

מבוא למדעי המחשב - סמסטר א' תשפ"א

עבודת בית מספר 5

צוות העבודה:

- מתרגלים אחראים: נדב קרן, איליה קאופמן
- מרצה אחראית: מיכל שמש

תאריך פרסום: 26.12.21

מועד אחרון להגשה: 9.1.21 בשעה 23:59

מה בתגבור:

- בתגבור 11 בתאריכים 26-29/12/21 נפתור את משימות: 1 (סעיף א), 2 (סעיף א).

תקציר נושא העבודה:

במהלך עבודה זו תתנסו במרכיבים שונים של תכנות מונחה-עצמים ושימוש במבני נתונים. תחשפו לנושאים הבאים:

1. שימוש בממשק איטרטור באופנים שונים.
2. מימוש פעולות במחלקות המייצגות עצמים.
3. מימוש הממשק Comparator ושימוש בו.
4. פעולות על עצים בינאריים ועצי חיפוש בינאריים.
5. עבודה עם מחלקות המכילות מספר מבני נתונים.

הנחיות מקדימות

- קראו את העבודה מתחילתה ועד סופה לפני שאתם מתחילים לפתור אותה. רמת הקושי של המשימות אינה אחידה: הפתרון של חלק מהמשימות קל יותר, ואחרות מצריכות מחשבה וחקירה מתמטית - שאותה תוכלו לבצע בעזרת מקורות דרך רשת האינטרנט. בתשובות שבהן אתם מסתמכים על עובדות מתמטיות שלא הוצגו בשיעורים או כל חומר אחר, יש להוסיף כהערה במקום המתאים בקוד את ציטוט העובדה המתמטית ואת המקור (כגון ספר או אתר).
- עבודה זו תוגש ביחידים במערכת המודל ניתן לצפות בסרטון הדרכה על אופן הגשת העבודה במערכת ה-VPL בלינק הבא: סרטון הדרכה.
- בכל משימה מורכבת יש לשקול כיצד לחלק את המשימה לתתי-משימות ולהגדיר פונקציות עזר בהתאם. בכל

- הסעיפים אפשר ומומלץ להשתמש בפונקציות מסעיפים קודמים
- בעבודה זו מספר מחלקות בהן תידרשו לכתוב קוד. שאר המחלקות המצורפות הן לשימושן ואין לשנות אותן. פירוט מדויק של מחלקות אלו יופיע בהמשך.
- המלצה על דרך העבודה** - אנו ממליצים לפתוח ב-Eclipse פרויקט בשם Assignment5 ולהעתיק אליו את הקבצים שתורידו מה-VPL.

הנחיות לכתיבת קוד והגשה

- בקבצי השלד המסופקים לכם קיים מימוש ברירת מחדל לכל משימה. יש למחוק את מימוש ברירת המחדל בגוף הפונקציות ולכתוב במקום זאת את המימוש שלכם לפי הנדרש בכל משימה.
- אין לשנות את החתימות של השיטות המופיעות בקבצי השלד.
- עבודות שלא יעברו קומפילציה במערכת יקבלו את ה**ציון 0** ללא אפשרות לערער על כך. אחריותכם לוודא שהעבודה שאתם מגישים עוברת תהליך קומפילציה במערכת (ולא רק ב-eclipse). להזכירכם, תוכלו לבדוק זאת ע"י לחיצה על כפתור ה-Evaluate. 
- עבודות הבית נבדקות גם באופן ידני וגם באופן אוטומטי. לכן, יש להקפיד על ההוראות ולבצע אותן במדויק.
- סגנון כתיבת הקוד ייבדק באופן ידני. יש להקפיד על כתיבת קוד יעיל, ברור, 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 מיותרות, 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.
- בעבודה זו ניתן להשתמש בידע שנרכש בקורס עד לתאריך פרסום העבודה. אין להשתמש בכל צורת קוד אחרת אשר לא נלמדה בכיתה.

