

# מבוא למדעי המחשב - סמסטר א' תשפ"ד


## עבודת בית מספר 4

### צוות העבודה:

- מרצה אחראית: חן קיסר.
  - מתרגלים אחראים: שחר אזולאי, רז לפיד.
- מועד פרסום:** 25.02.24 בשעה 14:00.
- מועד אחרון להגשה:** 10.03.24 בשעה 23:59.

### הוראות מקדימות:

#### הגשת עבודות בית

1. קראו את העבודה מתחילתה ועד סופה לפני שאתם מתחילים לפתור אותה. ודאו שאתם מבינים את כל המשימות. רמת הקושי של המשימות אינה אחידה: הפתרון של חלק מהמשימות קל יותר, ואחרות מצריכות חקירה מתמטית - שאותה תוכלו לבצע בעזרת מקורות דרך רשת האינטרנט. בתשובות שבהן אתם מסתמכים על עובדות מתמטיות שלא הוצגו בשיעורים, יש להוסיף כהערה במקום המתאים בקוד את ציטוט העובדה המתמטית ואת המקור (כגון ספר או אתר).
2. עבודה זו תוגש ביחידים במערכת המודל. ניתן לצפות ב**סרטון הדרכה על הגשת העבודה במערכת ה-vpl** תחת "קישורים שימושיים" באתר הקורס.
3. במערכת מופיעים קבצי Java שונים וקובץ ה-IntegrityStatement.java (הצהרה על יושר אקדמי). אלו הם קבצי השלד אותם עליכם לערוך ולהגיש. עליכם לערוך את הקבצים האלו בהתאם למפורט בתרגיל ולהגישם כפתרון. **אין לשנות את שמות קבצי השלד.**
4. **המלצה על דרך העבודה** - אנו ממליצים לפתוח פרויקט ב-eclipse בשם Assignment4. כשתעבדו, תערכו (לאחר שהורדתם את קבצי השלד וחילצתם אותם לתוך הפרויקט) בתוך הפרויקט את קבצי ה-Java בהתאם להוראות המשימה והגישו אותם לפי ההנחיות.
5. עבודות שלא יעברו **קומפילציה במערכת** או שבהן **לא נחתמה הצהרה** על יושר אקדמי (משימה 0) יקבלו את **הציון 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  Evaluate-ה.
6. עבודות הבית נבדקות גם באופן ידני וגם באופן אוטומטי. לכן, יש להקפיד על ההוראות ולבצע אותן במדויק.
7. סגנון כתיבת הקוד ייבדק באופן ידני. יש להקפיד על כתיבת קוד יעיל, ברור, על מתן שמות משמעותיים למשתנים, על הזחות (אינדנטציה), ועל הוספת הערות בקוד המסבירות את תפקידם של מקטעי הקוד השונים. אין צורך למלא את הקוד בהערות מיותרות, אך חשוב לכתוב הערות בנקודות קריטיות, המסבירות קטעים חשובים בקוד. **לרשותכם מדריך לכתיבת הערות בקוד.** הערות יש לרשום אך ורק באנגלית. כתיבת קוד אשר אינה עומדת בדרישות אלו תגרור הפחתה בציון העבודה.

#### 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 Java:
  - 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 התייעוד בסטנדרט זה (עשוי לסייע לכם בהבנת התייעוד).
6. בכל אחד מקבצי ה - Java 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, אין לשנות בנאים ריקים, את כותרות המחלקות, את 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 Java. קריאה חוזרת ונשנית לשיטה get המוגדרת במחלקה זו היא מאוד לא יעילה במחלקה זו. פתרונות יעילים משתמשים באיטרטור ככל שזה ניתן.**
10. כרגיל, הוטמעו במערכת ה - VPL בדיקות מדגמיות לנכונות של השיטות שכתבתם. שימו לב שבדיקות אלה **מדגמיות ביותר** (כלומר, בודקות רק חלק מהשיטות על חלק קטן מאוד מהקלטים). לא ניתן להסיק ממעבר מוצלח של הבדיקות המדגמיות שהקוד שהוגש נכון במלואו. מטרתן העיקרית הינה לאפשר לכם (באמצעות לחיצה על כפתור Evaluate, כרגיל), לוודא שהקוד שהגשתם עובר קומפילציה במערכת ושחתמתם על הצהרת היושר באופן תקין. שימו לב שחלק מהבדיקות מבצעות את פעולות החשבון על מספרים גדולים מאוד, כגון מספרים בסדר גודל של האיבר המאה בסדרת פיבונאצ'.
11. נדגיש למען הסר ספק: אין להשתמש בשום פנים ובאופן במחלקה BigInteger של Java! שימוש במחלקה זו יגרום לפסילת החלק בו נעשה שימוש בטיפוס זה.
12. בפירוט המשימות שזורים מספר קטעי "העשרה". הם נועדו לתת מוטיבציה לשימושי המשימות שתכנתו בעבודה זו, ולהרחיב את הידע התיאורטי לגביהם, למי שמעוניין בכך. רובכם תיתקלו במונחים המופיעים בהם בהמשך לימודיכם. סטודנטים החשים שקטעים אלה מעמיסים עליהם יותר מדי מידע בשלב זה, יכולים לדלג עליהם בבטחה. הם אינם הכרחיים לטובת ביצוע העבודה, וניתן להשלימה בצורה מלאה גם ללא העמקה בהם כלל. עם זאת, אנו בטוחים שסטודנטים סקרנים ימצאו בהם עניין רב, ומעודדים אתכם להתעניין ולהפליג בקריאה ככל שתמצאו.

