2020 / 02 / 28

האוניברסיטה העברית בירושלים

ביה"ס להנדסה ומדעי המחשב ע"ש רחל וסלים בנין

מבוא לתכנות מונחה עצמים (67125)

(מועד ב')

מרצה: יואב קן-תור

מתרגלים: ג'ונה הריס, אלעד גרשון

צארים: שחר רסיסי , רועי פרידמן

תעודת זהות של הסטודנט/ית: _____

הוראות:

- משך המבחן הינו **3 שעות**
- המבחן הינו עם חומר סגור. אין להשתמש בכל חומר עזר או אמצעי חישוב שהוא, לרבות מחשבון
- למבחן שני חלקים:
  - חלק א' – כולל 6 שאלות, מהן יילקחו 5 השאלות בעלות הניקוד הגבוה ביותר
  - חלק ב' – כולל 2 שאלות קוד גדולות

יש לענות על כל שאלות המבחן במחברת הבחינה.

שימו לב לנקודות הבאות:

- "Less is more" – רשמו תשובה קצרה ומדויקת, שלא תשאיר מקום לספק שאכן הבנת את החומר
- ייתכן שהקוד המוגש יעבוד, אבל למבחן בתכנות מונחה עצמים זה לא מספיק. הקוד צריך להיות קריא ומתוכנן היטב. תעדו את הקוד (אפשר בעברית) רק אם יש חשש שמישהו לא יבין אותו
- זכור כי בעיצוב אין תשובה אחת נכונה, אבל בהחלט יש תשובות שהן לא נכונות. נקודות מלאות יתקבלו רק באם השתמשתם באחת מהדרכים המתאימות ביותר
- כל קטעי הקוד הנדרשים בשאלות השונות צריכים להיות כתובים בשפת JAVA
- אנו ממליצים לקרוא את כל המבחן מתחילתו עד סופו לפני תחילת הפתרון
- המבחן כתוב בלשון זכר, אך מיועד לכלל הסטודנטיות והסטודנטים הלוקחים את הקורס במידה שווה

בהצלחה!

חלק א'

- כולל 6 שאלות, מהן יילקחו 5 השאלות בעלות הניקוד הגבוה ביותר. כל שאלה שווה, לכל היותר, 10 נקודות. הקפידו לנמק כאשר התבקשתם.
- חלק מן השאלות כוללות מספר סעיפים, אשר מופרדים לתיבות שונות במודל ומסומנים בהתאם (למשל, אם שאלה 1 כוללת שני סעיפים הם יסומנו כ"שאלה 1.1" ו"שאלה 1.2"). הקפידו לענות על כל חלקי השאלות חלקי השאלה הוא לנוחות הפתרון בלבד.
- בשאלות הכוללות כתיבת קוד, הקפידו לכתוב קוד **קצר, פשוט ומובן ב-JAVA המשתמש בכלים המתאימים ביותר**. רק מימושים כאלו יקבלו ניקוד מלא.

שאלה 1

בחרו את האפשרות המתאימה ביותר עבור כל היגד בפסקה הבאה:

1. ממשק פונקציונלי (Functional Interface) הוא ממשק (interface) בעל [**מתודה אחת בלבד\מתודה אבסטרקטית אחת בלבד\מספר מתודות**]
2. כדי להשתמש בממשק פונקציונלי [**חייבים\לא חייבים**] להגדיר מחלקה (class) הממשת (implements) את הממשק
3. לממשק פונקציונלי [**יכולות\לא יכולות**] להיות מספר מתודות דיפולטיביות (default methods)
4. לממשק יכולים להיות שדות (fields) בעלי מודיפייר [**public/protected/default/private/כולם/אין אפשרות לשדות כלל**]
5. ממשק פונקציונלי [**חייב\לא חייב**] להיות מלווה באנוטציה (annotation) המתאימה לממשק פונקציונלי

שאלה 2

תארו, הסבירו, ותנו דוגמא לשימוש עבור תבנית העיצוב אסטרטגיה (Strategy). תארו מהו משווה (Comparator) והסבירו את הקשר ביניהם.

אסטרטגיה זו תבנית עיצוב המשמשת אותנו למקרים בהם אנחנו רוצים לשנות פונקציונליות בזמן ריצה. במצב בו יש לנו פונקציונליות שאנחנו רוצים לשנות בזמן ריצה בהתאם לנסיבות, נגדיר ממשק (האסטרטג) ומספר מחלקות שיממשו אותו. בזמן ריצה נבחר מתוכם את המתאים ביותר ונשתמש בו בקוד. לדוגמה בתרגיל מלחמת החלליות, יצרנו חלליות מכל מיני סוגים וטענו לתוך כל אחת אובייקט שהיה "הטייס" שיקבל את ההחלטות.
קומפרטור הוא ממשק פונקציונלי בעל מתודה אבסטרקטית המקבלת שני אובייקטים ומחזיר 1 אם הראשון גדול מהשני 0 אם הם שווים ו-1 אם הראשון קטן.
השימוש בקומפרטור באוספים המבצעים מיון הוא דוגמא לתבנית העיצוב אסטרטגיה. אלגוריתם המיון זקוק ליחס סדר בין האובייקטים כדי למיין אותם. כך אנחנו יכולים לבחור את יחס הסדר עבור המיון בזמן ריצה (לדוגמא מגדול לקטן או מקטן לגדול)

טעויות נפוצות:

- אסטרטגיה, שכחו לציין השינוי הוא בזמן ריצה.
- קומפרטור שכחו לציין שהוא ממשק עם מתודה המשווה בין שני איברים

שאלה 3

תארו והסבירו בקצרה מה הם מבני הנתונים TreeSet ו- HashSet. תארו מקרה בו נעדיף אחד על פני האחר.

