בית ספר אפי ארזי למדעי המחשב
המרכז הבינתחומי הרצליה

מבוא למדעי המחשב

מבחן גמר, מועד א'

2009

הנחיות:

- זמן המבחן: שלוש שעות.
- זמן הבחינה מוגבל, ויש לעבוד ביעילות. אם נתקעים בסעיף מסוים, כדאי לעזוב אותו ולרוץ הלאה.
- חומר סגור. השימוש במחשבים או במחשבונים אסור.
- ענו על כל השאלות על טופס המבחן. מומלץ להשתמש במחברות הבחינה כטיוטא. ניתן גם לכתוב תשובות במחברת הבחינה ולהפנות אליהן בצורה ברורה ומפורשת מטופס המבחן. חובה למחוק באופן ברור כל דבר שאינכם רוצים שנבדוק.
- בגמר הבחינה יש להחזיר את טופס המבחן + מחברת המבחן + כל דפי העזר הנלווים.
- אם תרגישו צורך לעשות הנחה מסוימת כדי לענות על כל שאלה שהיא, ניתן לעשות זאת, כל עוד ההנחה היא סבירה ומנוסחת בדיוק ובבהירות.
- אם אתם לא מסוגלים לתת תשובה מלאה לשאלה מסוימת, תנו תשובה חלקית. תשובה נכונה באופן חלקי תקבל ניקוד חלקי.
- התשובות חייבות להיות קצרות ולעניין. כתב היד חייב להיות קריא וברור. תשובות לא קריאות יקבלו ציון 0.
- אם התבקשתם לכתוב תכנית שאמורה לפעול על קלט מסוים, אזי התכנית לא אמורה לבדוק אם הקלט תקין, אלא אם כן נאמר כך בשאלה במפורש.
- התכניות שתכתבו תישפטנה, בין היתר, לפי האורך והאלגנטיות שלהן. תוכניות ארוכות או מסורבלות ללא צורך יקבלו פחות נקודות, אפילו אם הן ממלאות את המשימה שהוגדרה בשאלה.
- כל החומרים להם תזדקקו במהלך המבחן מתועדים בדפי העזר שקיבלת.
- לא יורדו נקודות על טעויות סינטקס טריוויאליות.

בהצלחה!

חלק א' (6 שאלות, 2 נקודות כל שאלה)

1. מה מדפיס קטע הקוד הבא:

```
String b = new String("a");
if(b == "a")
  System.out.print("a");
if (b.equals("a"))
  System.out.print("b");
```

א. a

ב. b (*)

ג. ab (*)

ד. כלום, הקטע לא יעבור קומפילציה

(התשובה הנכונה היא (ב), אבל כדי להיות נחמדים קיבלנו גם את (ג))

2. מחסנית (stack) היא מבנה נתונים שבו מתקיים הכלל

א. first in, first out

ב. last in, last out

ג. first in, last in

ד. first out, last out

ה. אף תשובה לא נכונה (*)

3. נתון מערך ממוין בגודל מסוים, ונניח שאנו משתמשים בשיטה המהירה ביותר שידועה במדעי המחשב לחיפוש ערכים במערך הזה. מסתבר שבשיטה הזאת, זמן החיפוש של איבר כלשהו הוא לכל היותר c שניות. כעת מכפילים את גודל המערך. זמן החיפוש של איבר כלשהו במערך החדש יהיה לכל היותר:

א. c

ב. 2c

ג. c+1 (*)

ד. log (2c)

ה. אי אפשר לענות בלי לדעת את גודל המערך המקורי.

4. איזה יתרונות נותנת ה virtual machine של שפת Java?

א. לא צריך לקמפל מחדש את קוד המקור כל פעם שיש פלטפורמת חומרה חדשה

ב. הקוד בטוח יותר

ג. הקוד יעיל יותר

ד. (א) ו (ב) נכונות (*)

ה. (א) (ב) (ג) נכונות

ו. אף תשובה לא נכונה

5. התבונן במתודה הבאה:

```
public static int f(int x) {
  if (x == 0) return 1;
  return f(x-1)+f(x-1);
}
```

מה מוחזר ע"י f(10) ?

א. 1

ב. 10

ג. 100

ד. 1024 (*)

6. כשאומרים שלשפת Java יש virtual method calling מתכוונים לכך ש ...

