

Потоки выполнения



► Потоки — средство, которое помогает организовать одновременное выполнение нескольких задач, каждой в независимом потоке

- способы создания :

- на основе расширения класса Thread
- реализации интерфейса Runnable

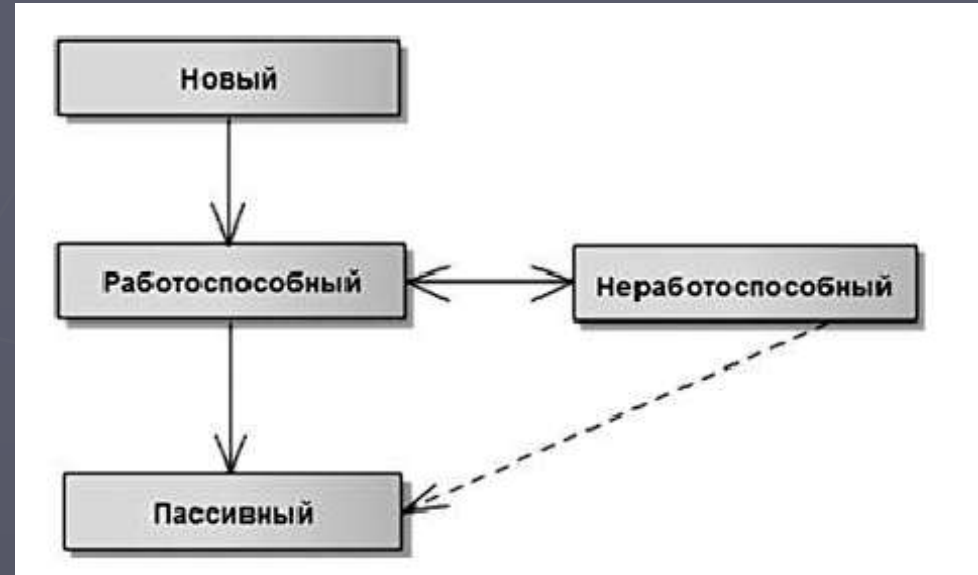
```
class ExThread extends Thread {  
    @Override  
    public void run() {  
        for (int i = 0; i < 10; i++) {  
            System.out.println("Thread");  
            try {  
                Thread.sleep(7); // остановка на 7 миллисекунд  
            } catch (InterruptedException e) {  
                System.err.print(e);  
            }  
        }  
    }  
}
```

```
}  
class ExRunnable implements Runnable {  
    @Override  
    public void run() {  
        for (int i = 0; i < 10; i++) {  
            System.out.println("Runnable");  
            try {  
                Thread.sleep(7);  
            } catch (InterruptedException e) {  
                System.err.println(e);  
            }  
        }  
    }  
}
```


Жизненный цикл потока

Thread.State:

- ▶ NEW ;
- ▶ RUNNABLE; - start
- ▶ BLOCKED;
- ▶ WAITING;
- ▶ TIMED_WAITING ;
- ▶ TERMINATED.



getState()

► WAITING

- `join()`,
- `wait()`, `<-> notify()` или `notifyAll()`
- `suspend()` (deprecated-метод) `<-> resume()`

► TIMED_WAITING

- `yield()`,
- `sleep(long millis)`,
- `join(long timeout)`
- `wait(long timeout)`

TERMINATED

- `interrupt()` `isInterrupted()`
- `stop()` (deprecated-метод) или `run()`
закончен

Управление приоритетами и группы Потоков

- ▶ 1 (константа MIN_PRIORITY)
- ▶ до 10 (MAX_PRIORITY)

```
ExThread min = new ExThread("Min");
ExThread max = new ExThread("Max");
ExThread norm = new ExThread("Norm");
    min.setPriority(Thread.MIN_PRIORITY); // 1
    max.setPriority(Thread.MAX_PRIORITY); // 10
    norm.setPriority(Thread.NORM_PRIORITY); // 5
    min.start();
    norm.start();
    max.start();
```

Min
Max
Norm
Max
Max
Max
Max
Max
Max
Max
Max
Max
Max
Norm
Max
Norm
Norm
Norm
Norm
Norm
Norm
Min
Min
Min
Min
Min
Min
Min
Min
Min
Min

► Создание групп

```
ThreadGroup tg = new ThreadGroup("Группа потоков");  
Thread t0 = new Thread(tg, "поток");
```

Поток с низким приоритетом – не измен.

