תאריך הבחינה: 16.2.2012

שמות המרצים: ד"ר טל גרינשפון
פרופ' משה זיפר
ד"ר פז כרמי
מר עדי סוויסה
פרופ' מייק קודיש
ד"ר חן קיסר
ד"ר צחי רוזן

שם הקורס: מבוא למדעי המחשב
מבוא לתכנות למערכות מידע

מספר הקורס: 202-1-1011, 202-1-1041

שנה: 2012 סמסטר: א' מועד: א

משך הבחינה: **3 שעות**

חומר עזר: **אסור**

<u>מועד א</u>

אנא קיראו היטב את ההוראות שלהלן:

- במבחן זה 5 שאלות. ענו על כל השאלות.
- בשאלות בהן לא צוין אחרת, ניתן לבחור בפתרון רקורסיבי או פתרון שאינו רקורסיבי, לבחירתכם.
- רשמו את תשובותיכם **בדפי התשובות בלבד**. המחברת שקיבלתם היא מחברת טיוטה והיא לא תימסר כלל לבדיקה. בסיום הבחינה נשמור אך ורק את דף התשובות. כל שאר החומר יועבר לגריסה.
- **שימו לב:** החשיבות העליונה היא **לנכונות** הקוד. מאידך, **<u>יעילות</u> ,סגנון וכתיבה ברורה** חשובים גם הם, ולכן תשובה יעילה ומסוגננת תזכה בציון גבוה יותר.
- בשאלות התכנות, מספר השורות העומדות לרשותכם בדף התשובות רומז על אורך הקוד הנדרש. הקפידו על כתב יד ברור. **תשובות מסורבלות או ארוכות מדי לא יזכו בניקוד מלא**. כיתבו פתרון לשאלה קודם במחברת הטיוטה ורק לאחר מכן העתיקו פתרון נקי לדף התשובות.
- אין להוסיף פונקציות עזר אלא אם נאמר במפורש שמותר.
- בדף האחרון של המבחן ישנה תזכורת לכמה פונקציות חשובות, שבהן תוכלו להשתמש במהלך המבחן.
- **הקפידו לרשום בכל אחד מדפי התשובות גם את מספר הנבחן ומספר החדר שבו אתם נבחנים.**
- **במידה ואינכם יודעים את התשובה לסעיף כלשהו, רשמו "לא יודע/ת" (במקום תשובה) ותזכו ב- 20% מניקוד הסעיף (מעוגל מטה). אם רשום "לא יודע/ת", ההתייחסות היא לכל הסעיף.**

בהצלחה !

<u>שאלה 1 (20=12+8 נקודות)</u>

<u>סעיף א (8 נק):</u> פגשנו את מושג הסדר המילוני ("הלקסיקוגרפי") בתרגיל בית מספר 3. השלימו בדף התשובות את הגדרת הפונקציה

public static boolean lexLT(String s1, String s2)

אשר מקבלת שתי מחרוזות ומחזירה ערך true אם ורק אם s1 קודמת ל s2 בסדר המילוני.
לדוגמא:

המחרוזת "abc" קודמת למחרוזת "quarsbecy" אך לא למחרוזת "aabc".
המחרוזת "abbc" קודמת למחרוזת "abbec" אך לא למחרוזת "abbac".
המחרוזת "ab" קודמת למחרוזת "abbec".

שום מחרוזת אינה קודמת לעצמה. המחרוזת הריקה קודמת לכל מחרוזת שאינה ריקה.
ניתן להניח כי המחרוזות s1 ו- s2 אינן null ומכילות אותיות קטנות באנגלית (a,...,z) בלבד.

<u>אסור להשתמש</u> ב- compareTo או ב- compare.

<u>סעיף ב (12 נק):</u> תזכורת – הפונקציה הרקורסיבית לחישוב האיבר ה-n (עבור n לא שלילי) בסידרת פיבונצ'י הינה:

```
public static int fibRec(int n) {
        int answer;
        if (n == 0)
                answer = 0;
        else if (n == 1)
                answer = 1;
        else
                answer =  fibRec(n-1) + fibRec(n-2);
        return answer;
}
```

כיתבו פונקציה יעילה יותר (לאו דווקא רקורסיבית) לחישוב האיבר ה-n בסידרת פיבונצ'י.

public static int fibIter(int n)

מותר להוסיף משתני עזר מסוג int בלבד (בפרט, אין להשתמש במערכים). זיכרו כי פתרון יותר יעיל מזכה ביותר נקודות. אין להוסיף פונקציות עזר.

