מכון טכנולוגי חולון
Holon Institute of Technology

תכנות לסביבת האינטרנט בשפת Java

פרוייקט: שרת משחקים אינטראקטיבי

מרצה: יפית ליבשיץ

סמסטר ב' 2019

מכון טכנולוגי חולון
Holon Institute of Technology

<u>פרטים טכניים</u>

פרויקט זה יכלול 3 אבני דרך שכל אחת מהן תהווה מטלה בפני עצמה.

את הפרוייקט תגישו בשלמותו, כלומר רק לאחר השלמת כל חלקיו.

על מנת לסיים את הפרויקט, שמהווה 80% מהציון הסופי ולהגיע לתוצאה הסופית (מערכת עובדת והצגתה) תצטרכו לבצע את כל המטלות (אבני הדרך) אך כאמור לא להגיש את כולן.

ניתן לבצע את הפרוייקט ביחידים או בזוגות.

ההמלצה שלי היא את לבצע את המטלה הראשונה לבד, על מנת להתנסות קצת בשפה ואת שאר המטלות כולל הגנת הפרויקט (סרטון הסבר שתתבקשו להגיש עם הפרוייקט) – לבצע בזוגות.

כל קטעי הקוד שלכם (מחלקות, ממשקים וכו') צריכים להיות כתובים ומעוצבים ע"פ כל עקרונות התכנות שנלמדו בקורס תכנות מונחה עצמים והקורס הנוכחי (יעילה, נקיה ומתועדת היטב).

בנוסף בחלק מהתרגילים תצטרכו ע"פ דרישה להוסיף קבצי בדיקה (Unit test), שהם בפני עצמם נדבך מאוד חשוב בעולם התוכנה, שנועד לבדוק את הקוד שלכם לפני שהוא עובר לבדיקה חיצונית.

בהצלחה!

מכון טכנולוגי חולון
Holon Institute of Technology

הקדמה

בפרוייקט זה נממש שרת משחקים אינטראקטיבי שרץ על גבי האינטרנט.

הפרוייקט כולו ייכתב בשפת Java ויכיל צד שרת וצד לקוח בעל ממשק גרפי.

השרת יכיל מספר אפשרויות למשחקים ויאפשר לשחק כמה משחקים במקביל, על אותו מחשב או ממספר מחשבים שונים.

אנחנו נממש שני משחקים: "איקס עיגול" ו"תפוס את הארנב".

שני המשחקים מיועדים לשחקן יחיד מול המחשב.

השרת שלנו יהיה בנוי ככה שניתן יהיה להוסיף לו בקלות משחקים נוספים לו היינו רוצים בכך.

הגדרות הפרוייקט נמצאות במסמך זה וב - API מצורף, שעל פיו יש להגדיר ולממש את המחלקות השונות.

יש להיצמד להגדרות המחלקות והממשקים ב API ואין לשנות חתימות של פונקציות.

מכון טכנולוגי חולון
Holon Institute of Technology

חלק א'

בחלק זה אנו נממש את שני המשחקים ואת הלוגיקה של כל אחד מהם.

כללי המשחק איקס עיגול

- המשחק מתקיים על גבי לוח בגודל 3 על 3, עבור 2 שחקנים.
- בתחילת המשחק לכל שחקן נקבע סימן – איקס או עיגול.
- מהלך המשחק: כל שחקן בתורו בוחר משבצת פנויה על הלוח (כלומר, שלא מכילה סימן איקס או עיגול) ומציב עליה את הסימן שלו.
- מנצח במשחק השחקן שהצליח ליצור ראשון רצף של שלושה סימנים (איקס או עיגול) הנמצאים על שורה אחת טור אחד או אלכסון אחד.
- אם הלוח התמלא ולא נוצר אף רצף – המשחק מסתיים בתיקו.

כללי המשחק תפוס את הארנב

- המשחק מתקיים על גבי לוח בגודל 9 על 9.
- ישנם 2 שחקנים: ילד וארנב
- כל אחת מהדמויות נמצאת במיקום מסויים על הלוח.
- בכל תור אחת מהדמויות זזה משבצת אחת.
- המשחק מסתיים כאשר הילד תופס את הארנב, כלומר זז למשבצת שהארנב נמצא בה או כאשר הילד עשה X צעדים (X הוא מספר המועבר באיתחול המשחק) ואז הארנב מנצח.
  כל אחת מהדמויות יכולה לזוז משבצת אחת בארבע כיוונים אפשריים: למעלה, למטה, ימינה או שמאלה.
- הכיוון הנבחר שבו יזוז הילד יקבע על ידי המשתמש
- הכיוון הנבחר שבו יזוז הארנב ייקבע על ידי המחשב

כללים למימוש המשותפים לשני המשחקים

- לבחירת המהלך של המחשב יהיו שני מצבים: בחירה מושכלת ובחירה אקראית.
- סוג הבחירה תיקבע בעת איתחול אובייקט של המשחק
- בשני המשחקים, כאשר בחירת מהלך המחשב תהיה אקראית, המחשב יגריל צעד אקראי (חוקי כמובן) והוא יהיה הצעד הנבחר
- כאשר בחירת מהלך המחשב תהיה מושכלת, על המחשב לחשב איזו בחירה תגדיל את סיכויו לזכות במשחק
  במימוש סעיף זה הינכם מתבקשים להגדיר את אלגוריתמי הבחירה בעצמכם
  על מנת לממש את האלגוריתמים ניתן להיעזר באחד או יותר ממבני הנתונים הקיימים ב Java אותם למדנו בכיתה (collections).