שני מבני הנתונים מדמים קבוצה (set) מתמטי, ולכן לא יכולים להיות בהם כפילויות של איברים, ושניהם ממשים את הממשק Set ואת Collection.
TreeSet הוא מבנה נתונים המדמה קבוצה מתמטית ע"י עץ בינארי, ולכן מוגדר יחס סדר בין איברי הקבוצה. הכנסה, הוצאה וחיפוש ב-Set מסוג זה לוקחים O(logn) זמן, אבל הקבוצה תמיד נשארת ממוינת.
HashSet הוא מבנה נתונים המדמה קבוצה מתמטית ע"י טבלת גיבוב (hash table), ולכן לא מוגדר יחס סדר בין איברי הקבוצה. הכנסה, הוצאה וחיפוש ב-Set מסוג זה לוקחים בממוצע O(1) זמן.

נעדיף להשתמש ב TreeSet על פני HashSet כאשר ידוע לנו שיש חשיבות לסדר האיברים שנכנסים לקבוצה וגם שיהיה צריך לגשת לאיברים ע"פ סדר זה מספר רב של פעמים.

טעויות נפוצות:

- לא להזכיר ש TreeSet הוא Set ממויין
- במספר תשובות נגעו בעובדה שקיימים מצבים בהם ב-HashSet תהיה התנגשות של ההאשים של אובייקטים ואז, חד פעמית, ההכנסה של אובייקטים יכולה לקחת O(n). העובדה הזאת נכונה, אבל זה אירוע כל כך נדיר שעדיין נעדיף את העלות (שהיא רק ב-worst case, מה שבדרך כלל לא קורה) של O(n) ב HashSet מאשר את העלות (הוודאית) ב O(nlogn) של TreeSet, ולכן לא קיבלנו את התשובה הזאת

שאלה 4

בחרו את האפשרות המתאימה ביותר עבור כל היגד בפסקה הבאה:

1. כאשר מריצים unit tests בעזרת JUnit, הבדיקות (tests) מורצות [**לפי סדר כתיבתם\בסדר אקראי**]
2. כדי להשתמש ב-JUnit, [**חייבים\לא חייבים**] לכתוב את האנוטציות (annotations) באופן מפורש
3. באנוטציות של JUnit [**יכולים\לא יכולים**] להופיע פרמטרים (parameters) המשפיעים על אופן ריצת הבדיקות
4. האנוטציה Before של JUnit מסמנת מתודה שמבוצעת (executed) [**פעם אחת, לפני שכל הבדיקות רצות\לפני כל בדיקה\אחרי כל בדיקה\פעם אחת, אחרי שכל הבדיקות רצות**]
5. [**ניתן\לא ניתן**] להגביל זמן ריצה של טסט ספציפי ב- JUnit

שאלה 5

ממשו מתודה המקבלת מחרוזת s ורשימה של מחרוזות stringArray ובודקת האם קיימת מחרוזת ב-stringArray שהיא הזחה של s.

קלט: מחרוזת s ורשימה של מחרוזות stringArray

פלט: true אם קיימת מחרוזת ב-stringArray שהיא הזחה של s, ואחרת false

הערה: בהינתן מחרוזת

s1=[c_1, c_2, c_3,....,c_n]

נקרא למחרוזת s2 הזחה של s1 אם

s2=[c_k, c_(k+1), c_(k+2),..., c_(n-1), c_n, c_1, c_2, c_3,...,c_(k-1)]

עבור k כלשהו, כאשר c_1, c_2,..., c_n הן האותיות במחרוזת s1.

לדוגמה:

s="Oops", stringArray=["sOop", "a word"] returns true
s="Oops", stringArray=["soop", "a word"] returns false
s="Oops", stringArray=["sOop", "psOo"] returns true
s="word", stringArray=["this is a word", "a word"] returns false
s="word", stringArray=["this is a word", "dwor"] returns true

```
public static boolean hasShift(String s, String[] stringArray){
   String allShifts = s + s;
   for (String str: stringArray){
      if  ( (allShifts.contains(str)) && (str.length() == s.length()) ){
         return true;
      }
   }
   return false;
}
```

שימו לב על ידי "הכפלה" של המחרוזת המקורית אנחנו מקבלים את ההזחות האפשריות. ולכן כל שנותר לבדוק הוא האם קיימת במערך מחרוזת בעלת אורך מתאים ומוכלת במחרוזת של כלל ההזחות האפשריות.

טעויות נפוצות:

- אם משתמשים ב-contains חייבים לבדוק אורך.
- ההשוואה צריכה להיות עם equlas לא עם ==
- פתרון שגם קיבל את כל הנקודות היה בנייה של מערך עם כל ההזחות האפשריות והשוואה שלו מול המחרוזות במערך בניגוד לבנייה של כל ההזחות האפשריות פר מחרוזת במערך.
- בניה לא נכונה של ההזחות - substring בין אם באמצעות רגקס שבדק הכלה (המחרוזת עם +) או עם substring (לא ניתן לתת לה ערך שגדול מאורך המחרוזת המקורית).

שאלה 6

ממשו מתודה המקבלת מחרוזת s המייצגת תרגיל חשבון מוגבל (פירוט בהמשך) ומחזירה את התוצאה שלו.

