



המחלקה להנדסת מחשבים
מעבדה במערכות משובצות מחשב ו-IOT

הרצאה 2- תהליכונים (THREADS) בJAVA

תהליכונים (Threads)

- תהליכון, *thread*, נים או חוט (כינויים בספרות) הוא קטע קוד מוגדר, אשר יבוצע בנפרד מה"זרימה" הרגילה של התוכנית, בצורה מקבילית.
- כל תוכנית בג'אווה תרוץ לפחות בתור תהליכון בודד, שייוצר ע"י המכונה הוירטואלית (JVM).

תהליכונים – מוטיבציה

- קל לראות כי המון זמן מחשב מתבזבז בהמתנה לסיום פעולות שדורשות הרבה זמן.
- ישנן כמה פעולות שכיחות, כמו קלט - אשר מחכה לתגובה ה"איטית" של המשתמש האנושי, פלט - אשר מחכה לאישור המדפסת, פניה לדיסק הקשיח וכו' שכאשר פונים אליהן, שבהן למעשה המעבד לא מנוצל באותו זמן.
- תוכניות אחרות שהיו יכולות לרוץ מחכות עד לסיום הפעולות היקרות בזמן.

תהליכונים – מוטיבציה

- כדי להתגבר על הבעיה משתמשים בעיבוד מקבילי.
- כלומר, כמה תוכניות ישתמשו לסירוגין ביכולת העיבוד של המחשב וכך הניצולת של המחשב תגדל.
- כמובן שאנו מדברים על עיבוד כמו-מקבילי, ולא מקבילי באמת.
- עיבוד מקבילי אמיתי נעשה ע"י מספר מעבדים / מכונות, וגם לו יש יתרונות עצומים משל עצמו. בכל אופן, אנו נתרכז במערכות חד-מעבדיות.

תהליכונים (Threads)

- הפתרון בג'אווה לריבוי משימות מבוקר הוא שימוש בתהליכונים – threads.
- תהליכון - יחידת קוד תוכנית לביצוע, מצביע להוראה הנוכחית בקוד (ip – instruction pointer) מחסנית וערמה.
- הכוונה בקטע הקוד - פונקציה מסוימת וכל הפונקציות להן היא קוראת.

תהליכונים - מימוש ב-Java

- המחלקה *Thread* היא מחלקה שממשת את *Runnable*: הממשק

```
interface Runnable
{
    public abstract void run();
}

class Thread implements Runnable
{
    public void run();
    . . .
}
```

תהליכונים - מימוש בJava

- כל תהליכון מקבל פרק זמן מסוים לריצה (קואנטום), שבתומו ה-JVM תעבור לתהליכון הבא בתור.
- מערכת ג'אווה תתן זכות קדימה לתהליכון עם עדיפות גבוהה. במקרה של עדיפויות שוות ההחלטה תהיה בידי מערכת ההפעלה.
- כדי למנוע מצב של הרעבה של התהליך, גם תהליך בעדיפות נמוכה יקבל, מדי פעם, "פרוסת" זמן עיבוד.

תהליכונים – מימוש בJava

- כל תהליך שנוצר עובר למצב ריצה עם הפעלת פונקציית *start()* שלו ואז הוא נכנס לתור ההמתנה.
- יש באפשרותו להשעות את עצמו, ע"י שימוש בפונקציית *yield()* שמוותרת על המשך פרק הזמן שהוקצה לתהליך למען תהליך בעדיפות זהה, או בפונקציית *sleep()* שקובעת זמן המתנה במילי-שניות.
- סיום הריצה מתבצע או כשהתהליך הסתיים, או בסיום יזום, ע"י קריאה לפונקציה *stop()* או בהצבת ערך null לתהליכון.
- שימוש בפונקציה *isAlive()* יחזיר את מצב התהליכון, אם הוא חי או לא.

יצירת תהליכון

- קיימות שתי דרכים ליצירת תהליכון:
- הגדרת מחלקה היורשת מ- Thread, ומימוש הפונקציה *run()*
- הגדרת מחלקה כממשת לממשק Runnable, מימוש הפונקציה *run()* ויצירת עצם Thread עוטף.

דוגמא לשיטה הראשונה

- נבנה מחלקה, שתיצור שני תהליכונים, מסוג שניצור בהמשך, ונפעיל את שני התהליכונים ע"י פונקציית start.

```
public class Thr
{
    public static main (String args[])
    {
        MyThread t1 = new MyTherad("t1");
        MyThread t2 = new MyTherad("t2");

        t1.start();
        t2.start();
    }
}
```

דוגמא לשיטה הראשונה

- נבנה את מחלקת התהליכון. פונקציית הקונסטרקטור תתן שם לתהליכון.
- התהליכון עצמו יריץ לולאה מספר פעמים, כשבתוכה הוא "ינוח" זמן אקראי כלשהו, ע"י פקודת sleep.
- בזמן שתהליכון אחד ישן המעבד יפנה לעבד את התהליכון השני.

