מבחן בקורס תוכנה 1

ביה"ס למדעי המחשב, אונ' תל אביב

סמסטר א' 2024, מועד א', 02/04/2024

מרצה: מיכל קליינבורט

מתרגלים: איליה שברין, מיכאל בילביץ', קורן שריג

משך הבחינה: 3 שעות.

ללא חומר עזר

סך הניקוד על השאלות בבחינה הוא 107, אך הציון המקסימלי אותו ניתן לקבל הוא 100.

במבחן 19 עמודים מודפסים – בידקו שכולם בידיכן. העמוד האחרון בבחינה הינו דף ריק, לשימוש במקרה הצורך.

- יש לענות על כל השאלות. אנו ממליצים לא "להיתקע" על אף שאלה, אלא להמשיך לשאלות אחרות ולחזור לשאלה אח"כ.
- יש להניח, אלא אם צוין אחרת, כי:
  - הקוד שמופיע במבחן מתאים לגירסה Java 21.
  - כל החבילות הדרושות יובאו, ואין צורך לכתוב שורות import בגוף הקוד.
  - כל מחלקה שהיא public מופיעה בקובץ Java משלה.
  - בכל שאלה, כל המחלקות מופיעות באותה חבילה (package).
  - בזמן הבחינה, אתם נדרשים לזהות שגיאות קומפילציה שנוצרות כתוצאה מהפרת עקרונות Java-יים ושימוש לא נכון במחלקות/פונקציות. אם ישנה טעות הקלדה (סוגר חסר, שימוש באות גדולה שלא לצורך וכו') אין לראות בסיבות אלה גורמים לשגיאות קומפילציה.
  - בסוף הבחינה מופיע נספח עם תיעוד של מחלקות בהן אתם עשויים לעשות שימוש בחלק הפתוח של הבחינה.
  - הקוד שאתם נדרשים לספק צריך להיות יעיל ולהימנע ממחזור קוד. חלק מהציון ניתן גם היבטים אלה, ולא רק על נכונות הפתרון.

בבחינה זו מופיע קוד שבחלקו אינו מתקמפל, אינו רץ או שנוגד את הסטנדרטים של Java כפי שנלמדו בקורס, וזאת מתוך מטרה לבחון ידע והבנה של נושאים מסוימים. אין לראות בקטעי קוד אלה דוגמה לכתיבה נכונה ב Java.

מבנה הבחינה:

הבחינה מורכבת משלושה חלקים: חלק שכולל שאלות אמריקאיות (8 שאלות, כל אחת שווה 4 נק'), חלק שכולל שאלת Design (20 נק'), וחלק פתוח (כמה סעיפים, שמסתכמים ל 55 נק'). עליכם לענות על הבחינה באופן הבא:

1. בשאלות האמריקאיות: יש להקיף בעיגול בצורה ברורה וחד משמעית את התשובה הנכונה לדעתכם. ניתן להוסיף נימוק. הנימוק אינו חובה, אך יכול לעזור לכם במקרים של ערעורים או קבלה של יותר מתשובה אחת נכונה.
2. בשאלת ה Design יש לתאר את מבנה המחלקות והמנשקים כמפורט בשאלה.
3. בסעיפי השאלה הפתוחה יש להשלים את הקוד החסר או התשובה המילולית במקומות המסומנים במסגרת.

בהצלחה!

שאלה 1 (32 נקודות)

בשאלה זו 8 סעיפים שאינם קשורים זה לזה. משקל כל סעיף 4 נק'.

