מבוא למדעי המחשב - סמסטר א' תש"פ

עבודת בית מספר 4

צוות התרגיל: חן קיסר, יערה שובל ועמיחי אלבוחר.
תאריך פרסום: 22.12.19
תאריך הגשה: 5.12.19 בשעה 12:00 בצהריים.

הקדמה

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

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

בעבודה זו 19 משימות וסך הנקודות המקסימלי הוא 100.
בעבודה זו **מותר להשתמש בידע שנלמד עד הרצאה** 16**(כולל), וכן עד תרגול** 10**(כולל).**

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.

קבצים

5. לעבודת בית זו מצורפים שלושה קבצים:
BitList.java, Bit.java, BinaryNumber.java
בקבצים אלו תערכו שינויים בהתאם למפורט בתרגיל. עליכם להגישם 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 מהמשימה שלכם (לדוגמה, בדיקות שכתבתם עבור עצמכם).

6. **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.
7. באתר הקורס תוכלו למצוא קובץ בדיקה עצמית לשימושכם.

בדיקת עבודות הבית

1. עבודות הבית נבדקות גם באופן ידני וגם באופן אוטומטי. הבדיקה האוטומטית מתייחסת אך ורק לפלט המוחזר או מודפס מהפונקציות.
2. 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.
3. 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.

עזרה והנחיה

1. 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.
2. **בתגבור** של השבוע (22.12 עד 25.12) נפתור באופן מודרך את משימות 1.1 (סעיף ב),2.1 (עבור השיטה addFirst()), 2.8 ו- 3.1.
3. בכל בעיה **אישית** הקשורה בעבודה (מילואים, אשפוז וכו'), אנא פנו אלינו דרך מערכת הפניות, כפי שמוסבר באתר הקורס.
4. אנחנו ממליצים בחום להעלות פתרון למערכת ההגשה 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 - זוהי העתקה, אשר תדווח לאלתר לוועדת משמעת. אם טרם עיינתם בסילבוס הקורס אנא עשו זאת כעת.

מומלץ לקרוא היטב את כל ההוראות המקדימות ורק לאחר מכן להתחיל בפתרון המשימות. **ודאו שאתם יודעים לפתוח קבוצת הגשה (עבור עצמכם) במערכת ההגשות.**

משימה 0: הצהרה (0 נקודות)

פתחו כל אחד משלושת קבצי ה-java:

Bit.java, BitList.java, BinaryNumber.java

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, אלא אם כן צוין אחרת.** אם הקלט אינו תקין עליכם לזרוק את החריגה `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. שימו לב לקבצים שעליכם להגיש המופיעים בסעיף 3 לעיל. יש להגיש רק את שלוש הקבצים הללו ללא חבילות (packages). גם במהלך כתיבת הקוד בסביבת הפיתוח, יש לדאוג כי כל הקבצים נמצאים תחת חבילת ברירת המחדל (default package), מה שקורה באופן אוטומטי ב- eclipse, כל עוד לא הגדרתם חבילה משלכם.
7. בכל אחד מקבצי הג'אווה שאתם מקבלים עם העבודה:
   BitList.java, Bit.java, BinaryNumber.java
   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.
8. **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.**
9. **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.**
10. **העבודה מתבססת על המחלקה `LinkedList` מהספריה 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.**
11. לנוחיותכם, באתר הקורס ניתן למצוא את הקובץ Tests.java. יש בו הרבה מאוד בדיקות לנכונות של השיטות שכתבתם. שימו לב שהבדיקות האחרונות מבצעות את פעולות החשבון על מספרים גדולים מאוד, ב סדר גודל של האיבר המאה בסדרת פיבונאצ'י.

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 (מ- 128- עד 127). מספרים גדולים יותר מהערך המקסימאלי המיוצג על ידי הטיפוס `long` (2^{63}-1) או קטנים יותר מהערך המינימאלי שהוא מיצג (-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

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 ויוצר עצם המייצג את הביט ***1*** אם הפרמטר הוא המספר אחד ו- ***0*** אם הפרמטר הוא המספר אפס.
3. שתי משתנים סטטיים ***ONE*** ו- ***ZERO*** המפנים לביטים המייצגים ***1*** ו- ***0*** בהתאמה.
4. השיטות toInt() ו- toString() המחזירות ייצוג של ביט כמספר (0 או 1) ו- מחרוזת ("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.

