מבוא לתכנות – עבודה 4

מועד פרסום: 21/12/2017
מועד אחרון להגשה: 7/1/2018

מתרגלת אחראית: יעל מה טוב

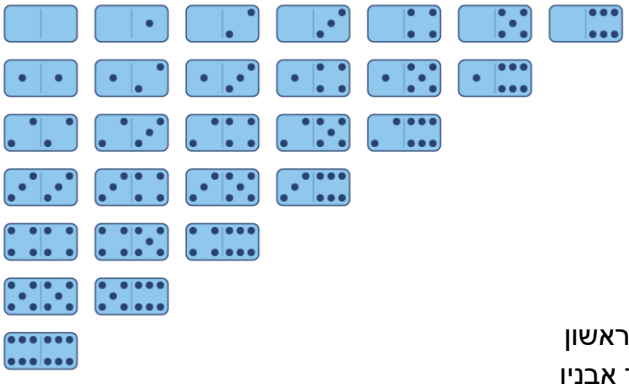
הוראות מקדימות

1. העבודה כתובה בלשון זכר (מטעמי נוחיות) אך פונה לשני המינים (ולחדי קרן).
2. קראו את העבודה במלואה לפני שאתם מתחילים לפתור אותה, וודאו הבנה של השאלות וההערות.
3. יש לבצע ולהגיש את העבודה **כיחידים**, כלומר כל סטודנט יבצע ויגיש את עבודתו שלו **לבד**.
4. בתחילת **כל קובץ** יש לשנות את השורה המתאימה בהערות כך שתכיל את שם המלא ותעודת הזהות של המגיש. לדוגמה, במקום "ADD YOUR NAME & ID" יש לכתוב "Yael Mathov 123456789"
5. יש להוסיף את הפתרונות שלכם לחמשת הקבצים שסיפקנו לכם ולהגיש אותם בקובץ zip יחיד. **אין להגיש קבצים נוספים** ואין ליצור תיקיות מיותרות (כמו אלו הנוצרות לדוגמה ביצירת package חדש ב-eclipse).
6. שם הקובץ המכווץ יהיה מספר תעודת הזהות של הסטודנט. לדוגמה, 123456789.zip.
7. **אין לשנות את שמות המחלקות או השיטות בקבצים**! במספר שיטות יש ערך החזרה שנועד לאפשר לקוד להתקמפל גם אם הוא לא מלא – ניתן להסיר אותם.
   עבודות שיוגשו בפורמט שונה מהמתואר לא ייבדקו **והמגיש יקבל אפס**.
8. את קובץ ה-zip יש להגיש דרך מערכת ההגשה: https://subsys.ise.bgu.ac.il/submission/login.aspx
9. ניתן להגיש את הקובץ מספר פעמים עד לתאריך ההגשה, ההגשה האחרונה היא זו שתיבדק.
10. העבודה תיבדק באופן אוטומטי לפי הפלט אשר התוכניות שלכם תדפיס למסך. לכן, אנא הדפיסו אך ורק את הפלט הנדרש, הקפידו על ההוראות ובצעו אותן במדויק. כל הדפסה אשר אינה עונה באופן מדויק לדרישות המופיעות בעבודה (כולל שורות, רווחים, סימני פיסוק, או כל תו אחר מיותר, חסר, או מופיע בסדר שונה מהנדרש), תגרור פגיעה משמעותית בציון (80 נקודות).
11. סגנון כתיבת הקוד ייבדק באופן ידני (20 נקודות). יש להקפיד על כתיבת קוד ברור, מתן שמות משמעותיים למשתנים, הזחות (אינדנטציה), והוספת הערות בקוד המסבירות את תפקידם של מקטעי הקוד השונים שלדעתכם אינם אינטואיטיביים להבנה במבט ראשון ובתחילת כל פונקציה שאתם כותבים.
12. במידה ואינכם בטוחים מהו הפירוש המדויק להוראה מסוימת או אם יש לכם שאלה אחרת הקשורה לתוכן העבודה, אנא היעזרו בפורום או בשעות הקבלה של האחראית על העבודה. בכל בעיה אישית הקשורה בעבודה (מילואים, אשפוז וכו') אנא צרו את הפניה המתאימה למייל הקורס (ise.intro.181@gmail.com).
13. ניתן לפתור את העבודה במעבדות המחשבים של האוניברסיטה או במחשבים האישיים. שימו לב! העבודות יבדקו במחשבי האוניברסיטה ולכן חובה לוודא שהעבודה עובדת כראוי על מחשבי האוניברסיטה!
14. **אין להשתמש בחומר שאינו נלמד** בכיתה עד לרגע פרסום התרגיל.
15. העבודות יבדקו למציאת העתקות. סטודנטים שיתפסו מעתיקים ישלח לוועדת משמעת. לכן אין לשתף קוד! כל סטודנט כותב את העבודה שלו לבד.

