



Первый слайд

Алексей Лисунов

alisunov@mera.ru

GIT

<http://students.mera.ru/>

Папка нашей группы

<http://students.mera.ru/javaEE/students>

Папка с лекциями

<http://students.mera.ru/javaEE/lectures>



Вопросы

Вопросы по лекции:

1. Не понял на лекции - сразу спроси
2. Не спросил на лекции – спроси после лекции
3. Не спросил после лекции – спроси по email



Вопросы

Вопросы по заданиям

1. Не получается сделать – попробуй сделать по-другому
2. Все равно не работает – «Окей, гугл»
3. Не помог гугл – пиши email/спрашивай на лекции

В письме хочу видеть следующее

Хочу сделать *<что делаешь>* для того, чтобы *<зачем ты это делаешь>*.

Не работает *<что именно не работает, подробно>*. Пробовал:

1.....

2.....

3.....

Вот мой проект на GIT *<ссылка на проект>*



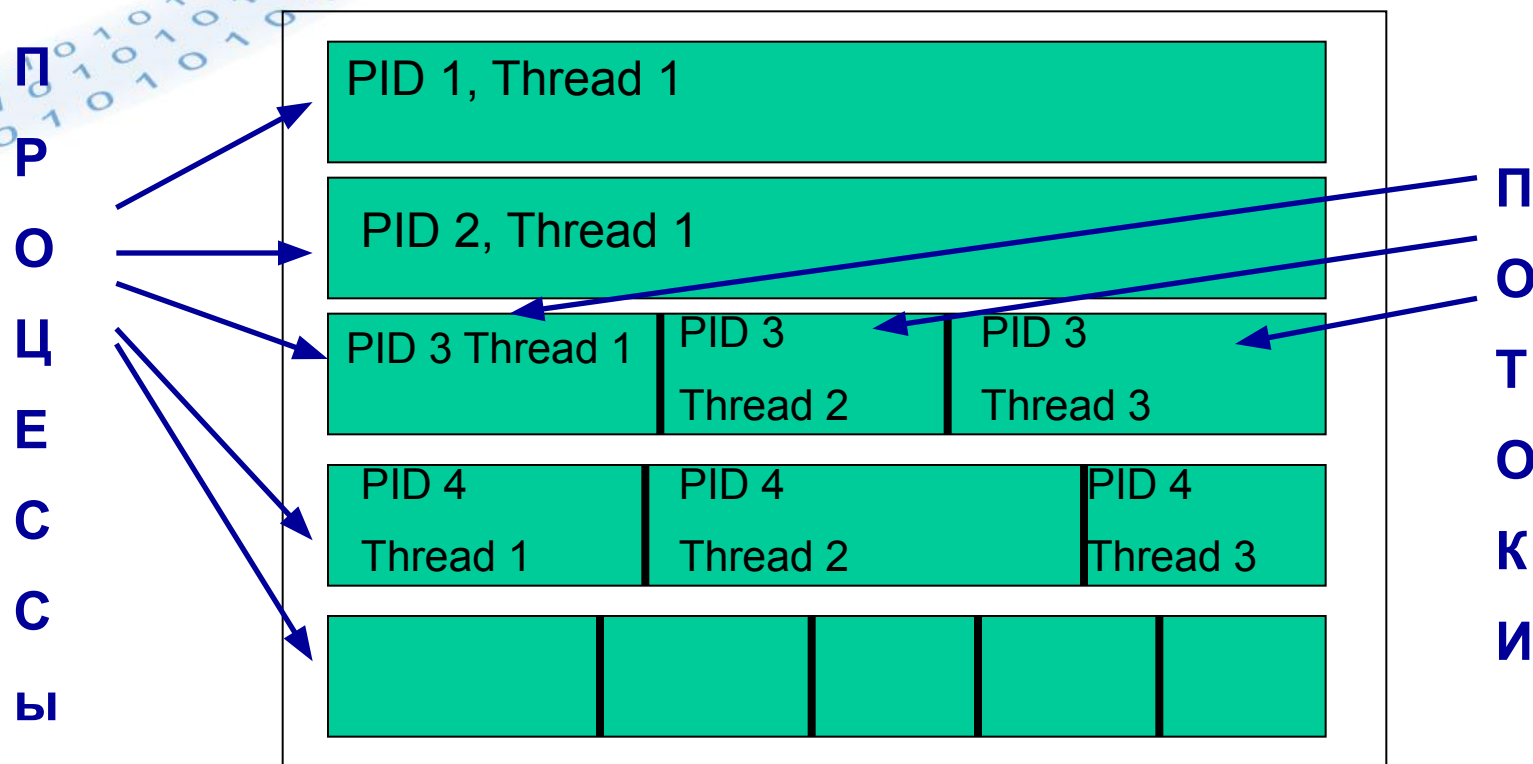
Многопоточное программирование на Java Процессы и потоки

