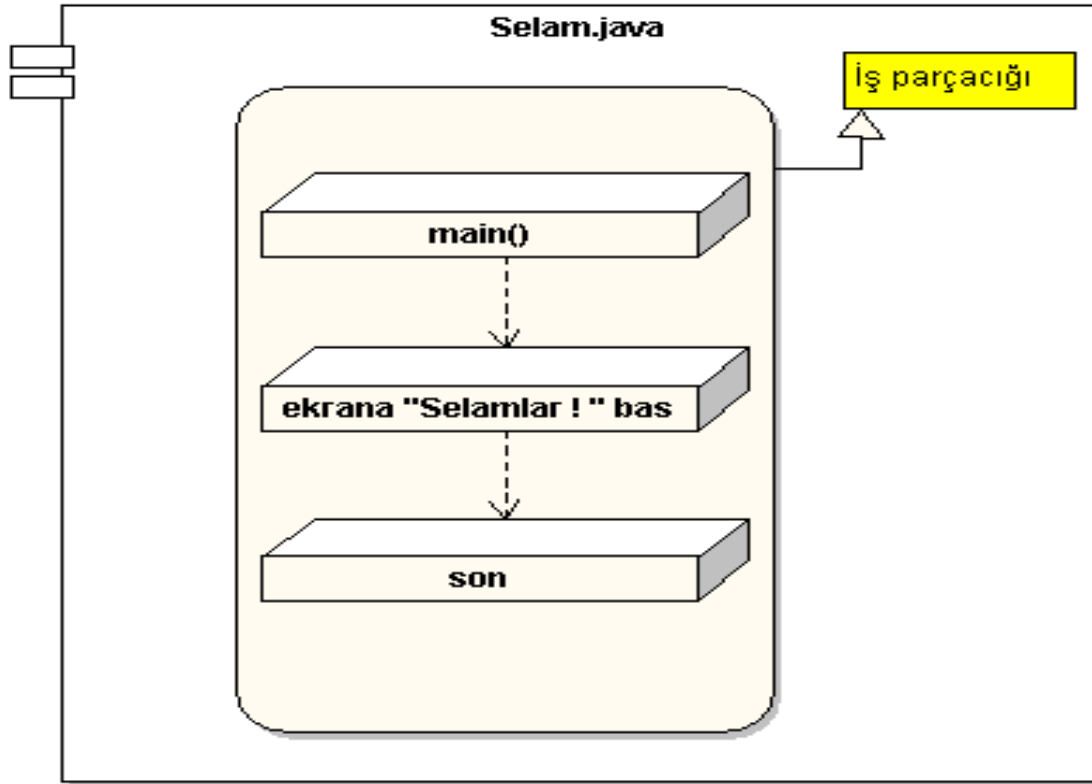


# İŞ PARÇACIKLARI (THREADS)

# İŞ PARÇACIKLARI

Geçen bölümlerde yapılan uygulama örnekleri hep sıralıydı.

Program başlar, belli bir yolu izleyerek işlemler yapar ve biterdi.



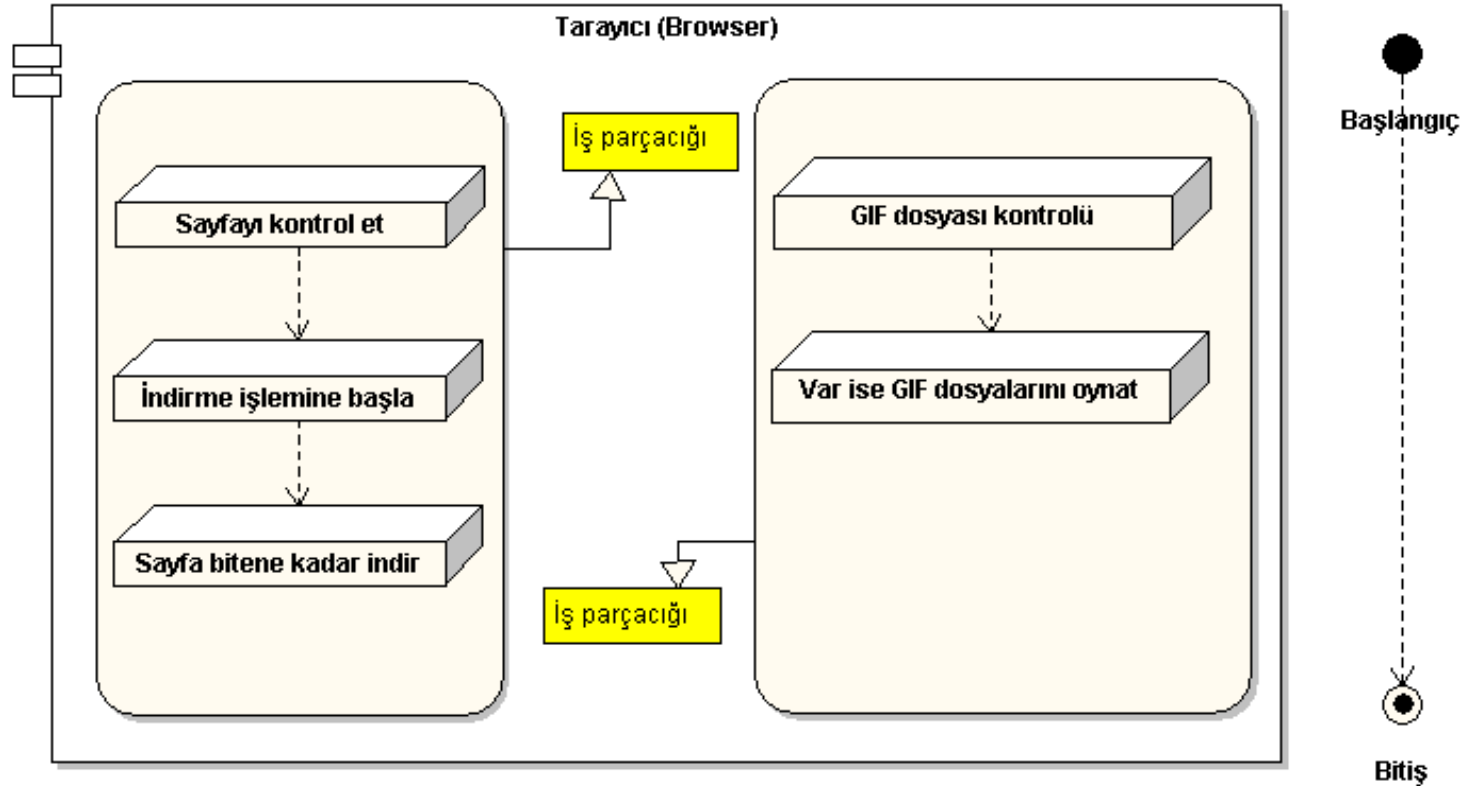
```
public class Selam{  
    public static void main(String args[]){  
        System.out.println("Selamlar !");  
    }  
}
```

# ÇOKLU İŞ PARÇACIKLARINA NE ZAMAN İHTİYAÇ DUYULUR ?

Bu durumlara en iyi örnek tarayıcılardır (browser).

İstenilen sayfanın indirilmesi için bir iş parçacığı

İndirilmiş olan GIF dosyalarını oynatmak için bir iş parçacığı



# ÇOKLU İŞ PARÇACIKLARINA NE ZAMAN İHTİYAÇ DUYULUR ?

Şimdi öyle bir uygulama düşünelim ki:

