

Görsel Programlama

DERS 11

ÇOKLU KULLANIM(Multi Threading)

Günümüz işletim sistemleri *çok görevli(multi tasking)* dir. Aynı anda farklı programlar çalışmakta, görevler yerine getirilmektedir.

Çoklu kullanım da çok görevliliğe benzemektedir, aynı anda bir programın farklı bölümleri çalışabilmektedir.

Aynı program içerisindeki birbirinden ayrı ve bağımsız çalışabilen alt bölümlerin her birine *iş parçacığı (thread)* denilir.

ÇOKLU KULLANIM(Multi Threading)

Bu iş parçacıkları(*thread*) sistemde tek bir işlemci var ise aynı işlemci üzerinde çok kısa sürelerde arka arkaya çalıştırılırlar, böylece işlemler aynı anda yapılıyormuş gibi algılanır.

Birden fazla işlemci var ise bu iş parçacıkları farklı işlemcilerde çalıştırılırlar.

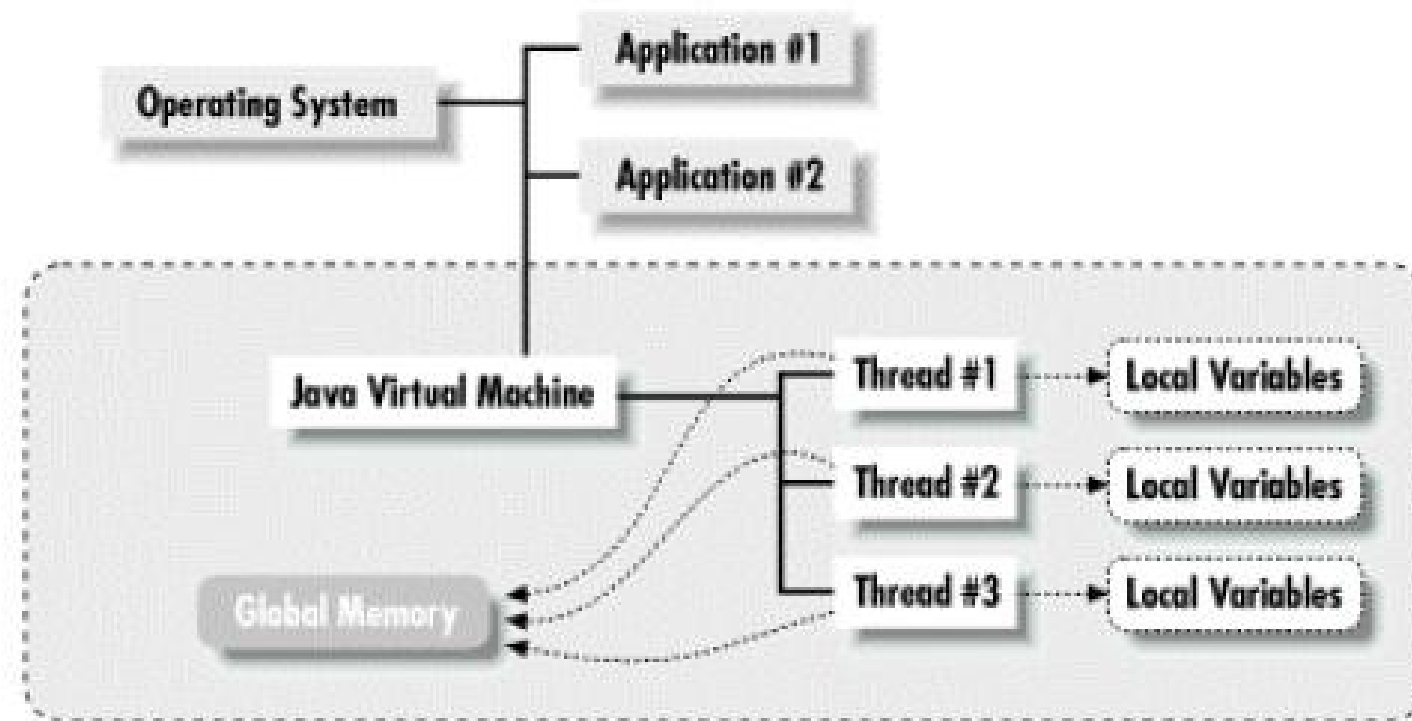
Kendi uygulamamızda da çeşitli zamanlarda iş parçacıklarını kullanmamız gerebilir; örneğin kullanıcıya internetten indirilen bir resim gösterilecektir. Bu resim bir iş parçacığı ile indirilirken kullanıcı uygulamadaki farklı işlemleri yapabilir.