בתרגיל זה תממשו משחק דומינו קבוצתי.
מטרת התרגיל הוא לימוד השימוש באובייקטים ושמירה על עקרון ההכמסה.

כללי המשחק

דומינו הוא משחק המורכב מ-28 חלקים הנקראים אבנים.
אבני המשחק מלבניות וקו מחלק אותן באמצע לשני ריבועים,
שעל כל אחד מהם מסומנות מספר נקודות (ראה תמונה 1).
נהוג להשתמש באבנים עם אפס עד שש נקודות, כאשר כל
אבן מופיעה פעם אחת בלבד.



תמונה 1. אבני הדומינו השונות.

בתחילת המשחק כל שחקן מקבל מאגר של אבני דומינו
הנסתרות מעיני האחרים (מספר האבנים יכול להשתנות ממשחק
למשחק), ומתחילים את המשחק עם שולחן ריק. אחרי שהשחקן הראשון
מניח את האבן הראשונה, השחקן הבא בתור לוקח אבן אחת מתוך אבניו
שאחד מריבועיה זהה לאחד מהריבועים החיצוניים של האבנים בלוח
ומניח אותה בסמוך אל אותה אבן (ראו תמונה 2).

שימו לב כי בכל רגע נתון במשחק ישנם שני ריבועים אשר ניתן להניח אבן בסמיכות להם.
למידע נוסף:

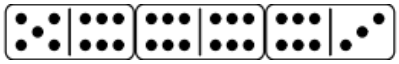
http://en.wikipedia.org/wiki/Dominoes

במשחק דומינו קבוצתי משחקות שתי קבוצות של שחקנים.
כל קבוצה בתורה מבצעת מהלך:
השחקן הבא בתור באותה קבוצה מניח אבן דומינו המתאימה לאחד הקצוות הפתוחים של הלוח.
אם אין לשחקן אף אבן מתאימה, התור עובר לשחקן הבא בקבוצה.
אם אין לאף שחקן בקבוצה אבן מתאימה התור עובר לקבוצה השנייה.
המשחק מסתיים כאשר לאחת הקבוצות נגמרו האבנים או אם לשתי הקבוצות אין דומינו מתאים.
בסיום המשחק מתבצע חישוב נקודות על האבנים שנותרו ברשות כל קבוצה, והקבוצה עם פחות הנקודות מנצחת.

האובייקטים בתרגיל

האובייקטים השונים בתרגיל זה:

1. `Tile` - מתאר אבן דומינו.
2. `Board` - מתאר את לוח המשחק.
3. `Player` - מתאר שחקן יחיד.
4. `Team` - מתאר קבוצת שחקנים.
5. `GameManager` - מתאר את מנהל המשחק.



תמונה 2. אבני דומינו בלוח משחק.
שימו לב כי ניתן להניח אבן חדשה
בתחילת שרשרת האבנים או בסופה.