כמו שצויין בהקדמה, אנחנו מעוניינים לממש את השרת כך שניתן יהיה להוסיף בקלות רבה משחקים נוספים.

על מנת לקיים את הדרישה הזאת, נשתמש בעיקרון ה **Strategy Pattern** אותו למדנו בכיתה:

נגדיר ממשק שדרכו יהיה ניתן להפעיל את המשחקים.

מחלקה שתנהל משחק תחזיק רפרנס לממשק זה ותשתמש בו, אך לא תכיר את המימושים השונים שלו (המחלקות הממשות, אשר נקראות Concrete Classes).

המחלקות שיממשו את שני המשחקים יממשו זאת על ידי מימוש הממשק (כלומר יכילו פונקציונליות דומה, עם אופן שימוש זהה).

כמו כן, דבר זה יאפשר לנו בקלות להחזיק משחק אחד על פי בחירה ולהחליף בעת הצורך.

הממשק אותו נגדיר ייקרא `IGameAlgo` והוא יכיל את המתודות הבאות, אשר יאפשרו לנהל משחק מתחילתו ועד סופו:

- המתודה הראשונה אותה הממשק מגדיר היא **getGameState**, שבעזרתה הקורא יוכל לדעת את מצב המשחק, להציג אותו בעת הצורך או להחליט על המשך או סיום המשחק. סיום משחק קורה כאשר מתרחש אחד מהמצבים הבאים: ניצחון / הפסד / תיקו.
- מתודה נוספת שתוגדר היא **updatePlayerMove**, אשר תעדכן צעד אחד של השחקן במשחק, בתנאי שהוא חוקי.
- והמתודה האחרונה **calcComputerMove** תחליט מה הצעד שהמחשב יעשה במשחק כאשר מגיע תורו. במחלקות הממשות את הממשק הזה, המתודה הזאת תגדיר בעצם את האלגוריתמים שהזכרנו למעלה. עבור כל משחק, יהיה מימוש שיחליט על צעד באופן רנדומלי, תוך שמירה על כללי המשחק, ומימוש אחד שיחליט על צעד באופן מושכל, כזה שיגדיל את סיכוי המחשב לנצח במשחק.

את הממשק הנ"ל תממש מחלקה אבסטרקטית בשם `GameBoard`. תפקידה לשמש מכנה משותף בין כל המשחקים הקיימים בתוכנה.

עבור כל משחק, נגדיר מחלקה אבסטרקטית: `TicTacTow` ו `CatchTheBunny`. מחלקות אלו יממשו את המשחקים איקס עיגול ותפוס את הארנב בהתאמה.

הן יממשו את כל המתודות המוגדרות בממשק `IGameAlgo` פרט למתודה **calcComputerMove**.

לכל אחת מהמחלקות `TicTacTow` ו `CatchTheBunny` נגדיר שתי מחלקות שיורשות מהן, אשר תפקיד כל אחת מהן לממש את המתודה **calcComputerMove** על פי האלגוריתם הנבחר.

המחלקות `TicTacTowGameRandom` ו – `CatchTheBunnyRandom` יממשו בחירה רנדומלית במשחקים איקס עיגול ותפוס את הארנב, בהתאמה, כמפורט למעלה.

והמחלקות `TicTacTowGameSmart` ו – `CatchTheBunnySmart` יממשו בחירת מושכלת במשחקים איקס עיגול ותפוס את הארנב, בהתאמה, כמפורט למעלה.

שימו לב, במחלקות אלו, מכיוון שהגדרת האלגוריתם היא חלק מהמשימה מומלץ מאוד **לתעד** את הקוד ככה שמי שקורא אותו יוכל להבין את כוונתכם.

שתי המחלקות האחרונות אותן נממש בחלק זה הן מחלקות בדיקות.

כפי שלמדנו בכיתה, יש ערך רב לבדיקות יחידה (unit testing) תוך כדי תהליך פיתוח, עוד לפני שאנחנו מתממשקים עם חלק אחר בתוכנה או מוסרים את התוכנה הלאה.

נממש שתי מחלקות בעזרת Junit framework: `TicTacTowTest` ו `CatchTheBunnyTest`

כל מחלקה תבדוק את שני האלגוריתמים של המשחק שלה ותכיל מתודת test@ עבור כל אחד מאלגוריתמי המשחק של המחשב.

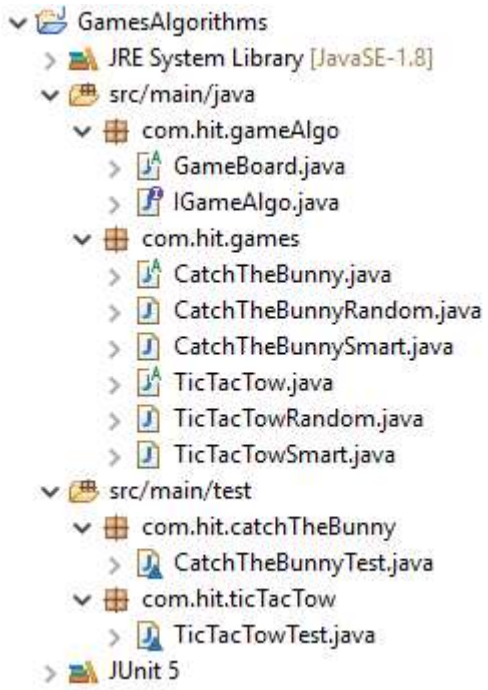
דמיינו שהמוצר הזה צריך להגיע ולרצות את הלקוח (במקרה זה הבודק)

לכן ככל שתבדקו יותר תגיעו לרמה טובה יותר ומוצר טוב יותר.