ÇOKLU KULLANIM(Multi Threading)

Java da iş parçacıklarını iki şekilde yazabiliriz. Birinci seçenek; yazdığımız sınıfın *java.lang.Thread* sınıfından extends ile türetebiliriz.

Bazı durumlarda sınıfımızın başka bir sınıftan türetilmesi zorunlu olabilir, bu gibi durumlarda sınıf tek bir sınıftan türetilebileceği için *java.lang.Runnable* interface i kullanılmaktadır.

Multithreading Paradigm



Thread sınıfının Temel Metotları

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.

```

RunnableOrnek.java X
package com.comu.ce;

public class RunnableOrnek implements Runnable{

    public void run() {
        System.out.println("Runnable arayüzü");
    }

    public static void main(String[] args) {
        Runnable T = new RunnableOrnek();
        Thread myParca = new Thread(T);
        myParca.start();
    }
}

```

İş parçacığının çalışabilmesi için start() metodunun çalıştırılması gereklidir.

```

ThreadOrnek.java X
package com.comu.ce;

public class ThreadOrnek extends Thread{

    @Override
    public void run() {
        System.out.println("Thread iş parçacığı");
    }

    public static void main(String[] args) {
        ThreadOrnek myParca = new ThreadOrnek();
        myParca.start();
    }
}

```

İş Parçacığının(Thread) belirli bir süre beklemesi

```
MesajYaz.java X
package com.comu.ce;

public class MesajYaz {
    public static void main(String[] args) {
        String mesajlar[] = { "mesaj1", "mesaj2", "mesaj3" };
        for (int i = 0; i < mesajlar.length; i++) {
            try {
                Thread.sleep(4000); //4 saniye bekleme
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println(mesajlar[i]);
        }
    }
}
```



```
package com.comu.ce;

public class SayiYazici extends Thread {
    @Override
    public void run() {
        for (int i = 0; i < 3; i++) {
            System.out.println(this.getName() + ":" + i);
        }
    }

    public static void main(String[] args) {
        SayiYazici isparcacigi1 = new SayiYazici();
        SayiYazici isparcacigi2 = new SayiYazici();
        SayiYazici isparcacigi3 = new SayiYazici();
        isparcacigi1.start();
        isparcacigi2.start();
        isparcacigi3.start();
    }
}
```

İş Parçacıklarının Öncelikleri

İş parçacıklarına *öncelik(priority)* vererek o iş parçacığının önceliği ile orantılı çalışmasını sağlayabiliriz.

Verilebilen öncelikler 1-10 arasındadır. 1 en düşük, 10 en yüksek öncelik değeridir.

public final int getPriority(): iş parçacığının önceliğini geri çevirir.

