

מבוא למדעי המחשב - סמסטר א' תשפ"ב

עבודת בית מספר 4

צוות העבודה:

- מתרגלים אחראים: לירן נחומסון, גל דהן.
- מרצה אחראי: יוחאי טוויטו.

תאריך פרסום: 12/12/2021

מועד אחרון להגשה: 26/12/2021

מה בתיגבור:

- בתגבורים של 13-16.12.21 נפתור את משימות: 1.1 (סעיף ב), 2.1 (עבור השיטה `addFirst()`), 2.8 ו-3.1.

תקציר נושא העבודה:

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

- ייצוג (בינארי) של מספרים גדולים, וביצוע פעולות אריתמטיות עליהם.
- דריסה של השיטות המתקבלות בירושה מהמחלקה `Object`.
- שימוש ברשימות מקושרות ומעבר עליהן באמצעות איטרטורים.
- מימוש הממשק `Comparable`.
- מימוש בנאים מסוגים שונים.
- שימוש בחריגות.

בעבודה זו 22 משימות וסך הנקודות המקסימלי הוא 100. בעבודה זו מותר להשתמש בידע שנלמד עד הרצאה 17 (כולל), וכן עד תרגול 10 (כולל).

בוודאי שמתם לב שהמסמך של עבודה זו ארוך יותר מאשר אלו של העבודות הקודמות. הסיבה המרכזית היא שישנם הרבה הסברים ודוגמאות. נציין בפרט שישנן שיטות רבות בעבודה שהמימוש שלהן קצר.


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 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
- עבודת הבית עוסקת בשלושה קבצים: `BitList.java`, `Bit.java`, `BinaryNumber.java`.
- יש להיעזר בנספח עבודת הבית המכסה מושגים ושיטות שונות לעבודה עם מספרים בינאריים.

המלצה על דרך העבודה - אנו ממליצים לפתוח ב-Eclipse פרויקט בשם Assignment4 ולהעתיק אליו את הקבצים שתורידו מה-vpl.

החלק הראשון של המטלה הוא השלמת הקובץ Bit.java, החלק השני של המטלה הוא השלמת הקובץ BitList.java, והחלק השלישי של המטלה הוא השלמת הקובץ BinaryNumber.java. את העבודה תגישו בשתי קבצי VPL שונים. הראשון עבור Bit.java ו- BitList.java והשני עבור BinaryNumber.java.

הנחיות לכתיבת קוד והגשה

- בקבצי השלד המסופקים לכם קיים מימוש ברירת מחדל לכל משימה. יש למחוק את מימוש ברירת המחדל בגוף השיטות 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 (אינדנטציות), 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 עלולה לגרור הפחתה בציון העבודה.

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. ואין לפנות לחברי צוות הקורס שאינם אחראים על עבודת הבית הנוכחית.
- ניתן להיעזר בפורום. צוות האחראיים על העבודה של 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!).

יושר אקדמי

הימנעו מהעתקות! ההגשה היא ביחידים. אם מוגשות שתי עבודות עם קוד זהה או אפילו דומה – זהו חשד להעתקה, אשר תדווח לאלתר לוועדת משמעת. אם טרם עיינתם [בסילבוס הקורס](#) אנא עשו זאת כעת.

=====

הוראות מקדימות

הערות כלליות

קבצים

משימה 0: הצהרה (0 נקודות)

ב-vpl פתחו את הקבצים של החלק הראשון וגם של החלק השני וכתבו בראשם 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. בכל המשימות בעבודה **אין להניח שהקלט תקין**. אם הקלט אינו תקין עליכם לזרוק חריגה מטיפוס `IllegalArgumentException` בלבד. אין להשתמש ב-`NullPointerException`. החריגה צריכה לקבל כפרמטר מחרוזת עם הודעת שגיאה משמעותית.
2. בעבודה זו אתם יוצרים את השיטה הציבורית `toString()` במחלקה `BinaryNumber`. **אין לקרוא לה מאף שיטה אחרת שאתם כותבים**.
3. בכל אחת מהמשימות מותר להוסיף שיטות עזר כראות עיניכם.
4. עבודה זו משתמשת בשלושה ממשקים מובנים של ג'אווה:
 - Comparable - <https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html>
 - Iterator - <https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html>
 - Iterable - <https://docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html>
5. חלק מההערות המובאות בקוד הן בסטנדרט Javadoc, אתם מוזמנים לבצע חיפוש של המילה Javadoc ברשת האינטרנט ולקרוא על פורמט התיעוד בסטנדרט זה (עשוי לסייע לכם בהבנת התיעוד, ובעבודה הבאה).
6. בכל אחד מקבצי הג'אווה שאתם מקבלים עם העבודה ישנם בנאים ו\או שיטות שעליכם להשלים לפי ההנחיות שבעבודה זו. בכל אחד מהם מופיעה השורה


```
throw new UnsupportedOperationException("Delete this line and implement the method.");
```

 יש למחוק את השורה כולה (החל מהמילה `throw` ועד הנקודה פסיק) ולכתוב מימוש מלא לבנאי/שיטה.
7. אין לשנות או להוסיף שדות למחלקות, ואין לשנות בנאים ריקים, את כותרות המחלקות ואת החתימות של השיטות והבנאים הציבוריים. כל המחלקות והממשקים שנדרשים לעבודה כבר יובאו בקבצים. אין לייבא מחלקות וממשקים נוספים.
8. מותר ורצוי להוסיף שיטות ובנאים פרטיים כדי למנוע שכפול קוד ולשפר את הקריאות של הקוד. אם אתם יוצרים שיטה או בנאי פרטיים הקפידו להסביר בהערה מה היא הפעולה שהם עושים. הוסיפו הערה כזו גם במקומות שאתם קוראים לשיטות ובנאים אלו. הקפידו על שמות משמעותיים לשיטות.
9. העבודה מתבססת על המחלקה `LinkedList` מהספרייה הסטנדרטית של ג'אווה. קריאה חוזרת ונשנית לשיטה `get` המוגדרת במחלקה זו היא מאוד לא יעילה במחלקה זו. פתרונות יעילים משתמשים באיטרטור ככל שזה ניתן.
10. לנוחיותכם, הוטמעו במערכת ה-vpl בדיקות לנכונות של השיטות שכתבתם. שימו לב שהבדיקות הללו מבצעות את פעולות החשבון על מספרים גדולים מאוד, כגון מספרים בסדר גודל של האיבר המאה בסדרת פיבונאצ'י.
11. נדגיש למען הסר ספק: אין להשתמש בשום פנים ואופן במחלקה `BigInteger` של ג'אווה! שימוש במחלקה זו יגרום לפסילת החלק בו נעשה שימוש בטיפוס זה.

