

מבוא למדעי המחשב - סמסטר ב' תשע"ט

עבודת בית מספר 5

צוות התרגיל: דניאל אלדמ

תאריך פרסום: 10.6.19

תאריך הגשה: 25.6.19, 12:00 בצהריים

הוראות מקדימות:

הגשת עבודות בית

1. **קראו את העבודה מתחילתה ועד סופה לפני שאתם מתחילים לפתור אותה.** ודאו שאתם מבינים את כל המשימות. רמת הקושי של המשימות אינה אחידה.
2. ניתן להגיש את העבודה לבד או בזוג. ההגשה מתבצעת דרך מערכת ההגשות (System Submission) המחלקתית. אין לפתור את תרגילי הבית עם כל אדם אחר.
3. לעבודה מצורפים קובצי Java. צרו תיקייה חדשה והעתיקו את קבצי ה-Java לתוכה. עליכם לערוך את הקבצים האלו בהתאם למפורט בתרגיל ולהגישם כפתרון, מכווצים כקובץ ZIP יחיד. שימו לב: עליכם להגיש רק את קבצי ה-Java. אין לשנות את שמות הקבצים. אין להגיש קבצים נוספים. שם קובץ ה-ZIP יכול להיות כרצונכם, אך באנגלית בלבד. בנוסף, הקבצים שתגישו יכולים להכיל טקסט המורכב מאותיות באנגלית, מספרים וסימני פיסוק בלבד. טקסט אשר יכיל תווים אחרים (אותיות בעברית, יוונית וכד') לא יתקבל.
4. קבצים שיוגשו שלא על פי הנחיות אלו לא ייבדקו. את קובץ ה-ZIP יש להגיש ב-Submission System. פרטים בעניין ההרשמה ואיך להגיש את העבודה תוכלו למצוא באתר.
5. אין להשתמש ב-**packages**. אם תעשו בהן שימוש עבודתכם לא תתקבל על ידי מערכת ההגשות. בידקו כי המילה **package** אינה מופיעה בקובצי ההגשה שלכם

בדיקת עבודות הבית

6. עבודות הבית נבדקות גם באופן ידני וגם באופן אוטומטי. הבדיקה האוטומטית מתייחסת לפלט התכנית המודפס למסך. לכן, יש להקפיד על ההוראות ולבצע אותן במדויק. כל הדפסה אשר אינה עונה בדיוק על הדרישות המופיעות בעבודה (כולל שורות, רווחים, סימני פיסוק או כל תו אחר - מיותרים, חסרים או מופיעים בסדר שונה מהנדרש), לא תעבור את הבדיקה האוטומטית ולכן תגרור פגיעה בציון.
7. סגנון כתיבת הקוד ייבדק באופן ידני. יש להקפיד על כתיבת קוד ברור, הימנעות משכפול קוד, על מתן שמות משמעותיים למשתנים, על הזחות (אינדנטציה), ועל הוספת הערות בקוד המסבירות את תפקידם של מקטעי הקוד השונים. אין צורך למלא את הקוד בהערות סתמיות, אך חשוב לכתוב הערות בנקודות קריטיות, המסבירות קטעים חשובים בקוד. הערות יש לרשום אך ורק באנגלית. כתיבת קוד אשר אינה עומדת בדרישות אלו תגרור הפחתה בציון העבודה.
8. אין להדפיס למסך דברים מיותרים (כגון: "please enter an integer").

עזרה והנחיה

9. לכל עבודת בית יש מתרגל האחראי עליה. ניתן לפנות למתרגל בשעות הקבלה.

הערות ספציפיות לעבודות בית זו

10. בעבודה זו ניתן להגדיר פונקציות (עזר) נוספות, לפי שיקולכם. פונקציות אלו ייכתבו בתוך קובצי המשימה הרלוונטיים
11. בחלק מהמחלקות המצורפות מופיעה השיטה Remove(), התעלמו ממנה. אין למחוק או לשנות אותה. היא נחוצה לשם הגשה במערכת ההגשה.