סעיף א' (4 נק')

האם המחלקה הבאה תתקמפל?

```
public class A {
	public void func(List<String> lst) {}
	public void func(ArrayList<Object> lst) {}
	public static void main(String [] args) {
		A a = new A();
	}
}
```

הקיפו: מתקמפל / לא מתקמפל

נימוק

סעיף ב' (4 נק')

האם המחלקה הבאה תתקמפל?

```
public class B {
	public void func(List<? extends Object> lst) {}
	public void func(List<Object> lst) {}
	public static void main(String [] args) {
		B b = new B();
	}
}
```

הקיפו: מתקמפל / לא מתקמפל

נימוק

סעיף ג' (4 נק')

נתון הקוד הבא בו ייתכן שחסרה מילה. נרצה לדעת מתי הקוד במחלקה Cin יוכל להשתמש בטיפוס T.

```
public class C<T> {
      public _________ class Cin { /* some code here */ }
}
```

הקיפו את הטענה הנכונה:

1. כדי שהקוד במחלקה Cin יוכל להשתמש בטיפוס T יש להחליף את _____ במילה static
2. כדי שהקוד במחלקה Cin יוכל להשתמש בטיפוס T יש למחוק את _____ ולא להוסיף דבר
3. בשתי האפשרויות (1,2) הקוד ב Cin יוכל להשתמש בטיפוס T

נימוק

סעיף ד' (4 נק')

נתון הקוד הבא, בו חסר טיפוס הפרמטר הפורמלי של הפונקציה func.

```
public class D {
      public static void func( ___________ lst) {
            lst.add(new String());
      }
      public static void main(String [] args) {
            List<String> lst= new ArrayList<>();
            func(lst);
      }
}
```

איזו מבין האפשרויות הבאות להשלמה במקום _____ תגרום לכך שהקוד **לא יתקמפל**? הקיפו את התשובה הנכונה:

1. List<String>
2. List<? extends String>
3. List<? super String>

נימוק

סעיף ה' (4 נק')

נתון הקוד הבא:

```
public class E {
      public static void main(String[] args) {
             List<Integer> ints = Arrays.asList(5,4,3,2);
             ints.stream()
                 .peek(x->{System.out.println("peek");})
                 .sorted((x,y)->{return Integer.compare(x, y); })
                 .forEach(x->{System.out.println("forEach");}  );
      }
}
```

הקיפו את הטענה הנכונה לגבי ההדפסות:

1. יופיעו 4 הדפסות של המילה peek ולאחריהן 4 הדפסות של המילה forEach
2. ההדפסות יהיו של המילים peek ו forEach לסירוגין (מתחילים מ peek)
3. ההדפסות יהיו של המילים forEach ו peek לסירוגין (מתחילים מ forEach)

נימוק

סעיף ו' (4 נק')

נתון המנשק הבא:

```
public interface F extends Comparable<Integer>{
      public int func(Integer a);
}
```

הקיפו הטענה הנכונה: F הוא מנשק פונקציונלי / F אינו מנשק פונקציונלי

נימוק

סעיף ז' (4 נק')

נתון הקוד הבא:

```
public class G {
      public G() { func(); }
      private void func() {
            System.out.println("G");
      }
}
public class SubG extends G {
      public SubG() {
           super();
           func();
      }
      private void func() {
            System.out.println("SubG");
      }
      public static void main(String [] arr){
            G e = new SubG();
      }
}
```

מה יודפס? סמנו את התשובה הנכונה

1. המחרוזת G ואחריה המחרוזת SubG
2. המחרוזת SubG ואחריה המחרוזת G
3. המחרוזת G ואחריה המחרוזת G
4. המחרוזת SubG ואחריה המחרוזת SubG

נימוק

סעיף ח' (4 נק')

נתון הקוד הבא:

```
public class H {
      public void pub() { func();}
      private void func() {
            System.out.println("H");
      }
}
public class SubH extends H {
      public void func() {
            System.out.println("SubH");
      }
      public static void main(String [] arr){
            SubH e = new SubH();
            e.pub();
      }
}
```

סמנו את הטענה הנכונה:

1. הקוד לא יתקמפל
2. הקוד יתקמפל ותודפס המחרוזת H
3. הקוד יתקמפל ותודפס המחרוזת SubH
4. הקוד יתקמפל אך תתקבל שגיאת זמן ריצה

נימוק

שאלה 2 (20 נק')

שאלה זו עוסקת ב Design של מחלקות ומנשקים. בשאלה זו אין צורך לספק מימושים, אלא רק את חתימות הפונקציות, הנראות שבחרתם לכל פונקציה והאם הפונקציה היא פונקציה אבסטרקטית, פונקציה סטטית , או פונקציה דורסת. לצורך שאלה זו, לפונקציות שאינן אבסטרקטיות ניתן לספק מימוש ריק.

