שיעורי בית Threads

מטרת התרגיל - ליצור 10 Threads שמפריחים בלון בצבע רנדומלי.

לכל בלון יש מספר וצבע רנדומלי.

כדי להגריל צבע כדאי להגדיר מערך של צבעים:

```
String[] colors = {"Red", "Orange", "Green", "Blue", "Yellow"};
```

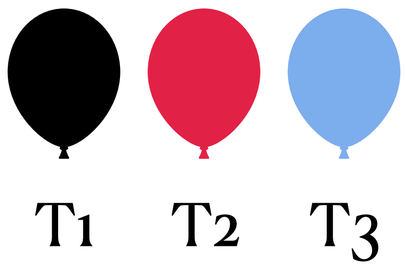
ולהגריל מספר בין 0 לכמות האיברים במערך.

נסו להפעיל את כל הבלונים יחד ולראות מה תהיה ההדפסה.

<u>מאוד דומה לתרגיל שמדפיס * ו#</u>

כדי להריץ 10 פעמים Thread כדאי להשתמש בלולאת for שרצה 10 פעמים.

פתרון לשיעורי בית Threads



פתרון לשיעורי בית Threads

```
String[] colors = {"Red", "Orange", "Green", "Blue", "Yellow"};
for (int i = 0; i < 10; i++) {
    //פעמים את אותה המשימה 10
    int balloonNumber = i + 1;
    //רק מייצר תהליכון ומפעיל אותו.
    new Thread(/*run()*/() -> {
        //נגריל מס' רנדומלי בין 0 ל-4 (גבולות המערך של הצבעים)
        int rand = IO.getRand(colors.length);

        //random Color:
        String color = colors[rand];

        //the body of the run method:
        System.out.println(color + " Balloon " + balloonNumber);

    }).start();
}
```

מה הסיבה שלא הצלחנו להשתמש בi?

פתרון לשיעורי בית Threads

```java
public class Main {

    public static void main(String[] args) {
        //המחלקה Thread מקבלת אובייקט כפרמטר
        //אנחנו כותבים פונקציה run שמחלקה אחרת תריץ אותה

        for (int i = 0; i < 10; i++) {
            Thread t1 = new Thread(new MyRunnable(i));
            t1.start();
        }

    }
}


//.יש להגריל 5 מספרים בלי כפילויות
```

Talking: Hackeru Class 17

```java
//ממשק שמגדיר Unit of work - משימה שצריכה להתבצע בתהליכון//
public class MyRunnable implements Runnable {

    //properties:
    private int number;

    //constructor:
    public MyRunnable(int number) { this.number = number; }

    @Override
    public void run() {
        String[] colors = {"Red", "Orange", "Green", "Blue", "Yellow"};
        // קוד של המשימה שתרוץ בתהליכון

        //:נגריל מספר בין 0 ל4 גבולות המערך
        int rand = IO.getRand(colors.length);
        //נשתמש במספר הרנדומלי שהגרלנו כדי למצוא צבע במערך
        String color = colors[rand];

        System.out.println(color + " balloon " + number);
    }
}
```

פתרון טוב.

חסרונות:

א) מחולק ל-2 קבצים.

ב) יותר קוד מלמבדה.

פתרון טוב.

יתרונות:

יותר מסודר.

למעשה - מדובר באותו פתרון.

אלא שJava8 מאוד מקלה בעזרת קיצורי-למבדה.

פתרון לשיעורי בית Threads

```
String[] colors = {"Red", "Orange", "Green", "Blue", "Yellow"};
for (int i = 0; i < 10; i++) {
    //פעמים את אותה המשימה 10
    int balloonNumber = i + 1;
    //רק מייצר תהליכון ומפעיל אותו.
    new Thread(/*run()*/() -> {
        //נגריל מס' רנדומלי בין 0 ל-4 (גבולות המערך של הצבעים)
        int rand = IO.getRand(colors.length);

        //random Color:
        String color = colors[rand];

        //the body of the run method:
        System.out.println(color + " Balloon " + balloonNumber);

    }).start();
}
```

למבדה הוא פשוט קיצור ל"יצירת מחלקה".

Java מעתיקה את הערך של i לתוך המחלקה שהיא יוצרת עבורנו ולכן דורשת ש i לא ישתנה.

תרגיל:

א)

הגדירו מחלקה עבור קלף משחק:

תכונות:

private String suit ❤️

private String rank 10

צרו מחלקה. כולל בנאי שמקבל את 2 התכונות. ופעולת toString.

ב) צרו מופע של קלף - 10 לב.

ב*) הגרילו קלף רנדומלי.

ג)צרו מחלקה עבור חפיסת קלפים.

בכל חפיסה רשימת-קלפים.

יש למלא את החפיסה בכל הקלפים האפשריים. ליצור מופע בMain ולבדוק.

*

תרגיל:

Deck.java | Card.java

```java
package edu.tomerbu;

public class Card {

    //שייך למחלקה ולא למופע בדד
    static String[] suits = {"❤️", "♠️", "♣️", "♦️"};

    //properties:
    private String rank;
    private String suit;

    //constructor:
    public Card( String rank, String suit) {
        this.rank = rank;
        this.suit = suit;
    }

    @Override
    public String toString() {
        //Ace of ♠️
```

```java
public static void main(String[] args) {


    //.ב סעיף
    Card c1 = new Card( rank: "10",  suit: "❤️");


    //- קלף רנדומלי ב*


    String suit = Card.suits[IO.getRand(Card.suits.length)];
    String[] ranks = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A"};
    String rank = ranks[IO.getRand(ranks.length)];

    Card c = new Card(rank, suit);
    System.out.println(c);


}
```

Talking: משה עמר

Deck.java

```java
package edu.tomerbu;

import java.util.ArrayList;

public class Deck {
    //properties:
    private ArrayList<Card> cards = new Ar

    public Deck() {
        //.מלאו את החפיסה בכל הקלפים האפשריים
        //.צרו מופע של החפיסה והדפיסו אותו
        //.נשים לב לא להשתמש ברנדום עכשיו
    }

    @Override
    public String toString() {...}
}
```

From Me to Everyo
tring[] suits = {"❤️
String[] suits = {"❤️

From ben bokobza
windows 10 לעשות
Win + ;
כן

From ben bokobza
Deck

From Nadav Avno
ArrayList of Cards
אבל יהיה כפילויות
אהההה נכון

To: Everyone
Type message he

You are screen sharing | Stop Share

פתרון:

```java
public class Main {

    public static void main(String[] args) {
        Deck d = new Deck();
        System.out.println(d);
    }

    static void demo() {...}
}


//.יש להגריל 5 מספרים בלי כפילויות
```

```java
5   public class Deck {
6       //properties:
7       private ArrayList<Card> cards = new ArrayList<>();
8
9       public Deck() {
10          //.מלאו את החפיסה בכל הקלפים האפשריים
11          String[] suits = {"❤️", "♠️", "♣️", "♦️"};
12          String[] ranks = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K
13
14          //:הכי קצר
15          //for each rank:
16          for (String rank : ranks) {
17              //for each suit (with a rank - 3)
18              for (String suit : suits) {
19                  cards.add(new Card(rank, suit));
20              }
21          }
22  //...
39      }
```

מתי נרצה להשתמש בThreads?

באפליקציות עם ממשק משתמש - כל פעולה שלוקחת יותר מ-100 מילישניות - נרצה ליצור עבורה תהליכון.

המשתמש עלול ללחוץ על כפתור - ואם האפליקציה תהיה "עסוקה" באותו רגע - הכפתור לא יגיב-> ביקורות שליליות. ANR.

דוגמאות חובה לThreads

גישה לאינטרנט גם לאתר בודד - ובמיוחד לכמה אתרים במקביל.

גישה לקבצים.

גישה לדטה-בייס.

לולאות שרצות 100,000 פעמים ומעלה.

מתי לא להשתמש בThreads?

לא נרצה לגשת למשאב משותף ממספר Threads יחד:

לא נרצה לכתוב לקובץ אחד מ-4 Threads בו זמנית. (לא נדע איזה טקסט יכתב ראשון ואיזה שני ואיזה שלישי).

לא נרצה לשתף Counter בין Threads.

לא נרצה ליצור תחרות על משאב בין הThreads.

Thread safety ?

תנאי מירוץ: לא נדע מה יודפס:

מספר תהליכונים רצים יחד - ומתחרים אחד עם השני.

```
0
3
4
2
1
6
7
5
8
9
```

```
//Thread safe version:
public class SheepHerd {
    private int counter;

    //איך נוכל להחזיר תמיד את אותה תוצאה.
    //מסונכרן. נרצה ליצור תור במתודה.
   public void incrementAndPrint(){//10 threads stand
        System.out.println(counter++);
    }

}
```

```
public static void main(String[] args) {

    SheepHerd herd = new SheepHerd();

    for (int i = 0; i < 10; i++) {
        new Thread(() -> {
            herd.incrementAndPrint();
        }).start();
    }
    //10 threads -> loop 10 times.
    //each thread calls the "incrementAndPrint" method.
}
```

תנאי מירוץ - איך נפתור?

***לא להכנס מלכתחילה ***

לא לתת משימה ל-10 Threads לעשות את אותה עבודה.

-אם כבר נתנו ל-10 Threads לעשות את אותה פעולה - צריך לסנכרן בינהם!

```
//Thread safe version:
public class SheepHerd {
    private int counter;

    //thread safe version?
    //איך נוכל להחזיר תמיד את אותה תוצאה.
    //מסונכרן. נרצה ליצור תור במתודה.
    //נרצה להדפיס לפי הסדר 1, 2, 3, 4, 5, 6
    public synchronized void incrementAndPrint(){//10 threads stand
        System.out.println(counter++);
    }
}
```

- synchronized method

לפעולה מסונכרנת - יכול להכנס רק תהליכון אחד בכל פעם.

אם שלחנו 10 תהליכונים לפעולה מסונכרנת - 9 יחכו בתור בזמן שאחד נכנס.

אחרי שהאחד סיים ייכנס תהליכון אחד במקומו ו-8 יחכו בכניסה.

תנאי מירוץ - איך נפתור?

במקום לסנכרן מתודה שלמה - לעיתים נרצה לסנכרן רק בלוק קטן של קוד בתוך מתודה.

במקרה כזה - נגדיר אובייקט שתפקידו מנעול:

```
private final Object LOCK = new Object();
```

```
public void doWork(){
    System.out.println("...");
    System.out.println();
    System.out.println();
    System.out.println();
    System.out.println();

    //כאן התור מתחיל:
    //synchronized block:
    //מוטיבציה: לא נרצה לנעול את כל המתודה - רק חלק ממנה
    synchronized (LOCK) {//1) aquire the lock
        //בעייתי עבור Threads:
        System.out.println(counter++);
    }//2: return the lock
    //כאן התור מסתיים:

    for (int i = 0; i < 10; i++) {
        System.out.println();
    }
}
```

- synchronized block

בלוק קצר יותר בתוך מתודה - שינעל כך שרק תהליכון אחד יוכל להכנס לאיזור הנעול.

בכניסה לבלוק - תהליכון "יקבל מנעול"
ביציאה מהבלוק - תהליכון "יחזיר את המנעול" ובכך יאפשר גישה לתהליכונים אחרים.