הערות לגבי המשימות השונות

במשימות הבאות תידרשו להגדיר טיפוסים שונים (מחלקות). כל מחלקה צריכה לקיים את הדרישות הבאות, גם אם הדרישה לא צוינה באופן מפורש:

1. כל שדות המחלקה **חייבים** להיות `private`.
2. כל מחלקה **חייבת** להכיל את השיטות `toString` ו-`equals` **גם אם לא צוין במפורש**.
3. לאורך כל העבודה **יש לשמור על עקרון ההכמסה!** שדות פרטיים לא צריכים להיות חשופים או ניתנים לשינוי על ידי המשתמש בשום צורה.
4. בהמשך לסעיף הקודם: כל שיטת עזר שאתם מוסיפים **חייבת** להיות פרטית. בפרט **לא** ניתן להוסיף getters ו-setters ציבוריים מעבר למה שצוין במפורש!
5. כל מחלקה חייבת להכיל **לפחות** את הבנאי שצוין בהוראות במפורש. ניתן להוסיף בנאים נוספים (שכמובן יכולים להיות ציבוריים) כרצונכם. אנא הקפידו לשמור על עקרון ההכמסה.
6. לא ניתן להניח שום דבר לגבי תקינות הפרמטרים לשיטות שעליכם לממש מלבד מה שנכתב במפורש!
7. **ניתן להניח לאורך כל המשחק כי אין חזרות או כפילויות של אבני המשחק.**

שימו לב – **אם הקוד שלכם אינו עומד בעקרון ההכמסה לא תוכלו להפעיל את הטסטים!**

תיעוד בעזרת Javadoc

כל הקוד בעבודה חייב להיות מתועד בעזרת Javadoc. מידע נוסף על תיעוד בעזרת Javadoc ניתן למצוא בלינק הבא: http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

בכדי לסייע לכם, סיפקנו את ההערות עבור חלק מהשיטות. עבור יתר השיטות ניתן לכתוב ב-eclipse בשורה מעל השיטה את התווים **/ ולאחר מכן ללחוץ על Enter – פעולה זו תייצר את הבסיס לתיעוד באופן אוטומטי.

בתחילת כל מחלקה יש לתאר למה היא משמשת ומי המחבר שלה (עם התיוג המתאים).
עבור כל שיטה, יש להקפיד לתאר בצורה ברורה מה הפונקציה עושה, מה הפרמטרים שהיא מקבלת ומה היא מחזירה. יש להשתמש בתיוגים המתאימים של Javadoc.

הקפידו לחשוף בהערות רק מידע נחוץ עבור המשתמש תוך שמירה על עיקרון ההכמסה.

בדיקות

אתם מחויבים לכתוב טסטים לבדיקת הקוד שלכם בעצמכם לפני העלאת העבודה למערכת ההגשה.
כתיבת הטסטים עוזרת לחדד ולהבין טוב יותר את הבעיה אותה נדרשתם לפתור.

במידה ואתם מעוניינים להרחיב את הידע שלכם אנו מזמינים אתכם לקרוא על JUnit. זהו כלי שנועד לסייע בבניית טסטים לתוכנות ג'אווה שניתן להשתמש בו בקלות ב-Eclipse. קיימים מדריכים רבים באינטרנט וניתן למצוא אותם בחיפוש פשוט ב-Google או ב-YouTube.

שימוש בכלי ב-JUnit הוא רשות אך כתיבת טסטים לעבודה היא חובה! ניתן לבדוק את הקוד שלכם ע"י הוספת מחלקה נוספת עם פונקציית main. בכל מקרה **אין להגיש את הטסטים או את המחלקה הנוספת.**

הטסטים במערכת ההגשה לא נועדו כדי לדבג לכם את הקוד! מסיבה זו קיימת מגבלת זמן בין ההגשות.