### 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.
2. בתגבור נפתור באופן מודרך את משימה **1.1 חלק ב'**. כמו כן, אתם יכולים להיעזר בפורום ולפנות בשאלות לחבריכם לכיתה. צוות הקורס עובר על השאלות ונותן מענה במקרה הצורך. שימו לב, **אין**

### **לפרסם פתרונות בפורום.**

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 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 - זוהי העתקה, אשר תדווח לאלתר לוועדת משמעת. אם טרם עיינתם [בסילבוס הקורס](#) אנא עשו זאת כעת.

**חובה לחתום על הצהרת יושר אקדמי בהתאם להנחיות במשימה 0 !**

## משימות:

יש להגיש את כל השאלות עד התאריך 25.02.24 תחת **עבודת בית 4 - VPL**. עקבו אחרי הוראות ההגשה בסוף העבודה.

### משימה 0 - הצהרה

פתחו את הקובץ `IntegrityStatement.java` וכתבו בו את שמכם ומספר תעודת הזהות שלכם במקום המסומן. משמעות פעולה זו היא שאתם מסכימים וחותמים על הכתוב בהצהרה הבאה:

I, <Israel Israeli> (<123456789>), certify that the work I have submitted is entirely my own. I have not received any part of it from any other person, nor have I given any part of it to others for their use.

I have not copied any part of my answers from any other source, and what I submit is my own creation.

I understand that formal proceedings will be taken against me before the BGU Disciplinary Committee if there is any suspicion that my work contains code/answers that is not my own.

If you have relied on or used an external source, you must cite that source at the end of your integrity statement. External sources include shared file drivers, Large Language Models (LLMs) including ChatGPT, forums, websites, books, etc.

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

**שימו לב! עבודות בהן לא תמולא ההצהרה, יקבלו ציון 0.**

### מוטיבציה

הבסיס לעבודה זו (ולכל האריתמטיקה במחשבים) הן שתי הספרות הבינאריות, המכונות ביטים,  $0$  ו- $1$ . בשפה `Java`, הטיפוסים הפרימיטיביים המייצגים מספרים שלמים (`long`, `int`, `short`, `byte`) משתמשים במספר קבוע של ביטים ( $8$ ,  $16$ ,  $32$  – בהתאמה) ולכן יש מגבלות מובנות על הגודל המקסימאלי של המספרים שהם יכולים לייצג. כך למשל הטיפול `byte` מיוצג על ידי  $8$  ביטים ולכן יכול לייצג  $2^8$  מספרים שונים ( $128$  ועד  $127$ ). מספרים גדולים יותר מהערך המקסימאלי המיוצג על ידי הטיפוס `long` ( $1 - 2^{63}$ ) או קטנים יותר מהערך המינימאלי שהוא מייצג ( $-2^{63}$ ) אפשר לייצג ב-`Java` רק על ידי טיפוסים מורכבים. כבר בעבודת הבית הראשונה של קורס זה הבנו את הצורך במספרים כאלו ובעבודה 3 הכרנו את הפתרון הסטנדרטי של `Java`, 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 **אי שליליים בלבד**. המספרים המיוצגים על ידי עצמים מהמחלקה `BinaryNumber` עשויים להיות גדולים 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. עליכם 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 המפנה לביטים המייצגים 1 ו-0 בהתאמה.
4. השיטות toInt() ו- toString() המחזירות ייצוג של ביט כ- int (0 או 1) ומחרוזת ("0" או "1") 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).

### משימה 1.1 – השיטות fullAdderSum(Bit, Bit, Bit) ו- fullAdderCarry(Bit, Bit, Bit) (6 נקודות)

| A | B | c <sub>in</sub> | 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), שנשמך 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 משמאל.