Bu uygulama bir dosyadan okuma yapsın,

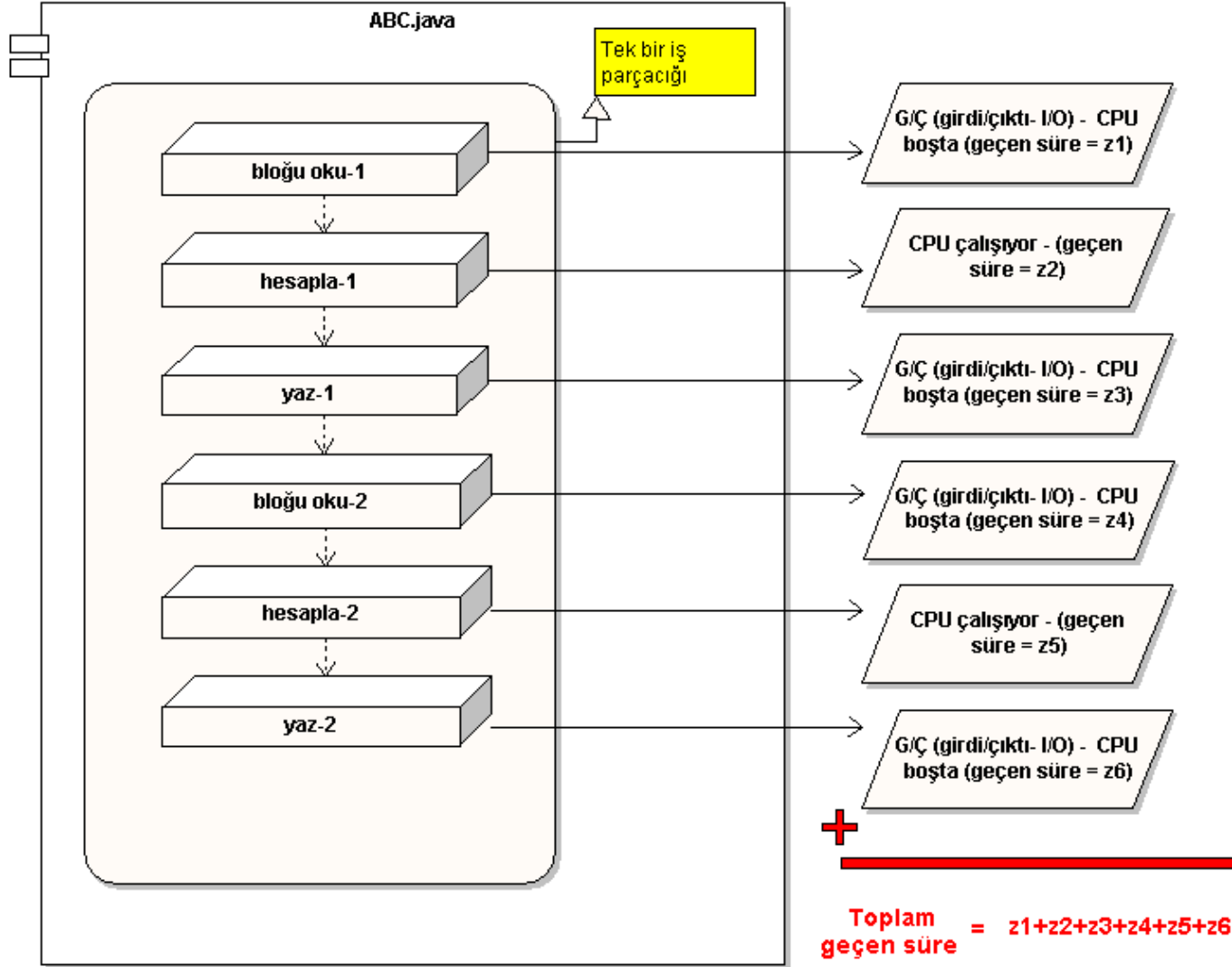
Okuduğu veri üzerinde hesaplama yapıp,

Hesaplamanın sonucunu başka bir dosyaya yazsın.

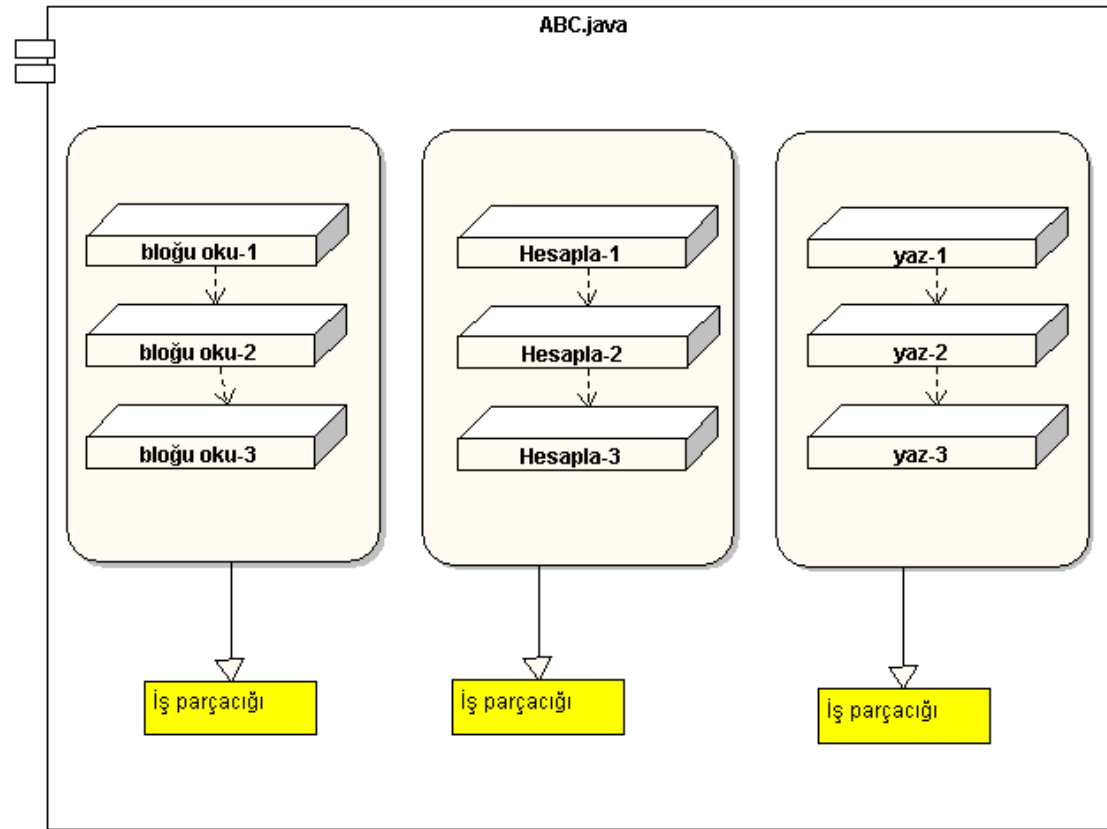
Burada kaç işlem den bahsediyoruz?

- 1.Dosyadan okuma yapma (G/Ç)
2. Okunan veri üzerinde hesaplama yapma (CPU çalışıyor)
- 3.Hesaplama sonucunu başka bir dosyaya yazma (G/Ç)

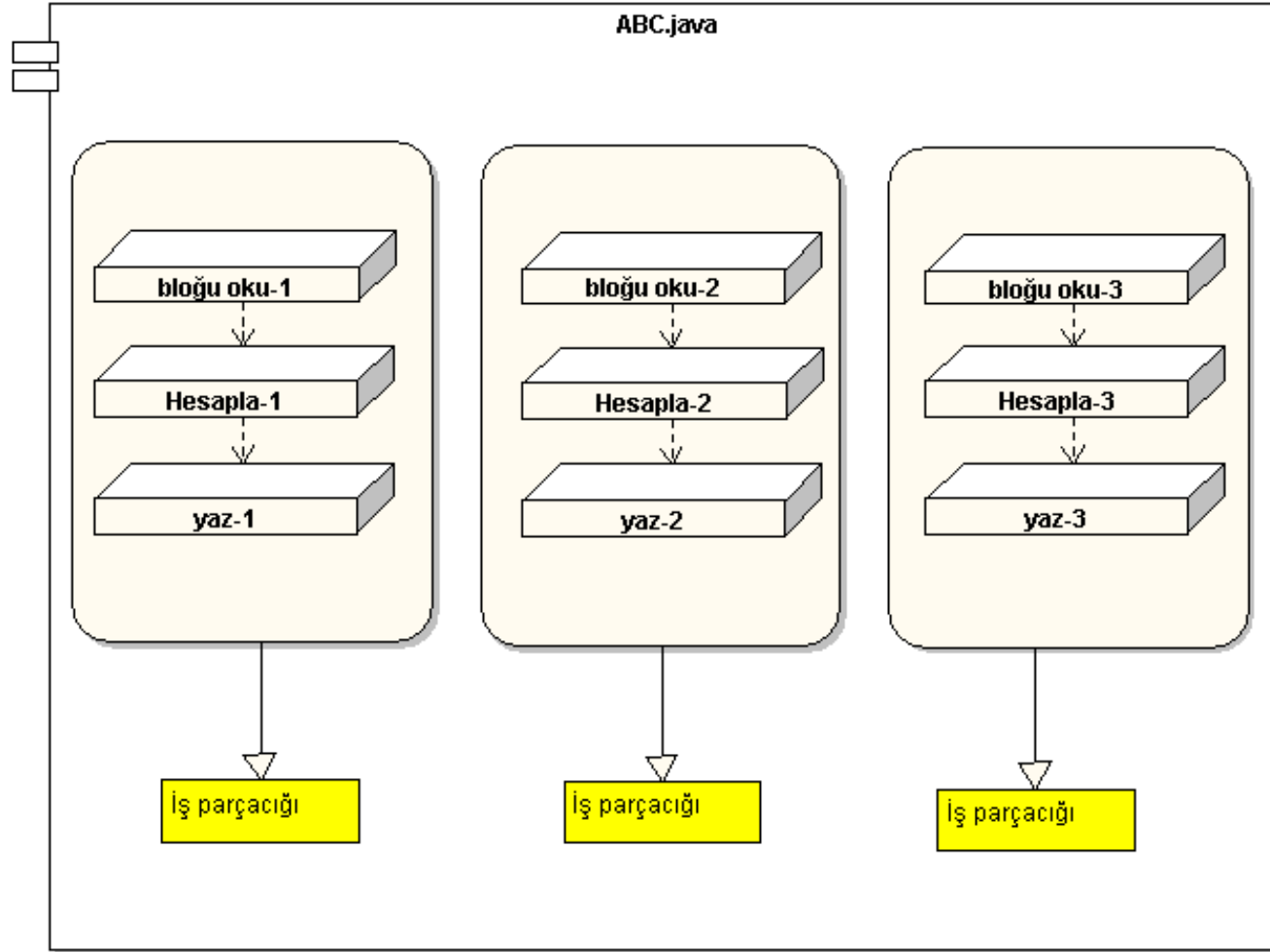
# TEK BİR İŞ PARÇACIĞINDAN OLUŞMUŞ UYGULAMANIN AŞAMALARI



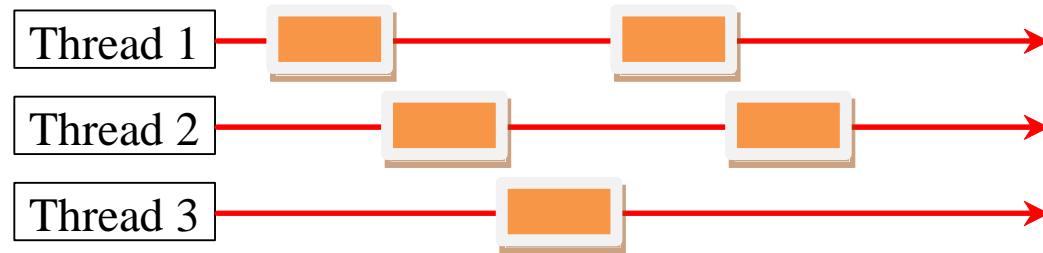
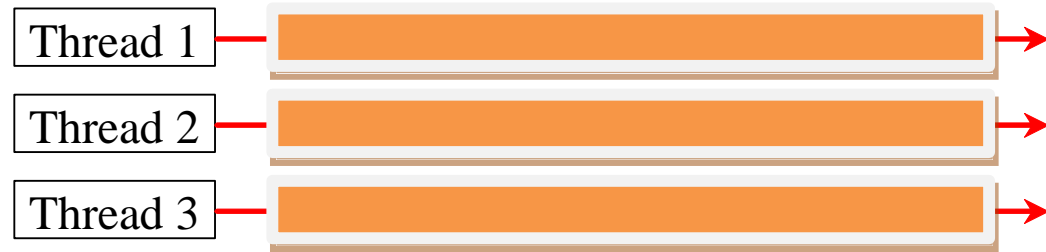
# TEK BİR İŞ PARÇACIĞINDAN OLUŞMUŞ UYGULAMANIN AŞAMALARI



# TEK BİR İŞ PARÇACIĞINDAN OLUŞMUŞ UYGULAMANIN AŞAMALARI



# İŞ PARÇACIĞI KAVRAMI





# GÖREV VE İŞ PARÇACIKLARININ OLUŞTURULMASI

Tek başına çalışabilen (standalone) uygulamaların başlangıç yeri statik **main()** yordamı(methods) olduğunu daha evvelden belirtmiştik.