<u>שאלה 2 (20 נקודות)</u>

נניח כי המערך int[] coins מציין את ערכי המטבעות במדינה מסוימת. למשל, במדינה מסוימת יש מטבעות בערכים הבאים int[] coins = {10, 25, 5, 1}. נשאלת השאלה: בכמה אופנים ניתן לפרוט סכום מסוים. למשל, במדינת הדוגמה ניתן לפרוט סכום של 10 בארבעה אופנים שונים. להלן ארבעת האופנים שבהם ניתן לפרוט סכום של 10 בעזרת המטבעות הנתונים:

```
1. {10}
2. {5,5}
3. {5,1,1,1,1,1}
4. {1,1,1,1,1,1,1,1,1,1}
```

כיתבו פונקציה **<u>רקורסיבית</u>**

public static int change(int sum, int[] coins)

אשר מקבלת ערך sum ומערך של סוגי מטבע coins ומחזירה את מספר האופנים שבהם ניתן לפרוט את sum בעזרת מטבעות מהסוגים הנתונים ב-coins. ניתן להניח כי coins אינו null ומכיל רק מספרים חיוביים השונים זה מזה.

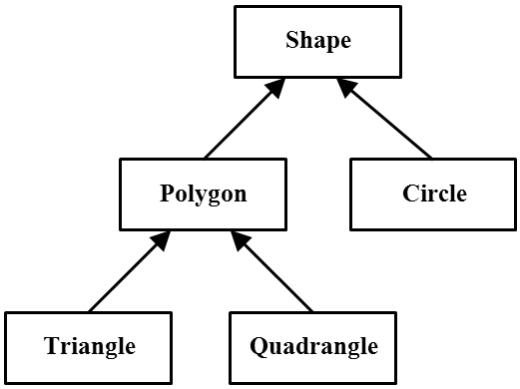
שימו לב: אם sum<0 אזי לא ניתן לפרוט והערך המוחזר צריך להיות 0. אם sum=0 אזי ניתן לפרוט באופן יחיד והערך המוחזר צריך להיות 1.

שימו לב: {5,1,1,1,1,1} ו- {1,1,1,1,1,5} הן אותה פריטה ואין לספור אותן פעמיים.

רמז: בשאלה זו מותר ומומלץ להגדיר פונקצית עזר. למשל, ניתן להוסיף פרמטר int from המציין שמותר להשתמש רק במטבעות מהסוג שהחל מהאינדקס from במערך coins.

<u>שאלה 3 (12+5+8=25 נקודות)</u>

שאלה זו היא שאלת תכנון המבוססת על תרגיל בית מספר 5. ההבדל העיקרי מהתרגיל הוא שבגרסה שתכתבו כאן לכל צורה יש מאפיין מהטיפוס "צבע" (Color). אנו ממליצים מאוד לקרוא את השאלה עד הסוף לפני שאתם מתחילים לפתור אותה. לצורך פתרון השאלה עומדות לרשותכם שתי מחלקות Color ו-Point . השיטות של שתי המחלקות האלו מופיעות בתחתית עמוד 5.



הציור מציג את היררכית "רכיבי התוכנה" בשאלה זו. כל מלבן מציג רכיב תוכנה: ממשק, מחלקה אבסטרקטית או מחלקה ממשית (לא אבסטרקטית). חץ מייצג מימוש של ממשק או הרחבה של מחלקה.

בטבלאות בעמוד 5 רשומות השיטות המשותפות לכל הצורות (Shape), והשיטות הייחודיות של Circle, Triangle ו Quadrangle (מרובע). כמו כן עומדים שם לרשותכם הבנאים והשיטות של המחלקות Color ו-Point.

בסעיפים א' ב' ו- ג' בדף התשובות עליכם להשלים את הקוד של שלושת רכיבי התוכנה: Shape, Polygon & Circle.

