תרגיל 1

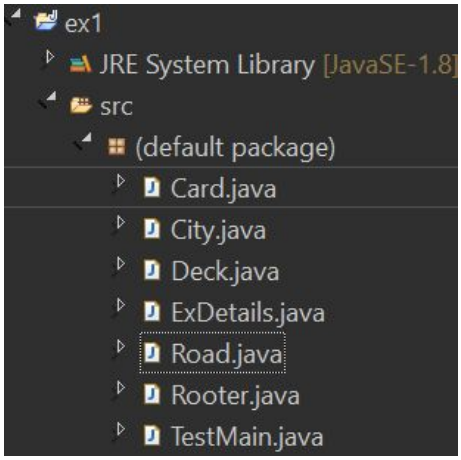
תכנות מונחה עצמים

תאריך הגשה: 12.4

נא לקרוא את דף ההנחיות להגשת תרגילים
אתם נדרשים להגיש את הקבצים הבאים ולא יותר או פחות מזה: Rooter.java, Road.java, City.java, Card.java, Deck.java, ExDetails.java.

הנה הקובץ TestMain.java, לפני ההגשה ודאו שהוא מתקמפל יחד עם הקבצים שלכם בדיוק כמו שהוא, ובריצה שלו כותב שעברתם את כל הבדיקות. אם זה עובד, כבר אתם במצב מצוין.

כך אמורה להראות התיקייה שלכם בתוך אקליפס, כאשר את TestMain.java אפשר למחוק לפני הגשה, ושם הפרויקט (ex1) לא חשוב:



שאלה 1

כתבו מחלקה בשם Rooter עם השיטות הבאות:

מאתחל את המחלקה עם רמת דיוק מסוימת.	public Rooter(double precision)
קובע את רמת הדיוק.	public void setPrecision(double precision)
מחשב את השורש הריבועי של x,	public double sqrt(double x)

כשהתשובה מדויקת עד כדי precision.	

כדי לממש זאת, למחלקה יהיה משתנה פרטי מסוג double שנקרא precision.

ניתן כמובן לחשב שורש של מספר בעזרת Math.sqrt, אבל פה השתמשו באלגוריתם הפשוט הבא לחישוב השורש של מספר x:

1. מתחילים ממועמד כלשהו one, שערכו ההתחלתי הוא x/2.
2. מחשבים מועמד שני: two = x / one.
   a. שימו לב, שכאשר one שווה ל-two, אז one הוא בדיוק השורש של x, וניתן להחזיר אותו.
   b. אחרת, אחד המועמדים גדול מ-x והשני קטן ממנו.
3. אם המועמדים מספיק קרובים אחד לשני (ההפרש קטן מ-precision) אז אפשר להחזיר אחד מהם וסיימנו.
4. אחרת, קבעו את one כממוצע של שני המועמדים, וחזרו לשלב 2.

למשל:

```
Rooter s = new Rooter(0.1);
System.out.println(s.sqrt(2));
s.setPrecision(0.00001);
System.out.println(s.sqrt(2));
```

ידפיס משהו כמו (לאו דווקא בדיוק):
1.4166666666666665
1.4142156862745097

שאלה 2

נתאר רשת של ערים ושל הכבישים הבין עירוניים ביניהן. יהיו לנו שתי מחלקות: City ו-Road.

לכל עיר יש שם, ורשימת כבישים אליהם היא מחוברת.

מערך המכיל את הכבישים המחוברים לעיר. ניתן להניח שעיר לא תהיה מחוברת לעולם ליותר מעשרה כבישים.	private Road[] roads

private int numRoads	מספר הכבישים שמחוברים לעיר עד כה.
public City(String name)	בנאי המייצר עיר עם השם הנתון.
public void connect(Road r)	מוסיף את הכביש לרשימת הכבישים של העיר.
public City nearestCity()	מחזיר את העיר שהכי קרובה לעיר הזאת או null אם אין אף עיר מחוברת.
Public String toString()	מחזיר את שם העיר.

