

מבוא למדעי המחשב – סמסטר א' תשע"ט

צוות העבודה:

מרצה אחראי: יוחאי טוויטו

מתרגלים אחראים: אופיר גגולשוילי, ירין קופר

תאריך פרסום: 20.12.18

תאריך הגשה: 3.1.18, 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 השאלות. רמת הקושי של המשימות אינה אחידה.
2. יש להגיש את העבודה לבד.
3. אין לשנות את שמות הקבצים או את חתימות הפונקציות כפי שהן מופיעות בקובצי העבודה.
4. אין להגיש קבצים נוספים.
5. 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.
6. קבצים שיוגשו שלא על פי הנחיות אלו לא ייבדקו.
7. את קובץ ה-ZIP יש להגיש ב-Submission System.
8. 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

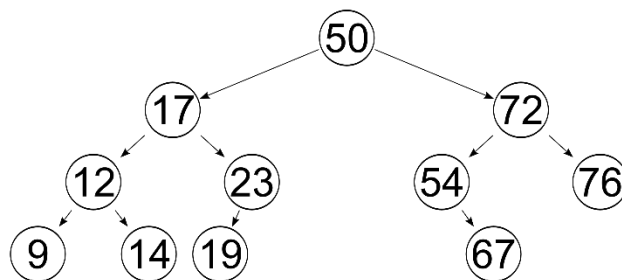
9. עבודות הבית נבדקות באופן ידני וכן באופן אוטומטי. הבדיקה האוטומטית מתייחסת לערכי ההחזרה של הפונקציות או לפעולות אשר הן מבצעות וכן לפלט התכנית המודפס למסך (אם קיים). לכן, יש להקפיד על ההוראות ולבצע אותן במדויק. כל הדפסה אשר אינה עונה בדיוק על הדרישות המופיעות בעבודה (כולל שורות, רווחים, סימני פיסוק או כל תו אחר - מיותרים, חסרים או מופיעים בסדר שונה מהנדרש), לא תעבור את הבדיקה האוטומטית ולכן תגרור פגיעה בציון.
10. סגנון כתיבת הקוד ייבדק באופן ידני. יש להקפיד על כתיבת קוד 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 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.

עזרה והנחיה

11. לכל עבודת בית בקורס יש צוות שאחראי לה (מרצה/ים ומתרגלים). ניתן לפנות לצוות בשעות הקבלה. פירוט שמות האחראים לעבודה מופיע באתר הקורס, כמו גם פירוט שעות הקבלה. בשאלות טכניות אפשר גם לגשת לשעות "עזרה במעבדה". כמו כן, אתם יכולים להיעזר בפורום ולפנות בשאלות לחבריכם לכיתה. צוות הקורס עובר על השאלות ונותן מענה במקרה הצורך. שימו לב, כי פורום העבודה נועד בעיקר לתקשורת ביניכם.
12. בכל בעיה אישית הקשורה בעבודה (מילואים, אשפוז וכו'), אנא צרו את הפנייה המתאימה במערכת הגשת העבודות, כפי שמוסבר באתר הקורס.

הערות ספציפיות לעבודת בית זו

13. לעבודה זו מצורפים קבצי java עם השמות הנדרשים כמפורט בכל משימה. צרו תיקייה חדשה והעתיקו את קובצי ה-java לתוכה. עליכם לערוך את הקבצים האלו בהתאם למפורט בתרגיל ולהגישם כפתרון, מכווצים כקובץ ZIP יחיד. שימו לב: עליכם להגיש רק את קובצי ה-java הנדרשים.
14. בעבודה זו ניתן להגדיר פונקציות (עזר) נוספות, לפי שיקולכם. פונקציות אלו ייכתבו בתוך קובצי המשימה הרלוונטיים.
15. בחלק מהמחלקות המצורפות מופיעה השיטה remove(), התעלמו ממנה. **אין למחוק או לשנות אותה.** היא נחוצה לשם הגשה במערכת ההגשה.
16. הגדרה: עץ בינארי מאוזן הינו עץ בינארי שבו ההפרש בין הגובה של שני תתי-עצים של אותו הצומת לעולם אינו גדול מאחד.



דוגמה:

יושר אקדמי

17. בראש כל קובץ אותו אתם מגישים (כולל קבצים שאתם לא משלימים וניתנו לכם לצורך העבודה) יש למלא את הפרטים שלכם כפי שמופיעים בהערה.
18. בנוסף לקבצי ה-Java יהיה עליכם להגיש את הקובץ readme.txt. בקובץ זה ישנה הצהרה על כך שהעבודה שהגשתם הינה מקורית ונכתבה על ידיכם. עליכם למלא את שמכם ותעודת הזהות שלכם במקום הנדרש לכך בקובץ.

הימנעו מהעתקות! ההגשה היא ביחידים. אם תוגשנה שתי עבודות עם קוד זהה או אפילו דומה - זוהי העתקה, אשר תדווח לאלתר לוועדת משמעת. אם טרם עיינתם ב**סילבוס הקורס** אנא עשו זאת כעת.

חלק 1: איטרטור של מספרים ראשוניים

משימה מספר 0 (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()` האיטראטור ימצא מספר ראשוני יחיד. אין לבצע עיבוד ראשוני של המספרים הראשוניים כשלב מקדים באתחול האיטראטור. חישבו כיצד להשתמש בשדה זה על מנת לממש את האיטראטור 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 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.

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 withdrawMoney(int amount)`

שיטה זו מקבלת פרמטר אחד – amount – אם הוא שלילי המתודה מחזירה false ללא לשנות את היתרה. אם כתוצאה מהמשיכה היתרה (המיוצגת ע"י השדה balance) מקבלת ערך שלילי, אין לאשר את המשיכה, כלומר אין לשנות את היתרה ויש להחזיר false. בכל מקרה אחר יש לעדכן את היתרה ולהחזיר true.

- `public boolean depositMoney(int amount)`

שיטה זו מקבלת פרמטר אחד – amount – אם הוא שלילי המתודה מחזירה false ללא לשנות את היתרה. אחרת יש לעדכן את היתרה ולהחזיר true.

משימה 2: השוואת חשבוניות (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 חשבוניות (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 בפירוטי 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.

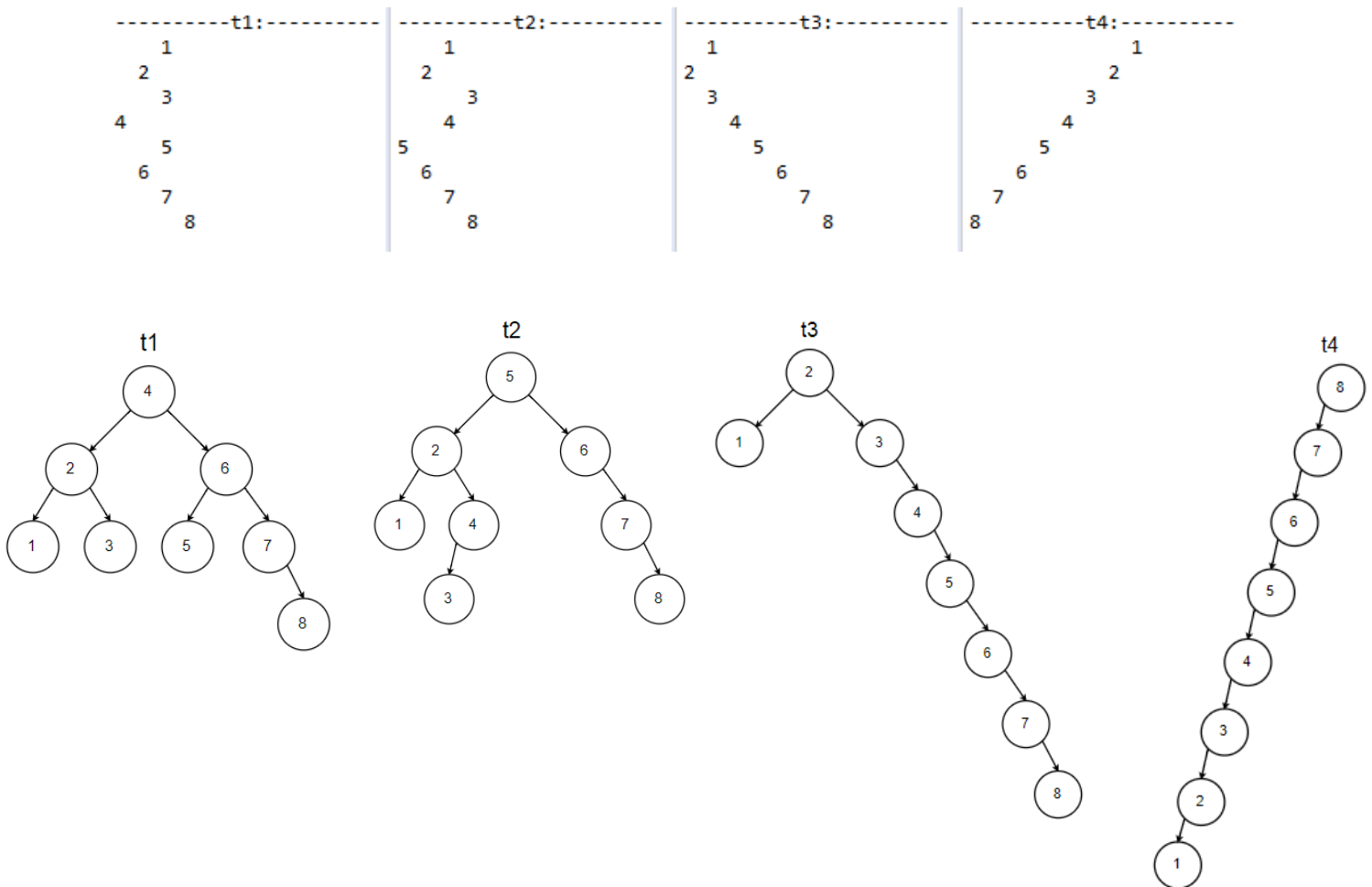
משימה 4: עצים בינאריים (10 נקודות)

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 $2 \cdot d$ 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` ידפיס למסך את הפלטים הבאים (הציורים מיועדים להמחשת מבנה העץ):



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> comparator`

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

למחלקה בנאי יחיד:

- `public BinarySearchTree(Comparator comparator)`

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` 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`.

