הנחיות לפתרון תרגילי הבית

- על הקוד המוגש להיות מתועד היטב ועליו לכלול:
  - מפרט, כפי שהודגם בתרגול.
  - תיעוד של כל מחלקה ומתודה ושל קטעי קוד רלוונטיים.
  - במידת הצורך, יש להוסיף תיעוד חיצוני.
  - שאלות ניתן להעלות בפורום באתר ה moodle.

הנחיות להגשת תרגילי בית

- תרגילי הבית הם חובה.
- ההגשה בזוגות בלבד.
- עם סיום פתירת התרגיל, יש ליצור קובץ דחוס להגשה המכיל את:
  - כל קבצי הקוד והתיעוד.
  - פתרון לשאלות ה"יבשות" בקובץ Word או PDF. על הקובץ להכיל את שמות ומספרי תעודות הזהות **של שני הסטודנטים** המגישים.
  - הקובץ המוגש יקרא hw0_<id1>_<id2>.zip כאשר <id1> ו-<id2> הם מספרי הזהות של הסטודנטים המגישים. לדוגמא hw1_12345678_9876541.zip (כמובן יש להשתמש במספרי הזהות **שלכם**)

הקפידו על הוראות ההגשה על מנת למנוע אי נעימות מיותרת.

- הגשת התרגיל היא **אלקטרונית בלבד**, דרך אתר הקורס **ע"י אחד מבני הזוג בלבד**.

- תרגיל שיוגש באיחור וללא אישור מתאים (כגון, אישור מילואים), יורד ממנו ציון באופן אוטומטי לפי חישוב של 5 נקודות לכל יום איחור ועד 2 ימי איחור שלאחריהם לא תתאפשר הגשה כלל.
- על הקוד המוגש לעבור הידור (קומפילציה). על קוד שלא עובר הידור יורדו 30 נקודות.

מועד ההגשה:
יום ג', 14/11/17

המטרות של תרגיל בית זה הן:
- ליצור בסיס משותף שממנו נתחיל את הקורס.
- להכיר את שפת Java ואת סביבת העבודה Eclipse.
- להתנסות במימוש מפרט בסיסי.

התרגיל פשוט יחסית ופתרונו אמור לגזול זמן לא ממושך מסטודנטים שהגיעו לקורס עם הרקע המתאים.

שאלה 1 (15 נקודות)

כתבו מסמך באנגלית המכיל רשימת כללים לאופן כתיבת קוד (coding style guidelines). עליכם להדגים את היישום של כללים אלה בשאלות הבאות ולהשתמש בהם באופן עקבי בכל תרגילי המחשב בהמשך הקורס. על רשימת הכללים לכלול:
- אופן נתינת שמות למחלקות, מתודות ומשתנים (naming conventions).
- אורך שורה, צורת האינדנטציה והעימוד, רווחים ושורות ריקות.
- צורת כתיבת הערות ותיעוד.
- כל מידע רלוונטי אחר.

במידת הצורך, יש לרשום במסמך קטעי קוד קצרים להדגמה.
ניתן למצוא דוגמאות למסמכים המכילים רשימת כללים באתרים הבאים:
http://geosoft.no/development/javastyle.html
http://www.javaranch.com/style.jsp
https://google.github.io/styleguide/javaguide.html

להגשה "יבשה": מסמך הכללים הנ"ל.

שאלה 2 (30 נקודות)

מדד הקריאות של Flesch הוא כלי עזר לניתוח הקריאות של מסמכי טקסט. המדד נמצא בשימוש נרחב במעבדי תמלילים מכיוון שהוא קל לחישוב ונותן תוצאות בעלות משמעות. תוצאת המדד היא לרב מספר בין 0 ל-100 כאשר מספר גבוה יותר מציין טקסט קריא יותר. לדוגמה, טקסט עם ציון בתחום [91 100] יהיה ניתן להבנה ע"י תלמיד בכיתה ה', טקסט עם ציון בתחום [51 60] יהיה ניתן להבנה ע"י תלמיד י"ב וטקסט עם ציון בתחום [0 30] יהיה ניתן להבנה ע"י בוגר אוניברסיטה.