מכון טכנולוגי חולון
Holon Institute of Technology

לבסוף ארזו את כל 10 המחלקות המתוארות ב – packages **במבנה (שימו לב היטב למבנה**

התיקיות) ובשמות הבאים:



מכון טכנולוגי חולון
Holon Institute of Technology

חלק ב'

בחלק זה נפתח את צד השרת של התוכנה שלנו.

ארכיטקטורת Client/Server היא אחת מתצורות ההתקשרות הנפוצות ביותר ברשתות מחשבים.

תצורה זו פשוטה ומבוססת בעיקרה על רעיון שבו קיימים שני צדדים בארכיטקטורה ולכל צד תפקיד משלו:

צד שרת - תפקידו לספק **שירותים** כלשהם עליהם הוא מצהיר שהוא יודע לספק.

צד לקוח – תפקידו **להפעיל** את אותם שירותים **לקבל את המידע** ולעשות איתו את **שהוגדר לו**.

בחלק זה של הפרויקט אנו נבנה שרת שיעבוד ע"ג פרוטוקול TCP כפי שנלמד בכיתה ונשלב את היכולת לנהל משחקים כחלק משרת זה.

מכון טכנולוגי חולון
Holon Institute of Technology

המחלקות שמימשתם בחלק הקודם ישמשו אתכם כתשתית על מנת להפעיל, לנהל ולהריץ משחקים.

נשתמש במחלקות מחלק א' כספרייה. כלומר, הקוד שלה אינו חלק מהפרוייקט הנוכחי, אלא יהיה כלול בו כמו שמשתמשים בספריות מוכרות ב java.

על מנת לייצר ספרייה ב Java בצעו את השלבים הבאים:

ארזו את הפרויקט שלכם **GamesAlgorithms** הכולל את מחלקות האלגוריתמים שלכם וה – Test (כלומר, כל הקוד שכתבתם בחלק הקודם) כ – jar נפרד ע"פ המדריך הבא (**שימו לב לארוז גם את ה – sources בסעיף החמישי**, כלומר, שיהיה ניתן לקרוא את הקוד):

1. Create jar file from eclipse

Creating a New JAR File

To create a new JAR file in the workbench:

1. In the Package Explorer, you can optionally pre-select one or more Java elements to export. (These will be automatically selected in the JAR Package Specification wizard page, described in Step 4.)
2. Either from the context menu or from the menu bar's **File** menu, select **Export**.
3. Expand the **Java** node and select **JAR file**. Click **Next**.
4. In the **JAR File Specification** page, select the resources that you want to export in the **Select the resources to export** field.
5. Select the appropriate checkbox to specify whether you want to **Export generated class files and resources** or **Export Java source files and resources**. **Note**: Selected resources are exported in both cases.
6. In the **Select the export destination** field, either type or click **Browse** to select a location for the JAR file.
7. Select or clear the **Compress the contents of the JAR file** checkbox.
8. Select or clear the **Overwrite existing files without warning** checkbox. If you clear this checkbox, then you will be prompted to confirm the replacement of each file that will be overwritten.
9. **Note**: The overwrite option is applied when writing the JAR file, the JAR description, and the manifest file.
10. You have two options:
    - Click **Finish** to create the JAR file immediately.
    - Click **Next** to use the JAR Packaging Options page to set advanced options, create a JAR description, or change the default manifest.

צרו פרויקט חדש בשם – **GameServerProject** ולאחר מכן עשו import ל –**GamesAlgorithms.jar** לפרויקט זה ע"פ המדריך בקישור הבא:

2. Adding Internal JARs (Method 1)

   https://m.wikihow.com/Add-JARs-to-Project-Build-Paths-in-Eclipse-(Java)

מכון טכנולוגי חולון
Holon Institute of Technology

אחרי שיצרנו פרוייקט חדש וכללנו את התשתית זה הזמן להתחיל לממש את רכיבי השרת.

המחלקה הראשונה שנממש בחלק זה של הפרויקט היא ה – `BoardGameHandler`.

חלק חשוב ובלתי נפרד בפיתוח תוכנה הוא ה – design (עיצוב מבנה התוכנה). לאחר שהדרישות מהלקוח הובנו ולפני שמתחילים לכתוב קוד, חייבים לעבור לשלב בו מתכננים את מבנה המערכת (System architecture).

בחלק זה של הפרויקט, נכתוב את יחידות המערכת ונעמוד על הקשר בניהן. בנוסף אנו נעצב את המערכת תוך שימוש בעקרונות בסיסיים ב- OOP **ושימוש בפתרונות סטנדרטיים** לבעיות מוכרות מעולם התוכנה שהם ה – design patterns.

