קרא בעיון את ההנחיות שלהלן:

- **בבחינה יש שלוש שאלות. עליכם לענות על כולן.**
- כל התכניות צריכות להיות מתועדות היטב. יש לכתוב תחילה **בקצרה** את האלגוריתם וכל הסבר נוסף הדרוש להבנת התכנית. יש לבחור בשמות משמעותיים למשתנים, לפונקציות ולקבועים שבתכנית. תכנית שלא תתועד כנדרש לעיל תקבל לכל היותר 85% מהניקוד.

1. יש להקפיד לכתוב את התכניות בצורה מבנית ויעילה. **תכנית לא יעילה לא תקבל את מלוא הנקודות.**
2. **אם ברצונכם להשתמש בתשובתכם בשיטה או במחלקה הכתובה בחוברת השקפים, אין צורך שתעתיקו את השיטה או את המחלקה למחברת הבחינה. מספיק להפנות למקום הנכון, ובלבד שההפניה תהיה מדויקת (פרמטרים, מיקום וכו').**
3. **אין להשתמש במחלקות קיימות ב- Java, חוץ מאלו המפורטות בשאלות הבחינה.**

- **יש לשמור על סדר; תכנית הכתובה בצורה בלתי מסודרת עלולה לגרוע מהציון.**
- בכל השאלות ניתן להניח כי הקלט תקין, אלא אם כן מצוין אחרת.
- בכתיבת התכניות יש להשתמש **רק** במרכיבי השפה שנלמדו בקורס זה.

- **שימו לב, התשובות לשאלות 1 ו- 3 צריכות להיכתב על גבי שאלון הבחינה. רק התשובה לשאלה 2 תיכתב במחברת הבחינה.**

חומר עזר המותר בשימוש הוא:

1. חוברת השקפים של הקורס

2. ספר הלימוד Java Software Solutions

אסור להשתמש במחשב מכל סוג שהוא!

שאלה 1 - 60 נקודות

בשאלה זו נספק לכם מספר מחלקות שלמות ומספר מחלקות שאתם תצטרכו להשלימן לפי המתואר להלן.

השיטות הכתובות בתיאור המחלקה ממומשות כבר. כלומר, כשכתוב {...} הכוונה היא שגוף השיטה כתוב ואינכם צריכים להשלימו. אתם יכולים להניח שהשיטה נכונה ומתבצעת כמתואר. שימו לב, הפתרונות לשאלות אינם תלויים זה בזה, וניתן לפתור כל אחת מהשאלות, גם אם לא פתרתם את קודמותיה.

הנהלת הבנק הממלכתי לישראל רוצה למחשב את הסניפים שלה. עבור כל סניף היא מעוניינת לשמור את חשבונות הלקוחות שלה.

לצורך כך נגדיר שלוש מחלקות:

המחלקה Customer שתייצג לקוח;

המחלקה Account שתייצג חשבון בנק;

המחלקה Bank שתייצג סניף בנק ובו כל החשבונות של הלקוחות.

המחלקה Customer מייצגת לקוח -

למחלקה Customer יש את התכונות הפרטיות (instance variables) הבאות:

- long _id – שמייצגת את מספר תעודת הזהות של הלקוח;
- String _firstName – שמייצגת את השם הפרטי של הלקוח.
- String _familyName – שמייצגת את שם המשפחה של הלקוח.

למחלקה Customer הוגדרו שני בנאים (constructors):

- האחד - בנאי המקבל שלושה פרמטרים (שם פרטי, שם משפחה ומספר תעודת זהות) של הלקוח.

```
public Customer(long id, String first, String family)
```

- השני - בנאי העתקה המקבל לקוח אחר, ומעתיק את ערכיו.

```
public Customer (Customer c)
```

כמו כן, הוגדרו פעולות האחזור:

```
getID(), getFirstName(), getFamilyName().
```

והפעולות הקובעות:

```
setFirstName(String s), setFamilyName(String s)
```

(אין פעולה שיכולה לשנות את מספר הזהות של לקוח)

שיטה נוספת במחלקה היא השיטה equals המקבלת כפרמטר לקוח מסוים ובודקת אם הוא זהה ללקוח שמיוצג על ידי האובייקט עליו מופעלת השיטה.

