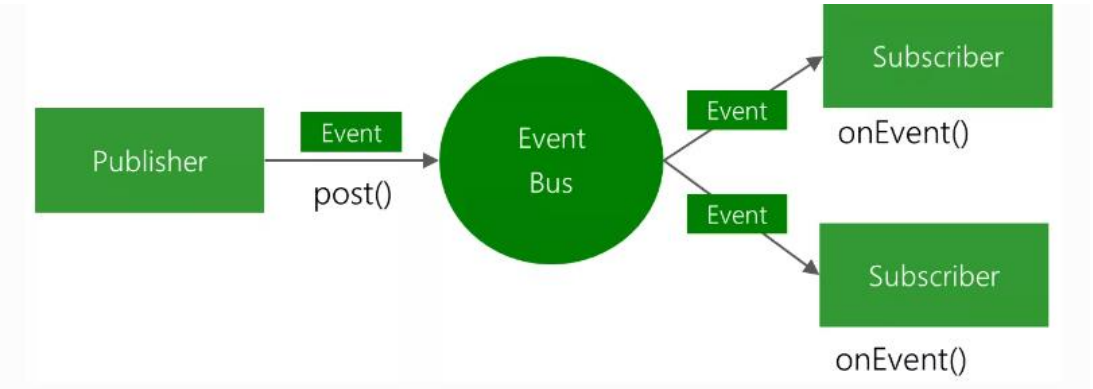
11. **דפוס התכן Publisher/Subscriber –**

- ידוע גם כ Pub\Sub
- הרחבה של דפוס התכן Observer.
- תבנית להגדרת תקשורת בין מפרסמים ומנויים. כאשר המפרסם יפרסם הודעה חדשה כל המנויים שלו יקבלו את ההודעה אוטומטית.
- דוגמה - חנות אינטרנטית בעלת רשימת תפוצה. כאשר יש עדכון בחנות (הגיע מוצר חדש/יש מבצע שווה/שינוי מדיניות וכו) נרצה שכל הלקוחות יקבלו התראה על כך. בעזרת Observer נוכל לממש זאת בקלות.
-

12. **EventBus –**

- ספריית open source ב-Java של GreenRobot המבוססת על דפוס התכן Pub\Sub.
- EventBus מאפשרת לנו לייצר אירועים שונים ולפרסם אותם על ה"אוטובוס" כך שמחלקות שנרשמו לEventBus יקבלו התראה על האירוע החדש שהתפרסם. אם למחלקה הרשומה יש מתודה שנרשמה כמנויה לסוג אירוע זה היא תיקרא אוטומטית.

- EventBus מאפשר לנו הרצה חלקה של הClient ועדכונו בצורה אוטומטית במקרה הצורך.



13. SimpleChat - אפליקציה להחלפת הודעות בין client ל־server

ה־Server

ה־SimpleChatServer היא המחלקה הראשית של השרת.

- המתודה main יוצרת מופע חדש של המחלקה SimpleServer המרחיבה את ה־AbstractServer.
- המופע מקשיב ל־clients ומטפל ב־connections עד שה־server מופסק.
- המתודה handleMessageFromClient מטפלת בהודעות שהתקבלו מהמשתמש ועונה לו בהתאם לבקשה.
- המתודה handleMessageFromClient קוראת ל־sendToAllClients כשנדרשת שליחה רחבה לכל הלקוחות (וזו שולחת את ההודעה לכל הלקוחות המנויים).

Key code in SimpleChatServer

```
public static void main( String[] args ) throws IOException
{
    server = new SimpleServer(3000);
    System.out.println("server is listening");
    server.listen();
}
```

Key code in SimpleServer

```
@Override
protected void handleMessageFromClient(Object msg, ConnectionToClient
client) {
   Message message = (Message) msg;
   String request = message.getMessage();
   try {
     if (request.equals("echo Hello")){
        message.setMessage("Hello World!");
        client.sendToClient(message);
  }
```

ה־Client

ה- SimpleChatClient היא המחלקה הראשית של הלקוח.

- המתודה start יוצרת מסך חדש ללקוח ומופע חדש של המחלקה SimpleClient המרחיבה את ה־ AbstractClient. כמו כן היא מתחברת לשרת בעזרת קריאה למתודה openConnection.
- המתודה start רושמת את הלקוח לEventBus.
- SimpleClient שולח בקשות לשרת בעזרת המתודה sendToServer.
- המתודה handleMessageFromServer מטפלת בהודעות שהתקבלו מהשרת.

