מבוא לתכנות מונחה עצמים מטלה 2

הנחיות כלליות:

1. את המטלה עושים בזוגות ,יש להגיש את כל המטלות בזמן לפי הנחיות ,על כל איחור לא מוצדק תהיה הורדת ניקוד.
2. המטלות תיבדקנה באופן אוטומטי באספקטים של" העתקות קוד "אין לבצע שום העתקה של קודים בין קבוצות שונות ,מותר לעשות שימוש בקוד פתוח ,אבל חובה לציין זאת בפירוש ולהביא את המקור המדויק .למען הסר ספק :שימוש בקוד פתוח (או כל קוד זמין ברשת) שלא יצוין מקור הקוד יחשב כהעתקה!
3. יש לתעד את כל המתודות של הפרויקט שלכם בעזרת javadoc.
4. **צריך להגיש קובץ ZIP שם הקובץ - מספר זהות ראשון מקו תחתון מספר זהות שני.**
   יש לקבץ קבצי JAVA בלבד.

חלק א שימוש בהורשה

בתרגיל זה נכיר צבים חביבים, ונשתמש בהורשה ופולימורפיזים כדי להרחיב את משפחת הצבים . המחלקה SimpleTurtle מגדירה עבורנו צב רגיל. יצירה של צב מציירת דמות של צב במרכזו של מסך גרפי. המסך הגרפי נוצר עם יצירתו של צב בפעם הראשונה. לצב יש יכולת תנועה. הוא יכול להסתובב סביב עצמו ימינה ושמאלה בכל מעלה שלמה ולפנות לכיוונים שונים. צב חדש נוצר כשהוא פונה כלפי מעלה. צב יכול גם לנוע קדימה אל הכיוון אליו הוא פונה לכל מרחק נתון.
לצב יש זנב. הזנב יכול להיות מורם או מורד. אם הזנב מורד והצב נע קדימה הצב משאיר לאורך מסלולו עקבות בצורה של קו על המסך הגרפי. אם הזנב מורם הצב לא משאיר עקבות. צב גם ניתן להסתרה וגילוי מחדש. בין אם הצב גלוי ובין אם הוא מוסתר הוא מבצע את כל הפעולות באותו האופן. כלומר, הוא יכול להסתובב, להתקדם קדימה, להשאיר עקבות כאשר זנבו מורד, או לא להשאיר סימן כאשר זנבו מורם.
לממשק הציבורי של המחלקה SimpleTurtle המאפיינים והיכולות הבאים:

```
SimpleTurtle (); // construct simple turtle
public void show (); // show yourself
public void hide (); // hide yourself
public void tailDown (); // lower the tail
public void tailUp (); // lift the tail
public void turnLeft (int degrees); // turn left in the given degrees
public void turnRight (int degrees); // turn right in the given degrees
public void moveForward (double distance); // advance forward in the given distance
public void moveBackward (double distance); // go backward in the given distance
```

כדי להשתמש במחלקה SimpleTurtle יש להוריד מאתר הקורס אל המחשב שלכם את החבילה **Turtle.jar**, שמאגדת בתוכה מספר קבצים הנדרשים לעבודה עם הצב, יש להגדיר פרויקט ולייבא (`import`) את החבילה כך שנוכל להשתמש בה בפרויקט (ראה תמונה), דוגמא פשוטה מאוד לקובץ main שמשתמש בחבילה ניתן למצוא בקובץ: **TestTurtle.java**.
לכתיבת החלק הראשון עקבו אחרי המשימות הבאות.

משימה ראשונה – הגדרה של צבים עם תכונות שונות ע"י הורשה

לא כל הצבים נוצרו שווים. יש חכמים, יש מוכשרים, ולא עלינו, מוזרים. הוסיפו מחלקות שיגדירו את הצבים הבאים.

- ***צב חכם*** (SmartTurtle) – צב חכם, מעבר להיותו צב רגיל לכל דבר ועניין, מבין גם משהו בגיאומטריה: הוא יודע לצייר ריבוע באורך נתון ומצולע משוכלל בעל מספר צלעות נתון באורך נתון. כתבו את המחלקה SmartTurtle, שימרו אותה בקובץ בשם SmartTurtle.java במחיצת העבודה, והוסיפו לה את שתי השיטות הבאות:

```
public void drawSquare (double size);  //draw a square in the given size
public void drawPolygon (int sides, double size);
// draw a polygon in the given sides and size
```

שימו לב: כיוון שהזויות העוברת כפרמטר בשיטות turnLeft ו- turnRight היא זווית שלמה עלולה להתעורר בעיה במקרה של פוליגון משוכלל עם זווית לא שלמה. *התעלמו מהבעיה*. דאגו רק שהצב החכם יצייר נכון פוליגונים בעלי זווית שלמה.