אם בחרתם שמחלקה תממש מנשק, או שתירש ממחלקה אחרת, ציינו זאת בצורה ברורה. ניתן גם להציג את התשובה בתור דיאגרמה של מחלקות/ מנשקים.

הסבירו בצורה ברורה ועניינית את מבנה המחלקות והמנשקים שבחרתם עבור בעיה זו. מותר להוסיף מחלקות או מנשקים לבחירתכם.

עליכם לעצב את מודול התביעות בחברת ביטוח כלשהי.

- על התוכנה לתמוך (כרגע) בסוגי התביעות הבאים: תביעת ביטוח רכב (AutoClaim), תביעת ביטוח דירה (HomeClaim), תביעת ביטוח בריאות (HealthClaim). ייתכן שבעתיד חברת הביטוח תרצה לתמוך בתביעות מסוגים נוספים.
- כל תביעת ביטוח כוללת את הפרטים הבאים:
  - claimID מטיפוס int – מזהה ייחודי לתביעה
  - policyHolderName מטיפוס מחרוזת – מייצג את שם המבוטח
  - amount מטיפוס double – מייצג את סכום התביעה
  - status מטיפוס מחרוזת - מייצג את סטטוס התביעה
- פרטים נוספים שכוללת תביעת הביטוח, אך ייחודיים לסוג התביעה:
  - תביעת ביטוח רכב תכיל גם מספר רכב carNumber כמחרוזת
  - תביעת ביטוח דירה תכיל גם כתובת address כמחרוזת
  - תביעת ביטוח בריאות תכיל גם שם קופת חולים clinic כמחרוזת
- ניתן להדפיס פרטי כל תביעה ע"י קריאה לפונקציה displayClaimDetails()
- קיים מנשק בשם ClaimProcess:

```
public interface ClaimProcess {
   public abstract void fileClaim();
   public abstract void updatedClaimStatus(String status);
}
```

בו מוגדרת הפונקציה fileClaim שמשמשת להגשת תביעה, והפונקציה updatedClaimStatus שמשמשת לעדכון סטטוס תביעה.

נרצה להשתמש בו כדי לאפשר הרצת קוד כדוגמת הקוד הזה: מדובר בפונקציה שמקבלת רשימת תביעות ומגישה אותן.

```
public void fileAllClaims(List<ClaimProcess> claims) {
   for (ClaimProcess claim : claims) {
   claim.fileClaim();
   }
}
```

- <u>שימו לב</u>: הלוגיקה להגשת תביעה היא ייחודית לסוג התביעה.

כתבו את תשובתכם בשני העמודים הבאים והקפידו להתייחס לכל הדרישות בשאלה.

תשובה :

הערות שניתנו בזמן הבחינה :
- צריך להתייחס לנראות השדות והתודות ולהתייחס לגישה לשדות.
- לא ניתן לשנות את המנשק ClaimProcess
- ניתן להוסיף מתודות שלא ציונו בדרישות
- הלוגיקה סטטוס תביעה זהה לכל התביעות (עדכון לשדה סטטוס)
- המנשק ClaimProcess מייצג תהליך של הגשת תביעה אחת מסוימת של לקוח מסוים. המתודה fileClaim מטפלת בלוגיקה של הגשת התביעה
- אין צורך לספק מימושים אבל ניתן ורצוי לנמק ולהסביר במילים את הפתרון שלכם

הצעה כללית לפתרון :

מחלקה אבסטרקטית AbstractClaim שimplements את המנשק ClaimProcess. כוללת שדות בנראות פרטית claimID, policyHolderName, amount, status ובנאי שמציב ערכים לשדות אלה. נוסיף gettersים לשדות הללו, ואת הפונקציה updateClaimStatus נממש (להיות setter לstatus).
לבסוף נוסיף מתודה פומבית עם מימוש (בסיסי) displayClaimDetails. המתודה fileClaim נשארת אבסטרקטית.