סעיף א (12 נקודות):

לפניכם המימוש ב- Java של המחלקה Customer. השלימו את הדרוש, במקומות שמסומנים בקווים. **מספר השורות הריקות לא זהה בהכרח למספר השורות בפתרון.**

```
public class Customer
{
    private _________ _id;
    private String ____________________________________;

    public Customer(____________________________________)
    {
        ___________ = id;
        ___________ = first;
        ___________ = family;
    }

    public Customer (Customer c)
    {
        ____________________________;
        ____________________________;
        ____________________________;
    }

    public long getId() { ...  }
    public String getFirstName(){ ...  }
    public String getFamilyName(){ ...  }

    public void ______________________(String s)
    {
        _firstName = s;
    }
    public void setFamilyName (String s) { ...  }

    public boolean equals(Customer c)
    {
        ____________________________________________________
        ____________________________________________________
        ____________________________________________________
        ____________________________________________________
        ____________________________________________________
        ____________________________________________________
        ____________________________________________________
    }
}
```

המחלקה Account מייצגת חשבון בנק.

למחלקה Account התכונות הפרטיות (instance variables) הבאות:

- long _accountNum – מספר חשבון
- Customer _customer – פרטי הלקוח
- double _balance – סכום היתרה בחשבון
- double _interest – גובה הריבית של החשבון

כמו כן, הוגדר קבוע פרטי שלם בשם FEE בגודל 3 שמייצג את **העמלה** שיש על פעולת משיכה.

למחלקה Account יש בנאי אחד שמקבל מספר חשבון, לקוח וריבית, ומעדכן את השדות המתאימים. היתרה בחשבון חדש היא תמיד בסכום 0 ₪.

במחלקה הוגדרו פעולות האחזור לפי השמות המקובלים.

כמו כן הוגדרו השיטות הבאות:

- setInterest הקובעת את גובה הריבית
- deposit המקבלת סכום ומפקידה אותו בחשבון (מעדכנת את היתרה)
- withdraw המקבלת סכום ומושכת אותו מהחשבון, (מורידה גם את העמלה מהיתרה)
- addInterest המעדכנת את היתרה לפי הריבית.

לדוגמא, אם היתרה הייתה 1000 ₪, והריבית של אותו חשבון היא 0.005 (כלומר חצי אחוז), אז לאחר העדכון, היתרה תהיה 1005 ₪.

סעיף ב (16 נקודות):

לפניכם המימוש ב- Java של המחלקה Account. השלימו את הדרוש, במקומות שמסומנים בקווים. **מספר השורות הריקות לא זהה בהכרח למספר השורות בפתרון.**

```
public class Account
{
    private ___________________________ = 3;

    ____________________________________________________;
    ____________________________________________________;
    ____________________________________________________;
    ____________________________________________________;
```

```
    public Account(long num, Customer c, double interest)
    {
        ________________________________________;
        ________________________________________;
        ________________________________________;
        ________________________________________;
    }

    public long getAccountNum()
    {
       return ______________________;
    }

    public ___________________ getCustomer()
    {
        ________________________________________;
    }

    public double getBalance()  { ... }

    public double getInterest() { ... }

    public void setInterest ______________________ { ... }


    public void deposit (double amount)
    {
        ________________________________________;
    }

    public void withdraw (double amount)
    {
        ________________________________________;
    }

    public void addInterest ()
    {
        ________________________________________;
    }

}
```

המחלקה Bank מייצגת סניף בנק ובו חשבונות בנק שונים.