עזרה והנחיה

- לכל עבודת בית בקורס יש צוות שאחראי לה. ניתן לפנות לצוות בשעות הקבלה. פירוט שמות האחראים לעבודה מופיע במסמך זה וכן באתר הקורס, כמו גם פירוט שעות הקבלה.
- ניתן להיעזר בפורום. צוות הקורס עובר על השאלות ונותן מענה במקרה הצורך. שימו לב, **אין לפרסם פתרונות בפורום**.
- בכל בעיה אישית הקשורה בעבודה (מילואים, אשפוז וכו'), אנא פנו אלינו דרך מערכת הפניות, כפי שמוסבר באתר הקורס.
- אנחנו ממליצים בחום להעלות פתרון למערכת המודל לאחר כל סעיף שפתרתם. הבדיקה תתבצע על הגרסה

האחרונה שהועלתה (בלבד!).

הצהרה

ב VPL-פתחו את הקובץ readme.txt כתבו בראשו את שמכם ואת מספר תעודת הזהות שלכם. משמעות פעולה זו היא שאתם מסכימים על הכתוב בו. דוגמה:

I, Israel Israeli (123456789), assert that the work I submitted is entirely my own.

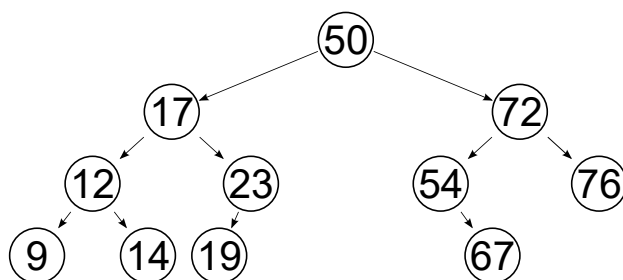
I have not received any part from any other person, nor did I give parts of it for use to others.

I realize that if my work is found to contain code that is not originally my own, a formal complaint will be opened against me with the BGU disciplinary committee.

הערות ספציפיות לעבודת בית זו

1. הגדרה: עץ בינארי מאוזן הינו עץ בינארי שבו ההפרש בין הגובה של שני תתי-עצים של אותו הצומת לעולם אינו גדול מאחד.

דוגמה:



חלק 1: איטרטור של מספרים ראשוניים ותשתית מערכת ניהול בנק

איטרטור של מחרוזות

משימה מספר 0 (30 נקודות)

בחלק זה של העבודה נממש איטרטור של מחרוזות. נתונה לכם המחלקה `StringIterator` המממשת את הממשק `Iterator` של `java`:

```
public class StringIterator implements Iterator<String> { ... }
```

שימו לב כי בקובץ המחלקה מופיעה השורה `import java.util.Iterator;` שורה זו מייבאת את הממשק `איטרטור` המוטמע ב-`java`.

איטרטור זה מקבל מחרוזות התחלתיות בבנאי ומחזיר בכל קריאה לשיטה `next()` את המחרוזות הבאה.

אם המחרוזות שהתקבלה בבנאי היא המחרוזות הריקה "", אוסף של קריאות ל-`next()` יחזיר את הסדרה הבאה:

"", "a", "b", ..., "z", "A", "B", ..., "Z", "aa", "ab", "ac", "ad",...

תחילה יש להחזיר את כל המחרוזות באורך 1 לפי סדר האלפבית. לאחר מכן יש להחזיר את כל המחרוזות באורך 2 לפי סדר האלפבית וכך הלאה. לאחר מעבר על כל האותיות הקטנות יש לעבור לאותיות גדולות. לדוגמה:

אם המחרוזות הנוכחית היא: "z", לאחר קריאה ל-`next()`, נצפה לקבל את המחרוזות "A".

אם המחרוזות הנוכחית היא: "Z", לאחר קריאה ל-`next()`, נצפה לקבל את המחרוזות "aa".

אם המחרוזות הנוכחית היא: "aZ", לאחר קריאה ל-`next()`, נצפה לקבל את המחרוזות "ba".

הדרכה: רק בעת הקריאה לשיטה `next()` האיטרטור יחשב את המחרוזות הבאה. אין לבצע חישוב מקדים של יותר ממחרוזות אחת בשלב אתחול האיטרטור.

בניגוד לתרגילי בית קודמים, בתרגיל זה תוכלו להוסיף שדות ושיטות כרצונכם.

ניתן להשתמש בכל השיטות במחלקה `String`.

עליכם להשלים את השיטות הבאות במחלקה:

- `public StringIterator(String start)`
בנאי המחלקה שמקבל את המחרוזת להתחלתית.
- `public boolean hasNext()`
- `public String next()`

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

הנחות: ניתן להניח שהאיטרטור לא ידרש להחזיר מחרוזות שאורכן גדול מהערך המקסימלי הניתן לייצוג על ידי משתנה מטיפוס `int`.