חלק 1: איטרטור של מספרים ראשוניים

(30 נקודות)

בחלק זה של העבודה נממש איטרטור של מספרים ראשוניים. נתונה לכם המחלקה `PrimeIterator` המממשת את הממשק `Iterator` של `java`:

```
public class PrimeIterator implements Iterator<Integer> {...}
```

שימו לב כי בקובץ המחלקה מופיעה השורה: `import java.util.Iterator`.

איטרטור זה מחזיר בכל קריאה למתודה `next()` את המספר הראשוני הבא, החל מהמספר 2 (כולל).

במחלקה שדה יחיד (אין להוסיף שדות נוספים)

```
private List<Integer> primes;
```

הדרכה: בכל קריאה לשיטה `next()` האיטרטור ימצא מספר ראשוני יחיד. אין לבצע עיבוד ראשוני של המספרים הראשוניים כשלב מקדים באתחול האיטרטור. חישבו כיצד להשתמש בשדה זה על מנת לממש את האיטרטור באופן יעיל.

עליכם להשלים את השיטות הבאות במחלקה:

- `public PrimeIterator()`

בנאי המחלקה מאתחל את שדה המחלקה.

- `public boolean hasNext()`
- `public Integer next()`

השיטות `hasNext`, `next` הן השיטות המפורטות בממשק `Iterator` המובנה ב-`java`.

במידה ותשלימו נכונה את שיטות המחלקה `PrimeIterator` הקוד בקובץ `TestPrimeIterator.java` ידפיס

למסך את הפלט הבא (20 המספרים הראשוניים הראשונים):

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71,

חלק 2: ניהול בנק

בחלק זה של העבודה נממש מערכת לניהול בנק. במערכת אוסף של חשבונות כך שכל חשבון מאופיין על ידי שם, מספר חשבון ויתרה. המערכת תומכת בפעולות הבאות: יצירת מערכת חדשה (ריקה) לניהול בנק, הוספת חשבון, מחיקת חשבון, חיפוש חשבון לפי שם, חיפוש חשבון לפי מספר חשבון והפקדה/משיכה של כסף מחשבון מסויים. כדי לתמוך בחיפוש יעיל לפי שם ולפי מספר חשבון במערכת יתוחקו שני עצי חיפוש בינאריים. בעץ אחד החשבונות יהיו ממוינים לפי שם ובעץ השני החשבונות יהיו ממוינים לפי מספר החשבון. מכיוון שפעולות ההוספה והמחיקה עלולות להוציא את העצים מאיזון המערכת תומכת גם בפעולה המאזנת את העצים.

במערכת ניהול הבנק שנממש שמות ומספרי חשבונות צריכים להיות יחודיים. לא ייתכנו שני חשבונות עם אותו השם וגם לא ייתכנו שני חשבונות עם אותו מספר חשבון.

משימה 1: מבנה החשבון (5 נקודות)

החשבונות במערכת ניהול הבנק מתוארים על ידי הקובץ `BankAccount.java`. במשימה זו תבצעו הכרות עם המחלקה הנתונה לכם בקובץ זה ושבה תשתמשו בהמשך העבודה.

במחלקה `BankAccount` בנאי יחיד

- `public BankAccount(String name, int accountNumber, int balance)`

השיטות הציבוריות במחלקה הן:

- `public String getName()`
- `public int getAccountNumber()`
- `public int getBalance()`
- `public String toString()`

קראו היטב את הקוד שבקובץ `BankAccount.java`. עליכם להכיר את כל פרטי המחלקה, את השדות, הבנאים והשיטות שלה. כפי שתראו בקוד, שמות מיוצגים על ידי מחרוזות לא ריקות ומספרי חשבונות על ידי מספרים חיוביים.

עליכם להשלים את השיטה הבאה במחלקה:

- `public boolean spendOrDepositMoney(int amount)`

שיטה זו מקבלת מספר `amount`, אשר מייצג סכום כסף להפקדה בחשבון/ משיכה מהחשבון במידה ו-`amount` חיובי/ שלילי (בהתאמה). במידה והסכום `amount` שלילי אין לאפשר את משיכת הכסף אם הדבר יגרום ליתרה (השמורה בשדה `balance`) שלילית. במקרה זה יש להחזיר `false`. אחרת, יש להוסיף ליתרה את הכמות `amount` ולהחזיר `true`.

