תרגיל 5

תכנות מונחה עצמים

קבצי הבדיקה:

- לשאלה 1: IteratorTest.java
- לשאלות 2+3 אין בדיקה אוטומטית, ולכן יבדקו ידנית.
- לשאלה 4: MinesTester.java בודק את החלק שהוא לא javaFX, החלק הגרפי ייבדק ידנית.

בשביל תרגילים 2+3+4, כדאי מאוד להסתכל על מסמך השימוש וההתקנה של JavaFX, ובמיוחד בגלל Scene Builder. יש שם פרטים קטנים וחשובים על העבודה עמו.

שאלה 1 (package iterator)

נבנה שתי מחלקות שיש להן איטרטור.

TwoArrays

למחלקה זו יהיה בנאי:

```
public TwoArrays(int[] a1, int[] a2)
```

והיא תממש את הממשק Iterable<Integer> על כל הכרוך בכך. האיטרטור שלה יחזיר בקריאה ראשונה ל-next את האבר הראשון של a1, אחר כך את הראשון של a2, ואז את הבא של a1 וכך הלאה, עד ששני המערכים נגמרים. למשל:

```
int[] a1 = { 1, 2, 3, 4 };
int[] a2 = { 100, 101, 102, 103, 104, 105, 106 };

TwoArrays aa = new TwoArrays(a1, a2);
for (int i : aa)
	System.out.print(i + " ");
```

ידפיס:

```
1 100 2 101 3 102 4 103 104 105 106
```

מי שלא זוכר על מה מדובר מוזמן להסתכל בהרצאה (וגם בתרגול כיתה) - יש שם כמה דוגמאות לאיטרטורים ואיך לגרום למחלקה להיות Iterable.

Combined<E>

זוהי הכללה של המחלקה הקודמת. יש לה בנאי שמקבל שני Iterable<E> - כלומר היא תוכל לקבל כל דבר שמממש Iterable<E>, כמו List או Set וכו':

```
public Combined(Iterable<E> first, Iterable<E> second)
```

היא בעצמה גם מממשת Iterable<E>, והרעיון הוא כמו קודם, שהיא תחזיר קודם מה-Iterable הראשון שקיבלה, אז מהשני וכך הלאה, עד ששניהם יגמרו. כך למשל:

```
public static void main(String[] args) {
	List<String> list = Arrays.asList("one", "two", "three");
	Set<String> set = new TreeSet<>();
	set.addAll(Arrays.asList("B", "A", "D", "C", "E"));
	Combined<String> c = new Combined<>(set, list);
	for (String s : c)
		System.out.print(s + " ");
}
```

ידפיס:

```
A one B two C three D E
```

הערה: לא יהיה נכון לשמור קודם את כל מה ששני האיטרטורים נותנים. חשבו למשל על המקרה שאחד או שני האיטרטורים הם אינסופיים, כלומר תמיד יש להם next וגם תמיד hasNext מחזיר true. זה בהחלט מצב אפשרי (למשל איטרטור שמחזיר כל פעם את מספר הפיבונאצ'י הבא).

אפילו לאיטרטורים סופיים, יכול להיות מאוד לא יעיל לעבור קודם על כל מה שהוא מייצר אי פעם, כי אולי מי שישתמש בו רק ישתמש בו עד שהוא מוצא את מה שהוא רוצה.

שאלה 2 (package simpleFX)

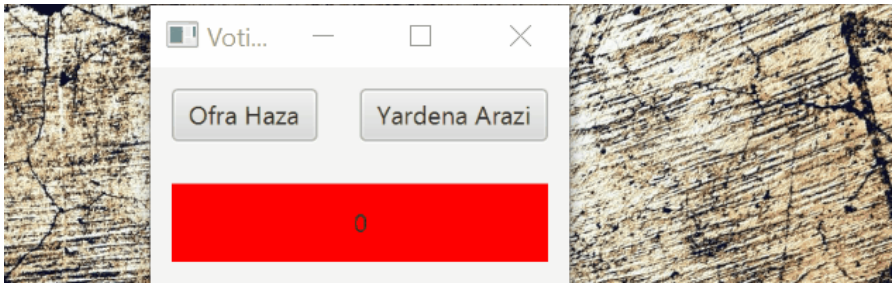
פעם פעם, לפני שנים רבות, היו שתי זמרות גדולות בארצנו: עפרה חזה וירדנה ארזי. חלק מעם העריץ את ירדנה ארזי, וחלק מהעם את עפרה חזה. כדי להכריע אחת ולתמיד מי יותר טובה, כתבו תוכנית javafx הנועדה לספירת כמות המעריצים של האחת מול השנייה. עשו זאת ללא שימוש ב-scene builder כלל, והגיעו למשהו שנראה כמה שיותר דומה לדוגמא הבאה.

