Многопоточность

Thread – нить (в отличие от stream – поток)

Выполнение нескольких действий одновременно (сервер обслуживает нескольких клиентов, игра обрабатывает действия пользователя и перерисовывает игровое поле)

Работа с разными устройствами компьютера (вычислительный процессор, оперативная память, клавиатура и мышь)

Основные понятия

Процесс — это совокупность кода и данных, разделяющих общее виртуальное адресное пространство. Чаще всего одна программа состоит из одного процесса.

Внутри одного процесса может быть запущено несколько потоков, отдельных единиц выполнения последовательностей действий.

Приоритет потоков определяет важность каждого потока относительно остальных.

Обслуживающие потоки называют демонами.

Класс Thread

Для создания своего потока:

Создать наследник класса Thread

Переопределить метод void run()

Создать объект класса

Вызвать метод void start()

Пример

```
public class MyThread extends Thread {
   public void run() {
      long sum=0;
      for (int i=0; i<1000; i++) {
         sum+=i;
      }
      System.out.println(sum);
   }
}

MyThread t = new MyThread();
t.start();
```

Интерфейс Runnable

Для создания своего потока:

Реализовать интерфейс Runnable (метод void run())

Создать объект класса

Создать объект класса Thread, передав в качестве аргумента конструктора созданный объект

Вызвать метод void start()

Пример

```
public class MyRunnable implements Runnable {
   public void run() {
      long sum=0;
      for (int i=0; i<1000; i++) {
         sum+=i;
      }
      System.out.println(sum);
   }
}

Runnable r = new MyRunnable();
Thread t = new Thread(r);
t.start();
```

Методы класса Thread

void sleep(long millis) throws InterruptedException

приостанавливает выполнение текущего потока на указанное количество миллисекунд, его можно вывести из этого состояния, вызвав метод interrupt() из другого потока выполнения, в результате метод sleep() прервется исключением InterruptedException

void yield()

временно приостанавливает свою работу и позволяет отработать другим потокам

Работа с приоритетами

Методы класса Thread

int getPriority()

void setPriority(int newPriority)

Константы

static int MAX_PRIORITY (10)

static int MIN_PRIORITY (1)

static int NORM_PRIORITY (5)

Потоки демоны

boolean isDaemon()

void setDaemon(boolean on)

Класс ThreadGroup

Все потоки находятся в группах, представляемых экземплярами класса ThreadGroup. Группа указывается при создании потока. Если группа не была указана, то поток помещается в ту же группу, где находится поток, породивший его.

Методы

int activeCount()

int enumerate(Thread[] list)

Хранение переменных в памяти

Для каждого потока создается его собственная рабочая память (working memory), в которую перед использованием копируются значения всех переменных (поля объектов, элементы массивов).

Поток создается с чистой рабочей памятью и должен перед использованием загрузить все необходимые переменные из основного хранилища. Потоки никогда не взаимодействуют друг с другом напрямую, только через главное хранилище.

Блокировки

В основном хранилище для каждого объекта поддерживается блокировка (lock), над которой можно произвести два действия – установить (lock) и снять (unlock). Только один поток в один момент времени может установить блокировку на некоторый объект.

В Java-программе для того, чтобы воспользоваться механизмом блокировок, существует ключевое слово synchronized. Применяется – для объявления synchronized -блока и как модификатор метода

Программист сам должен строить работу программы таким образом, чтобы неразрешимые блокировки не возникали.

Управление доступом к объектам

Каждый объект в Java имеет ассоциированный с ним монитор. Монитор представляет своего рода инструмент для управления доступа к объекту. Когда выполнение кода доходит до оператора synchronized, монитор объекта res блокируется, и на время его блокировки монопольный доступ к блоку кода имеет только один поток, который и произвел блокировку. После окончания работы блока кода, монитор объекта res освобождается и становится доступным для других потоков.

После освобождения монитора его захватывает другой поток, а все остальные потоки продолжают ожидать его освобождения.

Блокировка на уровне объекта

Это механизм синхронизации не статического метода или не статического блока кода, такой, что только один поток сможет выполнить данный блок или метод на данном экземпляре класса.

```
public class DemoClass{
    public synchronized void demoMethod(){}
}
```

Пример

```
public class DemoClass{
    public void demoMethod(){
        synchronized (this)       {
            //other thread safe code
        }
    }
}
```

Пример

```
public class DemoClass{
    private final Object lock = new Object();
    public void demoMethod(){
        synchronized (lock)       {
            //other thread safe code
        }
    }
}
```

Потоки в GUI

Существует отдельный поток, обслуживающий GUI

Все визуальные компоненты существуют в этом потоке

Выполнение параллельных задач осуществляется в отдельном потоке с помощью класса Task<T>