משימה 2: השוואת חשבונות (10 נקודות)

במשימה זו תשלימו את הגדרת שתי המחלקות הבאות בקבצים שקיבלתם.

- `public class AccountComparatorByName implements Comparator`
- `public class AccountComparatorByNumber implements Comparator`
מחלקות אלו מממשות את השיטה המוגדרת בממשק `Comparator<BankAccount>`:

`public int compare(BankAccount account1, BankAccount account2)`

במחלקה `AccountComparatorByName` שיטה זו משווה בין חשבונות (מסוג `BankAccount`) לפי שם (לפי הסדר הלקסיקוגרפי על מחרוזות) ובמחלקה `AccountComparatorByNumber` לפי מספר חשבון (לפי יחס הסדר הטבעי על מספרים). עליכם לממש את השיטה בשתי המחלקות.

שימו לב שבקבצים `AccountComparatorByName.java` ו-`AccountComparatorByNumber.java` מופיעה השורה `import java.util.Comparator`. זהו הממשק `Comparator` כפי שמוגדר ב-`java`. מומלץ להיזכר בפירטי הממשק `Comparator` כפי שמתואר ב API של `java`.

משימה 3: ממשקים נתונים / מחלקות נתונות (0 נקודות)

במשימה זו תבצעו הכרות עם הממשקים והמחלקות הבאים הנתונים לכם ושבהם תשתמשו בהמשך העבודה. אין לשנות את הקבצים הנתונים. שימו לב שהממשק `List` הנתון הוא חלקי ותואם את מטרות העבודה.

- `public interface Stack<T>`
- `public interface Queue<T>`
- `public interface List<T>`
- `public class StackAsDynamicArray<T> implements Stack<T>`
- `public class QueueAsLinkedList<T> implements Queue<T>`
- `public class DynamicArray<T> implements List<T>`
- `public class LinkedList<T> implements List<T>`

קראו היטב את הקוד בקבצים המתאימים. עליכם להכיר את כל פרטי המחלקות, את השדות, הבנאים והשיטות שלהן.

משימה 4: עצים בינאריים (10 נקודות)

במשימה זו נתונות לכם המחלקות `BinaryNode`, `BinaryTree`. מחלקות אלו זהות למחלקות שנלמדו בהרצאה. במשימה זו תשלימו במחלקה `BinaryNode` את הגדרת השיטה:

- `public String toString()`

השיטה `toString()` במחלקה `BinaryTree` נתונה לכם. אם העץ אינו ריק היא קוראת לשיטה `toString()` שבמחלקה `BinaryNode`. במחלקה `BinaryNode` השיטה פועלת כך שאם נדפס את המחרוזת שהיא מחזירה נקבל שורת הדפסה אחת ובה כל תת-עץ תחום בסוגריים, ובתוך הסוגריים מופיע (משמאל לימין) תת-העץ השמאלי, פסיק, השורש של תת-העץ, פסיק, ותת-העץ הימני (כלומר הקודקודים יודפסו בסדר `inorder`).

למשל, עץ ששורשו הוא 1, הבן השמאלי שלו הוא 2 והבן הימני שלו הוא 3 יודפס כך:
`tree:((2),1,(3))`
 עץ ששורשו הוא 1, הבן השמאלי שלו הוא 2 ואין לו בן ימני יודפס כך:
`tree:((2),1)`
 עץ ששורשו הוא 1, אין לו בן שמאלי והבן הימני שלו הוא 3 יודפס כך:
`tree:(1,(3))`