משימה 0 - הצהרה כי לא העתקת את הקוד

צרו קובץ טקסט בשם "CodeIntegrity.txt" חדש ובו תכתבו:

"אני **<שם פרטי ושם משפחה>** ת"ז **<ת"ז>** מצהיר/ה שהקוד שמוגש בעבודה זו נכתב אך ורק על ידי. לא חלקתי את הקוד הזה עם סטודנטים אחרים ולא נעזרתי בקוד של סטודנטים אחרים.

הובהר לי כי לא רק שהעתקה היא עבירה חמורה מאוד אלא גם שבהתאם לנהלי האוניברסיטה סגל הקורס לא יטפל בחשדות להעתקה וכי הטיפול באלו שיתגלו יועבר לוועדת משמעת."

עבודה שתוגש ללא קובץ זה תקבל ציון אפס.

משימה 1 - אובייקט "אבן דומינו"

הגדירו מחלקה בשם `Tile` שתייצג אבן דומינו. אבן דומינו תיוצג ע"י שני מספרים שלמים בטווח 0-6.
מספרים אלו מתארים את ערכי אבן הדומינו.

שימו לב כי ניתן לייצג אבן בשתי דרכים ולכן האבן <2,1> זהה לאבן <1,2>

Constructor

הבנאי של המחלקה `Tile` חייב לתמוך באפשרות הבאה:

```
public Tile(int leftNumber, int rightNumber)
```

המספרים `rightNumber` ו-`leftNumber` מתארים את ערכי אבן הדומינו בריבוע הימני והשמאלי בהתאמה.
ניתן להניח כי הקלט תקין (0-6).

Getters

הוסיפו getters לערך הימני והשמאלי של האבן הדומינו.

```
public int getLeftNumber()
public int getRightNumber()
```

flipTile

הגדירו שיטה `flipTile` אשר הופכת את האבן.
כלומר, לאחר ביצוע השיטה, החלק השמאלי יוחלף עם החלק הימני.

```
public void flipTile()
```

toString

השיטה `toString` של מחלקה זו צריכה להחזיר מחרוזת המכילה את ערכי האבן. לדוגמה, המחרוזת הבאה מתארת אבן משחק אשר הריבוע הימני מכיל שתי נקודות, והריבוע השמאלי מכיל נקודה אחת:

`"<1,2>"`

משימה 2 - אובייקט "לוח משחק דומינו"

הגדירו מחלקה בשם `Board` שתייצג את לוח משחק הדומינו. בכל רגע נתון, על אובייקט המשחק לדעת את סדר אבני הלוח, ובפרט מהן אבני הדומינו אשר נמצאות בקצוות שרשרת האבנים.
האבן הראשונה בלוח היא האבן השמאלית ביותר בלוח והאבן האחרונה היא הימנית ביותר.
שני לוחות יהיו זהים במידה והם מכילים את אותן אבנים באותו הסדר (משמאל לימין).

המלצה: השתמשו בשני מערכים, אחד עבור הרחבה ימינה ואחר עבור שמאלה.
ניתן לפתור את הבעיה בדרכים רבות ואין חובה לעקוב אחרי ההמלצה.

Constructor

הבנאי של המחלקה צריך לתמוך באפשרות הבאה לפחות:

```
public Board(int numOfTiles)
```

הבנאי מייצר לוח ריק כש-`numOfTiles` הוא מס' גדול מאפס וקטן או שווה ל-28 $(0 < \text{numOfTiles} \leq 28)$
שמיצג את מספר האבנים המרבי שיכולות להיות בלוח. הלוח לא יוכל להכיל יותר מ-`numOfTiles` אבנים.