Uygulama çalışmaya başladığında, ana işparçacığı oluşturulup olayların akışı başlatılır.

Java programlama dili ile yazdığımız uygulamaların içerisinde çoklu iş parçacıklarını kullanmak için `java.lang.Thread` sınıfını veya `java.lang.Runnable` arayüzünü kullanmamız gerekir.

*public void start():* iş parçacıklarının işlenmesini sağlar. `run()` metodunu çalıştırır.

*public void run():* iş parçacığının yapacağı işleri yerine getirir.

*public final void stop():* iş parçacığının durmasını sağlar.

# THREAD SINIFI

Seçenek 1: Thread sınıfının bir alt sınıfını bildirmek	Seçenek 2: Runnable ara yüzünü gerçekleştiren bir sınıfın örneğini geçirmek
<pre>class MyThread extends Thread {     ...     public void run(){         // görevi gerçekleştiren ifadeler     }     ... };  // MyThread in bir sınıf metodu içinde // bir örneğini yarat MyThread myThread = new MyThread(); ...</pre>	<pre>class MyTask implements Runnable{     ...     public void run(){         // görevi gerçekleştiren ifadeler     }     ... };  // MyThread in bir sınıf metodu içinde // bir örneğini yarat Thread myThread = new Thread(new MyTask());</pre>

Yaratılan is parçacığını yürütebilmek için **start()** yöntemi uyandırılmalı:  
**myThread.start();**

**Not:** Is parçacığını yürütmek için direk olarak **run()** metodunu çağırmayınız.  
Bu sıradan metot çağırısına karşılık gelecektir.

# GÖREV VE İŞ PARÇACIKLARININ OLUŞTURULMASI

`java.lang.Runnable`

TaskClass

```
// Custom task class
public class TaskClass implements Runnable {
    ...
    public TaskClass(...) {
        ...
    }

    // Implement the run method in Runnable
    public void run() {
        // Tell system how to run custom thread
        ...
    }
    ...
}
```

```
// Client class
public class Client {
    ...
    public void someMethod() {
        ...
        // Create an instance of TaskClass
        TaskClass task = new TaskClass(...);

        // Create a thread
        Thread thread = new Thread(task);

        // Start a thread
        thread.start();
        ...
    }
    ...
}
```

# GÖREV VE İŞ PARÇACIKLARININ OLUŞTURULMASI

Görevler nesnelerdir.

Bir görev Runnable arayüzünü implement etmelidir.

Runnable arayüzü run() metodunu içerir.

TaskClass task=new TaskClass(....);

Bir görevi başlatmak için

Thread t=new Thread(task); nesnesi oluşturulur.




Thread'i başlatmak için t.start() metodu yazılır.

Bu metot ile run() metodu çalıştırılır.

# ÖRNEK:

```
1  public class t1 implements Runnable{  
2      public void run() {  
3          System.out.println("Thread çalışıyor");  
4      }  
5      public static void main(String[] args) {  
6          t1 nesne=new t1();  
7          Thread t=new Thread(nesne);  
8          t.start();  
9      }  
10 }
```

Ekran Çıktısı:

Output - uygulamaGUI (run) X	
	run:
	Thread çalışıyor
	BUILD SUCCESSFUL (total time: 0 seconds)

# ÖRNEK: RUNNABLE ARAYÜZÜ İLE THREAD'LERİ OLUŞTURUP ÇALIŞTIRMAK

Amaç: Üç thread'in oluşturulup çalıştırılması:

İlk thread ekrana 100 kez a yazar.

İkinci thread ekrana 100 kez b yazar.

Üçüncü thread 1'den 100'e kadarki sayıları yazar.

# ÖRNEK:

```
package uygulama;

public class TaskDemo {

    public static void main(String[] args) {

        Runnable printA= new PrintChar('a',100);
        Runnable printB= new PrintChar('b',100);
        Runnable print100= new PrintNum(100);
        Thread t1=new Thread(printA);
        Thread t2=new Thread(printB);
        Thread t3=new Thread(print100);

        t1.start();

        t2.start();

        t3.start();

    }

}
```

```
class PrintChar implements Runnable {
    private char charToPrint;
    private int times;
    public PrintChar(char c,int t){
        charToPrint=c;
        times=t;
    }
    public void run(){
        for (int i = 0; i < times; i++) {
            System.out.print(charToPrint);

        }
    }
}
```

# ÖRNEK:

class PrintNum implements Runnable {

    private int lastNum;

    public PrintNum(int n){

        lastNum=n;

    }

    public void run(){

        for (int i = 0; i < lastNum; i++) {

            System.out.print(" "+i);

        }

    }

}

Ekran Çıktısı

```
obbbbbbbbbbbbbbbbbbaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaabbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
b 1 2
3aaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaa 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 50 51 52 53 54 55 56 57 58
59 60 61 62 63 64 65 66 67 68 69 70 71 72
73 74 75 76 77 78 79 80 81 82 83 84 85 86
87 88 89 90 91 92 93 94 95 96 97 98 99
```



# TASARIM -THREAD SINIFINDAN KALITIM

Bir sınıfa ait nesneyi iş parçacığına dönüştürmek için iki tasarım modeli bulunmaktadır.

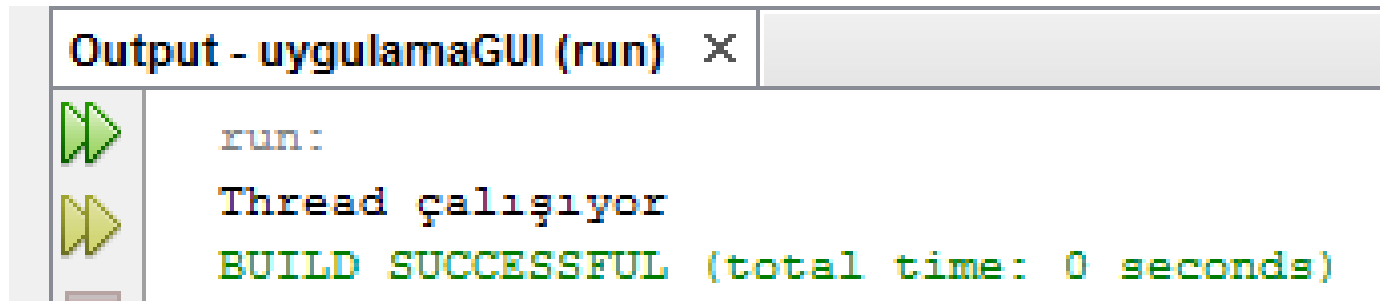
- Bunlardan ilki, şu ana kadar yaptığımız gibi ilgili sınıfı java.lang.Thread sınıfından türetmektir.

```
public class OrnekSinif extends Thread {  
  
    public void run() {  
        // ...  
    }  
  
}
```

# ÖRNEK:

```
1  public class t1 extends Thread{  
2      public void run() {  
3          System.out.println("Thread çalışıyor");  
4      }  
5      public static void main(String[] args) {  
6          t1 nesne=new t1();  
7          nesne.start();  
8      }  
9  }
```

Ekran çıktısı:



```
Output - uygulamaGUI (run) X  
run:  
Thread çalışıyor  
BUILD SUCCESSFUL (total time: 0 seconds)
```

