תכנות מונחה עצמים – תרגיל 1

הקדמה

הכרנו את המחלקה ChatterBot, פתרון נהדר לצ'אטים גרועים. להורדה עכשיו בהנחה של 40%. היום נרחיב את השימוש במחלקה הזאת.

תזכורת: ChatterBot משתמשת בשיטת replyTo:

```
String replyTo(String statement)
```

אם ההצהרה היא "say <X>", השיטה תחזיר את הפלט <X>. אחרת, היא תחזיר באופן אקראי אחת מהתשובות שסופקו לפונקציית הבנאי עבור מקרה שכזה. כמו כן, מטילה השיטה מטבע כדי לקבוע אם להוסיף לתגובה האקראית את ההצהרה שסופקה.

אתם מוזמנים לעבור על הקוד של ChatterBot!

במדריך זה נעבור על כל התהליך שלב אחרי שלב כדי לאפשר לכולם ליישר קו ולהשלים פערי ידע שייתכן שקיימים. ככל שניצור תכניות מורכבות יותר, כך יעבור הדגש מהסבר על 'כיצד לבצע את הפעולה' להסבר 'מדוע אנחנו מבצעים את הפעולה'.

מטרת המדריך היא להקנות לכם ביטחון בתכנות והידור ב־Java לפני שנמשיך הלאה. אנחנו נשתמש במחלקת ChatterBot הקיימת, נוודא שאפשר להדר אותה באופן מקומי ולסיום נוסיף לה פונקציונליות נוספת:

1. שם לכל בוט.
2. תמיכה בשיחה מבוססת תור עבור כל מספר של בוטים.
3. נוסיף תמיכה בתשובות מורכבות לבקשות חוקיות. לדוגמה: אותו בוט יוכל לענות לפקודה " say orange" בכל אחת מהדרכים הבאות:
    - orange
    - say orange? Okay: orange
    - say orange yourself!

    ואילן בוט אחר יענה:
    - orange is my favorite thing to say. Here: “orange. orange.”
4. נראה איזה סוגי שיחות תוכלו ליצור באמצעות תבניות תשובה משלכם!

אז לפני שנתחיל:

1. כבר התקנתם עורך קוד? אם לא, היעזרו במדריך ההתקנות.
2. כבר התקנתם Java? כמובן שכן, הרי כבר צפיתם בסרטון הזה או נעזרתם במדריך ההתקנות.
3. כמו כן, צריך להיות לכם קובץ בשם ChatterBot.java. העבירו אותו לתיקייה ייעודית. כדאי שהתיקייה הזאת תהיה תיקיית משנה בתוך התיקייה הראשית שתשמש אתכם לשמירת כל התרגילים בקורס.

4. פתחו חלון טרמינל ונווטו לתיקייה שבה נמצא הקובץ ChatterBot.java. רגע, תזכיר לי איך עושים את זה שוב? טוב ששאלתם, ממש במקרה הכנו מדריך מיוחד (פרק 3, מדריך ההתקנות).
5. וודאו שההידור של הקובץ ChatterBot.java הסתיים בהצלחה (הוראות במדריך הזה. שימו לב שעדיין אי אפשר להריץ את הקוד מכיוון שאין לו שיטת main.

השגת אפס פונקציונליות

הדבר הראשון שתמיד נעשה הוא להגיע **למשהו** שאנחנו יכולים להריץ ולבדוק. הכלל הראשון בתכנות (בין כל יתר הכללים הראשונים) הוא שלא כותבים קוד מבלי לבדוק אותו באדיקות אחרי כל שלב. לכן נתחיל בכתיבת קוד פשוט שרק עובד הידור ורץ (כלומר יש לו אפס פונקציונליות).