Getters

הגדירו שתי שיטות אשר יחזירו את ערכי המספרים הקיצוניים בלוח:

```
public int getRightValue()
public int getLeftValue()
```

במידה ולא הונחו עדיין אבנים כלל, השיטות יחזירו 1-.

getBoard

הגדירו שיטה בשם `getBoard` שתחזיר מערך של אבני דומינו כפי שמופיעים בלוח באותו רגע.

```
public Tile[] getBoard()
```

התא הראשון במערך יכיל את האבן הראשונה (הפנייה לאובייקט), והתא האחרון יכיל את האבן האחרונה (הפנייה לאובייקט). במידה והלוח ריק השיטה תחזיר null.

addToLeft\RightEnd

הגדירו שתי שיטות המאפשרות הוספה של אבן דומינו בקצותיו של הלוח:

```
public boolean addToRightEnd(Tile tile)
public boolean addToLeftEnd(Tile tile)
```

במידה ואחד מצדי האבן מתאים לקצה הנדרש, הפעולה **תעדכן את לוח** המשחק ותחזיר true. במידה ולא ניתן להוסיף את האבן ללוח (מכל סיבה), השיטה תחזיר false. במידה והלוח ריק והקלט תקין השיטה תצליח לכל אבן.

toString

השיטה `toString` של מחלקה זו צריכה להחזיר מחרוזת המכילה את ערכי האבנים כפי שמסודרות משמאל לימין. לדוגמה, המחרוזת הבאה מתארת לוח המכיל שלוש אבנים (שימו לב כמובן שהאבנים מונחות באופן חוקי):
`"<0,1>,<1,2>,<2,3>"`

האבן <0,1> היא השמאלית ביותר, ואילו <2,3> היא הימנית ביותר.
אם הלוח ריק השיטה תחזיר מחרוזת ריקה.

משימה 3 - אובייקט "שחקן"

הגדירו מחלקה בשם `Player` שתייצג שחקן. שחקן מתואר ע"י שמו ואוסף אבני הדומינו שברשותו.
שני שחקים זהים זה לזה אם הם בעלי אותו השם ואותו סט אבנים (גם אם לא באותו הסדר).

Constructor

המחלקה `Player` צריכה להכיל (לפחות) את הבנאים הבאים:

```
public Player (String name, Tile[] tiles)
```

בנאי המקבל את שם השחקן, ומערך של אבני הדומינו איתן הוא מתחיל את המשחק.

```
public Player (String name)
```

בנאי המקבל את שם השחקן בלבד.

בשני המקרים ניתן להניח כי הקלט תקין – מערך האבנים ריק (מערך מגודל אפס) או מערך מאורך שאינו עולה על 28 ומכיל בכל תא אבני דומינו חוקיות. המחרוזת, המערך והתאים בו שונים מ-null.

assignTiles

הגדירו שיטה `assignTiles` שתאפשר השמת אבנים לשחקן:

```
public boolean assignTiles(Tile[] tiles)
```

השיטה מחליפה את סט האבנים הישן (אותו היא זורקת) בסט האבנים החדש ותחזיר true. במידה ולא ניתן לבצע את ההשמה מלאה של **כל האבנים** בגלל תקינות קלט השיטה תחזיר false **ולא תשנה דבר**.
קלט תקין הוא מערך חוקי שבכל אחד מהתאים שלו יש אבן דומינו חוקית. השמה של יד ריקה (מערך ריק מגודל אפס) היא פעולה חוקית - ז"א, השיטה יכולה לקבל מערך ריק ובסוף הריצה לשחקן לא יהיו אבנים ביד.

playMove

הגדירו שיטה `playMove` שתייצג מהלך של השחקן:

```
public boolean playMove(Board board)
```

השיטה מקבלת את לוח המשחק ומחזירה true במידה ובוצע מהלך כלשהוא.
במידה ולא נמצאה אף אבן מתאימה, השיטה תחזיר false. שימו לב כי לאחר ביצוע המהלך, **הלוח מתעדכן.**
במידה ולשחקן יש כמה אפשרויות, הנכם רשאים לבצע כל מהלך מתאים כראות עינכם.