מה חשוב:

1. שיהיו מרווחים בין הכפתורים, מסביב לכל האוביקטים, ובין השורות.
2. שיהיה צבע איזשהו לרקע של ה-label עם הספירה.

3. שההתנהגות בלחיצה תהיה כמו שמופיע פה למטה.
4. שרוחב ה-label בו מופיעה הספירה יהיה על פני כל השורה התחתונה (חוץ מהמרווח).

לא חשוב איך בדיוק שני הכפתורים מסודרים בשורה העליונה (אפשר למשל שהם יהיו יותר ממורכזים).



שאלה 3 (package SimpleFx2)

בדיוק כמו שאלה 2, אבל הפעם עשו הכל בעזרת scene builder. כלומר הקובץ שאתם כותבים הוא מינימלי ורק טוען את ה-fxml. כמובן שתצטרכו לייצר איזשהו Controller בעזרת scene builder, ולהוסיף לו מעט קוד.

הערה קטנה: כדי לקבוע רקע של אוביקט בתוך scene builder, צריך באזור Properties, תחת Style, לבחור את:

-fx-background-color

ולבחור צבע כרצונכם.

שאלה 4 (package mines)

המטרה שלנו היא לכתוב את משחק שולה המוקשים הידוע. מי שלא מכיר מוזמן לחפש minesweeper בגוגל, ולשחק. בשלב הראשון נכתוב מחלקה שנועדה לשחק את המשחק ללא ממשק גרפי. לאחר מכן נשתמש בה כדי להחזיק את הלוגיקה של הממשק.

Mines

זוהי מחלקה שנועדה לנהל את המשחק עצמו, בנפרד מהעובדה שהוא יהיה חלק מאפליקציה גרפית.

public Mines(int height, int width, int numMines)	בנאי לאתחול משחק ברוחב ובגובה הנתונים, ובו בדיוק numMines מוקשים המונחים באקראי.
public boolean addMine(int i, int j)	מוסיף מוקש במקום במיקום הנתון. ניתן להניח ששיטה זו נקראת רק מיד לאחר הבנאי, ולפני כל השיטות האחרות (למעשה יש אותה כדי שאוכל לעשות בדיקות)
public boolean open(int i, int j)	מסמן שהמשתמש פותח את המיקום הזה. מחזיר true אם זהו לא מוקש. בנוסף, אם ליד מיקום זה אין אף מוקש, אז פותח את כל המיקומים השכנים - וממשיך כך רקורסיבית.
public void toggleFlag(int x, int y)	שם דגל במיקום הנתון, או מסיר אותו אם כבר יש שם דגל.
public boolean isDone()	מחזיר אמת אם כל המיקומים שאינם מוקשים פתוחים.
public String get(int i, int j)	מחזיר ייצוג כמחרוזת של המיקום: אם המיקום סגור יחזיר "F" אם יש עליו דגל, ואחרת "." אם המיקום פתוח, יחזיר "X" אם זהו מוקש, אחרת את מספר המוקשים שליד ואם מספר זה הוא 0 אז " ".
public void setShowAll(boolean showAll)	יקבע את ערך השדה showAll. שדה זה מאותחל ל-false, אבל כשהוא true, אז ב-get וגם ב-toString, ערך החזרה הוא כאילו כל התאים פתוחים (אבל הוא לא באמת פותח אותם)
public String toString()	מחזיר תיאור של הלוח כמחרוזת לפי get לכל מיקום.

למשל:

```
Mines m = new Mines(3, 4, 0);
m.addMine(0, 1);
m.addMine(2, 3);
m.open(2, 0);
System.out.println(m);
m.toggleFlag(0, 1);
System.out.println(m);
```

ידפיס:

```
....
112.
  1.

.F..
112.
  1.
```

כמה הערות:

1. אפשר להניח שכל הקלטים חוקיים.
2. כן צריך לשים לב למקרים הגיוניים - כמו שמישהו מנסה לפתוח מיקום שכבר פתוח.
3. אין צורך להתייחס לניצחון או להפסד פה. כלומר אם נפתח מוקש או שאין עוד מה לפתוח אז אתם יכולים להתנהג איך שמתאים לכם להמשך הריצה.
4. מומלץ לכתוב מחלקה פנימית שתתאר מה קורה במיקום אחד:
   - האם יש שם מוקש, האם פתוח, האם יש שם דגל, כמה מוקשים יש ליד - כולל שיטת toString.
   - ואז המחלקה הראשית יהיה מערך דו מימדי של כאלו.
5. עוד המלצה מעניינת, היא לכתוב מחלקה פנימית שמתארת מיקום בודד. כלומר יש לה פשוט שני שדות i ו-j. אז תוכלו להשתמש בה בכל מני דברים פנימיים.
   - אחד הדברים היפים שאפשר לעשות עם זה, הוא להוסיף לה שיטה שמחזירה את אוסף מיקומי השכנים שלה (כרשימה או כל אוסף אחר - או אפילו כאיטרטור אם ממש בא לכם). עקרונית יש שמונה שכנים, אבל אולי חלק מחוץ ללוח. יש לפחות שני מקומות שונים בהם שיטה כזאת יכולה לעזור.

הערה:
כדי שנוכל לבדוק להריץ בדיקות אוטומטיות על התרגילים, אנחנו מבקשים שתממשו כמה דברים בדרך ספציפית, שהיא בהחלט לא היחידה (ואולי אפילו לא הכי טבעית, אבל זה מה שיצא):

1. כשמישהו מנסה לפתוח מוקש, אז הוא לא נפתח, אבל open מחזיר false.
2. בנוסף, במצב כזה, אין קריאה אוטומטית ל-showAll, אלא מי שישתמש במחלקה זאת, במקרה כזה, כנראה יקרא לשיטה זו.

MinesFX

מחלקה זו תהיה הממשק הגרפי של המשחק שלנו. היא תממש את כל ענייני הממשק, ותחזיק מופע של המחלקה Mines, בו תשתמש כדי לעשות כל דבר שקשור למשחק עצמו. בסופו של דבר המשחק אמור להתנהל כמו בוידאו הקצר פה.

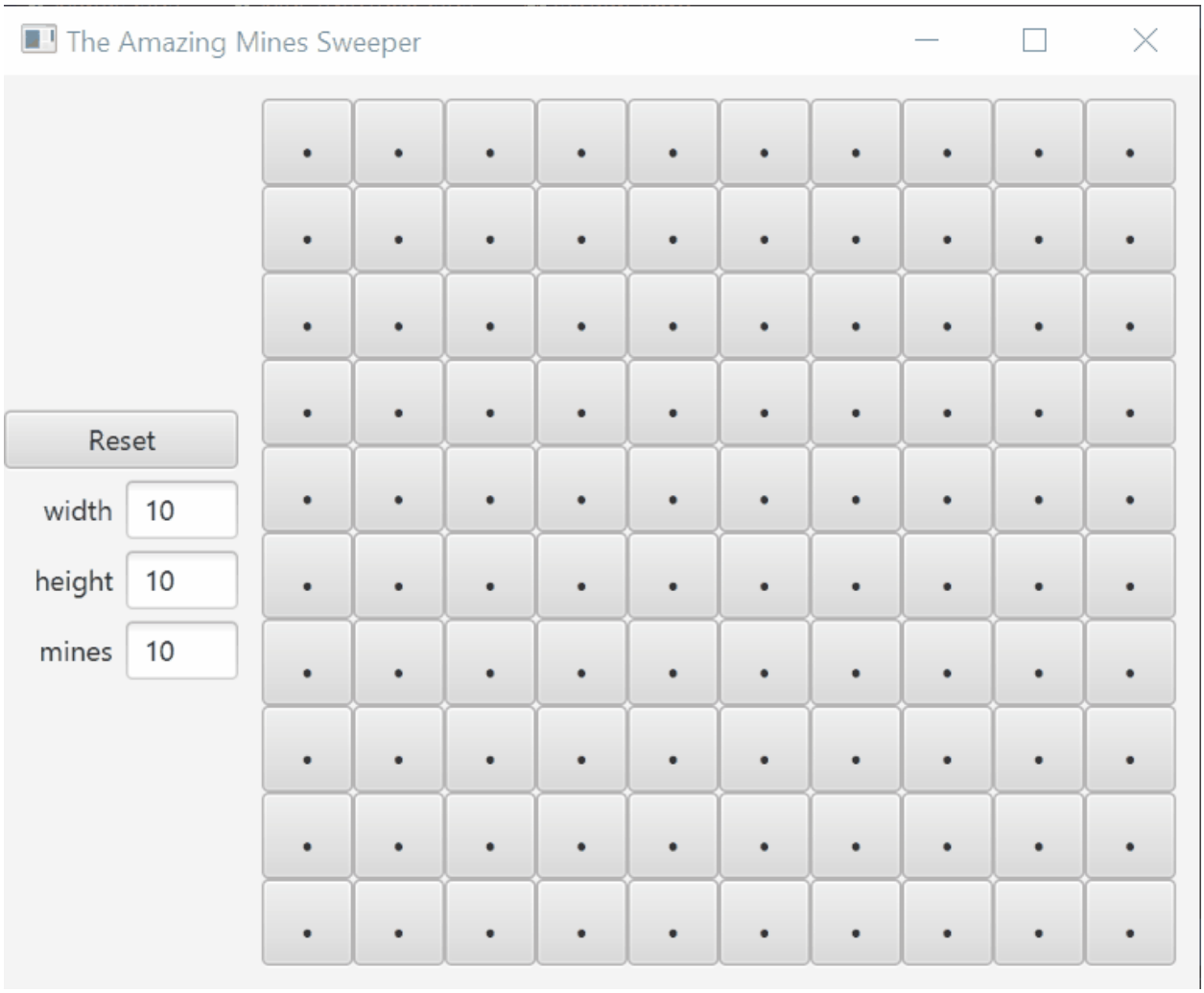
כמה דרישות והמלצות:

1. כמו שנאמר, הלוגיקה של המשחק צריכה להיות דרך המחלקה Mines ולא משוכפלת פה.
2. את המסך הראשי, מלבד הגריד עצמו יש לעשות ב- Scene Builder.
   - בתוכו יהיה רכיב כלשהו, למשל StackPane, שיהיה ריק, ולתוכו תכניסו את הגריד.
3. הגריד יווצר בקוד רגיל ולא ב-Scene Builder, ולכן גם הטיפול בארועים בו קורה בקוד רגיל (כמו בהרצאה).
4. כמובן שתהיה לכם גם מחלקת controller של ה-Scene Builder - תצטרכו לתקשר איתה כמובן. תזכרו שאתם יכולים לשנות את הקוד שלה איך שבא לכם. שימו לב לחלק "חיבור המחלקה שלכם וה-controller", שמסביר איך אפשר לקבל מצביע ל-controller.
5. הסימון של דגל (F) נעשה ע"י לחיצה על הכפתור הימני. תמיכה בו היא רשות, ותוסיף לכם 5 נקודות לציון התרגיל.
6. בהינתן לחיצה על כפתור תצטרכו למצוא מיהם ה-x וה-y שלו. יש כמה פתרונות לזה.
   - אולי הכי פשוט הוא להגדיר תת מחלקה של כפתור שבה יש גם שדות x ו-y, ואלו יהיו הכפתורים שלכם - לכן ברגע שילחצו על כפתור, אז בעזרת getSource תוכלו לקבל את הכפתור שלכם וממנו לקבל את x ו-y.
   - עוד דרך היא שה-EventHandler של לחיצה על כפתור תהיה מחלקה שיכולה לקבל בבנאי x ו-y, ואז לכל כפתור תצרו אוביקט חדש מהסוג הזה שמאותחל עם x ו-y משלו.
   - בקיצור, כשתגיעו לשם, תפתרו את הבעיה.

בונוס

אפשר לקבל בונוס של עד 15 נקודות על שאלה זו (לציון הסופי של התרגיל) ע"י מימוש תוספות מעניינות: אנימציות, רקעים, כל דבר שעולה ברוחכם. חשוב שזה יהיה מקורי ולא יופיע בעוד תרגיל של מישהו אחר. לא צריך המון המון כדי לקבל את כל הנקודות, אבל צריך יותר מהוספת קצת צבע ותכונות.

אם אתם רוצים להסביר לבודק מה הוספתם (כדי שלא יפספס) כתבו משהו קצרצר(!!) והוסיפו בקובץ README.txt באותו תיקייה כמו הקבצים של שאלה זו. לא חייבים לכתוב כזה, רק אם זה משהו שהוא צריך לדעת כדי להפעיל את מה שעשיתם. נא לא לשגע אותו עם דברים או הסברים מסובכים מדי. יש לו הרבה תרגילים לבדוק ומעט מאוד זמן.



הערה חשובה

את קבצי ה-fxml שלכם משאלה 3 ושאלה 4, דאגו להעתיק לפני ההגשה לתיקיית ה-src (כי יכול להיות שהם נמצאים רק בתיקיית ה-bin). אחרת הזיפ לא יכלול אותם והתרגילים שלכם לא יצליחו לרוץ.