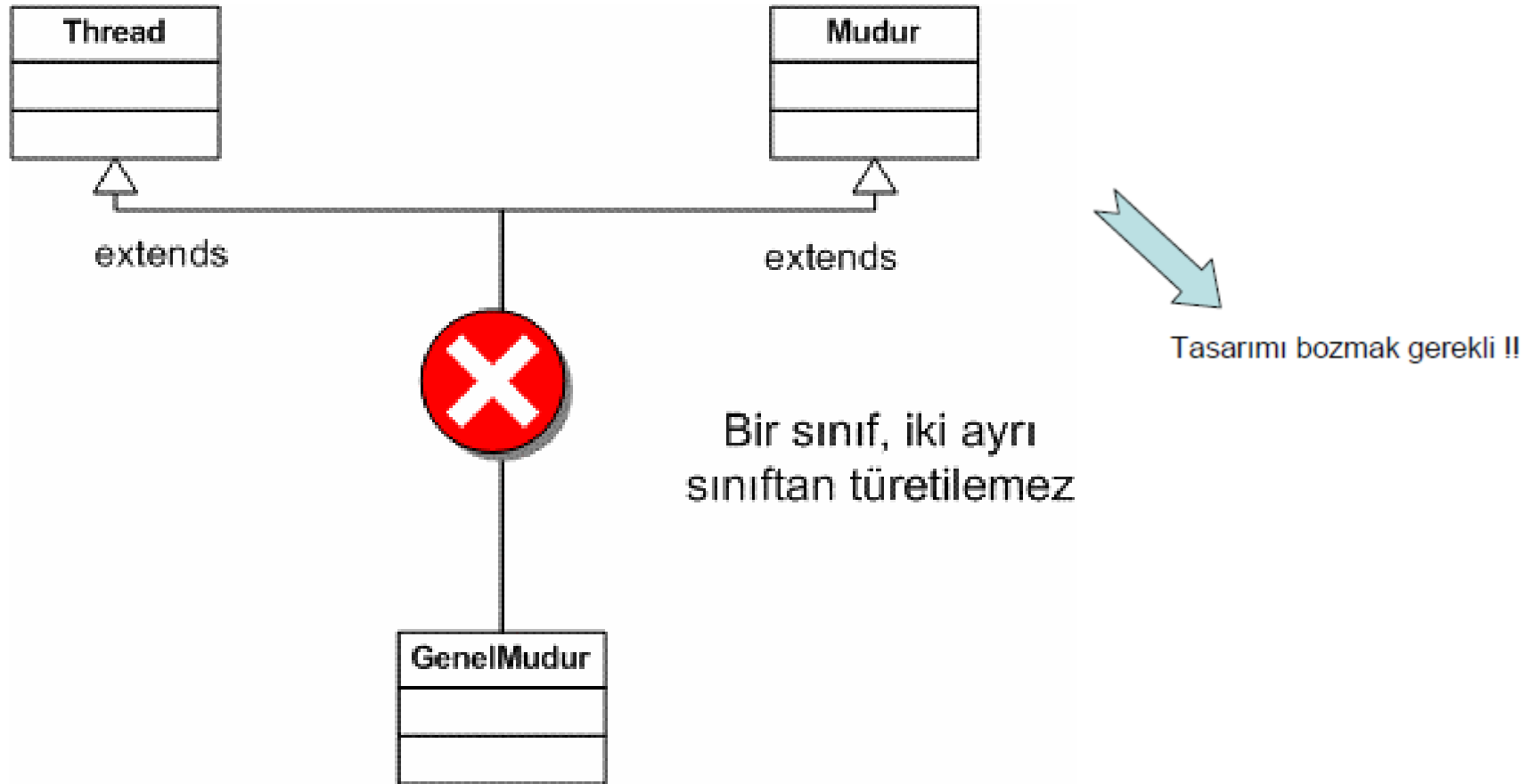
# TASARIM -THREAD SINIFINDAN KALITIMIN AVANTAJLARI

Bu tasarımın avantajı daha kolay kodlama denilebilir.

Örneğin **run()** yordamının içerisinde **getName()** yordamını direk çağırabiliriz.

# TASARIM -THREAD SINIFINDAN KALITIMIN DEZAVANTAJLARI

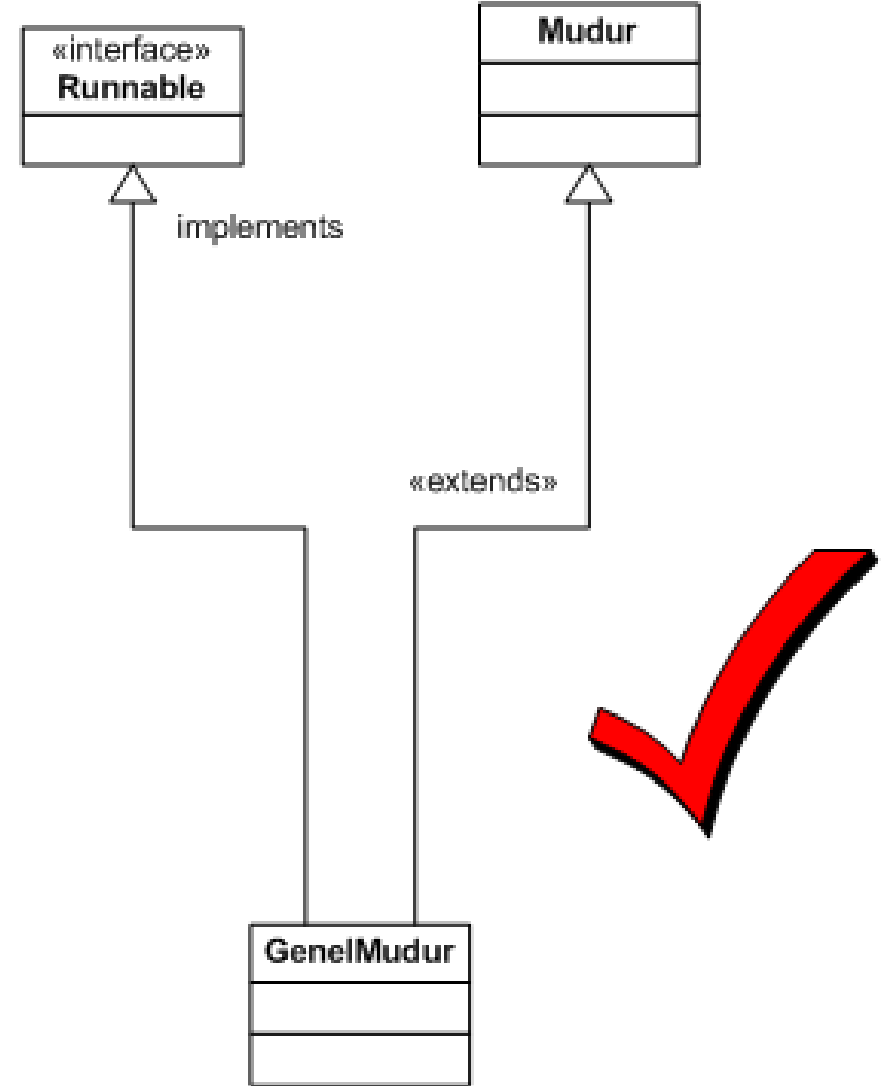
Java programlama dilinde bir sınıf ancak ve ancak tek bir diğer sınıftan türetilebildiği için (single inheritance) bu model kullanılarak tasarlanan iş parçacıklarında belirli kısıtlamalar gündeme gelebilir.



# RUNNABLE ARAYÜZÜ

Runnable arayüzü sayesinde bir sınıfı iş parçacığına dönüştürmek mümkündür.

Runnable arayüzünü kullanmanın dezavantajları olarak daha uzun kodlama denilebilir.



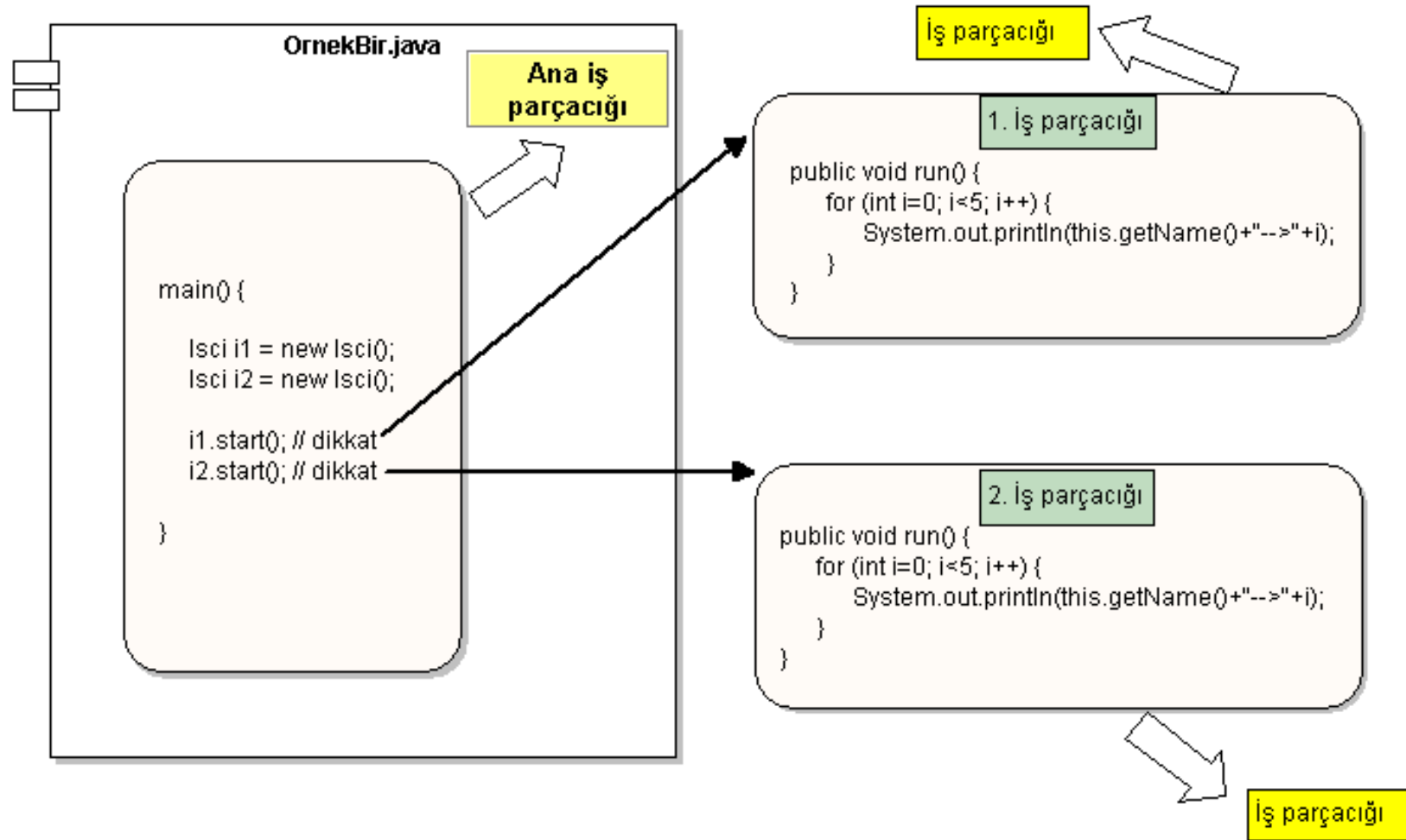
# ÖRNEK: AYNI THREAD'İ İKİ KEZ BAŞLATMAK

```
1  public class t1 extends Thread{
2      public void run() {
3          System.out.println("Thread çalışıyor");
4      }
5      public static void main(String[] args) {
6          t1 nesne=new t1();
7          nesne.start();
8          nesne.start();
9      }
10 }
```