הנחיות כלליות:

1. עליכם להחליט, עבור כל שיטה באיזה רכיב תוכנה לממש אותה כדי לקבל קוד תמציתי (ללא שכפולים מיותרים) וקריא.
2. שיטות כלליות לחישוב שטח של פוליגון ולקביעה האם נקודה מוכלת בתוכו, הן מורכבות ולא מתאימות למבחן. פתרון שיעקוף את הצורך במימושן יחשב כאן טוב.
3. אין לכתוב בנאים ללא פרמטרים. בכל רכיב תוכנה עליכם להחליט אם צריך בנאי בעל פרמטרים. במקרה כזה, אחד הפרמטרים (יכול להיות היחיד) יהיה מטיפוס Color.
4. בכל רכיב תוכנה עליכם להחליט אם צריך שדות ואם כן מה רמת הנראות שלהם (public, protected או private).
5. אין להוסיף במחלקה שיטה ציבורית שלא הוגדרה בטבלאות ואין גם לדרוס את השיטות equals ו toString .
6. אין לממש את רכיבי התוכנה Triangle ו- Quadrangle . שיטות ציבוריות שלהן, שאין דרך לממשן ברכיבים Shape או Polygon יישארו בלתי ממומשות. עם זאת התשובה שלכם צריכה לקחת אותם בחשבון.
7. אין צורך לבדוק תקינות פרמטרים ולזרוק חריגות. ניתן להניח שכל הפרמטרים הנשלחים לבנאים ולשיטות הם תקינים.
8. בכל מקום בו מועברים אובייקטים אל ומחוץ למחלקות נדרשת העתקה עמוקה. כלומר, יצירה של אובייקטים חדשים והעתקה של כל השדות.
9. להזכירכם שטחו של מעגל בעל רדיוס R הוא πR^2 והיקפו $2\pi R$.

שיטות ציבוריות משותפות לכל הצורות (Shape)	
שיטה	חתימה
מחזירה את היקף הצורה	`double getPerimeter()`
מחזירה את שטח הצורה	`double getArea()`
מחזירה את צבע הצורה	`Color getColor()`
הזזה של הצורה כך שכל נקודה בה תזוז בהתאם לערכי x ו- y של הנקודה שהתקבלה כפרמטר.	`void move(Point p)`
מחזירה true אם ורק אם הנקודה p מוכלת ממש בצורה. אם הנקודה נמצאת על שפת הצורה, השיטה מחזירה false	`boolean contains(Point p)`

שיטות ציבוריות יחודיות למחלקה Circle	
שיטה	חתימה
מחזירה את רדיוס המעגל	`double getRadius()`
מחזירה את מרכז המעגל	`Point getCenter()`

שיטות ציבוריות יחודיות למחלקה Triangle (לא למימוש)		
שיטות המחזירות את נקודות המשולש	`Point getP1()`	`Point getP2()`
	`Point getP3()`	

שיטות ציבוריות יחודיות למחלקה Quadrangle (לא למימוש)		
שיטות המחזירות את נקודות המרובע	`Point getP1()`	`Point getP2()`
	`Point getP3()`	`Point getP4()`

Point	
שיטה (לא למימוש)	חתימה
בנאי	`public Point(double x, double y)`
בנאי העתקה	`public Point(Point other)`
מחזירה את קואורדינטת ה x	`double getX()`
מחזירה את קואורדינטת ה y	`double getY()`
מוסיף את הערכים לנקודה הנוכחית	`void move(int x, int y)`
מוסיף את ערכי x ו- y של p לנקודה הנוכחית	`void move(Point p)`
השוואה לוגית בין נקודות	`boolean equals(Object other)`
חישוב המרחק בין שתי נקודות	`double distance(Point p)`

Color	
שיטה (לא למימוש)	חתימה
בנאי שמקבל מספר שלם המייצג צבע (אופן הייצוג אינו רלוואנטי לשאלה)	`public Color(int rgb)`
בנאי מעתיק	`public Color(Color copyMe)`
מחזיר מספר שלם המייצג צבע (אופן הייצוג אינו רלוואנטי לשאלה)	`public int getRGB()`

<u>שאלה 4 (7+8=15 נקודות)</u>

תזכורת: תור Queue הוא מבנה נתונים מופשט המיוצג על ידי הממשק

```
public interface Queue {
        public void enqueue(Object o);
        public Object dequeue();
        public boolean isEmpty();
}
```

אופן פעולת השיטות enqueue, dequeue, ו-isEmpty הינו בהתאם להגדרת תור, כפי שנלמד בקורס.