public final void setPriority(int yeniOncelik): iş parçacığının çalışma önceliğini değiştirir.

```
package com.comu.ce;

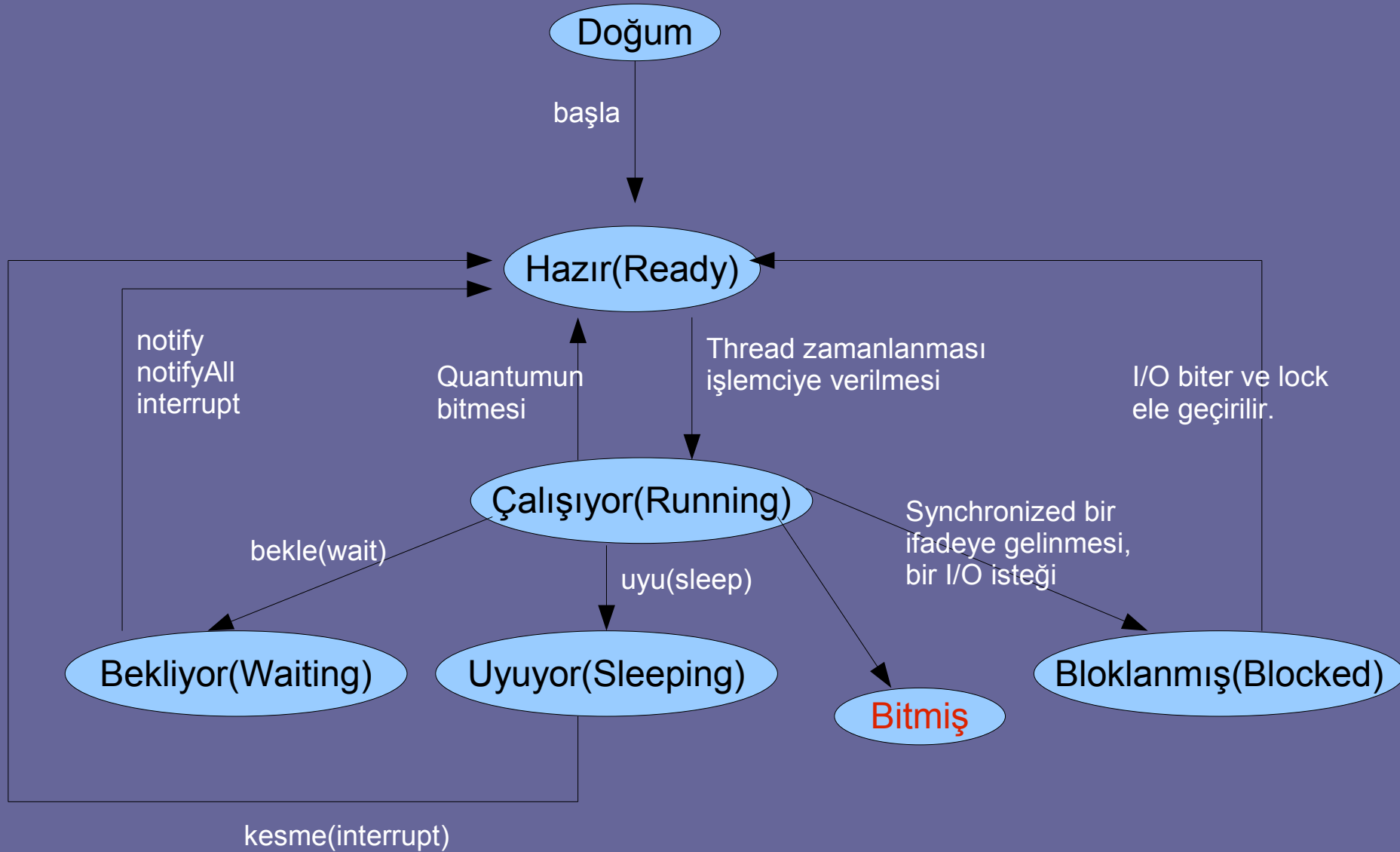
public class ThreadOncelik {
    public static void main(String[] args) {
        SayiYazici isparcacigi1 = new SayiYazici();
        SayiYazici isparcacigi2 = new SayiYazici();
        SayiYazici isparcacigi3 = new SayiYazici();

        isparcacigi1.setName("İs-1");
        isparcacigi2.setName("İs-2");
        isparcacigi3.setName("İs-3");

        isparcacigi1.setPriority(Thread.MIN_PRIORITY);
        isparcacigi2.setPriority(Thread.NORM_PRIORITY);
        isparcacigi3.setPriority(Thread.MAX_PRIORITY);

        isparcacigi1.start();
        isparcacigi2.start();
        isparcacigi3.start();
    }
}
```

İş Parçacığının durumları



join() metodu

join() metodu çalıştırıldığı iş parçacığının başka bir thread'in çalışmasını bitmesini beklemesini sağlar.

```
public final void join()  
public final void join(long milisaniye)  
public final void join(long milisaniye,int nanosaniye)
```

```
Thread t = new Thread();  
Thread t2 = new Thread();
```

t2 içerisinde t.join() : yazdığımızda t iş parçacığının çalışması bitene kadar t2 threadinin beklemesini sağlar.

join() metodu

```
Thread t = new Thread();  
Thread t2 = new Thread();
```

t.join(1000) :t iş parçacığının görevini tamamlaması için 1 saniye t2 bekler.

join() metodu

```
JoinOrnek.java
class A extends Thread{
    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(this.getName()+":"+i);
        }
    }
}
class B extends Thread{
    private Thread bekle;
    public void setBekle(Thread bekle){ this.bekle=bekle;}
    @Override
    public void run() {
        if (bekle!=null)
            try {
                bekle.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        for (int i = 0; i < 10; i++) {
            System.out.println(this.getName()+":"+i);
        }
    }
}
public class JoinOrnek {
    public static void main(String[] args) {
        A a = new A();
        B b = new B();
        a.setName("A");
        b.setName("B");
        b.setBekle(a);
        b.start();
        a.start();
    }
}
```

Çeşitli metotlar

public final boolean isAlive() : iş parçacığı start() ile başlatılmış ve çalışması bitmemiş ise true döndürür.

public static Thread currentThread() : o anki aktif iş parçacığını geri döndürür.

static int activeCount(): uygulamadaki iş parçacığı sayısını geri döndürür.

Senkronizasyon(Synchronized)

İş parçacıkları her zaman ayrı veriler üzerinde işlem yapmazlar bazen ortak veriler üzerinde işlem yapabilirler.

