פרק 17 - Thread

מה הוא Thread?

thread הוא רצף של פעולות שמתבצעות במקביל לקטעי קוד אחרים.

ריבוי Thread נקרא multi threading

לדוגמה : אני יכול לשמוע מוזיקה ובמקביל לערוך מסמך word ולכל אחד יש thread במקביל.

שני threadים יכולים לעבוד על אותם נתונים.

- לכל תכנית יש thread ראשי
- המטרה של ה thread היא לתת לנו את ההרגשה שרצים מספר קטעי קוד במקביל על מספר מעבדים.
- בפועל - קיים רק מעבד 1.
- תור הפקודות - בתור זה עומדות פקודות בהמתנה לביצוע.

במדעי המחשב מכונה וירטואלית היא תוכנה היוצרת סביבה הנחוצה להפעלתה של סביבה אחרת, מבלי שיהיה צורך במימוש פיזי של תוכנה זו

- cpu - כל thread הוא רצף של פעולות שמתבצע בנפרד.לכל thread מוקצה מעבד וירטואלי(virtual cpu).

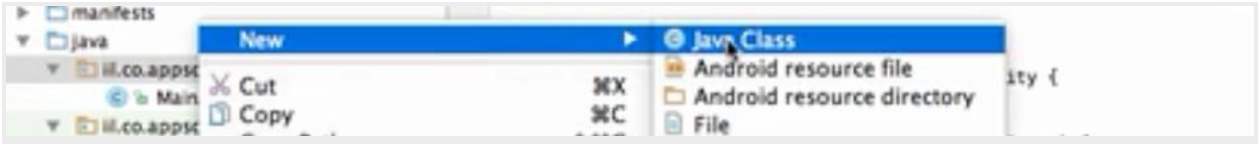
- data - נתוני ה-thread יכולים להיות משותפים **ליותר** מ-thread אחד.

כאשר נרצה ליצור Thread נבצע את סדר הפעולות הבא :

1. ניצור class שיורש מ - Thread
2. בקלאס שיצרנו - נממש את הפונקציה run
3. ב Main - נכריז על אובייקט מסוג ה - class שיצרנו
4. ב Main - נקרא לפקודה start, פקודה זו תפעיל את הפונקציה run שתרוץ ברקע.

ביחד נבנה Thread לדוגמה:

שלב 1 - נפתח יחד class חדש:



נוריש ל class שיצרנו את המחלקה Thread: (בדוגמה הזו קראנו ל class החדש שיצרנו Person)

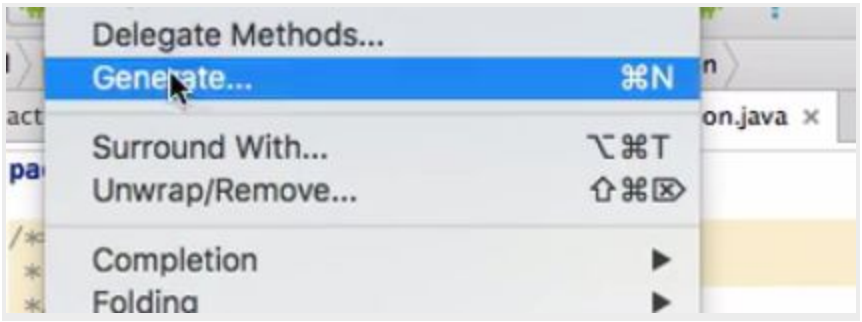
```
public class Person extends Thread {
```

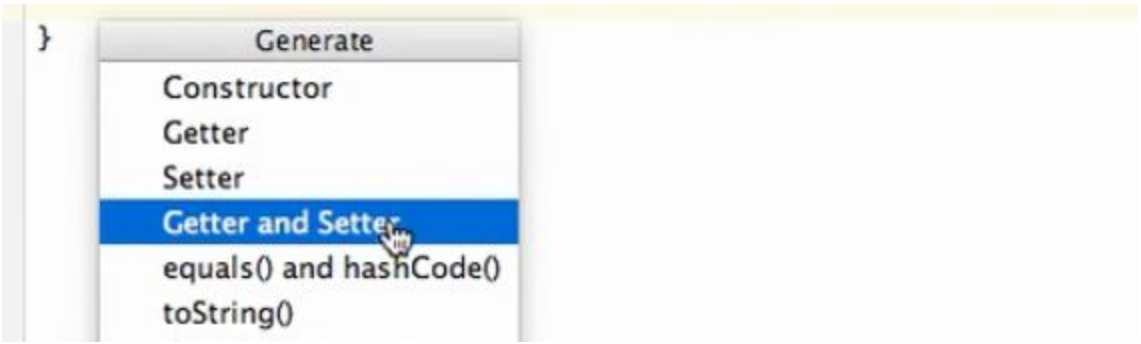
ניתן לקלאס תכונות, בנאי, getters ו setters:

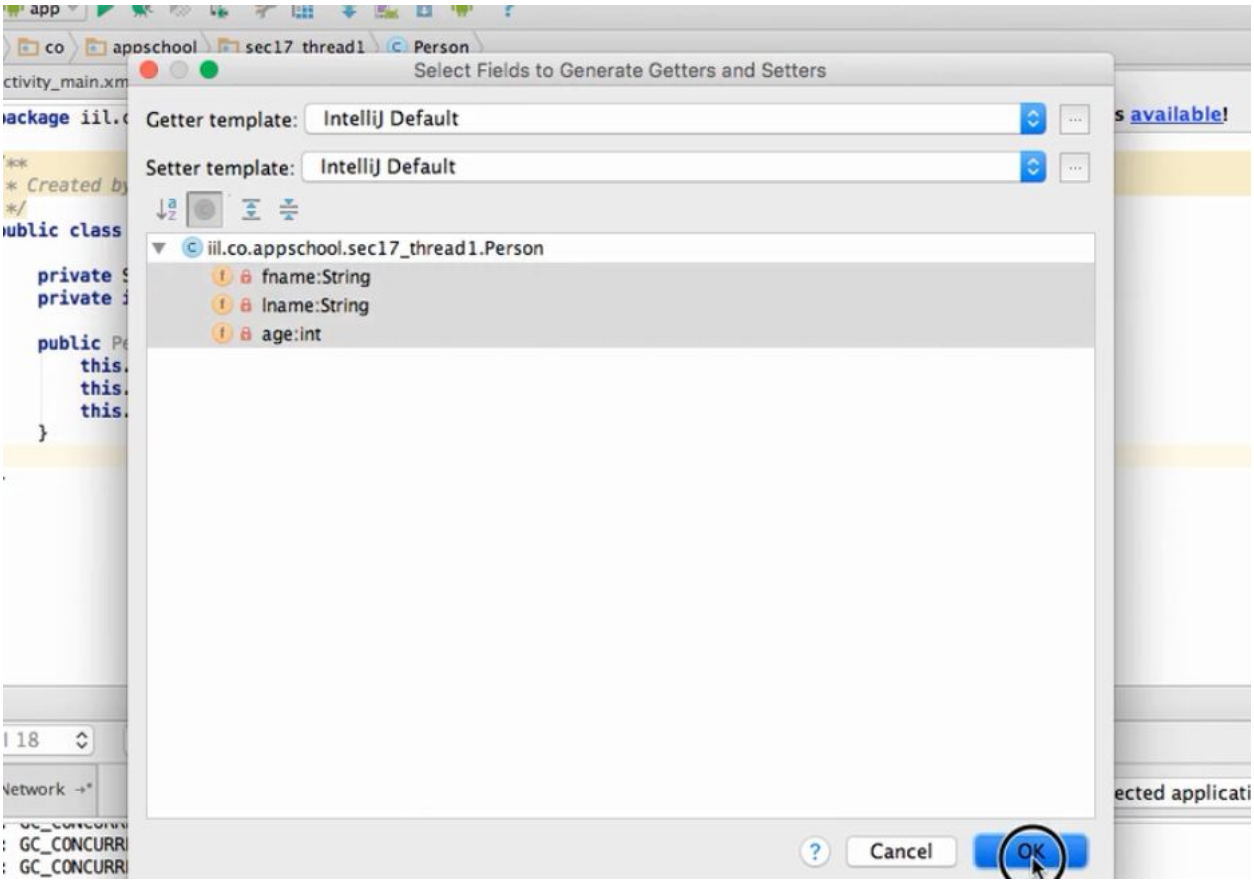
```
public class Person extends Thread
{
   private String fname;
    private String lname;
    int age;
    public Person(String fname, String lname, int age) {
       this.fname = fname;
       this.lname = lname;
       this.age = age;
    }
    public String getFname() {
       return fname;
    }
    public void setFname(String fname) {
       this.fname = fname;
    }
```

בשביל ליצור את הבנאי במהירות נוכל לגשת בסרגל שבמסך למעלה באנדרואיד סטודיו, לגשת ל code-> generate-> constructor.