Ekran çıktısı:

```
Output - uygulamaGUI (run) x
run:
Thread çalışıyor
Exception in thread "main" java.lang.IllegalThreadStateException
    at java.lang.Thread.start(Thread.java:684)
    at t1.main(t1.java:8)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

# ÖRNEKBİR.JAVA UYGULAMASININ ÇALIŞMASI



```
class Isci extends Thread {  
    public void run() {  
        for (int i=0; i<5; i++) {  
            System.out.println(this.getName()+"-->" +i);  
        }  
    }  
}  
  
public class OrnekBir {  
    public static void main(String args[]) {  
        Isci i1 = new Isci();  
        Isci i2 = new Isci();  
        i1.start(); // dikkat  
        i2.start(); // dikkat  
    }  
}
```



```

class Mudur {
    public void calis() {
        System.out.println("Mrb ben Mudur");
    }
}
class GenelMudur extends Mudur implements Runnable {
    public void calis() { // iptal etti- override
        System.out.println("Mrb ben Genel Mudur");
    }
    public void run() {
        try {
            for (int i=0; i<5; i++) {
                // this.sleep() ; //!hata
                // System.out.println(this.getName()+"<->" + i); //!hata
                Thread.currentThread().sleep(150);
                System.out.println(Thread.currentThread().getName()+"<->" + i);
            }
        } catch (InterruptedException iEx) {
            // bosver
        }
    }
}

```

## Output - uygulamaGUI (run) X

```

Thread-1<->1
Thread-0<->1
Thread-0<->2
Thread-1<->2
Thread-0<->3
Thread-1<->3
Thread-1<->4
Thread-0<->4

```

```

public class ArayuzTest1 {
    public static void main(String args[]) {
        GenelMudur gm1 = new GenelMudur();
        GenelMudur gm2 = new GenelMudur();
        Thread th1 = new Thread(gm1);
        Thread th2 = new Thread(gm2);
        th1.start();
        th2.start();
    }
}

```

# THREAD SINIFI

«interface»  
*java.lang.Runnable*



java.lang.Thread

- +Thread()
- +Thread(task: Runnable)
- +start(): void
- +isAlive(): boolean
- +setPriority(p: int): void
- +join(): void
- +sleep(millis: long): void
- +yield(): void
- +interrupt(): void

Varsayılan thread oluşturma.

Bir görev için bir thread oluşturma

JVM ile çağrılan run() metodunu çalıştırmak.

Thread'ın şu anda çalışıp çalışmadığını control etmek.

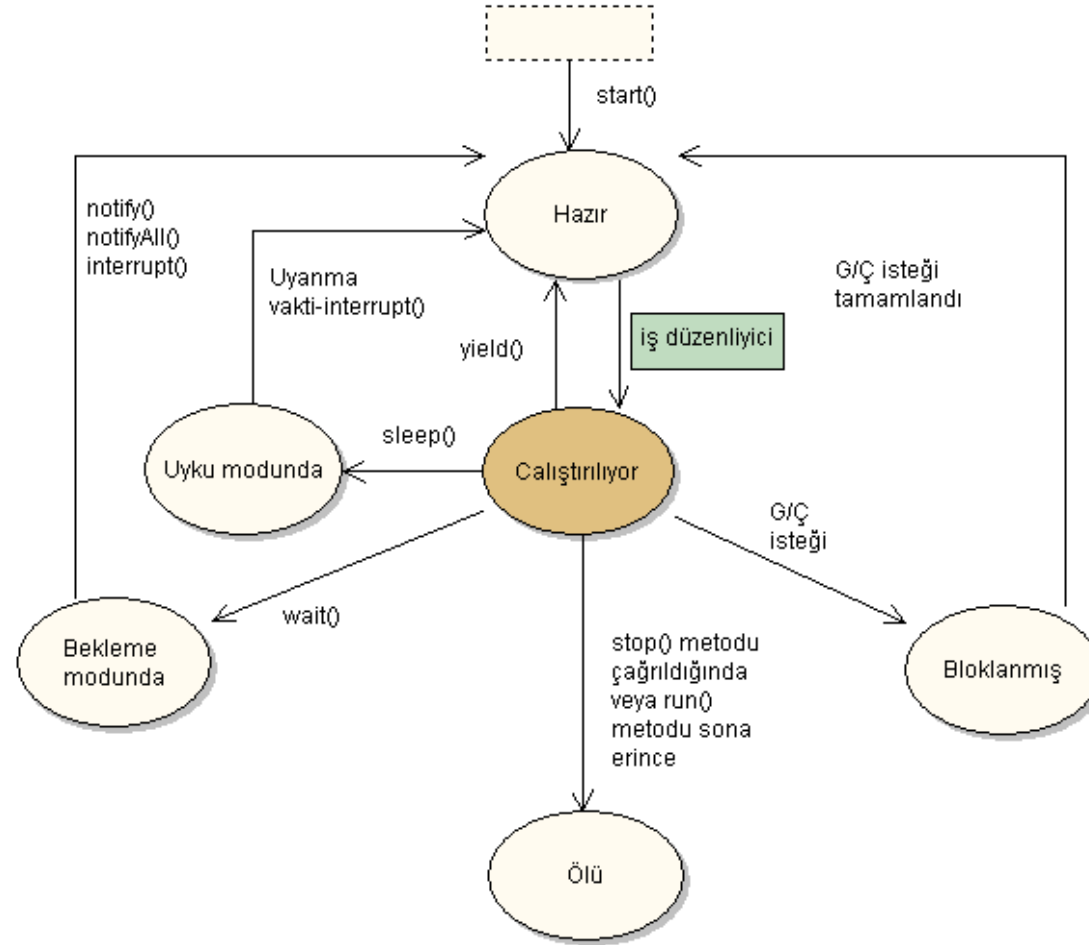
1 ile 10 aralığında thread'e öncelik verme.

Bu thread'in bitmesini bekleme.

Milisaniye olarak belirli bir süre için runnable nesnesini sleep modda gösterir.

Bu thread'i geçici olarak durdurur ve diğer thread'leri çalıştırır.

# HALLER



# ÖNCELİK SIRASI

Aynı öncelik sırasına sahip olan iş parçacıkları aynı hazır durum havuzunda bulunurlar.

**Thread.MIN\_PRIORITY = 1**

**Thread.NORM\_PRIORITY = 5**

**Thread.MAX\_PRIORITY = 10**

```

1 package uygulama;
2 class Robot extends Thread {
3     public Robot(String isim) {
4         super(isim);
5     }
6     public void run() {
7         try {
8             for (int i = 0; i < 5; i++) {
9                 System.out.println("Oncelik: " +
10                     this.getPriority() + "-" +
11                     this.getName() + "-->" + i);
12             }
13         } catch (Exception ex) {
14             System.out.println("Hata olustu -->" + ex);
15         }
16     }
17 }
18 public class Uygulama {
19     public static void main(String args[]) {
20         Robot r1 = new Robot("A");
21         Robot r2 = new Robot("B");
22         Robot r3 = new Robot("C");
23         Robot r4 = new Robot("D");
24         r1.setPriority(Thread.MIN_PRIORITY);
25         r2.setPriority(Thread.NORM_PRIORITY);
26         r3.setPriority(Thread.MAX_PRIORITY);
27         r4.setPriority(Thread.MAX_PRIORITY - 2); // 10-2 = 8
28         r1.start(); // dikkat
29         r2.start(); // dikkat
30         r3.start(); // dikkat
31         r4.start(); // dikkat
32     }
33 }

```

Oncelik: 1-A-->0  
 Oncelik: 10-C-->0  
 Oncelik: 10-C-->1  
 Oncelik: 10-C-->2  
 Oncelik: 10-C-->3  
 Oncelik: 1-A-->1  
 Oncelik: 8-D-->1  
 Oncelik: 5-B-->1  
 Oncelik: 8-D-->2  
 Oncelik: 1-A-->2  
 Oncelik: 10-C-->4  
 Oncelik: 1-A-->3  
 Oncelik: 8-D-->3  
 Oncelik: 5-B-->2  
 Oncelik: 8-D-->4  
 Oncelik: 1-A-->4  
 Oncelik: 5-B-->3  
 Oncelik: 5-B-->4

# İŞ PARÇACIKLARININ SONLANDIRILMASI

Bir işparçacığının sonlanması onun ölmesi anlamına gelir.





- Peki bir işparçacığı nasıl öldürebiliriz?

- Birinci yol ilgili işparçacığının **stop()** yordamını çağırarak gerçekleştirilebilir ama bu tavsiye edilmeyen bir yoldur.

- İkinci yol nasıl olabilir ?

# ÖRNEK: STOP() METODU