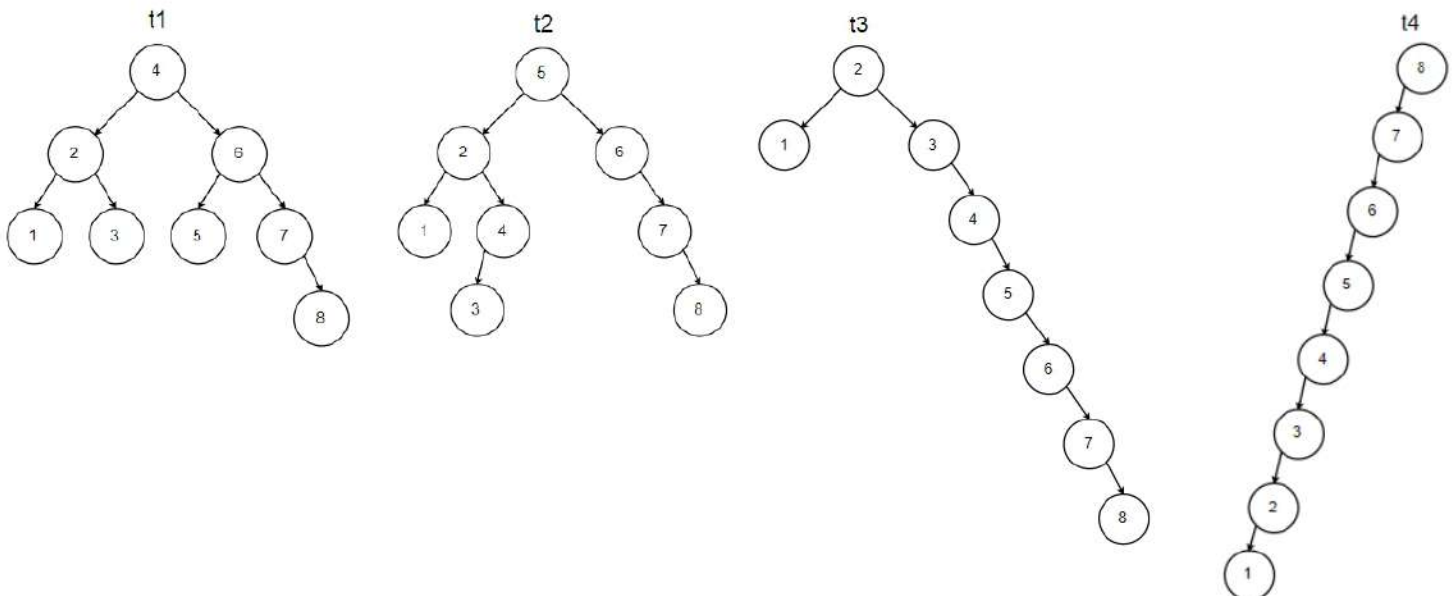
במידה ותשלימו נכונה את הגדרת השיטה `toString` במחלקה `BinaryNode` הקוד בקובץ `TestToString.java` ידפיס למסך את הפלטים הבאים (הציורים מיועדים להמחשת מבנה העץ):

-----t1:-----
`tree:(((1),2,(3)),4,((5),6,(7,(8))))`

-----t2:-----
`tree:(((1),2,((3),4)),5,(6,(7,(8))))`

-----t3:-----
`tree:(((1),2,(3,(4,(5,(6,(7,(8)))))))`

-----t4:-----
`tree:(((((((1),2),3),4),5),6),7),8)`



משימה 5: עצי חיפוש בינאריים, איטראטור

משימה 5א: הכרת המחלקות (0 נקודות)

המחלקה `BinarySearchTree`

נתונה לכם המחלקה `BinarySearchTree` בשלמותה. אין לשנות בה דבר. קראו היטב את הקוד שבקובץ `BinarySearchTree.java`. עליכם להכיר את כל פרטי המחלקה, את השדות, הבנאים, והשיטות שלה.

המחלקה `public class BinarySearchTree<T> extends BinaryTree<T> implements Iterable<T>{...}` יורשת את המחלקה `BinaryTree<T>` ומממשת את הממשק `Iterable<T>`.

במחלקה שדה יחיד

`Comparator<T> treeComparator`

בעזרתו המידע בעץ נשמר ממויין ומסודר על פי ה- `Comparator` המתקבל בעת יצירת העץ.

למחלקה בנאי יחיד:

- `public BinarySearchTree(Comparator myComparator)`

בנאי זה מקבל כפרמטר `Comparator` ובונה עץ חיפוש ריק.