Bu tip durumlarda bir iş parçacığı bir veri üzerinde çalışırken başka bir iş parçacığının bu veriyi değiştirmemesi son derece önemlidir.

Örneğin; bir iş parçası bir dosyadan okurken, başka bir iş parçasıda aynı dosya ya yazıyorsa hatalar olabilir.

Senkronizasyon(Synchronized)

Bu sorunu çözmek için *synchronized* kelimesi kullanılmaktadır.

Java da her nesneye ait *Monitor* denilen bir anahtar bulunmaktadır. Synchronized kelimesini kullanarak bu nesnenin anahtarını elde ederiz. Bu anahtar kimde olursa o nesne üzerinde değiştirme hakkı o iş parçacığına aittir. Bu nesneyi kullanmayı bekleyen diğer iş parçacıkları, biz anahtarı serbest bırakıncaya kadar beklerler.

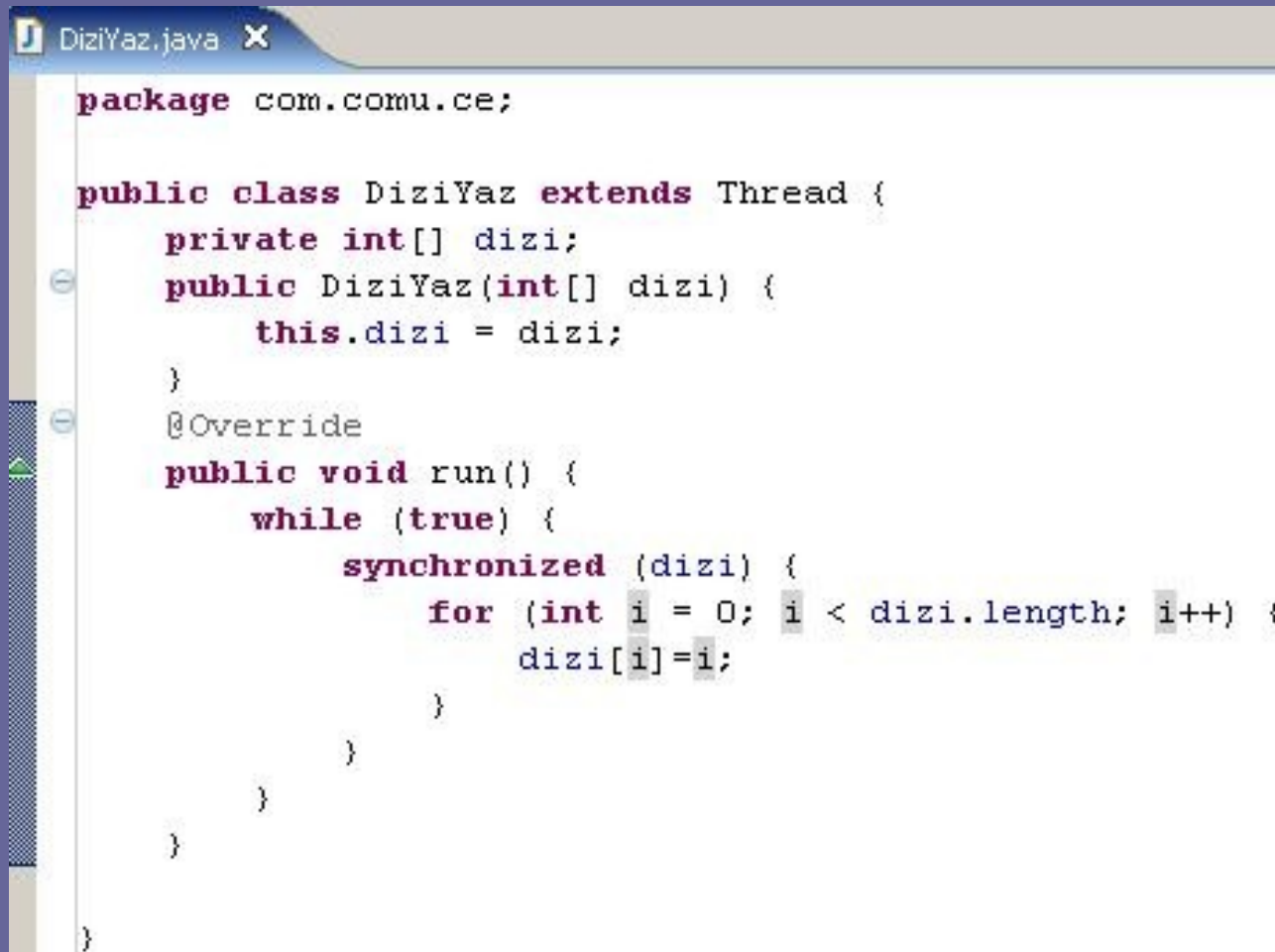
Senkronizasyon(Synchronized)