לדוגמה, קטע הקוד הבא

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

ידפיס

```
1  0
True
```

a	b	c_{in}	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

משימה 1.1: הפונקציות `fullAdderSum(Bit, Bit, Bit)` ו- `fullAdderCarry(Bit, Bit, Bit)`

חיבור של שלושה ביטים (קלט), שנסמן ב- a, b ו- c_{in}, 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.

ניתן לראות את הערכים בשנתי 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)

א. השלימו את הגדרת הפונקציה
```
Bit fullAdderSum(Bit A, Bit B, Bit Cin)
```
המקבלת שלושה ביטים ומחזירה את ביט הסכום של חיבורם.

ב. השלימו את הגדרת הפונקציה `Bit fullAdderCarry(Bit A, Bit B, Bit Cin)` המקבלת שלושה ביטים ומחזירה את ביט הנשא של חיבורם.

הנחיות:

1. ניתן להניח שהקלט תקין, כלומר שאף אחד מן הפרמטרים אינו `null`.
2. הקפידו על קוד פשוט ונקי. סבך של פקודות `if` ו `else` לא יקבל ניקוד מלא.

לדוגמה:

```
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
}
```

שימו לב שניתן לראות את השורות המודפסות כמספרים בינאריים בני שתי ספרות שהן הסכום של ערכי שלושת הביטים של הקלט.

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

חלק שני - השלמת המחלקה BitList

רקע

בעבודה זו אנחנו מייצגים מספרים באמצעות רשימות משורשרות של עצמים מטיפוס Bit. בחלק זה של העבודה נשלים את הגדרת המחלקה BitList המרחיבה את המחלקה LinkedList<Bit> מהספריה הסטנדרטית של ג'אווה. המחלקה BitList מספקת את השיטות הבסיסיות, שבהם משתמשת המחלקה BinaryNumber שנשלים 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:

בשעור 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 שלגרסה זו של LinkedList נמצאת ב- API של ג'אווה https://docs.oracle.com/javase/8/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. השיטות מחזירות את האיבר שהוסר. **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, וזמן הריצה תלוי באינדקס i.

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

שיטת המשלים ל-2 (two's complement) לייצוג מספרים חיוביים ושליליים:

הייצוג הרגיל (זה שלמדנו בבית הספר היסודי) של מספרים שליליים הוא באמצעות תו מיוחד "-" (מינוס) שהוא חלק מהמספר למרות שאינו ספרה. ייצוג בינארי של מספרים נעשה בעזרת שני ביטים ***1*** ו- ***0*** בלי תוספות "מיוחדות" כגון סימן המינוס. אחת הגישות הנפוצות לייצוג מספרים בינאריים שלמים במחשב נקראת המשלים ל-2 ובאנגלית two's complement (https://en.wikipedia.org/wiki/Two's_complement). בשיטה זו הביט השמאלי ביותר במספר מעיד על הסימן של הביט הזה. הביט ***0*** מעיד על מספר חיובי או אפס והביט ***1*** על מספר שלילי. כך למשל המספר העשרוני החיובי 7 מיוצג בשמונה ביטים על ידי ***00000111*** והביט השמאלי ביותר בייצוג הזה הוא ***0***. המספר הנגדי (אותו ערך מוחלט וסימן הפוך) 7- מיוצג בשמונה ביטים על ידי ***11111001*** שהביט השמאלי ביותר שלו היא ***1***. המספר אפס מיוצג בשמונה ביטים על ידי ***0000000***. לביט הימני ביותר של מספר בינארי נקרא "הפחות משמעותי" (least significant bit – LSB) ולשמאלי נקרא "המשמעותי ביותר" (most significant bit – MSB).