Процесс имеет собственную среду исполнения и собственный выделенный ему набор ресурсов.

JVM выполняется как один процесс.

Потоки (threads) – «легковесные» (lightweight) процессы, они имеют собственную среду исполнения, но их создание требует меньше ресурсов. Потоки всегда существуют внутри процесса, разделяя все его ресурсы. Каждое Java-приложение состоит как минимум из одного потока (main thread) и может создавать другие потоки по мере необходимости.

Адресное пространство ОС





Многопоточное программирование на Java

Когда все потоки останавливаются
программа завершается.



Потоки-демоны

JVM завершает работу, когда завершатся все потоки не демоны.

Чтобы установить поток-демон, нужно вызвать `setDaemon(true)` из класса `Thread`



Многопоточное программирование на Java Создание потоков

Поток в Java – объект класса Thread.

Создать поток можно 2-мя способами:

- унаследовать класс от Thread
- реализовать интерфейс Runnable

Второй способ предпочтительнее, т.к. наследование от Thread запрещает наличие других суперклассов.



Многопоточное программирование на Java Создание потоков

```
public class HelloThread extends Thread
{
    public void run()
    {
        System.out.println("Hello from a thread!");
    }
    public static void main(String args[ ])
    {
        (new HelloThread()).start();
    }
}
```



Многопоточное программирование на Java Создание потоков

```
public class HelloRunnable implements Runnable
{
    public void run()
    {
        System.out.println("Hello from a thread!");
    }
    public static void main(String args[ ])
    {
        (new Thread(new HelloRunnable())).start();
    }
}
```