מערכת לניהול 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 מסוים. כדי לתמוך בחיפוש יעיל לפי שם ולפי מספר חשבון במערכת יתוחזקו 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 ייחודיים. לא ייתכנו שני חשבונות עם אותו השם וגם לא ייתכנו שני חשבונות עם אותו מספר חשבון, כמו כן לא ייתכנו ערכי `null`.

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.

שדות המחלקה:

- `String name` - שם החשבון
- `int accountNumber` - מספר החשבון
- `int balance` - היתרה הנוכחית בחשבון

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 depositMoney(int amount)`

שיטה זו מפקידה את הסכום `amount` ליתרה הנוכחית בחשבון.
אם ערכו של `amount` שלילי אין לבצע שינוי ביתרה ויש להחזיר את הערך `false`. אחרת יש לעדכן את היתרה בחשבון ולהחזיר `true`.

- `public boolean withdrawMoney(int amount)`

שיטה זו מושכת את הסכום `amount` מהיתרה הנוכחית בחשבון.
אם ערכו של `amount` שלילי או אם ערכו גדול מהיתרה הנוכחית (דבר שיגרום ליתרה שלילית בחשבון) אין לאשר את המשיכה. במקרה זה השיטה תחזיר `false` ללא שינוי היתרה.
בכל מקרה אחר יש לעדכן את היתרה ולהחזיר `true`.

משימה 2: השוואת חשבונות (10 נקודות)

במשימה זו תשלימו את הגדרת שתי המחלקות הבאות בקבצים שקיבלתם.

- `public class AccountComparatorByName implements Comparator<BankAccount>`
- `public class AccountComparatorByNumber implements Comparator<BankAccount>`
מחלקות אלו מממשות את הממשק `Comparator<T>` המובנה ב-java ובו השיטה `compare`.

במחלקה `AccountComparatorByName` השיטה `compare` משווה בין חשבונות (מסוג `BankAccount`) לפי שם (לפי הסדר הלכסיקוגרפי 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 List<T>`
- `public class StackAsDynamicArray<T> implements Stack<T>`
- `public class DynamicArray<T> implements List<T>`
- `public class DynamicArrayIterator<T> implements Iterator<T>`
- `public class LinkedList<T> implements List<T>`
- `public class LinkedListIterator<T> implements Iterator<T>`
- `public class Link<E>`