א. תכניות java יכולות לקרוא למתודות של מכונה וירטואלית שפועלת לצד השפה

ב. בתכניות java יש אבחנה בין פרמטרים פורמליים לפרמטרים וירטואליים

ג. בתכניות java המתודה המתבצעת נקבעת לפי סוג האובייקט ב run-time (*)

ד. תכניות java פועלות במרחב וירטואלי שגודלו כגודל האינטרנט

ה. אף תשובה לא נכונה

* * *

<u>שימו לב:</u> פתרונות התכנות שיינתנו מכאן ואילך אינם יחידים. כרגיל, יש יותר מתשובה נכונה אחת לרוב השאלות שדורשות כתיבת קוד. יחד עם זאת, תשובות ארוכות ומסורבלות באופן משמעותי מאלה שניתנות כאן גררו הורדת נקודות.

<u>חלק ב'</u>

נתונה מחלקה בשם Time שמייצגת נקודות זמן. כל נקודת זמן מיוצגת ע"י זוג מספרים: שעות (hours) ודקות (minutes). למשל, השעה שתיים וחמש דקות אחה"צ מיוצגת ע"י הזוג (14,5) שייצוגו הטקסטואלי הוא "14:05". קיראו את תיעוד המחלקה בדף העזר ששמו "<u>מחלקת Time</u> " וענו על השאלות הבאות.

7. (5 נקודות) כיתבו מימוש של הקונסטרקטור Time. הקונסטרקטור בונה עצם מסוג Time, אלא אם כן הארגומנטים hours או minutes שגויים. במקרה זה הקונסטרקטור זורק IllegalArgumentException.

```
public Time(int hours, int minutes) {
   if (hours < 0 || hours > 23 || minutes < 0 || minutes > 59)
      throw new IllegalArgumentException();
   this.hours = hours;
   this.minutes = minutes;
}
```

8. (5 נקודות) כתבו מימוש של המתודה after. המתודה מקבלת עצם מסוג Time ומחזירה true אם העצם גדול מהעצם הנוכחי במונחי זמן, ו false אחרת. למשל, 17:05 גדול מ 3:15. כדי להקל על המימוש, אנו מניחים ששתי נקודות הזמן שייכות לאותן 24 שעות (אותו תאריך).

```
public boolean after(Time other) {
    if (hours == other.hours(
        return minutes > other.minutes;
    return hours > other.hours;
}
```

9. (5 נקודות) כתבו מימוש של המתודה toString. המתודה מחזירה String שמייצג את נקודת הזמן הנוכחית בתבנית "hh:mm" עם ריפוד מתאים של אפסים. למשל, (14,5) מיוצג ע"י "14:05" ו- (7,3) ע"י "07:03".

```
public String toString() {
  return ((hours < 10) ? "0" + hours : hours) + ":" +
         ((minutes < 10) ? "0" + minutes : minutes);
}
```

10. (5 נקודות) בספריית המחלקות של Java קיים ממשק (interface) ששמו Comparable. הממשק מכיל מתודה ששמה compareTo, שתפקידה לממש את היחס "גדול מ..." של שני עצמים. ענה על השאלות הבאות:

א. תאר בקיצור נמרץ מהו interface ולמה הוא מיועד.

תשובה: אפשר לתאר Interface כמחלקה אבסטרקטית לגמרי שמכילה רק חתימות של מתודות. Interface מגדיר חוזה: כל מחלקה שמממשת אותו חייבת לממש את כל המתודות שמופיעות ב interface.

ב. כמו שראינו, המעצב של מחלקת Time החליט לממש את היחס "גדול מ..." ע"י המתודה after. מעצב אחר יעץ לו: " עדיף לממש את זה בעזרת Comparable". תאר שני יתרונות של ההמלצה הזאת.

תשובה: המימוש יהיה יותר סטנדרטי, ולכן הקוד יהיה יותר קריא. כמו כן, מחלקות אחרות שבנויות לפעול על עצמים שמממשים את Comparable תוכלנה לפעול על עצמים מסוג Time.

<u>חלק ג'</u>

נתונה מחלקה בשם Schedule שמייצגת לוח זמנים ממוין. לדוגמא, נניח תחנת רכבת ממנה יוצאות רכבות כל חצי שעה בין השעות 12 בצהריים עד 3 אחה"צ. את לוח הזמנים של התחנה הזאת ניתן לייצג ע"י עצם מטיפוס Schedule שנראה כך:
(12,0),(12,30),(13,0),(13,30),(14,0),(14,30),(15,0).
קיראו את דף העזר ששמו <u>"מחלקת Schedule, מימוש ע"י מערך"</u> ואת דף העזר ששמו <u>"דוגמת שימוש במחלקת Schedule"</u> (דף העזר האחרון) וענו על השאלות הבאות.