MERA

Пример

ThreadRun



Области памяти в java

Stack

- адреса возврата
- аргументы методов
- локальные переменные

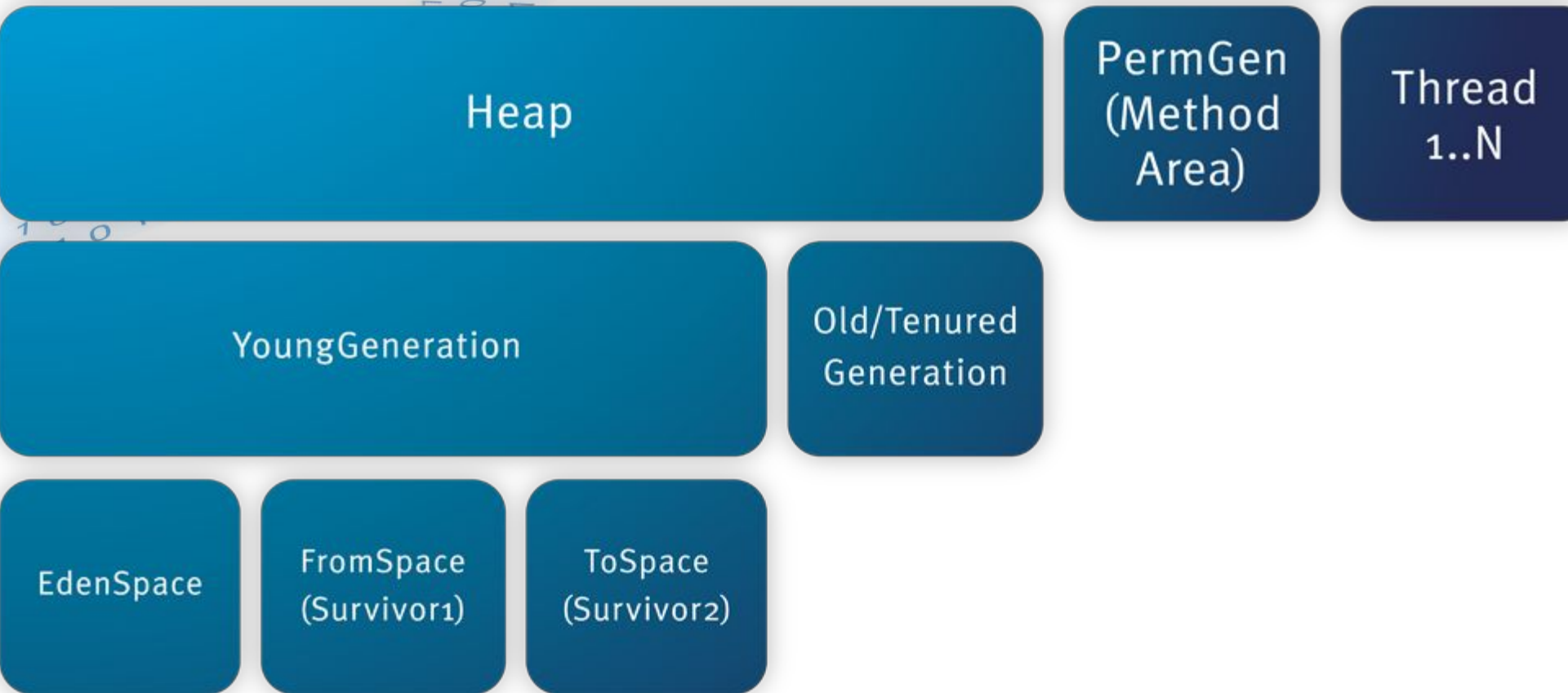
Heap

- динамически выделяемые объекты

PermGen (Metaspace – нативная память)

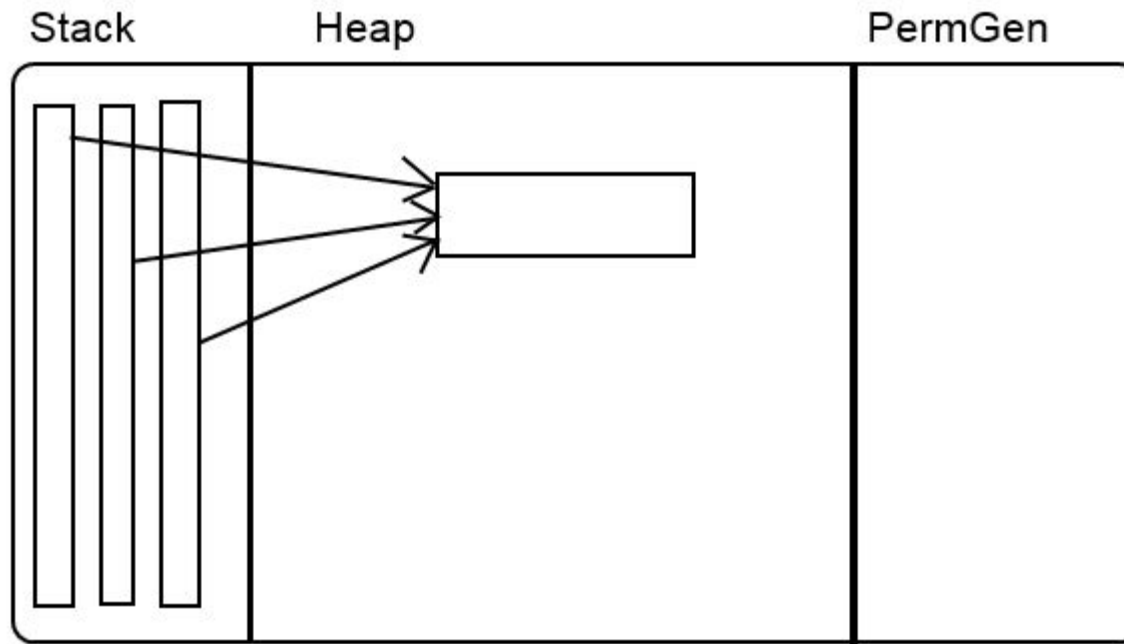
- классы
- статические поля (ссылки)