את ייצוגים הבינאריים החיוביים ואפס אפשר לפרש כסכום חזקות של 2. הביט הימני ביותר מוכפל ב- 2^0, השני מימין ב- 2^1 וכן הלאה. לדוגמה, המספר העשרוני 5 מיוצג בשמונה ביטים על ידי המספר הבינארי ***00000101*** שאותו ניתן לפרש

כסכום $1*2^0 + 0*2^1 + 1*2^2 + 0*2^3 + 0*2^4 + 0*2^5 + 0*2^6 + 0*2^7 = 5$. המספר אפס בייצוג זה הוא סכום של שמונה אפסים.

הייצוג של מספר שלילי מתקבל מהייצוג של החיובי הנגדי בשני שלבים: בשלב ראשון מחליפים כל ***0*** ב- ***1*** וכל ***1*** ב- ***0*** המספר שמתקבל נקרא המשלים (complement). בשלב השני מוסיפים אליו מוסיפים (בפעולת חיבור) את הייצוג הבינארי של המספר 1, בלי לשנות את מספר הביטים. לדוגמה, הייצוג הבינארי של המספר העשרוני 7 בארבעה ביטים הוא ***0111***. המשלים שלו הוא ***1000***. כשנחבר לו את 1 שמיוצג על ידי ***0001*** נקבל את הייצוג הבינארי של -7 בארבעה ביטים שהוא ***1001***. אותו התהליך פועל גם בכוון ההפוך, המשלים של המספר השלילי ***1001*** הוא ***0110*** והוספת ***0001*** נותנת ***0111*** הייצוג הבינארי של המספר 7. מה לגבי אפס המיוצג בארבעה ביטים על ידי ***0000*** ? המשלים שלו הוא ***1111*** והוספת ***0001*** נותנת ***10000*** שבו חמישה ביטים. מכיוון שאנחנו איננו משנים את מספר הביטים נוותר על הביט השמאלי ביותר (פעולה המכונה גלישה - overflow) ונקבל שוב את המספר ***0000***.

לכל אחד מהמספרים הניתנים לייצוג על ידי n ביטים יש ייצוג גם ב- m>n ביטים. הייצוגים האלו קשורים בפעולה הנקראת ריפוד (padding). נראה לדוגמה את המספר העשרוני 2. הייצוג שלו בשלושה ביטים הוא ***010***. הייצוג שלו בארבעה ביטים הוא ***0010*** בחמישה ביטים ***00010*** ובשישה ביטים ***000010*** כלומר כדי לעבור מייצוג של 2 (וכל מספר לא שלילי אחר) מייצוג ב- n ביטים לייצוג ב- m > n ביטים היינו צריכים לרפד אותו מצד שמאל ב m – n אפסים (המסומנים בקו תחתי).

באותה צורה הייצוג הבינארי של המספר העשרוני -2 בשלושה ביטים הוא ***110***, הייצוג שלו בארבעה ביטים הוא ***1110***, בחמשה ביטים ***11110*** וכן הלאה. כלומר כדי לעבור מייצוג של -2 (וכל מספר שלילי אחר) מייצוג ב n ביטים לייצוג ב m > n ביטים יש לרפד אותו משמאל בm – n אחדות .

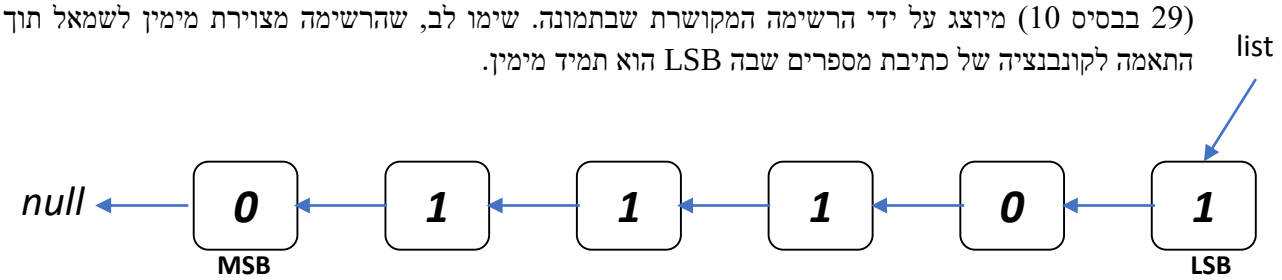
המספר	ייצוג ב 4 ביטים	ייצוג בינארי מינימאלי
7	***0111***	***0111***
6	***0110***	***0110***
5	***0101***	***0101***
4	***0100***	***0100***
3	***0011***	***011***
2	***0010***	***010***
1	***0001***	***01***
0	***0000***	***0***
-1	***1111***	***11***
-2	***1110***	***110***
-3	***1101***	***101***
-4	***1100***	***1100***
-5	***1011***	***1011***
-6	***1010***	***1010***
-7	***1001***	***1001***
-8	***1000***	***11000***