Kullanım şekli :

```
1) synchronized metot_adi(){  
  
}
```

```
2) synchronized(nesne_adi){  
  
}
```

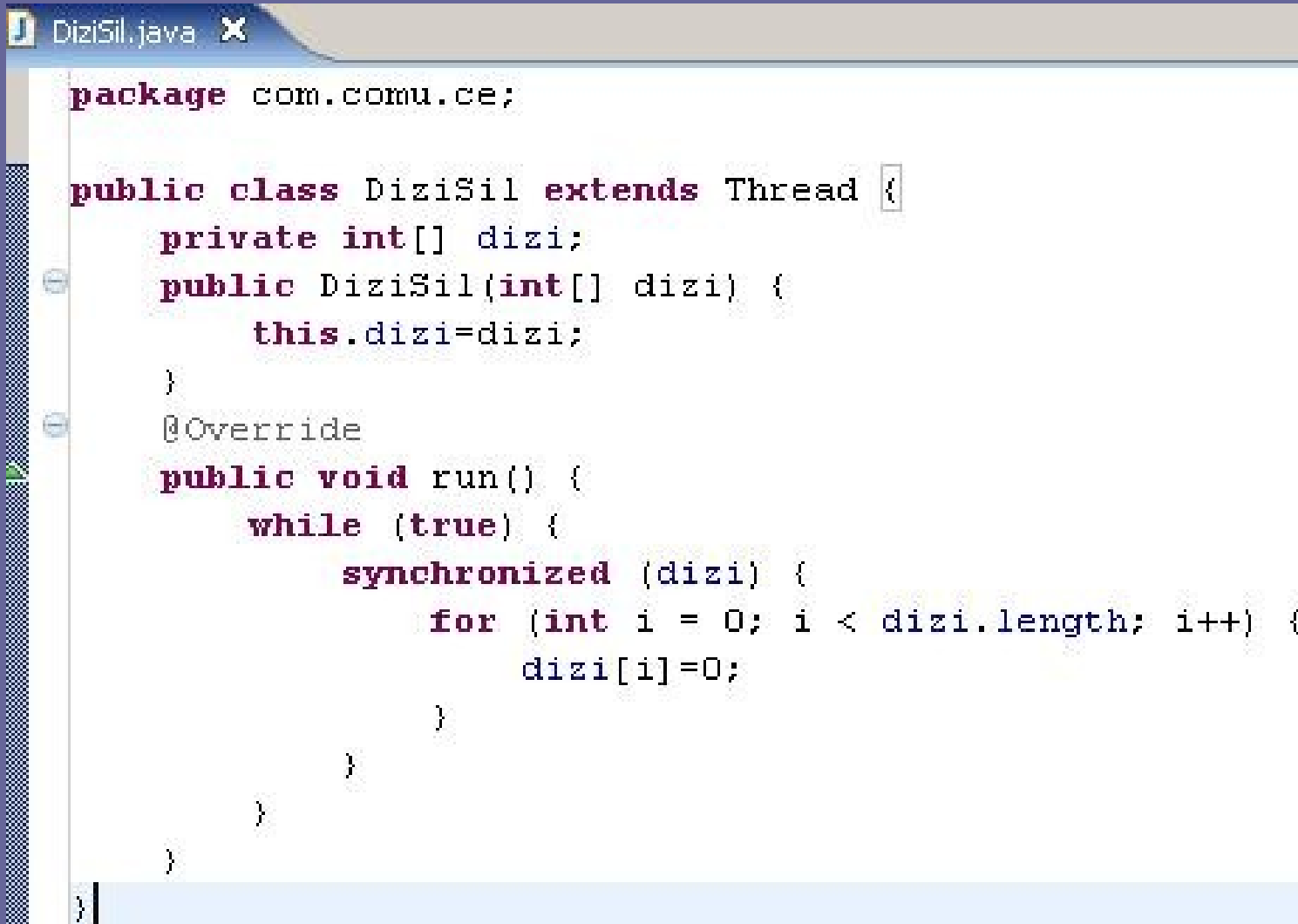
Senkronizasyon(Synchronized)



```
package com.comu.ce;

public class DiziYaz extends Thread {
    private int[] dizi;
    public DiziYaz(int[] dizi) {
        this.dizi = dizi;
    }
    @Override
    public void run() {
        while (true) {
            synchronized (dizi) {
                for (int i = 0; i < dizi.length; i++) {
                    dizi[i] = i;
                }
            }
        }
    }
}
```