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

רכיב אלקטרוני שממש חיבור של שלוש ספרות נקרא full-adder. רכיבים כאלו הם מרכיבים עיקריים במערכות דיגיטליות 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)
```

הפונקציה מקבלת כקלט שלושה ביטים ומחזירה את ביט הסכום של חיבורם.

ב. ממשו את השיטה

```
Bit fullAdderCarry(Bit A, Bit B, Bit Cin)
```

הפונקציה מקבלת כקלט שלושה ביטים ומחזירה את ביט הנשא של חיבורם. [סעיף זה ייפתר בתגבור השבועי.]

#### הנחיות:

- הקפידו על קוד פשוט ונקי. מימוש הטבלה באופן ישיר יקבל ניקוד חלקי, חישבו כיצד ניתן לממש את השיטה באמצעות חישוב אריתמטי. הנחייה: שימו לב לתוצאת החיבור של הביטים ולפעולות אריתמטיות שניתן לעשות עליה על מנת לקבל את carry ואת sum.
- כל עצם תופס מקום בזיכרון. מימושים של השיטות שיוצרים עצם חדש יקבלו ניקוד חלקי.

דוגמאות:

אם הקלט הוא `Bit b1 = Bit.ONE, Bit b0 = Bit.ZERO` אזי:

- הקריאה לפונקציה `fullAdderCarry(b0, b0, b0)` תחזיר את הערך 0 והקריאה `fullAdderSum(b0, b0, b0)` תחזיר את הערך 0.
- הקריאה לפונקציה `fullAdderCarry(b1, b0, b0)` תחזיר את הערך 0 והקריאה `fullAdderSum(b1, b0, b0)` תחזיר את הערך 1.
- הקריאה לפונקציה `fullAdderCarry(b1, b1, b0)` תחזיר את הערך 1 והקריאה `fullAdderSum(b1, b1, b0)` תחזיר את הערך 0.
- הקריאה לפונקציה `fullAdderCarry(b1, b1, b1)` תחזיר את הערך 1 והקריאה `fullAdderSum(b1, b1, b1)` תחזיר את הערך 1.

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

### חלק שני – השלמת המחלקה BitList

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 בספרייה הסטנדרטית של `Java`. המחלקה `BitList` מספקת את השיטות הבסיסיות בהן משתמשת המחלקה `BinaryNumber`, אותה נשלים בחלק השלישי של העבודה. לפני שנתחיל בעבודה עלינו להתוודע לגרסה החדשה, מבחינתנו, של המחלקה `LinkedList`, ולהכיר מספר מושגים.

#### **המחלקה `LinkedList<T>` בספרייה הסטנדרטית של `Java`:**

בשיעור כתבנו מחלקה בשם `LinkedList<T>` המממשת את הממשק `List<T>`. גם הספרייה הסטנדרטית של `Java` מציעה מחלקה כזו, שהיא מורכבת יותר ונותנת הרבה יותר שיטות. הרשימה המלאה של השיטות והבנאים של גרסה זו של `LinkedList` נמצאת ב-API של `Java`:

<https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html>

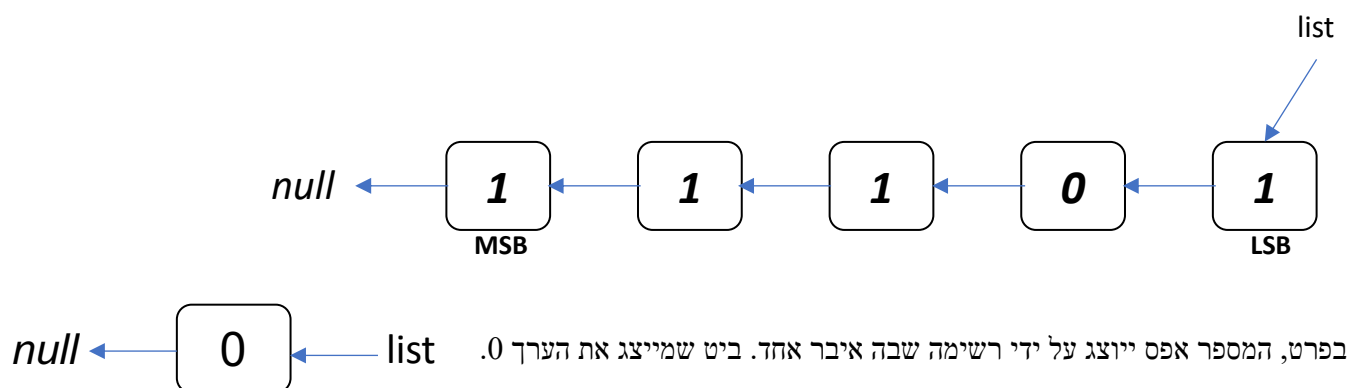
להלן מספר שיטות שהן שימושיות במיוחד לעבודה זו:

- `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)` – שיטות אלו מוסיפות איברים לרשימה במקום הראשון והאחרון 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()` – שיטות אלו מסירות את האיבר במקום הראשון והאחרון בהתאמה ומחזירות את האיבר שהוסר מטיפוס `T`. **שימו לב**, המימוש של שיטות אלו יעיל ואינו תלוי באורך הרשימה.
- `T get(int i)` – היא שיטה המחזירה את הערך במקום ה- `i`. **שימו לב**, שיטה זו אינה יעילה.
- `Iterator<T> iterator()` – היא שיטה המחזירה איטרטור, שעובר בצורה יעילה על אברי הרשימה מתחילת הרשימה לסופה.

לפני שאתם ממשיכים, גשו לנספח וקראו אותו. הנספח מפשט 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). לדוגמא, המספר הבינארי **11101** (29 בבסיס 10) מיוצג על ידי הרשימה המקושרת שבתמונה. שימו לב, הרשימה מצוירת מימין למשאל תוך התאמה למסוכמה של כתיבת מספרים שבה LSB הוא תמיד מימין.



### השלמת המחלקה BitList:

המחלקה BitList מרחיבה את המחלקה LinkedList מהספרייה הסטנדרטית של Java. יש לה שדה אחד `numberOfOnes` המייצג את מספר הביטים שערכם 1.

המחלקה BitList מייצגת רשימה של ביטים. רשימות כאלו יכולות לייצג מספרים בינאריים, אבל יש רשימות שאינן מייצגות מספרים כלל (למשל הרשימה הריקה).

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.

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