Поток с высоким – понижается до
приоритета группы

Управление потоками

```
public static class DemoThreds {
    static {
        System.out.println("Start main");
    }

    public static void main(String[] args) {
        ExThread t1 = new ExThread("1");
        ExThread t2 = new ExThread("2");

        t1.start();
        t2.start();
        try {
            t1.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println(Thread.currentThread().getName());
        // имя текущего потока
    }
}
```

t1.join(20);

Start main
1
2
1
1
1
1
1
1
1
1
2
1
2
1
2
1
main
2
2
2
2
2
2
2

```

public static void main(String[] args) {
    new Thread() { // анонимный класс
        public void run() {
            System.out.println("старт 1");
            Thread.yield();
            System.out.println("завершение 1");
        }
    }.start(); // запуск потока
    new Thread() {
        public void run() {
            System.out.println("старт 2");
            System.out.println("завершение 2");
        }
    }.start();
}

```

"C:\Program ...

```

старт 2
завершение 2
старт 1
завершение 1

```

Process finished with exit code 0

"C:\Program ...

```

старт 1
завершение 1
старт 2
завершение 2

```

Потоки-демоны

- ▶ для работы в фоновом режиме
 - `setDaemon(boolean value)`

```
ExThread usual = new ExThread("1");
ExThread daemon = new ExThread("2");
daemon.setDaemon(true);
daemon.start();
usual.start();
System.out.println("отработал main");
```

```
finally {
    if (!isDaemon()) {
        System.out.println("завершение обычного потока");
    } else {
        System.out.println("завершение потока-демона");
    }
}
```

```
"C:\Program ...
1
2
2
отработал main
1
завершение обычного потока
завершение потока-демона
```

Атомарные типы и модификатор volatile

volatile - неблокирующая синхронизация

java.util.concurrent.atomic

AtomicInteger,

AtomicLong,

AtomicReference...

Пример на синхронизацию

```
import java.util.Random;
import java.util.concurrent.atomic.AtomicLong;
public class Market extends Thread {

    private AtomicLong index;

    public Market(AtomicLong index) {
        this.index = index;
    }

    public AtomicLong getIndex() {
        return index;
    }
    @Override
    public void run() {
        Random random = new Random();
        try {
            while (true) {
                index.addAndGet(random.nextInt(10));
                Thread.sleep(random.nextInt(300));
                index.addAndGet(-1 * random.nextInt(10));
                Thread.sleep(random.nextInt(300));
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```
public class Bank extends Thread {  
    private Market market;  
    private static final int PAUSE = 500;  
    public Bank(Market market) {  
        this.market = market;  
    }  
    @Override  
    public void run() {  
        try {  
            while (true) {  
                System.out.println  
                ("Current index: " + market.getIndex());  
                Thread.sleep(PAUSE);  
            }  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
}
```


Методы *synchronized*

Задача: несколько потоков должны разделять ресурс, который может быть одновременно доступен только одному потоку.

Синхронизированный метод изолирует объект, после чего он становится недоступным для других потоков.

- ▶ Изоляция снимается
 - Метод завершен
 - `wait()` из изолированного метода

Монитор

- ▶ контролирует доступ к объекту
- ▶ реализует принцип блокировки

У каждого объекта в Java имеется свой монитор (встроен в язык) - синхронизировать можно любой объект.