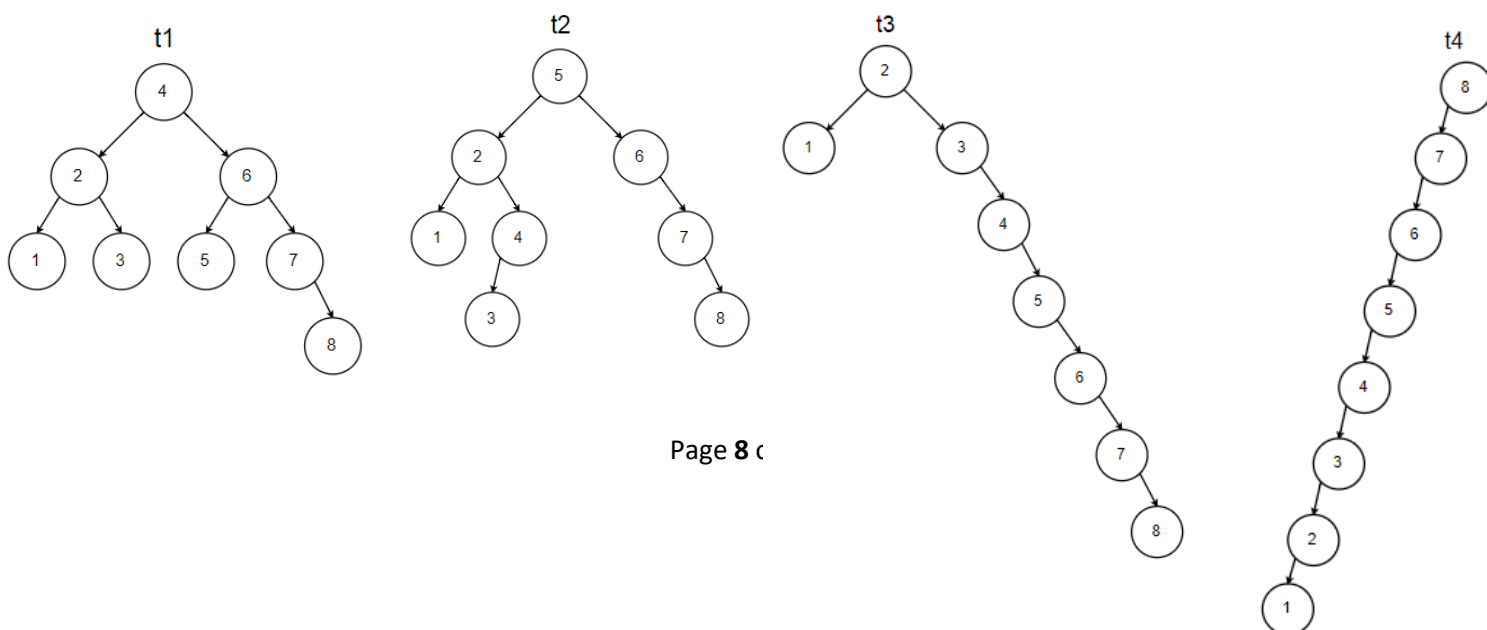
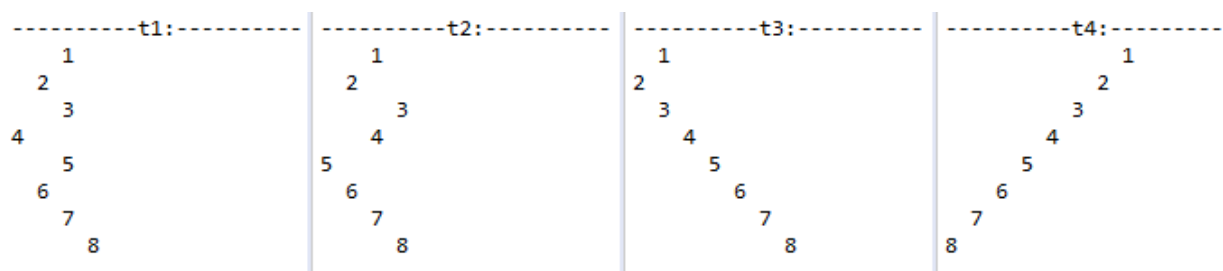
קראו היטב את הקוד בקבצים המתאימים. עליכם להכיר את כל פרטי המחלקות, את השדות, הבנאים והשיטות שלהן.

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 לכל קודקוד בעץ. בשורה זו יופיעו $2*d$ רווחים, כאשר d עומק הקודקוד בעץ ואח"כ יודפס (באותה השורה) המידע שבקודקוד. הקודקודים יודפסו בסדר `inorder`. במשימה זו מומלץ להשתמש בשיטות עזר פרטיות.
- נוח יותר לחשוב על פתרון רקורסיבי לבעיה הזו.

במידה ותשלימו נכונה את הגדרת השיטה `toString` במחלקה `BinaryNode` הקוד בקובץ `TestToString.java` ידפס למסך את הפלטים הבאים. הציורים מיועדים להמחשת מבנה העץ. אין לייחס חשיבות לאיברים שבציורי ההמחשה.



חלק 2: השלמת מערכת ניהול בנק

משימה 5: עצי חיפוש בינאריים, איטרטור

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> comparator`

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

למחלקה בנאי יחיד:

- `public BinarySearchTree(Comparator<T> comparator)`

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

נתונות השיטות הבאות:

- `public T findData(T element)`

שיטה זו מקבלת אובייקט `element`. השיטה מחפשת ומחזירה את ה- `data` של הקודקוד 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`.

- `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 אם לא קיים אובייקט זהה לו בעץ (על פי ה-Comparator).

- `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 BinaryTreeNodeIterator.

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> comparator`

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

למחלקה בנאי יחיד:

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

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

נתונות השיטות הבאות:

- `public T findData(T element)`

שיטה זו מקבלת אובייקט מטיפוס T בשם element מחפשת ומחזירה את ה- data של הקודקוד 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 השיטה מחזירה את הערך null.

- `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 במקום המתאים בתת העץ המושרש בקודקוד זה אם לא קיים אובייקט זהה לו בתת עץ זה (על פי ה Comparator).