הייצוג נעשה על-ידי מערך ששומר את רשימת החשבונות. התכונות במחלקה הן:

- מערך של החשבונות — Account [] _bank
- מספר החשבונות בסניף — int _noOfAccounts

כמו כן קיים במחלקה שלושה קבועים מספריים:

1. קבוע שלם המציין את המספר המקסימלי של חשבונות בסניף – 1000.
2. קבוע ממשי המציין את ריבית הזכות 0.005
3. קבוע ממשי המציין את ריבית החובה 0.007

החשבונות (כלומר האובייקטים מהמחלקה Account) נמצאים במערך ברצף, **ללא "חורים" מתחילת המערך. המערך צריך להישאר כך (ללא חורים) לאחר כל פעולה.**

סעיף ג (32 נקודות):

הנה המימוש ב- Java של המחלקה Bank. מימוש המחלקה כולל רק את הסעיפים שלהלן:

1. הגדרת הקבועים של המחלקה.
2. הגדרת התכונות של המחלקה.
3. בנאי שמאתחל את תכונות המחלקה כך שהמערך יהיה בגודל מקסימלי.
4. שיטה (addAccount) בוליאנית המוסיפה חשבון בנק לסניף. השיטה מקבלת את השם הפרטי, שם המשפחה ומספר תעודת הזהות של הלקוח, וכן את מספר החשבון. השיטה מחזירה ערך true אם ההוספה התבצעה כשורה, אם לא, השיטה תחזיר false.
5. שיטה (accountCustomer) שמקבלת לקוח ומחזירה מספר האומר כמה חשבונות בנק יש ללקוח זה.
6. שיטה (wealthy) שמחזירה את הלקוח העשיר ביותר. **בשיטה זו (בלבד) אתם יכולים להניח כי לכל לקוח יש חשבון אחד בלבד בסניף.**
7. שיטה (setInterestInAccounts) שמעדכנת את הריבית בכל החשבונות שבסניף. אם החשבון הוא בזכות, מתעדכנת הריבית בחשבון לפי הקבוע של ריבית זכות, ואז מעדכנים את היתרה בחשבון. אם החשבון הוא בחובה, מתעדכנת הריבית בחשבון לפי הקבוע של ריבית החובה, ואז מעדכנים את היתרה בחשבון (שימו לב שאז מורידים מהסכום את הריבית).
   לדוגמא,
   אם בחשבון היו 10,000 ₪ (כלומר, זכות), אז לאחר עדכון הריבית הוא יהיה 10,050 ₪
   אם בחשבון היו 10,000- ₪ (כלומר, חובה), אז לאחר עדכון הריבית הוא יהיה 10,070- ₪

בעמוד הבא נמצא המימוש ב- Java של המחלקה Bank.

השלימו את הדרוש, במקומות שמסומנים בקווים. **מספר השורות הריקות לא זהה בהכרח למספר השורות בפתרון.**

```
public class Bank
{
    private final int MAX_ACCOUNTS = 1000;
    private final __________ POSITIVE_INTEREST = 0.005;
    private final double NEGATIVE_INTEREST = __________;

    private ________________________________________
    private ________________________________________

    public Bank()
    {
        ____________________________________________
        ____________________________________________
    }

    public ________ addAccount(String first, String family,
                               long id, long accNum )
    {
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
    }

    public int accountCustomer(Customer c)
    {
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
    }
```

```
    public Customer wealthy()
    {
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
    }

    public void setInterestInAccounts()
    {
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
    }
}
```

שאלה 2 - 20 נקודות

נאמר ששורה של n מספרים שלמים מייצגת גבהים של נקודות ברכס, שיש בהם פסגות ועמקים.

כתבו שיטה שחתימתה:

public static int maximalDrop (int [] a)

המקבלת מערך ובו המספרים המייצגים את הגבהים האלו. השיטה צריכה להחזיר את הנפילה המקסימלית בין שני מספרים (לא בהכרח בתאים עוקבים), כך שהמספר הגבוה נמצא **לפני** המספר הנמוך במערך.