מוטיבציה

הבסיס לעבודה זו (ולכול האריתמטיקה במחשבים) הן שתי הספרות הבינאריות, המכונות ביטים, 0 ו- 1 . בשפה ג'אווה, הטיפוסים הפרימיטיביים המייצגים מספרים שלמים (`byte`, `short`, `int` ו-`long`) משתמשים במספר קבוע של ביטים (8, 16, 32 ו-64 בהתאמה) ולכן יש מגבלות מובנות על הגודל המקסימאלי של המספרים שהם יכולים לייצג. כך למשל הטיפוס `byte` מיוצג על ידי 8 ביטים ולכן יכול לייצג 2^8 מספרים שונים (מ-128 עד 127). מספרים גדולים יותר מהערך המקסימאלי המיוצג על ידי הטיפוס `long` ($2^{63}-1$) או קטנים יותר מהערך המינימאלי שהוא מייצג (-2^{63}) אפשר לייצג בג'אווה רק על ידי טיפוסים מורכבים. כבר בעבודת הבית הראשונה של קורס זה הבנו את הצורך במספרים כאלו ובעבודה 3 הכרנו את הפתרון הסטנדרטי של ג'אווה, המחלקה `BigInteger`. **בעבודה זו נעסוק בייצוג מספרים ע"י רשימות מקושרות של עצמים מהמחלקה `Bit`**, שאותה התחלתם לממש בעבודת בית מספר 3. לשם כך ניצור שתי מחלקות: `BitList` המייצגת רשימה של ביטים ו-`BinaryNumber` המייצגת מספרים שלמים חיוביים או שליליים. המספרים המיוצגים על ידי עצמים מהמחלקה `BinaryNumber` עשויים להיות גדולים או קטנים כרצוננו.

חלק ראשון - השלמת המחלקה `Bit` (VPL חלק 1)

את המחלקה `Bit` המייצגת ביט (ספרה בינארית) פגשתם בעבודת הבית מספר 3. כעת אתם מקבלים אותה (בשינויים קלים) בקובץ `Bit.java`. ועליכם להשלים בה שתי שיטות סטטיות שיוסברו בהמשך.

המחלקה כוללת:

1. בנאי המקבל ערך בוליאני ויוצר עצם המייצג את הביט 1 אם הפרמטר הוא `true` ו- 0 אם הפרמטר הוא `false`.
2. בנאי המקבל מספר מטיפוס `int` ויוצר עצם המייצג את הביט 1 אם הפרמטר הוא המספר אחד ו- 0 אם הפרמטר הוא המספר אפס, אחרת תוחזר חריגה.
3. שני משתנים סטטיים `ONE` ו-`ZERO` המפנים לביטים המייצגים 1 ו- 0 בהתאמה.
4. השיטות `toInt()` ו-`toString()` המחזירות ייצוג של ביט כ-`int` (0 או 1) ומחרוזת ("1" או "0") בהתאמה.
5. השיטה `equals(Object)` הדורסת את השיטה של המחלקה `Object`. השיטה מקבלת פרמטר מטיפוס `Object` ומחזירה ערך `true` אם ורק אם הפרמטר הוא ביט בעל ערך זהה לערכו של העצם הפועל (העצם שמפעיל את השיטה).
6. השיטה `negate` מחזירה ביט שערכו הפוך לערך של הביט המבצע את הפעולה.

דוגמה:

```
Bit b1 = new Bit(true);
Bit b2 = b1.negate();
System.out.println(b1+" "+b2);    // prints 1 0
Bit b3 = new Bit(1);
System.out.println(b1.equals(b3)); // prints true
```

משימה 1.1: השיטות fullAdderSum(Bit, Bit, Bit) ו-fullAdderCarry(Bit, Bit, Bit)

A	B	Cin	carry	sum
1	1	1	1	1
1	1	0	1	0
1	0	1	1	0
1	0	0	0	1
0	1	1	1	0
0	1	0	0	1
0	0	1	0	1
0	0	0	0	0

חיבור של שלושה ביטים (קלט), שנסמן ב-a, b ו-Cin, הוא פעולה אריתמטית בסיסית, שהפלט שלה הוא זוג ביטים: ביט סכום (sum) וביט נשא (carry). בסך הכל ייתכנו שמונה שלישיות קלט. שלישיות אלו והפלט של פעולת החיבור שלהן מוצגים בטבלה משמאל.

ניתן לראות את הערכים בשתי העמודות הימניות של הטבלה כספרות של מספר בינארי שהוא הסכום של שלושת הערכים בשלוש העמודות השמאליות.

רכיב אלקטרוני שמממש חיבור של שלוש ספרות נקרא full-adder. רכיבים כאלו הם מרכיבים עיקריים במערכות דיגיטאליות ובפרט במחשבים

[https://en.wikipedia.org/wiki/Adder_\(electronics\)](https://en.wikipedia.org/wiki/Adder_(electronics))

א. ממשו את השיטה

Bit fullAdderSum(Bit A, Bit B, Bit Cin)

המקבלת שלושה ביטים ומחזירה את ביט הסכום של חיבורם.

ב. ממשו את השיטה **Bit fullAdderCarry(Bit A, Bit B, Bit Cin)** המקבלת שלושה

ביטים ומחזירה את ביט הנשא של חיבורם. [סעיף זה יפתר בתגבור השבועי].

הנחיות:

- ניתן להניח שהקלט תקין, כלומר שאף אחד מן הפרמטרים אינו null.
- הקפידו על קוד פשוט ונקי. מימוש הטבלה באופן ישיר יקבל ניקוד חלקי, חישבו כיצד ניתן לממש את השיטה באמצעות חישוב אריתמטי.

דוגמה (שימו לב שניתן לראות את השורות המודפסות כמספרים בינאריים בני שתי ספרות שהן הסכום של ערכי שלושת הביטים של הקלט):

```
public static void main(String[] args) {
    Bit b1 = new Bit(true);
    Bit b0 = new Bit(false);
    System.out.println(Bit.fullAdderCarry( b0, b0, b0)+" "+
        Bit.fullAdderSum( b0, b0, b0)); // prints 0 0
    System.out.println(Bit.fullAdderCarry( b1, b0, b0)+" "+
        Bit.fullAdderSum( b1, b0, b0)); // prints 0 1
    System.out.println(Bit.fullAdderCarry( b1, b1, b0)+" "+
        Bit.fullAdderSum( b1, b1, b0)); // prints 1 0
    System.out.println(Bit.fullAdderCarry( b1, b1, b1)+" "+
        Bit.fullAdderSum( b1, b1, b1)); // prints 1 1
}
```

סיימתם חלק זה? כל הכבוד! שמרו את הגרסא האחרונה של עבודתכם במערכת ה-vpl.

