Thread:

Javada herhangi bir programı çalıştırdığımızda Main thread ayağı kaldırılır, buna ek olarak yazılımcılar da alt iş parçacıkları çalıştırabilirler.

Threadler thread havuzundan ihtiyaç duyulduğunda alınır ve işlem tamamlandıktan sornra tekrar thread havuzuna bırakılır. Eğer bir öncelik belirtilmezse java kendisi senkron bir şekilde threadleri çalıştırır.

Runnable:

Runable interface’I Thread oluşturmanın bir başka yoludur, javada yalnızca tek bir classdan extends edebildiğimiz için Runnable ile thread oluşturmak daha kullanışlı olabilir.

Thread durdurma ve bekletme:

Threadleri durdurmak içim sleep metodu kullanabiliriz, mili saniye cinsinden verdiğimiz süre boyunca thread’I durdurur sonra çalışmaya devam eder.

Critical Sections ve Race Condition:

Threadlerde kritik bölümler ve veriye erişim yarışması diye türkçeye çevrilen bu tanımlar, birden fazla threadle çalıştığımız zaman bu threadler çıktı alabilmek için birbirleri ile yarışırlar buna da race condition denir, ancak biri işlem yaparken diğerlerinin onu beklemesine Critical section denir.

Threadler çalışırken birbirleriyle bir yarış halinde çalışırlar, bu da çakışma sorunları yaratabilir, örneğin bir websitemizde ürün sattığımızı düşünelim, elimizde X ürününden sadece 1 adet var, threadlerdeki bu çakışma sorunundan birden fazla müşteriye o 1 adet olan üründen satma durumu ortaya çıkabilir bunun önüne geçmek için senkron şekilde çalışmalarını sağlamamız gerekir.

Threadlerin çakışma yaratacak şekilde çalışmas

OrderMatic orderMatic=new OrderMatic();  
List<Thread> islemler=new ArrayList<>();  
for (int i=0;i<100;i++){  
 Thread t=new Thread(orderMatic);  
 t.start();  
}  
for (Thread t: islemler){  
 t.join();  
}

Senkton şekilde çalışmalarını sağlamak için kullanımı

synchronized (LOCK){  
 this.setOrnderNo(this.getOrnderNo()+1);  
 System.*out*.println("Thread: "+Thread.*currentThread*().getName()+" OrderNo: "+this.getOrnderNo());  
}

Kod bu şekilde yazıldığında Bir thread ilgili kod blokunu çalıştırdığında o kısmı kitler ve başka bir thread o kodlara erişemez, işlemi tamamladıktan sonra serbest bırakır ve başka bir thread ilgili kod blokunu kullanabilir. Bu kullanımda synchorized içerisine yerleştirdiğimiz nesne Object türünde boş bir nesnedir.

Synchorized metod:

public synchronized void increaseOrder(){  
 this.setOrnderNo(this.getOrnderNo()+1);  
 System.*out*.println("Thread: "+Thread.*currentThread*().getName()+" OrderNo: "+this.getOrnderNo());  
}

Bu kullanımda ise ilgili metodun senkton şekilde çalışmasını sağlarız.

Volatile:

Bilgisayarda çalışan programlar tüm bilgileri ram üzeinden alırlar ve ram ile işlemci arasındaki trafiği hızlandırmak için cacheler kullanılır, ancak bazen veri değeri güncellenmesine ragmen cachede eski değeri kalabilir, bu sorunun önüne geçmek için volatile yapısı kullanılır. Volatile tanımlanmış değişkenin değeri için Threadler her zaman işlemciden güncel değeri alırlar. Ancak her değişken volatile tanımlanırsa CPU yükü artacağından performans düşer, bu nedenle sadece ihtiyaç duyulan değişkenler için kullanılır.

ÖR:

private volatile int orderNo;

ThreadPooling:

Thread oluşturmak maliyetli bir iştir, bu nedenle Threadpooling kullanarak ihtiyacımız olduğunda havuzdan Thread alıp işimiz bitince havuza geri bırakıp tekrar kullanılabilir hale getiriyoruz.

OrderMatic orderMatic=new OrderMatic();  
ExecutorService pool=Executors.*newFixedThreadPool*(10);  
  
for (int i=0;i<100;i++){  
 pool.execute(orderMatic);  
}