Java Memory Model





Общий доступ



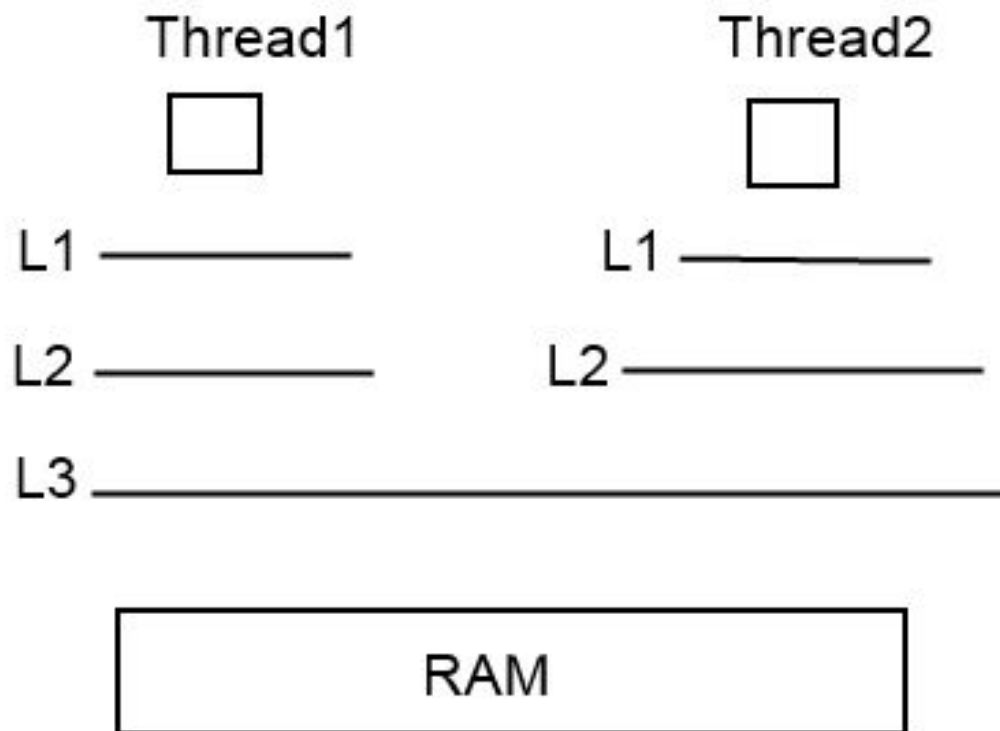


пример

JMMExample, CuncurrentCounter_1



Общий доступ





volatile

- Для нее не работают кеши
- Не может быть локальная переменная
 - Либо поле, либо статическое поле
- Данные записываются и считываются используя реальную память
- Требуется больше времени