חלק שני - השלמת המחלקה BitList (VPL חלק 1)

רקע

בעבודה זו אנחנו מייצגים מספרים באמצעות רשימות מקושרות של עצמים מטיפוס Bit. בחלק זה של העבודה נשלים את הגדרת המחלקה BitList המרחיבה את המחלקה LinkedList<T> שקיימת בספריה הסטנדרטית של ג'אווה. המחלקה BitList מספקת את השיטות הבסיסיות בהן משתמשת המחלקה BinaryNumber, אותה נשלים בחלק השלישי של העבודה. לפני שנתחיל בעבודה עלינו להתוודע לגרסה חדשה, מבחינתנו, של המחלקה LinkedList, ולהכיר מספר מושגים.

המחלקה LinkedList<T> בספריה הסטנדרטית של ג'אווה:

בשיעור כתבנו מחלקה בשם LinkedList<T> המממשת את הממשק List<T>. גם הספריה הסטנדרטית של ג'אווה מציעה מחלקה כזו, שהיא מורכבת יותר ונותנת הרבה יותר שיטות. הרשימה המלאה של השיטות והבנאים של גרסה זו של LinkedList נמצאת ב-API של ג'אווה:

<https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html>

להלן מספר שיטות שהן שימושיות במיוחד לעבודה זו:

`int size()` – היא שיטה המחזירה את מספר האיברים ברשימה. **שימו לב**, המימוש של שיטה זו הוא יעיל ואינו תלוי באורך הרשימה.

`void addFirst(T element)` ו-`void addLast(T element)` – מוסיפות איברים לרשימה במקום הראשון ובמקום האחרון בהתאמה. **שימו לב**, המימוש שלהן יעיל ואינו תלוי באורך הרשימה. שימו לב גם ששיטות אלו מאפשרות להוסיף לרשימה את הערך `null`.

`T removeFirst()` ו-`T removeLast()` – מסירות מהרשימה את האיבר הראשון והאחרון בהתאמה ומחזירות את האיבר שהוסר מטיפוס T. **שימו לב**, המימוש של שיטות אלו יעיל ואינו תלוי באורך הרשימה.

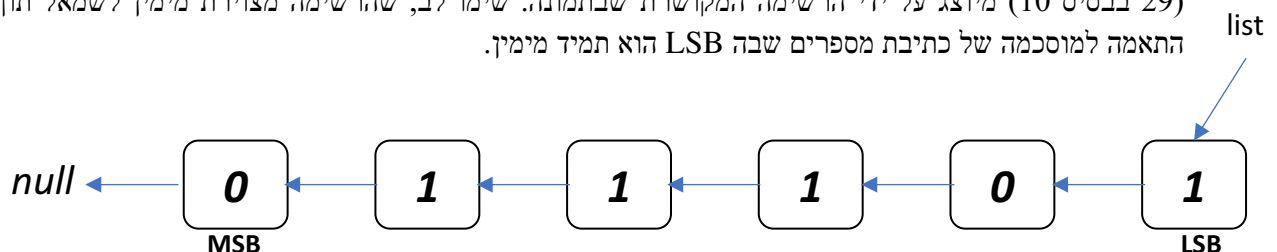
`T get(int i)` – השיטה מחזירה את הערך במקום ה-`i`. **שימו לב**, שיטה זו אינה יעילה.

`Iterator<T> iterator()` – השיטה מחזירה איטרטור, שעובר בצורה יעילה על אברי הרשימה מתחילת הרשימה לסופה.

לפני שאתם ממשיכים, גשו לנספח וקראו אותו. הנספח מפרט הרבה מושגים וכלים בהם תשתמשו בעבודה.

שימוש ברשימה מקושרת לייצוג בינארי של מספרים:

בעבודה זו אנחנו משתמשים ברשימה של איברים מהטיפוס Bit כדי לממש ייצוג בינארי מינימאלי של מספרים. האיבר הראשון ברשימה הוא הספרה הפחות משמעותית (Least Significant Bit, LSB) והאיבר האחרון ברשימה יהיה הספרה המשמעותית ביותר (Most significant Bit, MSB). לדוגמה, המספר הבינארי **011101** (29 בבסיס 10) מיוצג על ידי הרשימה המקושרת שבתמונה. שימו לב, שהרשימה מצוירת מימין לשמאל תוך התאמה למוסכמה של כתיבת מספרים שבה LSB הוא תמיד מימין.



בפרט, המספר אפס ייוצג על ידי רשימה שבה איבר אחד. ביט שמייצג את הערך 0.



השלמת המחלקה BitList

המחלקה BitList מרחיבה את המחלקה LinkedList, מהספריה הסטנדרטית של ג'אווה. יש לה שדה אחד `numberOfOnes` המייצג את מספר הביטים שערכם 1.

המחלקה BitList מייצגת רשימה של ביטים. רשימות כאלו יכולות לייצג מספרים בינאריים מינימאליים בשיטת המשלים ל-2, אבל רשימות רבות אינן מייצגות מספרים כלל (למשל הרשימה הריקה) או מייצגות אותם בצורה לא מינימאלית.

השיטות של המחלקה LinkedList מאפשרות הכנסה של הערך `null` לרשימה. כדי לוודא שברשימת הביטים לא מופיע `null` ושערך השדה `numberOfOnes` תמיד מייצג את המצב של העצם, יש לדרוס את כל השיטות שמכניסות ומציאות איברים מהרשימה. בקובץ `BitList.java` שקיבלתם השיטות כבר דרוסות וזורקות חריגת `UnsupportedOperationException`. עליכם להשלים ארבע מהן במשימה 2.1. אין לשנות את האחרות.

בנאי ריק של המחלקה ממומש ואין לשנותו.

מימשנו עבורכם את השיטה `int getNumberOfOnes()` המחזירה את מספר המופעים של 1 ברשימה, ואין לשנותה.

בכל המשימות הבאות אין להוסיף שדות למחלקה.

משימה 2.1: ממשו את השיטות `void addFirst(Bit)`, `void addLast(Bit)`, `Bit removeFirst()` ו-`Bit removeLast()` במחלקה BitList.

[תת-המשימה של מימוש השיטה `addFirst` תיפתר בתגבור השבועי.]

המחלקה BitList דורסת את השיטות

```
void addFirst(Bit), void addLast(Bit), Bit removeFirst(), Bit removeLast()
```

- עליכם להשלים את הגדרת השיטות האלו כך שתיוזק חריגת זמן ריצה אם המשתמש ינסה להכניס לרשימה ערך `null`.
- כמו כן, על השיטות לעדכן את הערך של השדה `numberOfOnes` כך שייצג את המצב של העצם.

לדוגמה, ברשימה הריקה (שנסמן על ידי $\langle \rangle$) ערך השדה `numberOfOnes` יהיה 0. אחרי ביצוע הפקודה `addFirst(ONE)` הוא ישתנה ל 1, ואחרי ביצוע הפקודה `removeFirst()` יחזור להיות 0. [ראו את הדוגמא של המשימה הבאה ליתר בהירות.]