נוסיף שלוש מחלקות שיורשות מAbstractClaim – AutoClaim, HomeClaim, HealthClaim.

כל אחת מהן תכיל את השדה הנוסף המתאים לה, בנראות פרטית, ותקבל אותו כתוספת לבנאי.
הן כולן דורסות את הפונקציה displayClaimDetails, ובדריסה קוראות לsuper.displayClaimDetails() ומדפיסות בנוסף גם את השדה הנוסף שלהן. כדאי להוסיף גם getter לשדות שכל מחלקה מוסיפה.
הן גם דורסות (מוסיפות מימוש) לפונקציה fileClaim.

כל סוג חדש של תביעת ביטוח יירש מהמחלקה AbstractClaim, יגדיר את השדות הרלוונטיים בנראות פרטית, ייצור gettersים מתאימים יגדיר בנאי מתאים בהתאם לשדות. בנוסף ידרוס את displayClaimDetails כמו שתואר מעלה, ויוסיף מימוש לפונקציה fileClaim – כי היא ייחודית לכל סוג תביעה.

שאלה 3 (55 נק')

בשאלה זו נתייחס למנשק Expression<T> שמייצג ביטוי מטיפוס כלשהו המובע בעזרת משתנים. למנשק יש מתודה בשם evaluate שעושה הערכה (evaluation) של הביטוי, כלומר מחשבת את ערכו, על פי השמה assignment, למשתנים שמוגדרים בביטוי.

```
public interface Expression<T> {
      T evaluate(Map<String, T> as);
}
```

בשאלה זו אנחנו נממש את מחלקה BooleanExpression. המחלקה מייצגת ביטוי בוליאני, כלומר כזה המורכב אך ורק ממשתנים שיכולים לקבל את הערך true או false, מאופרטורים בוליאניים בינאריים "&&" (אופרטור "וגם"), "||" (אופרטור "או"), מאופרטור השלילה האונרי "!" ומסימני סוגריים.

לדוגמה: "(!a&&a) || (c&&b)" . בדוגמה הזו a, b, c הם משתנים בוליאניים.

הגדרה: ביטוי בוליאני פשוט הוא ביטוי שמכיל רק משתנה ללא אופרטורים בינאריים, למשל "a", או "!a".

הגדרה: ביטוי בוליאני מורכב הוא ביטוי שמכיל אופרטור בינארי ותתי ביטויים. למשל, "a || b" או "(c && (b || a)".

השדות הפרטיים של המחלקה נתונים להלן:

```
public class BooleanExpression implements Expression<Boolean> {

      private BooleanExpression left;
      private BooleanExpression right;
      private Operator op;
      private boolean isNegated;
      private String var;

      public static enum Operator {
            AND,
            OR;
      }
}
```

השדות left ו right מצביעים לתת הביטויים (מאותו הטיפוס BooleanExpression) משני צידי האופרטור op. השדה var מייצג את שם המשתנה.

אם הביטוי פשוט, השדות op, left, right יהיו null.
אם הביטוי מורכב, השדה var יהיה null.
השדה op הוא מטיפוס שהוא enum בעל 2 ערכים אפשריים.

הניחו כי קיימת דריסה של המתודה toString שמחזירה מחרוזת נוחה לקריאה של הביטוי הבוליאני. במחרוזת שתוחזר יהיה רווח בודד אחרי כל משתנה, אופרטור, סימן סוגריים, או סימן קריאה. דוגמה לפלט של המתודה: "((a && b) && ! (c && a !))".