<u>קלט:</u> מחרוזת s המייצגת תרגיל חשבון מוגבל. תרגיל חשבון מוגבל כולל אך ורק מספרים שלמים בין 0-9, סוגריים לא מקוננים ופעולות חיבור וכפל.

<u>פלט:</u> int המייצג את תוצאת התרגיל הנתון.

על חתימת הפונקציה להיות

public static int mathOperations(String s);

<u>לדוגמא:</u>

s="5+7*2+7", output: 26
s="(5+7)*(2+7)", output: 108
s="(3+3*3)*(5+6)", output: 132
s="1+1+1+1+8*2", output: 20

<u>הערות נוספות:</u>

- המחרוזת כוללת ספרות בודדות בפורמט של int (לא 6.0 אלא רק 6)
- תוצאת התרגיל יכולה להיות גדולה מ-9
- סוגריים מקוננים הם סוגריים בתוך סוגריים. הפונקציה אינה צריכה להתמודד עם קלטים מסוג זה. לדוגמה, המחרוזת הבאה **<u>אינה קלט חוקי לפונקציה:</u>**

  5*(3*(6+3)+2)
- הפונקציה יכולה להניח כי הקלט בפורמט תקין

```
public static int mathOperations(String s){

   int opener=s.indexOf('('),closer=-1;
   while(opener!=-1) {
       closer = s.indexOf(')');
       int answer = mathOperations(s.substring(opener + 1, closer));
       s=s.substring(0,opener)+Integer.toString(answer)+s.substring(closer+1);
       opener=s.indexOf('(');
   }
   return clacWithoutPar(s);
}

/**
* given string without parentheses and multiplication and addition calculate and return the answer
to it
* @param s string without parenthesis
```

```java
* @return answer to it.
*/
public static int clacWithoutPar(String s){
   int answer=0;
   //split into all the addition
   String[] sums=s.split("\\+");
   for (String sumi:sums) {
       //split into all the multiplication
       String[] multi=sumi.split("\\*");
       int tempAnswer=1;
       for (String element:multi)
           tempAnswer*=Integer.parseInt(element);
       answer += tempAnswer;
   }
   return answer;
}
```

טעויות נפוצות:

- ברוב התשובות לא התייחסו לאופרציות מחוץ לסוגריים או בין סוגריים
- במספר תשובות היה שימוש ב-regex מסורבל - בבעייה מהסוג הזה, regex בעיקר מקשה על הפתרון, במקום ליעל אותו

חלק ב'

- קראו בעיון את שתי השאלות הבאות. קראו כל שאלה עד הסוף לפני תחילת הפתרון
- לשאלות יכולות להיות מספר סעיפים ותתי-סעיפים. החלוקה לסעיפים נועדה לנוחות הפתרון בלבד
- שאלה ללא נימוק תאבד נקודות
- כאשר אתם כותבים נימוק הקפידו לפרט באילו עקרונות ותבניות עיצוב (design patterns) בחרתם להשתמש ומדוע
- הקפידו על עיצוב יעיל
- בכל פעם שהוגדרה מתודה למימושכם, אם חתימתה חסרה, עליכם להשלים אותה לפי שיקולכם
- ניתן להשתמש בקוד Java של מחלקות שלמדנו עליהן (לדוגמא הפונקציה sort של Collections)

שאלה 1

רקע

השנה היא 4580 בעיר המפלצות Monstropolis.
בעיר המתקדמת והטכנולוגית כל הטלפונים גם הם חכמים ומורכבים. לאחרונה שמו לב יצרני הטלפונים כי המפלצות המבוגרות בעיר מתקשות להסתדר עם מכשירים כה מורכבים. לשם כך החליטו לבנות טלפונים נוספים, פשוטים יותר, שיקלו על חייהן של המפלצות המבוגרות, וגייסו אתכם לעזור במשימה.

נתונות לכם המחלקות SmartPhone ו- Contact (אינכם צריכים לממש אותן). המחלקה SmartPhone מייצגת את הטלפון היחיד שקיים כיום בשוק. המחלקה Contact מייצגת איש קשר בטלפון.

לפניכם ה- API של שתי המחלקות:

public class Contact	
public Contact(String name, int phoneNumber, String address)	יצירת איש קשר חדש עם הפרטים הנתונים

public String getName()	מתודה המחזירה את השם של איש הקשר
public int getPhoneNumber()	מתודה המחזירה את מספר הטלפון של איש הקשר
public String getAddress()	מתודה המחזירה את הכתובת של איש הקשר

public class SmartPhone	
public SmartPhone()	יצירת SmartPhone חדש
public String searchBrowser(String query)	חיפוש של הערך **query** בדפדפן של הטלפון. המתודה מחזירה את תוצאת החיפוש
public void sendMessage(String contactName, String msg)	שליחת ההודעה **msg** לאיש הקשר ששמו **contactName**
public void dial(int phoneNumber)	חיוג למספר הטלפון הנתון
public void pickupCall()	מתודה העונה לשיחה נכנסת לטלפון זה
public void navigate(String destination)	מתודה המנווטת ליעד **destination** בעזרת אפליקציית הניווט של הטלפון
public void playMusic()	התחל לנגן מוזיקה
public void stopMusic()	הפסק לנגן מוזיקה
public void addContact(Contact contact)	הוספה של איש הקשר הנתון לרשימת אנשי הקשר בטלפון
public void addContact(String name, int phoneNumber, String address)	הוספה של איש קשר חדש לטלפון עם הפרטים התונים
public Contact getContact(String contactName)	מתודה המחזירה את איש הקשר עם השם הנתון
public void callContact(String contactName)	מתודה המחייגת למספר הטלפון של איש הקשר עם השם הנתון

כפי שניתן לראות, SmartPhone הוא טלפון משוכלל מאוד ומכיל שלל פונקציות.