סיימתם חלק זה? כל הכבוד! שמרו את הגירסא האחרונה של עבודתכם במערכת ה-vpl.

משימה 2.2: ממשו את השיטה `String toString()` במחלקה BitList.

המחלקה BitList דורסת את השיטה `toString` של `LinkedList`, ומחזירה מחרוזת שבה הביטים מופיעים מימין לשמאל (הביט הראשון הוא הימני ביותר LSB) ומוקפים בסוגרים זוויתיים.

דוגמה:

```
BitList b1 = new BitList(); // <>
b1.addFirst(ZERO);         // <0>
b1.addFirst(ZERO);         // <00>
```

```
b1.addFirst(ONE); // <001>
System.out.println(b1); // prints <001>
```

שימו לב כי לאחר הוספת שלושת הביטים למשתנה b1 מתקבלת רשימה מקושרת שהאיבר הראשון בה הוא ONE, האיבר השני הוא ZERO והאיבר האחרון הוא ZERO.

סיימתם חלק זה? כל הכבוד! שמרו את הגירסא האחרונה של עבודתכם במערכת ה-vpl.

משימה 2.3: ממשו את הבנאי המעתיק של המחלקה BitList.

הבנאי המעתיק של המחלקה, יוצר עצם חדש השווה (לפי שיטת equals הנורשת מ-LinkedList) לפרמטר שלו. ההעתקה צריכה להיות עמוקה. כלומר, העצם המקורי והחדש שווים (לפי equals) מיד כאשר החדש נוצר. אולם אם אחר כך אחד מהם משתנה, השני אינו משתנה והם כבר לא שווים. אין להניח שהקלט תקין. יש לבדוק תקינות של הקלט ולזרוק חריגה מהטיפוס IllegalArgumentException אם הקלט אינו תקין.

דוגמה:

```
BitList b1 = new BitList(); // <>
b1.addFirst(ZERO); // <0>
b1.addFirst(ZERO); // <00>
b1.addFirst(ONE); // <001>
BitList b2 = new BitList(b1); // <001>
System.out.println(b2); // prints <001>
b2.addFirst(ONE); // <0011>
b2.addFirst(ONE); // <00111>
b2.addFirst(ONE); // <001111>
System.out.println(b1); // prints <001>
System.out.println(b2); // prints <001111>
```

סיימתם חלק זה? כל הכבוד! שמרו את הגירסא האחרונה של עבודתכם במערכת ה-vpl.

משימה 2.4: ממשו את השיטה boolean isNumber() במחלקה BitList.

בייצוג הבינארי המינימאלי שבו אנו משתמשים, לא לכל רשימת ביטים יש משמעות מספרית:

1. לרשימת ביטים ריקה אין משמעות מספרית.
2. לרשימה שהביט השמאלי ביותר שלה (כלומר האחרון ברשימה) הוא 1 וכל שאר הביטים שלה הם 0 (למשל 10000000) אין משמעות מספרית בייצוג הבינארי המינימאלי, כי הוא הנגדי של עצמו (מקבלים אותו בחזרה אם הופכים את הביטים ל 01111111 ומחברים ל 01).

פורמאלית, לרשימה יש משמעות מספרית אם:

1. אורכה לפחות 1.
2. היא מסתיימת (הביט השמאלי ביותר) בביט 0 או שיש בה יותר ממופע אחד של הביט 1.

עליכם להשלים את הגדרת השיטה `isNumber` המחזירה את הערך `true` אם ורק אם העצם המפעיל את השיטה מייצג מספר חוקי (לאו דווקא בייצוג מינימלי).

דוגמאות:

- הרשימות `<0100>` ו-`<01001>` (מייצגות את המספרים העשרוניים 4 ו-9 בהתאמה) הן מספרים כי אורכן ארבע וחמש בהתאמה (הסוגריים הזוויתיים אינם נספרים) ושניהן מסתיימות ב-0.
- הרשימות `<11000>` ו-`<10010>` (מייצגות את המספרים העשרוניים 8 ו-14 בהתאמה) הן מספרים כי יש בהן חמש ספרות ושני מופעים של הביט 1.
- הרשימות `<1000>` ו-`<1>` אינן מייצגות מספרים חוקיים כי הן אינן מסתיימות ב-0 ויש בכל אחת מהן רק מופע אחד של הביט 1.
- הסדרה $\langle \rangle$ אינה חוקית כי אורכה אפס.

דוגמה:

```
BitList b1 = new BitList();           // <>
System.out.println(b1.isNumber());    // prints false
b1.addFirst(ONE);                      // <1>
b1.addFirst(ZERO);                    // <10>
b1.addFirst(ZERO);                    // <100>
System.out.println(b1.isNumber());    // prints false
b1.addLast(ONE);                      // <1100>
System.out.println(b1.isNumber());    // prints true
b1.addLast(ONE);                      // <11100>
System.out.println(b1.isNumber());    // prints true
```

סיימתם חלק זה? כל הכבוד! שמרו את הגירסא האחרונה של עבודתכם במערכת ה-vpl.

משימה 2.5: ממשו את השיטות `boolean isReduced()` ו-`void reduce()` במחלקה `BitList`.

רשימת הביטים `<000001>` ארוכה יותר מהרשימה `<01>` אבל שתיהן ייצוגים בינאריים של המספר 1. אי אפשר לקצר עוד את `<01>` מבלי להפוך אותה לבלתי חוקית בייצוג מינימאלי (`<1>`) או לשנות את ערכה (`<0>`). לכן נקרא ל-`<01>` רשימה מינימאלית (reduced) ונאמר שהיא הייצוג הבינארי המינימאלי של המספר העשרוני 1. ל-`<000001>` נקרא רשימה לא מינימאלית, וכמובן אפשר לקצר אותה על ידי הסרת אפסים משמאל בלי לשנות את החוקיות שלה או הערך שהיא מייצגת.

באופן פורמאלי, רשימת ביטים היא מינימאלית אם:

1. היא ייצוג חוקי.
 2. מתקיים לפחות אחד משלושת התנאים הבאים:
 - א. היא אחת משלוש הסדרות: `<0>`, `<01>`, ו-`<11>` (הייצוגים הבינאריים המינימאליים של 0, 1 ו-1 בהתאמה), או
 - ב. יש בה לפחות שלושה ביטים, והשניים האחרונים (השמאליים בהדפסה) הם `01` או `10`.
- דוגמאות: `<0100>`, `<01011>`, `<101>`, `<10101>` (הייצוגים הבינאריים המינימאליים של המספרים העשרוניים 4, 11, 3 ו-11 בהתאמה), או

ג. יש בה לפחות שלושה ביטים שמהם רק שניים הם 1 והם האחרונים.
 דוגמאות: $\langle 110 \rangle$, $\langle 1100 \rangle$, $\langle 11000 \rangle$ (הייצוגים הבינאריים המינימאליים של המספרים העשרוניים 2, -4 ו-8)

לפי ההגדרה הנ"ל ניתן לראות שכל רשימת ביטים שעומדת בתנאים, מחזירה את הייצוג של מספר בבסיס 10 על ידי מספר n מינימאלי של ביטים (בדקו זאת בעצמכם).