דוגמא לשיטה הראשונה

```
class MyThread extends Thread
{
    MyThread(String caption)
    {
        super(caption);
    }

    public void run()
    {
        for(int i = 1; i < 5; i++)
        {
            System.out.println(i+" "+getName());
            try{
                sleep ((long)(Math.random()*1000));
            }catch (InterruptedException e){}
        }
    }
}
```

דוגמא לשיטה הראשונה

■ פלט אפשרי:

1	t2
1	t1
2	t2
2	t1
3	t1
3	t2
4	t2
4	t1

דוגמא לשיטה השנייה

```
public class RunThr
{
    public static void main(String args[])
    {
        MyRunThread t1 = new MyRunThread ("t1");
        MyRunThread t2 = new MyRunThread ("t2");

        Thread t11 = new Thread(t1);
        Thread t22 = new Thread(t2);

        t11.start();
        t22.start();
    }
}
```

דוגמא לשיטה השנייה

```
class MyRunThread implements Runnable
{
    String name;

    MyRunThread(String n)
    {
        name = n;
    }

    public void run()
    {
        for (int i = 1; i < 5; i++)
        {
            System.out.println(i+" "+name);
            try{
                Thread.sleep((long)(Math.random()*1000));
            }catch (InterruptedException e){}
        }
    }
}
```

עדיפויות

- ע"י הפונקציה (*setPriority*) ניתן לקבוע את רמת העדיפות של התהליכון. קיימים שלושה סוגי רמות עדיפות עיקריים:

MAX_PRIORITY
NORM_PRIORITY
MIN_PRIORITY

- תהליכון שנוצר ע"י JVM יהיה בעל עדיפות נורמלית. כמו כן, כל תהליכון שיווצר ממנו יורש את אותה עדיפות, אלא אם כן נשנה אותה.

עדיפויות

- לדוגמא נוסיף עדיפות בתוכנית שכתבנו:

```
public class PriThr
{
    public static void main(String args[])
    {
        MyPriThread t1 = new MyPriThread ("t1");
        MyPriThread t2 = new MyPriThread ("t2");

        Thread t11 = new Thread(t1);
        Thread t22 = new Thread(t2);

        t11.setPriority(Thread.MAX_PRIORITY);
        t22.setPriority(Thread.MIN_PRIORITY);

        t11.start();
        t22.start();
    }
}
```

עדיפויות

```
class MyPriThread implements Runnable
{
    String name;

    MyPriThread(String n)
    {
        name = n;
    }

    public void run()
    {
        for (int i = 1; i < 5; i++)
        {
            System.out.println(i+" "+name);
            for(int j=0; j==1000; j++);
        }
    }
}
```

עדיפויות

■ פלט אפשרי:

1	t1
2	t1
3	t1
4	t1
1	t2
2	t2
3	t2
4	t2

עדיפויות


- שינינו את זמן ההמתנה מפונקצית שינה (sleep), שבה התהליכון השני היה מקבל זמן עיבוד בזמן שינת התהליכון הראשון, ללולאה שמתבצעת במעבד ושבה לא עובר זמן העיבוד לתהליכון אחר.
- התהליכון השני יכול היה לקבל זמן עיבוד באמצע עיבוד התהליכון הראשון, במקרה שהלולאה היתה גוזלת זמן מוגדר גדול, על מנת למנוע הרעבה.

סנכרון תהליכים

- בדרך כלל, עבודה בתהליכונים תעשה בצורה מתואמת ביניהם. למשל, תהליכון אחד שולח הודעות והתהליכון השני מקבל אותן. למצב כזה דרוש סנכרון בין התהליכונים.
- סנכרון כזה יעשה ע"י שימוש בפונקציות *wait ()* ו-*notify ()* ובמנגנון *synchronized*.
- *wait ()* - תגרום לתהליכון לחכות. התהליכון יכנס לתור הממתינים ותהליכון אחר יתפוס את מקומו.
- *notify ()* - קריאה מהתהליכון שמתבצע לתהליכון אחר, שממתין בתור, להכנס לפעולה (*notifyAll ()*) יעיר כמה תהליכונים שממתינים).