נתונות השיטות הבאות:

- `public T findData(T element)`

כאשר נחפש איבר בעץ חיפוש בינארי, נעשה זאת בעזרת ה- `Comparator`. יתכן שהאיבר שנחפש לא יהיה זהה לזה שנמצא בעץ (שדה ה- `data` שבאחד הקודקודים של העץ) אך יהיה שווה לו לפי ה- `Comparator`.

שיטה זו מקבלת אובייקט `element`. השיטה מחפשת ומחזירה את ה- `data` השווה ל- `element` (על פי ה- `Comparator`) הנמצא בעץ המפעיל את השיטה, במידה וקיים. במידה ולא קיים בעץ קודקוד עם שדה `data` השווה ל- `element` (על פי ה- `Comparator`), השיטה מחזירה ערך `null`.

דוגמאות:

1. בקריאה לשיטה זו כאשר ה- `Comparator` של העץ הוא מטיפוס `AccountComparatorByName`, `element` מפנה אל החשבון `<"Dan", 1, 0>`, והעץ מכיל את החשבון `<"Dan", 86471234, 130>`, תוחזר הפניה לאותו החשבון `<"Dan", 86471234, 130>`.
2. בקריאה לשיטה זו כאשר ה- `Comparator` של העץ הוא מטיפוס `AccountComparatorByNumber`, `element` מפנה אל החשבון `<"dummy", 86471234, 0>`, והעץ מכיל את החשבון `<"Dan", 86471234, 4000>`, תוחזר הפניה לאותו החשבון `<"Dan", 86471234, 4000>`.

- `public Comparator getComparator()`

שיטה זו מחזירה את ה- `Comparator` של העץ.

- `public void insert(T toInsert)`

שיטה זו מקבלת אובייקט מטיפוס `T` בשם `toInsert` ומכניסה אותו לעץ. זיכרו כי במערכת ניהול הבנק שנממש שמות ומספרי חשבונות צריכים להיות יחודיים. לא ייתכנו שני חשבונות עם אותו השם וגם לא ייתכנו שני חשבונות עם אותו מספר החשבון. במידה ו- `toInsert` מתנגש עם דרישה זו השיטה לא תשנה את העץ.

- `public void remove(T toRemove)`

שיטה זו מקבלת אובייקט `toRemove` ומסירה אותו מהעץ, במידה והוא קיים בו.

- `public Iterator iterator()`

שיטה זו מחזירה `Iterator` של העץ מטיפוס `BinaryTreeInOrderIterator`.

המחלקה `BinarySearchNode`

נתונה לכם המחלקה `BinarySearchNode` בשלמותה. אין לשנות בה דבר. קראו היטב את הקוד שבקובץ `BinarySearchNode.java`. עליכם להכיר את כל פרטי המחלקה, את השדות, הבנאים, והשיטות שלה.

המחלקה `{...} public class BinarySearchNode extends BinaryNode` יורשת את המחלקה `BinaryNode`.

במחלקה שדה יחיד

`Comparator<T> treeComparator`

בעזרתו המידע בעץ נשמר ממויין ומסודר על פי טיפוס ה- `Comparator` המתקבל בעת יצירת קודקוד.

למחלקה בנאי יחיד:

- `public BinarySearchNode(T data, Comparator<T> myComparator)`

בנאי זה מקבל אובייקט מטיפוס `T` בשם `data` ו- `Comparator` ובונה קודקוד חיפוש.

נתונות השיטות הבאות:

- `public T findData(T element)`

שיטה זו מקבלת אובייקט מטיפוס `T` בשם `element` מחפשת ומחזירה את ה- `data` השווה ל- `element` (על פי ה- `Comparator`) הנמצא בתת העץ המושרש בקודקוד המפעיל את השיטה, במידה וקיים. במידה ו- `element` לא קיים בתת עץ זה על השיטה להחזיר את הערך `null`. ראו דוגמאות לשיטה `findData` במחלקה `BinarySearchTree`.

- `public T findMin()`

השיטה מחזירה את שדה ה- `data` של הקודקוד המכיל את ה- `data` ה"קטן ביותר" על פי ה- `Comparator` בתת העץ המושרש בקודקוד המפעיל את השיטה.

- `public Comparator<T> getComparator()`

שיטה זו מחזירה את ה- `Comparator` של העץ.