שימו לב: רשימת ביטים חוקית לא מינימאלית ניתן לצמצם על ידי הסרת הביטים השמאליים ביותר, כל זמן שהרשימה נשארת לא מינימאלית (וחוקית). פעולה זו אינה משנה את הערך המספרי של הרשימה.

עליכם להשלים את השיטה `boolean isReduced()` כך שתחזיר את הערך `true` אם ורק אם העצם הפועל הוא רשימה מינימאלית.

עליכם להשלים את השיטה `void reduce()` כך שהעצם הפועל יהיה מינימאלי בסוף הריצה שלה. אם העצם היה מינימאלי מלכתחילה לא יחול בו כל שינוי.

דוגמאות:

- רשימת הביטים $\langle 00000 \rangle$ מייצגת את המספר אפס אך אינה ייצוג מינימאלי שלו. אפשר לצמצם אותה על ידי הסרת ארבעת הביטים השמאליים ולקבל את הייצוג הבינארי המינימאלי של אפס $\langle 0 \rangle$.
- רשימת הביטים $\langle 11101 \rangle$ מייצגת את המספר העשרוני 3- אך אינה מינימאלית. אפשר לצמצם אותה על ידי הסרת שני הביטים השמאליים ולקבל $\langle 101 \rangle$ שהיא הייצוג הבינארי המינימאלי של 3-.
- רשימת הביטים $\langle 1111100 \rangle$ מייצגת את המספר העשרוני 4- אך אינה מינימאלית. אפשר לצמצם אותה על ידי הסרת שלושת הביטים השמאליים ולקבל את הייצוג הבינארי המינימאלי של 4- שהוא $\langle 1100 \rangle$.

סיימתם חלק זה? כל הכבוד! שמרו את הגרסה האחרונה של עבודתכם במערכת ה-vpl.

משימה 2.6: ממשו את השיטה `BitList complement()` במחלקה `BitList`.

השיטה `BitList complement()` מחזירה רשימת ביטים (חדשה), בה כל ביט בעצם הפועל מוחלף במשלים שלו. המשלים של 1 הוא 0 ו- המשלים של 0 הוא 1. שימו לב, אין להתבלבל בין שיטה זו לשיטת המשלים ל-2.

דוגמה:

```
BitList b1 = new BitList(); // <>
b1.addFirst(ZERO); // <0>
b1.addFirst(ZERO); // <00>
b1.addFirst(ONE); // <001>
BitList c = b1.complement(); // <110>
System.out.println(b1) // prints <001>
System.out.println(c); // prints <110>
```

יצירת המשלים למספר היא הצעד הראשון בחישוב המספר הנגדי, והמחלקה `BinaryNumber` תשתמש בשיטה זו.

סיימתם חלק זה? כל הכבוד! שמרו את הגרסה האחרונה של עבודתכם במערכת ה-vpl.

משימה 2.7: ממשו את השיטות `Bit shiftRight()` ו-`void shiftLeft()` במחלקה `BitList`.

הזזה ימינה (`shift right`) של רשימת ביטים, היא הפעולה של הסרת הביט הראשון (הימני ביותר).

לדוגמה הזזה ימינה של הרשימה $\langle 011 \rangle$ יוצרת את הרשימה $\langle 01 \rangle$. הזזה ימינה של רשימה בת ביט אחד, למשל $\langle 1 \rangle$ יוצרת רשימה ריקה $\langle \rangle$, והזזה ימינה של רשימה ריקה אינה משנה אותה.

הזזה שמאלה (shift left) של רשימת ביטים היא הפעולה של הוספת הביט 0 בתחילת הרשימה (במקום הימני ביותר). לדוגמה, הזזה שמאלה של הרשימה $\langle 011 \rangle$ יוצרת את הרשימה $\langle 0110 \rangle$, והזזה נוספת שמאלה יוצרת את הרשימה $\langle 01100 \rangle$.

פעולות הזזה האלו הן פעולות אריתמטיות בסיסיות במדעי המחשב, ולעתים קרובות ממומשות בחומרה.

(https://en.wikipedia.org/wiki/Arithmetic_shift)

אם רשימת הביטים מייצגת מספר חיובי, להזזות ימינה ושמאלה יש משמעות של חלוקה וכפל ב-2 בהתאמה, כאשר הביט שמוסר בהזזה ימינה הוא שארית החלוקה ב-2.

המשמעות של פעולות ההזזה במקרה של רשימות המייצגות אפס או מספרים שליליים דומה, אבל קצת יותר מסובכת ולא נכנס אליה בעבודה זו.

במשימה זו עליכם להשלים את שתי השיטות Bit shiftRight() ו-void shiftLeft(). השיטה Bit shiftRight() משנה את העצם המפעיל אותה על ידי הסרת הביט הראשון שלו, ומחזירה את הערך של הביט שהוסר. אם אורכה של הרשימה אפס, היא אינה משתנה ומוחזר הערך null.

השיטה void shiftLeft() משנה את העצם המפעיל אותה על ידי הוספת הביט 0 בתחילתו, ואינה מחזירה כל ערך.

דוגמאות:

1. הרשימות $\langle 011 \rangle$, $\langle 0110 \rangle$, ו- $\langle 011000 \rangle$ שמתקבלות האחת מהשנייה על ידי הזזה שמאלה הן

הייצוגים הבינאריים המינימאליים של הערכים העשרוניים 3, 6, 12 ו-24.

2. רשימת הביטים $\langle 0111 \rangle$ מייצגת את המספר העשרוני 7 והזזתה ימינה (על ידי הסרת הביט הימני 1, שארית

החלוקה של 7 ב-2) יוצרת את הסדרה $\langle 011 \rangle$ המייצגת את המספר העשרוני 3. הזזה נוספת ימינה יוצרת את

$\langle 01 \rangle$ המייצגת את 1 (ושארית 1) והזזה נוספת את $\langle 0 \rangle$.

דוגמה עבור shiftRight:

```
BitList b1 = new BitList(); // <>
b1.addFirst(ZERO); // <0>
b1.addFirst(ZERO); // <00>
b1.addFirst(ONE); // <001>
b1.shiftRight(); // <00>
System.out.println(b1); // prints <00>
```

דוגמה עבור shiftLeft:

```
BitList b2 = new BitList(); // <>
b1.addFirst(ZERO); // <0>
b1.addFirst(ZERO); // <00>
b1.addFirst(ONE); // <001>
b1.shiftLeft(); // <0010>
System.out.println(b2); // prints <0010>
```

סיימתם חלק זה? כל הכבוד! שמרו את הגירסה האחרונה של עבודתכם במערכת ה-vpl.