<u>סעיף א. (7 נקודות)</u>

נתונה המחלקה CircularLinkedList המיצגת רשימה מקושרת מעגלית, כלומר רשימה שבה השדה next של החוליה האחרונה מצביע על החוליה הראשונה. השיטות של מחלקה זו מפורטות בטבלה למטה. העזרו בה כדי להשלים את המחלקה QueueAsCircularList

שימו לב: עליכם להשתמש בדיוק בשתיים מבין ארבע השיטות 4,5,6,7 של CircularLinkedList בטבלה.

```
public class QueueAsCircularList implements Queue {

  private CircularLinkedList cl;
  public QueueAsCircularList () {
      cl = new CircularLinkedList ();
  }

  public boolean isEmpty() {
            //אין צורך לממש
  }

  public void enqueue(Object o) {
        // יש להשלים בדף התשובות
  }

  public Object dequeue(){
        // יש להשלים בדף התשובות
  }
}// QueueAsCircularList
```

CircularLinkedList		
	חתימה	**שיטה**
1	CircularLinkedList()	בנאי
2	int size()	מחזירה את מספר האיברים ברשימה
3	Boolean isEmpty()	מחזירה true אם הרשימה ריקה, אחרת מחזירה false
4	void addFirst(Object data)	מוסיפה איבר בראש הרשימה. האיבר החדש יהיה הראשון
5	void addLast(Object data)	מוסיפה איבר בסוף הרשימה. האיבר החדש יהיה האחרון
6	Object removeFirst()	מסיר איבר מראש הרשימה ומחזיר אותו. האיבר שהיה שני (אם היה כזה) יהיה עכשיו ראשון
7	Object removeLast ()	מסיר איבר מסוף הרשימה ומחזיר אותו. האיבר שהיה לפני אחרון (אם היה כזה) יהיה עכשיו אחרון.

סעיף ב. (8 נקודות)

להלן קוד חלקי של המחלקה CircularLinkedList המייצגת רשימה מקושרת מעגלית, כלומר רשימה שבה השדה next של החוליה האחרונה מצביע על החוליה הראשונה. בתשובתכם לסעיף א' נדרשתם להשתמש בדיוק בשתים מתוך ארבע שיטות ההוספה וההסרה של המחלקה (4-7 בטבלה). עליכם להשלים את הקוד של שתי השיטות האלו.

הנחיות:

1. המחלקה CircularLinkedList משתמשת בחוליות (עצמים מהמחלקה Link הנתונה) המקושרות במעגל. ברשימה כזאת אין אף חוליה ששדה ה-next שלה הוא null. המחלקה Link היא מחלקה פנימית והקוד שלה מופיע בעמוד 8.
2. במחלקה מוגדר רק שדה אחד (pointer) . אסור לכם להוסיף שדה נוסף למחלקה.
3. כדי להימנע מהסתבכות מיותרת עם מקרי קצה, עליכם לטפל רק במקרה של רשימות שיש בהן לפחות 2 איברים.
4. אם בסעיף א' כתבתם "לא יודע" , אזי כעת בחרו שתי שיטות מתוך הארבע (4-7) והשלימו אותן.

```
public class CircularLinkedList {
    private Link pointer; // מצביע אחד לשימוש כרצונכם

    public CircularLinkedList() {pointer = null; } // בונה היוצר רשימה ריקה

    public int size() { אין לממש}      // מחזיר את מספר האיברים ברשימה

    public boolean isEmpty(){ אין לממש}  // מחזיר "אמת" אם הרשימה ריקה

    public void addFirst(Object data) { // מוסיף איבר בראש הרשימה
            if (size() < 2) {אין צורך לממש }
            else {
                    השלימו בדף התשובות אם השתמשתם בשיטה זו בסעיף א' //
                    }
    }

    public void addLast(Object data) { // מוסיף איבר בסוף הרשימה
            if (size() < 2) {אין צורך לממש }
            else {
                    השלימו בדף התשובות אם השתמשתם בשיטה זו בסעיף א' //
                    }
    }

    public Object removeFirst(){ // מסיר איבר מראש הרשימה ומחזיר אותו
            if (size() < 2) {אין צורך לממש }
            else {
                    השלימו בדף התשובות אם השתמשתם בשיטה זו בסעיף א' //
                    }
    }

    public Object removeLast(){ // מסיר איבר מסוף הרשימה ומחזיר אותו
            if (size() < 2) {אין צורך לממש }
            else {// השלימו בדף התשובות אם השתמשתם בשיטה זו בסעיף א' }
    }
```