```
1  class C extends Thread{
2      public void run() {
3          for (int i = 1; i < 5; i++) {
4              if(i==3) stop();
5              System.out.println("A- "+i);
6          }
7          System.out.println("A den çıkış");
8      }}
9
10 public class t1 {
11     public static void main(String[] args) {
12         C nesnel=new C();
13         nesnel.start();
14     }}
```

Output - uygulamaGUI (run) X	
	run:
	A- 1
	A- 2
	BUILD SUCCESSFUL (total time: 0 seconds)

```

1  class Robot extends Thread {
2      private int donguSayisi;
3      public Robot(String isim, int donguSayisi) {
4          super(isim);
5          this.donguSayisi = donguSayisi;
6      }
7
8      public void run() {
9          try {
10             if (donguSayisi == 0) return; // is parcacigini sonlandir.
11             for (int i=0; i<donguSayisi; i++) {
12                 System.out.println(this.getName()+"-->" + i);
13             }
14             catch (Exception ex) { System.out.println("Hata olustu -->" + ex); }
15         }
16     }
17
18     public class Uygulama {
19         public static void main(String args[]) {
20             Robot r1 = new Robot("A", 2);
21             Robot r2 = new Robot("B", 3);
22             Robot r3 = new Robot("C", 3);
23             Robot r4 = new Robot("D", 2);
24             r1.start(); // dikkat
25             r2.start(); // dikkat
26             r3.start(); // dikkat
27             r4.start(); // dikkat
28         }
29     }
30 }

```

Output - uygulama (run) %

```

run:
A-->0
D-->0
C-->0
B-->0
B-->1
B-->2
C-->1
D-->1
A-->1
C-->2
BUILD SUCCESSFUL (total time: 0 seconds)

```



# YIELD() METODU

Fazla zaman tüketen bir iş parçacığı, **Thread** sınıfının **yield()** metodunu çağırarak diğer iş parçacıklarına yürütölmeleri için izin verebilir.

Eğer bir iş parçacığı **yield()** metodunu çağırırsa, programlayıcı güncel olan iş parçacığını durdurup bekleyenlerden birine izin verebilir.

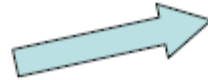
Bu nedenle, **yield()** metodunu çağıran iş parçacığı programcının tekrar kendisine dönmesi için bekleyecektir.

# YIELD() KULLANIMI

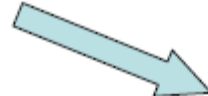
Bir iş parçacığı çalıştırılıyor halindeken, bu iş parçacığı ile aynı öncelik sırasına sahip başka bir iş parçacığına çalıştırılma fırsatı vermek istiyorsak **yield()** yordamını kullanmamız gereklidir.



*YieldOrnek.java*



Normal bir şekilde çalıştıralım...



Şimdi 9. satırındaki yorum satırını açalım, derleyelim ve çalıştıralım...

```

1 class YieldOrnek extends Thread {
2     public void run() {
3         for (int i=0 ; i<5; i++) {
4             System.out.println(this.getName()+"<->" + i);
5             Thread.yield(); // 9
6             // this.yield(); // bu da dogru
7         }
8     }
9     public static void main(String args[]) throws Exception {
10         YieldOrnek yo1 = new YieldOrnek();
11         YieldOrnek yo2 = new YieldOrnek();
12         // tum is parcaciklari ayni oncelik sirasina sahip = 5
13         yo1.start();
14         yo2.start();
15     }
16 }
17
18

```

#### Output - uygulama (run)



run:



Thread-1<->0



Thread-0<->0



Thread-1<->1



Thread-0<->1

Thread-1<->2

Thread-0<->2

Thread-1<->3

Thread-0<->3

Thread-1<->4

Thread-0<->4

Yield kullanarak

#### Output - uygulama (run)



run: Yield kullanmadan



Thread-0<->0



Thread-1<->0



Thread-0<->1



Thread-1<->1

Thread-0<->2

Thread-1<->2

Thread-0<->3

Thread-1<->3

Thread-0<->4

Thread-1<->4

# ÖRNEK:

Kod	Çıktı yield() kullanmadan	Çıktı yield() kullanarak
<pre>public class MyThread extends Thread {     public MyThread(String name){         super(name);     }     public void run(){         for(int i=0; i&lt;3; i++){             System.out.println(getName());             yield();         }     }     public static void main(String[] args) {         MyThread t1 = new MyThread("t1");         MyThread t2 = new MyThread("t2");         t1.start();         t2.start();     } }</pre>	t1 t1 t1 t2 t2 t2 t2	t1 t2 t1 t2 t1 t2

# ÖRNEK:

```
1  class A extends Thread{
2      public void run(){
3          for (int i = 1; i < 5; i++) {
4              if(i==2) yield();
5              System.out.println("A- " + i);
6          }
7          System.out.println("A den çıkış");
8      }}
9  class B extends Thread{
10     public void run(){
11         for (int i = 1; i < 5; i++) {
12             System.out.println(" B-" + i);
13         }
14         System.out.println("B den çıkış");
15     } }
16  public class t1 {
17     public static void main(String[] args) {
18         A nesne1=new A();
19         B nesne2=new B();
20         nesne1.start();
21         nesne2.start();
22     }
```

Output - uygulamaGUI (run) ×

```
run:
B-1
A- 1
B-2
A- 2
B-3
A- 3
B-4
A- 4
B den çıkış
A den çıkış
BUILD SUCCESSFUL (total time: 0 seconds)
```

# İŞ PARÇACIKLARININ KONTROLÜ

**sleep()** :Çalışan iş parçacığının belirli bir süre uyumasını sağlar. Bu statik bir yordamdır; yani bu **sleep()** yordamını çağırmak için java.lang.Thread sınıfından türemiş alt bir sınıfın **new()** anahtar kelimesi ile oluşturulması gerekmez.

```

1  public class t1 extends Thread {
2      int i;
3      public t1(int a1) {
4          i = a1;
5      }
6      public void run() {
7          try {
8              while (true) { // sonsuz dongu
9                  i++;
10                 System.out.println("uyuyor....");
11                 Thread.sleep(60 * 10);
12                 if (i == 5) {
13                     System.out.println("uyandı");
14                     return;
15                 }
16             } catch (InterruptedException iEx) {}
17         }
18         public static void main(String args[]) throws Exception {
19             t1 ut = new t1(0);
20             ut.start();
21         }
22     }

```

Output - uygulamaGUI (run) X

```

run:
uyuyor....
uyuyor....
uyuyor....
uyuyor....
uyuyor....
uyandı
BUILD SUCCESSFUL (total time: 3 seconds)

```

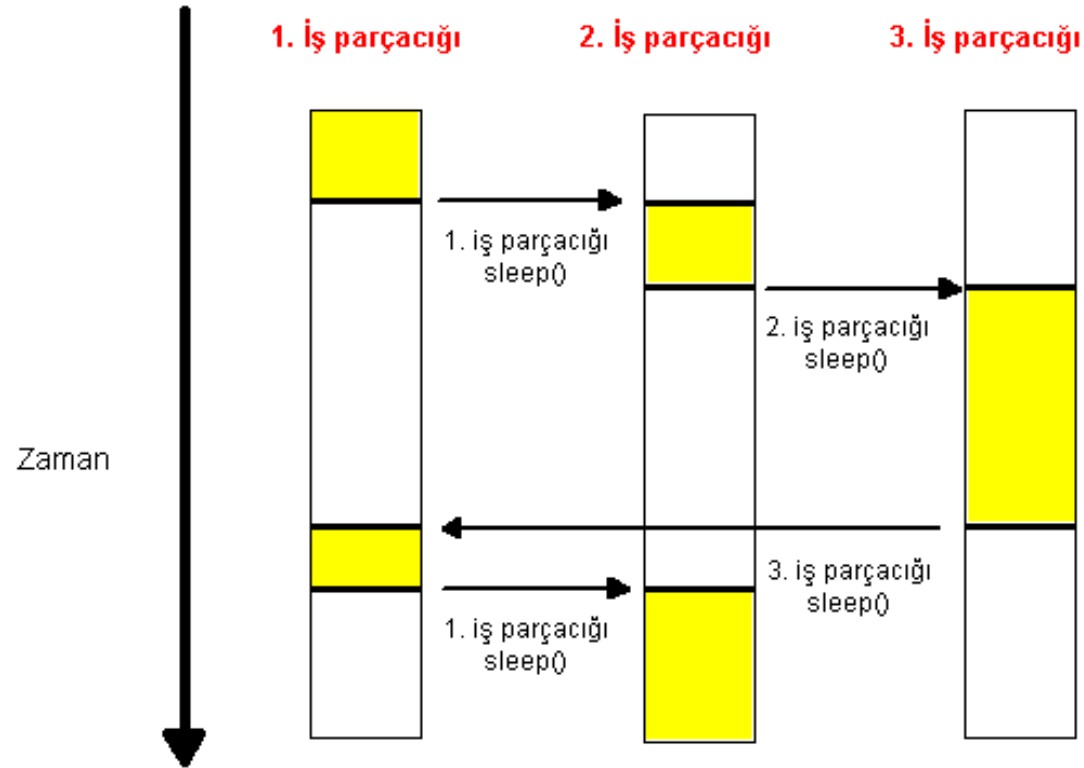
**interrupt()** :Uyuyan bir iş parçacığını uyandırmanın yolu onu rahatsız etmektir. Bu rahatsızlık verme olayını interrupt() yordamını çağırarak başarabiliriz.

```
public class UyanmaVakti extends Thread {  
    public void run() {  
        try {  
            for (;;) { // sonsuz dongu  
                System.out.println("uyuyor....");  
                Thread.sleep(60 * 10);  
            }  
        } catch (InterruptedException iEx) {  
            // bosver  
        }  
    }  
}  
  
    public static void main(String args[]) throws Exception{  
    UyanmaVakti uv = new UyanmaVakti();  
    uv.start();  
    uv.interrupt(); // dikkat  
    }  
}
```



# SLEEP() YORDAMI

Elimizde 3 adet iş parçacığı olduğunu ve bu üç iş parçacığının da aynı anda başlatıldıklarını hayal edelim...



Koyu alanlar iş parçacıklarının çalıştıklarını zamanı gösterir.