סעיף א'

בסעיף זה נבנה טלפון פשוט יותר מהטלפון המורכב הקיים. הטלפון הפשוט יתמוך אך ורק בפעולות הבאות:

- חיוג לאיש קשר אחד שהוגדר כ"איש קשר ראשי" בעת יצירת הטלפון
- חיוג למספר חירום (מספר הטלפון של שירות החירום הוא 911)
- לענות לשיחה

עליכם לממש את המחלקה SimplePhone, אשר ה-API שלה מוגדר בדיוק על ידי המתודות הבאות:

public SimplePhone(Contact primaryContact)	יצירת SimplePhone חדש עם איש הקשר הראשי הנתון
public void callPrimaryContact()	חיוג לאיש הקשר הראשי של הטלפון

public void callEmergency()	חיוג למספר 911
public void pickupCall()	מענה לשיחה נכנסת לטלפון

הבהרות:

- שימו לב, באפשרותכם להיעזר במחלקה SmartPhone כדי לממש את המחלקה SimplePhone המחלקה SmartPhone כבר מכילה קוד שעשוי לעזור לכם, כמו למשל פעולת חיוג, ועוד.
- הטיפוס SimplePhone אינו חושף אף מתודה שאינה ב-API הנ"ל

סעיף ב'

יצרני הטלפון החליטו כי הטלפון SimplePhone הוא פשוט מדי. כעת הם רוצים לבנות סוג של SimplePhone, אשר יתמוך בהוספת פעולות נוספות שלא הוגדרו מראש. לשם כך נממש סוג חדש של SimplePhone אשר מאפשר הוספה של **פרוצדורות**. פרוצדורה היא אוסף של פעולות שמתבצעות ברצף על טלפון חכם.

בסעיף זה נממש טלפון חדש התומך בפרוצדורות. להלן הסבר מפורט.

לפניכם דוגמאות לפרוצדורות שנרצה להיות מסוגלים להגדיר בטלפון החדש (**אין צורך לממשן**):

1. פרוצדורה בשם stopMusicAndCallAlice, אשר מפסיקה את המוזיקה בטלפון ומתקשרת לאיש הקשר ששמו **Alice**. פרוצדורה זו אינה מקבלת ארגומנטים כלל
2. פרוצדורה בשם broadcastMessage, אשר שולחת הודעת טקסט למספר רב של אנשי קשר. הפרוצדורה מקבלת n ארגומנטים. הארגומנט הראשון מייצג את ההודעה. יתר הארגומנטים הם שמות של אנשי קשר בטלפון, אליהם רוצים לשלוח את ההודעה. פרוצדורה זו תומכת בכל מספר $n \geq 2$ של ארגומנטים (הודעה + לפחות איש קשר אחד)

שימו לב: פרוצדורות שונות עלולות לדרוש מספר שונה של ארגומנטים.

נתון לכם הממשק SmartPhoneProcedure המייצג פרוצדורה על טלפון חכם

```
@FunctionalInterface
public interface SmartPhoneProcedure {

   /**
    * execute this procedure on smartphone p with arguments given in args
    *
    * @param p    the SmartPhone on which to operate
    * @param args list of arguments to this procedure
    */
   void execute(SmartPhone p, String[] args);
}
```

הפונקציה execute בממשק מקבלת שני ארגומנטים:

1. הטלפון החכם עליו נדרשים לבצע את הפרוצדורה
2. מערך של מחרוזות שהן הארגומנטים הנדרשים לביצוע הפרוצדורה. זאת על מנת לאפשר מספר שונה של ארגומנטים עבור פרוצדורות שונות

הטלפון הפשוט החדש אשר יתמוך בפרוצדורות ייקרא *SimpleP hone* ++ (SimplePhone Plus Plus). טלפון זה יוגדר ע"י המחלקה SimplePPP. עליכם לממש את המחלקה SimplePPP במלואה, ובה השיטות הבאות:

SimplePPP(Contact primaryContact)	Create a new SimplePPP with the given primary contact
public void addProcedure(String procedureName, SmartPhoneProcedure f)	Add the procedure **f** with the name **procedureName** to this phone
public void executeProcedure(String procedureName, String[] args)	Execute the procedure with name **procedureName** with the given arguments

סעיף ג'

בסעיף זה נשתמש במחלקה שבנינו בסעיף קודם: ניצור פרוצדורה, נוסיף אותה לטלפון מסוג SimplePPP ונפעיל אותה.

תחילה נגדיר פרוצדורה בשם DriveToFriend. הפרוצדורה מקבלת ארגומנט אחד בלבד, שהוא שם של איש קשר. הפרוצדורה מבצעת את הפעולות הבאות:

1. שליחת הודעה לאיש הקשר הנתון עם הטקסט: "On my way"
2. ניגון מוזיקה בטלפון
3. ניווט לכתובת של איש הקשר

כתבו פונקציית main אשר מבצעת את הפעולות הבאות:

1. יצירת טלפון מסוג SimplePPP עם איש קשר ראשי עם הפרטים הבאים
   $Name: \text{“}Mike\ Wazowski\text{”}, phoneNumber: 12899, address: \text{“}Monstropolis, A-113\text{”}$
2. הוספת הפרוצדורה DriveToFriend לטלפון, כפי שהוגדרה למעלה
3. הפעלת הפרוצדורה עם ארגומנט "Mike Wazowski" (שימו לב, זהו אכן שם של איש קשר בטלפון ולכן קלט תקין לפרוצדורה)

