תרגיל 1

תכנות מונחה עצמים

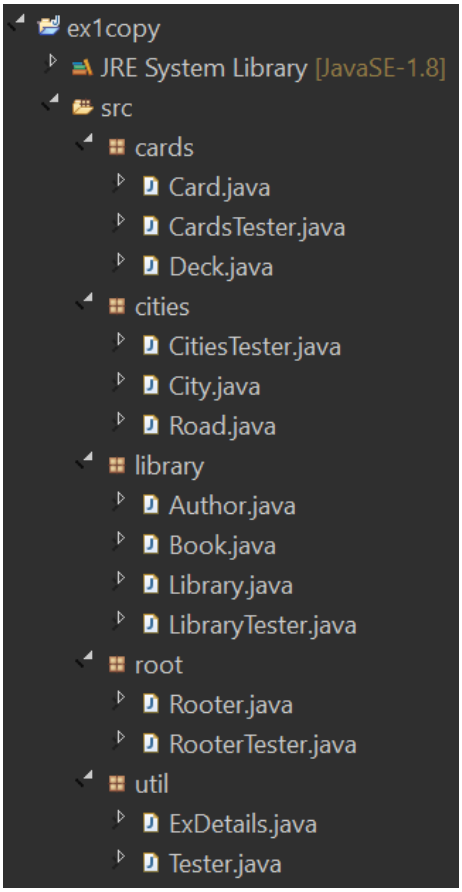
בתרגול הראשון מופיעות הרבה הנחיות טכניות של עבודה עם אקליפס שיכולות לעזור לכם בגישה ראשונית לתרגיל הזה. היעזרו בו.

נא לקרוא את דף ההנחיות להגשת תרגילים

הנה קובצי הבדיקה המתאימים לכל שאלה:

1. RooterTester.java,
2. LibraryTester.java,
3. CitiesTester.java,
4. CardsTester.java.

בסופו של דבר, הפרויקט שלכם באקליפס צריך להיראות כמו בתמונה פה (שם הפרויקט לא חשוב). לשם כך:



1. עליכם קודם כל להוריד את שני הקבצים Tester.java ו-ExDetails.java ולשים אותם בחבילה בשם util (הקישורים בדף ההנחיות)
2. התחילו מלהוריד את קובץ הבדיקות של השאלה הראשונה, ושימו אותו בחבילה המתאימה (root במקרה הזה).
3. פתרו את השאלה הראשונה ע"י כתיבת המחלקה Rooter (המשויכת לקובץ Rooter.java), שתהיה גם היא באותה חבילה. תוכלו להריץ את ה-main של קובץ הבדיקות שלה ולראות שהוא מתקמפל והטסטים עוברים.
4. המשיכו כך לשאלות הבאות, והקפידו על שם חבילה מדויק לכל שאלה.
5. לא צריך להתעסק עם בדיקות קלט ונכונות מעבר למה שמצוין בתיאור התרגיל.

חשוב מאוד!

1. לפני ההגשה, אף קובץ פה לא יהיה עם סימן אדום - זה אומר שיש בעיית קומפילציה, ולא נוכל לבדוק אותו (ולא תקבלו עליו בכלל נקודות)

2. **שנו את הקובץ ExDetails.java** כך שיכיל את הפרטים האישיים שלכם (ולא השארתם שם את אריק ובנץ). אם הגשתם לבד, אז שנו את השיטה secondStudent כך שתהיה בה רק השורה:
   `return null;`

3. הרצתם כל קובץ בדיקות בנפרד. זה כבר עניינכם, אבל ככה תדעו שהבסיס של התרגיל שלכם בסדר וסביר להניח שתקבלו את רוב הנקודות.

הגישו אך ורק את קובץ הזיפ שמיוצר אוטומטית ע"י הרצת הקובץ Tester.java (זה הקובץ שהורדתם ושמתם בחבילה main). הוא יכין את הזיפ וישים אותו בתיקיית הפרויקט.
אל תעשו זיפ לבד!!!

