תכנות מונחה עצמים – מטלה 1- מחלקות הורשה

חלק 1 – מחלקת שבר (Fraction)

בתרגיל זה נזכור את עבודה עם מחלקות ואובייקטים. עליך לכתוב מחלקה **Fraction** המהווה שבר פשוט. השבר הפשוט שבנוי באמצעות אחד מבנאים המחלקה צריך להיות מצומצם (כדאי להשתמש בפונקציה gcd(int p, int q) שמחשבת ומחזירה המחלק המשותף הגדול ביותר של מספרים p and q. המחלקה מכילה פונקציות הבאות:

Matala0

Class Fraction

```
java.lang.Object
  └Matala0.Fraction
```

```
public class Fraction
extends java.lang.Object
```

Constructor Summary

Fraction()	Default Constructor fraction: 0/1
Fraction(Fraction f)	Copy Constructor
Fraction(int n, int d)	Constructor

Method Summary

void	**add**(Fraction f) Addition of two fractions
int	**compare**(Fraction f) Compares its two arguments for order.
void	**dev**(Fraction f) division of two fractions
boolean	**equals**(Fraction f)
void	**mul**(Fraction f) multiplication of two fractions
void	**sub**(Fraction f) Subtraction of two fractions
java.lang.String	**toString**() Returns a string representation of the the fraction: (n/d)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Fraction

```
public Fraction(int n,
                int d)
```

Constructor

Parameters:
`n` - - numerator
`d` - - denominator

Fraction

`public` **`Fraction`**`()`
Default Constructor fraction: 0/1

Fraction

`public` **`Fraction`**`(Fraction f)`
Copy Constructor
Parameters:
`f` - - fraction

Method Detail

add

`public void` **`add`**`(Fraction f)`
Addition of two fractions
Parameters:
`f` - - fraction

---

sub

`public void` **`sub`**`(Fraction f)`
Subtraction of two fractions
Parameters:
`f` - - fraction

---

mul

`public void` **`mul`**`(Fraction f)`
multiplication of two fractions
Parameters:
`f` - - fraction

`public void` **`dev`**`(Fraction f)`
division of two fractions
Parameters:
`f` - - fraction

---

equals

`public boolean **equals**(Fraction f)`

Parameters:
`f` - - fraction
Returns:
true if the fractions are equals otherwise return false

---

compare

`public int **compare**(Fraction f)`

Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.
Parameters:
`f` - - fraction
Returns:
-1, 0 or 1

---

toString

`public java.lang.String **toString**()`

Returns a string representation of the the fraction: (n/d)
Overrides:
`toString` in class `java.lang.Object`

חלק 2 – מחלקת קבוצת שברים.

המחלקה מהווה קבוצה של שברים פשוטים. הקבוצה אינה מכילה איברים כפולים. המחלקה מכילה שיטות הבאות:

Class FractionSet

```
java.lang.Object
  └Matala0.FractionSet
```

`public class **FractionSet**`
`extends java.lang.Object`

class, that represents set of fractions, cannot contain duplicate elements data members: set - array of fractions, size - number of fractions in the array

---

Constructor Summary

FractionSet() default constructor initial size of the array is 10
FractionSet(FractionSet other) copy constructor
FractionSet(int size) constructor

Method Summary

void	**add**(Fraction elem) Appends the specified fraction (elem) to the end of this set

boolean	**containAll**(FractionSet other) check if this set contains all the elements of the specified (other) set
int	**contains**(Fraction f)
void	**difference**(FractionSet other) Removes from this set all of its elements that are contained in the specified (other) set
boolean	**equals**(FractionSet other) Compares the specified set (other) with this set for equality.
void	**intersection**(FractionSet other) Retains only the elements in this set that are contained in the specified (other) set
void	**remove**(Fraction elem) Removes the specified fraction (elem) from this set
int	**size**()
java.lang.String	**toString**() Returns a string representation of the set
void	**union**(FractionSet other) Adds all of the elements in the specified collection (other) to this set
void	**xor**(FractionSet other) creates the result of the XOR operation of this and other sets

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

FractionSet

public **FractionSet**()

default constructor initial size of the array is 10

FractionSet

public **FractionSet**(int size)

constructor

Parameters:

size - - the initial size of the array

FractionSet

public **FractionSet**(FractionSet other)

copy constructor

Parameters:

other - - the Fraction

Method Detail

contains

public int **contains**(Fraction f)

Parameters:

Fraction - - f

Returns:
index of f if the set contains f, otherwise return -1

---

add

```
public void add(Fraction elem)
```

Appends the specified fraction (elem) to the end of this set

Parameters:

`elem` -

---

remove

```
public void remove(Fraction elem)
```

Removes the specified fraction (elem) from this set

Parameters:

`elem` -

---

union

```
public void union(FractionSet other)
```

Adds all of the elements in the specified collection (other) to this set

Parameters:

`other` -

---

difference

```
public void difference(FractionSet other)
```

Removes from this set all of its elements that are contained in the specified (other) set

Parameters:

`other` -

---

intersection

```
public void intersection(FractionSet other)
```

Retains only the elements in this set that are contained in the specified (other) set

Parameters:

`other` - - FractionSet

---

containAll

```
public boolean containAll(FractionSet other)
```

check if this set contains all the elements of the specified (other) set

Parameters:

`other` - - the specified set

Returns:

true if this set contains all the elements of the specified (other) set, otherwise returns false

---

equals

`public boolean **equals**(FractionSet other)`

Compares the specified set (other) with this set for equality.

Parameters:

`other` -

Returns:

true if the two sets have the same size, and the two sets consist the same elements (the order is not significant) otherwise returns false

---

xor

`public void **xor**(FractionSet other)`

creates the result of the XOR operation of this and other sets

Parameters:

`other` - - FractionSet

---

toString

`public java.lang.String **toString**()`

Returns a string representation of the the set

Overrides:

`toString` in class `java.lang.Object`

---

size

`public int **size**()`

Returns:

number of the fractions in this set

חלק 3 – בדיקה. בחלק זה מצורפת תכנית בדיקה של המחלקות:

```
import java.util.Random;
public class TestFractionSet {
	public static void timeTest(){
		FactionSet fs1 = new FactionSet();
		Random gen = new Random(123);
		int len = 10000, to = 1000;
		for (int i=0; i<len; i++) {
			fs1.add(new Fraction(gen.nextInt(to)+1, gen.nextInt(to)+1));
		}
		System.out.println("size fs1: "+fs1.size());
		FactionSet fs2 = new FactionSet();
		for (int i=0; i<len; i++) {
			fs2.add(new Fraction(gen.nextInt(to)+1, gen.nextInt(to)+1));
		}
		System.out.println("size: fs2: "+fs2.size());
		fs2.xor(fs1);
		System.out.println("fs2 xor fs1 size: "+fs2.size());
	}
```

```java
    public static void smallTest(){
        FactionSet t1 = new FactionSet();
        t1.add(new Fraction(1,2));
        t1.add(new Fraction(1,3));
        t1.add(new Fraction(2,4));
        t1.add(new Fraction(1,4));
        t1.add(new Fraction(2,8));
        t1.add(new Fraction(1,5));
        t1.add(new Fraction(1,6));
        System.out.println("t1: "+t1);
        FactionSet t2 = new FactionSet();
        t2.add(new Fraction(5,2));
        t2.add(new Fraction(1,3));
        t2.add(new Fraction(2,4));
        t2.add(new Fraction(7,4));
        t2.add(new Fraction(3,7));
        t2.add(new Fraction(3,5));
        t2.add(new Fraction(5,6));
        System.out.println("t2: "+t2);
        System.out.println("t1 EQUALS t2: "+t1.equals(t2));
        System.out.println("t1 EQUALS t1: "+t1.equals(t1));
        FactionSet t3 = new FactionSet(t1);
        FactionSet t4 = new FactionSet(t2);
        System.out.println("t3: "+t3);
        System.out.println("t4: "+t4);
        t3.union(t2);
        System.out.println("t3 UNION t2: "+t3);
        System.out.println("t3 contains t2: "+t3.containAll(t2));
        System.out.println("t2 contains t3: "+t2.containAll(t3));
        t3 = new FactionSet(t1);
        t1.intersection(t2);
        System.out.println("t1 INTERSECRION t2: "+t1);
        System.out.println("t3: "+t3);
        System.out.println("t4: "+t4);
        t3.xor(t4);
        System.out.println("t3 xor t4 : "+t3);
    }

    public static void main(String[] args) {
        System.out.println("**************  smallTest ************************");
        smallTest();
        System.out.println("\n**************  timeTest ************************");
        long start = System.currentTimeMillis();
        timeTest();
        long end = System.currentTimeMillis();
        System.out.println("time: "+(end-start)+" msc");
    }
}
```