לכל כביש יש את שתי הערים שהוא מחבר ביניהן, ואת אורכו כמספר שלם של קילומטרים.

public Road(City city1, City city2, int length)	בנאי שמאתחל את הכביש כרגיל, אבל בנוסף גם מוסיף את הכביש שנוצר לשתי הערים city1 ו-city2. הוא יעשה זאת בעזרת השיטה connect של City.
public City getCity1()	מחזיר את עיר הראשונה.
public City getCity2()	מחזיר את העיר השנייה.
public int getLength()	מחזיר את אורך הכביש.

שימו לב: בשאלה זו תצטרכו להשתמש ב-this כדי לקבל את המצביע לאוביקט שאתם כרגע בתוכו.

למשל, אם יש לנו שיטת main שבה הקוד הבא:

```
City karmiel = new City("Karmiel");
City metula = new City("Metula");
City telAviv = new City("Tel-Aviv");
City jerusalem = new City("Jerusalem");

new Road(karmiel, metula, 50);
new Road(karmiel, telAviv, 100);
new Road(telAviv, jerusalem, 80);
new Road(jerusalem, metula, 175);

System.out.println(karmiel.nearestCity());
```

אז יודפס: Metula.

שימו לב לנקודה מעניינת בקוד, וזה שאנחנו יוצרים אובייקט מסוג Road אבל כלל לא שומרים אותו במשתנה מקומי. זה כי הבנאי ישמור אותו בתוך רשימות הכבישים של הערים, ושם חשוב לנו שהוא יהיה. פה אין לנו בו צורך.

הערה: כמו שנכתב, לעיר יכולים להיות לכל היותר 10 כבישים. לכן, אין צורך להגדיל את מערך הכבישים בכל פעם שמוסיפים לה כביש. עדיף להקצות מראש מקום לעשרה כבישים, ולשמור את מספר הכבישים שיש כרגע כעוד משתנה. זה יחסוך את עלות ההקצאה המחודשת והעתקת כל הכבישים. דוגמא כזאת אפשר לראות בהרצאה, במימוש של StringStack.

שאלה 3

נבנה מחלקה המתארת קלף משחק, ומחלקה המתארת חפיסת קלפים.

הקלפים שלנו יהיו בנויים ממספר טבעי כלשהו, וסוג (suit), כאשר הסוג הוא גם מספר בין 0 ל-3, כש: 0 הוא clubs (תלתן), 1 הוא diamonds (יהלום), 2 הוא hearts (לב), ו-3 הוא spades (עלה).

מחלקה ראשונה תקרא Card ותייצג קלף אחד כזה. יהיו לה השיטות:

בנאי רגיל.	public Card(int num, int suit)
מחזירה את המספר של הקלף.	public int getNum()
מחזירה את סוג הקלף.	public int getSuit()
תחזיר את המספר צמוד לאות הראשונה של סוג הקלף (כאות גדולה). למשל 5 לב יחזיר 5H.	public String toString()
בהינתן קלף אחר, תחזיר 0 אם שני הקלפים שקולים לחלוטין (אותו מספר וסוג). תחזיר מספר חיובי כלשהו אם הקלף גדול יותר מאשר הקלף other, ומספר שלילי אם להיפך.	public int compareTo(Card other)

הסדר בין קלפים מוגדר כך שקודם כל המספר קובע, ואם שני המספרים זהים, אז הסוג קובע, לפי המספור של הסוגים שמוזכר למעלה.

המחלקה Deck מתארת חפיסת קלפים. יהיה לה בעצם מערך של Card, וכן את מספר הקלפים שיש בו - ויש לה מגוון בנאים ופעולות:

public Deck(int num)	מייצרת חפיסה שבה לכל מספר מ-0 עד num-1 יש את כל הסוגים. לפי סדר עולה. למשל, אם num=2, היא תהיה: [0C, 0D, 0H, 0S, 1C, 1D, 1H, 1S]
public Deck(Deck from, int num)	מייצרת חפיסה ע"י לקיחת קלף אחרי קלף מהחפיסה from. סה"כ לוקחת num קלפים (אם אין מספיק, אז כמספר הקלפים ב-from).
public Deck(Deck first, Deck second)	מייצרת חפיסה ע"י שילוב החפיסות. לוקחת קלף מהראשונה, ואז מהשנייה, וחוזר חלילה. שימו לב ששתי החפיסות מתרוקנות לגמרי בסוף השיטה.
public int getNumCards()	מחזירה את מספר הקלפים בחפיסה.
public Card takeOne()	לוקחת קלף אחד מסוף החפיסה, מורידה אותו מהחפיסה ומחזירה אותו. אם אין כזה, מחזירה null.
public String toString()	מחזירה מחרוזת מהצורה: "[2C, 7D, 13H, 1S]"
public void sort()	ממיינת את החפיסה לפי סדר עולה. אפשר לממש בעזרת מיון לא יעיל (כמו בועות).

למשל, הרצת הקוד הבא:

```
Deck d1 = new Deck(3);
System.out.println(d1);

Deck d2 = new Deck(d1, 4);
System.out.println(d2);

Deck d3 = new Deck(d1, d2);
System.out.println(d1);
System.out.println(d2);
System.out.println(d3);

d3.sort();
System.out.println(d3);
```

תיתן:

```
[0C, 0D, 0H, 0S, 1C, 1D, 1H, 1S, 2C, 2D, 2H, 2S]
[2S, 2H, 2D, 2C]
[]
[]
[1S, 2C, 1H, 2D, 1D, 2H, 1C, 2S, 0S, 0H, 0D, 0C]
[0C, 0D, 0H, 0S, 1C, 1D, 1H, 1S, 2C, 2D, 2H, 2S]
```

הנה דוגמא לשימוש במחלקה הזאת כדי לערבב כמו שמערבבים במציאות. אפשר היה לשפר זאת ע"י הוספת אקראיות בחלוקת החפיסות:

```
Deck d = new Deck(5);
System.out.println(d);
for (int i = 0; i < 3; i++) {
    Deck d2 = new Deck(d, d.getNumCards()/2);
    Deck d3 = new Deck(d, d2);
    d = d3;
}
System.out.println(d);
```

זה אמור להדפיס:

```
[0C, 0D, 0H, 0S, 1C, 1D, 1H, 1S, 2C, 2D, 2H, 2S, 3C, 3D, 3H, 3S, 4C, 4D, 4H, 4S]
[0H, 1S, 3C, 4D, 4C, 2S, 1H, 0D, 0S, 2C, 3D, 4H, 3S, 2H, 1D, 0C, 1C, 2D, 3H, 4S]
```

הערות:

שימו לב להנחייה הבאה מתוך קובץ ההנחיות לשיעורי הבית:

אל תוסיפו שיטות ומשתנים שאינם פרטיים לאלו שהתבקשתם בתרגיל, חוץ מבתרגילים בהם אנחנו נותנים לכם את החופש הזה. לעומת זאת, פרטיים אתם יכולים להוסיף כמה שאתם רוצים.

וזה אומר למשל שלא ניתן לשנות ערכים בתוך קלף (כי אין שיטות set שם). ועוד משם:

הימנעו כמה שיותר משיכפול קוד. השתמשו בשיטות פרטיות, בירושה (כשנלמד), ובתכנות חכם באופן כללי.

לדוגמא: השתמשו ב-compareTo של קלפים כדי להשוות ביניהם כשאתם ממיינים, השתמשו ב-takeOne כדי להעביר קלפים מראש חפיסה לחפיסה אחרת, השתמשו ב-toString של קלף בקוד של toString של חפיסה.

בנוסף: בכל מקום שאתם מעבירים קלף ממקום למקום - אין שום צורך לייצר קלפים חדשים! צריך פשוט להעתיק פוינטרים. ספציפית ב-takeOne למשל, וכך גם בבנאים השני והשלישי.

ועוד: אין שום הגבלה על גודל החפיסות (למשל לא 52), צרו את המערך לפי הגודל שאתם צריכים בבנאי (כל בנאי יקצה מקום לפי כמה שהוא צריך).