Senkronizasyon(Synchronized)



```
package com.comu.ce;

public class DiziSil extends Thread {
    private int[] dizi;
    public DiziSil(int[] dizi) {
        this.dizi=dizi;
    }
    @Override
    public void run() {
        while (true) {
            synchronized (dizi) {
                for (int i = 0; i < dizi.length; i++) {
                    dizi[i]=0;
                }
            }
        }
    }
}
```

```
package com.comu.ce;
import java.awt.Graphics;
public class SenkronizeOrnek extends JApplet implements Runnable{
    Thread t;
    int i=0;
    int[] dizi;
    DiziYaz diziyaz;
    DiziSil dizisil;
    public void init() {
        dizi = new int[10];
        diziyaz = new DiziYaz(dizi);
        dizisil= new DiziSil(dizi);
        diziyaz.start();
        dizisil.start();
        t= new Thread(this);
        t.start();
    }
    public void paint(Graphics g) {
        super.paint(g);
        synchronized (dizi) {
            for (int i = 0; i < dizi.length; i++) {
                g.drawString(String.valueOf(dizi[i])+" ", 15+i*15, 15);
            }
        }
    }
    public void run() {
        while(i<=10){
            repaint();
            try {
                Thread.currentThread().sleep(2000);
                i++;
            } catch (InterruptedException e) {
            }
        }
    }
}
```

Üretici(Producer)/Tüketici(Consumer)

Üretici bir iş parçacıdır ve bir veri üreterek onu paylaşılan ortak bir veri alanına koyar.

Tüketici de bir iş parçasıdır ve üretici tarafından üretilen bir değer ortak alanda var ise bu değeri alır ve kullanır.

Tüketici, üretici değer üretip ortak alana koymadan Bir şey okumamalıdır; üretici de ürettiği ürün kullanılmadan yenisini üretip koymamalıdır.

Üretici(Producer)/Tüketici(Consumer)

Üretici yeni bir değer üretir ve *wait()* beklemeye geçer. Tüketici değeri okur ve *notify()* ile üreticiyi bilgilendirir ve *wait()* beklemeye geçer. Üretici de yeni değeri ürettikten sonra *notify()* ile tüketiciyi bilgilendirir.

Üretici(Producer)/Tüketici(Consumer)

```
Buffer.java X
package com.comu.ce;

public interface Buffer {
    public void ata(int deger);
    public int al();
}
```

```
Uretici.java X
package com.comu.ce;

public class Uretici extends Thread {
    private Buffer paylasilanAlan;
    public Uretici(Buffer paylasim) {
        super("Uretici");
        this.paylasilanAlan=paylasim;
    }
    @Override
    public void run() {
        for (int i = 1; i < 4; i++) {
            try {
                Thread.sleep((int) (Math.random() * 3001));
                paylasilanAlan.ata(i);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println(getName()+" Üretim yaptı.Sonlanıyor.");
    }
}
```

Üretici(Producer)/Tüketici(Consumer)

```
*Tuketici.java x
package com.comu.ce;

public class Tuketici extends Thread {
    private Buffer paylasilanAlan;
    public Tuketici(Buffer paylasim) {
        super("Tüketici");
        this.paylasilanAlan=paylasim;
    }
    @Override
    public void run() {
        int toplam=0;
        for (int i = 1; i <=4; i++) {
            try {
                Thread.sleep((int) Math.random() *3001);
                toplam+=paylasilanAlan.al();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println(getName()+
            " deger okuyor ve toplaniyor:"+toplam);
    }
}
```

Üretici(Producer)/Tüketici(Consumer)

```
SenkronizeOlmayanBuffer.java X

package com.comu.ce;

public class SenkronizeOlmayanBuffer implements Buffer {
    private int deger=-1;
    public int al() {
        System.out.println(Thread.currentThread().getName()+
            "deger="+deger+" i okuyor");
        return deger;
    }

    public void ata(int deger) {
        System.out.println(Thread.currentThread().getName()+
            "deger="+deger+" i yaziyor");
        this.deger=deger;
    }
}
```

Üretici(Producer)/Tüketici(Consumer)

```
PaylasilanBufferTest.java X
package com.comu.ce;

public class PaylasilanBufferTest {
    public static void main(String[] args) {
        Buffer paylasilanYer = new SenkronizeOlmayanBuffer();
        Uretici uretici = new Uretici(paylasilanYer);
        Tuketici tuketici = new Tuketici(paylasilanYer);
        uretici.start();
        tuketici.start();
    }
}
```