הערות שניתנו במהלך הבחינה:

דוגמה: עבור הביטוי "(b||a)! && c", האובייקט BooleanExpression יכיל בשדה left את האובייקט שמתאים לביטוי "c", ובשדה right את האובייקט שמתאים לביטוי "(b||a)!". השדה op יכיל Operator.AND. השדה var יהיה null.

כמו כן, השדה isNegeted, שמציין האם הביטוי הבוליאני מכיל משמאלו את האופרטור "!", יהיה false.
בדוגמה שנתנו כאן בהבהרות, הערך של right.isNegated יהיה true.

המשתנה var הוא כל מחרוזת של תווים באנגלית (ללא רווחים).

בשאלה זו ניתן להוסיף מתודות עזר.

בכל סעיפי השאלה הניחו שקיים בנאי שמאתחל אובייקט כהלכה, ושכל המתודות מופעלות על אובייקט תקין.

בשאלה זו לא יהיו ביטויים מהצורה "a!!" או "(ביטוי מורכב)!!"

גם ל List וגם ל Set יש מתודת size() ומתודת addAll שמקבלת Collection

המתודות בסעיפים א,ד,ה הן מתודות של המחלקה BooleanExpression

סעיף א (10 נק')

השלימו את מימוש המתודה getVariables שמחזירה עצם מטיפוס Set<String> שיכיל את המחרוזות של שמות המשתנים השונים בביטוי.

לדוגמה, עבור האובייקט שמייצג את הביטוי "((! a && c) && ! (b && a))", בסוף הריצה של המתודה האוסף המוחזר יכיל שלוש מחרוזות "c", "b", "a".

```
public Set<String> getVariables() {

      Set<String> res = new HashSet<>();

      if (var != null) {
            res.add(var);
      }
      else {
            res.addAll(left.getVariables());
            res.addAll(right.getVariables());
      }
      return res;


ניתן היה גם לפתור ע"י קריאה ל toString() ופיצול המחרוזת המוחזרת לפי רווחים, ומעבר על כל המחרוזות שאינן אופרטורים או סוגריים.



}
```

סעיף ב (8 נק')

כיתבו מחלקה בשם ExpSizeComparator שתממש את המנשק Comparator<BooleanExpression>.
ממשו דריסה של המתודה int compare(T o1, T o2) שתבצע השוואה לפי מספר המשתנים השונים בשני הביטויים הבוליאניים. המתודה תחזיר 1- אם מספר המשתנים השונים בo1 קטן יותר מאשר ב o2, תחזיר 1 אם מספר המשתנים השונים o1 גדול יותר, ו- 0 אם מספר המשנים השונים שלהם שווה.

הערה שניתנה במהלך הבחינה: בחתימה של *compare*, עליכם להחליף את *T* בטיפוס המתאים

```
public class ExpSizeComparator implements Comparator<BooleanExpression>
{
      @override
      public int compare(BooleanExpression o1, BooleanExpression o2)
      {
            Set<String> o1Vars = o1.getVariables();
            Set<String> o2Vars = o2.getVariables();
            return Integer.compare(o1Vars.size(), o2Vars.size());
      }

}
```

סעיף ג (4 נק')

השלימו את הקוד בשורה החסרה כך שהרשימה תמויין לפי הסדר המוגדר בסעיף ב.

```
ArrayList<BooleanExpression> lst = new ArrayList<>();
// Code omitted: fill array with entries
Collections.sort(__lst, new ExpSizeComparator()_____________);
```

סעיף ד (12 נק')