- ***צב שיכור*** (DrunkTurtle) – צב שיכור הוא צב רגיל ששתה מעט וכתוצאה מכך קשה לו קצת ללכת. כשהוא מתבקש לנוע קדימה למרחק X הוא מבצע את הפעולות הבאות:
  - הוא מתקדם למרחק מקרי בין 0 ל- X2.
  - פונה בזווית מקרית בין 30+ ל- 30- מעלות.

  כתבו את המחלקה DrunkTurtle ושימרו אותה בקובץ בשם DrunkTurtle.java במחיצת העבודה. שנו בה את הדרוש שינוי.

- ***צב מקרטע*** (JumpyTurtle) – צב מקרטע הוא צב חכם מקרטע: כאשר הוא מתקדם הוא הולך ומנתר לסירוגין. התוצאה היא שכאשר זנבו מורד הוא משאיר קו מקווקו. כתבו את המחלקה JumpyTurtle ושימרו אותה בקובץ בשם JumpyTurtle.java במחיצת העבודה שלכם. שנו בה את הדרוש שינוי.

שימו ♥: כיוון שצב מתקדם על פני סריג של נקודות, הצב לא יכול להתקדם תמיד למרחק הנדרש בדיוק נמרץ (למה?). התקדמות למרחק קצר עלולה ליצור אי דיוק גדול יחסית (למה?). יוצא איפה שאם ההתקדמות של צב מקרטע תשבר לרצף ארוך של פסיעות ודילוגם קטנים, אי הדיוק יצטבר והצב עלול להתקדם בפועל למרחק שונה מהנדרש. התעלמו מהבעיה. שיברו את המרחק אליו מתבקש הצב המקרטע להתקדם למספר פסיעות וניתורים קטן, והניחו תמיד שהצב יידרש להתקדם למרחקים גדולים.

חשוב ביותר:
על כל הצבים מכל הסוגים לא להשאיר מאחוריהם עקבות כאשר הזנב שלהם מורם.

משימה שנייה – ניהול צבא של צבים באמצעות פולימורפיזים

כתבו תכנית בשם Army.java שתנהל צבא (מערך) של 5 צבים.

- בשלב ראשון אפשרו למשתמש לבחור את צבא הצבים כרצונו. הציגו לפניו את התפריט שלמטה וקבלו את בחירתו עבור כל אחד מחמשת הצבים בנפרד. אפשרו לו לבחור כל תערובת של צבים. להלן התפריט:

Choose the type of a turtle:

1. Simple
2. Smart
3. Drunk
4. Jumpy

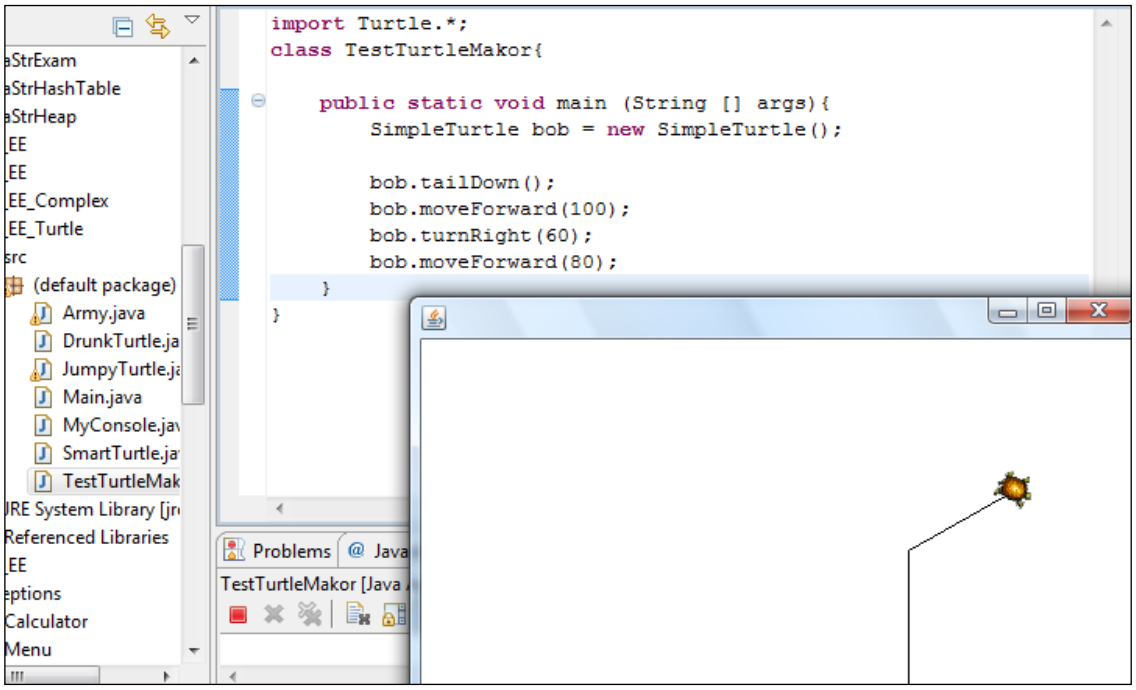
- בשלב שני צרו את הצבים הנדרשים וקדמו אותם יחד שלב אחר שלב על פני השלבים הבאים:

1. הורדת זנב.
2. צעידה קדימה למרחק של 100.
3. פניה של 90 מעלות ימינה
4. צעידה קדימה למרחק של 60
5. לכל מי שיודע לצייר מצולע, ציור של מצולע בן 6 צלעות באורך של 70.
6. העלמות.

שימו ♥ על הצבים להתקדם כולם יחד. אף צב לא יכול לעבור לשלב גבוה יותר בטרם גמרו כל הצבים האחרים את השלב הקודם.

הריצו את התוכנית ובדקו אותה. הסבירו בתיעוד את השימוש בפולימורפיזם.
הסבירו את השימוש ב- casting.

שימו ♥, אל תשתמשו ב- casting אלא אם כן הוא הכרחי.



חלק ב טיפול בחריגות

בחלק זה על המשתמש יש לפתור משוואה ריבועית: $ax^2 + bx + c = 0$.

א) יש להזין שלושה מספרים ממשיים (a,b,c), שמייצגים את מקדמי משוואה ריבועית.

ב) יש לבדוק האם הפתרון קיים, כלומר קיימים שני מספרים ממשיים (לא בהכרח שונים) שמהווים את פתרון המשוואה.

ג) אם יש שני פתרונות שונים (ממשיים) , יש להדפיס: x_1=..., x_2=...

אם קיים פתרון יחיד למשוואה, יש להדפיס: x_1=x_2=...

ד) כאשר למשוואה אין פתרון יש לטפל במצב זה ולזרוק חריגה מטיפוס SquareEquationException (חריגה של המשתמש) עם הודעה מתאימה:

a. כאשר ביטוי בתוך השורש שלילי יש להדפיס:

Error: NO real roots!

b. כאשר a=0, b=0, c=0 יש להדפיס:

x can be any number - trivial!

c. כאשר a=0, b=0, c≠0 יש להדפיס:

Error, no answer!!

מעטפת התוכנית. עליכם לכתוב את המסגרת שתציג למשתמש תפריט לבחירתו:

0 – יציאה מהתוכנית (או כל מספר אחר ששונה מ 1).

1 – לקליטת נתונים וחישוב פתרונות של משוואה ריבועית.

לאחר סיום החישובים וטיפול בחריגות על התוכנית לחזור ולהציג את התפריט ההתחלתי.

