ריבוי נימים (multithreading) בשפת Java

חלק ראשון

1. מבוא - נימים

1.1. מהו Thread (נים)?

Thread הוא למעשה מתודה שיכולה לרוץ במקביל לנימים אחרים. את האנלוגיה למתודה נוכל לראות בתרגול זה ובתרגולים הבאים:

- כמו מתודה גם נים בהיבט הטכני יכול לקבל פרמטרים להחזיר ערכים, ניתן אפילו לכתוב נים שמתנהג בצורה רקורסיבית. (נראה דוגמא בהמשך).
- גם לנים אם מוגדר בצורה טובה יש משימה אחת לבצוע המגדירה אותו.

1.2. למה צריך נימים?

2 דוגמאות לצרכים:

- אם שרת שמקבל הרבה בקשות בו זמנית לא היה משתמש בנימים אז השימוש באתר היה יכול להיות מעיק ביותר. היינו צריכים לחכות זמן רב עד אשר השרת יסיים לטפל בבקשה קודמת.
- עבור חישובים גדולים, במידה ויש לנו מחשב עם הרבה מעבדים ניתן לחלק את העבודה בין המעבדים.

2. נימים בשפת JAVA

2.1. הגדרת של נים באמצעות ירושה ממחלקת Thread

דוגמא בסיסית של נים:

```
class CounterThread extends Thread {
	private int count;
	public void run(){
		count=0;
		while(true) {
			System.out.print( count + "   ");
			if (count ==100)  count =0;
			else count ++;
		}
	}
}

class PrintThread extends Thread {
	String msg;
	public PrintThread(String p_msg){
		msg = p_msg;
	}
	public void run(){
		while(true)
			System.out.print( msg + '   ' );
	}

}

public class ThreadTest {
	public static void main(String[] args){
		Thread  thread1=new CounterThread();
		Thread  thread2=new PrintThread("***");

		// Do not call "run" directly !
		thread1.start();
		thread2.start();
	}
}
```

תיאור הנימים:

1. נים ראשון counterThread מדפיס בלולאה אינסופית מספרים בין 0-99 כשאר המונה מגיע ל-100 הוא מאופס ל - 0.
2. נים שני printThread מדפיס מחרוזת שניתנת לו במתודה הבונה בדוגמא שלנו 3 כוכביות.

תוצאה אפשרית:

```
0 *** 1 *** 2 3 *** 4 *** *** 5 6   ...
```

שימו לב שהתוצאה היא רק אפשרית ויש אינסוף תוצאות אפשריות.

2.1.1. אופן הגדרת נים שיטה I

<u>אופן ההגדרה</u>

1. הגדרת מחלקה חדשה אשר נורשת מ- Thread .
2. מימוש הפונקציה run.

הפונקציה run היא לב הנים ובו אנו מגדירים את התוכן והמשמעות של הנים.

דוגמא:

```
class CounterThread extends Thread {
      public void run() {
            //Thread code
      }
}
```

2.1.2. אופן הרצה הנים

1. יצירת אובייקט מסוג המחלקה שהגדרנו.
2. קריאה לפונקציה start ולא run !!!

דוגמא:

```
CounterThread  ct = new CounterThread();
ct.start();
```

<u>הערות נוספות</u>

1. העברת פרמטרים לנים נעשית רק בפונקציה הבונה ולא דרך run .
2. החזרת ערכים מהThread תעשה ע"י משתני מחלקה או משתנים גלובליים (נראה בהמשך).

שימו לב שאין אפשרות לכתוב:

```
public int run()
```

3. לפעמים רושמים:

```
Thread ct = new CounterThread();
ct.start();
```

זה שימושי אם רוצים למשל רוצים להריץ מערך של נימים שונים.

2.2. הגדרת של נים באמצעות ממשק runnable

במקרים מסוימים ייתכן שלא תרצו להצהיר על ירושה extends Thread (למשל, מאחר שלדעתכם המחלקה שלכם אינה "סוג של נים", וחשוב יותר: מאחר שהמחלקה שלכם כבר יורשת ממשהו אחר - זכרו כי ב- Java אין ירושה מרובה!).
פתרון: ב- Java מוגדר interface Runnable, ובו המתודה public void run() .
בעזרתו ניתן להגדיר Thread גם באופן הבא: (למען הסדר הטוב, אולי תעדיפו להכניס את הנים לתוך המחלקה):

```
class Counter implements Runnable {
      private   int           count;
      private   Thread   thread = new  Thread(this);
      public void run(){
            count=0;
            while(true) {
                  System.out.print( count + "   ");
                  if (count ==100)  count =0;
                  else count ++;
            }
      }
      public  void  startMe(){      // arbitrary  name
            thread.start();
      }
}
public class Test {
      public  static  void   main(String[ ]  args) {
            Counter  c = new  Counter( ) ;
            c.startMe();
      }
}
```

2.2.1. אופן הגדרת נים שיטה II

1. הגדרת מחלקה המממשת ממשק Runnable .
2. הגדרת הפונקציה run בדומה לדוגמא קודמת.