ממשו את המתודה evaluate מתוך ההגדרה של המנשק Expression, שמבצעת הערכה של הביטוי לפי השמה של משתנים. המתודה תקבל כפרמטר Map של משתנים עם הערך הבוליאני שלהם ותחזיר true או false בהתאם לתוצאת הביטוי כאשר מציבים בו את ערכי המשתנים שנתונים ב Map. ניתן להניח שההשמה מכילה את כל המשתנים שמשתתפים בביטוי אבל אולי גם מחרוזות של שמות משתנים שלא נמצאים בביטוי.
לדוגמה, עבור הביטוי "((!a && c) && (b && a))" וההשמה {a:true,b:false,c:true,d:false} המתודה תחזיר false. בסעיף זה עליכם לכתוב גם את החתימה של המתודה.

```
@override
public Boolean evaluate(Map<String, Boolean> as) {

      Boolean result;

      if (var != null) {
            result = as[var];
      }
      else {
            Boolean lhs = left.evaluate(as);
            Boolean rhs = right.evaluate(as);
            switch (op) {
                  case AND: result = lhs && rhs; break;
                  case OR: result = lhs || rhs; break;
            }
      }
      if (isNegated) result = !result;
      return result;
}
```

סעיף ה (13 נק')

ממשו מתודת equals שמשווה בין שני ביטויים בוליאניים באופן סמנטי, כלומר בודקת האם שני הביטויים שקולים לוגית. כלומר אם b1 אובייקט שמייצג את הביטוי "(a && (b || c))" ו b2 אובייקט שמייצג את הביטוי "((a && b) || (c && a))", אז יתקיים b1.equals(b2) == true.
ישנן הרבה דרכים לביצוע חישוב זה באופן יעיל, אבל אנו נממש את הפתרון הפשוט, שעובר על כל ההשמות האפשריות של איחוד קבוצות המשתנים של שני הביטויים, ובודק האם לכל השמה אפשרית תוצאת שני הביטויים זהה. שימו לב שפתרון זה אינו יעיל כי מספר ההשמות האפשריות הוא מעריכי במספר המשתנים.

לנוחיותכם, קיימת פונקציה סטטית getAssignments שמחזירה רשימה של רשימות של בוליאניים שמיצגים את כל האפשרויות לערכים בוליאניים ל- n משתנים שונים. למשל עבור n=4 הפונקציה תחזיר את רשימת הרשימות הבאה

[[false, false, false, false],
[false, false, false, true],
[false, false, true, false],
…
[true, true, true, true]]

```
// Already implemented for us
private static List<List<Boolean>> getAssignments(int n) {

      List<List<Boolean>> assignments = new ArrayList<>();
      for (int k = 0; k < Math.pow(2, n); k++) {
            List<Boolean> assignment = new ArrayList<>();
            for (int i = n - 1; i >= 0; i--) {
                  assignment.add((k & (1 << i)) != 0);
            }
            assignments.add(assignment);
      }
      return assignments;
}
```

שימו לב שהחתימה של equals מקבלת כפרמטר אובייקט מטיפוס Object. לנוחיותכם הוספנו את הבדיקות הסטנדרטיות ש Eclipse מוסיף באופן אוטומטי בתחילת כל מימוש של equals. אם הבדיקות האלו עברו בהצלחה ניתן להניח שיש לנו אובייקט BooleanExpression.

```
@Override
public boolean equals(Object obj) {
      // First we verify that obj is a BooleanExpression
      if (this == obj)
            return true;
      if (obj == null)
            return false;
      if (getClass() != obj.getClass())
            return false;
      BooleanExpression other = (BooleanExpression) obj;

      // Your code goes here
```

```
        Set<String> vars = new Set<>();
        vars.addAll(getVariables());
        vars.addAll(other.getVariables());

        // Convert vars to a list so that iteration order will be
        // guaranteed to be consistent.
        List<String> varList = new ArrayList<>();
        varList.addAll(vars);

        List<List<Boolean>> assignments = getAssignments(varList.size());

        for (List<Boolean> assignment : assignments) {

            Map<String, Boolean> as = new HashMap<>();
            for (int i = 0; i < varList.size(); i++)
                as.put(varList[i], assignment[i]);

            if (evaluate(as) != other.evaluate(as)) return false;
        }

        return true;

    }
```