תוצאות ההרצה:

```
aX^2+bX+c=0: Enter a,b,c:
Enter a: -2.3
Enter b: 5.1
Enter c: -12.62
-2.3X^2+5.1X+-12.62=0:
SquareEquationException: Error: NO real roots!
Enter 0 or any number to Exit or 1 to solve aX^2+bX+c=0:
at matala2B.Exceptions.SquareEquation.sqEq(SquareEquation.java:28)
at matala2B.Exceptions.SquareEquation.sqEquation(SquareEquation.java:44)
Exceptions.SquareEquation.main(SquareEquation.java:55)
1
aX^2+bX+c=0: Enter a,b,c:
Enter a: -2.3
Enter b: 5.1
Enter c: 12.98
-2.3X^2+5.1X+12.98=0:
x1:-1.5128848463076623    x2:3.730276150655489
Enter 0 or any number to Exit or 1 to solve aX^2+bX+c=0:
1
aX^2+bX+c=0: Enter a,b,c:
Enter a: 1
Enter b: -5
Enter c: 6
1.0X^2+-5.0X+6.0=0:
x1:3.0    x2:2.0
Enter 0 or any number to Exit or 1 to solve aX^2+bX+c=0:
```

```
1
aX^2+bX+c=0: Enter a,b,c:
Enter a: 1
Enter b: -2
Enter c: 1
1.0X^2+-2.0X+1.0=0:
x1=x2:1.0
Enter 0 or any number to Exit or 1 to solve aX^2+bX+c=0:
1
aX^2+bX+c=0: Enter a,b,c:
Enter a: 0
Enter b: 2
Enter c: 5
0.0X^2+2.0X+5.0=0:
x1=:-2.5
Enter 0 or any number to Exit or 1 to solve aX^2+bX+c=0:
1
aX^2+bX+c=0: Enter a,b,c:
Enter a: 0
Enter b: 0
Enter c: 0
0.0X^2+0.0X+0.0=0:
Enter 0 or any number to Exit or 1 to solve aX^2+bX+c=0:
matala_2Exceptions.SquareEquationException: x1 can be any number - trivial!
	at matala_2Exceptions.SquareEquation.sqEq(SquareEquation.java:17)
	at matala_2Exceptions.SquareEquation.sqEquation(SquareEquation.java:44)
	at matala_2Exceptions.SquareEquation.main(SquareEquation.java:55)
1
aX^2+bX+c=0: Enter a,b,c:
Enter a: 0
Enter b: 0
Enter c: 3
0.0X^2+0.0X+3.0=0:
Enter 0 or any number to Exit or 1 to solve aX^2+bX+c=0:
matala_2Exceptions.SquareEquationException: Error, no answer!!
	at matala_2Exceptions.SquareEquation.sqEq(SquareEquation.java:16)
	at matala_2Exceptions.SquareEquation.sqEquation(SquareEquation.java:44)
	at matala_2Exceptions.SquareEquation.main(SquareEquation.java:55)
1
aX^2+bX+c=0: Enter a,b,c:
Enter a: 1
Enter b: 0
Enter c: 0
1.0X^2+0.0X+0.0=0:
x1=x2:0.0
Enter 0 or any number to Exit or 1 to solve aX^2+bX+c=0:
0
Bye-bye!
```

חשוב ביותר:

יש לבצע את כל הבדיקות באמצעות כלי JUNIT.

חלק ג שימוש ב- ListIterator

בחלק זה יש לכתוב מחלקה המייצגת **רשימה מקושרת גנרית דו-כיוונית**.

1) יש להגדיר קדקוד של הרשימה **NodeDouble** כמחלקה גנרית פנימית private.
   - ♦ במחקה זו יש לכתוב בנאי שמקבל רק data ומתודה toString.

2) יש להגדיר מחלקה גנרית **LinkedListDouble** המייצגת רשימה מקושרת דו-כיוונית.
   במחלקה זו יש לממש מתודות הבאות:
   - ♦ בנאי רגיל ללא ארגומנטים.
   - ♦ מתודת הוספת איבר: add(T item){…}
   - ♦ toString() - מתודה שמייצגת את הרשימה.

בתוך מחלקת LinkedListDouble יש להגדיר מתודה בשם listIterator()
המחזירה אובייקט חדש מטיפוס ListIterator.
בתוך מתודה זו יש להגדיר מחלקה פנימית אנונימית גנרית המייצגת ListIterator.
בתוך מחלקה פנימית אנונימית יש לממש מתודות:

```
T next()// returns the next element in the list and advances the cursor position.
Throws NoSuchElementException - if the iteration has no next element

boolean hasNext()//Returns true if this list iterator has more elements when
                 // traversing the list in the forward direction.

T previous()// Returns the previous element in the list and moves the
            //  cursor position backwards.
Throws NoSuchElementException - if the iteration has no previous element


boolean hasPrevious()//Returns true if this list iterator has more elements
                     //when traversing the list in the reverse direction.
```

שימו ♥ אין צורך לממש את שאר הפונקציות שמוגדרות בממשק ListIterator -
הן יכולים להיראות כך:

```
@Override
public void remove() {
     throw new UnsupportedOperationException();
}
```

3) **חשוב ביותר:**

יש לבצע את כל הבדיקות באמצעות כלי JUNIT.

תוצאות הרצה, כאשר הרשימה מכילה מחרוזות {"a", "b", "c", "d", "e"}:

```
direct walk:
a, b, c, d, e,

reverse walk:
e, d, c, b, a,
```

בהצלחה רבה!!!