כמובן שברגע ששחקן השתמש באבן מסוימת, היא יוצאת מהאוסף שלו.

countTiles

הגדירו שיטה `countTiles` שתשמש לחישוב נקודות השחקן בסוף המשחק:

```
public int countTiles()
```

השיטה תחזיר את סכום אבני הדומינו שברשות השחקן. סכום של אבן דומינו אחת הוא סכום הנקודות על שני ריבועי האבן. במידה ולשחקן יש יד ריקה השיטה תחזיר אפס.

hasMoreTiles

הגדירו שיטה `hasMoreTiles` שמחזירה true במידה ולשחקן יש עוד אבנים במלאי, ו-false אחרת.

```
public boolean hasMoreTiles()
```

toString

השיטה `toString` של מחלקה זו צריכה להחזיר מחרוזת המכילה את שם השחקן ואת ערכי האבנים שביד השחקן (אין חשיבות לסדר). לדוגמה, המחרוזת הבאה מתארת שחקן בשם "Happy Unicorn" שמחזיק שלוש אבנים:

```
"Happy Unicorn:[<0,6>,<6,2>,<2,3>]"
```

במידה ואין אבנים הסוגריים המרובעים יהיו ריקים.

משימה 4 - אובייקט "קבוצת שחקנים"

הגדירו מחלקה בשם `Team` שתייצג קבוצת שחקנים ומזוהה לפי שמה והשחקנים שלה.
יש חשיבות לסדר השחקנים בקבוצה (קבוצות עם אותו השם ואותם שחקנים אך בסדר שונה הן שונות).

Constructor

הבנאי של המחלקה `Team` צריך לתמוך באפשרות הבאה (לפחות):

```
public Team (String name, Player[] players)
```

בנאי מקבל את שם הקבוצה, ומערך של שחקני הקבוצה. ניתן להניח שהקלט תקין וכולל מערך חוקי מגודל אפס (קבוצה ללא שחקנים) ומעלה, שבכל תא יש שחקן חוקי והשם, המערך וכל התאים שלו שונים מ-null.

Getters

הגדירו שיטה `getPlayers` שתחזיר את שחקני הקבוצה במערך:

```
public Player[] getPlayers()
```

אם אין שחקנים בקבוצה השיטה תחזיר null.

הגדירו שיטה `getName` שתחזיר את שם הקבוצה:

```
public String getName()
```

playMove

הגדירו שיטה `playMove` שתייצג מהלך אחד של הקבוצה:

```
public boolean playMove(Board board)
```

השיטה מקבלת מצביע ללוח המשחק, מעדכנת אותו לאחר ביצוע מהלך ומחזירה true במידה ובוצע מהלך כלשהוא. במידה ולא נמצאה אף אבן מתאימה או לא ניתן לבצע מהלך מכל סיבה שהיא, השיטה תחזיר false.

החיפוש של אבן מתבצע באופן הבא:
השחקנים ינסו לבצע מהלך לפי סדר הופעתם בקבוצה. אם לשחקן אין אף אבן מתאימה, תורו עובר לשחקן הבא. אם לשחקן יש אבן מתאימה, החיפוש מסתיים וזוהי האבן שהשיטה תניח בלוח המשחק. במקרה ולאף שחקן מהקבוצה אין אבן מתאימה, השיטה תחזיר false.
שימו לב! סדר המשחק יהיה לפי מיקום המשחקן. קודם ישחק השחקן הראשון, לאחר מכן השחקן השני וכן הלאה.

countTiles

הגדירו שיטה `countTiles`:

```
public int countTiles()
```

השיטה תחזיר את סכום אבני הדומינו שברשות שחקני הקבוצה.
התוצאה תהיה סכום הניקוד של כל השחקנים (בדומה להגדרות בפונקציה הדומה במחלקת `Player`).