- `public void insert(T toInsert)`

שיטה זו מקבלת אובייקט מטיפוס `T` בשם `toInsert` ומכניסה אותו לקודקוד חדש במקום המתאים לו בתת העץ המושרש בקודקוד המפעיל את השיטה. אם תת העץ המושרש בקודקוד מכיל את `toInsert` אז אובייקט זה לא ייכנס לעץ.

- `public boolean contains(T element)`

שיטה זו מקבלת אובייקט מטיפוס `T` בשם `element` ומחזירה `true` אם תת העץ המושרש בקודקוד המפעיל את השיטה מכיל את `element`.

- `public BinaryNode<T> remove(T toRemove)`

שיטה זו מקבלת אובייקט מטיפוס T בשם `toRemove` ומסירה אותו מהעץ המושרש בקודקוד המפעיל את השיטה במידה והוא שווה לאחד האיברים בעץ (על פי ה- `Comparator` של העץ). השיטה מחזירה מצביע לשורש העץ המושרש בקודקוד המפעיל את השיטה לאחר ההסרה.

המחלקה `BinaryTreeInOrderIterator`

נתונה לכם המחלקה `BinaryTreeInOrderIterator` המממשת את הממשק `Iterator` של `java`. שימו לב כי בקובץ המחלקה מופיעה השורה: `import java.util.Iterator;` איטרטור זה עובר על המידע השמור בעץ החיפוש לפי סדר `.inorder`.

משימה 5ב: (25 נקודות)

המחלקה `BankAccountBinarySearchTree` יורשת את המחלקה `BinarySearchTree<BankAccount>`. למחלקה בנאי יחיד:

- `public BankAccountBinarySearchTree(Comparator<BankAccount> myComparator)`
בנאי זה מקבל קומפרטור `myComparator` וקורא לבנאי של המחלקה אותה הוא יורש.
- עליכם להשלים את שתי השיטות הבאות במחלקה:
- `public void balance()`
שיטה זו מאזנת את העץ `this` כך שסדר ה-`inorder` שלו נשמר כפי שהיה. בקוד השיטה ישנה קריאה לשיטת העזר הפרטית `buildBalancedTree` רקורסיבית.
- הדרכת חובה: את השיטה `balance()` יש להשלים בעזרת שיטת העזר הפרטית הבאה (אין להוסיף שיטות עזר נוספות).
- `private void buildBalancedTree(BankAccountBinarySearchTree tree, List<BankAccount> list, int low, int high)`
שיטה רקורסיבית זו מקבלת עץ `tree`, רשימה `list` של חשבונות ומספרים שלמים `low` ו-`high`. מומלץ מאוד כי בקריאה הראשונית לשיטה זו מהשיטה `balance()` ישלחו המשתנים הבאים (לפי סדר הפרמטרים):
- עץ ריק.
- רשימה המכילה את חשבונות הבנק שבעץ על פי סדר ה-`inorder` שלהם בעץ.
- האינדקס 0.
- האינדקס `list.size()-1`.

עליכם להשלים את השיטה באופן רקורסיבי, כך שכל החשבונות שברשימה יוכנסו לעץ. בסוף התהליך העץ `tree` יכיל את כל החשבונות שברשימה ויהיה מאוזן (ראו הגדרה בתחילת העבודה). נחזור ונדגיש כי סדר ה-`inorder` של החשבונות חייב להישמר כפי שהיה ברשימה (זהו אותו הסדר שהיה בעץ לפני תהליך האיזון).

במידה ותשלימו נכונה משימה זו הקוד בקובץ TestBalance.java ידפיס למסך את הפלטים הבאים:

```
-----unbalanced t1:-----  
tree:(((1),2,(3)),4,((5),6,(7,(8))))
```

```
-----balanced t1:-----  
tree:(((1),2,(3)),4,((5),6,(7,(8))))
```

```
-----unbalanced t2:-----  
tree:(((1),2,((3),4)),5,(6,(7,(8))))
```

```
-----balanced t2:-----  
tree:(((1),2,(3)),4,((5),6,(7,(8))))
```

```
-----unbalanced t3:-----  
tree:((1),2,(3,(4,(5,(6,(7,(8)))))))
```

```
-----balanced t3:-----  
tree:(((1),2,(3)),4,((5),6,(7,(8))))
```

```
-----unbalanced t4:-----  
tree:(((((((1),2),3),4),5),6),7),8)
```

```
-----balanced t4:-----  
tree:(((1),2,(3)),4,((5),6,(7,(8))))
```