### בכל המשימות הבאות אין להוסיף שדות למחלקה.

## משימה 2.1 – השיטות void addLast(Bit), void addFirst(Bit), Bit removeFirst() ו- Bit removeLast() במחלקה BitList (4 נקודות)

המחלקה BitList דורסת את השיטות:

- void addFirst(Bit)
- void addLast(Bit)
- Bit removeFirst()
- Bit removeLast()

עליכם להשלים את הגדרת השיטות האלו כך שתזרוק 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.

דוגמאות:

ברשימה הריקה (שנסמן על ידי  $\langle \rangle$ ) ערך השדה numberOfOnes יהיה 0. אחרי ביצוע הפקודה addFirst(Bit.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! העלו את הגרסה האחרונה של עבודתכם למערכת המודל.

## משימה 2.2 – השיטה String toString() במחלקה BitList (2 נקודות)

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

דוגמאות:

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

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

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



## משימה 2.3 – הבנאי המעתיק של המחלקה BitList (5 נקודות)

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.

דוגמאות:

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

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

## משימה 2.4 – השיטה boolean isNumber() במחלקה BitList (2 נקודות)

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

- לרשימת ביטים ריקה אין משמעות מספרית.

פורמאלית, לרשימה יש משמעות מספרית אם:

- אורכה לפחות 1.

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

דוגמאות:

- הרשימות <110> ו- <1101> (מייצגות את המספרים העשרוניים 6 ו- 13 בהתאמה) הן מספרים כי אורכן שלוש וארבע בהתאמה (הסוגריים הזוויתיים אינם נספרים).
- הרשימות <0110> ו- <01101> (מייצגות את המספרים העשרוניים 6 ו- 13 בהתאמה) הן מספרים כי אורכן ארבע וחמש בהתאמה (גם אם הן מסתיימות ב- 0).
- הרשימה <> אינה מייצגת מספר כי אורכה אפס.

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

**משימה 2.5 – השיטות `boolean isReduced()` ו-`void reduce()` במחלקה `BitList` (4 נקודות)**

רשימת הביטים `<000001>` ארוכה יותר מהרשימה `<1>` אבל שתיהן ייצוגים בינאריים של המספר 1. אי אפשר לקצר עוד את `<1>` מבלי להפוך אותה לבלתי חוקית (`<>>`) או לשנות את ערכה (`<0>`). לכן נקרא ל-`<1>` רשימה מינימאלית (`reduced`) ונאמר שהיא הייצוג הבינארי המינימאלי של המספר העשרוני 1. ל-`<00001>` 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** מהתנאים הבאים:
  - א. היא הרשימה `<0>` או `<1>`.
  - ב. הביט השמאלי ביותר אינו 0.

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 שלו. אם העצם היה מינימאלי מלכתחילה לא יחול בו שינוי.

**דוגמאות:**

- 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>`.
- רשימת הביטים `<00010>` מייצגת את המספר שתיים אך אינה ייצוג מינימאלי שלו. אפשר לצמצם אותה על ידי הסרת שלושת הביטים השמאליים ולקבל את הייצוג הבינארי המינימאלי של שתיים `<10>`.
- הרשימה `<1010>` מייצגת את המספר עשר והיא הייצוג המינימאלי שלו.

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

## משימה 2.6 – השיטות Bit shiftRight() ו void shiftLeft() במחלקה BitList (5 נקודות)

הזזה ימינה (shift right) של רשימת ביטים, שאורכה יותר מביט אחד, היא הפעולה של הסרת הביט הראשון (הימני ביותר). לדוגמא, הזזה ימינה של הרשימה <011> יוצרת את הרשימה <01>. הזזה ימינה של רשימה בת ביט אחד, למשל <1> יוצרת את הרשימה <0>, 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). לדוגמא, 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, ולעיתים קרובות ממומשות בחומרה. ([https://en.wikipedia.org/wiki/Arithmetic\\_shift](https://en.wikipedia.org/wiki/Arithmetic_shift))

להזזת ימינה ושמאלה יש משמעות של חלוקה וכפל ב-2 בהתאמה עבור unsigned integer, כאשר הביט שמוסר בהזזה ימינה הוא שארית החלוקה ב-2.