הבהרות:

- בסעיף זה הנכם נדרשים בין היתר לממש את הפרוצדורה DriveToFriend
- באפשרותכם לרשום קוד עזר נוסף במתודות/מחלקות אחרות אם יש צורך בכך
- הפרוצדורה DriveToFriend יכולה להניח קלט תקין - מערך args בגודל 1 המכיל שם של איש קשר קיים בטלפון (אין צורך לבדוק זאת במימושכם)

פתרון:

<u>סעיף א</u>

האובייקט SimplePhone תומך במספר פעולות מצומצם אשר ממומשות ב- SmartPhone. למעשה הוא גרסה 'קלה' יותר של SmartPhone וחושף API פשוט יותר. לפיכך, מתאים לממש את SimplePhone בתור Facade למחלקה SmartPhone.
כתוב במפורש ש- SimplePhone אינו חושף אך מתודה פרט לאלו שצוינו, לכן תהיה זו טעות לממש את SimplePhone כמחלקה שיורשת מ- Smartphone. לשם מימוש המחלקה נחזיק אובייקט מסוג SmartPhone בתור שדה ונעביר אליו בקשות.

<u>סעיף ב</u>

האובייקט SimplePPP הוא סוג של SimplePhone, ותומך בכל פעולותיו ועוד פעולות נוספות. לכן נממש את SimplePPP בתור מחלקה היורשת מ- SimplePhone. על מנת לממש את ההתנהגות הנדרשת בפעולות הנוספות שהוגדרו, נחזיק במחלקה מיפוי בין שם של פרוצדורה לפרוצדורה.
במיפוי זה נאכסן פרוצדורות שיוסיפו לטלפון, ומשם נגיע אליהם (דרך השם) כשנרצה להפעיל אותן.

<u>סעיף ג</u>

עלינו לממש אובייקט מסוג SmartPhoneProcedure, אשר מתודת ה- execute שלו מממשת את כל הפעולות שהוגדרו עבור הפרוצדורה DriveToFriend. נוכל לעשות זאת ע"י מימוש מחלקה המממשת את הממשק, ע"י שימוש במחלקה אנונימית או בעזרת ביטוי למבדה (משום שזהו ממשק פונקציונלי).

```
public class SimplePhone {
```

```java
  protected SmartPhone phone;
  private String primaryContactName;

  public SimplePhone(Contact primaryContact) {
    this.phone = new SmartPhone();
    this.phone.addContact(primaryContact);
    this.primaryContactName = primaryContact.getName();
  }

  public void callPrimaryContact() {
    this.phone.callContact(this.primaryContactName);
  }

  public void pickupCall() {
    this.phone.pickupCall();
  }

  public void callEmergency() {
    this.phone.dial(911);
  }
}

public class SimplePPP extends SimplePhone {
  private HashMap<String, SmartPhoneProcedure> procedures = new HashMap<>();

  SimplePPP(Contact primaryContact) {
    super(primaryContact);
  }

  public void addProcedure(String procedureName, SmartPhoneProcedure f) {
    this.procedures.put(procedureName, f);
  }


  public void executeProcedure(String procedureName, String[] args) {
    SmartPhoneProcedure f = this.procedures.get(procedureName);
    f.execute(this.phone, args);
  }
}

public class DriveToFriendProcedure implements SmartPhoneProcedure {

  @Override
  public void execute(SmartPhone p, String[] args) {
    String contactName = args[0];
    Contact c = p.getContact(contactName);
    p.sendMessage(contactName, "On my way");
    p.playMusic();
```

```
    p.navigate(c.getAddress());
  }

  public static void main(String[] args) {
    Contact c = new Contact("Mike Wazowski", 12899, "Monstropolis, A-113");
    SimplePPP p = new SimplePPP(c);
    DriveToFriendProcedure dtf = new DriveToFriendProcedure();
    String driveToFriendProcedureName = "drive-to-friend";
    p.addProcedure(driveToFriendProcedureName, dtf);
    p.executeProcedure(driveToFriendProcedureName, new String[]{"Mike Wazowski"});
  }
}
```

טעוית נפוצות:

- המחלקה SimplePhone יורשת מ- SmartPhone. נאמר במפורש ש- SimplePhone צריכה שלא לחשוף אף מתודה פרט לאלו שצוינו. ע"י ירושה מ- SmartPhone אנו מגדילים את ה-API ולא שומרים על הטלפון "פשוט" יותר
- המחלקה SimplePPP מממשת את SmartPhoneProcedure. זוהי שגיאה מוחלטת - המחלקה SimplePPP אמנם עובדת עם פרוצדורות, אך היא איננה פרוצדורה בעצמה
- קריאה למתודה executeProcedure עם ארגומנט שני שהוא שם של איש קשר (String). המתודה צריכה לקבל מערך של מחרוזות, ולא מחרוזת בודדת
- גישה לשדה פרטי שהוגדר במחלקת אב. לא ניתן לגשת לשדות פרטיים של מחלקה כלשהי מחוץ למחלקה, גם לא מתוך מחלקות יורשות

שאלה 2

רקע

לאחר ששיחק במספר רב של משחקים, החליט קים באג אין לפתח משחק מחשב משלו - משחק גולף! כמובן, פיתוח משחק מחשב איננה משימה פשוטה. לצורך הפיתוח נדרש יישום של מספר טיפוסים, כגון מצלמה (Camera) ואובייקטי משחק (Game Object).
כל אובייקט משחק מורכב מטקסטורה ומודל תלת-מימדי. מספר אובייקטים יכולים לחלוק את אותה הטקסטורה. טקסטורה היא דבר מורכב - היא מכילה מידע אודות הצבע של מספר רב של פיקסלים. לכן, העבודה עם טקסטורות יקרה במשאבים (הן מבחינת זמן והן מבחינת זכרון).
כל אובייקט נמצא במיקום מסוים במשחק, הנתון ע"י 3 קואורדינטות במרחב וזווית סיבוב ביחס לציר ה-Y.