משימה 2.8: ממשו את השיטה void padding(int newLength) במחלקה BitList.

[משימה זו תיפתר בתגבור השבועי.]

ריפוד של רשימת ביטים היא פעולה שבה משכפלים את הביט האחרון (השמאלי) מספר פעמים.

בזמן שנממש את הפעולות האריתמטיות במחלקה BinaryNumber יתכן שיהיה לנו נוח לעבוד עם רשימות ביטים שאינן מינימאליות. "ריפוד" הרשימה בחזרות על הביט 0 (למספרים חיוביים ואפס) או 1 (למספרים שליליים) יוצר רשימת ביטים חדשה המייצגת את אותו מספר.

לדוגמה, ניתן להגיע מהרשימה המינימאלית <101>, המייצגת את המספר 3-, לרשימה הלא מינימאלית <111101> המייצגת את אותו מספר על ידי הוספת הביט 1 שלוש פעמים.

עליכם להשלים את השיטה void padding(int newLength). שמשנה את העצם המפעיל אותה על ידי הוספת הביט האחרון של הרשימה לסופה עד שאורך הרשימה מגיע לערך של הפרמטר newLength. אם הפרמטר קטן או שווה לאורך הרשימה, השיטה אינה עושה דבר.

דוגמה:

```
BitList b1 = new BitList(); // <>
b1.addFirst(ZERO); // <0>
b1.addFirst(ZERO); // <00>
b1.addFirst(ONE); // <001>
b1.padding(10); // <0000000001>
System.out.println(b1); // prints <0000000001>
b1.padding(5); // <0000000001>
System.out.println(b1); // prints <0000000001>
```

סיימתם חלק זה? כל הכבוד! שמרו את הגירסא האחרונה של עבודתכם במערכת ה-vpl.

חלק שלישי - השלמת המחלקה BinaryNumber (VPL חלק 2)

בחלק זה של העבודה תשלימו את המחלקה BinaryNumber המייצגת מספרים בינאריים שלמים (חיוביים ושליליים) בשיטת המשלים ל-2. לשם כך היא מחזיקה שדה פרטי יחיד bits מהטיפוס BitsList.

מימשנו עבורכם את השיטות והשדות הבאים (חלקם בעזרת השיטות שרשמתם בחלקים בקודמים):

1. **בנאי פרטי** BinaryNumber(int i), המקבל את אחד המספרים: 0 או 1 ויוצר עצם המייצג אותו. כל קלט אחר לבנאי זה גורם לזריקת חריגת זמן ריצה. אין לשנות בנאי זה.
2. **בנאי מעתיק**. BinaryNumber(BinaryNumber number)
3. שני משתנים סטטיים פרטיים ZERO ו-ONE המייצגים מספרים אלו. אין לשנות את ההגדרה שלהם.
4. השיטה boolean isLegal() מחזירה את הערך true אם ורק אם העצם המפעיל אותה חוקי. עצם מהטיפוס BinaryNumber יקרא חוקי אם השדה bits הוא מספר המיוצג באופן מינימאלי. אין לשנות את השיטה. שימו לב ששיטה זו מסתמכת על השיטות isNumber() ו- isReduced() מחלק 2.
5. השיטה int length() מחזירה את מספר הביטים של המספר. גם אותה אין לשנות.

6. שתי שיטות: `BinaryNumber multiplyBy2()` ו-`BinaryNumber divideBy2()` המחזירות מספרים (חדשים) שהם תוצאת הכפלה וחילוק ב 2, של העצם הפועל בהתאמה. שימו לב שעבור פעולת החילוק מוחזר פתרון שלם בלבד. לדוגמא עבור $9/2$ יוחזר 4.

הנחיות כלליות:

- עליכם לוודא שכל המופעים של `BinaryNumber` הם חוקיים (מספר + מינימלי).
 - הבהרה: ניתן להשתמש בייצוג לא מינימלי ובלבד שבסיום השיטה הייצוג יחזור להיות מינימלי.
- אין להוסיף למחלקה שדות או משתנים סטטיים נוספים.
- למחלקה אין בנאי ריק.
- למחלקה אין שיטות ציבוריות המשנות את המצב שלה.

דיון קצר על תכנון (design)

החלטנו לפצל את המימוש של מספר בינארי לשתי מחלקות: `BitList` ו-`BinaryNumber`. לכאורה אין בכך צורך, כי ל-`BitList` אין הרבה משמעות מלבד ככלי עזר ליצירת `BinaryNumber`, ויכולנו לממש את כל הפעולות של המחלקה `BitList` בתוך המחלקה `BinaryNumber`. הסיבה לפיצול היא שאנחנו רוצים שמבחינת המשתמשים, העצמים במחלקה `BinaryNumber` (שהיא "המוצר" שלנו) תמיד יהיו "חוקיים". בפועל, תוך כדי מימוש השיטות השונות נוצרות, באופן זמני, רשימות של ביטים שאינן מייצגות מספר חוקי (למשל רשימה ריקה). המחלקה `BitList` מאפשרת לנו להשתמש (בזהירות) ברשימות כאלו בלי להסתכן בכך שהמשתמשים יחשפו אליהן.

משימה 3.1: ממשו את הבנאי `BinaryNumber(char c)` של המחלקה `BinaryNumber`.

[משימה זו תיפתר בתגבור השבועי.]

הבנאי מקבל תו המייצג ספרה עשרונית 0-9 ויוצר עצם המייצג את המספר הבינארי בייצוג מינימאלי שערכו שווה לספרה. אם מתקבל תו שאינו ספרה עשרונית הבנאי זורק חריגת זמן ריצה `IllegalArgumentException`. דוגמאות בהמשך.

סיימתם חלק זה? כל הכבוד! שמרו את הגירסא האחרונה של עבודתכם במערכת ה-vpl.

משימה 3.2: ממשו את השיטה `String toString()` במחלקה `BinaryNumber`.

השיטה דורסת את השיטה `toString()` של `Object`. היא מחזירה מחרוזת שבה סדרת הביטים מימין לשמאל. הביט במקום האפס הוא הימני ביותר והביט באינדקס הגדול ביותר הוא השמאלי (כמו `toString()` של `BitsList` אבל בלי הסוגריים הזוויתיים).

שתי השורות הראשונות של השיטה נתונות לכם. הן כוללות קריאה לשיטה `isLegal` וזריקת חריגה אם העצם אינו חוקי. אין לשנות שורות אלו.

דוגמה: הייצוג הבינארי המינימאלי של המספר העשרוני 5 הוא `<0101>`

```
BinaryNumber bn1 = new BinaryNumber('5'); // 0101 (5)
System.out.println(bn1); // prints 0101
```

סיימתם חלק זה? כל הכבוד! שמרו את הגרסא האחרונה של עבודתכם במערכת ה-vpl.