2.2.2. אופן הרצה שיטה II

שיטה זו הינה השיטה העדיפה. השלבים:

1. הגדרת משתנה פנימי מסוג Thread אשר בפונקציה הבונה שלו יקבל את האובייקט של המחלקה המממשת את הנים. למחלקה Thread יש פונקציה בונה המחכה לאובייקט מסוג Runnable. כאשר מגדירים Thread בצורה זו הוא מקבל את הפונקציה run כאילו הוגדרה בתוכו.
2. הוספת פונקציה startMe אשר תריץ את ה-Thread .

שימו לב כי ממשק Runnbable מורכב מהפונקציה run בלבד ונועד לפתור את בעיית הירושה.

3. תרגילים

3.1. תרגיל 1

<u>תיאור המשימה</u>

כתבו תכנית המריצה מספר מסויים של נימים. כל נים מגריל מספר בין 0 ל – 10.
הנים המרכזי מחשבת את הסכום של כל המספרים המוגרלים.

<u>פתרון</u>

בתרגיל זה נראה שימוש במתודת join אשר נמצאת במחלקה Thread .
(בשיעור הבא נגדיש זמן לחקירת ה- API של המחלקה הזו).
יש כאן בעיה של תזמון: אנחנו רוצים שקודם המספרים יחושבו ואח"כ יחושב הסכום, כלומר אנחנו דורשים סדר ביצוע מסוים.

הפקודה join אומרת: מי שמפעיל את הפקודה מחכה ל-thread ממנו הפעיל את join .

Main

t.join()

פונקצית ה-main מחכה עד אשר t יסתיים. **שימו לב שהפעולה לא פועלת כלל על t!**

קוד:

```
package rollexample;
public class RollTrd extends Thread{
  private int value;
  final   int UPPER_BOUND  = 10;
  public void run(){
    value = (int)(UPPER_BOUND*Math.random());
    System.out.println(value);
  }
  public int getValue(){
    return value;
  }
}
```

הנים פשוט מגריל מספר.
שימו לב שכדי לקבל ערך חוזר מנים צריך להגדיר משתנה מחלקה.

המחלקה הראשית:

```
package rollexample;

public class RollTest {
  final static int NUMBER_OF_THREADS = 5;
  public static void tst(){
    Thread[] trds = new Thread [NUMBER_OF_THREADS];
    for(int i=0;i<trds.length;i++){
      trds[i] = new RollTrd();
      trds[i].start();
    }
    for(int i=0;i<trds.length;i++)
    try{
      trds[i].join();
    }
    catch(InterruptedException e){
    }
    int sum = 0;
    for(int i=0;i<trds.length;i++)
      sum += ((RollTrd)trds[i]).getValue();
    System.out.println("-----------------");
    System.out.println(sum);
  }
  public static void main(String[] args) {
    tst();
  }
}
```

שימו לב אין לשלב את הקריאה ל-join בתוך בלוק הקריאה ל-start. זה יוצר סדרתיות!

3.2. תרגיל 2

<u>תיאור המשימה</u>

כתבו מחלקה היוצרת 5 נימים. כל נים מקבל מערך ומספר. כל נים מוסיף את המספר שקיבל לכל אחד מתאי המערך.

דרישות נוספות:

- הנימים צריכים לרוץ במקביל.
- הנים הראשי מדפיס את המערך בסוף.

<u>דוגמא</u>

אם נוצרו 5 נימים שקיבלו את המספרים מ-1 עד 5. אז בכל תא במערך בסופו של דבר יהיה הערך 15.

פתרון

הנים:

```
package adder;
public class AdderThread extends Thread{
  int[] arr;
  int   value;
  public AdderThread (int[] p_arr,int p_value){
    arr   = p_arr;
    value = p_value;
  }
  public void run(){
    for(int idx = 0;idx < arr.length;idx++)
      arr[idx] += value;
  }
}
```

הנים הראשי:

```
package adder;
public class AdderTest {
  final static int NUMBER_OF_THREADS = 5;
  final static int NUMBER_OF_CELLS   = 5;
  public static void tst(){
    Thread[] trds = new Thread [NUMBER_OF_THREADS];
    int[] arr = new int [NUMBER_OF_CELLS];
    for(int i=0;i<trds.length;i++){
      trds[i] = new AdderThread(arr,i+1);
      trds[i].start();
    }
    for(int i=0;i<trds.length;i++)
    try{
      trds[i].join();
    }
    catch(InterruptedException e){
    }

    for(int i=0;i<trds.length;i++)
      System.out.println(arr[i]);
  }
  public static void main(String[] args) {
    tst();
  }
}
```

הערה: במימוש יש בעית סנכרון אותה נלמד לפתור בהמשך.