<u>שימו לב: בכל השאלות הבאות יש להניח שמחלקת Time קיימת וממומשת במלואה.</u>

11. (4 נקודות) כיתבו מימוש של הקונסטרקטור Schedule. הקונסטרקטור בונה מערך ריק של 100 נקודות זמן (עצמים מסוג Time) ומכניס למערך את נקודת הזמן היחידה (0,0). שימו לב: במציאות היינו מצפים לקבל מערך מאותחל ריק. יחד עם זאת, הכנסת נקודת הזמן (0,0) מקלה על המשימות הבאות במבחן ולכן הכנסנו אותה לסיפור.

```
public Schedule() {
  times = new Time[100];
  times[0] = new Time(0,0);
  numTimes = 1;
}
```

12. (7 נקודות) כיתבו מימוש של המתודה nextTimeAfter. המתודה מקבלת כקלט נקודת זמן נתונה (עצם מסוג Time), ומחזירה את נקודת הזמן בלוח הזמנים הקרובה אליה ביותר מלמעלה. למשל, אם לוח הזמנים הוא (0,0),(9,34),(14,12),(15,8) ונקודת הזמן הנתונה היא (13,55), המתודה מחזירה את נקודת הזמן (14,12). בקונטקסט של לוח זמני רכבות, המתודה מחזירה את זמן היציאה של הרכבת הבאה, בהינתן זמן כלשהו.

```
public Time nextTimeAfter(Time t) {
  int i = 0;
  while (i < numTimes && t.after(times[i]))
    i++;
  return times[i];
}
```

13. (6 נקודות) כיתבו מימוש של המתודה toString. המתודה מחזירה String שמייצג את לוח הזמנים. למשל, אם לוח הזמנים הוא (0,0),(9,34),(14,12),(15,8) המתודה מחזירה "00:00 09:34 14:12 15:08" (נקודות זמן מופרדות ע"י רווחים).

```
  public String toString() {
    String s = "";
    for (int i = 0; i < numTimes ; i++)
      s = s + times[i] + " ";
    return s;
  }
}
```

תשובה שלמה צריכה לעשות trim של הרווח האחרון אחרי הלולאה, אבל החלטנו לא לדרוש את זה.

<u>ענו על שאלה 14 או 15:</u>

14. (10 נקודות) כיתבו מימוש של המתודה addTime. המתודה מוסיפה ללוח הזמנים נקודת זמן נתונה, תוך שמירה על סדר הזמנים. למשל, אם לוח הזמנים הנוכחי הוא (0,0),(9,34),(15,8) ונקודת הזמן הנתונה היא (14,12), המתודה תשנה את לוח הזמנים ל: (0,0),(9,34),(14,12),(15,8) שימו לב: אפשר להניח שבלוח הזמנים יש לפחות נקודת זמן אחת: (0,0). כמו כן אפשר להניח שבמערך יש מספיק מקום להכיל גם את נקודת הזמן החדשה.

```
public void addTime(Time t) {
   int pos = 0;
   while (pos < numTimes && t.after(times[pos]))
     pos++;
   for(int j = numTimes ; j > pos ; j-- )
     times[j] = times[j-1];
   times[pos] = t;
   numTimes++;
 }
```

15. (10 נקודות) כיתבו מימוש של המתודה deleteTime. המתודה מבטלת נקודת זמן נתונה מלוח הזמנים. אפשר להניח שנקודת הזמן הזאת קיימת בלוח הזמנים.

```
  public void deleteTime(Time t) {
    int pos = 0;
    while ((pos < numTimes) && !(t.equals(times[pos])))
      pos++;
    for(int j = pos ; j < numTimes ; j++)
      times[j] = times[j+1];
    numTimes--;
  }
```

16. (3 נקודות) כפי שראינו, המעצב של מחלקת Schedule החליט לייצר לוח זמנים בגודל קבוע של 100. האדם שהזמין את כתיבת המחלקה הזאת רואה את העיצוב הנוכחי ומעיר כדלקמן: "טוב, תשאיר את זה כמו שזה, אבל תאפשר גם לקוד ה client לייצר לוחות זמנים בכל גודל אחר שה client רוצה". איך אפשר לממש את הדרישה הזאת? הסבר בקיצור נמרץ, אין צורך לכתוב קוד.