Key code in SimpleChatClient

```
@Override
public void start(Stage stage) throws IOException {
   EventBus.getDefault().register(this);
   client = SimpleClient.getClient();
   client.openConnection();
    scene = new Scene(loadFXML("primary"), 640, 480);
    stage.setScene(scene);
    stage.show();
}
```

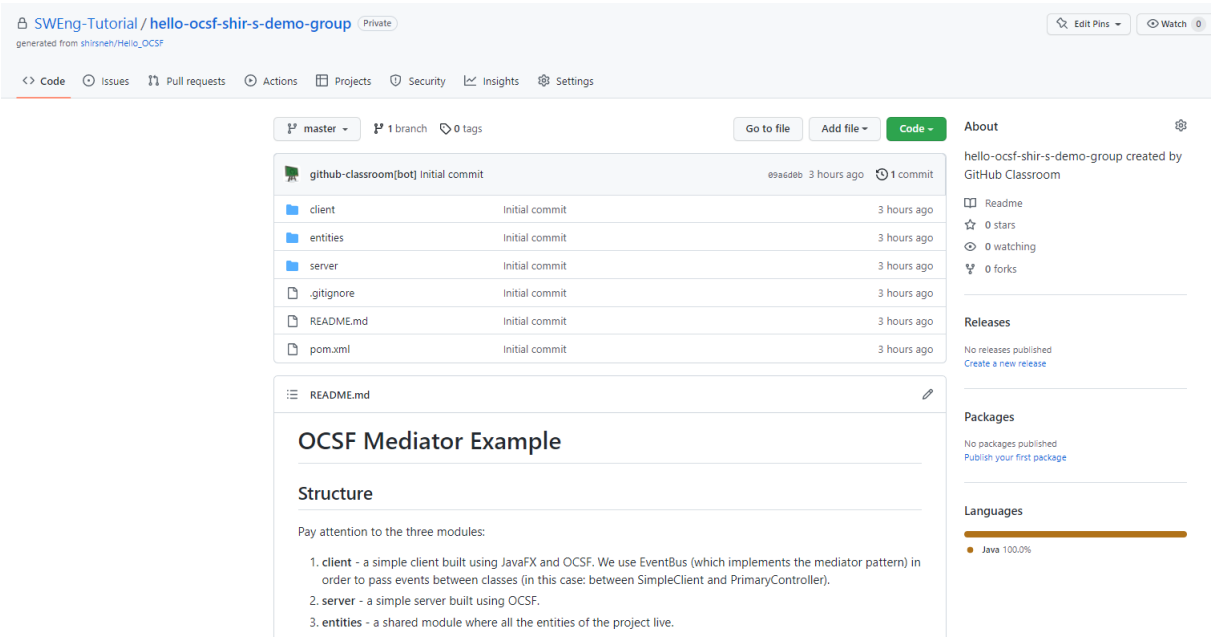
Key code in SimpleClient

```
public void sendToServer(Object msg) throws IOException
{
  if (clientSocket == null || output == null) {
    throw new SocketException("socket does not exist");
  }
  output.reset();
  output.writeObject(msg);
}
```

```
@Override
protected void handleMessageFromServer(Object msg) {
   Message message = (Message) msg;
   if(message.getMessage().equals("update submitters IDs")){
      EventBus.getDefault().post(new UpdateMessageEvent(message));
```