נוכל לעשות זאת גם ל getters וה setters.







שלב 2 - בניית הפונקציה Run():

```
@Override
public void run() {
   for(int i=0;i<10;i++)
   {
      System.out.println(Thread.currentThread().getName());
      try
      {
         Thread.sleep(2000);
      }
      catch (InterruptedException e)
      {
         e.printStackTrace();
      }
      System.out.println(this.fname);
   }
}
}
```

הפעולה תרוץ בלולאה על 10 thread'ים, תדפיס את השם של ה thread הנוכחי, תחכה שתי שניות ולאחר מכן תדפיס גם את השם משפחה

שלב 3 - בניית ה Thread ומימוש הפונקציה run ב Main:

```
public class Program {
public static void main(String[] args) {
// TODO Auto-generated method stub
    person p1=new person("asaf","david",5);
    person p2=new person("eli","shalom",6);
    p1.start();
    p2.start();
}
```

ניצור ThreadStart שיפעיל את הפונקציה run

ניצור Thread שיפעיל את ה ThreadStart

נפעיל את ה Thread

בגלל שיצרנו Thread, יודפס למסך Hello how do you feel עוד לפני שה Thread שלנו סיים לפעול

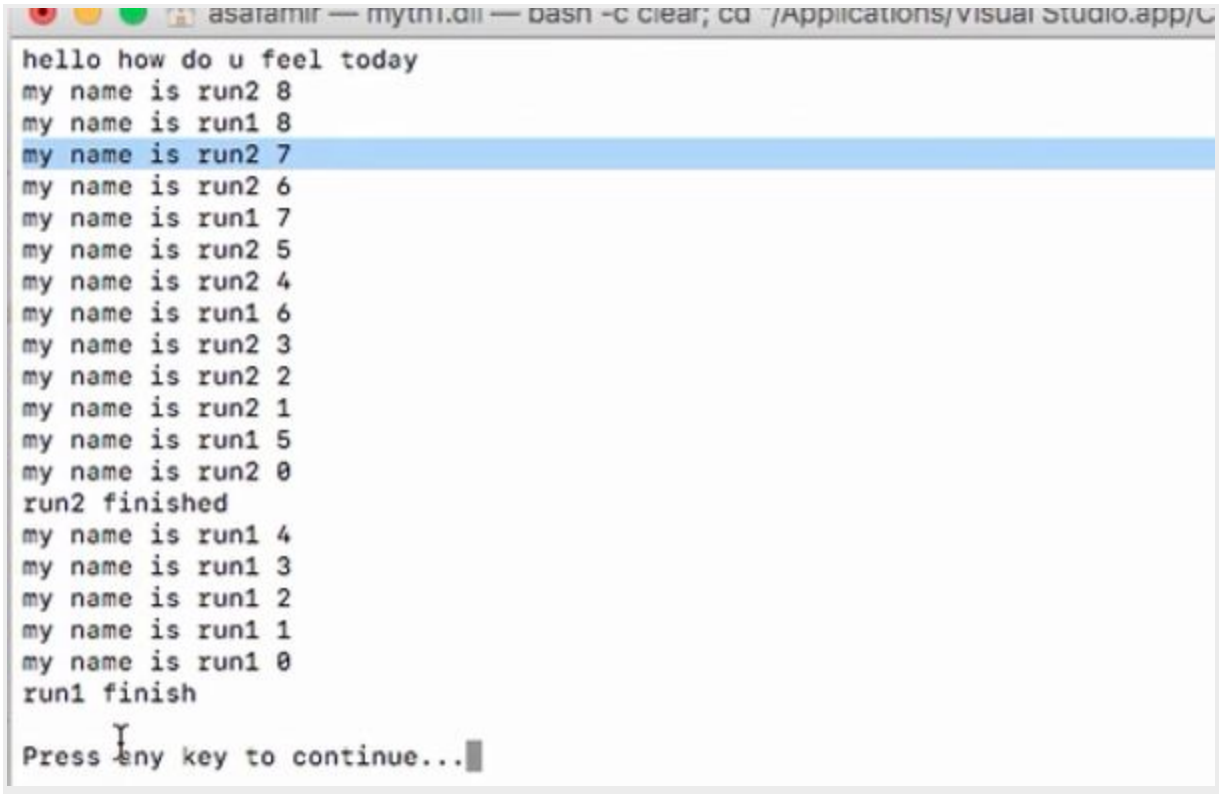
סיכום ביניים :

4 שלבים ליצירת thread -

1. ניצור class שיורש מ - Thread
2. בקלאס שיצרנו - נממש את הפונקציה run
3. ב Main - נכריז על אובייקט מסוג ה - class שיצרנו
4. ב Main - נקרא לפקודה start, פקודה זו תפעיל את הפונקציה run שתרוץ ברקע.