synchronized

Синхронизированные методы

- **Пример:** один склад, разгружаются две машины, принять можно только один товар.



```
class ResourceObj {  
    private FileWriter slkad;  
  
    public ResourceObj(String file) throws IOException {  
        // проверка наличия файла  
        slkad = new FileWriter(file, true);  
    }  
    public void dumping(String str, int i) {  
        try {  
            slkad.append(str );  
            System.out.print(str );  
            Thread.sleep((long) (Math.random() * 50));  
            slkad.append("->" + i + " товар выгружен ");  
            System.out.println("->" + i + " товар выгружен ");  
        } catch (IOException e) {  
            System.err.print("ошибка файла: " + e);  
        } catch (InterruptedException e) {  
            System.err.print("ошибка потока: " + e);  
        }  
    }  
    public void close() {  
        try {  
            slkad.close();  
        } catch (IOException e) {  
            System.err.print("ошибка закрытия файла: " + e);  
        }  
    }  
}
```

```
class SyncThread extends Thread {  
    private ResourceObj logistCenter;  
  
    public SyncThread(String name, ResourceObj rs) {  
        super(name);  
        this.logistCenter = rs;  
    }  
  
    public void run() {  
        for (int i = 1; i < 6; i++) {  
            logistCenter.dumping(getName(), i);  
            // место срабатывания синхронизации  
        }  
    }  
}
```

```

public class SynhroRun {
    public static void main(String[] args) {
        ResourceObj s = null;
        try {
            s = new ResourceObj("data\\synh.txt");
            SyncThread t1 = new SyncThread("Машина 1", s);
            SyncThread t2 = new SyncThread("Машина 2", s);
            SyncThread t3 = new SyncThread("Машина 3", s);
            t1.start();
            t2.start();
            t3.start();
            t1.join();
            t2.join();
            t3.join();
        } catch (IOException e) {
            System.err.print("ошибка файла");
        } catch (InterruptedException e) {
            System.err.print("ошибка поток");
        } finally {
            s.close();
        }
    }
}

```

"C:\Program ...

Машина 1Машина 2Машина 3->1 то
выгружен
Машина 1->1 товар выгружен
Машина 2->2 товар выгружен
Машина 2->1 товар выгружен
Машина 3->2 товар выгружен
Машина 3->2 товар выгружен
Машина 1->3 товар выгружен
Машина 2->3 товар выгружен
Машина 3->4 товар выгружен
Машина 2->3 товар выгружен
Машина 1->4 товар выгружен
Машина 3->5 товар выгружен
->4 товар выгружен
Машина 1->5 товар выгружен
->5 товар выгружен

Process finished with exit cod

1) Метод должен быть синхронизирован (synchronized в его объявлении)

2) объект блокируется, и ни один синхронизированный метод этого объекта не может быть вызван другим потоком

3) Потоки ожидают до тех пор, пока не будет разблокирован объект

```

public synchronized void dumping(String str, int i) {
    try {
        slkad.append(str );
        System.out.print(str );
        Thread.sleep((long) (Math.random() * 50));
        slkad.append("->" + i + " товар выгружен ");
        System.out.println("->" + i + " товар выгружен ");
    } catch (IOException e) {
        System.err.print("ошибка файла: ");
    } catch (InterruptedException e) {
        System.err.print("ошибка потока: ");
    }
}

```

"C:\Program ...

```

Машина 1->1 товар выгружен
Машина 2->1 товар выгружен
Машина 3->1 товар выгружен
Машина 3->2 товар выгружен
Машина 3->3 товар выгружен
Машина 3->4 товар выгружен
Машина 2->2 товар выгружен
Машина 1->2 товар выгружен
Машина 1->3 товар выгружен
Машина 1->4 товар выгружен
Машина 1->5 товар выгружен
Машина 2->3 товар выгружен
Машина 2->4 товар выгружен
Машина 2->5 товар выгружен
Машина 3->5 товар выгружен

```

Process finished with exit

Задача "Producer-Consumer"

```
class Store {  
    int counter = 0; // счетчик товаров  
    final int N = 5; // максимально допустимое число  
  
    // синхронизированный метод для производителей  
    synchronized int put() {  
        if(counter<=N) //если товаров меньше  
        {  
            counter++; // кладем товар  
            System.out.println ("на складе " + counter + " товар(ов)");  
            return 1; // в случае удачного выполнения возвращаем 1  
        }  
        return 0; // в случае неудачного выполнения возвращаем 0  
    }  
    // метод для покупателей  
    synchronized int get() {  
        if(counter>0) //если хоть один товар присутствует  
        {  
            counter--; //берем товар  
            System.out.println ("на складе " + counter + " товар(ов)");  
            return 1; // в случае удачного выполнения возвращаем 1  
        }  
        return 0; // в случае неудачного выполнения возвращаем 0  
    }  
}
```