```

class Robot extends Thread {
    public Robot(String isim) {
        super(isim);
    }
    public void run() {
        try {
            String isim = this.getName();
            for ( int i=0; i<5; i++ ) {
                if ( ( isim.equals("Robot1") && (i==3) ) ) {
                    System.out.println(isim+"-> uyutuluyor");
                    Thread.sleep(100);
                } else if ( (isim.equals("Robot2") && (i==2) ) ) {
                    System.out.println(isim+"-> uyutuluyor");
                    Thread.sleep(150);
                } else if ( (isim.equals("Robot3") && ( i==4)) ) {
                    System.out.println(isim+"-> uyutuluyor");
                    Thread.sleep(250);
                }
                System.out.println(isim+"-->" +i);
            }
        } catch ( InterruptedException iEx ) {
            System.out.println("Hata olustu -->" + iEx);
        }
    }
}

```

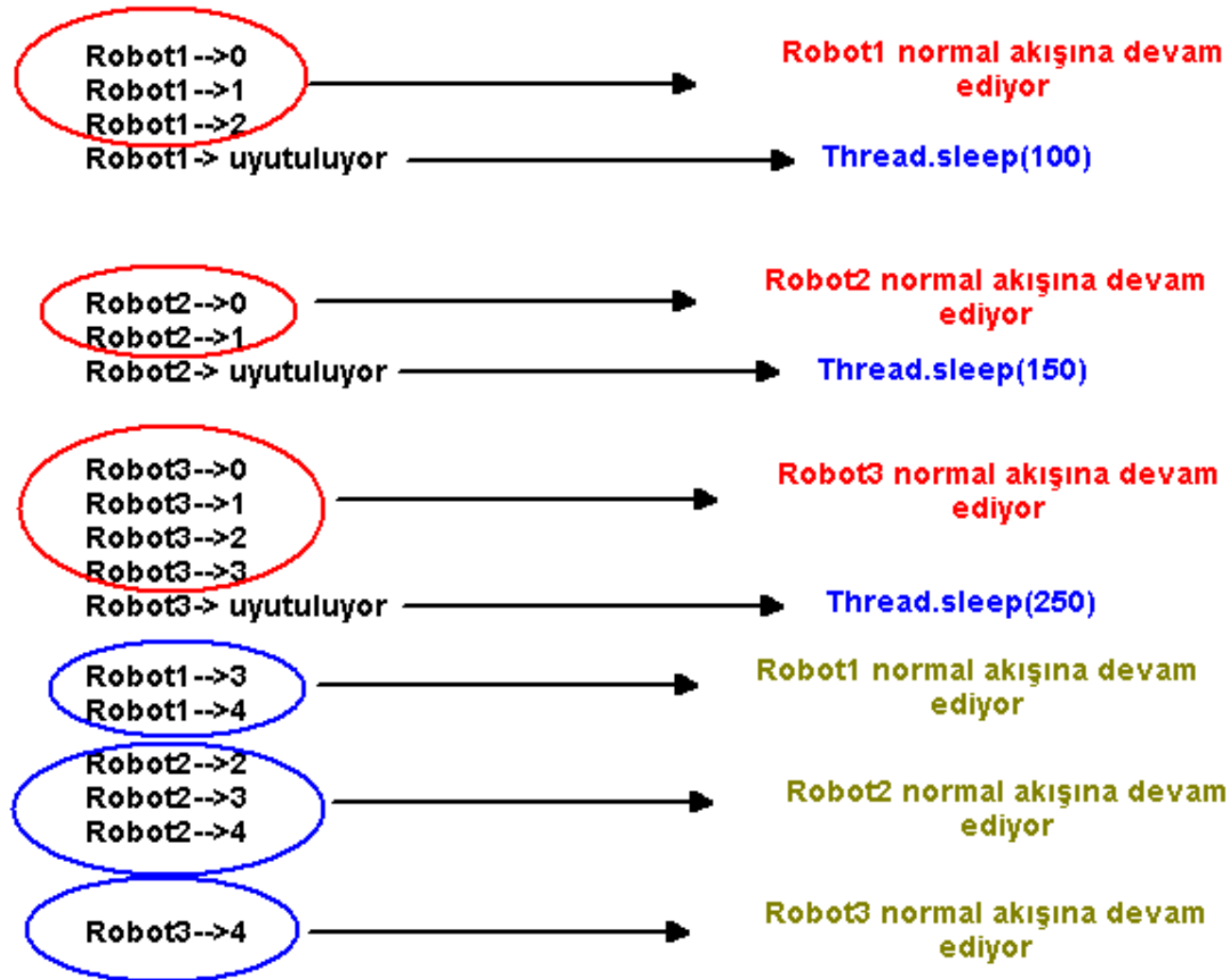
```

public class UyumGezer {
    public static void main(String args[]) {
        Robot r1 = new Robot("Robot1");
        Robot r2 = new Robot("Robot2");
        Robot r3 = new Robot("Robot3");

        r1.start(); // dikkat
        r2.start(); // dikkat
        r3.start(); // dikkat
    }
}

```

# EKRAN ÇIKTISI



# STATIC SLEEP(MILLISECONDS) METODU

Sleep (long mills) metodu milisaniye olarak belirtilen süre boyunca thread'i uyku moduna sokar.

Örneğin, daha önceki örnekte run() metodu aşağıdaki gibi değiştirdiğimizi varsayalım.

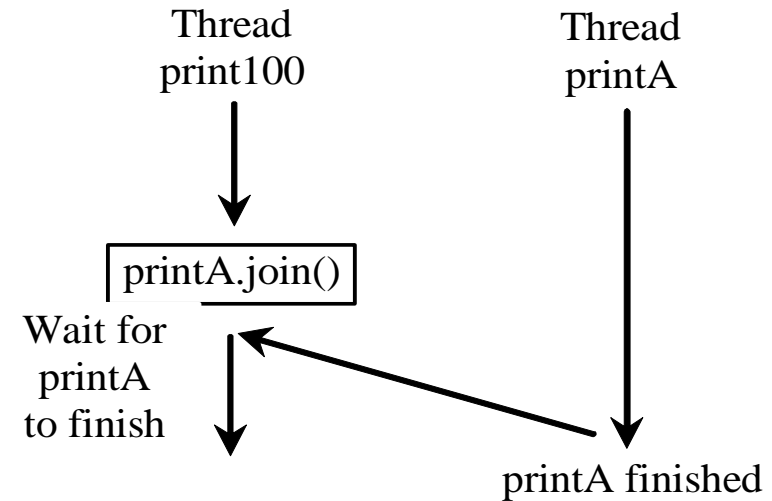
```
public void run() {  
    for (int i = 1; i <= lastNum; i++) {  
        System.out.print(" " + i);  
        try {  
            if (i >= 50) Thread.sleep(1);  
        }  
        catch (InterruptedException ex) {  
        }  
    }  
}
```

Sayının 50'den büyük veya eşit olduğu her durumda print100 threadi 1 milisaniye boyunca uyku moduna girer.

# JOIN() METODU

Bir thread'in diğer thread'in bitmesini beklemesine zorlayabilirsiniz. Örneğin, run() metodunu aşağıdaki şekilde değiştirdiğimizi varsayalım.

```
public void run() {  
    Thread thread4 = new Thread(  
        new PrintChar('c', 40));  
    thread4.start();  
    try {  
        for (int i = 1; i <= lastNum; i++) {  
            System.out.print(" " + i);  
            if (i == 50) thread4.join();  
        }  
    }  
    catch (InterruptedException ex) {  
    }  
}
```



# JOIN METODU

`join()` yordamı, bir iş parçacığının diğer bir iş parçacığını beklemesi için kullanılır.

- `join()` yordamının üç adaş yordamı (overloaded) bulunur.

<code>join()</code>	Belirtilen iş parçacığı bitene (ölü haline gelene kadar) kadar bekletir.
<code>join(long milisaniye)</code>	Belirtilen iş parçacığını, verilen milisaniye kadar bekletir.
<code>join(long milisaniye, int nanosaniye)</code>	Belirtilen iş parçacığını, verilen milisaniye + nano saniye kadar bekletir.