לכל מספר עשרוני, חיובי או שלילי, יש ייצוג בינארי מינימאלי כלומר כזה שבו אין ביטים המשמשים כריפוד. עבור המספר העשרוני 2 למשל הייצוג המינימאלי הוא בן 3 ביטים ***010***, כאשר הביט השמאלי ביותר מסמן שהמספר החיובי. מסיבה דומה גם הייצוג המינימאלי של -2 הוא בן שלושה ביטים. הסרת הביט השמאלי הייתה נותנת לנו ***10***, שנראה כמו מספר שלילי, אבל לפי כללי המשלים ל-2 המספר המיוצג על ידי ***10*** הוא הנגדי של עצמו. לא סביר למספר שאינו אפס.

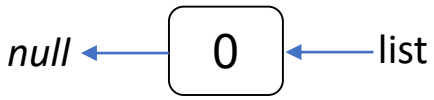
באופן כללי המספר המינימאלי של ביטים שנדרש כדי לייצג את המספר אפס הוא אחד , ולכל $i \neq 0$ המספר המינימאלי של ביטים הנדרשים כדי לייצג את i. הוא השלם הקרוב ביותר מלמטה ל- $\log_2(abs(i))+2$. הטבלה בעמוד זה מציגה בעמודה השמאלית את כל המספרים העשרוניים הניתנים לייצוג בינארי בארבעה ביטים בדיוק, בעמודה האמצעית את הייצוג שלהם בארבעה ביטים בדיוק ובעמודה הימנית שלהם את הייצוג הבינארי מינימאלי. שימו לב שבמקרה אחד, -8, הייצוג הבינארי המינימאלי הוא בן חמשה ביטים.

שימוש ברשימה משורשרת 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 לקונבנציה 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 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.

המחלקה BitList מרחיבה את המחלקה LinkedList, מהספריה הסטנדרטית של ג'אווה. יש לה שדה אחד numberOfOnes המייצג את מספר הביטים שערכם ***1***, ושני שדות סטטיים ***ONE*** ו- ***ZERO*** המייצגים את הערכים ***1*** ו- ***0*** בהתאמה.

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 ומוציאות 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.

השיטה ()getNumberOfOnes int המחזירה את מספר המופעים של ***1*** ברשימה ממומשת ואין לשנותה.

<u>אין להוסיף שדות למחלקה</u>

<u>משימה 2.1:</u> השלימו את הגדרת השיטות void addFirst(Bit) , void addLast(Bit), Bit removeFirst() ו- Bit removeLast() במחלקה BitList.

המחלקה BitList דורסת את השיטות
void addFirst(Bit), void addLast(Bit), Bit removeFirst(), Bit removeLast()

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

לדוגמה, ברשימה הריקה (שנסמן על ידי <>) ערך השדה numberOfOnes יהיה 0. אחרי ביצוע הפקודה addFirst(***ONE***) הוא ישתנה ל 1, ואחרי ביצוע הפקודה removeFirst() יחזור להיות 0.

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

<u>משימה 2.2:</u> השלימו את הגדרת 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) ומוקפים בסוגרים זוויתיים.

לדוגמה:

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

בדוגמה זו ובבאות אחריה ההערה מציגה את המחרוזת המוחזרת על ידי השיטה ()toString של העצם שנוצר באותה שורה, או שפועל בה.

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

<u>משימה 2.3:</u> השלימו את הגדרת 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

לדוגמה:

```
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>
```

אין להניח שהקלט תקין. יש לבדוק תקינות של הקלט ולזרוק חריגת `IllegalArgumentException` אם הקלט אינו תקין.