מנגנון synchronized

- זהו מנגנון המאפשר לנעול קטע קוד (עצם, פונקציה בודדת, או קטע כלשהו), כך שרק תהליכון אחד יוכל להשתמש בו באותו הזמן, וכך נפתרת בעיית עיבוד מקבילי ידועה - בעיית הקטע הקריטי.
- 

דוגמא לשימוש במנגנון **synchronized**:

- מחלקת Message, מחלקה כללית לשליחת מסר, הכוללת: ערך של המסר ודגל, שמראה אם תיבת המסרים מלאה או ריקה.
- בנוסף - שתי פונקציות: שליחת מסר - שמקבלת ערך של מסר ושולחת אותו אם תיבת המסרים ריקה (דגל שלילי), או מחכה למשיכת המסר הקודם ורק אז תשלח את החדש; קבלת מסר - מחכה עד שיש משהו בתיבת המסרים, ואז קוראת אותו.
- שתי הפונקציות מוגנות ע"י מנגנון הסנכרון כך שלא תאבדנה את הפיקוח באמצע הפעולה.

```
public class Message
{
    int contents;
    boolean flag = false;

    public synchronized void put(int i)
    {
        while (flag == true)
        {
            try{
                wait();
            }
            catch(InterruptedException e){}
        }
        contents = i;
        flag = true;
        notify();
    }

    public synchronized int get()
    {
        while (flag == false)
        {
            try{
                wait();
            }
            catch(InterruptedException e){}
        }
        flag = false;
        notify();
        return contents;
    }
}
```


מחלקת Send המשמשת לשליחת ההודעות:

```
public class Send extends Thread
{
    Message message;
    int id;

    public Send(Message m, int i)
    {
        message = m;
        id = i;
    }

    public void run()
    {
        for (int g=0; g<5; g++)
        {
            message.put(g);
            System.out.println("the "+g+" message was sent by: "+id);
            try{
                sleep((long)(Math.random()*1000));
            }
            catch(InterruptedException e){}
        }
    }
}
```

מחלקת Get אשר מקבלת את ההודעות

```
public class Get extends Thread
{
    Message message;
    int id;

    public Get(Message m, int i)
    {
        message = m;
        id = i;
    }

    public void run()
    {
        int value = 0;
        for (int g=0; g<5; g++)
        {
            value = message.get();
            System.out.println("the "+value+" message was received by: "+id);
        }
    }
}
```

מחלקת UseProg-המחלקה הראשית:

```
public class Useprog
{
    public static void main (String[] args)
    {
        Message message = new Message();
        Send s = new Send(message ,1);
        Get g = new Get(message ,2);

        s.start();
        g.start();
    }
}
```

תוצאת הפלט

```
the 0 message was sent by: 1
the 0 message was received by: 2
the 1 message was sent by: 1
the 1 message was received by: 2
the 2 message was sent by: 1
the 2 message was received by: 2
the 3 message was sent by: 1
the 3 message was received by: 2
the 4 message was sent by: 1
the 4 message was received by: 2
```

התוכנית יצרה שני תהליכונים, שהעבירו ביניהם
עצמים מסוג *Message* בצורה מסונכרנת.

עוד על תהליכונים:

- אם נגדיר זמן ל-*wait* () התהליכון ימשיך גם ללא הפונקציה *notify* ()
- אפשר לחטוף זמן מהתהליכון, ע"י הפרעה - *interrupt* (*InterruptedException*) שתגרום לזריקת חריגת
- ניתן להשעות תהליכון ע"י קריאה לפונקציית ההשעיה - *suspend* () שלו ולהחזירו לפעולה ע"י פונקציית ההפעלה מחדש - *resume* ()
- תהליכון יחכה לתהליכון אחר בעזרת פונקציית *join* (של התהליכון השני.

אובייקט אנונימי

AnonymousObjectExample.java X

```
1 public class AnonymousObjectExample {
2     public static void main(String[] args) throws InterruptedException {
3
4         Thread t = new Thread(new Runnable() {
5
6             @Override
7             public void run() {
8                 for (int i = 0; i < 50; i++) {
9                     System.out.print(".") + i + ".\t");
10                    if (i % 10 == 0)
11                        System.out.println();
12                }
13            }
14        });
15        t.start();
16
17        for (int i = 0; i < 50 ; i++) {
18            System.out.print("_" + i + "_\t");
19            if (i%10 == 0)
20                System.out.println();
21        }
22
23    }
24 }
```

יצירת אובייקט זמני מטיפוס
Runnable ומימוש השיטה run

עבודה עם סדרי עדיפויות - priority

- לכל thread יש עדיפות מ-1 עד 10. ב"מ היא 5. ככל שהעדיפות יותר גבוהה כך ה-thread יועדף בעת ההרצה.
- שימו לב! ה-main הוא גם Thread עם עדיפות ב"מ.

```
1 public class ThreadPriorityExample {
2
3     public static void main(String[] args) {
4         Thread t = new Thread();
5         t.setPriority(5);
6
7         System.out.println("main's priority: " + Thread.currentThread().getPriority());
8         System.out.println("thread1 priority: " + t.getPriority());
9         t.start();
10
11         for (int i = 0; i < 50; i++) {
12             System.out.print("_" + i + "_\t");
13             if (i % 10 == 0)
14                 System.out.println();
15         }
16     }
17 }
```

ערך ב"מ. כלומר פקודה
זו כרגע מיותרת.