העיקרון הראשון והבסיסי אליו נצמד הוא היכולת לתת גמישות למערכת מבחינת הוספת יכולות ומימושים נוספים בצורה פשוטה וקלה אך בד בבד לא לאפשר שינויים של ה – Application Programming Interface

עקרון חשוב זה נקרא:

Open/Close principal - open for extension, but closed for modification

ה – BoardGameHandler יכיל בתוכו (כ – member) רכיב אחד **בלבד**: IGameAlgo. מחלקה זו צריכה לדעת לפנות לרכיב ה - IGameAlgo במטרה לנהל מהלך משחק. כלומר:

כל עוד המשחק ממשיך (אין מנצח או תיקו):

- לעדכן את הצעד הנבחר מהשחקן
- לבדוק האם המשחק ממשיך או הסתיים (ניצחון או תיקו)
- אם המשחק ממשיך, לחשב את צעד המחשב ולעדכן אותו על הלוח
- לבדוק האם ממשיך או הסתיים (ניצחון או תיקו)

ה - BoardGameHandler עושה שימוש ב – IGameAlgo שיועבר לו כפרמטר **ב – Constructor** וישמר אצלו כ – member.

השימוש של ה – BoardGameHandler ב- IGameAlgo הוא שימוש ב – API בלבד (כלומר, קריאה לפונקציות של ה IGameAlgo), ללא הבנה כיצד מומשו וללא אפשרות לשנות אותו (closed for modification).

אותן מחלקות ש"יוצקות" מימוש לאותו API מועברות ל – BoardGameHandler מבחוץ ולכן ניתן בקלות להעביר סוגים שונים של מימושים, להוסיף בכל שלב מימושים חדשים ובאותה צורה להעביר גם אותם (open for extension). בכך השלמנו את ה – **strategy pattern** ושמרנו על עקרון ה – open/close principal.

המחלקה השנייה אותה נכתוב בחלק זה תקרא **CLI**:

CLI - A command-line interface or command language interpreter (CLI), also known as command-line user interface is a means of interacting with a computer program where the user (or client) issues commands to the program in the form of successive lines of text (command lines).

מחלקה זו תהיה אחראית לממשק מול הלקוח על מנת להפעיל את השרת ולהפסיק את פעולתו במידת הצורך.

ה – CLI יתמוך בפקודות בסיסיות להפעלת השרת ולהעברת הקונפיגורציה שנדרשת לריצת השרת.

חשבו שתמיד אפשר להרחיב את הפונקציונליות באמצעות ממשק זה. הפקודות שיש לתמוך בהן:

<GAME_SERVER_CONFIG {capacity}> - פקודה זו תאפשר להעביר את מספר המשחקים שהשרת מאפשר לשחק בו זמנית (במקביל). **שימו לב ללא פקודה זו עליכם לספק default configuration**.

<START> - פקודה זו תפעיל את השרת בכדי להיות זמין לבקשות מלקוחות.

<SHUTDWON> - פקודה זו תפסיק את השרת כך שתאפשר לבקשות שהתקבלו לסיים בצורה תקינה.

פעולת ה – CLI תתבצע כ – Thread נפרד לכן עליה לממש את הממשק Runnable.

בנוסף ה – CLI תהיה חלק מ – Observer Pattern אשר ימומש באמצעות PropertyChangeListener.

בכל שלב שתהיה לה פקודה מוכנה **היא תעדכן את רשימת ה – Observers שמקושרים אליה** (באמצעות אובייקט של PropertyChangeSupport)

<u>שימו לב,</u> בכל שלב אתם צריכים לטפל רק בפקודות שמוכרות ל – CLI.

כל פקודה אחרת שמתקבלת יש להודיע ללקוח שהפקודה אינה תקינה ולבקש שוב קלט תקין.

דוגמאות לתקשורת CLI מול הלקוח:

דוגמא 1:

```
> Please enter your command
> GAME_SERVER_CONFIG
>
```

מכון טכנולוגי חולון
Holon Institute of Technology

דוגמא 2:

```
> Please enter your command
> START
> Starting server.....
```

דוגמא 3:

```
> Please enter your command
> sfeesw
> Not a valid command
```

דוגמא 4:

```
> Please enter your command
> SHUTDWON
> Shutdown server
```

<u>המחלקה השלישית</u> אותה נממש תקרא **Server**. היא תחזיק אובייקט מסוג **ServerSocket שיאזין ל – port שיתקבל ב – constructor של המחלקה,** ותנהל את התקשורת עם הלקוחות.

מחלקה זו (בדומה למה שלמדנו בכיתה) תכיל מתודה שחתימתה היא: `public void run()`

מכון טכנולוגי חולון
Holon Institute of Technology

במתודה זו עליכם לאתחל את כל הרכיבים שרלוונטיים לשרת שלכם וכמו כן לדאוג להאזין לבקשות מהלקוחות ולדעת לנתב אותם כ – **thread נפרד** למחלקות שאחראיות לטיפול בבקשה.

בנוסף מחלקה זו תממש שני ממשקים על מנת לבצע את פעולת השרת:

PropertyChangeListener – בכדי לקבל פקודות רלוונטיות מה – CLI.

Runnable – על מנת לרוץ ב – Thread נפרד בעת ההפעלה.

המחלקה הרביעית אותה נממש תיקרא **UnknownIdException**. מחלקה זו תירש מהמחלקה Exception ותייצג exception אשר מתאר אירוע של ניסיון לגשת למשחק שלא קיים.