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 ייחודיים. 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> 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` ובונה 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 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 המורשש בקודקוד המפעיל את השיטה במידה והוא שווה לאחד האיברים בעץ (על פי ה- `Comparator` של העץ). השיטה מחזירה מצביע לשורש העץ המורשש 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).

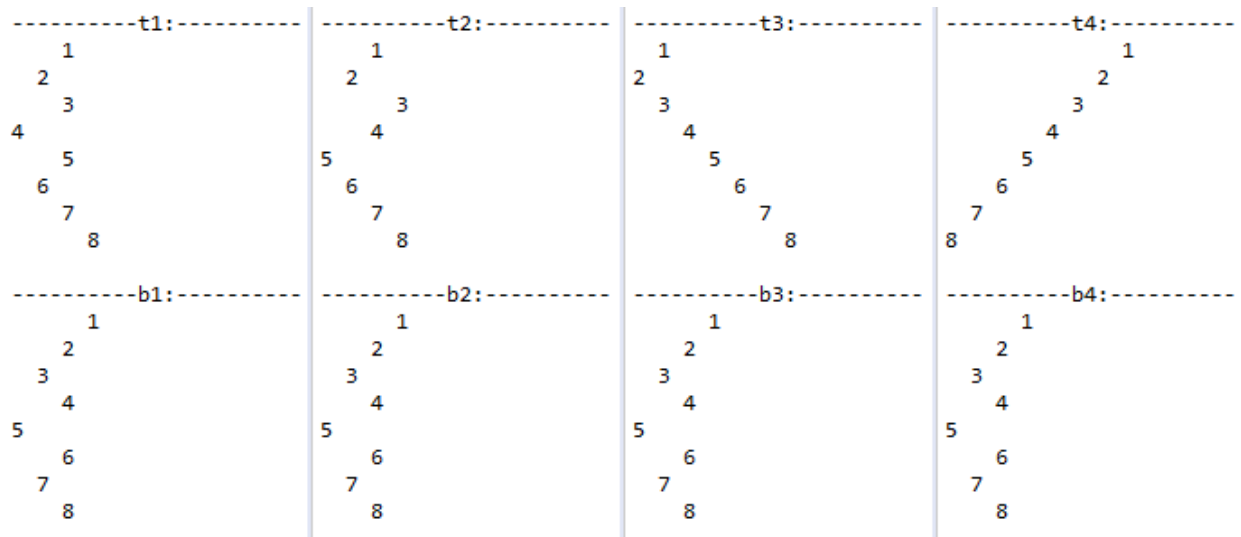
- `private void buildBalancedTree(BankAccountsBinarySearchTree tree, List<BankAccount> list, int low, int high)`

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 (אחת האפשרויות, יתכנו מספר פתרונות לאיזון עץ):



משימה 5ג (10 נקודות)

במשימה זו תשלימו את הגדרת המחלקה

`FilteredBankAccountsIterator<BankAccount> implements Iterator<BankAccount>`

בקובץ `FilteredBankAccountsIterator.java`.

מחלקה זו מממשת את הממשק `Iterator`.

למחלקה 2 שדות:

```
private BankAccountsBinarySearchTree bankAccountsTree
private BankAccount current;
```

עליכם לממש את שלושת המתודות הבאות:

```
public FilteredBankAccountsIterator (BankAccountsBinarySearchTree bankAccountsTree)
עליכם לממש את בנאי המחלקה אשר מקבל עץ של חשבונות בנק ומאתחל את השדות.
```

```
public boolean hasNext()
```

בשיטה הזו עליכם להחזיר `true` אם קיימים איברים נוספים שלא עברנו עליהם, אחרת על הפונקציה להחזיר `false`.

```
public BankAccount next()
```

בשיטה זו עליכם להחזיר את חשבון הבנק הבא ברשימה (inorder), החזירו חשבונות בנק עם יתרה (balance) גדולה מ-100 בלבד. במידה ואין איברים נוספים על הפונקציה 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 בינארי. עצים אלו מכילים את אוסף החשבונות (מסוג BankAccount) הקיים בבנק. בעץ הראשון החשבונות ממויינים 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.

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 accountNumber 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 ש-bankAccount.
 - אין במערכת ניהול הבנק חשבון קיים עם אותו מספר חשבון ש-bankAccount.
 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 accountNumber 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 depositMoney(int amount, int accountNumber) **(2.5 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) **(2.5 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.

בנוס (5 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 במשימה?

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!