- `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 אם עץ זה מכיל את toRemove. השיטה מחזירה מצביע לשורש העץ המושרש בקודקוד המפעיל את השיטה לאחר ההסרה.

המחלקה BinaryTreeInOrderIterator

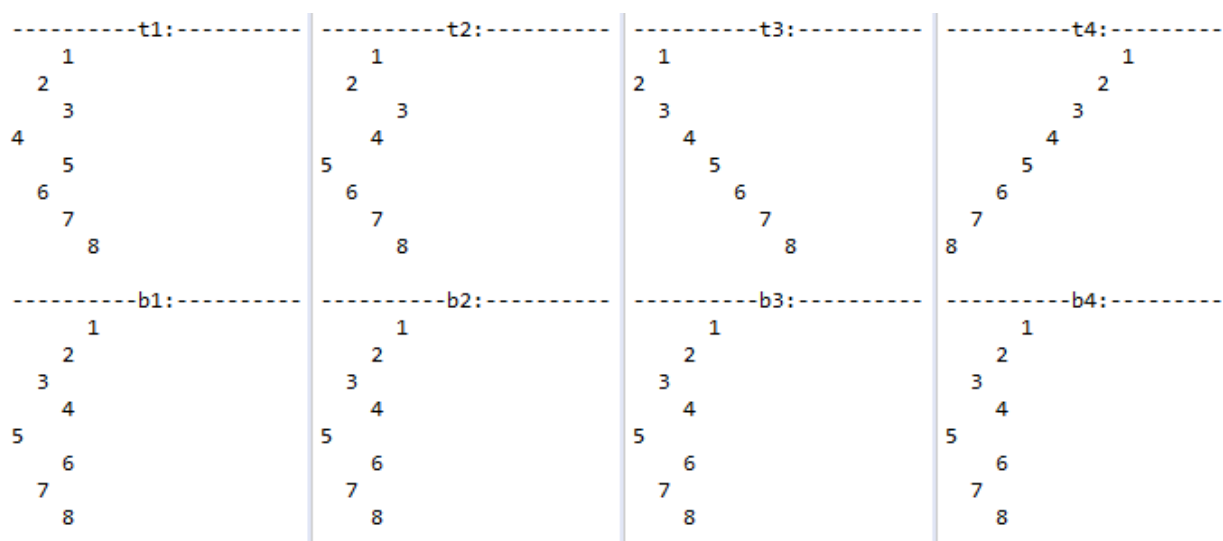
נתונה לכם המחלקה BinaryTreeInOrderIterator המממשת את הממשק Iterator של java. שימו לב כי בקובץ המחלקה מופיעה השורה: `import java.util.Iterator;` איטרטור זה עובר על המידע השמור בעץ החיפוש לפי סדר inorder.

משימה 5ב: (15 נקודות)

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

- `public BankAccountsBinarySearchTree(Comparator<BankAccount> comparator)`
בנאי זה מקבל קומפרטור `comparator` 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`.
 - הדרכת חובה: את השיטה `balance()` יש להשלים בעזרת שיטת העזר הפרטית הבאה (אין להוסיף שיטות עזר נוספות).
 - `private void buildBalancedTree(List<BankAccount> list, int low, int high)`
שיטה רקורסיבית זו מקבלת רשימה `list` של חשבונות ומספרים שלמים `low` ו-`high`. מומלץ מאוד כי בקריאה הראשונית לשיטה זו מהשיטה `balance()` ישלחו המשתנים הבאים (לפי סדר הפרמטרים):
 - 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 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).

אם תשלימו נכונה משימה זו הקוד בקובץ TestBalance.java ידפיס למסך ארבעה עצים, וגרסה מאוזנת שלהם. אחת האפשרויות לפלט היא (בדוגמא מוצג רק מספר החשבון ולא כל הפרטים שלו):



משימה 5ג (3 נקודות)

נתון ממשק `Filter<T>`. בממשק זה שיטה אחת:

- `Public boolean accept(T element)`

שיטה זו תחזיר ערך `true` אם `element` מקיים את התנאי הפילטר וערך `false` אחרת.

עליכם להשלים את הגדרת המחלקה `FilterByBalance` המממשת את הממשק `Filter<BankAccount>` בקובץ `FilterByBalance.java`

מחלקה זו מממשת פילטר המעביר רק חשבונות בנק אשר יתרתם גדולה או שווה לערך סף מסוים אשר נקבע בעת יצירת הפילטר.