Üretici(Producer)/Tüketici(Consumer) Senkronize

```
Tuketici.java  Üretici.java  SenkronizeBuffer.java x
package com.comu.ce;

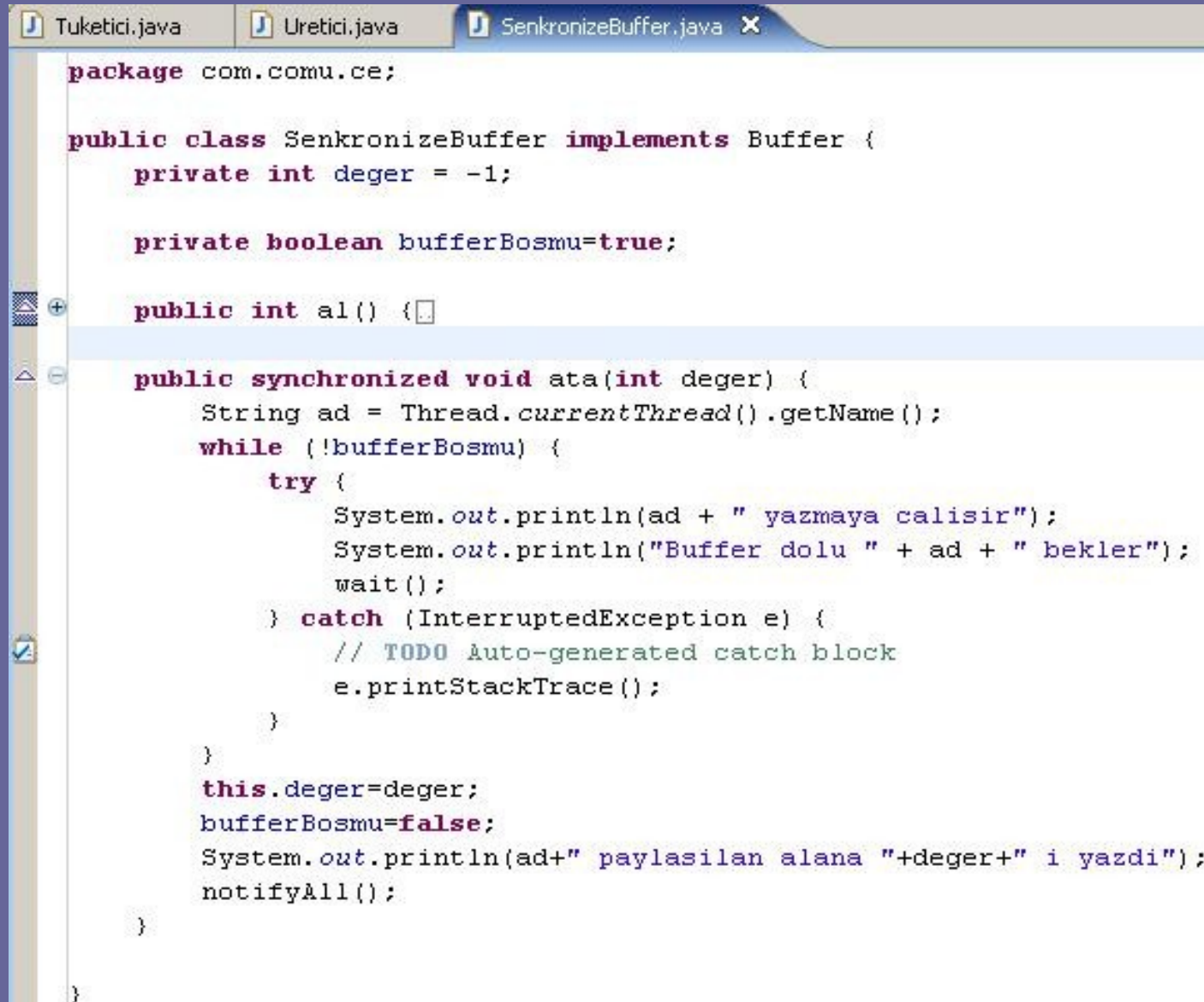
public class SenkronizeBuffer implements Buffer {
    private int deger = -1;

    private boolean bufferBosmu=true;

    public int al() {
        String ad = Thread.currentThread().getName();
        while (bufferBosmu) {
            //buffer da eleman yok bekler
            try {
                System.out.println(ad + " okumaya calisir");
                System.out.println("Buffer bos " + ad + " bekler");
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        bufferBosmu=true;
        notifyAll();
        System.out.println(ad+" paylasilan alandan "+deger+" ini okudu");
        return this.deger;
    }

    public synchronized void ata(int deger) {}
}
```