העדיפות של ה-
main thread

העדיפות של ה-
thread שיצרתי

```
main's priority: 5
thread1 priority: 5
```

0				
1	_2_	_3_	.0.	_4_
5	.1.	_6_	.2.	_7_

בגלל שלשני ה-thread'ים עדיפות
זהה, ניתן לראות שהם רצו במקביל

שיטת run במחלקת Thread1 שיצרנו שמרחיבה (extends) את Thread

```
public void run() {  
    for (int i = 0; i < 50; i++) {  
        System.out.print(".") + i + ".\t");  
        if (i % 10 == 0)  
            System.out.println();  
    }  
}
```


(2) priority

```
1 public class ThreadPriorityExample {
2
3     public static void main(String[] args) {
4         Thread t = new Thread();
5         t.setPriority(1);
6
7         System.out.println("main's priority: " + Thread.currentThread().getPriority());
8         System.out.println("thread1 priority: " + t.getPriority());
9         t.start();
10
11        for (int i = 0; i < 50; i++) {
12            System.out.print("_" + i + "_\t");
13            if (i % 10 == 0)
14                System.out.println();
15        }
16    }
17 }
```

מתן ערך עדיפות נמוך

ניתן לראות שהגלל של thread יש
עדיפות נמוכה הוא רץ אחרי ה-
thread המרכזי.

main's priority: 5

thread1 priority: 1

0									
1	_2_	_3_	_4_	_5_	_6_	_7_	_8_	_9_	_10_
11	_12_	_13_	_14_	_15_	_16_	_17_	_18_	_19_	_20_
21	_22_	_23_	_24_	_25_	_26_	_27_	_28_	_29_	_30_
31	_32_	_33_	_34_	_35_	_36_	_37_	_38_	_39_	_40_
41	_42_	_43_	_44_	_45_	_46_	_47_	_48_	_49_	.0.
.1.	.2.	.3.	.4.	.5.	.6.	.7.	.8.	.9.	.10.
.11.	.12.	.13.	.14.	.15.	.16.	.17.	.18.	.19.	.20.
.21.	.22.	.23.	.24.	.25.	.26.	.27.	.28.	.29.	.30.
.31.	.32.	.33.	.34.	.35.	.36.	.37.	.38.	.39.	.40.
.41.	.42.	.43.	.44.	.45.	.46.	.47.	.48.	.49.	

(3) priority

```
1 public class ThreadPriorityExample {
2
3     public static void main(String[] args) {
4         Thread t = new Thread();
5         t.setPriority(10);
6
7         System.out.println("main's priority: " + Thread.currentThread().getPriority());
8         System.out.println("thread1 priority: " + t.getPriority());
9         t.start();
10
11         for (int i = 0; i < 50; i++) {
12             System.out.print("_" + i + "_\t");
13             if (i % 10 == 0)
14                 System.out.println();
15         }
16     }
17 }
```

מתן ערך עדיפות גבוה

ניתן לראות שהגלל של thread יש
עדיפות גבוהה, כאשר הוא נכנס
לפעולה הוא רץ יותר זמן.

```
main's priority: 5
thread1 priority: 10
```

```
_0_
_1_    .0.
.1.    .2.    .3.    .4.    .5.    .6.    .7.    .8.    .9.    .10.
.11.   .12.   .13.   .14.   .15.   .16.   .17.   .18.   .19.   .20.
.21.   .22.   .23.   .24.   .25.   .26.   .27.   .28.   .29.   .30.
.31.   .32.   .33.   .34.   .35.   .36.   .37.   .38.   .39.   .40.
.41.   .42.   .43.   .44.   .45.   .46.   .47.   .48.   .49.   _2_
```

חידע על thread

```
1 public class ThreadInfoExample {
2
3     public static void main(String[] args) {
4         System.out.println("id: " + Thread.currentThread().getId());
5         System.out.println("name: " + Thread.currentThread().getName());
6         System.out.println("priority: " + Thread.currentThread().getPriority());
7         System.out.println("toString: " + Thread.currentThread());
8
9         Thread.currentThread().setName("myThread");
10        System.out.println("toString: " + Thread.currentThread());
11
12        Thread4 t1 = new Thread4();
13        Thread1 t2 = new Thread1();
14        System.out.println("t1 toString: " + t1.toString());
15        System.out.println("t2 toString: " + t2.toString());
16    }
17 }
```

id: 1
name: main
priority: 5
toString: Thread[main,5,main]
toString: Thread[myThread,5,main]
t1 toString: Thread[Thread-0,5,main]
t2 toString: Thread[Thread-1,5,main]

עדיפות

שם ה-thread

שם ה-process

שם הב"ח ל-thread