תשובה: צריך להוסיף (להעמיס) למחלקה קונסטרקטור נוסף, נניח Schedule(int maxNumTimes), שמגדיר מערך בגודל maxNumTimes.

שימו לב: תשובה מלאה חייבת לציין את הצורך <u>להוסיף</u> עוד קונסטרקטור, ולא רק לשנות את הקונסטרקטור הנוכחי.

17. (3 נקודות) המימוש של המתודה addTime הוא נאיבי, כי הנחנו שבמערך יש מספיק מקום להכיל את נקודת הזמן החדשה. מה צריך לשנות או להוסיף כדי שהתכנית תמשיך לרוץ גם אם המערך מלא? ענה בקיצור נמרץ, אין צורך לכתוב קוד.

תשובה: המתודה צריכה קודם כל לבדוק את גודל המערך. אם הגודל הוא 100, המתודה צריכה לזרוק exception שכותבת הודעה מתאימה.

<u>חלק ד'</u>

הניחו שהמחלקה Schedule שהצגנו בחלק ג' של המבחן לא קיימת. במקומה קיימת מחלקה Schedule אחרת שיש לה בדיוק אותו ממשק (API) ותיעוד כמו קודם. קיראו את תיאור המחלקה בדף העזר ששמו <u>"מחלקת Schedule, מימוש ע"י רשימה מקושרת"</u> וענו על השאלות הבאות.

<u>בכל השאלות הנותרות במבחן יש להניח שמחלקות Time ו TimeNode קיימות וממומשות במלואן.</u>

<u>שימו לב:</u> חלק מהשאלות הבאות זהות לגמרי בניסוחן לשאלות של חלק ג'. ההבדל הוא שעכשיו צריך לענות עליהן בקונטקסט של רשימה מקושרת.

18. (10 נקודות) כיתבו מימוש של המתודה nextTimeAfter. המתודה מקבלת כקלט נקודת זמן נתונה (עצם מסוג Time), ומחזירה את נקודת הזמן בלוח הזמנים הקרובה אליה ביותר מלמעלה. למשל, אם לוח הזמנים הוא (0,0),(9,34),(14,12),(15,8) ונקודת הזמן הנתונה היא (13,55), המתודה מחזירה את נקודת הזמן (14,12). בקונטקסט של לוח זמני רכבות, המתודה מחזירה את זמן היציאה של הרכבת הבאה, בהינתן זמן כלשהו.

```
public Time nextTimeAfter(Time t) {
  TimeNode p = list;
  while ((p != null) && t.after(p.getTime())) {
    p = p.getNext();
  }
  return p.getTime();
}
```

19. (10 נקודות) כיתבו מימוש של המתודה toString. המתודה מחזירה String שמייצג את לוח הזמנים. למשל, אם לוח הזמנים הוא (0,0),(9,34),(14,12),(15,8) המתודה מחזירה "00:00 09:34 14:12 15:08" (נקודות זמן מופרדות ע"י רווחים).

```
public String toString () {
    String result = "";
    TimeNode p = list;
    while (p != null) {
       result += p.getTime() + " ";
       p = p.getNext();
    }
    return result;
}
```

תשובה שלמה צריכה לעשות trim של הרווח האחרון אחרי הלולאה, אבל החלטנו לא לדרוש את זה.

<u>ענו על שאלה 20 או 21:</u>

20. (12 נקודות) כיתבו מימוש של המתודה addTime. המתודה מוסיפה ללוח הזמנים נקודת זמן נתונה, תוך שמירה על סדר הזמנים. למשל, אם לוח הזמנים הנוכחי הוא (0,0),(9,34),(15,8) ונקודת הזמן הנתונה היא (14,12), המתודה תשנה את לוח הזמנים ל: (0,0),(9,34),(14,12),(15,8)
שימו לב: אפשר להניח שבלוח הזמנים יש לפחות נקודת זמן אחת: (0,0). כמו כן אפשר להניח שבמערך יש מספיק מקום להכיל גם את נקודת הזמן החדשה.

```
public void addTime (Time t) {
   TimeNode p = list;
   TimeNode prev = list;
   while ((p != null) && t.after(p.getTime())) {
      prev = p;
      p = p.getNext();
   }
   TimeNode newTimeNode = new TimeNode(t);
   prev.setNext(newTimeNode);
   newTimeNode.setNext(p);
}
```

21. (12 נקודות) כיתבו מימוש של המתודה deleteTime. המתודה מבטלת נקודת זמן נתונה מלוח הזמנים. אפשר להניח שנקודת הזמן הזאת קיימת בלוח הזמנים.

```
public void deleteTime(Time t) {
   TimeNode p = list;
   TimeNode prev = list;
   while ((p != null) && !(t.equals(p.getTime()))) {
     prev = p;
     p = p.getNext();
   }
   prev.setNext(p.getNext());
}
```

22. (3 נקודות) כפי שראינו, לשתי הגרסאות של מחלקת Schedule יש בדיוק אותו ממשק (API). תאר יתרון חשוב אחד של האחידות הזאת.