// поток производителя

```
class Producer extends Thread {  
    Store store; //объект склада, куда кладем товар  
    int product=5; // количество товаров, которые надо добавить  
  
    Producer(Store store)  
    {  
        this.store=store;  
    }  
  
    public void run() {  
        try  
        {  
            while(product>0){ //пока у производителя имеются товары  
                product=product-store.put(); //кладем один товар на склад  
                System.out.println  
                    ("производителю осталось произвести " + product + " ")  
                sleep(100); // время простоя  
            }  
        }  
        catch(InterruptedException e)  
        {  
            System.out.println ("поток производителя прерван");  
        }  
    }  
}
```

// поток покупателя

class Consumer **extends** Thread {

Store **store**; *//объект склада, с которого покупатель будет брать товар*
int product=0; *//текущее количество товаров со склада*

Consumer(Store store)
{
 this.store=store;
}

public void run() {
 try
 {

while(**product**<5) {*// пока количество товаров не будет равно 5*

product=**product**+**store**.get(); *//берем по одному товару со склада*
 System.**out**.println ("Потребитель купил " + **product** + " ")
 sleep(100);

 }

 }

catch(InterruptedException e)
 {
 System.**out**.println ("поток потребителя прерван");
 }

}

}

```
public static void main(String[] a
```

```
Store store = new Store();  
new Producer(store).start();  
new Consumer(store).start();
```

```
}
```

```
"C:\Program ...  
на складе 1 товар(ов)  
производителю осталось произвести 4  
на складе 0 товар(ов)  
Потребитель купил 1  
на складе 1 товар(ов)  
производителю осталось произвести 3  
на складе 0 товар(ов)  
Потребитель купил 2  
на складе 1 товар(ов)  
производителю осталось произвести 2  
на складе 0 товар(ов)  
Потребитель купил 3  
Потребитель купил 3  
на складе 1 товар(ов)  
производителю осталось произвести 1  
на складе 2 товар(ов)  
производителю осталось произвести 0  
на складе 1 товар(ов)
```

odConsum



```
"C:\Program ...  
на складе 1 товар(ов)  
производителю осталось произвести 4  
на складе 0 товар(ов)  
Потребитель купил 1  
на складе 1 товар(ов)  
производителю осталось произвести 3  
на складе 0 товар(ов)  
Потребитель купил 2  
на складе 1 товар(ов)  
производителю осталось произвести 2  
на складе 0 товар(ов)  
Потребитель купил 3  
на складе 1 товар(ов)  
производителю осталось произвести 1  
на складе 0 товар(ов)  
Потребитель купил 4  
на складе 1 товар(ов)  
производителю осталось произвести 0  
на складе 0 товар(ов)  
Потребитель купил 5
```

```
Process finished with exit code 0
```