לשם בניית משחק הגולף, התחיל קים ויצר את המחלקות הבאות (אין צורך לממשן):

Class Vector3	
public Vector3(float x,float y,float z)	יצירת וקטור חדש עם הקואורדינטות הנתונות
public float distance(Vector3 other)	מתודה המחזירה את המרחק האוקלידי מהוקטור הנתון לוקטור זה
public float length()	מתודה המחזירה את אורך הוקטור (הנורמה)
public void normalized()	מתודה המשנה את אורך הוקטור ל-1
public void add(Vector3 addedVector)	מתודה המוסיפה את הוקטור הנתון לוקטור זה

Class Texture	
public Texture(String path)	יצירת טקסטורה חדשה ע"י טעינתה מהנתיב הנתון
public int getWidth()	מתודה המחזירה את אורך הטקסטורה
public int getHeight()	מתודה המחזירה את גובה הטקסטורה
public int getID()	מתודה המחזירה מספר מזהה של הטקסטורה

Class Model	
public Model(String path)	יצירת מודל חדש ע"י טעינתו מהנתיב הנתון
public int getID()	מתודה המחזירה מספר מזהה של המודל
public int getVertexCount()	מתודה המחזירה את מספר הקודקודים במודל

סעיף א'

משחק הגולף מתואר בעזרת קבצי טקסט. כל קובץ מכיל מידע אודות האובייקטים השונים במשחק וכן מספר פרטים נוספים. על כל קובץ להיות כתוב בפורמט מדויק שנקבע מראש. קים הטיל עליכם את המשימה לוודא את תקינות הפורמט של הקבצים.

<u>פורמט הקובץ</u>:

תוכן הקובץ	Format
• שורות המתארות מצלמות (מתחילות באות C) ○ השורה מכילה את מיקום המצלמה • שורות המתארות אובייקטי משחק (מתחילות באות G) ○ השורה מכילה את קוד המודל, קוד הטקסטורה ומיקום • שורה יחידה המתארת את נקודת ההתחלה של המשחק (מתחילה באות S) ○ מכילה מיקום • שורה יחידה המתארת את נקודת הסיום של המשחק (מתחילה באות E) ○ מכילה מיקום	**C:** x1,y1,z1 **C:** x2,y2,z2 … **C:** xk,yk,zk **G: M:**mc1 **T:**tc1 **:** x1,y1,z1 **G: M:**mc2 **T:**tc2 **:** x2,y2,z2 …. **G: M:**mcn **T:**tcn **:** xn,yn,zn **S:** x,y,z **E:** x,y,z

הסברים:

- האותיות המודגשות הם קבועים (keywords) המופיעים בקובץ כפי שהם
- x,y,z,x1,y1,z1,x2,y2,z2 וכו', הם מספרים ממשיים (double, חיובי או שלילי) המייצגים קואורדינטות
- mc1, mc2 וכו', הם מספרים מזהים של מודל, המיוצגים ע"י מספר שלם (int)
- tc1, tc2 וכו', הם מספרים מזהים של טקסטורה, המיוצגים ע"י מספר שלם (int)
- בקובץ משחק תקין חייבת להיות מצלמה אחת לפחות
- בקובץ משחק תקין אין בהכרח אובייקטי משחק

לפניכם דוגמה לתוכן של קובץ המתאר משחק גולף. בקובץ זה מתוארים שתי מצלמות, שלושה אובייקטי משחק, נקודת התחלה ונקודת סיום:

```
C: 1.2,-5.4,6.1
C: -2.1,4,-6.2
G: M:1 T:1 : 5.1,4,2
G: M:3 T:1 : 3,7.3,1
G: M:2 T:2 : 6.4,9,4.8
S: -1.0,2.4,3.5
E: 4.1,-7.5,7
```

ממשו את המתודה verify, המקבלת תוכן של קובץ משחק ומחזירה תשובה בוליאנית - האם הקובץ בפורמט תקין:

```
public static boolean verify(String fileData)
```

הבהרות:

- הפרמטר fileData מכיל את כל התוכן של הקובץ. בפרט, מכיל תווים של ירידת שורה

סעיף ב'

מחלקת אובייקט משחק מוגדרת באופן הבא:

Class GameObject	
public GameObject(Model model, Texture texture, Vector3 position)	בנאי היוצר אובייקט משחק חדש. זווית הסיבוב ביחס לציר ה-Y מאותחלת לאפס
public void walk(float x,float y,float z)	מתודה המקדמת את האובייקט לפי הקואורדינטות בכל ציר
public void setYRotation(float rotation)	מתודה המשנה את זווית הסיבוב של האובייקט לזווית הנתונה
public int getModeID()	מתודה המחזירה את המספר המזהה של המודל
public int getTextureID()	מתודה המחזירה את המספר המזהה של הטקסטורה

על מנת להציג את האובייקטים, בנה קים בנוסף את הפונקציות הבאות (אין צורך לממש ואסור להרחיב):

Class RendererUtils	
public static void uploadTexture(Int textureID)	מתודה הטוענת את הטקסטורה הנתונה. בכל רגע טעונה טקסטורה אחת בלבד. כל טעינה דורסת את הטקסטורה הקודמת שהוטענה
public static void drawModel(Int modelID)	מתודה המציגה את המודל הנתון עם הטקסטורה הטעונה כעת

כדי להציג אובייקט משחק במרחב, יש להשתמש בשיטה drawModel, כאשר הטקסטורה הטעונה כעת מתאימה לאובייקט המשחק. זכרו כי עבודה עם טקסטורות היא יקרה במשאבים (מבחינת זמן וזיכרון), לכן **יש להשתמש בפעולה uploadTexture כמה שפחות**!