תוצאות הבדיקה:

```
**************  smallTest ************************
t1: [(1/2), (1/3), (1/4), (1/5), (1/6)]
t2: [(5/2), (1/3), (1/2), (7/4), (3/7), (3/5), (5/6)]
t1 EQUALS t2: false
t1 EQUALS t1: true
t3: [(1/2), (1/3), (1/4), (1/5), (1/6)]
t4: [(5/2), (1/3), (1/2), (7/4), (3/7), (3/5), (5/6)]
t3 UNION t2: [(1/2), (1/3), (1/4), (1/5), (1/6), (5/2), (7/4), (3/7), (3/5), (5/6)]
t3 contains t2: true
t2 contains t3: false
t1 INTERSECRION t2: [(1/3), (1/2)]
t3: [(1/2), (1/3), (1/4), (1/5), (1/6)]
t4: [(5/2), (1/3), (1/2), (7/4), (3/7), (3/5), (5/6)]
t3 xor t4 : [(1/4), (1/5), (1/6), (5/2), (7/4), (3/7), (3/5), (5/6)]

**************  timeTest ************************
size fs1: 9702
size: fs2: 9740
fs2 xor fs1 size: 18524
time: 1787 msc
```

חלק 3 – ירושה INHERITANCE

בתרגיל זה נתאמן בשימוש בירושה, את התרגיל ניתן להגיש בזוגות (או לבד).

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

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 לב: 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 מאחריהם עקבות כאשר הזנב שלהם מורם.

משימה שנייה – ניהול צבא של צבים באמצעות פולימורפיזים

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

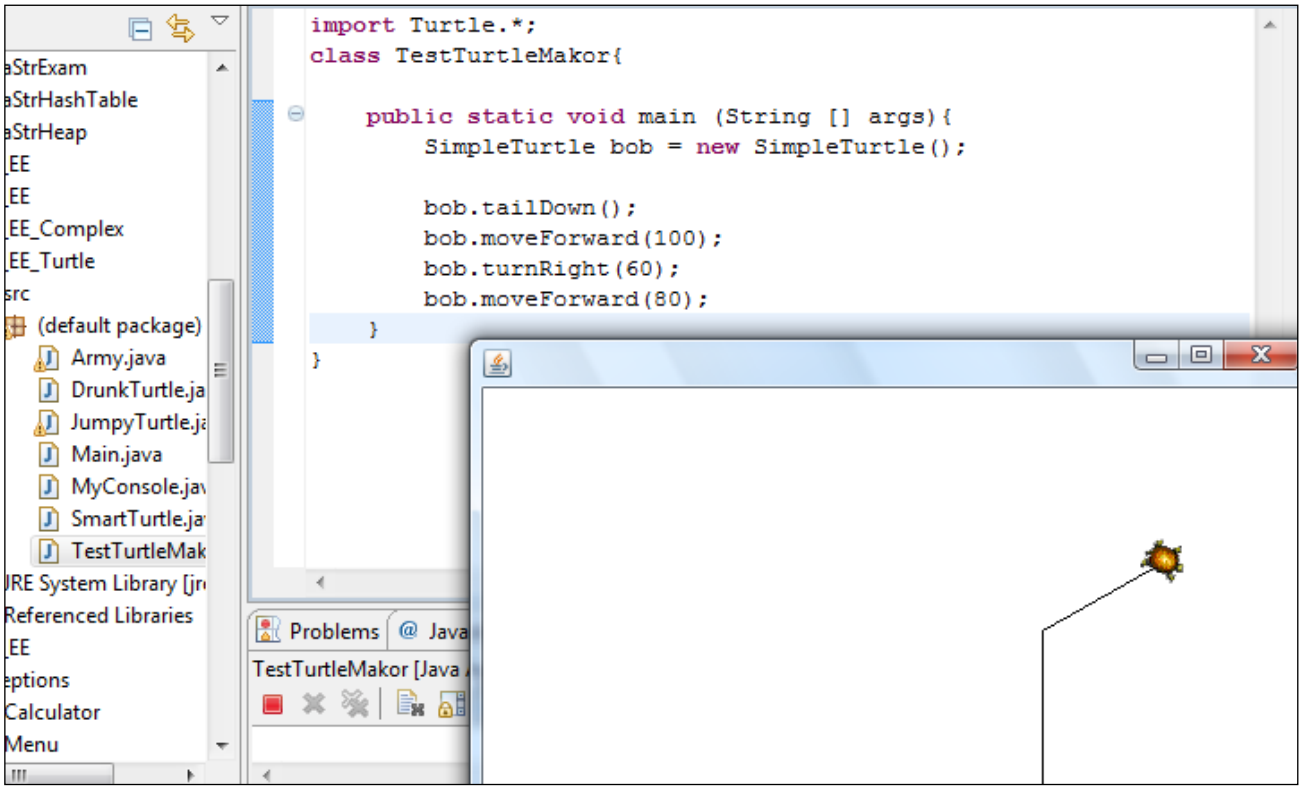
- בשלב ראשון אפשרו למשתמש לבחור את צבא הצבים כרצונו. הציגו לפניו את התפריט שלמטה וקבלו את בחירתו עבור כל אחד מחמשת הצבים בנפרד. אפשרו לו לבחור כל תערובת של צבים. להלן התפריט:

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

תיהנו !