Инструкция `synchronized`

```
synchronized (s) {
```

```
}
```

- ▶ Происходит блокировка объекта `s` и он становится недоступным для других синхронизированных методов и блоков
- ▶ Только для объектов
- ▶ Сужает область синхронизации

```

"C:\Program . public class StringThred {
A
AA      public static int counter = 0;
AAA      public static void main(String args[ ]) {
AAAB      final StringBuilder s = new StringBuilder();
AAABB      new Thread() {
AAABBB          public void run() {
                    do {
                        s.append("A");
                        System.out.println(s);
                    } while (StringThred.counter++ < 2);
                }
            }.start();
            new Thread() {
                public void run() {
                    while (StringThred.counter++ < 6) {
                        s.append("B");
                        System.out.println(s);
                    }
                }
            }.start();
        }
    }
}

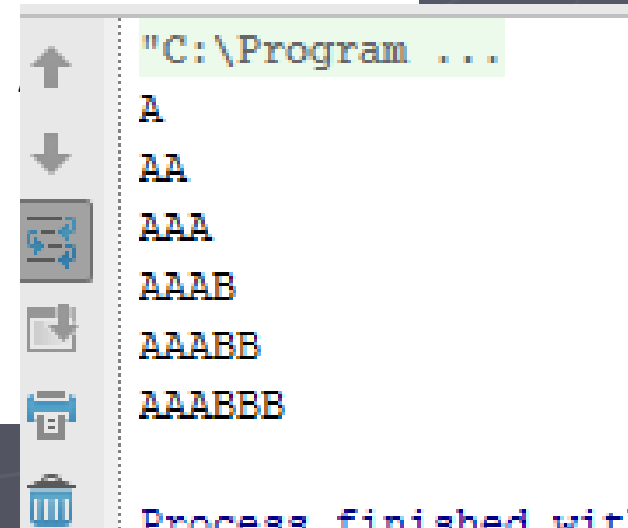
```

Process finish

```

public static int counter = 0;
    public static void main(String args[ ]) {
        final StringBuilder s = new StringBuilder();
        new Thread() {
            public void run() {
                synchronized (s) {
                    do {
                        s.append("A");
                        System.out.println(s);
                    } while (StringThred.counter++ < 2);
                } // конец synchronized
            }
        }.start();
        new Thread() {
            public void run() {
                synchronized (s) {
                    while (StringThred.counter++ < 6) {
                        s.append("B");
                        System.out.println(s);
                    }
                } // конец synchronized
            }
        }.start();
    }
}

```



Методы `wait()`, `notify()`

`wait()` - останавливает выполнение текущего потока и освобождает от блокировки захваченный объект

`notify()` - возвращает блокировку объекта потоку

- 1) никогда не переопределяются
- 2) вызываются только внутри синхронизированного блока или метода на объекте, монитор которого захвачен
- 3) в случае ошибки `IllegalMonitorStateException`

- **Задача:** Производитель – потребитель . 5 товаров - произвести и купить через склад. Одновременно на складе может находиться не более 3 товаров.



// Класс Магазин, хранящий произведенные товары

```
class StoreMMM{
    private int product=0;
    private boolean available = false;

    public synchronized void get() {
        while (product<1) {
            try { wait(); }
            catch (InterruptedException e) { }
        }
        product--;
        System.out.println("Россия купила 1 трактор");
        System.out.println("Товаров на складе: " + product);
        notify();
    }

    public synchronized void put() {
        while (product>=3) {
            try { wait(); }
            catch (InterruptedException e) { }
        }
        product++;
        System.out.println("МАЗ добавил 1 трактор");
        System.out.println("Товаров на складе: " + product);
        notify();
    }
}
```

освобождает монитор
объекта Store и
блокирует выполнение
метода get

// класс Производитель

```
class ProducerMAZ extends Thread{
    StoreMMM store;
    ProducerMAZ (StoreMMM store) {
        this.store=store;
    }
    public void run() {
        for (int i = 1; i < 6; i++) {
            store.put();
        }
    }
}
```

// Класс Потребитель

```
class ConsumerRUSSIA extends Thread{
    StoreMMM store;
    ConsumerRUSSIA (StoreMMM store) {
        this.store=store;
    }
    public void run() {
        for (int i = 1; i < 6; i++) {
            store.get();
        }
    }
}
```

```
public static void main(String[] args) {  
  
    StoreMMM store=new StoreMMM();  
    ProducerMAZ producer = new ProducerMAZ(store);  
    ConsumerRUSSIA consumer = new ConsumerRUSSIA(store);  
  
    producer.start();  
    consumer.start();  
}
```