על מנת להציג את האובייקטים הנכם נדרשים לממש את המחלקה Renderer

Class Renderer	
public void addGameObject(GameObject gameObject)	הוספת אובייקט משחק אחד ל- Renderer זה.
public void renderAll()	מתודה המציגה את כל האובייקטים ב Renderer זה

ממשו את המחלקות GameObject ו- Renderer.

הבהרות:

- כפי שצוין קודם, המתודה uploadTexture היא מתודה כבדה ורצוי שתתבצע כמה שפחות
- ניתן להניח כי זווית הסיבוב היא בטווח תקין (אין צורך לוודא את הקלט)

סעיף ג'

בנוסף, בכל משחק יש אוסף מצלמות. אלו מגדירות זוויות שונות, מהן ניתן לראות את המשחק. בכל משחק קיימת לפחות מצלמה אחת - המצלמה הראשית. נוסף אליה ניתן להוסיף מצלמות משניות.

אובייקט מצלמה מוגדר באופן הבא:

Class Camera	
Camera(Vector3 position)	בנאי המחלקה, יוצר מצלמה חדשה עם המיקום הנתון וזווית סיבוב 0
public void setYRotation(float rotation)	מתודה המשנה את זווית הסיבוב של המצלמה
public void moveTo(Vector3 newPosition)	מתודה המשנה את המיקום הנוכחי למיקום החדש
public static Camera getMainCamera()	פונקציה המחזירה את המצלמה הראשית

המצלמה הראשונה שנוצרת בתכנית היא המצלמה הראשית

כדי לבנות מצלמות נגדיר את המחלקה CameraFactory ובה מתודה אחת בלבד (אין להוסיף שדות/פונקציות סטטיות/פונקציות לא סטטיות):

Class CameraFactory	
public static List<Cameras> buildCameras(List<Vector3> cameraLocations)	פונקציה היוצרת אוסף של מצלמות במיקומים הנתונים

ממשו את המחלקות Camera ו- CameraFactory.

הבהרות:

- בזמן משחק יכולים להיות מספר אובייקטי מצלמה שונים

לפניכם דוגמה לשימוש בקוד שכתבנו בשאלה זו. קוד זה אמור לעבוד עם המימוש שלכם

```
//build game objects according to the given fileData
public static  List<GameObject> getGameObjects(String fileData){/***/}
//returns a list of all camera positions according to the given fileData
public static  List<Vector3> getCamerasConfiguration(String fileData){/***/}
//activate the given camera
public static void activateCamera(Camera c){/***/}
//read the file and return its content as String
public static String readInformation(String pathToFile){/***/}

.public static void main(String[] args){
  String fileData = readInformation(args[0]);
  Renderer renderer = new Renderer();

  if(!verify(fileData))
     return;
  List<GameObject> gameObjectsToAdd = getGameObjects(fileData);
  for(GameObject gObject : gameObjectsToAdd)
          renderer.addGameObjects(gObject);
  List<Camera> cameras = CameraFactory.buildCameras(getCamerasConfiguration(fileData));
  Camera.getMainCamera().setYRotation(43.2f);
  activateCamera(Camera.getMainCamera());
  renderer.renderAll();
}
```

פתרון:

סעיף א יש צורך להשתמש בRegex כדי לראות שהקובץ תקין, לא היה צורך בקלוט את את המידע על ידי שימוש בBuffers וקולטי מידע מן הקובץ, אשר אנשים לא ידעו כיצד להשתמש. היה צורך להתחשב במספרים מסוג double ולכן היה צורך לקלוט גם מספרים שליליים וגם עם נקודה לאחר מכן. הפותר החכם היה יודע שאפשר לחבר regex ולכן היה צורך רק להגדיר פעם אחת את המספר double ואז לחבר 3 לקורדינטה.

סעיף ב כפי שהיה רשום לא מעט, היה צורך להשתמש בuploadTexture כמה שפחות, לכן העיקר היה לבנות מבנה נתונים המאפשר הוספה של פרטים כך שבעלי אותו texture ובכך בעלי אותו textureID יקראו uploadTexture רק פעם אחת. ואז כל מודל ב gameObject השונים יצויר בdrawModel ולא יותר. אני השתמשתי בHashMap עם list אך קיבלתי פתרונות של מבני נתונים העושים מיון.