דוגמאות:

- עבור המערך: {5, 21, 3, 27, 12, 24, 7, 6, 4} התשובה שתוחזר תהיה 23 (שזה 27-4)
- עבור המערך: {5, 21, 3, 22, 12, 7, 26, 14} התשובה שתוחזר תהיה 18 (שזה 21-3)
- עבור המערך: {5, 15, 3, 22, 12, 7, 27, 14} התשובה שתוחזר תהיה 15 (שזה 22-7)

שימו לב, רק תשובה (נכונה) שתעבור על המערך פעם אחת בלבד תקבל את מלוא הנקודות. תשובה שתעבור על המערך יותר מפעם אחת תקבל לכל היותר 15 נקודות.

שאלה 3 - 20 נקודות - ענו את התשובה על גבי השאלון. תשובה שתיכתב במקום אחר לא תיבדק!

נתונות המחלקות הבאות (כל מחלקה בקובץ נפרד, כמובן):

```
public class A
{
    private int _x, _y;

    public A() {
        _x = 0;
        _y = 0;
    }
    public A(int x, int y) {
        _x = x;
        _y = y;
    }
    public A(A other)  {
        _x = other._x;
        _y = other._y;
    }

    // המשך המחלקה בעמוד הבא
```

```
    public int getX()  {
        return _x;
    }

    public int getY()  {
        return _y;
    }

    public void increment()  {
       _x = _x+1;
       _y = _y+1;
    }

    public boolean equals (A other)  {
        return (_x == other._x) && (_y == other._y);
    }
}
```

```
public class B
{

    private double _x, _y;

    public B()
    {
        _x = 1.0;
        _y = -1.0;
    }
    public B(double x, double y)
    {
        _x = x;
        _y = y;
    }

    public boolean equals (B other)
    {
        return (_x == other._x) && (_y == other._y);
    }

    public boolean equals (A other)
    {
        return (_x == other.getX()) && (_y == other.getY());
    }
}
```

נתונה השיטה main הבאה:

```
public static void main (String [] args)
{
   A a1 = new A(1, 2);
   A a2 = new A(2, 1);
   A a3 = a1;
   A a4 = new A(a1);
   a1.increment();

   B b1 = new B();
   B b2 = new B(1, 2);
   B b3 = new B();

   // כאן יופיעו שורות ההדפסה שלהלן
}
```

סעיף א (14 נקודות)

עבור כל אחד מהשורות הבאות, כתבו לצידה מה היא תדפיס:

1. System.out.print (a1 == a2); _____
2. System.out.print (a1 == a3); _____
3. System.out.print (a1 == a4); _____
4. System.out.print (a3 == a4); _____
5. System.out.print (b1 == b2); _____
6. System.out.print (b2 == b3); _____
7. System.out.print (b1 == b3); _____
8. System.out.print (a1.equals(a3)); _____
9. System.out.print (a2.equals(a1)); _____
10. System.out.print (a4.equals(a1)); _____
11. System.out.print (b1.equals(b2)); _____
12. System.out.print (b2.equals(a4)); _____
13. System.out.print (b1.equals(b3)); _____
14. System.out.print (b2.equals(a1)); _____

- המשך השאלה בעמוד הבא -

נניח שהשורות הבאות נכתבו בשיטה main לאחר כל ההדפסות שלעיל.

```
a1 = a3;
a3 = null;
a2 = a3;
a4 = new A();
```

סעיף ב (3 נקודות)

במהלך הפעלת הקוד בשיטה main (הקטע כולו, כולל ארבע השורות הנוספות) נוצרו מספר אובייקטים (null אינו נחשב כאובייקט). כמה אובייקטים **מטיפוס A** מייצר הקוד?

התשובה היא: _____

סעיף ג (3 נקודות)

במהלך הפעלת הקוד בשיטה main (הקטע כולו, כולל ארבע השורות הנוספות) נוצרו מספר אובייקטים (null אינו נחשב כאובייקט). אנו נאמר כי אובייקט הוא מוכר אם קיים משתנה הפונה אליו (מצביע עליו). כמה אובייקטים **מטיפוס A** מוכרים מיד לאחר ביצוע הפקודה האחרונה בקוד?

התשובה היא: _____

ב ה צ ל ח ה