נוכל להפעיל עוד Thread שירוץ במקביל ל Thread הראשון.

Output של שני threadים שרצים במקביל:



דוגמה 2 - מימוש Thread ב class:

דרך שניה לביצוע Thread היא לממש את הממשק Runnable שמחייב אותנו לממש את הפונקציה run.

שלב 1 - יצירת קלאס Person (בדוגמה שלנו) ומימוש הממשק Runnable:

```
public class Person2 implements Runnable {
private String fname;
private String lname;
int age;

public Person2(String fname, String lname, int age) {
   this.fname = fname;
   this.lname = lname;
   this.age = age;
}

public String getFname() {
   return fname;
}

public void setFname(String fname) {
   this.fname = fname;
}
```

שלב 2 - מימוש הפונקציה run:

```
@Override
public void run() {
   for (int i = 0; i < 10; i++) {
      System.out.println(Thread.currentThread().getName());
      try {
         Thread.sleep(2000);
      } catch (InterruptedException e) {
         e.printStackTrace();
      }
      System.out.println(this.fname);
   }
 }
}
```

שלב 3 - יצירת מופע של Person ב Main והפעלת ה Thread:

```
public static void main(String[] args) {
// TODO Auto-generated method stub
    Person1 p1=new person("asaf","david",5);
    Person1 p2=new person("eli","shalom",6);
    Thread t1=new Thread(p1);
    Thread t2=new Thread(p2);
    t1.start();
    t2.start();
}
```

4 שלבים ליצירת פונקציה שרצה על thread במחלקה:

1.בקלאס שיצרנו - מבצעים implement לממשק Runnable

2.בקלאס שיצרנו - מבצעים override לפונקציה run

3.ניגשים ל-mainActivity ויוצרים אובייקט מסוג Thread ומעבירים לו ב-constructor אובייקט שכבר יצרנו שמבצע implement ל -Runnable

4.ב Main מפעילים את הפונקציה start לאובייקט מסוג Thread

פונקציות נוספות ב - Thread:

Thread.sleep(x)

היא פונקציה סטטית שמשהה את פעולת ה-thread למשך x אלפיות שניה.

:stop

לעומת זאת, הפעולה stop עוצרת את ה-thread.

רצוי לא להשתמש בפעולה זו כי היה יכולה להשאיר אותנו במצב לא צפוי.

בדרך כלל נעצור thread באמצעות דגלים,counter וכו.וכך לא נגיע למצבים בלתי צפויים.

wait notify synchronize:

לפעמים נרצה לבצע פעולות מסוימות או להפעיל thread מסוים רק לאחר שה Thread שלנו הסתיים. לשם הסנכרון נוכל להשתמש ב wait notify synchronize:

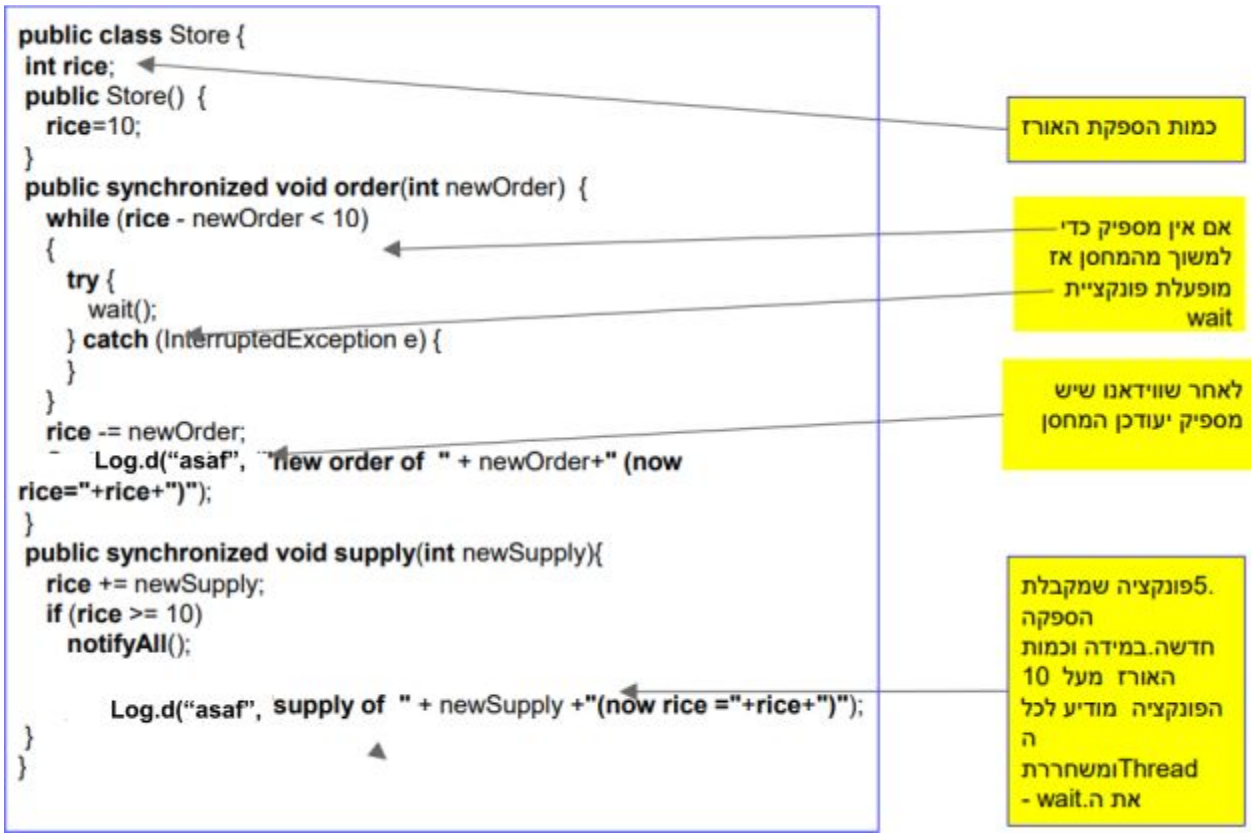
notify() and wait()

1. פעולות אלו שייכות למחלקה Object
2. אם במהלך Thread מופעלת הפונקציה wait אזי ה-Thread עוצר, כדי להעיר אותו נאלץ להפעילו על ידי notify אבל הוא לא יוכל להתעורר כי הוא קפוא. ולכן הערתו תעשה על ידי Thread אחר.
3. כל עוד לא הופעל על ה Thread -פונקציית notify או notifyall אזי ה thread-ימשיך להיות קפוא.
4. notify או wait יכולים להיות מופעלות רק מתוך בלוק של synchronized

דוגמה שמשלבת את כל פקודות ה - Thread

1. start,sleep,wait,synchronize,notify
2. נריץ דוגמה של מחסן האוכל
3. בדוגמה זו אנו מנסים למשך אוכל ממחסן האוכל. במידה ואין יתרה למשיכה של אוכל אנו ממתינים (wait) על ידי פונקציית ומחכים להספקה נוספת.

שלב 1 - ניצור class Store:



ניצור קלאס נוסף בשם NewSupply, שבעזרתו נזרים אורז לחנות:

```
public class NewSupply implements Runnable {
 private Store store;

 public NewSupply(Store store)
 {
   this.store = store;
 }

 public void run()
 {
   int newSupplyOfRice=0;
   for (int i = 0; i < 30; i++)
   {
     newSupplyOfRice = (int) (Math.random() * 500);
     store.supply(newSupplyOfRice);
     try
     {
       Thread.sleep((int) (Math.random() * 3000));
     }
     catch (Exception e)
     {

     }
   }
 }

}
```