Многопоточное программирование на Java Приостановка потоков

```
public static void main(String args[ ]) throws InterruptedException
{String Info[ ] = { "String 1", "String 2", "String 3", "String 4" };
  for (int i = 0; i < Info.length; i++)
  {   //Pause for 4 seconds
      Thread.sleep(4000);
      //Print a message
      System.out.println(Info[i]);
  }
}
```



MERA

пример

Thread_sleep



Thread

- `yield()`
- `getState()` используется только для мониторинга, не для синхронизации
- `run()` Вызов данного метода не приводит к созданию нового потока
- `isAlive()` true если вызван `start()`, и поток еще не завершен



пример

StartVsRun, ThreadRunExample



Многопоточное программирование на Java Связывание потоков

Метод: `public final void join()`

Вызывающий поток останавливается и ждет завершения потока `t`

Приоритеты потоков

В классе `Thread` существуют методы

`public final int getPriority()`

`public final void setPriority(int newPriority)`

и три константы:

`MIN_PRIORITY = 1`

`MAX_PRIORITY = 10`

`NORM_PRIORITY = 5`



пример

JoinExample, Priority

Многопоточное программирование на Java Прерывание потоков

Метод: `public void interrupt()`
Реакция на прерывание:

```
try
{ Thread.sleep(4000);
}
catch (InterruptedException e)
{ return;
}
```

```
if (Thread.interrupted())
{
    return;
}
```

Первый вариант работает если в процессе выполнения часто вызываются методы определенные как ... throws InterruptedException, в противном случае надо использовать второй метод



пример

InterruptExample

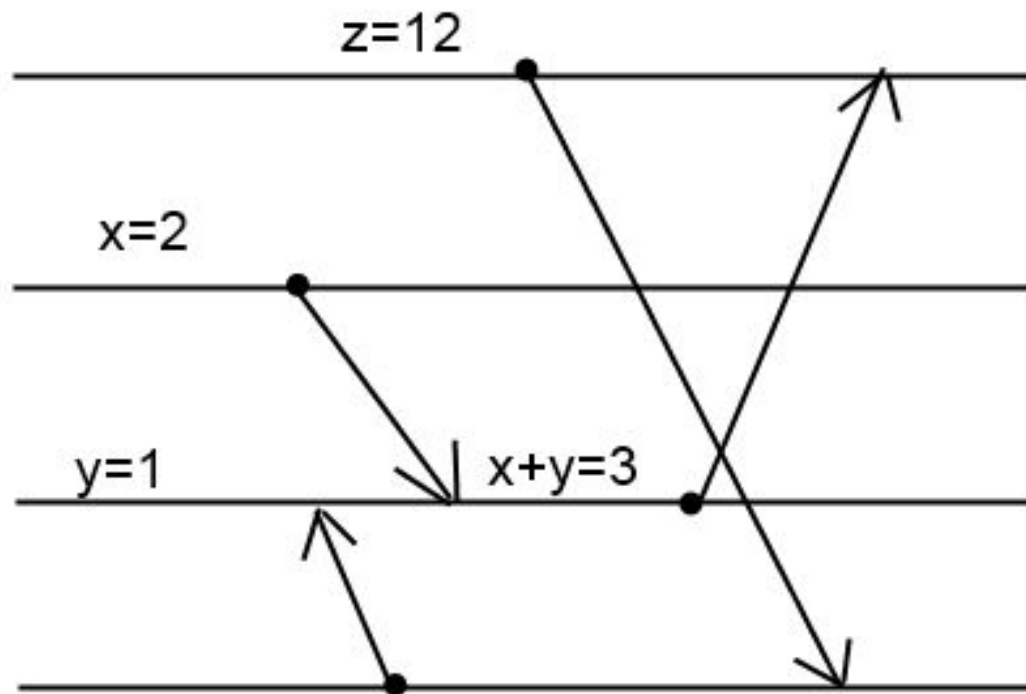


Synchronized

- Порядок инструкций сохраняется
- Happens-before
- Данные кеша сбрасываются в RAM
- Поток, зашедший в synchronized, увидит ровно то, что оставил вышедший из synchronized поток