Üretici(Producer)/Tüketici(Consumer) Senkronize



```
package com.comu.ce;

public class SenkronizeBuffer implements Buffer {
    private int deger = -1;

    private boolean bufferBosmu=true;

    public int al() {

    }

    public synchronized void ata(int deger) {
        String ad = Thread.currentThread().getName();
        while (!bufferBosmu) {
            try {
                System.out.println(ad + " yazmaya calisir");
                System.out.println("Buffer dolu " + ad + " bekler");
                wait();
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        this.deger=deger;
        bufferBosmu=false;
        System.out.println(ad+" paylasilan alana "+deger+" i yazdi");
        notifyAll();
    }
}
```

Üretici(Producer)/Tüketici(Consumer) Senkronize

```
Tuketici.java  Üretici.java x
package com.comu.ce;

public class Üretici extends Thread {
    private Buffer paylasilanAlan;

    public Üretici(Buffer paylasim) {
        super("Üretici");
        this.paylasilanAlan = paylasim;
    }

    @Override
    public void run() {
        for (int i = 1; i <= 4; i++) {
            synchronized (paylasilanAlan) {

                try {
                    paylasilanAlan.ata(i);
                    Thread.sleep((int) (Math.random() * 3001));
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
        System.out.println(getName() + " Üretim yaptı.Sonlanıyor...");
    }
}
```


Üretici(Producer)/Tüketici(Consumer) Senkronize

```
Tuketici.java x
package com.comu.ce;

public class Tuketici extends Thread {
    private Buffer paylasilanAlan;

    public Tuketici(Buffer paylasim) {
        super("Tüketici");
        this.paylasilanAlan = paylasim;
    }

    @Override
    public void run() {
        int toplam = 0;
        for (int i = 1; i <= 4; i++) {
            synchronized (paylasilanAlan) {
                try {
                    Thread.sleep((int) (Math.random() * 3001));
                    toplam += paylasilanAlan.al();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
        System.out.println(getName() + " deger okuyor ve toplaniliyor:" + toplam);
    }
}
```


Üretici(Producer)/Tüketici(Consumer) Senkronize

```
PaylasilanBufferTest.java X
package com.comu.ce;

public class PaylasilanBufferTest {

    public static void main(String[] args) {
        Buffer paylasilanYer=new SenkronizeBuffer();
        Uretici uretici = new Uretici(paylasilanYer);
        Tuketici tuketici = new Tuketici(paylasilanYer);
        uretici.start();
        tuketici.start();
    }
}
```

Kilitlenme (Deadlock)

Bazen iki sınıf anı anda birbirlerinin mevcut anda kullandıkları kaynakları beklerler. Bu bekleme nedeniyle iki iş parçacığı da iş yapamazlar ve kilitlenirler. Bu olaya *kilitlenme(dead lock)* denilir. Örneğin;

```
package com.comu.ce;
class Nesne{
    private Object nesne1 = new Object();
    private Object nesne2 = new Object();

    public void birinciMethot(){
        synchronized (nesne1) {
            try {
                Thread.sleep(1);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            synchronized (nesne2) {
                System.out.println("Nesnenin birinci metodu"+
                    Thread.currentThread().getName() +
                    " tarafından çağrıldı");
            }
        }
    }

    public void ikinciMethot(){
        synchronized (nesne2) {
            try {
                Thread.sleep(1);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            synchronized (nesne1) {
                System.out.println("Nesnenin birinci metodu"+
                    Thread.currentThread().getName() +
                    " tarafından çağrıldı");
            }
        }
    }
}
```

Kilitlenme (Deadlock)

```
class IsParcacigi1 extends Thread{
    Nesne nesne;
    public IsParcacigi1(Nesne nesne) {
        this.nesne=nesne;
    }
    @Override
    public void run() {
        nesne.birinciMethot();
    }
}

class IsParcacigi2 extends Thread{
    Nesne nesne;
    public IsParcacigi2(Nesne nesne) {
        this.nesne=nesne;
    }
    @Override
    public void run() {
        nesne.ikinciMethot();
    }
}
```

Kilitlelenme (Deadlock)

```
public class DeadLockOrnek2 {  
    public static void main(String[] args) {  
        Nesne n1 = new Nesne();  
        IsParcacigi1 isparcacigi1 = new IsParcacigi1(n1);  
        IsParcacigi2 isparcacigi2 = new IsParcacigi2(n1);  
  
        isparcacigi1.start();  
        isparcacigi2.start();  
    }  
}
```

Görsel Programlama

DERS 11