hasMoreTiles

הגדירו שיטה `hasMoreTiles` שמחזירה true במידה ולשחקני הקבוצה יש עוד אבנים במלאי, ו-false אחרת.

```
public boolean hasMoreTiles()
```

getNumberOfPlayers

הגדירו שיטה `getNumberOfPlayers` המחזירה את מספר השחקנים בקבוצה.

```
public int getNumberOfPlayers()
```

assignTilesToPlayers

הגדירו שיטה המאפשרת השמת אבני דומינו לכל אחד מהשחקנים בקבוצה. הפרמטר הוא מערך של מערכי אבנים (כל מערך מכיל את האבנים המיועדים לשחקן מסוים לפי סדר השחקנים).

```
public boolean assignTilesToPlayers(Tile[][] allHands)
```

אם לא ניתן לחלק את **כל האבנים לכל השחקנים** (למשל בגלל אי תקינות הקלט) השיטה מחזירה false **ולא מעדכנת את האבנים לאף אחד מהשחקנים**. אחרת, השיטה תעדכן את האבנים לכל השחקנים ותחזיר true.

קלט תקין הוא מערך שאורכו ככמות השחקנים ומכיל בכל תא מערך ריק (יד ריקה) או מערך שבכל תא שלו יש אבן חוקית.

toString

השיטה `toString` של מחלקה זו צריכה להחזיר מחרוזת המכילה את שם הקבוצה ואת פרטי השחקנים והאבנים שבידם לפי הסדר שלהם בקבוצה.

לדוגמה, המחרוזת הבאה מתארת את הקבוצה "The Unicorns" ושלושת חבריה (ללא ירידת שורה):

```
"Team:_The_Unicorns_{Happy_Unicorn:[<0,1>,<1,2>,<2,3>],
   Sad_Unicorn:[<4,1>],So_So_Unicorn:[<5,5>,<4,2>]}"
```

במידה ואין שחקנים בקבוצה יודפס:

```
"Team:_The_Unicorns_{No_players_in_the_team}”
```

המקף התחתון האדום מסמל רווח.

משימה 5 - אובייקט מנהל משחק

הגדירו מחלקה בשם `GameManager` שתייצג את מנהל המשחק.
מנהל המשחק אחראי על חלוקת האבנים לשחקנים, ועל ניהול המשחק.

מנהל המשחק מחזיק בידו את כל אבני המשחק – **28 במספרם,** כפי שנראה בתמונה 1.
שימו לב כי לא בהכרח כל האבנים משתתפות במשחק (ראו בהמשך).

מנהל משחק מאופיין על ידי שתי הקבוצות שמשתתפות במשחק.

Constructor

על המחלקה להכיל את שני הבנאים הבאים (לפחות):

```
public GameManager(Team team1, Team team2)
```

בנאי מקבל את שתי הקבוצות המשתתפות.
שימו לב כי בנאי זה מתאים למקרה בו השחקנים כבר קיבלו את אבני הדומינו (ניתן להניח זאת).

```
public GameManager(Team team1, Team team2, int tilesPerPlayer)
```

בנאי זה מתאים למקרה בו השחקנים עדיין לא קיבלו אבני דומינו או שיש להם אבנים שצריך להחליף.
במקרה זה, הבנאי מקבל את שתי הקבוצות המשתתפות במשחק ואת מספר אבני הדומינו אשר יחולקו לכל שחקן. הבנאי אחראי לחלק לכל שחקן מספר אבני דומינו כפי שמציין המשתנה `tilesPerPlayer`, בעזרת השיטה שמצוינת בסעיף הבא. משמע – לכל שחקן בכל אחת מהקבוצות **יהיו בדיוק `tilesPerPlayer` אבני דומינו** שחולקו לו באופן רנדומלי.
במידה ושחקן החזיק אבנים לפני הקריאה לבנאי, האבנים שהוא <u>החזיק יוחלפו באבנים שחולקו לו באופן רנדומלי</u>.