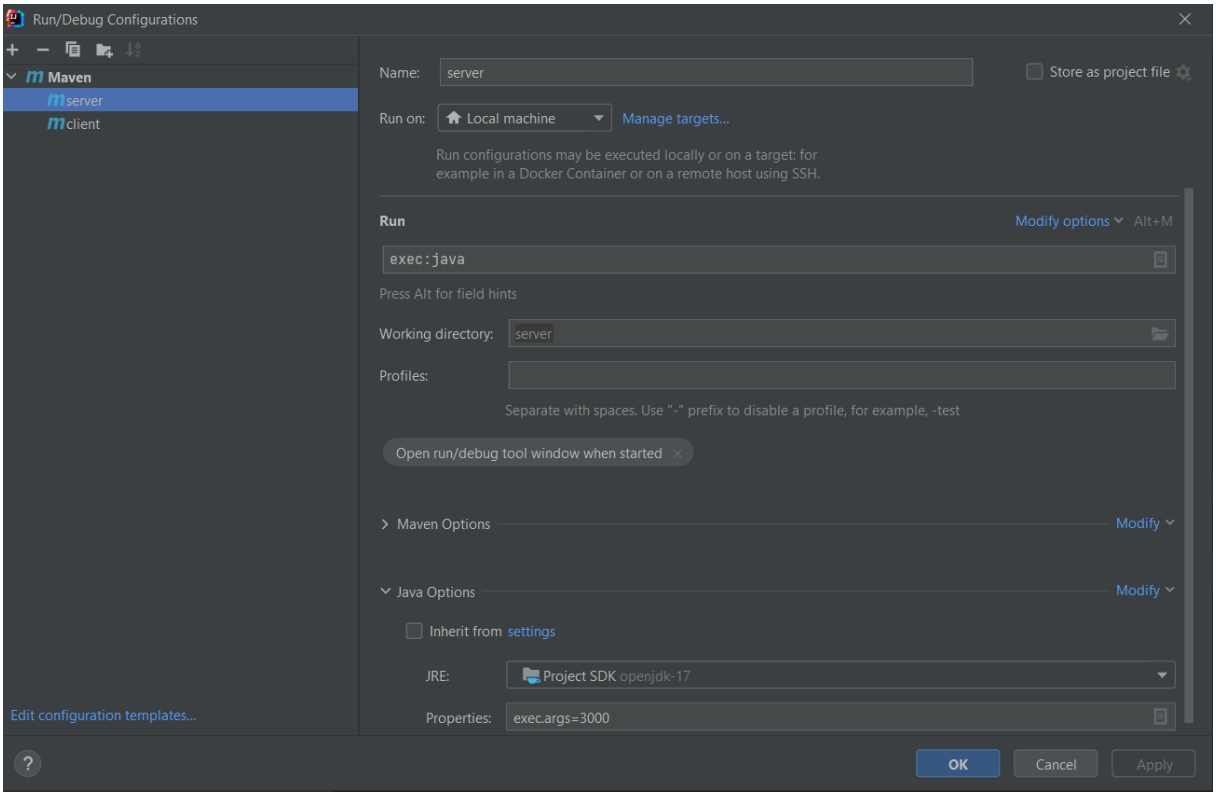
משימות לביצוע

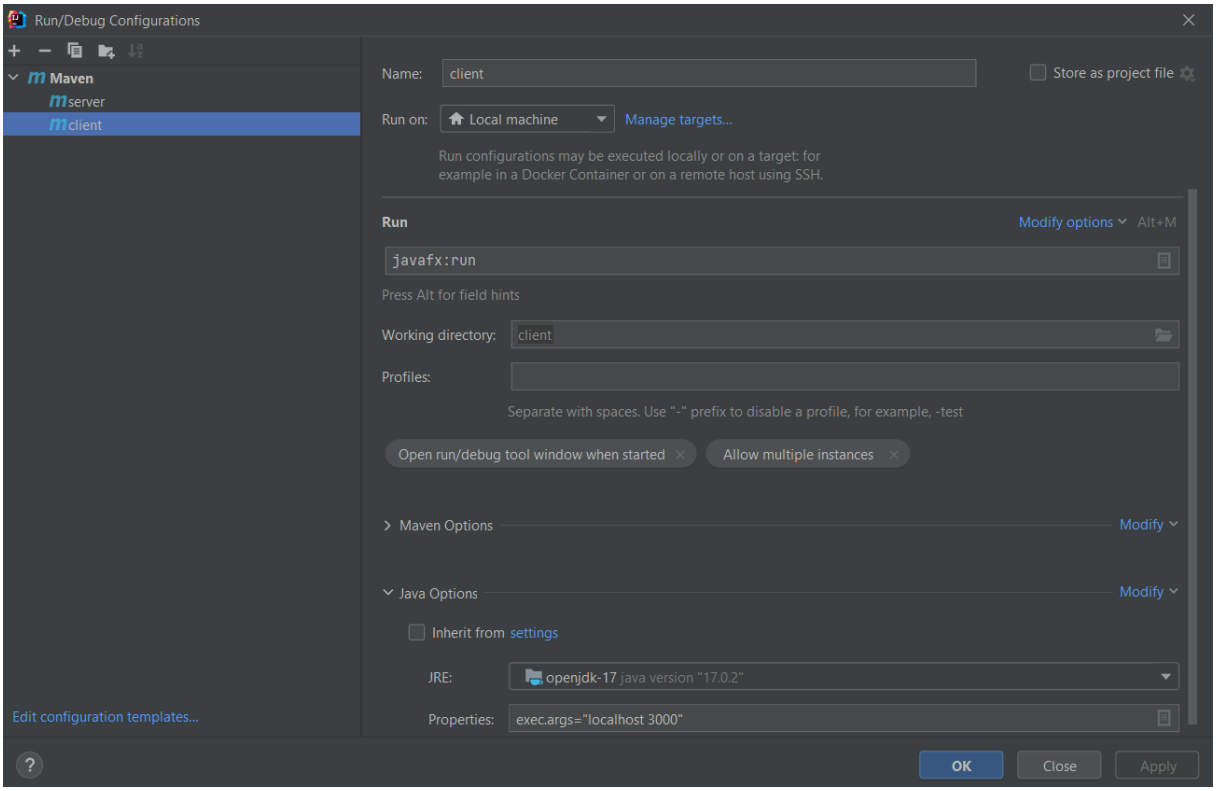
1. היכנסו לקישור הבא: https://classroom.github.com/a/L4Oj_l-z וצרו לעצמכם פרויקט בגיט כמו בלמדנו לעשות במעבדה הקודמת (מעבדה 4 – Git).