תשובה: ניתן לשנות את המימוש של Schedule בלי שום צורך לגעת בקוד של מחלקות אחרות שמשתמשות בשירותים של Schedule.

<u>טעויות נפוצות בחלק ד' (שאלות 18 עד 22):</u>

1. חייבים להגדיר משתנה שרץ על הרשימה המקושרת. בהרבה תשובות עבדו ישירות על השדה list, דבר שהורס את הרשימה.

2. תנאי קצה: בהרבה תשובות לא טרחו לבדוק שמגיעים לסוף הרשימה (null). תשובות אחרות בדקו זאת, אך בתוך הלולאה עשו פעולה כמו `p.getNext().getNext()`, שיכולה להוביל ל null pointer.

3. תשובות שכללו קוד מסורבל, לא אלגנטי, ולא קריא גררו הורדת נקודות.

(סוף המבחן)

<u>שימו לב: המבחן נבדק באופן ליברלי וסלחני ביותר. לכן, אנו ממליצים לחשוב פעמיים לפני שמערערים. כל ערעור שאינו טכני (ספירת נקודות לא נכונה) יגרור בדיקה מקיפה של כל המבחן, ולא רק של השאלה עליה ערערתם. בפרט, כל קטע קוד שלא עובד כראוי (פרט לטעויות סינטקס קלות) יספוג הורדת נקודות משמעותית, דבר שלא עשינו בבדיקה הנוכחית.</u>

דף עזר: מחלקת Time

מחלקת Time מייצגת נקודות זמן. כל נקודת זמן מיוצגת ע"י זוג מספרים: שעות (hours) ודקות (minutes). למשל, השעה שתיים וחמש דקות אחה"צ מיוצגת ע"י הזוג (14,5) שייצוגו הטקסטואלי הוא "14:05".

```
// Represents a point in time.

public class Time {
   private int hours, minutes;

   // Constructs a Time object from given hours and minutes
   // arguments. If hours or minutes contain illegal values,
   // throws an IllegalArgumentException.
   public Time(int hours, int minutes) {
     // Code omitted.
   }

   public int getHours() {
      return hours;
   }

   public int getMinutes() {
      return minutes;
   }

   // Returns true if this time is the same as the other time
   // and false otherwise.
   public boolean equals(Time other) {
     // Code omitted.
   }

   // Compares an other time to this time. If the other time is greater
   // than this time, returns true. Otherwise returns false.
   public boolean after(Time other) {
     // Code omitted.
   }

   // A textual representation of this time, in the form hh:mm.
   // For example, if hours=7 and minutes=13, returns the string "07:13".
   public String toString() {
     // Code omitted.
   }
}
```

דף עזר: מחלקת Schedule, מימוש ע"י מערך (array)

מחלקת Schedule מייצגת לוח זמנים ממוין. לדוגמא, נניח תחנת רכבת ממנה יוצאות רכבות כל חצי שעה בין השעות 12 בצהריים עד 3 אחה"צ. את לוח הזמנים של התחנה הזאת ניתן לייצג ע"י עצם מטיפוס Schedule שנראה כך:
(12,0),(12,30),(13,0),(13,30),(14,0),(14,30),(15,0).

```
// Represents an ordered list of time points.

public class Schedule {
  private Time[] times;
  private int numTimes;

  // Constructs a schedule as an array of 100 Time objects, and inserts
  // to the schedule a single time point: (0,0)
  public Schedule() {
    // Code omitted.
  }

  // Adds a given time to the right location in this schedule, keeping it
  // always sorted. For example, if the schedule is (0,0),(9,34),(15,8)
  // and the given time is (14,12), the schedule becomes
  // (0,0),(9,34),(14,12),(15,8).
  public void addTime(Time t) {
    // Code omitted.
  }

  // Finds the time in the schedule which is closest to a given
  // time "from above". For example, if the schedule is
  // (0,0),(9,34),(14,12),(15,8) and the given time is (13,55),
  // returns the time (14:12).
  // Assumes that there exists at least one time in the schedule
  // which is after the given time.
  public Time nextTimeAfter(Time t) {
    // Code omitted.
  }

  // Deletes a given time from the schedule.
  // Assumes that the given time exists in the schedule.
  public void deleteTime(Time t) {
    // Code omitted.
  }

  // Returns a textual description of this schedule. For example,
  // if the schedule is (0,0),(9,34),(14,12),(15,8), returns
  // the string 00:00 09:34 14:12 15:08 (the time points are
  // padded with zeroes when needed and separated by spaces).
  public String toString() {
    // Code omitted.
}
```