בשני המקרים ניתן להניח כי הקלט תקין, כלומר team1 ו-team2 קבוצות חוקיות שונות מ-null
ו-`tilesPerPlayer` מייצג מספר חוקי כפי שמתואר בהמשך.

dealTiles

הגדירו שיטה **פרטית** אשר מקבלת את מספר האבנים לכל שחקן, ומחלקת לכל אחד מהשחקנים משתי הקבוצות מספר זהה של אבנים:

```
private void dealTiles(int numberOfTiles)
```

כל שחקן מקרב הקבוצות, יקבל אותו מספר אבנים (**בדיוק `numberOfTiles` כל אחד**), אשר יוגרלו באופן **<u>אקראי</u>** מבין אבני המשחק. עליכם להשתמש בשיטה `Math.random()` בעת ביצוע החלוקה.

ניתן להניח שהקלט הוא חוקי, כלומר **(מספר השחקנים (*) מספר האבנים לכל שחקן) =< 28.**

toString

שיטת toString צריכה להדפיס את המידע על הקבוצות בפורמט לבחירתכם.
עם זאת, הפלט **חייב להכיל מידע שכולל את שמות הקבוצות והשחקנים ואילו אבנים יש בידיי הקבוצה.**

play

הגדירו שיטה play שתייצג משחק בין שתי הקבוצות:

```
public String play()
```

השיטה מריצה משחק דומינו בין הקבוצות ומחזירה מחרוזת המייצגת את מהלך המשחק.
שימו לב, כל שורה צריכה להסתיים בתו "\n" המסמל ירידת שורה. יש להשתמש אך ורק בתו הזה.

המשחק מתחיל כאשר הקבוצה הראשונה מניחה אבן כרצונה (בהתחלה לוח המשחק ריק).
מיד אחריה, הקבוצה השנייה מתחילה לשחק. כל קבוצה בתורה מבצעת מהלך עד שבתחילת אחד התורות לאחת הקבוצות יגמרו האבנים, או שלשתי הקבוצות לא תהיה אבן מתאימה ללוח (כלומר המשחק "נתקע").

לאחר כל מהלך, השיטה תדפיס את מצב הלוח המעודכן, ותציין האם הקבוצה המבצעת הצליחה להניח אבן או לא. אם לשתי הקבוצות לא היו אבנים אז קודם יודפסו המהלכים של שתי הקבוצות ורק אז המשחק יעצור.

לדוגמה, עבור 2 קבוצות אשר נקראות "The Majestic Pegasi" ו-"The Magical Unicorns" פלט אפשרי:

מהלך מוצלח יתואר ע"י המילה: **success**

```
"The_Magical_Unicorns,_success:_<0,1>,<1,2>,<2,3>\n"
```

מהלך לא מוצלח יתואר ע"י המילה **pass** (קבוצה לא הניחה אף אבן):

```
"The_Majestic_Pegasi,_pass:_<0,1>,<1,2>,<2,3>\n"
```

בצורה פורמלית יותר (ועם פחות יצורים מיתולוגיים):

```
GroupName + ",_" + ["success"|"pass"] + ":_" + board + "\n"
```

בסיום המשחק השיטה תשרשר למחרוזת את סכום האבנים עבור כל קבוצה, ותכריז על המנצח במשחק (הקבוצה עם סכום האבנים הנמוך יותר).

```
"The_Majestic_Pegasi,_score:_54\n" +
"The_Magical_Unicorns,_score:_13\n" +
"The_Magical_Unicorns_wins\n"
```

במידה ואין מנצח תשורשר במקום השורה האחרונה המחרוזת: "Draw!_-_the_house_wins\n"

כלומר (בצורה פורמלית):

```
First_Group_Name + ",_score:_" + score + "\n" +
Second_Group_Name + ",_score:_" + score + "\n" +
Winning_Group_Name + "_wins\n"
```

בהצלחה