סעיף ג Main Camera מתנהגת כמו singlaton, לעומת שאר המצלמות המשניות. היה צורך לעשות אותה כstatic ולהגדיר אותה ב constructor הראשון, חוץ מזה היה צורך בנימוק להסביר שזה בהחלט דומה לsingalton.

```
//------------סעיף א------------------
public static boolean verify(String fileData)
{
   String singleNumber="-?\\d+(\\.[\\d]+)?";
   String location=singleNumber+","+singleNumber+","+singleNumber+"\n";
   String CameraRegex="(C: "+location+")+";
   String GameObjectRegex="(G: M:\\d+ T:\\d+ :"+location+")*";
   String StartRegex="S: "+location;
   String EndRegex="E: "+location;
   String All=CameraRegex+GameObjectRegex+StartRegex+EndRegex;
   Pattern p=Pattern.compile(All);
   Matcher m=p.matcher(fileData);
   return m.matches();
}

//------------סעיף ב------------------
class GameObject{
   private Texture texture;
   private Model model;
   private Vector3 pos;
   private float rotationY;

   public GameObject(Model m,Texture t,Vector3 position){
       model=m;
       texture=t;
       pos=position;
       rotationY=0;
   }
   public void walk(float x, float y,float z){pos.addVector(new Vector3(x,y,z));}
   public void setYRotation(float rotation) {rotationY=rotation;}
   public int getModelID(){return model.getID();}
   public int getTextureID(){return texture.getID();}
}

class Renderer {
   private static HashMap<Integer, List<GameObject>> gameObjects;

   public void addGameObjects(GameObject gameobject) {
       if (gameObjects == null) gameObjects = new HashMap<>();
       List<GameObject> batch;
```

```
        if (gameObjects.containsKey(gameobject.getTextureID())) {
            batch = gameObjects.get(gameobject.getTextureID());
            batch.add(gameobject);
        } else {
            batch = new ArrayList<>();
            batch.add(gameobject);
        }
        gameObjects.put(gameobject.getTextureID(), batch);
    }
    public void renderAll()
    {
        for (int textureID:gameObjects.keySet()) {
            RendererUtils.uploadTexture(textureID);
            for (GameObject g:gameObjects.get(textureID))
                RendererUtils.drawModel(g.getModelID());
        }
    }
}
//------------סעיף ג------------------
class Camera{
    private Vector3 pos;
    private float rot;
    //singleton for main camera
    private static Camera mainCamera;

    public Camera(Vector3 pos){
        if(mainCamera==null) mainCamera=this;
        this.pos=pos;
        this.rot=0;
    }

    public void moveTo(Vector3 newPos) {
        pos=newPos;
    }
    public static Camera getMainCamera(){return  mainCamera;}
    public void setYRotation(float rotateBy){rot=rotateBy;    }
}

class CameraFectory{
    public static List<Camera> createCameras(List<Vector3> cameraToAdd){
        List<Camera> cameras=new ArrayList<>();
        for(Vector3 v:cameraToAdd)
            cameras.add(new Camera(v));
        return cameras;
    }
}
```

טעויות נפוצות

- לא התחשבו במספרים double עבור מיקום בקורדינציה.
- לא השתמשו במבני נתונים נכון עבור renderAll, או שבכלל לא השתמשו בuploadTexture או שהשתמשו עבור כל איבר.
- לא הוסיפו נימוק עבור singleton, שיש לנו static ובניה אחת עבורו, צריך לזרוק מילה על זה.

Object Oriented Programing Exam Cheatsheet (2020)

- The APIs included in this sheet are incomplete - they only represent a small subset of the methods in the classes they refer to
- The provided methods shouldn't necessarily be used in order to fully answer the exam

Collections

class HashMap<K,V>	
V get(Object key)	Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key
Set<K> keySet()	Returns a Set view of the keys contained in this map
V put(K key, V value)	Associates the specified value with the specified key in this map. If the map previously contained a mapping for the key, the old value is replaced.
V remove(Object key)	Removes the mapping for the specified key from this map if present
Collection<V> values()	Returns a Collection of the values contained in this map

class HashSet<E>	
boolean add(E e)	Adds the specified element to this set if it is not already present
boolean contains(Object o)	Returns true if this set contains the specified element
boolean remove(Object o)	Removes the specified element from this set if it is present
Iterator<E> iterator()	Returns an iterator over the elements in this set

class LinkedList<E>	
void addFirst(E e)	Inserts the specified element at the beginning of this list
void addLast(E e)	Appends the specified element to the end of this list
E getFirst()	Returns the first element in this list
E getLast()	Returns the last element in this list
E pollFirst()	Retrieves and removes the first element of this list, or returns null if this list is empty
E pollLast()	Retrieves and removes the last element of this list, or returns null if this list is empty
E get(int index)	Returns the element at the specified position in this list
int indexOf(Object o)	Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element

class TreeSet<E>	
TreeSet()	Constructs a new, empty tree set, sorted according to the natural ordering of its elements
TreeSet(Comparator<? super E> comparator)	Constructs a new, empty tree set, sorted according to the specified comparator
boolean add(E e)	Adds the specified element to this set
Iterator<E> iterator()	Returns an iterator over the elements in this set in ascending order
boolean remove(Object o)	Removes the specified element from this set if it is present

Regex

class Pattern	
static Pattern compile(String regex)	Compiles the given regular expression into a pattern
Matcher matcher(String input)	Creates a matcher that will match the given input against this pattern

class Matcher	
boolean matches()	Attempts to match the entire region against the pattern
boolean find()	Attempts to find the next subsequence of the input sequence that matches the pattern
int start()	Returns the start index of the previous match
int end()	Returns the offset after the last character matched
String group(int group)	Returns the input subsequence captured by the given group during the previous match operation

Lambda Expressions

A general lambda expression structure:

```
(ARGUMENT_LIST) -> {BODY}
```

Examples:

```
(int x, int y) -> {int z = x+y; return z+1;}
(x,y) -> x+y
x -> foo(x)
(Cat c1, Cat c2) -> c1.compareTo(c2)
```