MERA

Synchronized





Многопоточное программирование на Java Синхронизация

```
public class ThreadTest implements Runnable
{ private static ThreadTest shared = new ThreadTest();
  public void process()
  {
    for (int i=0; i<3; i++)
    {System.out.println(Thread.currentThread().getName()+i);
      Thread.yield();
    }
  }
}
```



Многопоточное программирование на Java Синхронизация

```
public void run()
{
    shared.process();
}
public static void main(String s[ ])
{
    for (int i=0; i<3; i++)
    {
        new Thread(new ThreadTest(), "Thread-"+i).start();
    }
}
}
```



Многопоточное программирование на Java Синхронизация

Пример вывода:

Thread-0 0

Thread-1 0

Thread-2 0

Thread-0 1

Thread-2 1

Thread-0 2

Thread-1 1

Thread-2 2

Thread-1 2

```

public void run()
{
    synchronized (shared)
    {
        shared.process();
    }
}
public static void main(String s[ ])
{
    for (int i=0; i<3; i++)
    {
        new Thread(new ThreadTest(), "Thread-"+i).start();
    }
}
}

```



Многопоточное программирование на Java Синхронизация

Пример вывода:

Thread-0 0

Thread-0 1

Thread-0 2

Thread-1 0

Thread-1 1

Thread-1 2

Thread-2 0

Thread-2 1

Thread-2 2



Многопоточное программирование на Java Синхронизация

Synchronized-методы работают аналогичным образом. Прежде, чем начать выполнять их, поток пытается заблокировать объект, у которого вызывается метод. После выполнения блокировка снимается.

```
public class ThreadTest implements Runnable
{ private static ThreadTest shared = new ThreadTest();
  public void synchronized process()
  {
    for (int i=0; i<3; i++)
    {System.out.println(Thread.currentThread().getName()+i);
      Thread.yield();
    }
  }
}
```



Многопоточное программирование на Java Синхронизация

Также допустимы методы `static synchronized`. При их вызове блокировка устанавливается на объект класса `Class`, отвечающего за тип, у которого вызывается этот метод.



Многопоточное программирование на Java Тупики (deadlocks)

```
public class DeadlockDemo
{public final static Object one=new Object(), two=new Object();
  public static void main(String s[ ]) {
    Thread t1 = new Thread()
    { public void run()
      {synchronized(one)
        {Thread.yield();
          synchronized (two) {System.out.println("Success!"); }
        }
      }
    }
  };
```

Многопоточное программирование на Java Тупики (deadlocks)

```
Thread t2 = new Thread()  
{  
    public void run()  
    {  
        synchronized(two)  
        {  
            Thread.yield();  
            synchronized (one) {System.out.println("Success!"); }  
        }  
    }  
};  
t1.start(); t2.start();  
}
```



пример

Deadlock



Многопоточное программирование на Java Wait-set методы

Каждый объект в Java имеет не только блокировку для `synchronized` блоков и методов, но и так называемый `wait-set`, набор потоков исполнения. Любой поток может вызвать метод `wait()` любого объекта и таким образом попасть в его `wait-set`. При этом выполнение такого потока приостанавливается до тех пор, пока другой поток не вызовет у этого же объекта метод `notifyAll()`, который пробуждает все потоки из `wait-set`. Метод `notify()` пробуждает один случайно выбранный поток из данного набора.



Многопоточное программирование на Java Wait-set методы

Однако применение этих методов связано с одним важным ограничением. Любой из них может быть вызван потоком у объекта только после установления блокировки на этот объект. То есть либо внутри `synchronized`-блока с ссылкой на этот объект в качестве аргумента, либо обращения к методам должны быть в синхронизированных методах класса самого объекта.