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:

- רשימת הביטים <10> מייצגת את המספר שתיים, לאחר הפעלת השיטה shiftLeft(), הרשימה המתקבלת היא <100> המייצגת את המספר 4.
- רשימת הביטים <110> מייצגת את המספר שש, לאחר הפעלת השיטה shiftLeft(), הרשימה המתקבלת היא <1100> המייצגת את המספר 12.
- רשימת הביטים <101> מייצגת את המספר חמש, לאחר הפעלת השיטה shiftRight(), (על ידי הסרת הביט הימני 1, שארית החלוקה של 5 ב-2) נוצרת הסדרה <10> המייצגת את המספר 2. הזזה נוספת ימינה יוצרת את <1> המייצגת את 1 וההזזה נוספת את <0> (ושארית 1).
- דוגמה עבור shiftRight():

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

- דוגמה עבור shiftLeft():

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

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

## משימה 2.7 – השיטה void padding(int newLength) במחלקה BitList (5 נקודות)

ריפוד של רשימת ביטים היא פעולה שבה מרפדים באפסים משמאל לביט האחרון (השמאלי) מספר פעמים.

בזמן שנממש את הפעולות האריתמטיות במחלקה BinaryNumber ייתכן שיהיה לנו נוח לעבוד עם רשימות ביטים שאינן מינימאליות. "ריפוד" הרשימה בחזרות על הביט 0 יוצר רשימת ביטים חדשה המייצגת את אותו מספר.

לדוגמה, ניתן להגיע מהרשימה המינימאלית <11> המייצגת את המספר 3, לרשימה הלא מינימאלית <00011> המייצגת את אותו מספר על ידי הוספת הביט 0 שלוש פעמים.

עליכם להשלים את השיטה void padding(int newLength). שיטה זו משנה את העצם המפעיל אותה על ידי הוספת 0 לסופה עד שאורך הרשימה מגיע לערך של הפרמטר newLength. אם הפרמטר קטן או שווה לאורך הרשימה, השיטה אינה עושה דבר.

[דוגמאות:](#)

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

## חלק שלישי – השלמת המחלקה BinaryNumber

בחלק זה של תשלימו את המחלקה BinaryNumber המייצגת מספרים בינאריים שלמים (חיוביים בלבד). לשם כך, היא מחזיקה שדה פרטי יחיד bits מהטיפוס BitList.

מימשנו עבורכם את השיטות והשדות הבאים (חלקם בעזרת השיטות שכתבתם בחלקים הקודמים):

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:

- בשיטות שתכתבו עליכם לוודא שכל המופעים של `BinaryNumber`, כפרמטרים והערך המוחזר, הם חוקיים (מספר + מינימאלי).
- הבהרה: ניתן להשתמש בייצוג לא מינימאלי ובלבד שבסיום השיטה יחזור להיות מינימאלי.
- 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.

### הערה חשובה:

ככלל, אתם נדרשים לכתוב פתרונות יעילים בתשובותיכם. בחלק מן הסעיפים בחלק זה של העבודה ניתן אף לחשוב על מימוש רקורסיבי יעיל לפתרון. עם זאת, אין להשתמש ברקורסיה בפתרון הבעיות. כידוע אנחנו מאוד אוהבים רקורסיה בקורס והיא חלק בלתי נפרד מהתחום שלנו אבל ב-Java יש מגבלה על מספר הקריאות הרקורסיביות שיכולות להיות פתוחות בו-זמנית. כתוצאה מכך פתרונות רקורסיביים יהיה מוגבלים בגודל הקלט שהם יכולים לקבל בעוד שהמוטיבציה לעבודה הזו היא הסרת המגבלות שיש לנו בעבודה עם מספרים במימוש הקיים ב-Java. אנו מזמינים אתכם לנסות ולחשוב על מימושים רקורסיביים לצורכי תרגול אך נבקש שלא תגישו פתרונות אלו.

## משימה 3.1 – הבנאי `BinaryNumber(char c)` במחלקה `BinaryNumber` (4 נקודות)

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

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.

**משימה 3.2 – השיטה `String toString()` במחלקה `BinaryNumber` (4 נקודות)**

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 `BitList` אבל בלי הסוגריים הזוויתיים).

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

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

[דוגמאות:](#)

- הייצוג הבינארי המינימאלי של המספר העשרוני 5 הוא `<101>`

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

**משימה 3.3 – השיטה `boolean equals(Object other)` במחלקה `BinaryNumber` (4 נקודות)**

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

[דוגמאות:](#)