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

. **<u>משימה 2.4:</u> השלימו את הגדרת 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***.

דוגמאות:

הרשימות <***0100***> ו- <***01001***> (מייצגות את המספרים העשרוניים 4 ו- 9 בהתאמה) הן מספרים כי אורכן ארבע וחמש בהתאמה (הסוגריים הזוויתיים אינם נספרים) ושתיהן 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 מספרים כי הן אינן מסתיימות ב ***0*** ויש בכל אחת מהן רק מופע אחד של הביט ***1***.

הסדרה <> אינה חוקית כי אורכה אפס

עליכם להשלים את הגדרת השיטה `isNumber` המחזירה true אם ורק אם העצם המפעיל את השיטה חוקי.

לדוגמה, קטע הקוד הבא:

```
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
```

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

<u>משימה 2.5:</u> השלמו את השיטות `boolean isReduced()` ו- `void reduce()` במחלקה `BitList`.

רשימת הביטים <***000001***> ארוכה יותר מהרשימה <***01***> אבל שתיהן ייצוגים בינאריים של המספר1. אי אפשר לקצר עוד את <***01***> מבלי להפוך אותה לבלתי חוקית <***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 שאפשר לקצר אותה על ידי הסרת אפסים משמאל בלי לשנות את החוקיות שלה או הערך שהיא מייצגת. באופן כללי, נגדיר כמינימאלית רשימת ביטים חוקית שאם מסירים את הביט האחרון (השמאלי ביותר) שלה היא הופכת לבלתי חוקית או משנה את ערכה המספרי (בשיטת המשלים לשתיים).

באופן פורמאלי, רשימת ביטים היא מינימאלית אם:

1. היא חוקית.
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: <110> , <1100>, <11000> (הייצוגים הבינאריים המינימאליים של המספרים העשרוניים 2-, 4- ו- 8-)

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.

דוגמאות:

1. רשימת הביטים <***00000***> 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 <***0***>.
2. רשימת הביטים <***11101***> מייצגת את המספר העשרוני 3- אך אינה מינימאלית. אפשר לצמצם אותה על ידי הסרת שני הביטים השמאליים ולקבל <***101***> שהיא הייצוג הבינארי המינימאלי של 3-.
3. רשימת הביטים <1111100> 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 <1100> .

עליכם להשלים את השיטה `boolean isReduced()` כך שתחזיר `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.

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

<u>משימה 2.6:</u> השלימו את השיטה `BitList complement()` במחלקה `BitList`.

השיטה `BitList complement()` מחזירה רשימת ביטים באורך של העצם הפועל ושבה כל ביט בעצם הפועל מוחלף במשלים שלו. המשלים של ***1*** הוא ***0*** ו- המשלים של ***0*** הוא ***1***.

לדוגמה:

```
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! העלו את הגרסה האחרונה של עבודתכם למערכת ההגשה.

<u>משימה 2.7:</u> השלימו 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 <***011***> יוצרת את הרשימה <***01***>. הזזה ימינה של רשימה בת ביט אחד, למשל <***1***> יוצרת רשימה ריקה <> 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 <***011***> יוצרת את הרשימה <***0110***>, והזזה נוספת שמאלה יוצרת את הרשימה <***01100***>.

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.
דוגמאות:

1. הרשימות <***011***> , <***0110***>, <***01100***> ו- <***011000***> שמתקבלות האחת מהשנייה על ידי הזזות ימינה ושמאלה הן הייצוגים הבינאריים המינימאליים של הערכים העשרוניים 3, 6, 12 ו- 24.
2. רשימת הביטים <***0111***> מייצגת את המספר העשרוני 7 והזזתה ימינה (על ידי הסרת הביט הימני ***1***, שארית החלוקה של 7 ב 2) יוצרת את הסדרה <***011***> המייצגת את המספר העשרוני 3. הזזה נוספת ימינה יוצרת את <***01***> המייצגת את 1 (ושארית 1) והזזה נוספת את <***0***>.

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`.

לדוגמה:

```
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>
```

השיטה `void shiftLeft()` אינה מחזירה ערך אלא משנה את העצם המפעיל אותה על ידי הוספת הביט ***0*** בתחילתו.

לדוגמה:

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

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

<u>משימה 2.8:</u> השלימו את השיטה void padding(int newLength) במחלקה BitList.

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 רשימת ביטים שאינה מינימאלית. "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.

לדוגמה:

```
BitList b1 = new BitList();    //          <>
b1.addFirst(ZERO);             //         <0>
b1.addFirst(ZERO);             //        <00>
b1.addFirst(ZERO);             //       <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! העלו את הגרסה האחרונה של עבודתכם למערכת ההגשה.