1. צרו קובץ חדש בשם Chat.java (הוראות ליצירת קובץ קוד חדש תוכלו למצוא בפרק 3, עמוד 6 במדריך ההתקנות).
2. פתחו בעורך הקוד את הקובץ Chat.java החדש ואת הקובץ ChatterBot.java שקיבלתם (אפשר לגרור אותם לחלון עורך הקוד). ברוב עורכי הקוד אפשר לעבור במהירות בין הקבצים בעזרת קיצור המקשים Ctrl+Tab.
3. בקובץ Chat.java, הוסיפו את המחלקה Chat:

```
class Chat {

}
```

4. בתוך המחלקה, הוסיפו את השיטה main:

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

}
```

תזכורת: Java מחפשת שיטה עם ההצהרה המדויקת הנ"ל. ההצהרה חייבת להיות *public* ו-*static* (מה שזה לא יהיה), עליה להחזיר *void*, שמה חייב להיות *main* (באותיות קטנות) ורשימת הפרמטרים שלה צריכה לכלול רק את מערך המחרוזות שבו לא נשתמש (שם הפרמטר יכול להיות כל דבר ואנחנו משתמשים בשם *args* רק כי כך נהוג). זאת החתימה של השיטה *main*.

5. עכשיו אפשר להדר את שני הקבצים האלה ולהריץ את Chat באמצעות הפקודה "java Chat". הפקודה צריכה להחזיר את הפלט --- מבלי לעשות משהו נוסף.

יש! זאת אפס פונקציונליות. אולי זה עוד לא עושה הרבה... אבל היי, זה הרבה יותר כיף מהודעת שגיאה.

יצירת צ'אט בסיסי

עכשיו כשהידרנו את הקוד והוא רץ, אפשר להוסיף פונקציונליות. רשימת המטרות שלנו מגדירה את הפיצ'רים שנצטרך להוסיף, אבל מאיפה מתחילים? מה מתקנים קודם במכונית, את המנוע או את מד המהירות?

כלל אצבע: **מתחילים עם הפיצ'ר שיאפשר לבדוק את כל האחרים בצורה הטובה ביותר.**

במקרה שלנו, כל עוד הקוד שכתבנו לא באמת עושה משהו, לא נוכל לבדוק **שום דבר**! מסיבה זו אנחנו צריכים להתחיל עם צ'אט בסיסי שייצור בוטים וישתמש בהם. כך נוכל לבדוק פיצ'רים נוספים כשנוסיף אותם. אם נתחיל בפונקציונליות מורכבת יותר, נצטרך **להניח** שהקוד של הצ'אט הבסיסי עובד לפני שבדקנו אותו. ואני מניח שאתם יודעים מה אומרים על להניח.

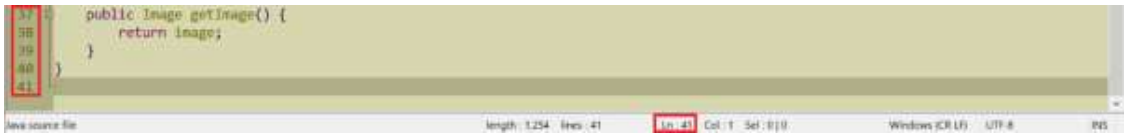
בתוך השיטה *main*, צרו מערך של שני ChatterBots. עבור בקשות לא חוקיות, ישיב בוט אחד "what" או "say I should say", והבוט השני "whaaat" או "say say". נדרשים כאן שלושה מערכים:

1. מערך אחד מכיל שני מצביעים אל ChatterBot (שברגע מצביעים אל כלום, null).
2. לכל אחד מהמצביעים האלה יוקצו עצמי ChatterBot חדשים. נסתכל על פונקציית הבנאי של ה־ChatterBot: נכון היא מצפה לקבל מערך של מחרוזות? לכן, צריך להתחיל ביצירת ה־ChatterBot הראשון עם המערך {"what", "say I should say"}.
3. ולהמשיך ליצירת ה־ChatterBot השני עם המערך {"whaaat", "say say"}.

כדי להיזכר כיצד לאתחל את המערכים האלה, אפשר להיעזר בטיפים למימוש.
רק רגע! לא לשכוח שצריך להדר את הקוד לפני שממשיכים.

תיקון שגיאות הידור

השיטה הטובה ביותר היא להתמקד רק בשגיאת ההידור ***הראשונה****, לתקן את הבעיה ולנסות שוב. צריך לקרוא את הודעות השגיאה של המהדר בסבלנות. הוא באמת משתדל לעזור! בכל שגיאת הידור מוזכר מספר שורה. גם עורך הקוד מציג את מספרי השורות, אם לא לצד שורות הקוד עצמן, אז בתחתית החלון מוצג מספר השורה שבה נמצא כרגע הסמן:*



שימו לב למספר השורה שבהודעת השגיאה של המהדר וקראו בעיון את תיאור השגיאה. בדרך כלל אפשר למצוא כאן רמז למקור הבעיה.

ראשית חכמה, נצטרך ליצור את הפרמטר "String statement" ששומר את המשפט הנוכחי של השיחה. צריך להחליט עבורו על ערך ראשוני כלשהו (אשר אליו יגיב הבוט הראשון) ולהגדיר את המשתנה.

עכשיו ללולאה האינסופית שתעבור בין כל הבוטים שבמערך לפי הסדר (זה צריך לעבוד עבור מערכים בכל הגדלים). עקרונית, כבר אפשר לעשות את זה באמצעות הידע שיש לכם ב־Java. בכל זאת, מכיוון שאנחנו כאן כדי ללמוד, בואו נכיר שלושה מנגנונים של Java שיכולים לעזור לנו כאן.

1. <u>לולאת foreach</u>

ב־Java, המונח foreach הוא **לא** מילת מפתח. הוא מתייחס לשימוש מסוים במילת המפתח for שאתם כבר מכירים.

ב-Java יש שתי דרכים להשתמש במילת המפתח for:

a. השימוש המוכר והאהוב:

```
for([initial statement] ; [condition] ; [statement to perform after
each iteration] ) { ... }
```

לדוגמה:

```
for(int i = 0 ; i > 10 ; i++) {
        System.out.println(i);
}
```

b. דרך נוספת להשתמש במילת המפתח for נקראת לולאת foreach. היא מניחה שכבר יש לכם מבנה נתונים מסוים שאתם רוצים להתייחס לכל אחד מאיבריו לפי הסדר. לדוגמה, נניח שיש לכם מערך של מספרים שלמים: "int[] arr = {1, 2, 3};". אזי זו דוגמה של לולאת foreach שמדפיסה את האיברים שלו:

```
for(int num : arr) {
System.out.println(num);
}
```

כמובן, אפשר להריץ תהליך איטרטיבי דומה על מערכים מכל הסוגים ולא רק של מספרים שלמים.

אם יש לכם היכרות קודמת עם שפות תכנות נוספות, אתם כבר בטח מכירים לפחות אחד מהשימושים האלה למילת המפתח *for*. למי שמגיע מרקע של C++11 (רובכם), שני השימושים עובדות בג'אווה בדיוק באותה צורה.

כדי לבצע לולאה איטרטיבית אינסופית על הבוטים, אפשר להשתמש בלולאת foreach באופן דומה לדוגמה הבאה:

```
Scanner scanner = new Scanner(System.in);
while(true) {
        for(ChatterBot bot : bots) {
                statement = bot.replyTo(statement);
                System.out.print(statement);
                scanner.nextLine(); // מחכים ל"אנטר" לפני שממשיכים
        }
}
```

שימו לב: מקטע הקוד הזה מניח שהקובץ שלכם מכיל import java.util.Scanner לפני תחילת המחלקה.

2. <u>האופרטור modulo</u>

כמעט בכל שפות התכנות יש דרך מובנית לחשב את *שארית החלוקה* של פעולת חילוק. זאת פונקציה כל כך נפוצה עד שהיא אחת הבודדות שהמעבד יכול לבצע בכוחות עצמו (יחד עם חיבור, כפל וכן הלאה), והיא נקראת *modulo*. בשפות תכנות למטרות כלליות (כמו Java), פונקציית modulo מחושבת עם הסימן '%' (אחוז). למשל:

```
5 % 2 == 1 // השארית של חלוקת 5 ב־2 היא 1
12 % 3 == 0 // השארית של חלוקת 12 ב־3 היא 0
27 % 8 == 3
17 % 23 == 17 // עבור כל N<M, N%M שווה N
```

בשלב הזה לא נדבר על האופן שבו modulo פועלת על ערכים שלילים.

איך זה שימושי לאיטרציה על בוטים? טוב ששאלתם. נניח שהפרמטר *len* מבטא את אורך המערך *arr*. אזי, עבור כל מספר לא שלילי *i* הביטוי *i % len* הוא אינדקס חוקי עבור *arr* (מדוע?). עם קידום הערך של *i*, עובר הביטוי *i % len* בין האינדקסים של *arr* לפי הסדר. כך אפשר להשיג את אותה פונקציונליות של פונקציית ה־foreach הנ"ל גם באופן הבא:

```
Scanner scanner = new Scanner(System.in);
for(int i = 0 ; ; i = i+1) {
       statement = bots[ i % bots.length ].replyTo(statement);
       System.out.print(statement);
       scanner.nextLine();
}
```

הלו, אדוני! מה זה שם באמצע הצהרת ה־for – השארת את התנאי שלך ריק! אמת (סליחה, הייתי חייב): בלולאת *for* תנאי ריק פירושו לולאה אינסופית (אפשר להחליף בערך *true*).

3. <u>האופרטורים =+, ++</u>

טוב, מספיק עם כל ה-"i = i + 1" הזה. בדומה לשארית חלוקה, קידום משתנה מספרי הוא פעולה כל כך נפוצה עד שהוא זכה לאופרטור ייעודי משלו:

```
i++; // כמובן זה עובד לכל שם משתנה
```

באופן דומה, הגדלת ערכו של המשתנה *num* בסכום amount מסוים היא פונקציה שימושית מאוד וכמו ב־Python (ואחרות) אפשר לכתוב אותה באופן הבא:

```
num += amount;
```

מכאן שהביטוי

```
i += 1;
```

משרת את אותה מטרה כמו

```
i++;
```

כל זאת רק בשביל לומר שאפשר לכתוב גם:

```
Scanner scanner = new Scanner(System.in);
```

```
for(int i = 0 ; ; i++) {
        statement = bots[ i % bots.length ].replyTo(statement);
        System.out.print(statement);
        scanner.nextLine();
}
```

כדאי לדעת: ב־Java קיימים אופרטורים נוספים לביצוע פעולות דומות, ובהם --, =-, =*, =/, =%. אתם אולי כבר יכולים לנחש מה תפקידו של כל אחד, או לחילופין אתם מוזמנים לחפש אותם ברשת!

עכשיו הוא הזמן (עכשיו) להדר את הקוד (עכשיו) ולהריץ אותו, ועדיף שעכשיו. בתי הקברות מלאים במתכנתים שפשוט הניחו שהקוד שלהם עובד.

נפתח לכם צ'אט מטופש? נהדר! לא נפתח? הקשיבו למהדר; הוא כן נותן מידע שימושי.

הוספת שם לכל בוט

לתת לבוטים אנונימיים לדבר בלי הפסקה עם בוטים אנונימיים אחרים זאת קצת התעללות. אפשר לעזור להם בדרך הבאה:

1. הוספת השדה "String name" למחלקה ChatterBot.
2. לקבל שם בתור הפרמטר הראשון של הבנאי ולהקצות אותו לשדה.
3. הוספת השיטה "String getName()" שתחזיר את השם הזה.
4. בקובץ Chat.java, במקום בו מאותחל מערך ה־*bots*, הקצות להם שמות לבחירתכם. לדוגמה:

```
new ChatterBot("Kermit", … );
```

5. בלולאת הצ'אט להדפיס גם את שם הבוט לפני התשובה.
הצ'אט יראה ככה:

```
Sammy: what
Ruthy: what what
…
```

עדיף להשתמש בשיטה "bot.getName()" על פני "bot.name". מדוע? הכל יתברר מיד בסיום התרגיל.

עכשיו, הדרו את הקוד ובדקו אותו!

הוא עובד? מצויין! דברים מעניינים עוד לפנינו.

תשובות מורכבות לבקשות חוקיות

כרגע מגיב הבוט לבקשות בסגנון "say <phrase>" כך: <phrase>. המטרה שלנו היא שהבוט ייתן תשובה מורכבת יותר ולא סתם <phrase>. לדוגמה: בתגובה לבקשה "say door" יענה הבוט משהו בסגנון של "you want me to say door, do you? alright: door". או שהוא יענה משהו כמו " okay, here goes: door".

הבוט ישתמש בתבניות תשובה שלתוכן יכניס במקומות מתאימים את הביטוי *phrase* שעליו התבקש לחזור. הנה דוגמה לתבנית תשובה כזאת:

"You want me to say <phrase>, do you? alright: <phrase>"

לאחר מכן, כשהבוט מקבל את הבקשה "say door" הוא יחליף את כל המופעים של תת המחרוזת <phrase> שבתבנית במחרוזת "door" (אל חשש, אתם תקבלו את הקוד שמבצע את ההחלפה הזאת).

כדי להפוך את הדברים למעניינים עוד יותר, לכל בוט יוגדר מערך של תבניות תשובה והבוט יבחר מתוכו תשובה באופן אקראי. כדי לעשות את זה, תקבל המחלקה פרמטר נוסף בפונקציית הבנאי שלה: מערך של תשובות אפשריות. כך מתקבלת רשימת הפרמטרים הבאה של הבנאי של המחלקה ChatterBot:

```
ChatterBot(String name, String[] repliesToLegalRequest, String[]
repliesToIllegalRequest)
```

אם נחזור לדוגמה האחרונה שלנו, אתחול הבוט יתבצע באופן הבא:

```
ChatterBot bot = new ChatterBot(
"botty", // שם הבוט
new String[]{ "say <phrase>? okay: <phrase>" }, // תשובה אפשרית
לבקשות חוקיות
new String[]{ "what" } // תשובה לבקשות לא חוקיות
);
```

במקרה כזה, השיטה הבאה:

```
bot.replyTo("say door")
```

תחזיר:

```
"say door? okay: door"
```

והשיטה:

```
bot.replyTo("say something")
```

תחזיר:

```
"say something? okay: something"
```

מאוד בוגר.

לסיכום: כשמקבל הבוט בקשה חוקית (כלומר בקשה המתחילה בטקסט " say"), יבחר הבוט תשובה באופן אקראי מתוך המערך השובות לבקשות חוקיות. לאחר מכן, יחליף הבוט את כל המופעים של תת המחרוזת "phrase>" שבתבנית התשובה שנבחרה בביטוי שבבקשה ויחזיר את התוצאה.

1. צריך להוסיף את השדה "String[] repliesToLegalRequests" למחלקה ChatterBot. כמו כן, צריך להוסיף פרמטר בנאי עם אותו שם ולאתחל את השדה באמצעות הפרמטר. כך מתבצע

האתחול עם שדה בעל השם הזהה "repliesToLegalRequests". אבל רגע, תזכיר לי שוב מדוע יש לולאה בבנאי? לרענון, כדאי לקרוא שוב את ההיגיון והסיבה שמאחורי התחביר הזה.

2. כדי שאפשר יהיה לבדוק את הקוד (כלומר להריץ אותו), צריך לתקן את הקריאות לפונקציית הבנאי בקובץ Chat.java. אם הידור הקוד נכשל, לא להתרגש. קחו נשימה וקראו בעיון את הודעת השגיאה. תזכורת: צריך להתייחס קודם כל לשגיאת ההידור הראשונה, לנסות לתקן אותה ואז להריץ שוב את המהדר. לא פעם, בשלב זה נעלמות כל יתר השגיאות מעצמן!
3. צריך לאתר את הקוד שמטפל בבקשות חוקיות. הקוד פשוט מחזיר את המילה או משפט שבאו אחרי הטקסט " say". במקום להחזיר את הביטוי עצמו, שמרו אותו במשתנה מקומי בשם *String phrase* (לא שדה – עדיף להשתמש במשתנים מקומיים במקום בשדות כשזה אפשרי).
4. השדה החדש "repliesToLegalRequests" מכיל את תבניות התשובה האפשריות. צריך לבחור אחת באופן אקראי. איך? הקוד הזה כבר נמצא בשיטה *respondToIllegalRequest* שבוחרת באופן אקראי איבר מתוך המערך – כדאי להעיף בה מבט.
5. החזירו את התשובה שנבחרה, כשתת המחרוזת "<phrase>" מוחלפת במשתנה המקומי *phrase* שבו שמור הביטוי עצמו. אפשר לעשות זאת עם השיטה *String.replaceAll*:

```
String reply = responsePattern.replaceAll("<phrase>", phrase);
```

6. זה המקום לעצור, להדר את הקוד ולהריץ אותו. האם הבוטים מדברים ביניהם? המשיכו רק אם התשובה היא כן.

7. הקוד לטיפול בבקשות חוקיות נמצא כרגע כנראה בשיטה *replyAll*. זה לא רעיון רע לייצא את הקוד הזה לשיטה חדשה:

```
String respondToLegalRequest(String statement)
```

הדרו את הקוד!

8. הממם... עכשיו יש בקוד מה שנקרא מחרוזת קסם (Magic String). המחרוזת הזאת נותנת משמעות מיוחדת לתת המחרוזת "<phrase>" גם במחלקה ChatterBot וגם בשיטה *main*. נניח שרצינו להשתמש במחרוזת שונה למטרה הזאת, לדוגמה:
"<the_stuff_the_bot_was_asked_to_say>"? יהיה עלינו לעבור על כל הקוד ולבצע את ההחלפות באופן יזום.

במקום, הגדירו את הקבוע הבא במחלקה ChatterBot:

```
static final String REQUESTED_PHRASE_PLACEHOLDER = "<phrase>";
```

ונשנה את הקוד כך שישתמש בקבוע הזה במקום בתת המחרוזת "<phrase>" המובנית בקוד (בשני הקבצים). שימו לב שכדי לגשת לקבוע הזה מהשיטה *main* צריך לציין קודם כל את שם המחלקה. במילים אחרות, ב־*main* תכתבו:

```
ChatterBot.REQUESTED_PHRASE_PLACEHOLDER
```

תשובות מורכבות יותר לבקשות לא חוקיות

חשבו על בקשות לא חוקיות (לא מתחילות בתחילית REQUEST_PREFIX). כרגע, בוחר הבוט אחת מבין התשובות המוגדרות ומטיל מטבע כדי להחליט אם להוסיף את הבקשה המקורית אחרי התשובה הזאת. לדוגמה: בתגובה לבקשה הלא חוקית "orange", ובהנחה שהבוט בחר את התשובה "what", יענה הבוט "what" או "what orange".

חלק זה של התרגיל משתמש באותו מנגנון שכבר יצרנו לטיפול בבקשות חוקיות, רק שהפעם הוא ישמש לטיפול בבקשות לא חוקיות. במילים אחרות: גם כאן התשובות לבקשות הלא חוקיות יכללו תת־מחרוזת המוגדרת בקבוע ILLEGAL_REQUEST_PLACEHOLDER. לאחר בחירת תשובה מסוימת באופן אקראי, כל המופעים של תת המחרוזת ממלאת המקום יוחלפו בבקשה הלא חוקית עצמה (רק שהפעם בשיעור של 100% מהזמן במקום 50% מהזמן).
לדוגמה: בתגובה לבקשה לא חוקית יענה הבוט משהו כמו:

```
“say what?” + ILLEGAL_REQUEST_PLACEHOLDER + “? what’s” +
ILLEGAL_REQUEST_PLACEHOLDER + “?”
```

כך שאם נשלחה הבקשה "hello there!", ישיב הבוט:

```
“say what? hello there!? what’s hello there!?”
```

כדאי להתחיל בהגדרת הקבוע:

```
static final String ILLEGAL_REQUEST_PLACEHOLDER = “<request>”;
```

ולהמשיך לבנות על הבסיס הזה.

שימו לב שעבשיו הטיפול בבקשות חוקיות ולא חוקיות נעשה באופן דומה מאוד! יצאו את הקוד המשותף לשיטה חדשה שתקבל מערך מחרוזות של תשובות אפשרויות שממנו תיבחר תשובה, תת־מחרוזת ממלאת מקום שהתשובות האפשריות עשויות להכיל, ואת המחרוזת שתחליף את כל המופעים של אות תת מחרוזת ממלאת המקום. מה תהיה החתימה של השיטה הזאת? צריך לתת לה שם שמייצג את תפקידה, משהו כמו: replacePlaceholderInARandomPattern.

מדוע הפונקציונליות הישנה לטיפול בבקשות לא חוקיות (הטלת מטבע כדי להחליט אם להוסיף את הבקשה הלא חוקית לסוף התשובה) היא מקרה מיוחד של הפונקציונליות החדשה? במילים אחרות, איך תשתמשו בפונקציונליות החדשה, מבוססת ההחלפה, כדי לחקות את הפונקציונליות הישנה?

Kermit: what
Frog: say say what
Kermit: I don't want to say say what
Frog: say say I don't want to say say what
Kermit: I don't want to say say I don't want to say say what
Frog: say say I don't want to say say I don't want to say say what
Kermit: I don't want to say say I don't want to say say I don't want to say say what
Frog: whaaat
Kermit: what whaaat
Frog: whaaat

Kermit: what
Frog: whaaat
Kermit: say what? whaaat? what's whaaat?
Kermit: say what? whaaat? what's whaaat?
Kermit: I don't want to say what? whaaat? what's whaaat?? okay: what? whaaat? what's whaaat?

סוף דבר

בטח כבר עלו לכם רעיונות לתבניות תשובה משעשעות כתגובה לבקשות חוקיות או לא חוקיות. יש לכם רעיונות לשיחות מעניינות שאפשר ליצור בעזרת המנגנונים שיצרנו? את השיחות שהתקבלו כתבו בקובץ ה-README של התרגיל, ואם השיחה ממש טובה אולי נפרסם אותה!

חומר למחשבה: השיחה תהיה הרבה יותר מהנה ומעניינת אם בכל מערך תבניות תשובה יהיו לפחות שתי תשובות, אחרת השיחה תהיה צפויה ותחזור על עצמה.

זה הסוף! מקווים שנהניתם מהמיני-תרגיל הזה.

עיבוד שפה טבעית

אם אי פעם התחשק לכם ליצור בוט עם קצת בינה מלאכותית, יש לא מעט גישות קיימות שיכולות לעזור לכם להתחיל ברגל ימין; חלקן אפילו מאוד פשוטות לשימוש.
אחד מהצ'אטבוטים הוותיקים והמפורסמים ביותר הוא Eliza המחקה פסיכולוגית ממוקדת מטופל (פסיכולוגית שתפקידה מצומצם ובדרך כלל תגיב עם שאלות מנחות פשוטות). חפשו " chat with Eliza bot": יש כמה וכמה אתרים שיאפשרו לכם להתנסות בשיחה איתה כבר עכשיו, וזה אפילו בחינם. תוכלו גם לקרוא על הרעיון הפשוט שמאחוריה. כיום קיימים כמובן מנועים טובים ומשוכללים יותר: העוזרת האישית שבטלפון שלכם יכולה כנראה לספר לכם משהו על זה.
תת התחום במדעי המחשב שעוסק בניתוח ויצירת טקסט/דיבור בשפה אנושית נקרא "עיבוד שפה טבעית" (ובראשי תיבות באנגלית: NLP).

טיפים למשתמש

כרגע אנחנו מכירים רק דרך אחת להצהיר על משתנה מערך: "ChatterBot[] bots".
אבל יש עשרות דרכים שונות לאתחל אותו. אנחנו נמנה כמה מהן כאן כדי שתוכלו לבחור בזאת שנראית לכם הכי מתאימה, אבל תוכלו כמובן להשתמש גם בדרכים אחרות שלא מופיעות ברשימה.

אפשרות 1:

```
String[] illegalReplies1 = { “say I should say”, “what” };
```

```
String[] illegalReplies2 = { "say say", "whaaat" };
ChatterBot[] bots = new ChatterBot[2];
bots[0] = new ChatterBot(illegalReplies1);
bots[1] = new ChatterBot(illegalReplies2);
```

אפשרות 2:

```
ChatterBot[] bots = new ChatterBot[2];
bots[0] = new ChatterBot(
       new String[] {
              "say I should say",
              "what"
       }
);
bots[1] = new ChatterBot(
       new String[] {
              "say say",
              "whaaat"
       }
);
```

אפשרות 3:

```
ChatterBot[] bots = {
new ChatterBot(
              new String[] { "say I should say", "what" }
       ),
new ChatterBot(
              new String[] { "say say", "whaaat" }
       )
};
```

הערות כלליות בנושא אתחול מערך:

1. התחביר המקוצר:

```
int[] arr = { 1, 2 };
```

חוקי ועובד רק בליווי הצהרת מערך מתאימה.

התחביר הבא לא יעבוד:

```
int [] arr;
arr = { 1, 2 }; // שגיאה!
```

לעומת זאת, התחביר המלא לאתחול מערך תמיד חוקי ויעבוד:

```
int[] arr1;
arr1 = new int[] { 1, 2 }; // חוקי
Int[] arr2 = new int[] { 1, 2 }; // גם חוקי
```

2. המהדר של Java מתעלם משורות ריקות כך שהבחירה ביצד לרווח את הקוד נתונה לשיקולכם והיא קוסמטית בעיקרה.

הוראות הגשה

עליכם להגיש קובץ jar בשם ex1.jar המכיל את הקבצים הבאים:

1. ChatterBot.java – בקובץ זה יופיע המימוש שלכם למחלקה ChatterBot (בהתבסס על התבנית שסופקה לכם). שימו לב כי **אין צורך להגיש את הקובץ Chat.java**. הקפידו על קריאות הקוד, הימנעות מ"קבועי קסם" ותיעוד נכון. שימו לב שהמחלקה צריכה לכלול בנאי, את המתודות replyTo ו-getName וכן את הקבועים REQUEST_PREFIX, REQUESTED_PHRASE_PLACEHOLDER ו-ILLEGAL_REQUEST_PLACEHOLDER.
2. קובץ README (ללא סיומת) – בשורה הראשונה בקובץ זה יופיע שם המשתמש שלכם, בשורה השנייה יופיע מספר תעודת הזהות שלכם, השורה השלישית תהיה ריקה, ומהשורה הרביעית ואילך עליכם לצרף דוגמה לשיחה של הבוטים שלכם.

בהצלחה!