המחלקה החמישית אותה נממש היא ה **GamesService**. היא תוחזק ב - **GameServerController** (כמפורט להלן) ותהיה אחראית לעבוד אך ורק עם בקשות שיגיעו מהלקוח. לכן לא צריכים להיות מוכלים בה אלמנטים של תקשורת, כגון Socket, inputStream/outstream כלל.

מחלקה זו צריכה לתמוך במספר משתנה של משחקים שהשרת מאפשר לשחק בו זמנית. על מנת לעשות זאת היא תשמור עבור כל משחק שכרגע מתקיים אובייקט של **BoardGameHandler** ומספר מזהה ייחודי (ניתן להשתמש בערך מספרי פשוט)

על מנת לשמור את המשחקים והמספרים המזהים יש להשתמש באחד ממבני הנתונים המוכרים ב Java עליהם למדנו בכיתה.

אם יועבר למחלקה מזהה של משחק אשר אינו קיים, היא תזרוק exception מטיפוס **UnknownIdException**.

המחלקה השישית אותה נממש נקראת **GameServerController**. מטרתה ליצור שכבת הפרדה בין ה - **GamesService** לשכבת ה – Networking, ממשו את ה – API שנתון ע"י קריאה למטודות הרלוונטיות ב - **GamesService**.

המחלקה השביעית שנממש היא ה – **HandleRequest**. תפקידה של מחלקה זו הוא לקבל

בקשות מלקוח מסויים, עבור משחק מתחילתו ועד סופו

הבקשות מתקבלות מה – Server (מה socket ששייך ללקוח הרלוונטי). עבור כל בקשה, על המחלקה הנ"ל לקרוא את המידע (כמפורט להלן) המגיע במבנה JSON בעזרת ספרייה (קובץ JAR) לניתוח הודעות JSON, כפי שלמדנו בכיתה (JSON Simple) ולפי סוג ההודעה להפעיל את המתודה הרלוונטית ב – **GameServerController**.

משחק מסתיים כאשר מתקבלת הודעה מסוג Stop-Game או כאשר השרת שולח תגובה שמכילה GameState של סיום משחק.

מכון טכנולוגי חולון
Holon Institute of Technology

הודעות JSON שהשרת צריך לתמוך בקבלתן

1. משחק חדש

```
{"type":"New-Game",
"game": "Catch The Bunny", <could also be "Tic Tac Toe">
"opponent": "Random"} <could also be "Smart">
```

עבור הודעה זו השרת ישלח כתגובה הודעה בפורמט הבא:

```
{"type":"New-Game",
"ID": 8, <any ID number that is greater than 0, or -1 in case a game cannot be opened>
"board":["-","K","-", …. , "B"]} <K = kid position, B = Bunny position>
```

2. עדכן צעד

```
{"type":"Update-Move",
"ID": 4, <the ID that was passed by the server as a response to the message New-Game>
"row": 2,
"col":1}
```

עבור הודעה זו השרת ישלח כתגובה הודעה בפורמט הבא:

```
{"type":"Update-Move",
"ID":4,
"state":4,  <values if enum GameState [0-4]>
"board":["x","o","-","x","o","-","x","o","-"]}
```

במקרה של צעד לא חוקי:

```
{"type":"Update-Move",
"ID":4,
"state":0, <the corresponding value of GameState.ILLEGAL_PLAYER_MOVE >
"board":null}
```

3. כאשר המחשב מתחיל את המשחק ראשון:

{"type":"Start-Game",

"ID":3}

עבור הודעה זו השרת ישלח כתגובה הודעה בפורמט הבא:

{"type":"Start-Game",
"ID":3,
"board":["-","K","-", …. , "B"]}

4. הפסק משחק (כאשר החלון נסגר או השחקן רוצה לסיים את המשחק באמצע)

{"type":"Stop-Game",
"ID":3}

המחלקה השמינית והאחרונה בצד השרת תהיה ה – **GameServerDriver** וזה הקוד היחיד שאמור להיות מוכל בה:

```
public class GameServerDriver {

	public static void main(String[] args) {
		CLI cli = new CLI(System.in, System.out);
		Server server = new Server(34567);
		cli.addPropertyChangeListener(server);
		new Thread(cli).start();

	}
}
```

מחלקה זו היא למעשה החלק שמחבר לנו את כל **רכיבי המערכת וגם מפעיל אותם**.

מכון טכנולוגי חולון
Holon Institute of Technology

מספר דגשים חשובים לחלק זה:

- היצמדו ל – API המצורף לתרגיל, העתיקו את החתימות של המתודות למחלקות שתבנו.
  השלימו את ה – design של המערכת ורק לאחר מכן התחילו לממש.
- השתמשו ב – Decorator מתאימים על מנת לעטוף את input/output streams של הבקשה שמגיעה מהלקוחות.
  אני השתמשתי ב – Decorator אלו:

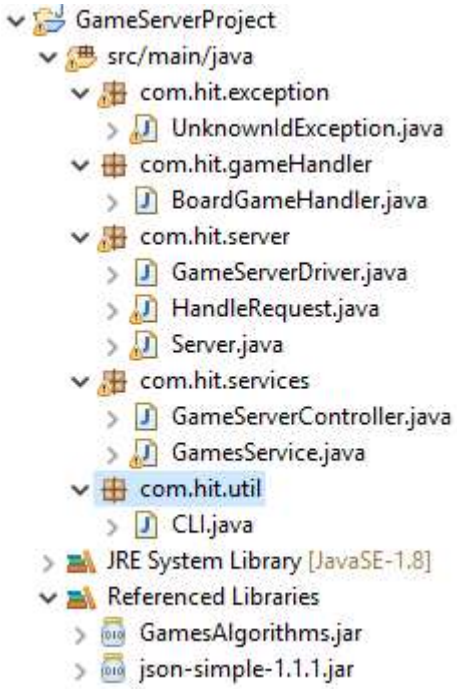
```
Scanner reader = new Scanner(new
InputStreamReader(socket.getInputStream()));
PrintWriter writer = new PrintWriter(new
OutputStreamWriter(socket.getOutputStream());
```

- המערכת שלכם כעת היא מערכת multi threaded לכן **דאגו לסנכרן את המערכת היכן שצריך !!!**

מכון טכנולוגי חולון
Holon Institute of Technology

לבסוף ארזו את כל 8 המחלקות **החדשות** המתוארות ב – packages במבנה ובשמות הבאים:

* שימו לב לצירוף של ה – json-simple.jar:



מכון טכנולוגי חולון
Holon Institute of Technology

חלק ג'

בחלק זה נפתח את צד הלקוח.

כפי שתואר לעיל, תפקידו של צד הלקוח **להפעיל** את השירותים או **לקבל את המידע** שהשרת מספק ולעשות איתו את **שהוגדר לו**.

על מנת לבצע את המשימה בצורה טובה ברוב המוחלט של המקרים צד הלקוח מכיל ממשק גרפי.

רקע (ויקיפדיה) - את המונח "חוויית המשתמש" טבע לראשונה דונלד נורמן (Donald Norman) באמצע שנות ה-90. בעבר התייחסו בעיקר לממשק משתמש UI, הממשק בין הטכנולוגיה לאדם לצורך השגת מטרה כלשהי, אך בשנים אלו חילחלה ההבנה כי ממשק המשתמש מהווה רק אספקט מסוים מתוך עולם חוויית המשתמש המתייחס גם לעיצוב, ארכיטקטורת המידע, אסטרטגיה, שיווק וטכנולוגיה וגם להיבטים רגשיים, אנושיים ותחושתיים. חוויית משתמש טובה יכולה לחסוך מאמצי הטמעה והשקעה בפעילות ניהול ידע לעידוד השימוש וניהול השינוי. בשנים האחרונות תחום חוויית המשתמש הפך למשמעותי בכל פתרון מחשובי בכלל, ופתרונות ניהול ידע בפרט, ונתפס כמרכיב קריטי והכרחי (Critical Success Factor) .

אחד האנשים שתרם רבות להכרת והבנת המושג "חוויית משתמש" ומשמעותו הוא סטיב ג'ובס, אשר התבסס על ממשק משתמש גרפי בעת פיתוח המחשב האישי "מקינטוש."

את הממשק הגרפי במערכת שלנו אתם נדרשים לבנות ע"פ מודל ה – MVC, כפי שנלמד בכיתה.
מודל זה הוא Design pattern בו משתמשים במערכות רבות המבוססות UI.
למודל ה – MVC יתרונות רבים אך העיקרון ה – OOP החשוב ביותר במודל זה הוא עקרון ה - Loosely coupled .
עקרון זה ביסודו מניח כי כל רכיב במערכת או שכבה (שיכולה להיות מיוצגת ע"י מספר רכיבים) צריך להיות בלתי תלוי ברכיב אחר במערכת.
ארכיטקטורת מערכת שבנויה ע"פ עקרון זה צריכה לאפשר גמישות מירבית של **שינוי או החלפה** של כל אחת מהשכבות/רכיבים ללא השפעה כלל או השפעה מינימלית על השכבות האחרות במודל ה - MVC.

בנוסף כל שכבה מבודדת את הלוגיקה שלה ולכן במקרה שקיים באג במערכת השכבה שבה נמצא הבאג לא משפיע על השכבות האחרות, מה שמסייע באיתור וטיפול בבאגים במערכות מורכבות.

במודל ה – MVC קיימות 3 שכבות עיקריות **M**odel, **V**iew, **C**ontroller ולכל שכבה תפקיד משלה.

ה – Model הוא החלק שאחראי להכיל את ה – data לכן כל פעולה שהמערכת תבצע, נגזרת מהמידע או חלק מהמידע שנמצא ב – Model (ברוב המקרים בעולם האמיתי ישמר/ו המודל/ים במסדי הנתונים ה – DB ויקראו משם).

ה – View הוא החלק שאחראי על ממשק המשתמש ואינטרקציה עם המשתמש. חלק זה משקף בעיקר את הנתונים ששמורים ב- Model, לעיתים גם משנה ה - View את המודל מפעולות שיוזם המשתמש.

ה – Controller הוא החלק שמחבר את שני החלקים הנ"ל. ה – Model וה – View לעולם יהיו חלקים מופרדים ללא אפשרות "לדבר" אחד עם השני. בכל זאת, ללא העברת המידע בין ה – Model ל – View אין משמעות למודל ה – MVC ולכן תפקידו של ה - Controller הוא להעביר את המידע ובנוסף לבצע את לוגיקת החיבור בין שני החלקים במידה וקיימת.