```
BinaryNumber bn5 = new BinaryNumber('5'); // 101 (5)
BinaryNumber bn5a = new BinaryNumber('5'); // 101 (5)
BinaryNumber bn6 = new BinaryNumber('6'); // 110 (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 המודל.

**משימה 3.4 – השיטה `BinaryNumber add(BinaryNumber addMe)` במחלקה****`BinaryNumber` (5 נקודות)**

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

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

**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.

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

דוגמאות:

```

BinaryNumber bn3 = new BinaryNumber('3'); // 11 (3)
BinaryNumber bn5 = new BinaryNumber('5'); // 101 (5)
BinaryNumber bn8 = new BinaryNumber('8'); // 1000 (8)
System.out.println(bn3.add(bn5));          // 1000 (the minimal
                                           // representation of 8)

System.out.println(bn8.add(bn3));          // 1011 (the minimal
                                           // representation of 11)

```

**משימה 3.5 – השיטה `int compareTo(BinaryNumber other)` במחלקה `BinaryNumber`**(5 נקודות)

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.

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

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

**הכוונה:**

1. שימו לב, שעבור `unsigned int`, ההפרש מוגדר רק עבור מספר קטן שמחוסר ממספר גדול, אחרת – אי אפשר לבצע חיסור.

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

דוגמאות:

```

BinaryNumber bn5 = new BinaryNumber('5'); // 101 (5)
BinaryNumber bn4 = new BinaryNumber('4'); // 100 (4)
BinaryNumber bn4a = new BinaryNumber('4'); // 100 (4)
System.out.println(bn5.compareTo(bn4));    // prints 1
System.out.println(bn4.compareTo(bn4a));    // prints 0
System.out.println(bn4.compareTo(bn5));    // prints -1

```

### משימה 3.6 – השיטה `BinaryNumber subtract(BinaryNumber subtractMe)` במחלקה `BinaryNumber` (5 נקודות)

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

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

**הכוונה:**

1. שימו לב, שעבור `unsigned int`, ההפרש מוגדר רק עבור מספר קטן שמחוסר ממספר גדול, אחרת – אי אפשר לבצע את החיסור ועליכם להחזיר חריגה מהטיפוס `IllegalArgumentException`.
2. אל תישכחו שהערך המוחזר צריך להיות לא רק נכון, אלא גם מינימאלי.

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

דוגמאות:

```
BinaryNumber bn3 = new BinaryNumber('3'); // 11 (3)
BinaryNumber bn5 = new BinaryNumber('5'); // 101 (5)
BinaryNumber bn8 = new BinaryNumber('8'); // 1000 (8)

System.out.println(bn3.subtract(bn5)); // IllegalArgumentException
System.out.println(bn5.subtract(bn3)); // 10 (2)
System.out.println(bn8.subtract(bn3)); // 101 (5)
```

### משימה 3.7 – השיטה `int toInt()` במחלקה `BinaryNumber` (5 נקודות)

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

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

דוגמאות:

```
BinaryNumber bn5 = new BinaryNumber('5'); // 101 (5)
BinaryNumber bn4 = new BinaryNumber('4'); // 100 (4)
System.out.println(bn5.add(bn4).toInt()); // prints 9
System.out.println(bn5.subtract(bn4).toInt()); // prints 1
```



### משימה 3.8 – השיטה BinaryNumber multiply(BinaryNumber multiplyMe) במחלקה BinaryNumber (6 נקודות)

השיטה מחזירה עצם המייצג את המכפלה המיוצגת על ידי העצם המבצע והמספר המיוצג על ידי הפרמטר, כלומר העצם המבצע כפול הפרמטר.

דוגמאות:

```
BinaryNumber bn5 = new BinaryNumber('5'); // 101 (5)
BinaryNumber bn4 = new BinaryNumber('4'); // 100 (4)
BinaryNumber bnM20 = bn5.multiply(bn4); // 1100 (20)
System.out.println(bnM20.toInt()); // prints 20
```

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

הנחיה: עקרונית, ניתן בפשטות יחסית לממש כפל כחיבור חוזר.  $(n \times m = \overbrace{n + n + \dots + n}^{m \text{ times}})$  יעיל. ניקוד מלא יינתן רק לפתרון יעיל.

### משימה 3.9 – השיטה BinaryNumber divide(BinaryNumber divisor) במחלקה BinaryNumber (6 נקודות)

השיטה מחזירה עצם המייצג את מנת החלוקה המיוצגת על ידי העצם המבצע במספר המיוצג על ידי הפרמטר, כלומר מפעולת החלוקה מוחזר ערך שלם גם כאשר קיימת שארית. לדוגמא  $8/3=2$ .

דוגמאות:

```
BinaryNumber bn9 = new BinaryNumber('9'); // 1001 (9)
BinaryNumber bn3 = new BinaryNumber('3'); // 11 (3)
BinaryNumber bnM3 = bn9.divide(bn3); // 11 (3)
System.out.println(bnM3.toInt()); // prints 3
BinaryNumber bnM0 = bn3.divide(bn9); // 0 (0)
System.out.println(bnM0.toInt()); // prints 0
```

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

הנחיה: ניתן בפשטות יחסית לממש חילוק בעזרת חזרה על פעולת החיסור, אולם מימוש זה הוא לא יעיל. ניקוד מלא יינתן רק לפתרון יעיל. אין להניח שהקלט תקין. יש לבדוק את תקינות הקלט ולזרוק חריגה מהטיפוס IllegalArgumentException אם הקלט אינו תקין.