חלק שלישי - השלמת המחלקה BinaryNumber

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. עליכם לוודא שבכל העצמים המוגשמים במחלקה (על ידי בנאים או שיטות ציבוריות) רשימת הביטים תהיה מינימאלית.

נתון לכם 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.

כמו כן נתון לכם בנאי מעתיק.

למחלקה שני משתנים סטטיים פרטיים ZERO ו- ONE המייצגים מספרים אלו. אין לשנות את ההגדרה שלהם.

אין להוסיף למחלקה שדות או משתנים סטטיים נוספים.

עצם מהטיפוס BinaryNumber יקרא חוקי אם השדה bits הוא מספר ומינימאלי. השיטה boolean isLegal() מחזירה את הערך true אם ורק אם העצם המפעיל אותה חוקי. שיטה זו נתונה לכם. אין לשנותה.

כמו כן נתונה לכם השיטה int length() המחזירה את אורכו של המספר בביטים. גם אותה אין לשנות.

למחלקה אין בנאי ריק.

למחלקה אין שיטות ציבוריות המשנות את המצב שלה.

נתונות לכם שתי שיטות: BinaryNumber multiplyBy2() ו- BinaryNumber divideBy2() המחזירות הפניה למספרים שהם תוצאת הכפלה וחילוק, בהתאמה, של העצם הפועל ב 2.

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 BitList בתוך BinaryNumber. הסיבה לפיצול היא שאנחנו רוצים שמבחינת המשתמשים, העצמים במחלקה BinaryNumber (שהיא "המוצר" שלנו) תמיד יהיו "חוקיים" (נסביר מונח זה בהמשך). 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: השלימו את הגדרת הבנאי BinaryNumber(char c) של המחלקה BinaryNumber

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

דוגמאות בהמשך.

<u>משימה 3.2:</u> השלימו את הגדרת השיטה `String toString()` במחלקה `BnaryNumber`.

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
```

בדוגמה זו ובבאות אחריה ההערה היא המחרוזת שמחזירה השיטה `toString()` של העצם שאליו מפנה המשתנה ובסוגריים המספר העשרוני שהוא מייצג.

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

<u>משימה</u>
<u>3.3:</u> השלימו את הגדרת השיטה `boolean equals(Object other)` במחלקה `BnaryNumber`.

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

דוגמה:

```
BnaryNumber bn5 = new BinaryNumber('5');  // 0101 (5)
BnaryNumber bn5a = new BinaryNumber('5'); // 0101 (5)
BnaryNumber 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! העלו את הגרסה האחרונה של עבודתכם למערכת ההגשה.

משימה 3.4: השלימו את הגדרת השיטה BinaryNumber add(BinaryNumber addMe) **במחלקה** BinaryNumber.

השיטה מחזירה עצם המייצג את הסכום של המספר המיוצג על ידי העצם המבצע את הפעולה עם המספר המיוצג על ידי הפרמטר.

דוגמה:

```
BnaryNumber bn3  = new BinaryNumber('3'); //   011 (3)
BnaryNumber bn5  = new BinaryNumber('5'); //  0101 (5)
BnaryNumber 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.
```

הכוונה:

א. השתמשו בשיטות הסטטיות שהגדרתם במחלקה Bit

ב. השיטות padding ו- reduce של BitList עשויות לעזור לפשט את המשימה.

ג. אל תישכחו שהערך המוחזר צריך להיות לא רק נכון אלא גם מינימאלי.

ד. אחרי ביצוע המשימה הבאה (negate) ודאו 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.

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

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

משימה 3.5: השלימו את השיטה BinaryNumber negate() במחלקה BinaryNumber.