סעיף ו (8 נק')

נעם כתבה את המחלקה ArithmeticExpression שמייצגת ביטויים אריתמטיים

```
public class ArithmeticExpression implements Expression<Integer> {
      //Noam's implementation
      . . .
}
```

נעם רוצה לכתוב פונקציה שמשתמשת בפולימורפיזם ועושה הערכה לביטויים מסוגים שונים שניתנים ברשימה. למשל רשימה שכוללת גם ביטויים אריתמטיים וגם ביטויים בוליאניים.

היא כתבה את הפונקציה הבאה ששמה evaluateAll שמקבלת כקלט רשימה כזו וכן רשימה של השמות מתאימות. כלומר, אם ה Expression ה- i הוא בוליאני ההשמה ה-i תהיה לערכים מטיפוס Boolean, ואם הוא אריתמטי ההשמה ה-i תהיה לערכים מטיפוס Integer.

```
public static <T> void evaluateAll(List<Expression<T>> expList,
                                   List<Map<String,T>> assignList) {
      for (int i=0; i<expList.size(); ++i) {
            Expression<T> exp = expList.get(i);
            Map<String,T> assignment = assignList.get(i);
            System.out.println (exp.evaluate(assignment));
       }
}
```

האם הפונקציה שכתבה נעם מקיימת את הדרישה? הסבירו

הפונקציה לא מקיימת את הדרישה.
השימוש בטיפוס הגנרי T גורם לכך שהקומפיילר ידרוש שרשימת הקלט expList תכיל איברים מטיפוס אחיד Expression<T> עבור T ספציפי. כלומר, לא ייתכן שהרשימה שתועבר לפונקציה תכיל גם איברים מטיפוס Expression<Boolean> וגם איברים מטיפוס Expression<Integer>. טיעון דומה תופס גם לגבי הרשימה assignList.

public interface Map<K,V>

Modifier and Type	Method and Description
Boolean	containsKey(Object key) Returns true if this map contains a mapping for the specified key.
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.
V	getOrDefault(Object key, V defaultValue) Returns the value to which the specified key is mapped, or defaultValue 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. Returns the previous value associated with key, or null if there was no mapping for key.
V	remove(Object key) Removes the mapping for a key from this map if it is present (optional operation). Returns the previous value associated with key, or null if there was no mapping for key.
Collection<V>	values() Returns a Collection view of the values contained in this map.

public interface Set<E> extends Collection<E>

boolean	add(E e) Adds the specified element to this set if it is not already present.
boolean	addAll(Collection<? Extends E> c) Adds all of the elements in the specified collection to this set if they're not already present. Returns true if this set changed as a result of the call.
void	clear() Removes all of the elements from this set
boolean	contains(Object o) Returns true if this set contains the specified element.
boolean	isEmpty() Returns true if this set contains no elements.
boolean	remove(Object o) Removes the specified element from this set if it is present.

public interface List<E> extends Collection<E>

boolean	add(E e) Appends the specified element to the end of this list. Always returns true.
void	add(int index, E e) Inserts the specified element at the specified position in this list.
boolean	contains(Object o) Returns true if this list contains the specified element.
E	get(int index) Returns the element at the specified position in this list.
boolean	remove(Object o) Removes the first occurrence of the specified element from this list, if it is present. Returns true if this list contained the specified element

public final class String

char	charAt(int index) Returns the char value at the specified index.
boolean	contains(CharSequence s) Returns true if and only if this string contains the specified sequence of char values.
int	length() Returns the length of this string.
String[]	split(String regex) Splits this string around matches of the given regular expression.
char[]	toCharArray() Converts this string to a new character array.

עמוד נוסף למקרה הצורך