משימה 6: מערכת ניהול הבנק (20 נקודות)

במשימה זו תשלימו את הגדרת המחלקה Bank בקובץ Bank.java. למחלקה שני שדות

```
private BankAccountBinarySearchTree namesTree;  
private BankAccountBinarySearchTree accountNumbersTree;
```

שהינם עצי חיפוש בינארי. עצים אלו מכילים את אוסף החשבונות (מסוג BankAccount) הקיים בבנק. בעץ הראשון החשבונות ממוינים לפי שמות ובעץ השני לפי מספרי חשבון. נדגיש כי כל חשבון קיים במערכת ניהול הבנק רק פעם אחת, ובכל עץ קיים לה קודקוד ובו שדה BankAccount data המפנה אליו. בנואי המחלקה public Bank() מגדיר מערכת לניהול בנק ריקה (עם שני עצי חיפוש ריקים).

נתונות השיטות הבאות (אין לשנות את הגדרתן):

- `public BankAccount lookUp(String name)`
שיטה זו מקבלת שם name ומחזירה את החשבון במערכת ניהול הבנק עם השם name במידה וקיים כזה. אחרת השיטה תחזיר את הערך null.
- `public BankAccount lookUp(int accountNumber)`
שיטה זו מקבלת מספר number ומחזירה את החשבון במערכת ניהול הבנק עם מספר חשבון number במידה וקיים כזה. אחרת השיטה תחזיר את הערך null.
- `public void balance()`
שיטה זו מיועדת לשמירה על יעילות השימוש במערכת ניהול הבנק.
שיטה זו בונה מחדש את שני עצי החיפוש כך שתכולתם תישאר זהה אך מבנה העץ יהיה מבנה של עץ מאוזן (ראו הגדרה בתחילת העבודה). פעולה זו מתבצעת על ידי שתי קריאות למתודה balance() של המחלקה BankAccountBinarySearchTree (קריאה אחת לכל אחד מהעצים).

עליכם להשלים את השיטות הבאות במחלקה:

- `public boolean add(BankAccount newAccount)` (5 נקודות)
שיטה זו מקבלת חשבון חדש newAccount ומוסיפה אותו למערכת ניהול הבנק במידה והתנאים הבאים מתקיימים:
 - אין במערכת ניהול הבנק חשבון קיים עם אותו השם שב-newAccount.
 - אין במערכת ניהול הבנק חשבון קיים עם אותו מספר חשבון שב-newAccount.הפונקציה מחזירה true אם ההוספה התבצעה בהצלחה ומחזירה false אחרת. יש להוסיף את אותו החשבון לשני העצים המוגדרים בשדות המחלקה.
- `public boolean delete(String name)` (5 נקודות)
שיטה זו מקבלת שם name ומוחקת את החשבון במערכת ניהול הבנק עם השם name במידה וקיים כזה. זיכרו כי במידה והחשבון קיים יש להסיר את ההפניה אליו משני העצים. השיטה מחזירה true אם התבצעה מחיקה ו-false אחרת.
- `public boolean delete(int accountNumber)` (5 נקודות)
שיטה זו מקבלת מספר number ומוחקת את החשבון במערכת ניהול הבנק עם מספר חשבון accountNumber במידה וקיים כזה. זיכרו כי במידה והחשבון קיים יש להסיר את ההפניה אליו משני העצים. השיטה מחזירה true אם התבצעה מחיקה ו-false אחרת.
- `public boolean spendOrDepositMoney(int amount, int accountNumber)` (5 נקודות)
שיטה זו מקבלת מספר amount ומספר חשבון accountNumber, מוצאת את החשבון המתאים, קוראת למתודה spendOrDepositMoney(amount) עבור חשבון זה. השיטה מחזירה true אם הפעולה הצליחה ו-false אחרת.

בהצלחה!