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

דוגמה:

```
BinaryNumber bn5 = new BinaryNumber('5'); // 0101 (5)
BinaryNumber bnM5 = bn5.negate();         // 1011 (-5)
System.out.println(bn2); //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! העלו את הגרסה האחרונה של עבודתכם למערכת ההגשה.

משימה 3.6: השלימו 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'); //              (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 חריגת `IllegalArgumentException` אם הקלט אינו תקין.

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

משימה 3.7: השלימו את הגדרת 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 bnM3 = bn3.subtract(bitNumber5); //  101    (-3)
System.out.println(bn3.signum());     //prints -1
```

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

משימה 3.8: השלימו את הגדרת השיטה `int compareTo(BinaryNumber other)` במחלקה BinaryNumber.

המחלקה `BinaryNumber` מממשת את הממשק `Comparable<BinaryNumber>` ולכן עליה לממש את השיטה `int 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 bn3 = new BinaryNumber('4'); // 0100 (4)

System.out.println(bn5.compareTo(bn4)); // prints 1
System.out.println(bn3.compareTo(bn3)); // prints 0
System.out.println(bn4.compareTo(bn5)); // print -1
```

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

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

משימה 3.9: השלימו את הגדרת השיטה `int toInt()` במחלקה BitsNumber.

השיטה מחזירה את המספר שהעצם הפועל מייצג בצורת ערך עשרוני מהטיפוס `int`. אם המספר גדול או קטן מכדי להיות מיוצג 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! העלו את הגרסה האחרונה של עבודתכם למערכת ההגשה.

<u>משימה 3.10:</u> השלימו את הגדרת 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)`** הממשת 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
```

הנחיה: ניתן בפשטות יחסית לממש כפל כחיבור חוזר. $(n*m = \overbrace{n + n + \ldots + n}^{\text{m times}})$ אולם מימוש זה הוא מאוד לא יעיל. ניקוד מלא יינתן רק לפתרון יעיל. קל יותר לחשוב על פתרון רקורסיבי יעיל, אבל גם פתרון יעיל אחר יקבל את מלוא הנקודות.

משימה 3.11: השלימו את הגדרת השיטה `BinaryNumber divide(BinaryNumber divisor)`

השיטה מחזירה את ערך מנת החלוקה של המספר המיוצג על ידי העצם הפועל במספר המיוצג על ידי הפרמטר.

את המשימה הזו תבצעו בשני שלבים:

1. כתבו שיטה פרטית **`BinaryNumber dividePositive(BinaryNumber divisor)`** הממשת 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. קל יותר לחשוב על פתרון רקורסיבי יעיל, אבל גם פתרון יעיל אחר יקבל את מלוא הנקודות.
אין להניח שהקלט תקין, יש לבדוק את התקינות של הקלט ולזרוק חריגת `IllegalArgumentException` אם הקלט אינו תקין.

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

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

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

אם הקלט אינו תקין: `null`, מחרוזת ריקה, או מחרוזת שלא מייצגת מספר עשרוני, השיטה תזרוק חריגת `IllegalArgumentException`

דוגמה:

```
BinaryNumber bn10  = new BinaryNumber("10");   // 01010 (10)
BinaryNumber bnM10 = new BinaryNumber("-10");  // 10110 (-10)
System.out.println(bn10.toInt());  // prints 10
System.out.println(bnM10.toInt()); // prints -10
```

<u>משימה 3.13:</u> השלימו את הגדרת השיטה `String toIntString()`.

השיטה `String toIntString()` מחזירה מחרוזת המייצגת את העצם הפועל כמספר עשרוני (חיובי או שלילי) בלי קשר לגודלו (כלומר גם אם לא ניתן לייצג את הערך שלו באחד הטיפוסים הפרימיטיביים).

דוגמה:

```
BinaryNumber fib100 = new BinaryNumber("354224848179261915075");
// 01001100110011110110110111011010100111110001011001010010111111111000011
System.out.println(fib100); // prints 354224848179261915075
BinaryNumber mFib100 = fib100.negate();
// 10110011001100001001001000100101011000001110100110101101000000000111101
System.out.println(mFib100); // prints -354224848179261915075
```

בהצלחה!