המשך (המחלקה הפנימית Link) בעמוד הבא

```
        private static class Link {
                public Object data;
                public Link next;

                public Link(Object data, Link next) {
                        this.data = data;
                        this.next = next;
                }

                public Link (Object data) {
                        this.data = data;
                        this.next = null;
                }

          } // class Link

} // class CircularLinkedList
```

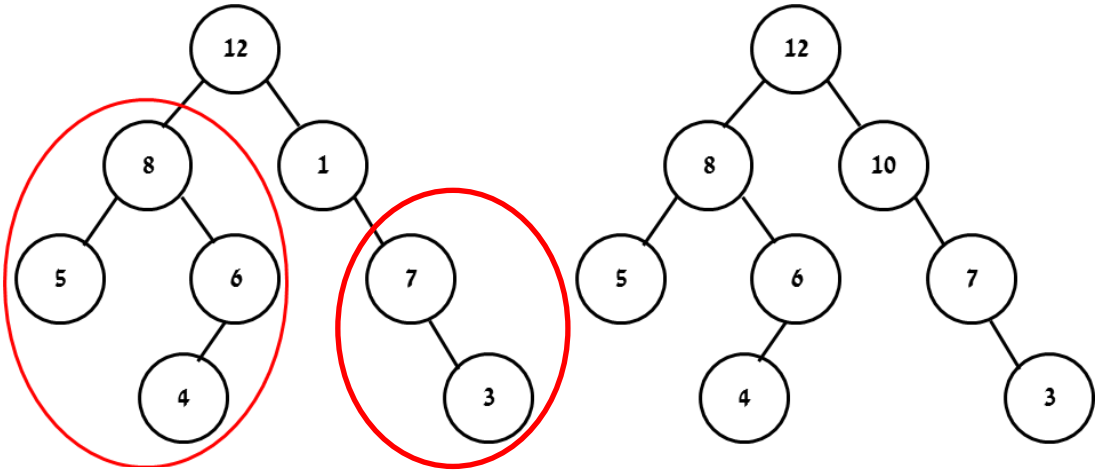
שאלה 5 (20 = 8+7+5 נקודות)

בשאלה זו נתייחס לעץ בינארי שבקודקודיו יש ערכים שלמים (int). ניתן להניח שכל הערכים בעץ שונים זה מזה וחיוביים.

הגדרה: עץ בינארי הוא *ערימה* אם הערך בכל צומת גדול ממש מכל שאר האיברים בתת העץ המושרש בו (כלומר ערך השורש גדול מערכי כל האיברים בתתי העצים השמאלי והימני).

הגדרה: ערימה בעץ היא תת-עץ שהוא ערימה.

דוגמא:



העץ הימני הוא ערימה. העץ השמאלי אינו ערימה שכן תת העץ שבשרשו הערך 1 מכיל ערכים הגדולים ממנו: 7 ו- 3 (לעומת זאת, תתי העצים המוקפים בעיגולים הם ערימות בנות 4 ו- 2 איברים).

בהמשך נתונות הגדרות חלקיות של המחלקות IntBinaryTree ו-IntBinaryNode:

סעיף א (5 נק): השלימו בשתי המחלקות IntBinaryTree ו-IntBinaryNode את השיטה

```
public int sum()
```

אשר מחזירה את סכום האיברים בעץ. במחלקה IntBinaryNode יחזיר קודקוד את סכום האיברים בעץ המושרש בו.

לדוגמה: סכום האיברים בעץ הימני

12+8+10+5+6+7+4+3 = 55

שימו לב: סכום האיברים בעץ ריק הוא אפס.

סעיף ב (7 נק): השלימו בשתי המחלקות IntBinaryTree ו-IntBinaryNode את השיטה

```
public boolean isHeap()
```

אשר מחזירה true אם ורק אם העץ הוא ערימה. במחלקה IntBinaryNode יחזיר קודקוד true אם העץ המושרש בו הוא ערימה.

<u>סעיף ג (8 נק):</u> השלימו בשתי המחלקות IntBinaryTree ו-IntBinaryNode את השיטה

public int maxHeapSum()

אשר מוצאת את הערימה בעץ שסכום איבריה הוא הגדול ביותר ומחזירה סכום זה.
למשל בדוגמא למעלה, עבור העץ הימני השיטה תחזיר 55 (סכום כל איברי העץ) ועבור העץ השמאלי השיטה תחזיר 23 (סכום כל איברים בערמה המסומנת בעיגול השמאלי).

כמו בסעיפים הקודמים, המימוש ב IntBinaryNode מתייחס לעץ המושרש בקודקוד.

הערה: להזכירכם ניתן להניח שכל המספרים בעץ חיוביים.

```
public class IntBinaryTree {
        private IntBinaryNode root;

        public IntBinaryTree(){  root = null; }

        public int sum() {
        // שיטה המחזירה את סכום המספרים בקדקודי העץ
                השלימו בדף התשובות
        }

        public boolean isHeap() {
  //שיטה המחזירה true אם ורק אם העץ הוא ערימה
                השלימו בדף התשובות
        }

        public int maxHeapSum {
     //שיטה המחזירה את סכום האיברים המקסימלי בערמה שבעץ
                השלימו בדף התשובות
        }

}// IntBinaryTree

public class IntBinaryNode {
        private int data;
        private IntBinaryNode left;
        private IntBinaryNode right;

        public IntBinaryNode(int data) {
                this.data = data;
                left = null;
                right = null;
        }

        public int getData() {
                return data;
        }
```

המשך בעמוד הבא

```
        public int sum() {
                שיטה המחזירה את סכום המספרים בקדקודי העץ המושרש בקודקוד זה//
                השלימו בדף התשובות//

        }

        public boolean isHeap() {
               שיטה המחזירה true אם ורק אם העץ המושרש בקודקוד זה הוא ערימה //
               השלימו בדף התשובות  //
        }

        public int maxHeapSum {
          שיטה המחזירה את סכום האיברים המקסימלי בערמה שבעץ המושרש בקודקוד זה//
          השלימו בדף התשובות//
        }
} // IntBinaryNode
```

<u>תזכורת:</u>

- `int` **`length`**`()`

  שיטה במחלקה String המחזירה את אורך המחרוזת.

- `int` **`indexOf`**`(char c)`

  שיטה במחלקה String, המחזירה את האינדקס הראשון במחרוזת, בו מופיע התו c, או 1- במידה והתו כלל לא מופיע במחרוזת.

- `char` **`charAt`**`(int index)`

  שיטה במחלקה String, המחזירה את התו במיקום index במחרוזת.

- `String` **`substring`**`(int beginIndex)`

  שיטה במחלקה String המחזירה מחרוזת חדשה, שהיא תת-מחרוזת של עצם המפתח החל מהתו beginIndex ועד סוף המחרוזת.

- `String` **`substring`**`(int beginIndex, int endIndex)`

  שיטה במחלקה String המחזירה מחרוזת חדשה, שהיא תת-מחרוזת של עצם המפתח בין התווים beginIndex ו-endIndex, (כולל התו ה-beginIndex ולא כולל התו ה-endIndex).

- `double` **`Math.pow`**`(double a, double b)`

  פונקציה המחזירה את הערך של a בחזקת b.

- `double` **`Math.sqrt`**`(double a)`

  פונקציה המחזירה את השורש הריבועי החיובי של a.

- `double` **`Math.abs`**`(double a)`

  פונקציה המחזירה את ערכו המוחלט של a.

- `int` **`Math.abs`**`(int a)`

  פונקציה המחזירה את ערכו המוחלט של a.

- `int` **`Math.min`**`(int a, int b)`

  פונקציה המחזירה את הערך המינימלי מבין a ו- b.

- `double` **`Math.min`**`(double a, double b)`

  פונקציה המחזירה את הערך המינימלי מבין a ו- b.

- `int` **`Math.max`**`(int a, int b)`

  פונקציה המחזירה את הערך המקסימלי מבין a ו- b.

- `double` **`Math.max`**`(double a, double b)`

  פונקציה המחזירה את הערך המקסימלי מבין a ו- b.

- `double` **`Math.random`**`()`

  פונקציה המחזירה ערך אקראי בטווח 0 (כולל) עד 1 (לא כולל).

- `double` **`Math.PI`**

  שדה המכיל את הקבוע המתמטי π.