КӨП АГЫМДУУ ПРОГРАММАЛОО.

Көп агымдуулук, борбордук процессорду максималдуу колдонуучу эффективдүү программаларды жазууга мүмкүнчүлүк түзөт. Анткени мындай программаларды түзүү менен борбордук процессордун күтүү убактысын минимумга жеткирүүгө болот. Мисалы, колдонуучунун маалыматтарды терип киргизүүсү, компьютердин иштешинен жай, файлдык системага берилиштердин жазылуусу, борбордук процессордун итешинен жай ж.б.у.с. Көбүнчө учурда борбордук процессор күтүү абалында болору бизге белгилүү, ошол себептен көп агымдуу программалоо ушул убакыттарды үнөмдүү пайдаланууга өбөлгө түзөт.

Java нын көп агымдуу системасы Thread классы жана Runnable интерфейси аркылуу ишке ашат жана Runnable интерфейсине байланышкан Thread ыкмаларында ж-а классында түзүлгөн.

Thread агымды коргоого алгандыгы (инкапсуляция) үчүн, биз түздөн-түз агымдын өзү менен иш алып бара албайбыз. Анын объектиси (экземпляры) аркылуу иштейбиз.

Жаңы агым түзүү үчүн Thread классын кеңейтүүбүз жана Runnable интерфейсин мурастообуз керек.

Агымды башкаруу үчүн Thread классы бир нече ыкмаларды колдонот:

GetName() - агымдын атын алуу

GetPriority () - агымдын приоритетин алуу

IsAlive () - агым аткарылып жатабы аныктоо үчүн колдонулат

Join () - агымдын бүтүшүн күтүү

Run () - агымдын кирүү чекитин көрсөтөт

Sleep () - бир нече убакытка агымды токтотот

Start () - run() ыкмасынын жардамы менен агымды иштетет

Пограмма иштегенде автоматтык түрдө башкы агым иштейт, аны башкаруу үчүн шилтемени currentThread() ыкмасы менен алабыз. Мисалы, төмөнкү коддо биринчи башкы агымды алып, андан соң анын атын өзгөртүп жатабыз. Чыгарып салуу механизми try дын ичинде цикл бешке чейин

айланып, ар бир итерацияда агым бир секундага sleep() ыкмасы менен кармалууда.

```
class Agym{
public static void main(String args[]){
Thread t= Thread.currentThread();
System.out.println("Учурда иштеп жаткан агым"+t);
// агымдын атын өзгөртүү
t.setName("My Thread");
System.out.println("аты өзгөргөн агым"+t);
try{
for(int n=5; n>0; n--){
System.out.println(n);
Thread.sleep(1000);
}
}catch(InterruptedException e){
System.out.println("Башкы агым аяктады");
}
}
}
```

Жогоруда биз эки агымды (башкы жана башкы агымдын алдына түзүлгөн) гана колдондук. Эгер кааласак башкы агымдын алдына бир нече агым түзө алабыз. Мисалы, төмөнкү коддо үч агым түзүү көрсөтүлгөн. Иштөө тартиби жогорку коддун иштөө тартибине окшош. Бирок, бул жерде эки класс түзүлүп, эки башка файлга сакталган. Ошондой эле, isAlive() ыкмасы менен агымдардын азыркы учурда иштеп жатабы текшерилди. Бул ыкма true же false ыкмаларын кайтарат. Класстарды компилеп, программанын кирүү чекити жайгашкан классты чакырсак, анда агымдардын түзүлүшүн жана isAlive() ыкмасынын иштешин ачык көрө алабыз.

```
class NewThread implements Runnable{
  String name;
```

```
   Thread t;
   NewThread(String threadname){
     name = threadname;
     t = new Thread(this, name);
     System.out.println("Jany agym: " + t);
     t.start();
}
 public void run(){
     try{
     for(int i = 5; i > 0; i --){
     System.out.println(name + ": " + i);
     Thread.sleep(1000);
    }
  }catch (InterruptedException e){
   System.out.println(name + "toktotuldu.");
  }
   System.out.println(name + "ayaktady. ");
  }
 }
class Demojoin {
public static void main(String args[])  {
NewThread ob1 = new NewThread("Birinchi");
NewThread ob2 = new NewThread("Ekinchi");
NewThread ob3 = new NewThread("Uchunchu");
 System.out.println("Birinchi agym ishtep jatat: " +  ob1.t.isAlive());
 System.out.println("Einchi agym ishtep jatat: " +  ob2.t.isAlive());
 System.out.println("Uchunchi agym ishtep jatat: " +  ob3.t.isAlive());
try {
    System.out.println("Agymdyn ayaktashyn kutuu.");
    ob1.t.join();
```

```
    ob2.t.join();
    ob3.t.join();
} catch (InterruptedException e) {
  System.out.println ("Bashky agymdyn ayaktoosu");
}
 System.out.println("Birinchi agym ishtep jatat: " +  ob1.t.isAlive());
 System.out.println("Einchi agym ishtep jatat: " +  ob2.t.isAlive());
 System.out.println("Uchunchi agym ishtep jatat: " +  ob3.t.isAlive());
}
}
```