עליכם לממש את שתי השיטות הבאות:

```
public FilterByBalance (int balanceThreshold)
```

עליכם לממש את בנאי המחלקה אשר מקבלת יתרה ומאתחל את השדה.

```
public boolean accept(BankAccount elem)
```

שיטה זו תחזיר ערך true אם היתרה בחשבון element גדולה או שווה לערך שהתקבל בבנאי וערך false אחרת.

משימה 5ד (7 נקודות)

במשימה זו תשלימו את הגדרת המחלקה `FilteredBankAccountsIterator implements Iterator<BankAccount>`

בקובץ `FilteredBankAccountsIterator.java`

מחלקה זו מממשת את הממשק `Iterator`. האיטרטור המתואר במחלקה זו מבצע מעבר על איברי העץ בסדר `inorder`, ומחזיר אך ורק חשבונות בנק אשר עומדים בתנאי הפילטר שמתקבל בבנאי המחלקה.

למחלקה 3 שדות:

- `private BinaryTreeInOrderIterator<BankAccount> iterator;`
- `private Filter<BankAccount> filter;`
- `private BankAccount current;`

עליכם לממש את שלוש השיטות הבאות:

```
public FilteredBankAccountsIterator
```

```
(BankAccountsBinarySearchTree bankAccountsTree, Filter<BankAccount> filter)
```

עליכם לממש את בנאי המחלקה אשר מקבל עץ של חשבונות ופילטר ומאתחל את השדות.

```
public boolean hasNext()
```

השיטה מחזירה true אם קיים חשבון שעומד בתנאי של הפילטר בעץ שטרם הוחזר. אחרת השיטה מחזירה false.

```
public BankAccount next()
```

השיטה מחזירה את חשבון הבנק הבא, שהינו עומד בתנאי של הפילטר. במידה ואין חשבון כזה, על השיטה לזרוק חריגה מסוג `NoSuchElementException`.

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 בינאריים. 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 לו קודקוד ובו שדה BankAccount data המפנה אליו. כלומר, לכל חשבון יש שתי הפניות בסך הכל, אחת בכל עץ. בנאי 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 accountNumber ומחזירה את החשבון במערכת ניהול הבנק עם מספר חשבון number במידה וקיים כזה. אחרת השיטה תחזיר את הערך null.

עליכם להשלים את השיטות הבאות במחלקה:

- public boolean add(BankAccount newAccount) (4 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. ניתן להניח שnewAccount אינו null.
- public boolean delete(String name) (4 נקודות)
שיטה זו מקבלת שם name ומוחקת את החשבון במערכת ניהול הבנק עם השם name אם קיים כזה. זיכרו כי אם החשבון קיים יש להסיר את ההפניה אליו משני העצים. השיטה מחזירה true אם התבצעה מחיקה ו-false אחרת.

- **public boolean delete(int accountNumber) (4 נקודות)**
שיטה זו מקבלת מספר number ומוחקת את החשבון במערכת ניהול הבנק עם מספר חשבון accountNumber אם קיים כזה. זיכרו כי אם החשבון קיים יש להסיר את ההפניה אליו משני העצים. השיטה מחזירה true אם התבצעה מחיקה ו- false אחרת.
- **public boolean depositMoney(int amount, int accountNumber) (2 נקודות)**
שיטה זו מקבלת מספר amount ומספר חשבון accountNumber, מוצאת את החשבון המתאים, קוראת למתודה depositMoney (amount) עבור חשבון זה. השיטה מחזירה true אם הפעולה הצליחה ו- false אחרת.
- **public boolean withdrawMoney(int amount, int accountNumber) (2 נקודות)**
שיטה זו מקבלת מספר amount ומספר חשבון accountNumber, מוצאת את החשבון המתאים, קוראת למתודה withdrawMoney (amount) עבור חשבון זה. השיטה מחזירה true אם הפעולה הצליחה ו- false אחרת.
- **public boolean transferMoney(int amount, int accountNumber1, int accountNumber2) (2 נקודות)**
שיטה זו מקבלת מספר amount ושני מספרי חשבון. אם קיימת יתרה הגדולה שווה מ-amount בחשבון accountNumber1, יש להעביר את הסכום amount מהחשבון accountNumber1 לחשבון accountNumber2. השיטה מחזירה true אם הפעולה הצליחה ו- false אחרת.
- **public boolean transferMoney(int amount, int accountNumber, String name) (2 נקודות)**
שיטה זו מקבלת מספר amount ומספר חשבון ושם. אם קיימת יתרה הגדולה שווה מ-amount בחשבון accountNumber, יש להעביר את הסכום amount מהחשבון accountNumber לחשבון עם השם name במידה וקיים כזה. השיטה מחזירה true אם הפעולה הצליחה ו- false אחרת.