משימה 3.3: ממשו את השיטה `boolean equals(Object other)` במחלקה `BinaryNumber`.

השיטה דורסת את השיטה `equals()` של המחלקה `Object`. היא מחזירה את הערך `true` אם ורק אם הפרמטר הוא עצם במחלקה `BinaryNumber` המייצג את אותו המספר.

דוגמה:

```
BinaryNumber bn5 = new BinaryNumber('5'); // 0101 (5)
BinaryNumber bn5a = new BinaryNumber('5'); // 0101 (5)
BinaryNumber bn6 = new BinaryNumber('6'); // 0110 (6)

System.out.println(bn5.equals(bn5a)); // prints true
System.out.println(bn5.equals(bn6)); // prints false
```

סיימתם חלק זה? כל הכבוד! שמרו את הגרסא האחרונה של עבודתכם במערכת ה-vpl.

משימה 3.4: ממשו את השיטה `BinaryNumber add(BinaryNumber addMe)` במחלקה `BinaryNumber`.

השיטה מחזירה עצם המייצג את הסכום של המספר המיוצג על ידי העצם המבצע והמספר המיוצג על ידי הפרמטר, כלומר העצם המבצע ועוד הפרמטר.

הכוונה:

1. השתמשו בשיטות הסטטיות שהגדרתם במחלקה `Bit`.
 2. השיטות `padding` ו-`reduce` של `BitList` עשויות לעזור לפשט את המשימה.
 3. אל תישכחו שהערך המוחזר צריך להיות לא רק נכון אלא גם מינימאלי.
 4. השיטה צריכה לטפל במקרים של חיבור מספרים שליליים וחיבור של מספרים שליליים וחיוביים (ראו משימה 3.5).
- אין להניח שהקלט תקין, יש לבדוק את התקינות של הקלט ולזרוק חריגה מהטיפוס `IllegalArgumentException` אם הקלט אינו תקין.

דוגמה:

```
BinaryNumber bn3 = new BinaryNumber('3'); // 011 (3)
BinaryNumber bn5 = new BinaryNumber('5'); // 0101 (5)
BinaryNumber bn8 = new BinaryNumber('8'); // 01000 (8)
System.out.println(bn3.add(bn5)); // prints 01000 which is the
// minimal binary representation
// of 8.
System.out.println(bn8.add(bn3)); // prints 01011 which is the
// minimal binary representation
```

```
// of 11.
BinaryNumber bn5n = bn5.negate(); // (-5)
BinaryNumber bn3n = bn3.negate(); // (-3)
System.out.println(bn5n.add(bn3n)); // prints 11000
```

סיימתם חלק זה? כל הכבוד! שמרו את הגרסא האחרונה של עבודתכם במערכת ה-vpl.

משימה 3.5: ממשו את השיטה `BinaryNumber negate()` במחלקה `BinaryNumber`.

השיטה מחזירה עצם המייצג את המספר הנגדי (שווה בערכו המוחלט, הפוך בסימן) לעצם המבצע את הפעולה (ראו בנספח).

הכוונה:

אחרי ביצוע המשימה, חזרו למשימה 3.4 וודאו שהקוד שלכם מטפל היטב גם במקרה של חיבור מספרים שליליים וחיבור של מספרים שליליים וחיוביים. הביט השמאלי (המגדיר את הסימן של המספר) הוא נקודת תורפה.

דוגמה:

```
BinaryNumber bn5 = new BinaryNumber('5'); // 0101 (5)
BinaryNumber bnM5 = bn5.negate(); // 1011 (-5)
System.out.println(bnM5); //prints 1011
BinaryNumber bn1 = new BinaryNumber('1'); // 01 (1)
BinaryNumber bn6 = bn1.add(bn5); // 0110 (6)
System.out.println(bn5.add(bnM5)); // prints 0
System.out.println(bn6.add(bnM5)); // prints 01
```

סיימתם חלק זה? כל הכבוד! שמרו את הגרסא האחרונה של עבודתכם במערכת ה-vpl.

משימה 3.6: ממשו את השיטה `BinaryNumber subtract(BinaryNumber subtractMe)` במחלקה `BinaryNumber`.

השיטה מחזירה עצם המייצג את ההפרש שבין המספר המיוצג על ידי העצם המבצע והמספר המיוצג על ידי הפרמטר, כלומר העצם המבצע פחות הפרמטר.

דוגמה:

```
BinaryNumber bn5 = new BinaryNumber('5'); // 0101 (5)
BinaryNumber bn3 = new BinaryNumber('3'); // 011 (3)
BinaryNumber bnM2 = bn3.subtract(bn5); // 110 (-2)
BinaryNumber bn2 = bn5.subtract(bn3); // 010 (2)
System.out.println(bn2); // prints 010
System.out.println(bnM2.subtract(bn2)); // prints 1100
```

אין להניח שהקלט תקין, יש לבדוק את התקינות של הקלט ולזרוק חריגה מהטיפוס `IllegalArgumentException` אם הקלט אינו תקין.

סיימתם חלק זה? כל הכבוד! שמרו את הגרסא האחרונה של עבודתכם במערכת ה-vpl.

משימה 3.7: ממשו את השיטה `int signum()` במחלקה `BinaryNumber`.

השיטה מחזירה 1- (הערך מינוס אחד) אם העצם מייצג מספר שלילי, 0 אם הוא מייצג את המספר אפס ו-1 אם הוא מייצג מספר חיובי.

דוגמה:

```
BinaryNumber bn5 = new BinaryNumber('5'); // 0101 (5)
BinaryNumber bn2 = new BinaryNumber('2'); // 010 (2)
BinaryNumber bn3 = bn5.subtract(bitNumber2); // 011 (3)
System.out.println(bn3.signum()); // prints 1
BinaryNumber bnM2 = bn3.subtract(bitNumber5); // 110 (-2)
System.out.println(bnM2.signum()); // prints -1
```

סיימתם חלק זה? כל הכבוד! שמרו את הגירסא האחרונה של עבודתכם במערכת ה-vpl.

משימה 3.8: ממשו את השיטה `compareTo(BinaryNumber other)` במחלקה `BinaryNumber`.

המחלקה `BinaryNumber` מממשת את הממשק `Comparable<BinaryNumber>` ולכן עליה לממש את השיטה `compareTo(BinaryNumber)`. השיטה משווה בין העצם המבצע והפרמטר על ידי השוואה בין המספרים שהם מייצגים.

השיטה מחזירה 1- (הערך מינוס אחד) אם העצם הפועל קטן מהפרמטר, 0 אם הם שווים (לפי `equals`) ו-1 אם העצם הפועל גדול מהפרמטר.