שאלה 1 (package root)

כתבו מחלקה בשם Rooter עם השיטות הבאות:

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

כדי לממש זאת, למחלקה יהיה שדה 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 (package library)

בשאלה זו נבנה מערכת קטנטנה לשמירת מידע על הספרים בספרייה שלנו. יהיו לנו שלוש מחלקות:

Author

מחלקה המתארת סופר ולה שלוש שיטות (החליטו בעצמכם איזה שדות פרטיים צריכים להיות לה):

public Author(String name, int birthYear)	בנאי המקבל את שם הסופר ואת שנת הלידה שלו.
public String getName()	מחזירה את שם הסופר.
public int getBirthYear()	מחזירה את שנת הלידה של הסופר.
public int getAge(int thisYear)	מחזירה את גיל הסופר בהנתן השנה הנוכחית.
public String toString()	מחזירה יצוג של הסופר כמחרוזת. למשל אם שם הסופר הוא Joshua, והוא נולד ב-1990, אז נקבל פה את המחרוזת: (Joshua(1990

Book

מחלקה המתארת ספר, ולה השיטות:

public Book(String title, Author auth)	בנאי המקבל את שם הספר ומופע של מחלקת Author - המתאר את הסופר של הספר.
public String getTitle()	מחזירה את שם הספר.
public String getAuthorName()	מחזירה את שם הסופר.
public int getAuthorBirthYear()	מחזירה את שנת הלידה של הסופר.
public String toString()	מחזירה יצוג של הספר כמחרוזת. אם הסופר הוא הסופר מהדוגמא שלמעלה, ושם הספר הוא "Magic", אז נקבל פה: (Magic written by Joshua(1990

Library

מחלקה המחזיקה מערך (שלא משנה את גודלו) של ספרים. יש לה השיטות:

public Library(int size)	בנאי המאתחל ספרייה ריקה עם size ספרים.
public void setBook(int bookNum, String title, Author auth)	מייצרת מופע של המחלקה Book עם השם title ועם הסופר auth, ושמה אותו במקום ה-bookNum בספרייה.
public Book getBook(int bookNum)	מחזירה את הספר הנמצא במקום ה-bookNum בספרייה. אם אין כזה, היא מחזירה null.

למשל, אם נריץ את הקוד הבא (למשל בתוך איזושהי שיטת main שנכתוב):

```
Library l = new Library(3);
Author a1 = new Author("Miguel de Cervantes", 1547);
Author a2 = new Author("Nikolai Gogol", 1809);
l.setBook(1, "Don Quixote", a1);
l.setBook(0, "Dead Souls", a2);
System.out.println(l.getBook(1));
System.out.println(l.getBook(0));
```

אז יודפס:

```
Don Quixote written by Miguel de Cervantes(1547)
Dead Souls written by Nikolai Gogol(1809)
```

שאלה 3 (package cities)

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. ואלו השיטות של City:

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

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 לאובייקט שאתם כרגע בתוכו, ואותו להעביר לשיטה connect.
(הקפידו להזין ערכים בשדות של האובייקט הנוכחי לפני שאתם קוראים לשיטה connect).

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 4 (package cards)

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:

public Card(int num, int suit)	בנאי רגיל.
public int getNum()	מחזירה את המספר של הקלף.
public int getSuit()	מחזירה את סוג הקלף.
public String toString()	תחזיר את המספר צמוד לאות הראשונה של סוג הקלף (כאות גדולה). למשל 5 לב יחזיר 5H.
public int compareTo(Card other)	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.

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, כי יתכן שלא כל המערך בשימוש. מומלץ להסתכל על StringStack בהרצאה 1, גם שם אנחנו שומרים איברים במערך, אבל לאו דווקא משתמשים בכולו. למחלקה Deck יש 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 (מתחילים מהקלף האחרון). 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 את הקלף האחרון 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).