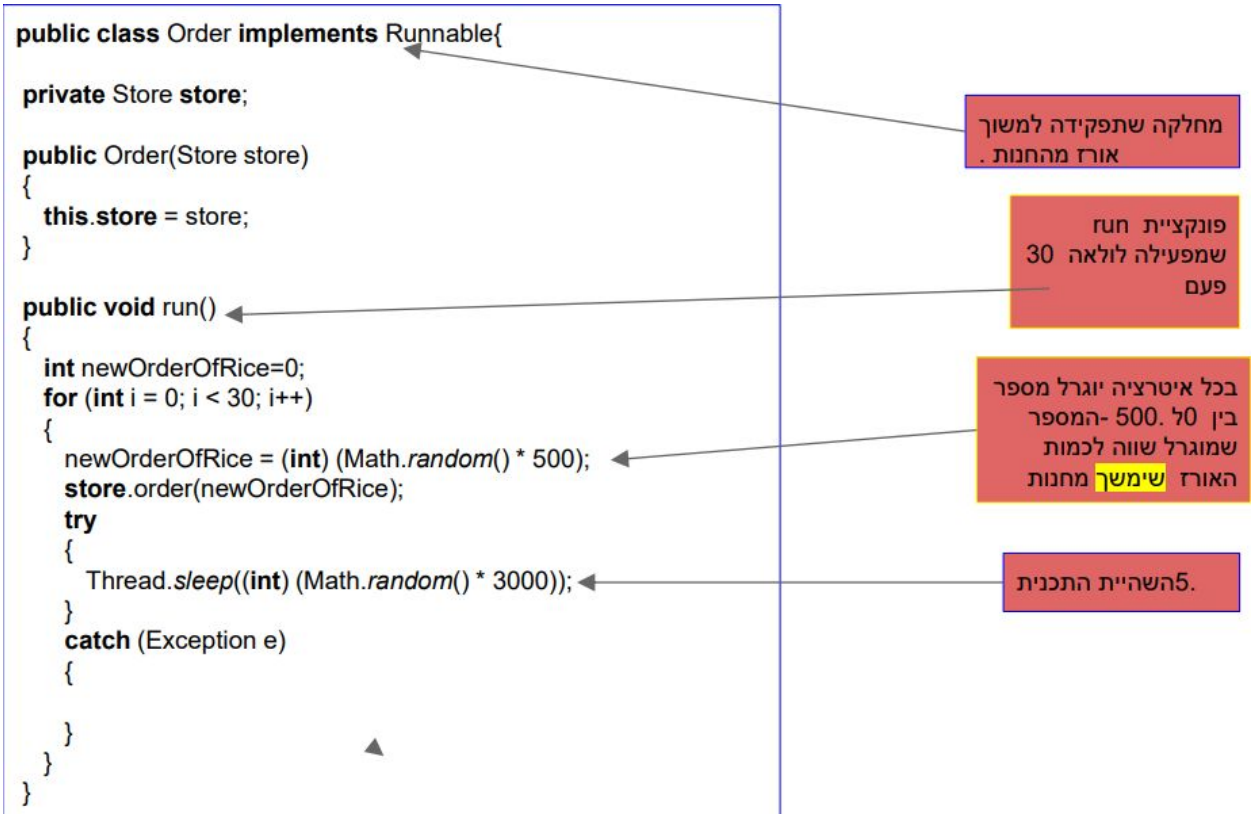
מחלקה שתפקידה להוסיף אורז לחנות .

פונקציית run שמפעילה לולאה 30 פעם

בכל איטרציה יוגרל מספר בין 0 ל 500. -המספר שמוגרל שווה לכמות האורז שיוזרם לחנות

השהיית התכנית 5.

נקים קלאס של Order, שבעזרתו נזמין ונשלוף אורז מהחנות:



נערוך את ה main שלנו:

```java
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);


        Store store = new Store();
        NewSupply ns1 = new NewSupply(store);
        NewSupply ns2 = new NewSupply(store);
        Order or1 = new Order(store);
        Order or2 = new Order(store);
        Thread t1 = new Thread(ns1);
        Thread t2 = new Thread(ns2);
        Thread t3 = new Thread(or1);
        Thread t4 = new Thread(or2);
        t1.start();
        t2.start();
        t3.start();
        t4.start();
    }
}
```

```
:51.710 2781-2795/? D/asaf: new supply of  399 (now rice=1048)
:51.840 2781-2797/? D/asaf: new order of  161 (now rice=887)
:52.730 2781-2796/? D/asaf: new order of  0 (now rice=887)
:53.280 2781-2797/? D/asaf: new order of  79 (now rice=808)
:53.360 2781-2797/? D/asaf: new order of  307 (now rice=501)
:53.740 2781-2794/? D/asaf: new supply of  346 (now rice=847)
:53.770 2781-2795/? D/asaf: new supply of  474 (now rice=1321)
:54.300 2781-2796/? D/asaf: new order of  316 (now rice=1005)
:54.400 2781-2796/? D/asaf: new order of  391 (now rice=614)
:55.280 2781-2797/? D/asaf: new order of  99 (now rice=515)
:55.420 2781-2794/? D/asaf: new supply of  50 (now rice=565)
:56.050 2781-2795/? D/asaf: new supply of  373 (now rice=938)
:56.590 2781-2796/? D/asaf: new order of  132 (now rice=806)
:57.290 2781-2795/? D/asaf: new supply of  371 (now rice=1177)
```