דף עזר: מחלקת Schedule, מימוש ע"י רשימה מקושרת (linked list)

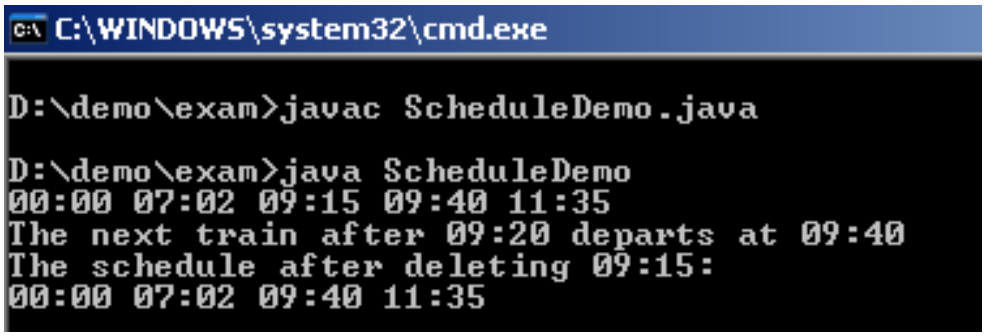
הניחו שהמחלקה Schedule שהצגנו קודם לא קיימת. המחלקה הזאת מחליפה אותה. שימו לב: לשתי המחלקות יש בדיוק אותו ממשק (interface), ולכן אנו לא חוזרים כאן על תיעוד המתודות.

```
// Represents an ordered list of time nodes.

public class Schedule {
   private TimeNode list; // Points to the first node in the linked list.

   // Constructs a schedule, implemented as a linked list of TimeNode
   // objects, and inserts into the list the single time point (0,0).
   public Schedule() {
      this.list = new TimeNode(new Time(0,0));
   }

   public void addTime (Time t) {
     // Code omitted.
   }

   public Time nextTimeAfter(Time t) {
     // Code omitted.
   }

   public void deleteTime(Time t) {
     // Code omitted.
   }

   public String toString () {
     // Code omitted.
   }
}
```

מחלקת TimeNode (הקוד של מחלקה זאת ניתן במלואו)

```
// Represents a node in a linked list. The node holds a point of time.

public class TimeNode {
  private Time time;
  private TimeNode next;

  public TimeNode (Time time) {
    this.time = time;
    this.next = null;
  }

  public Time getTime() { return time;  }

  public void setTime(Time time) { this.time = time;  }

  public TimeNode getNext() { return next; }

  public void setNext(TimeNode timeNode) { this.next = timeNode;  }

}
```

דף עזר: דוגמת שימוש במחלקת Schedule

הקוד שלפנינו מדגים את השימוש במחלקת Schedule ע"י בניית לוח זמנים של רכבות יוצאות והרצת מספר פעולות טיפוסיות. הקוד בונה לוח זמנים, מדפיס אותו, מדפיס את זמן היציאה של הרכבת הבאה אחרי השעה 09:15, מבטל את נקודת הזמן (9,15) מלוח הזמנים, ושוב מדפיס את הלוח.

```
public class ScheduleDemo {
  public static void main (String[] args) {
    Schedule trainDepartures = new Schedule();
    trainDepartures.addTime(new Time(7,2));
    trainDepartures.addTime(new Time(11,35));
    trainDepartures.addTime(new Time(9,15));
    trainDepartures.addTime(new Time(9,40));
    System.out.println(trainDepartures);
    Time currentTime = new Time(9,20);
    System.out.println("The next train after " + currentTime +
                       " departs at " +
                       trainDepartures.nextTimeAfter(currentTime));
    Time cancelledTime = new Time(9,15);
    trainDepartures.deleteTime(cancelledTime);
    System.out.println("The schedule after deleting " +
                       cancelledTime + ":");
    System.out.println(trainDepartures);
  }
}
```