אנו נייצג כל שכבה באמצעות interface ומחלקה אחת לפחות **(אתם רשאים להוסיף מחלקות נוספות)**:

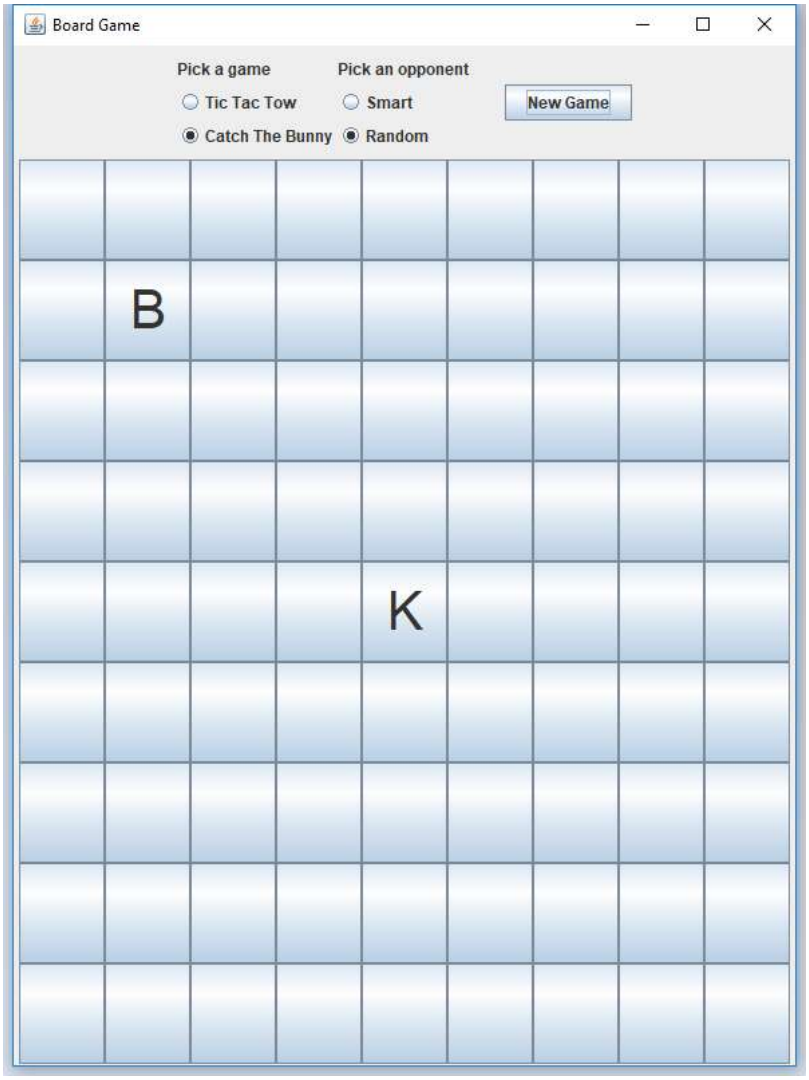
- **השכבה הראשונה ה –** Model תכיל לפחות שתי מחלקות:
  GamesModel, GamesClient וה - interface **Model**.
  תפקידה של המחלקה **GamesModel** הוא להוות את הממשק אל מול השרת שבנינו בחלק השני, להעביר את ההודעות המתאימות על פי מה שהוגדר בחלק השני, לקבל את התגובות, אם ישנן ולעדכן את הממשק הגרפי.
  הנכם נדרשים לתמוך בהודעות "New-Game" ו "Update-Move" (עמוד 14) **בלבד**.
  (כלומר, לייצר את ההודעות הנ"ל בפורמט שהשרת מצפה לו, בהתאם לפקודות המגיעות מהמשתמש).
  בכדי לתקשר עם השרת עליכם ליצור מחלקה שנקראת **GamesClient** שתהיה אחראית על התחברות לשרת עם port שניתן לה (כתובת ה IP שנתחבר אליה בפרוייקט היא local host), על שליחת הודעות וקבלת תגובות.
- **השכבה השנייה ה –** Controller תיוצג ע"י ה - **Controller** interface והמחלקה **GamesController** שתפקידה לקבל ב – constructor את ה – Model וה – View ולבצע את החיבור בניהם, כפי שלמדנו בכיתה. ה – **Controller** מממש את הממשק **PropertyChangeListener** ולכן מאפשר לרשום אותו כמאזין גם ב - Model וגם ב – View.

מכון טכנולוגי חולון
Holon Institute of Technology

- **השכבה השלישית ה** – View תיוצג ע"י ה – **View** interface והמחלקה **GamesView**.

ה - **GamesView** תכיל את הממשק הגרפי של המערכת ותיצור אותו בעת הפעלת המערכת, בכל שלב שיש מידע או פעולה שרלוונטיים לשרת, יש להעביר אותם דרך ה – Controller ל – Model.