2. משכו את הפרויקט ל־IntelliJ.
3. הריצו את השרת ואת הלקוח (יש לכם הוראות כיצד לעשות זאת ב־README בתוך ה־repository).
   a. הריצו את שני קבצי הJar על אותו המחשב.
   b. הריצו את שני קבצי הJar בנפרד על שני מחשבים שונים (אפשר על מכונה וירטואלית).

תמצאו הסבר כיצד להריץ ב־README.


Run/Debug Configurations
Maven
server
client
Name: server
Store as project file
Run on: Local machine
Manage targets...
Run configurations may be executed locally or on a target: for example in a Docker Container or on a remote host using SSH.
Run
Modify options Alt+M
exec:java
Press Alt for field hints
Working directory: server
Profiles:
Separate with spaces. Use "-" prefix to disable a profile, for example, -test
Open run/debug tool window when started
Maven Options
Modify
Java Options
Modify
Inherit from settings
JRE: Project SDK openjdk-17
Properties: exec.args=3000
Edit configuration templates...
OK
Cancel
Apply


Run/Debug Configurations
Maven
server
client
Name: client
Store as project file
Run on: Local machine
Manage targets...
Run configurations may be executed locally or on a target: for example in a Docker Container or on a remote host using SSH.
Run
Modify options Alt+M
javafx:run
Press Alt for field hints
Working directory: client
Profiles:
Separate with spaces. Use "-" prefix to disable a profile, for example, -test
Open run/debug tool window when started
Allow multiple instances
Maven Options
Modify
Java Options
Modify
Inherit from settings
JRE: openjdk-17 java version "17.0.2"
Properties: exec.args="localhost 3000"
Edit configuration templates...
OK
Close
Apply

חשוב!!!

- שימו לב שה-port שאתם מגדירים לשרת וללקוח הוא אותו port (במקרה שלי הגדרתי 3000 אך אתם יכולים להגדיר כל מספר מעל 3000).
- שימו לב להגדיר עבור הלקוח אפשרות הרצה של כמה לקוחות במקביל (בהגדרות קונפיגורציה כנסו ל run->modify options -> allow multiple instances).
- שימו לב שה working directory מוגדר לdirectory של הלקוח/שרת בהתאם לנדרש.

4. שנו את המתודה handleMessageFromClient הנמצאת בSimpleServer בהתאם לרשום בהערות הנמצאות במתודה וכך ש:
   a. בקבלת ההודעה "send Submitters IDs" השרת ישלח חזרה מחרוזת המכילה את תעודות הזהות של המגישים, מופרדים בעזרת פסיקים, כאשר אחרי פסיק יש רווח (כמו במשפט רגיל בשפה). למשל: "123456789, 987654321".
   b. בקבלת ההודעה "send Submitters" השרת ישלח חזרה מחרוזת המכילה את שמות המגישים, מופרדים בעזרת פסיקים, כאשר אחרי פסיק יש רווח (כמו במשפט רגיל בשפה). למשל, אם המגישים הם אליס ובוב נשלח "Alice, Bob" (תוכלו לשלוח רק את השמות הפרטיים או את השמות המלאים, לבחירתכם).
   c. בקבלת ההודעה "what's the time?" השרת ישלח חזרה מחרוזת עם השעה (בפורמט: HH:mm:ss).
   d. בקבלת הודעה מהצורה "multiply n*m" כאשר n ו m הם מספרים שלמים (integers) השרת ישלח חזרה מחרוזת עם תוצאת המכפלה.
   e. בקבלת הודעה כלשהי שאינה תואמת להודעות הנ"ל השרת ישלח חזרה את ההודעה שקיבל לכל המנויים שלו.
      לדוגמה: אם השרת קיבל הודעה: "Good morning friend" הוא ישלח "Good morning friend" לכל החברים שלו (לכל הלקוחות שמנויים אליו).
      (תוכלו להיעזר במתודה sendToAllClients הנמצאת בSimpleServer).

<u>הוראות הגשה לדוח מעבדה 5:</u>

1. ההגשה בזוגות בלבד באמצעות הגשה אלקטרונית. ניתן להגיש מספר פעמים עד לשעת הסיום.
2. יש להגיש קובץ PDF ובו תצלומי מסך של הפלט של הרצת התוכניות (סעיף a3, b3, 4 על כלל סעיפיו) וכן קישור ל־repository שלכם. כמובן שהקוד המעודכן צריך להיות שם, אך אין דרישה לצורת עבודה מסוימת, כמו שהייתה במעבדה מספר 4.
   שם הקובץ הוא מספרי ת"ז של הסטודנטים המגישים, מופרדים בקו תחתון.

ע ב ו ד ה נ ע י מ ה!