```

class Robot775 extends Thread {
    private Robot337 r337;
    public Robot775( String isim, Robot337 r337) {
        super(isim);
        this.r337 = r337 ;
    }
    public void run() {
        try {
            System.out.println(this.getName()+" beklemeye basliyor");
            r337.join(); // Robot337 bitene kadar bekle
            //r337.join(150); // Robot337 yi 150 ms bekle
            //r337.join(150, 90); // Robot337 yi 150 ms + 90 ns bekle
            for (int i=0; i<5; i++) {
                System.out.println(this.getName()+"<->" +i);
            }
        } catch (InterruptedException iEx) {}
    }
}

class Robot337 extends Thread {
    public Robot337(String isim) {
        super(isim);
    }
    public void run() {
        for (int i=0; i<5; i++) {
            System.out.println(this.getName()+"<->" +i);
        }
    }
}

```

```

public class JoinTest {
    public static void main(String args[]) {
        Robot337 r337 = new Robot337("Robot337");
        Robot775 r775 = new Robot775("Robot775", r337);
        r775.start();
        r337.start();
    }
}

```

# NESNENİN KİLİDİ

Her nesnenin kendisine ait bir kilidi bulunur.

Bir sınıfa ait bir nesne oluşturulunca bu kilit otomatik olarak oluşur.

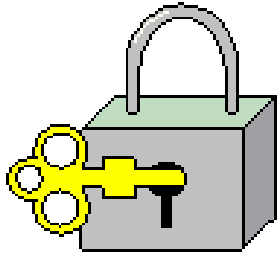
Bu kilidi eline geçiren iş parçacığı, kritik alan üzerinde işlem yapmaya hak kazanır.

Kritik alan, birden fazla iş parçacığının aynı anda üzerinde işlem yapmaması gereken bölgedir.



# NESNENİN KİLİDİ

SerbestPazar nesnesi  
(this)



Gerekli  
metodlar ve  
alanlar

Sadece nesnenin  
kilitine sahip olan iş  
parçacığı buraya  
girebilir

```
public void veriKoy(int gelenVeri) {  
    synchronized(this) {  
        // Aynı anda tek bir iş parçacığı erisebilir  
    }  
}
```

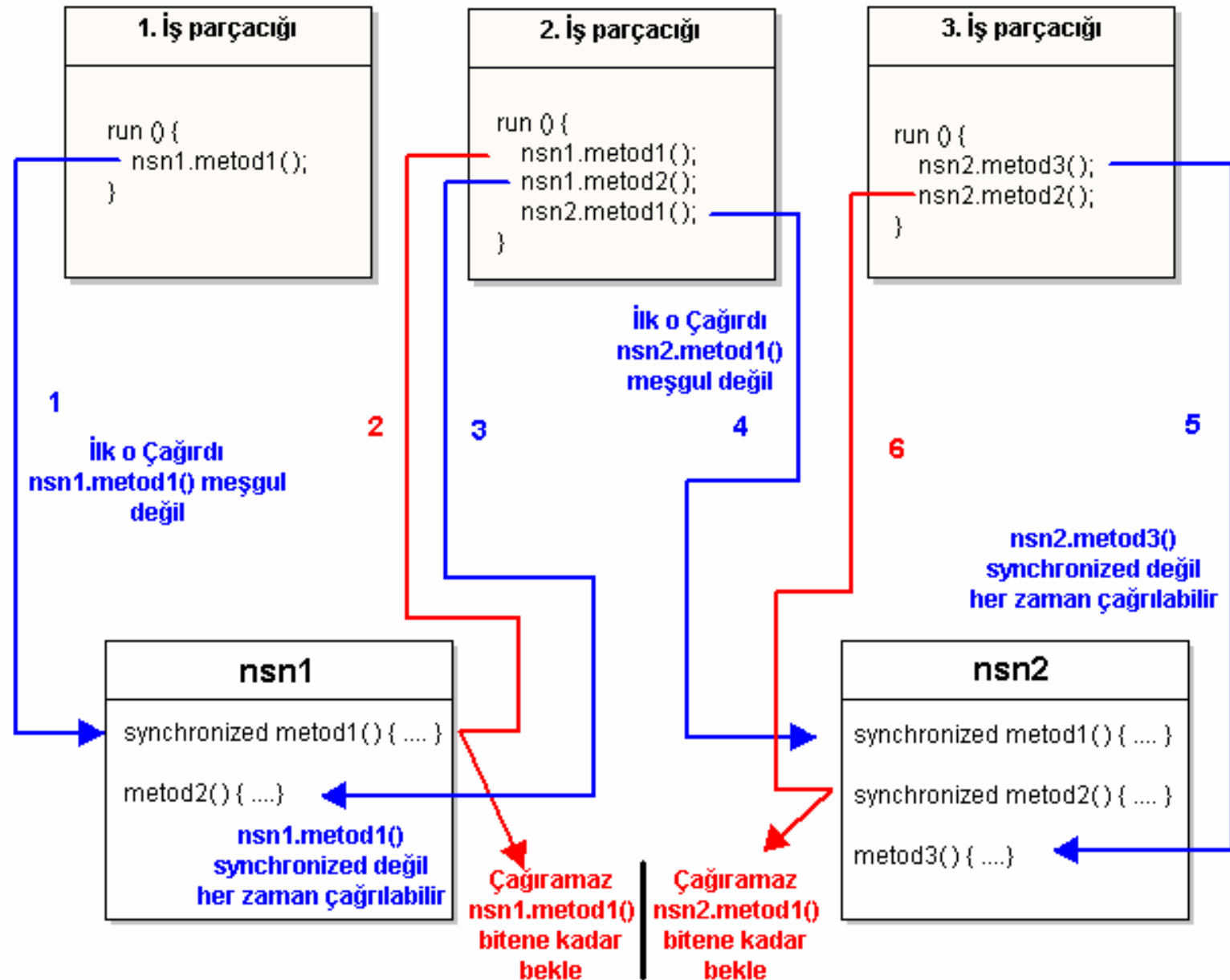
# SYNCHRONIZED ANAHTAR KELİMESİ -I

- Bir yordam veya yordamın içerisindeki kritik bir bölge *synchronized* anahtar kelimesi ile korunma altına alınabilir.
- Korunma altına alınmaktan kasıt, aynı anda iki veya daha fazla iş parçacığının bu kritik bölgeye veya yordamın komple kendisine erişmesini engellemektir.

```
public synchronized void veriKoy(int gelenVeri) {  
    // tum yordam koruma altinda  
    // ayni anda bir tek is parcacigi erisebilir  
}
```

```
public void veriKoy(int gelenVeri) {  
    synchronized(this) {  
        // sadece belirli bir kisim koruma altinda  
    }  
}
```

# SYNCHRONIZED ANAHTAR KELİMESİ -I



# ÖRNEK:

```
1  class A implements Runnable {
2      public void toplama(int i){
3          for (int n = 0; n < 5; n++)
4              System.out.println(Thread.currentThread().getName() +
5                  ":"+(i+n));
6      }
7      public void run(){
8          toplama(10);
9      }}
10 public class t1 {
11     public static void main(String[] args) {
12         A a=new A();
13         Thread t1=new Thread(a);
14         Thread t2=new Thread(a);
15         t1.setName("T1");
16         t2.setName("T2");
17         t1.start();
18         t2.start();
19     }
20 }
```

```
2  public synchronized void toplama(int i){
3      Thread t=Thread.currentThread();
4      for (int n = 0; n < 5; n++)
5          System.out.println(t.getName()+":"+ (i+n));
6  }
```

## Output - uygulamaGU

```
run:
T1:10
T2:10
T1:11
T2:11
T1:12
T2:12
T1:13
T2:13
T1:14
T2:14
```

## Output - uygulamaGUI (run)

```
run:
T1:10
T1:11
T1:12
T1:13
T1:14
T2:10
T2:11
T2:12
T2:13
T2:14
```

```
Public class FotokopiMakinasi{  
    public synchronized void kopyalariAl(Dokuman d, int kopyaSayisi){  
        //ayni anda yalnica tek bir iş parçacığı kopyalama yapabilir  
    }  
    public void kagitYukle(){  
        //birden fazla iş parçacığı buraya erişebilir.  
        synchronized (this){  
            //ayni anda yalnizca tek bir is parcacigi buraya ersebilir  
        }  
    }  
}
```

# WAIT(), NOTIFY() VE NOTIFYALL() YORDAMLARI

Her nesnenin bir kilidi olduğu gibi bir de bekleme havuzu(object's monitor)bulunur.

Bu bekleme havuzuna iş parçacıkları atılır -**wait()**-veya bu havuzdan dışarı çıkartılır -**notify()/notifyAll()**-

Bu beş yordam (**wait()**yordamının iki de adaş yordamı bulunur), java.lang.Object nesnesinin içerisinde bulunur.

# SEMAFOR (SEMAPHORE)

Kaynağın az olduğu durumlarda bir çok iş parçacığı arasında bir düzen sağlamak gereklidir.

```
public class Semaphore {
    private int sayi;
    public Semaphore(int n) {
        this.sayi = n;
    }
    public synchronized void eleGecir() {
        while (sayi == 0) {
            System.out.println("* -> " + Thread.currentThread().getName() + " beklemede" );
            try {
                wait();
            } catch (InterruptedException e) {
                //denemeye devam et
            }
        }
        System.out.println("* -> " + Thread.currentThread().getName() + " baglantiyi aldi" );
        sayi--;
    }
    public synchronized void bırak() {
        System.out.println("+ -> " + Thread.currentThread().getName() + " birakti...." );
        sayi++;
        notify(); // bekleme havuzundaki is paracıklarından birini disariya al
    }
}
```



```
class VeriTabani {  
    // kendisine ayni anda ancak 1 tane baglanti veriyor  
    public void baglantiVer() { } ;  
    public void baglantiKopart() { } ;  
}  
class Uygulamalar extends Thread {  
    Semaphore s;  
    VeriTabani vt ;  
    public Uygulamalar(Semaphore s, VeriTabani vt) {  
        this.s = s ;  
        this.vt = vt;  
    }  
    public void run() {  
        s.eleGecir() ; // dikkat  
        vt.baglantiVer(); // kiritik alan  
        // .. gerekli calismayi yap  
        try {  
            sleep( ( (int) Math.random()) * 100 );  
        } catch (InterruptedException iEx) {  
            // bosver  
        }  
        vt.baglantiKopart();  
        s.birak(); // dikkat  
    }  
}
```

```
public class SemaphoreTest {  
    Semaphore s ;  
    public SemaphoreTest(int sayi) {  
        VeriTabani vt = new VeriTabani();  
        Semaphore s = new Semaphore(sayi);  
        Uygulamalar u1 = new Uygulamalar(s, vt);  
        Uygulamalar u2 = new Uygulamalar(s, vt);  
        Uygulamalar u3 = new Uygulamalar(s, vt);  
        Uygulamalar u4 = new Uygulamalar(s, vt);  
        u1.start();  
        u2.start();  
        u3.start();  
        u4.start();  
    }  
    public static void main(String args[]) throws Exception {  
        SemaphoreTest s = new SemaphoreTest(2); // 61  
    }  
}
```