הממשק הגרפי יכלול לפחות את הרכיבים הבאים:



כפי שניתן לראות קיימים במממשק הגרפי שני חלקים:

- החלק הראשון מכיל כפתורים לבחירת משחק חדש. תחילה על המשתמש לבחור:
  - באיזה סוג משחק הוא מעוניין לשחק, מתוך האפשרויות הקיימות
  - מול איזה יריב הוא מעוניין לשחק (מחשב מסוג smart או random)

ולאחר מכן, כאשר לוחץ על כפתור ה new game, ייווצר משחק חדש על פי בקשתו (על ידי פניה לשרת כמובן)

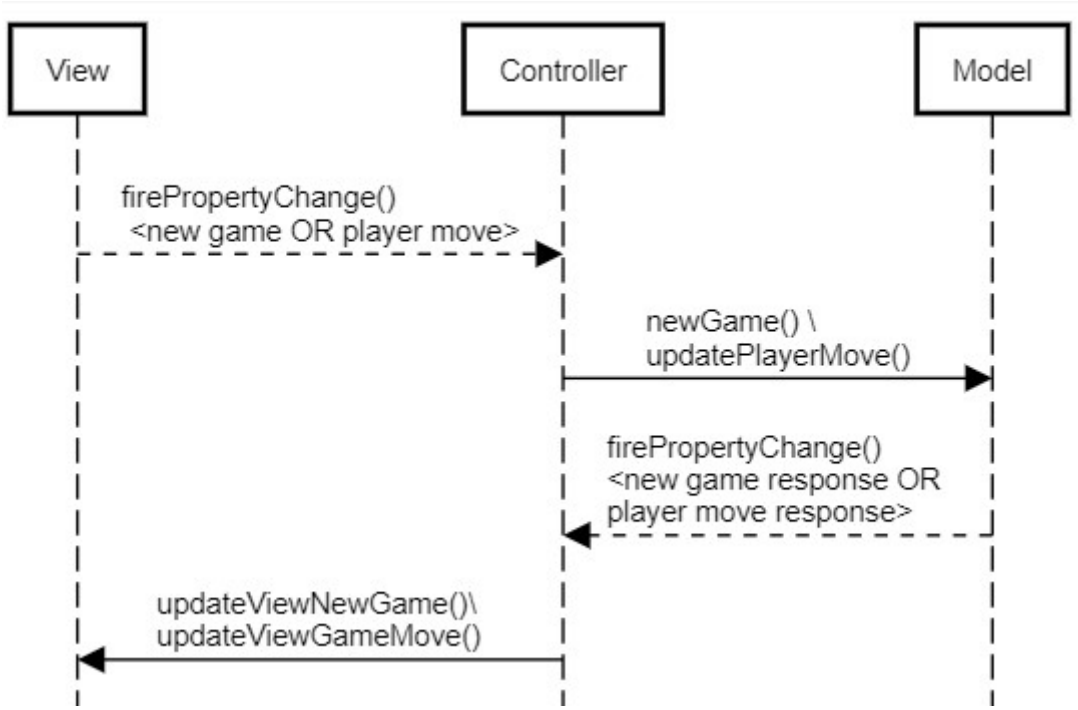
מכון טכנולוגי חולון
Holon Institute of Technology

- החלק השני מכיל את לוח המשחק ויאותחל לאחר בחירת המשחק.

אני מימשתי את לוח המשחק על ידי אובייקט מטיפוס **`JButton`** לכל משבצת על הלוח. ואת האפשרויות לבחירת משחק חדש על ידי אובייקטים מטיפוס **`JRadioButton`**

הנכם רשאים להשתמש באובייקטים אלו או אובייקטים גרפיים אחרים לבחירתכם.

להלן תיאור ה - flow של צד הלקוח:



המחלקה האחרונה בצד הלקוח תהיה ה – **GamesClientDriver** וזה הקוד היחיד שאמור להיות מוכל בה:

```
public static void main(String[] args){

      Model model = new GamesModel();
      View view = new GamesView();
      Controller controller = new GamesController(model, view);
      ((GamesModel)model).addPropertyChangeListener(controller);
      ((GamesView)view).addPropertyChangeListener(controller);
      view.start();
}
```

מחלקה זו למעשה יוצרת לנו את המערכת בארכיטקטורת MVC, מחברת את הרכיבים השונים וגם מפעילה אותם.

בונוס 1 (עד 5 נקודות):

תמיכה בהודעות הנוספות שהוגדרו בחלק השני ("start-game", "end-game").

התמיכה מתבטאת גם ב - model וגם ב - view, כלומר, יש להוסיף ממשק גרפי מתאים ויכולת לשלוח הודעות בפורמט המתאים ולעבד את תגובות השרת, אם ישנן.

בונוס 2 (עד 5 נקודות):

- כיום כפי שהוסבר, ממשק המשתמש הוא חלק חשוב מאוד בכל מוצר תוכנה ועשוי לעיתים להכריע בבחירת הלקוח ולכן כל תוספת בעיצוב הממשק שתעזור להבין את פעולות המערכת בצורה טובה יותר תזכה בנקודות.
- סידור הרכיבים והעיצוב של ה – UI איננו חייב להיות ע"פ מה שהוצג. אתם רשאים לעצב כרצונכם.

מספר דגשים חשובים לחלק זה:

- הנכם רשאים להוסיף מתודות נוספות מעבר למוגדר ב API המצורף על מנת לתקשר עם מחלקות נוספות שאתם בוחרים להוסיף.
- מתודות private\ protected הנכם רשאים להוסיף בכל מחלקה בפרוייקט.

מכון טכנולוגי חולון
Holon Institute of Technology

ארגו את כל המחלקות החדשות (לפחות 8) המתוארות ב – packages במבנה ובשמות הבאים: