מבוא למדעי המחשב – סמסטר א' תש"פ

צוות העבודה: איליה קאופמן, בר סימן טוב, לירון כהן

תאריך פרסום: 5.1.20

תאריך הגשה: 24.1.20, 12:00 בצהריים.

הקדמה

בעבודת בית זו נתרגל את השימוש במבני נתונים שונים.

בעבודה 6 משימות וסך הנקודות המקסימלי הוא 100. הניקוד לכל משימה מפורט במסמך.

בעבודה זו מותר להשתמש בכל הידע שנלמד בקורס.

הוראות מקדימות

הערות כלליות

1. קראו את העבודה מתחילתה ועד סופה לפני שאתם מתחילים לפתור אותה. ודאו שאתם מבינים את כל המשימות.
2. עבודה זו תוגש ביחידים. על מנת להגיש את העבודה יש להירשם למערכת ההגשות (Submission System). את הרישום למערכת ההגשות מומלץ לבצע כבר עכשיו, טרם הגשת העבודה (קחו בחשבון כי הגשה באיחור אינה מתקבלת). את הגשת העבודה ניתן לבצע רק לאחר הרישום למערכת.
3. בכל משימה מורכבת יש לשקול כיצד לחלק את המשימה לתתי-משימות ולהגדיר פונקציות עזר בהתאם.
4. בכל הסעיפים אפשר ומומלץ להשתמש בפונקציות מסעיפים קודמים.
5. לכמה מן המשימות מסופקים קבצי בדיקה, כמפורט בהמשך. באחריותכם לייצר ולהריץ קבצי בדיקה לכלל העבודה.

קבצים

6. ישנם מספר קבצים המצורפים לתרגיל זה. מתוכם תצטרכו לערוך שינויים ולהגיש את הקבצים הבאים:

BinaryNode.java
BinarySearchNode.java
BinarySearchTree.java
BinaryTree.java
BinaryTreeInOrderIterator.java
DynamicArray.java
DynamicArrayIterator.java
Filter.java
FilterByActive.java
FilterByCourse.java
FilteredStudentCardIterator.java
List.java
NumberComparator.java
SimpleIntPair.java
Stack.java
StackAsDynamicArray.java
StringComparator.java
StudentCard.java
StudentCardBinarySearchTree.java
StudentComparatorByAverage.java
StudentComparatorById.java
TestAll.java
University.java

7. בחלק מן הקבצים הללו תערכו שינויים בהתאם למפורט בתרגיל. עליכם להגיש את כל הקבצים שמצויינים ברשימה לעיל.
8. כפתרון, מקובצים כקובץ **ZIP** יחיד. שימו לב: עליכם להגיש רק את קובצי ה־**Java**. אין לשנות את שם הקבצים, ואין להגיש קבצים נוספים, בקובץ ה-**ZIP** אסור שתהיה תיקיה, אלא הקבצים בלבד. שם קובץ ה־**ZIP** יכול להיות כרצונכם, אך באנגלית בלבד. נוסף על כך, הקובץ שתגישו יכול להכיל טקסט המורכב מאותיות באנגלית, מספרים וסימני פיסוק בלבד. טקסט אשר יכיל תווים אחרים (אותיות בעברית, יוונית וכדומה) לא יתקבל. הקפידו לא להשאיר בהגשה חלקי קוד שאינם חלק מהתכנית (לדוגמה, בדיקות שכתבתם עבור עצמכם).
9. קבצים שיוגשו שלא על פי הנחיות אלו לא ייבדקו. את קובץ ה־**ZIP** יש להגיש ב־ Submission System. פרטים לגבי ההרשמה ואיך להגיש את העבודה תוכלו למצוא באתר.
10. בחלק מהקבצים נתונה לכם פונקצית main המכילה טסטים (בהערה) לשימושכם.

בדיקת עבודות הבית

11. עבודות הבית נבדקות גם באופן ידני וגם באופן אוטומטי. הבדיקה האוטומטית מתייחסת אך ורק לפלט המוחזר או מודפס מהפונקציות.
12. שימו לב במשימות בהם אתם קוראים לפונקציה שכתבתם במשימה אחרת (למשל כאשר הפונקציה **f** קוראת לפונקציה **g**): טעות בפונקציה הנקראת (**g**) תגרור טעות גם בפונקציה הקוראת (**f**). משמעות

הדבר הוא שבמקרה כזה תהיה הפחתה בציון עבור פתרון הפונקציה **f**.

13. סגנון כתיבת הקוד ייבדק באופן ידני. יש להקפיד על כתיבת קוד יעיל וברור, 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. יש לתכנן את הקוד בצורה נכונה כך שמשימות מורכבות יחולקו לתתי משימות המבוצעות על ידי פונקציות עזר. כתיבת קוד שאינה עומדת בדרישות אלו תגרור הפחתה בציון העבודה.

<u>עזרה והנחיה</u>

14. 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. כמו כן, אתם יכולים להיעזר בפורום ולפנות בשאלות לחבריכם לכיתה. צוות הקורס עובר על השאלות ונותן מענה במקרה הצורך.
15. בתגבור של השבוע (05.01 עד 08.01) נפתור באופן מודרך את משימות 2.2, 3.2, 6.1, 7.2.
16. בכל בעיה אישית הקשורה בעבודה (מילואים, אשפוז וכו'), אנא פנו אלינו דרך מערכת הפניות, כפי שמוסבר באתר הקורס.
17. אנחנו ממליצים בחום להעלות פתרון למערכת ההגשה לאחר כל סעיף שפתרתם. הבדיקה תתבצע על הגרסה האחרונה שהועלתה (בלבד!).

<u>יושר אקדמי</u>

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 בסילבוס הקורס אנא עשו זאת כעת.

משימה 0: הצהרה (0 נקודות)

פתחו כל אחד מקבצי ה-java הבאים:

StudentCard, StudentComparatorById, StudentComparatorByAverage, BinaryNode, StudentCardBinarySearchTree, FilterByCourse, FilterByActive, FilteredStudentCardIterator, University

וכיתבו בראשם את שמכם ואת מספר תעודת הזהות שלכם.

משמעות פעולה זו היא שאתם מסכימים לכתוב בפסקה הראשונה. 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.

מערכת ניהול סטודנטים

בעבודה זו נממש מערכת לניהול כרטיסי סטודנטים. במערכת אוסף של כרטיסי סטודנטים כך שכל כרטיס מכיל את המידע הבא: שם, מספר תעודת זהות (ייחודי), רשימת קורסים וציוניהם , מספר הקורסים שהושלמו וממוצע ציונים בקורסים שהושלמו.

המערכת תומכת בפעולות הבאות: יצירת מערכת חדשה (ריקה) לניהול סטודנטים, הוספת כרטיס סטודנט חדש, מחיקת כרטיס סטודנט, חיפוש כרטיס סטודנט לפי מספר תעודת זהות, רישום סטודנט לקורס, סיום קורס אליו הסטודנט רשום, חישוב ממוצע כללי בקורס מסוים ומציאת הסטודנטים הרשומים לקורס מסוים.

כדי לתמוך בחיפוש יעיל לפי מספר תעודת זהות ולפי ממוצע ציונים במערכת יתוחזקו שני עצי חיפוש בינאריים. בעץ אחד כרטיסי הסטודנט יהיו ממוינים לפי תעודת זהות ובעץ השני כרטיסי הסטודנט יהיו ממוינים לפי ממוצע הציונים העדכני.

משימה 1: 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. קראו היטב את הקוד בקבצים המתאימים. עליכם להכיר את כל פרטי המחלקות: השדות, הבנאים והשיטות.

- public interface Stack<T>
- public interface List<T>
- public class SimpleIntPair
- public class StackAsDynamicArray<T> implements Stack<T>
- public class DynamicArray<T> implements List<T>
- public class DynamicArrayIterator<T> implements Iterator<T>

משימה 2: מבנה כרטיס הסטודנט (15 נקודות)

כרטיסי הסטודנט במערכת ניהול הסטודנטים מתוארים על ידי הקובץ StudentCard.java.

במחלקה StudentCard השדות הבאים (אין להוסיף שדות):

- private String name

עבור שם הסטודנט.

- private int id

עבור מספר תעודת הזהות של הסטודנט. מספרי תעודות הזהות יהיו יחודיים, וכן נניח את תקינות מספרים אלו בהמשך העבודה.

- private List<SimpleIntPair> courses

עבור רשימת הקורסים אותם למד הסטודנט וציוניהם. קורסים מיוצגים על ידי מספרים חיוביים (ייחודיים). זו היא רשימה של זוגות סדורים כך שהאיבר הראשון בכל זוג הוא מספר הקורס, ואילו האיבר השני הוא ציון הסטודנט.

- private double average

עבור ממוצע הסטודנט בקורסים אותם השלים. במערכת ניהול הסטודנטים שנממש בעבודה זו כל ציון סופי בקורס מעיד על השלמת הקורס. כלומר, גם קורס שהושלם בציון סופי 30 נחשב כקורס שהושלם.

- private int numOfCompletedCourses

עבור מספר הקורסים אותם השלים הסטודנט.

.

- private static final int DEFAULT_NOT_COMPLETED_GRADE = -100

עבור הציון הדיפולטי בו נשתמש עבור קורסים אקטיביים (100-). כלומר, כל קורס אליו הסטודנט רשום אך טרם השלים, ימצא ברשימת הקורסים עם ציון זה.

למחלקה StudentCard בנאי יחיד:

- public StudentCard(String name, int id)

נתונות השיטות הבאות:

- public String getName()
- public int getId()
- public double getAverage()
- public String toString()

קראו היטב את הקוד שבקובץ StudentCard.java. עליכם להכיר את כל פרטי המחלקה, את השדות, הבנאים והשיטות שלה.

עליכם להשלים את השיטות הבאות במחלקה:

- public int hasCourse(int course) **(2.5 נקודות)**

שיטה זו בודקת האם הקורס course מופיע ברשימת הקורסים. אם כן, יוחזר מיקומו ברשימת הקורסים, אחרת יוחזר 1-.

- public boolean isActive() **(2.5 נקודות)**

שיטה זו בודקת האם הסטודנט אקטיבי, כלומר, האם יש לו קורסים אקטיביים (שטרם הושלמו). אם כן, יוחזר הערך true, אחרת יוחזר הערך false.

- public int courseGrade(int course) **(2.5 נקודות)**

שיטה זו מחזירה את ציון הסטודנט בקורס course אם הוא קיים ברשימה, אחרת תזרק חריגה מסוג IllegalArgumentException.

- public boolean registerCourse (int course) **(2.5 נקודות)**

שיטה זו מוסיפה לרשימת הקורסים של הסטודנט קורס שמספרו הוא course רק אם קורס זה אינו קיים כבר ברשימה, ומאתחלת את הציון שלו להיות הציון הדיפולטי עבור קורסים אקטיביים. כדי לתמוך בחיפוש יעיל על רשימת הקורסים להיות ממוינת לפי מספרי הקורסים ולכן ההוספה תתבצע באופן השומר על רשימת הקורסים ממוינת לפי מספר הקורס (עפ"י הסדר הטבעי על int). אם הקורס נוסף לרשימה השיטה תחזיר את הערך true, אחרת יוחזר הערך false.

- public boolean completeCourse (int course, int grade) **(5 נקודות)**

שיטה זו מקבלת מעדכנת את ציון הסטודנט בקורס course לציון grade. אם הציון חורג מהערכים התקינים (0-100) השיטה תזרוק חריגה מסוג IllegalArgumentException. אם הסטודנט תלמיד פעיל בקורס course (כלומר, הקורס מצוי ברשימת הקורסים של הסטודנט עם הציון 100-), השיטה מעדכנת את הציון בקורס להיות grade ומחזירה את הערך true. אחרת, אם הקורס אינו נמצא או שהוא נמצא אך אינו אקטיבי היא מחזירה את הערך false. שימו לב שעליכם לתחזק את השדות average ו-numOfCompletedCourses בהתאם.

סיימתם חלק זה ?כל הכבוד !העלו את הגרסה האחרונה של עבודתם למערכת ההגשה.

משימה 3: השוואת סטודנטים (5 נקודות)

במשימה זו תשלימו את הגדרת המחלקות הבאות בקבצים שקיבלתם.

- public class StudentComparatorById implements Comparator<StudentCard>

מחלקה זו מממשת את השיטה המוגדרת בממשק Comparator<StudentCard> :

public int compare(StudentCard student1, StudentCard student2) .

שיטה זו משווה בין כרטיסי סטודנטים לפי מספר תעודת זהות (לפי יחס הסדר הטבעי על מספרים). אם אחד מהקלטים null יש לזרוק חריגה מטיפוס IllegalArgumentException.

כמו כן, השלימו את הגדרת המחלקה הבאה בקבצים שקיבלתם.

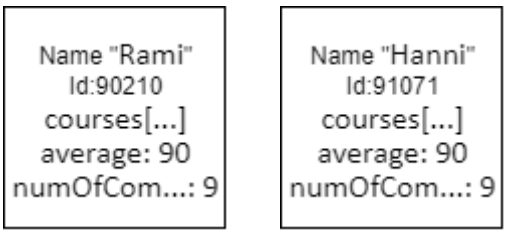
- public class StudentComparatorByAverage implements Comparator<StudentCard>

מחלקה זו מממשת את השיטה המוגדרת בממשק Comparator<StudentCard>:

public int compare(StudentCard student1, StudentCard student2) .

שיטה זו משווה בין כרטיסי סטודנטים לפי ממוצעי הציונים שלהם (לפי יחס הסדר הטבעי על מספרים), ואם הממוצעים שווים עוברת להשוות לפי מספרי תעודות הזהות (לפי יחס הסדר הטבעי על מספרים). אם אחד מהקלטים null יש לזרוק חריגה מטיפוס IllegalArgumentException.

לדוגמה, עבור הסטודנטים בדוגמה מטה, הסטודנטית Hanni גדולה יותר מהסטודנט Rami.



שימו לב שבקובץ studentComparator.java מופיעה השורה import java.util.Comparator. זהו הממשק Comparator כפי שמוגדר ב- java. מומלץ להיזכר בפרטי הממשק Comparator כפי שמתואר ב API של java.

סיימתם חלק זה ?כל הכבוד !העלו את הגרסה האחרונה של עבודתם למערכת ההגשה.

משימה 4: עצים בינאריים (10 נקודות) -- **רשות**

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

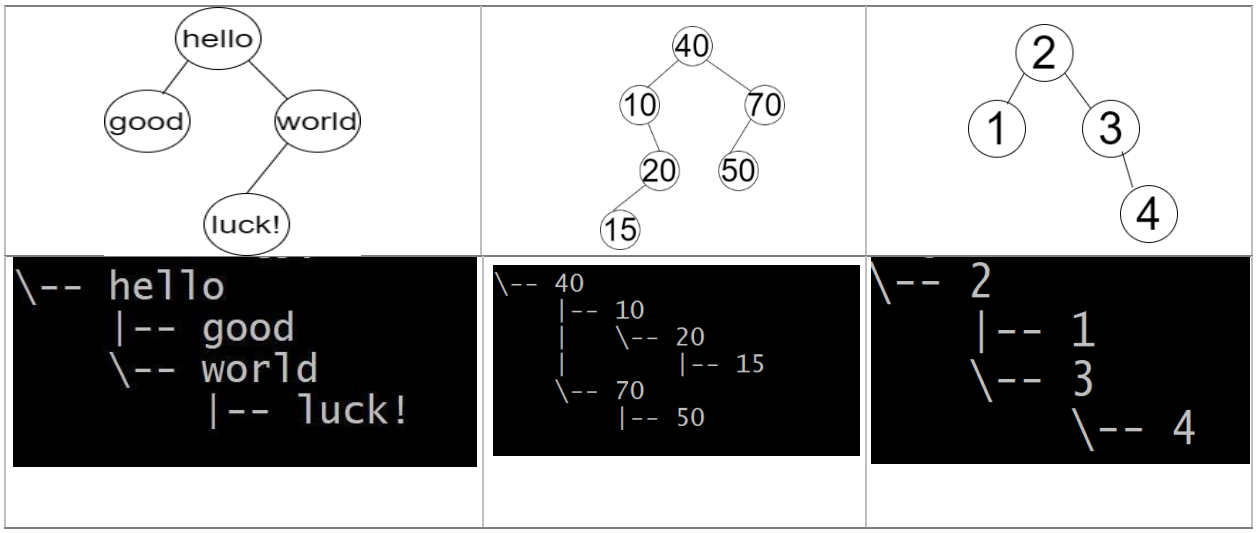
- public String toString()

השיטה toString() במחלקה BinaryTree נתונה לכם. אם העץ אינו ריק היא קוראת לשיטה toString() שבמחלקה BinaryNode. במחלקה BinaryNode השיטה פועלת כך שאם נדפיס את המחרוזת שהיא מחזירה נקבל הצגה ויזואלית של העץ באופן הבא:

- כל צומת בעץ יודפס בשורה נפרדת.
- לאחר כל קודקוד יודפסו קודם השורות שמגדירות את ה tostring של תת-העץ השמאלי שלו ולאחר מכן השורות שמגדירות את ה tostring של תת-העץ הימני שלו.
- לפני בן שמאלי יודפס "--|" מלווה ברווח אחד ואילו לפני בן ימני יודפס "--\" מלווה ברווח אחד (בשני המקרים הרווח בין הסימון לבן).
- שורש העץ יחשב כבן ימני.
- בין שני בנים של אותה צומת יודפס "|" מלווה בשלושה רווחים (עבור כל שורה המפרידה בניהם).
- מתחת בן ימני יודפסו ארבעה רווחים (עבור כל שורה מתחתיו).

במשימה זו מומלץ להשתמש בשיטות עזר פרטיות. נוח יותר לחשוב על פתרון רקורסיבי לבעיה הזו.

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



סיימתם חלק זה ?כל הכבוד !העלו את הגרסה האחרונה של עבודתם למערכת ההגשה.

משימה 5: עצי חיפוש בינאריים (20 נקודות)

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> וממשת 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 משתנה מטיפוס Comparator ובונה עץ חיפוש ריק.

<u>נתונות השיטות הבאות:</u>

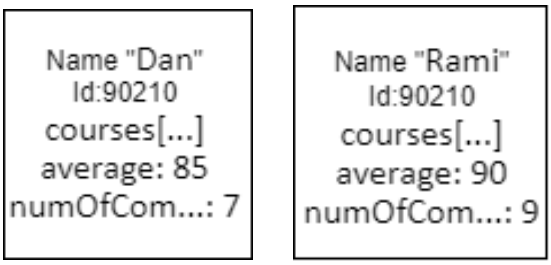
- public T findData(T element)

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.

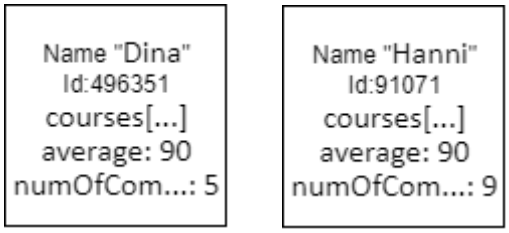
שיטה זו מקבלת אובייקט element. השיטה מחפשת ומחזירה את ה- data השווה ל- element (על פי ה-Comparator) הנמצא בעץ המפעיל את השיטה, במידה וקיים. במידה ולא קיים בעץ קודקוד עם שדה data השווה ל-element (על פי ה- Comparator), השיטה מחזירה ערך null.

דוגמאות:

1. בקריאה לשיטה זו כאשר ה- Comparator של העץ הוא מטיפוס StudentComparatorById , element מפנה אל כרטיס הסטודנט של Dan (המתואר מטה) והעץ מכיל את כרטיס הסטודנט של Rami (המתואר מטה) תוחזר הפנייה לכרטיס הסטודנט של Rami.



2.בקריאה לשיטה זו כאשר ה- Comparator של העץ הוא מטיפוס studentComparatorByAverage, element מפנה אל כרטיס הסטודנט של Hanni(המתואר מטה), והעץ מכיל את כרטיס הסטודנט של Dina (המתואר מטה) תוחזר הפניה לכרטיס הסטודנט של Dina.



- 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 הסטודנטים שנממש מספרי תעודות הזהות הם ייחודיים, וגם שילוב ממוצעי הציונים עם תעודות הזהות. אם האובייקט 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 אוביקט 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 ורק 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**

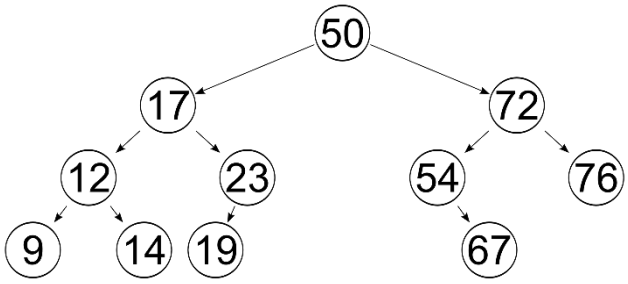
נתונה לכם המחלקה BinaryTreeInOrderIterator המממשת את הממשק Iterator של java. שימו לב כי בקובץ המחלקה מופיעה השורה import java.util.Iterator;. איטרטור זה עובר על המידע השמור בעץ החיפוש לפי סדר inorder.

משימה 5ב: (20 נקודות)

במשימה זו נשלים שיטות שתפקידן לייצר עץ מאוזן.

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

דוגמה:



המחלקה StudentCardBinarySearchTree יורשת את המחלקה BinarySearchTree<StudentCard>.

למחלקה בנאי יחיד:

- public StudentCardBinarySearchTree(Comparator<StudentCard> 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 הרקורסיבית buildBalancedTree.

הדרכת חובה: את השיטה ()balance יש להשלים בעזרת שיטת העזר הפרטית הבאה (אין להוסיף שיטות עזר נוספות).

- private void buildBalancedTree(StudentCardBinarySearchTree tree, List<StudentCard> 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 כרטיסי הסטודנט שבעץ על פי סדר ה-inorder שלהם בעץ.
- האינדקס 0.
- האינדקס list.size()-1.

עליכם להשלים את השיטה באופן רקורסיבי, כך שכל כרטיסי הסטודנט שברשימה יוכנסו לעץ. בסוף התהליך העץ tree יכיל את כל כרטיסי הסטודנט שברשימה ויהיה מאוזן (ראו הגדרה בתחילת העבודה). נחזור ונדגיש כי סדר ה-inorder של כרטיסי הסטודנט 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 (אחת האפשרויות, יתכנו מספר פתרונות לאיזון עץ) :

```
----------t1:----------
\-- Hanni 91071
    |-- Ran 90210
    \-- Dina 496351
        |-- Zohar 317317

------b1 balance:------
\-- Zohar 317317
    |-- Hanni 91071
    |   |-- Ran 90210
    \-- Dina 496351

----------t2:----------
\-- Hanni 91071
    |-- Ran 90210
    |   |-- Dina 496351
    |   |   |-- Zohar 317317

------t2 balance:------
\-- Ran 90210
    |-- Dina 496351
    |   |-- Zohar 317317
    \-- Hanni 91071
```

משימה 6: איטראטורים (15 נקודות)

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

- public boolean accept(T element)

סיימתם חלק זה ?כל הכבוד !העלו את הגרסה האחרונה של עבודתם למערכת ההגשה.

שיטה זו תחזיר ערך true אם element עובר את הפילטר וערך false אחרת.

עליכם להשלים את הגדרת שתי המחלקות הבאות הממשות את הממשק :Filter<StudentCard>

- public class FilterByCourse implements Filter<StudentCard> **(2.5 נקודות)**
  בקובץ FilterByCourse.java.
  מחלקה זו מממשת פילטר המעביר רק סטודנטים אשר סיימו קורס מסוים.

- public class FilterByActive implements Filter<StudentCard> **(2.5 נקודות)**
  בקובץ FilterByActive.java.
  מחלקה זו מממשת פילטר המעביר רק סטודנטים אשר יש להם קורסים פעילים (עם ציון 100-).

כעת עליכם להשלים את הגדרת המחלקה

- public class FilteredStudentCardIterator implements Iterator<StudentCard>
  בקובץ FilteredStudentCardIterator.java.
  מחלקה זו מממשת את הממשק Iterator.

למחלקה 3 שדות:

```
private BinaryTreeInOrderIterator<StudentCard> iterator;
private StudentCard current;
private Filter<StudentCard> filter;
```

עליכם לממש את הבנאי ושתי השיטות הבאות:

- public FilteredStudentCardIterator(StudentCardBinarySearchTree StudentCardsTree, Filter<StudentCard> filter) **(2.5 נקודות)**
  עליכם לממש את בנאי המחלקה אשר מקבל עץ של כרטיסי סטודנט ומאתחל את השדות.

- public boolean hasNext() **(2.5 נקודות)**
  בשיטה הזו עליכם להחזיר ערך true אם קיימים איברים נוספים שלא עברנו עליהם המקיימים את דרישת הפילטר. אחרת על הפונקציה להחזיר ערך false.

- public StudentCard next() **(5 נקודות)**
  בשיטה זו עליכם להחזיר את כרטיס הסטודנט הבא ברשימה (מסדר inorder) המקיים את דרישת הפילטר. אם אין איברים נוספים על הפונקציה לזרוק חריגה מסוג NoSuchElementException.

סיימתם חלק זה ?כל הכבוד !העלו את הגרסה האחרונה של עבודתם למערכת ההגשה.