העשרה (0 נקודות)

: Javadoc

הקדמה:

שיטת התיעוד הפשוטה, המוכרת לנו זה מכבר, מוגבלת למדי: היא אמנם מקילה על הבנת הקוד, אך הדרך היחידה להיעזר בה היא לקרוא את ההערות בגוף הקוד עצמו. אם, לדוגמה, נכתוב מחלקה ונרצה להכין ללקוח דף הסבר על השיטות שבה - נצטרך לעמול זמן רב. סטנדרט התיעוד Javadoc מאפשר להתגבר על בעיה זו, כשהוא מאפשר ליצור תיעוד אחיד וברור, ממנו ניתן ליצור בקלות רבה דפי הסבר חיצוניים (לפרסום ללקוח או ברשת האינטרנט).

תיעוד של תכניות באמצעות Javadoc צריך להיעשות על פי העיקרון המוכר של הכימוס (encapsulation). תפקיד התיעוד הוא לתאר מה עושה המחלקה והשיטות, לא איך הדבר נעשה. אין פירוש הדבר שצריך להזניח את הערות ההסבר בתוך הקוד ואת התיעוד של חלקים פרטיים במחלקה: התיעוד שנוצר באמצעות ה-Javadoc מטרתו ליצור מעין מדריך חיצוני למחלקה, המתאר כל פרט שראוי שמשתמשים חיצוניים במחלקה יכירו. עדיין, רצוי מאוד לבנות קוד ברור שיקל על מתכנתים להבין מה כתוב בו וכיצד הוא עושה מה שכתוב בו.

Javadoc מאפשר שימוש בתגים. קיים מגוון של תגים שונים, כאשר המשותף לכולם הוא סימון של "@" בתחילתם. נפרט כאן חלק מהם.

מתי נרצה להשתמש ב-Javadoc:

- בראש כל מחלקה יש לשים בלוק Javadoc הכולל תיאור של המחלקה. תיאור המחלקה חשוב מאוד; התגים המפורטים כאן למטה - פחות.
- תג ה-author@ מציין מיהו כותב הקוד.
- תג ה-version@ מציין מה גרסת התוכנית.

כמובן שקיימים תגים רבים נוספים.

בראש כל שיטה ציבורית, יש לשים בלוק Javadoc הכולל תיאור של השיטה. כאן תפקיד התגים חשוב מאוד.

- תג ה-param@ מתאר את הפרמטרים אותם מקבלת השיטה. יש לכתוב את שם המשתנה, ואז את תיאורו. על כל משתנה - יש לשים תג param. אין צורך לציין טיפוס - הכלי האוטומטי מסוגל לזהות זאת בעצמו. עם זאת, חשוב להקפיד על כתיבה נכונה של שמות המשתנים.
- תג ה-return@ מתאר מה השיטה מחזירה. כאן אין צורך לכתוב את שם המשתנה, מספיק לכתוב את תיאורו.
- תג ה-throws@ דרוש במקרה והשיטה זורקת חריגות זמן-ריצה כלשהן.
- גם התגים author@ ו-version@ ניתנים לשימוש כאן, אך הם בדרך כלל פחות רלוונטיים.

מה עליכם לעשות?

במחלקה **BankAccount.java** הוספנו תיעוד Javadoc לדוגמא, תוכלו לראות כיצד השתמשנו ב-Javadoc על מנת לתעד את המחלקה ואת הפונקציות הציבוריות שקיימות בה, השתמשו בדוגמא זו ותעדו בעצמכם את המחלקה **Bank.java** עליכם לתעד בראש המחלקה את תפקידה והוסיפו גם את תג ה-author. בנוסף עליכם לתעד את כל הפונקציות הציבוריות במחלקה. מומלץ לקרוא עוד על Javadoc ברשת, בדגש על אופן תיעוד שיטות/מתודות ואופן תיעוד מחלקות.

בהצלחה!