עליכם לכתוב תכנית שתקרא קובץ טקסט ותדפיס את מדד Flesch שלו. התכנית תקבל את שם הקובץ בשורת הפקודה (command line argument). אם הפרמטר שסופק אינו שם קובץ קיים או אם מספר הארגומנטים שסופקו שגוי, תודפס הודעת שגיאה מתאימה.
יש לחשב את המדד באופן הבא:

- מחשבים את מספר המילים בקובץ. מילה היא רצף תווים המופרדים ברווח או באחד מהתווים הבאים: .,' : ; ? { } [] = - + _ ! @ # $ % ^ & * ()
- מחשבים את מספר ההברות המרכיבות את המילים בקובץ. כדי לפשט את תהליך זה, נגדיר את הכללים הבאים:
  - כל קבוצה של תנועות (a, e, i, o, u, y) סמוכות נחשבת כהברה אחת. לדוגמה, "ea" במילה "real" נספרת כהברה אחת אבל "a … e" במילה legal נספרת כשתי הברות.
  - יוצאת דופן היא האות "e" בסוף מילה שלא נספרת כהברה.
  - בכל מילה יש לפחות הברה אחת גם אם שני הכללים הקודמים נתנו את המספר אפס.
- מחשבים את מספר המשפטים בקובץ. משפט מסתיים בנקודה, פסיק, נקודה-פסיק, סימן שאלה או סימן קריאה.
- המדד מחושב לפי הנוסחה הבאה:

$$Flesch = 206.835 - 84.6 \times \frac{\text{מספר ההברות}}{\text{מספר המילים}} - 1.015 \times \frac{\text{מספר המילים}}{\text{מספר המשפטים}}$$

הנחיה 1: ניתן לטפל במקרי קצה שונים בכל דרך הגיונית.
הנחיה 2: ניתן להיעזר במחלקות java.io.BufferedReader, java.io.FileReader, java.util.File, java.util.StringTokenizer

להגשה ממוחשבת: קובץ המכיל את הפתרון (כולל תיעוד).
להגשה "יבשה": פלט דוגמה של התוכנית על קבצי הטקסט:

https://archive.org/stream/TheLittleEngineThatCould_201603/The%20little%20engine%20that%20could_djvu.txt

https://archive.org/stream/grapeswrath00cablgoog/grapeswrath00cablgoog_djvu.txt

שאלה 3 (45 נקודות)

נתונים הקבצים Coin.java ו-Wallet.java המכילים מפרט ללא מימוש של מחלקות המייצגות מטבע וארנק.

א.
ממשו את המחלקות כך שיעמדו במפרט הנתון וכתבו תכנית בדיקה שתדגים את פעולתן.

הנחיה: ניתן להיעזר במחלקה java.util.ArrayList. כדי להגדיר, למשל, רשימה של איברים מטיפוס String ניתן לרשום:

```
List<String> myList = new ArrayList<>();
```

ב.
הוחלט להוסיף מתודה חדשה שתקרא payMinimum(). המתודה בעלת מפרט דומה לזה של מתודה pay() אך כעת נוספה הדרישה שמספר המטבעות שישולם הוא מספר המטבעות המינימלי. פסקאות ה @returns וה @modifies ללא שינוי מהמתודה pay() .

```
* @effects tries to match at least the sum "sum" with the minimum number
of coins available from the wallet.
* If transaction is possible, removes the paid coins from the wallet; else;
changes nothing
```

ב1. כיצד יש לשנות את המתודה pay() כל שתתאים למפרט החדש?
ב2. האם המפרט החדש חזק יותר או חלש יותר כעת? הסבירו.
ב3. ממשו את המתודה payMinimum()

ג.
הוחלט להוסיף למפרט של המתודה pay() במחלקה Wallet את הפסקה הבאה:

```
@requires sum < total money in wallet
```

ג1. אילו שינויים ידרשו במימוש של מתודה זו כדי שתעמוד במפרט החדש? הסבירו.
ג2. האם המפרט החדש חזק יותר או חלש יותר כעת? הסבירו.
ג3. האם יש צורך בשינויים נוספים במפרט בעקבות השינוי

להגשה ממוחשבת: הקבצים Coin.java, Wallet.java, WalletTest.java
להגשה "יבשה": א. פלט תכנית הבדיקה. ב. תשובות לסעיפים ב' וג'.

שאלה 4 (10 נקודות)

כעת נרצה לממש אוסף של מטבעות. אוסף של מטבעות מכיל מטבע **יחיד** מכל **סוג**.
(10אג' 50 אג' וכו).
השתמשו ב Wallet וב Coin על מנת לממש אוסף מטבעות CoinCollection **ללא שימוש בהורשה**.

להגשה: מימוש CoinCollection.java הכולל תיעוד וקובץ בדיקה אשר בודק את המימוש.