דוגמה:

```
BinaryNumber bn5 = new BinaryNumber('5'); // 0101 (5)
BinaryNumber bn4 = new BinaryNumber('4'); // 0100 (4)
BinaryNumber bn4a = new BinaryNumber('4'); // 0100 (4)
```

```
System.out.println(bn5.compareTo(bn4)); // prints 1
System.out.println(bn4.compareTo(bn4a)); // prints 0
System.out.println(bn4.compareTo(bn5)); // print -1
```

אין להניח שהקלט תקין, יש לבדוק את התקינות של הקלט ולזרוק חריגה מהטיפוס `IllegalArgumentException` אם הקלט אינו תקין.

סיימתם חלק זה? כל הכבוד! שמרו את הגירסא האחרונה של עבודתכם במערכת ה-vpl.

משימה 3.9: ממשו את השיטה `toInt()` במחלקה `BinaryNumber`.

השיטה מחזירה את המספר שהעצם הפועל מייצג בצורת ערך עשרוני מהטיפוס `int`. אם המספר גדול או קטן מכדי להיות מייצג ב- `int` נזרקת חריגת זמן ריצה `RuntimeException`.

דוגמה:

```
BinaryNumber bn5 = new BinaryNumber('5'); // 0101 (5)
BinaryNumber bn4 = new BinaryNumber('4'); // 0100 (4)
```



```
System.out.println(bn5.add(bn4).toInt()); // prints 9
System.out.println(bn4.subtract(bn5).toInt()); // prints -1
```

סיימתם חלק זה? כל הכבוד! שמרו את הגרסא האחרונה של עבודתכם במערכת ה-vpl.

משימה 3.10: ממשו את השיטה

.BinaryNumber multiply(BinaryNumber multiplyMe)

השיטה מחזירה את ערך המכפלה של העצם הפועל עם הפרמטר.

את המשימה הזו תבצעו בשני שלבים:

1. כתבו שיטה פרטית **.BinaryNumber multiplyPositive(BinaryNumber multiMe)** המממשת כפל של מספרים חיוביים.
2. השתמשו בשיטה זו כדי להשלים את הגדרת השיטה הציבורית.

דוגמה:

```
BinaryNumber bn5    = new BinaryNumber('5'); // 0101 (5)
BinaryNumber bnM5    = bn5.negate();         // 1011 (-5)
BinaryNumber bn4     = new BinaryNumber('4'); // 0100 (4)
BinaryNumber bnM20   = bnM5.multiply(bn4);   // 101100 (-20)
System.out.println(bnM20.toInt());           // prints -20
```

m times

הנחיה: עקרונית, ניתן בפשטות יחסית לממש כפל כחיבור חוזר. $(n * m = n + n + \dots + n)$ אולם מימוש זה הוא מאוד לא יעיל. ניקוד מלא יינתן רק לפתרון יעיל. ניתן לחשוב על פתרון רקורסיבי יעיל, אבל גם פתרונות יעילים אחרים יקבלו את מלוא הנקודות.

סיימתם חלק זה? כל הכבוד! שמרו את הגרסא האחרונה של עבודתכם במערכת ה-vpl.

משימה 3.11: ממשו את השיטה **.BinaryNumber divide(BinaryNumber divisor)**

השיטה מחזירה את ערך מנת החלוקה של המספר המיוצג על ידי העצם הפועל במספר המיוצג על ידי הפרמטר. כלומר מפעולת החלוקה מוחזר ערך שלם גם כאשר קיימת שארית. לדוגמא $2 = 8/3$.

את המשימה הזו תבצעו בשני שלבים:

1. כתבו שיטה פרטית **.BinaryNumber dividePositive(BinaryNumber divisor)** המממשת חילוק של מספרים חיוביים שלמים.
2. השתמשו בשיטה זו כדי להשלים את השיטה הציבורית.

דוגמה:

```
BinaryNumber bn9    = new BinaryNumber('9'); // 01001 (9)
BinaryNumber bnM9    = bn9.negate();         // 10111 (-9)
BinaryNumber bn3     = new BinaryNumber('3'); // 011 (3)
```

```

BinaryNumber bn2    = new BinaryNumber('2'); //    010 (2)
BinaryNumber bnM3    = bnM9.divide(bn3);      //    101 (-3)
BinaryNumber bnM4    = bnM9.divide(bn2);      //   1100 (-4)
System.out.println(bnM3.toInt());             // prints -3

```

הנחיה: ניתן בפשטות יחסית לממש חילוק בעזרת חזרה על פעולת החיסור, אולם מימוש זה הוא לא יעיל. ניקוד מלא יינתן רק לפתרון יעיל. ניתן לחשוב על פתרון רקורסיבי יעיל, אבל גם פתרונות יעילים אחרים יקבלו את מלוא הנקודות. אין להניח שהקלט תקין. יש לבדוק את התקינות של הקלט ולזרוק חריגה מהטיפוס `IllegalArgumentException` אם הקלט אינו תקין.

סיימתם חלק זה? כל הכבוד! שמרו את הגירסא האחרונה של עבודתכם במערכת ה-vpl.

משימה 3.12: ממשו את הבנאי `BinaryNumber(String s)`.

הבנאי מקבל מחרוזת באורך כלשהו (גדול מאפס) המייצגת מספר עשרוני שלם, חיובי או שלילי. הבנאי יוצר עצם המייצג מספר זה.

דוגמה:

```

BinaryNumber bn10    = new BinaryNumber("25"); // 011001 (25)
BinaryNumber bnM10    = new BinaryNumber("-25"); // 100111 (-25)
System.out.println(bn10.toInt()); // prints 25
System.out.println(bnM10.toInt()); // prints -25

```

סיימתם חלק זה? כל הכבוד! שמרו את הגירסא האחרונה של עבודתכם במערכת ה-vpl.

משימה 3.13: ממשו את השיטה `String toString()`.

השיטה `String toString()` מחזירה מחרוזת המייצגת את העצם הפועל כמספר בבסיס 10 (חיובי או שלילי). הערה: השיטה צריכה לפעול לכל עצם בלי קשר לגודלו, גם אם העצם מייצג מספר בינארי שלא ניתן לייצג את הערך שלו באחד הטיפוסים הפרימיטיביים.

דוגמה:

```

BinaryNumber bn9      = new BinaryNumber('9'); // 01001
System.out.println(bn9.toString()); // prints 9

BinaryNumber bnM9     = bn9.negate();          // 10111 (-9)
System.out.println(bnM9.toString()); // prints -9

BinaryNumber fib100   = new BinaryNumber("354224848179261915075");
// 01001100110011110110110110110110110011111000101100101001011111111000011
System.out.println(fib100.toString());
// prints 354224848179261915075
BinaryNumber mFib100  = fib100.negate();
// 1011001100110000100100100010010101100000111010011010110100000000111101

```

```
System.out.println(mFib100.toString());  
// prints -354224848179261915075
```

סיימתם חלק זה? כל הכבוד! שמרו את הגירסא האחרונה של עבודתכם במערכת ה-vpl.

בהצלחה!