Агымдардын приоритеттери (артыкчылыктары)

Java ар бир агымга приоритет дайындайт, ал ар бир агымдын салыштырмалуу иштөө тартибин аныктайт. Агымдын артыкчылыгы деп бул бир агымдын экинчисине салыштырмалуу артыкчылыгын аныктоочу бүтүн сандарды түшүнсөк болот. Жогорку артыкчылыктуу агым төмөнкү артыкчылыктуу агымдан тез иштебейт, ал жөн гана бир агымдан экинчи агымга өтүүдө колдонулат. Бул контекстти алмаштыруу деп аталат. Контекстти алмаштыруунун эрежелери төмөнкүдөй

-Агым өз ыктыяры менен киргизүү-чыгаруу каражаттарынан берилиштерди күткөндө бөгөттөлөт.

-Агым иштеп жаткандыгына карабастан артыкчылыктуу агымдын таасиринде убактылуу токтотулат.

Агымдардын пландагычы, агымдардын приоритеттерине карата алардын ишке ашуусуна же ашпоосуна уруксаат берип турары бизге белгилүү. Приоритети жогору агымдар, приоритети төмөн агымдарга караганда борбордук процессорду колдонуу убактысы көбүрөөк болот. Төмөнкү мисалда, башкы агымдын алдына эки агым түзүлүп алардын биринин нормалдуу приоритети (Thread.NORM_PRIORITY) эки даражага жогорулатылып, ал эми экинчисинин приоритети эки даражага төмөндөтүлөт. Ар бир агым циклда бир нече итерация жүргүзөт. Эки агымга он секунд

иштөөгө мүмкүнчүлүк берилет. Андан соң, башкы агым аякталып, агымдар канча итерация кылууга жетишкен саны экранга чыгат.

```
class clicker implements Runnable {
  int click = 0;
 Thread t;
  private volatile boolean running = true;

  public clicker(int p){
  t = new Thread(this);
  t.setPriority(p);
 }

   public void run(){
    while (running){
     click ++;
   }
}
   public void stop(){
    running = false;
}
   public void start() {
   t.start();
 }
}
class Pri{
  public static void main(String args[]) {
    Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
    clicker hi = new clicker(Thread.NORM_PRIORITY + 1);
    clicker lo = new clicker(Thread.NORM_PRIORITY + 5);
    lo.start();
```

```
    hi.start();
    try {
      Thread.sleep(10000);
    }  catch (InterruptedException e) {
        System.out.println("Bashky agymdy toktotuu. ");
    }
    lo.stop();
    hi.stop();
    try {
      hi.t.join();
      lo.t.join();
    }  catch (InterruptedException e) {
      System.out.println("InterruptedException caught");
    }
      System.out.println("Приоритети төмөн агым: " + lo.click);
      System.out.println("Приоритети жогору агым: " + hi.click);
    }
}
```

Бул программанын жыйынтыгы, компьютердин иштетүү системасынын түзүлүшүнө жараша ар кандай чыгат. Кодду компилеп, иштетсек, приоритети жогору агым көп жолу итерация жасоого жетишкендигин көрө алабыз.

```
class NewThread implements Runnable{
  Thread t;
  NewThread(){
    t = new Thread(this, "Demo thread");
    System.out.println("Jany agym: " + t);
    t.start();
  }
 public void run(){
    try{
```

```
    for(int i = 5; i > 0; i --){
    System.out.println("Jany agym:" + i);
    Thread.sleep(1000);
    }
  }catch (InterruptedException e){
  System.out.println("Jany agym toktotuldu.");
  }
  System.out.println("Jany agym ayaktady. ");
 }
}

class Demojoin {
public static void main(String args[])  {
new NewThread();
    try{
    for(int i = 5; i > 0; i --){
    System.out.println("bashky agym " + i);
    Thread.sleep(1000);
    }
    }catch (InterruptedException e){
  System.out.println("bashky agymdy uzuu.");
    }
  System.out.println("bashky agym toktotuldu");
}
}
```