Есть еще `blocked-set`, содержащий объекты, попытавшиеся зайти в `synchronized` блок



Многопоточное программирование на Java Wait-set методы

```
public class WaitThread implements Runnable
{ private Object shared;
  public WaitThread(Object o)
  { shared=o;}
  public void run()
  { synchronized (shared) {
    try { shared.wait(); } catch (InterruptedException e) {}
    System.out.println("after wait");
  }
}
```




Многопоточное программирование на Java Wait-set методы

```
public static void main(String s[])
{
    Object o = new Object();
    WaitThread w = new WaitThread(o);
    new Thread(w).start();
    try { Thread.sleep(100); } catch (InterruptedException e) {}
    System.out.println("before notify");
    synchronized (o) { o.notifyAll(); }
}
}
```



Многопоточное программирование на Java Wait-set методы

Вывод программы:

before notify

after wait



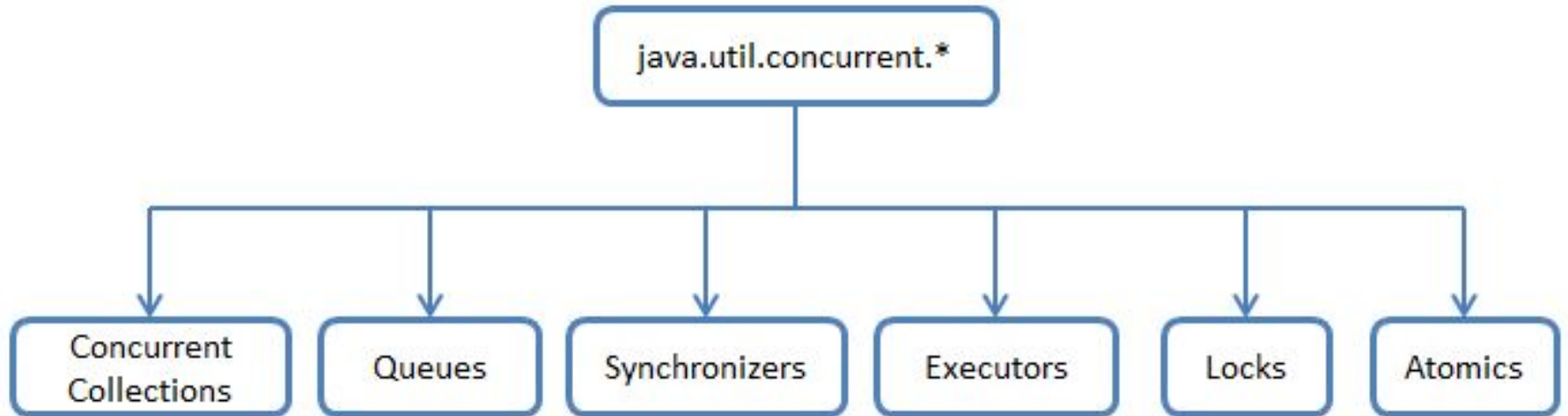
пример

WaitExample1-4



пример

BufferExampleMainProducer
BufferExampleMain
BufferExampleMainProducerBuffer





ReentrantLock

- **void lock():** ожидает, пока не будет получена блокировка
- **boolean tryLock():** пытается получить блокировку, если блокировка получена, то возвращает true. Если блокировка не получена, то возвращает false. В отличие от метода lock() не ожидает получения блокировки, если она недоступна
- **void unlock():** снимает блокировку
- **Condition newCondition():** возвращает объект Condition, который связан с текущей блокировкой



Практика 1

Есть два счета. Необходимо перевести деньги с одного счета на другой



Практика 2

Есть класс – Робот. У него две ноги. На каждую ногу создается свой поток.

Необходимо сделать так, чтобы ноги ходили поочередно:

Левая

Правая

Левая

Правая

...



Практика 3

Задание из практики 2, но у робота 4 ноги.