### משימה 3.10 – השיטה BinaryNumber mod(BinaryNumber modulus) במחלקה BinaryNumber (6 נקודות)

השיטה מחזירה עצם המייצג את שארית החלוקה של העצם המבצע במספר המיוצג על ידי הפרמטר. לדוגמה  $8\%3=2$ .

דוגמאות:

```
BinaryNumber bn8 = new BinaryNumber('8');           // 1000 (8)
BinaryNumber bn3 = new BinaryNumber('3');           // 11 (3)
BinaryNumber bn8MOD3 = bn8.mod(bn3);                // 10 (2)
System.out.println(bn8MOD3.toInt());                // prints 2
BinaryNumber bn3MOD8 = bn3.mod(bn8);                // 11 (3)
System.out.println(bn3MOD8.toInt());                // prints 3
```

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

הנחיה: ניתן בפשטות יחסית לממש את פעולת חישוב השארית בעזרת חזרה על פעולת החיסור עד לקבלת מספר הנמוך בערכו מערך המודולוס ולהחזירו, אולם מימוש זה הוא לא יעיל. ניקוד מלא יינתן רק לפתרון יעיל. אין להניח שהקלט תקין. יש לבדוק את תקינות הקלט ולזרוק חריגה מהטיפוס IllegalArgumentException אם הקלט אינו תקין.

### משימה 3.11 – הבנאי BinaryNumber(String s) במחלקה BinaryNumber (5 נקודות)

הבנאי מקבל מחרוזת באורך כלשהו המייצגת מספר עשרוני שלם, אי שלילי. הבנאי יוצר עצם המייצג מספר זה.

דוגמאות:

```
BinaryNumber bn11 = new BinaryNumber("11");         // 1011 (11)
BinaryNumber bn15 = new BinaryNumber("15");         // 1111 (15)
System.out.println(bn11.toInt());                   // prints 11
System.out.println(bn15.toInt());                   // prints 15
```

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

### משימה 3.12 – השיטה `toString()` במחלקה `BinaryNumber` (6 נקודות)

השיטה `String toString()` מחזירה מחרוזת המייצגת את העצם הפועל כמספר בבסיס 10.

הערה: השיטה צריכה לפעול לכל עצם בלי קשר לגודלו, גם אם העצם מייצג מספר בינארי שלא ניתן לייצג את הערך שלו באחד הטיפוסים הפרימיטיביים.

דוגמאות:

```
BinaryNumber bn9 = new BinaryNumber('9'); // 1001 (9)
System.out.println(bn9.toString()); // prints 9
BinaryNumber fib100 = new BinaryNumber("354224848179261915075")
//
100110011001111011011011
101101010011111000101100
101001011111111000011
System.out.println(fib100.toString()); // prints
354224848179261915075
```

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

### חלק רביעי – השלמת המחלקה `BinaryPrimesIterator`

בחלק זה של העבודה נממש איטרטור של מספרים ראשוניים שעושה שימוש במחלקה `BinaryNumber` שמימשנו בחלק הקודם. האיטרטור יעשה שימוש בלעדי במחלקה זו וגם המספרים הראשוניים שהוא יחזיר יהיו מיוצגים בבסיס בינארי ע"י מופע של המחלקה `BinaryNumber`.

שימו לב שזהו איטרטור גנרטיבי – הערך הבא להחזרה מחושב מראש בתוך המחלקה `BinaryPrimesIterator` ולא מגיע מאוסף נתונים קיים (בדומה לאיטרטור פיבונאצ'י שנלמד בהרצאות).

את בדיקת הראשוניות של מספר מועמד אנחנו נעשה כפי שלמדנו בהרצאות הראשונות (חיפוש מחלק של המספר שאנחנו בודקים) **ולא** כפי שעשינו בעבודה 1.

במה האיטרטור שנממש יהיה שונה מהאיטרטור פיבונאצ'י שראינו בהרצאות?

המחלקה `BinaryNumber` מאפשרת לנו לעבוד עם מספרים גדולים כרצוננו וזאת להבדיל מעבודה עם המשתנים הקיימים ב – Java שמוגבלים מראש בטווח הערכים שהם יכולים לקבל. בשל כך, אנחנו לא נגביל את האיטרטור בערך מקסימלי וכל עוד המשתמש ירצה בכך, הוא יוכל לקבל את המספר הראשוני הבא.

כמו כן נזכיר שוב את הטענה הנשמרת עבור איטרטורים והיא שהערך הבא להחזרה כבר מחושב מבעוד מועד ומוכן לשליפה. לפני החזרתו נדאג שיהיה שמור לנו כבר האיבר הבא אחריו.