משימה 7: מערכת מנהל תלמידים (40 נקודות)

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

private StudentCardBinarySearchTree idTree;
private StudentCardBinarySearchTree avgTree;
שהם עצי חיפוש בינאריים. עצים אלו מכילים את אוסף כרטיסי הסטודנט (מסוג StudentCard) של הסטודנטים באוניברסיטה. בעץ הראשון כרטיסי הסטודנט ממוינים לפי מספרי תעודות זהות ובעץ השני לפי ממוצעי ציונים ומספרי תעודות הזהות. נדגיש כי כל סטודנט קיים במערכת ניהול האוניברסיטה רק פעם אחת, ובכל עץ קיים לו קודקוד ובו שדה StudentCard data המפנה אליו.

בנאי המחלקה public University() מגדיר מערכת ריקה לניהול סטודנטים (עם שני עצי חיפוש ריקים).

נתונות השיטות הבאות (אין לשנות את הגדרתן):

- public StudentCard lookUp(int idNumber)

שיטה זו מקבלת מספר idNumber ומחזירה את הסטודנט במערכת עם מספר תעודת הזהות idNumber אם קיים בזה. אחרת השיטה תחזיר את הערך null.

- public void balance()

שיטה זו מיודעת לשמירה על יעילות השימוש במערכת ניהול הסטודנטים.
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 StudentCardBinarySearchTree(קריאה אחת לכל אחד מהעצים).

עליכם להשלים את השיטות הבאות במחלקה:

- public boolean add(StudentCard newStudent) **(נקודות 5)**

שיטה זו מקבלת כרטיס סטודנט newStudent ומוסיפה אותו למערכת ניהול הסטודנטים אם אין במערכת סטודנט קיים עם אותה תעודת זהות שב- newStudent. הפונקציה מחזירה ערך true אם ההוספה בוצעה בהצלחה ומחזירה ערך false אחרת.
יש להוסיף את אותו הסטודנט למקום המתאים בשני העצים המוגדרים בשדות המחלקה.

- public boolean delete(StudentCard newStudent) **(נקודות 5)**

שיטה זו מקבלת מספר כרטיס סטודנט newStudent ומוחקת את הכרטיס במערכת ניהול הסטודנטים אם קיים בזה. זיכרו כי אם הסטודנט קיים יש להסיר את ההפניה אליו משני העצים. השיטה מחזירה ערך true אם התבצעה מחיקה וערך false אחרת.

- public boolean register(int id, int course) **(נקודות 2.5)**

שיטה זו מקבלת מספר קורס courseומספר תעודת זהות id, מוצאת את כרטיס הסטודנט המתאים, וקוראת לשיטה register(course) עבור סטודנט זה. השיטה מחזירה ערך true אם הפעולה הצליחה וערך false אחרת.

- public boolean complete(int id, int course, int grade) **(נקודות 10)**

שיטה זו מקבלת מספר קורס course, ציון grade ומספר תעודת זהות id, מוצאת את כרטיס הסטודנט המתאים, קוראת למתודה completeCourse(course, grade) עבור סטודנט זה. השיטה מחזירה ערך true אם הפעולה הצליחה וערך false אחרת. שימו לב כי יש גם למקם מחדש את הסטודנט בעץ הממוין לפי ממוצעים.

- public double courseAverage (int course) **(נקודות 7.5)**

שיטה זו מקבלת מספר קורס ,course, ומחזירה את ממוצע הציונים בקורס (עבור סטודנטים שהשלימו את הקורס בלבד). אם אין סטודנטים שהשלימו את הקורס על הפונקציה לזרוק חריגה מסוג NoSuchElementException.

- public void activeStudentsByAverage() **(נקודות 10)**

שיטה זו מדפיסה את כל הסטודנטים האקטיביים (שיש להם קורסים פעילים) לפי סדר ממוצע ציוניהם. כלומר, הסטודנט האקטיבי עם ממוצע הציונים הגבוה ביותר יודפס ראשון וזה עם הממוצע הנמוך ביותר אחרון. אם ישנם כמה סטודנטים עם אותו ממוצע סדר יודפסו לפי סדר תעודות הזהות. כל סטודנט יודפס בשורה חדשה. אם אין סטודנטים אקטיביים, לא יודפס דבר.

סיימתם חלק זה ? כל הכבוד ! העלו את הגרסה האחרונה של עבודתם למערכת ההגשה.

בהצלחה!