### מוזמנים לעצור ולחשוב – אילו שדות היינו רוצים במחלקה שלנו?

**השלמת המחלקה BinaryPrimesIterator:**

המחלקה BinaryPrimesIterator מממשת את הממשק `Iterator<T>` מהספרייה הסטנדרטית של Java כאשר `T=BinaryNumber`. במחלקה BinaryPrimesIterator יהיו 2 שדות:

```
private BinaryNumber numberOfGeneratedPrimes;
private LinkedList<BinaryNumber> primes;
```

השדה `numberOfGeneratedPrimes` ייצג את מספר המספרים הראשוניים שנוצרו עד כה ע"י האיטרטור. השדה `primes` יהיה רשימת המספרים הראשוניים שנוצרו עד כה וישמש אותנו לבדיקת ראשוניות של המועמדים הבאים.

מימשנו עבורכם את השיטות הבאות ואין לשנותן:

- `public BinaryPrimesIterator()` – בנאי ריק
- `hasNext()` – שיטת ממשק
- `get()` – שיטת
- `numberOfGeneratedPrimes` – שיטה לקידום השדה

**הערה:** נשים לב שגודל הרשימה `primes` שווה ערך למונה `numberOfGeneratedPrimes` אך בכל זאת בחרנו להשתמש במשתנה במחלקה שייצג את מספר הראשוניים שנוצרו עד כה. הסיבה להחלטה הזו נובעת מהעובדה שבמחלקה `LinkedList<T>` של Java, גודל הרשימה מוחזר ע"י משתנה מטיפוס `int` מה שעלול ליצור בעיה אם המשתמש ייצר יותר מספרים ראשוניים מערכו המקסימלי של משתנה מטיפוס `int`. אפילו  $2^{32}+1$  הוא מספר גדול של מספרים ראשוניים ובאופן מעשי קרוב לוודאי שלמחשבים שבהם אנחנו (מורים ותלמידים) משתמשים כיום אין מספיק זיכרון כדי לשמור כל כך הרבה מספרים גדולים בייצוג שבתנו (להערכתנו, לפחות טרה בייט). אבל, כבר כיום קיימים מחשבים בעלי כמות כזו של זיכרון ובעתיד מן הסתם הם יהיו שווים לכל נפש. מבחינת זמן ריצה אנחנו מעריכים (באופן גס מאוד) שאלמלא מגבלת הזיכרון החישוב היה דורש כיומיים. הרבה חישוביים מדעיים שימושיים דורשים זמן מחשב רב בהרבה. הקוד שתכתבו יהיה מוגבל רק על ידי האילוצים של החומרה, ותיאורטית יוכל לתמוך במספרים בכל סדר גודל, אם תמצא החומרה המתאימה.

**משימה 4.1 – השיטה BinaryNumber next() במחלקה****BinaryPrimesIterator (6 נקודות)**

השיטה `BinaryNumber next()` מחזירה את המספר הראשוני הבא ביחס למספרים הראשוניים שכבר הוחזרו.





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

**דוגמאות:**

```
BinaryPrimesIterator bpi = new BinaryPrimesIterator();
System.out.println(bpi.next().toIntString()); // 2
System.out.println(bpi.next().toIntString()); // 3
System.out.println(bpi.next().toIntString()); // 5
System.out.print("Generated primes so far: ");
System.out.println(bpi.getNumberOfGeneratedPrimes().toIntString()); // 3
System.out.print("For loop...");
for (int i = 0; i < 8; i += 1) { bpi.next(); };
System.out.println("Generated primes so far: ");
System.out.println(bpi.getNumberOfGeneratedPrimes().toIntString()); // 8
System.out.println(bpi.next().toIntString()); // 23
```

**הנחיה:** יש להתבסס על המימוש היעיל ביותר שלמדנו.

הוראות הגשה:

1. גשו ל-**עבודת בית 4 – VPL** באתר הקורס.
2. גשו ללשונית Edit (עריכה).
3. לחצו על הכפתור ה-.
4. יפתחו לכם עוד אופציות, בין היתר אופציה של  upload לחצו על הכפתור ובחרו את הקבצים שערכתם בפרויקט Assignment4. **ודאו כי לא חסרים קבצים וכי הקבצים שהעליתם הם הקבצים המעודכנים ביותר.**
5. שמרו את השינויים (יש לחוץ על כפתור השמירה) .
6. לחצו על Evaluate .
7. אתם אמורים לקבל פידבק עבור הצלחתכם בבדיקות החלקיות שרצות בזמן הגשה זו (בדיקות נוספות יתבצעו בתום תאריך ההגשה).
8. אנו חוזרים ואומרים, זו אחריותכם לבדוק שהקבצים שהגשתם עוברים תהליך קומפילציה **במערכת**. **עבודות שלא יתקמפלו יקבלו את הציון 0.**

## בהצלחה !