- ▶ Взаимоблокировка
- ▶ Состояние гонки

```
"C:\Program ...  
MAZ добавил 1 трактор  
Товаров на складе: 1  
MAZ добавил 1 трактор  
Товаров на складе: 2  
MAZ добавил 1 трактор  
Товаров на складе: 3  
Россия купила 1 трактор  
Товаров на складе: 2  
Россия купила 1 трактор  
Товаров на складе: 1  
Россия купила 1 трактор  
Товаров на складе: 0  
MAZ добавил 1 трактор  
Товаров на складе: 1  
MAZ добавил 1 трактор  
Товаров на складе: 2  
Россия купила 1 трактор  
Товаров на складе: 1  
Россия купила 1 трактор  
Товаров на складе: 0
```

пакет `java.util.concurrent`

- ▶ Ограниченно потокобезопасные (thread safe) коллекции
 - `ConcurrentHashMap`, `ConcurrentLinkedQueue`
 - `CopyOnWriteArrayList` и `CopyOnWriteArraySet`
 - `BlockingQueue` и `BlockingDeque`

пакет `java.util.concurrent`

► Вспомогательные классы управления потоками

- Executor - запуск пула потоков и службы их планирования
- класс Exchanger - позволяет потокам обмениваться объектами
- классы-барьеры
 - CountdownLatch (потоки ожидать завершения заданного числа операций)
 - Semaphore (поток ожидает завершения действий в других потоках)
 - CyclicBarrier (потоки ожидают точки, после чего барьер снимается)
 - Phaser

java.util.concurrent.locks

- ▶ интерфейс Lock
- ▶ класс семафор ReentrantLock (отказы от блокировок)
- ▶ класс ReentrantReadWriteLock

Перечисление TimeUnit : NANOSECONDS, MICROSECONDS, MILLISECONDS, SECONDS, MINUTES, HOURS, DAYS

Блокирующие очереди

► интерфейсы BlockingQueue и BlockingDeque

```
void put(E e)
    // добавляет элемент в очередь + ждет
boolean offer(E e, long timeout, TimeUnit unit)
    // добавляет элемент + ждет timeout
boolean offer(E e)
    // добавляет элемент в очередь
E take()
    // извлекает и удаляет элемент + ждет
E poll(long timeout, TimeUnit unit)
    //извлекает и удаляет + ждет timeout
E poll()
    //извлекает и удаляет элемент из очереди
```

```
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;

public class BlockQu {

    public static void main(String[] args) {
        final BlockingQueue<String> queue =
            new ArrayBlockingQueue<String>(2);
        new Thread() {
            public void run() {
                for (int i = 1; i < 6; i++) {
                    try {
                        queue.put( " " + i); // добавление
                        System.out.println("Книгу " + i + " вернули");
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        }.start();
    }
}
```



```
new Thread() {  
    public void run() {  
        for (int i = 1; i < 6; i++) {  
            try {  
                Thread.sleep(1_000);  
                // извлечение одного  
                System.out.println("Книгу "  
                    + queue.take() + " взяли");  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}.start();  
}
```

"C:\Program ...

Книгу 1 вернули

Книгу 2 вернули

Книгу 1 взяли

Книгу 3 вернули

Книгу 2 взяли

Книгу 4 вернули

Книгу 3 взяли

Книгу 5 вернули

Книгу 4 взяли

Книгу 5 взяли

Process finished with exit

Семафоры

- ▶ Для управления доступом - счетчик (количество разрешений)
- ▶ счетчик > 0 , поток получает доступ ; счетчик $--$;
- ▶ Поток — закончил \rightarrow счетчик $++$;
- ▶ счетчик $==$ нулю \rightarrow поток блокируется и ждет.

Семафор всегда устанавливается на предельное положительное число потоков

```
Semaphore(int permits)
```

```
Semaphore(int permits, boolean fair)
```

fair = true -> порядок соот. запросам

= false -> порядок не определен

```
void acquire() // получение одного разрешения
```

```
void acquire(int permits) // нескольких разрешений
```

```
void release() //освобождение разрешения
```

```
void release(int permits)
```

```
class CommonResource{ int x=100; }
class CountThread extends Thread{

    CommonResource res;
    Semaphore sem;
    String name;
    CountThread(CommonResource res, Semaphore sem, String name){
        this.res=res;
        this.sem=sem;
        this.name=name;
    }

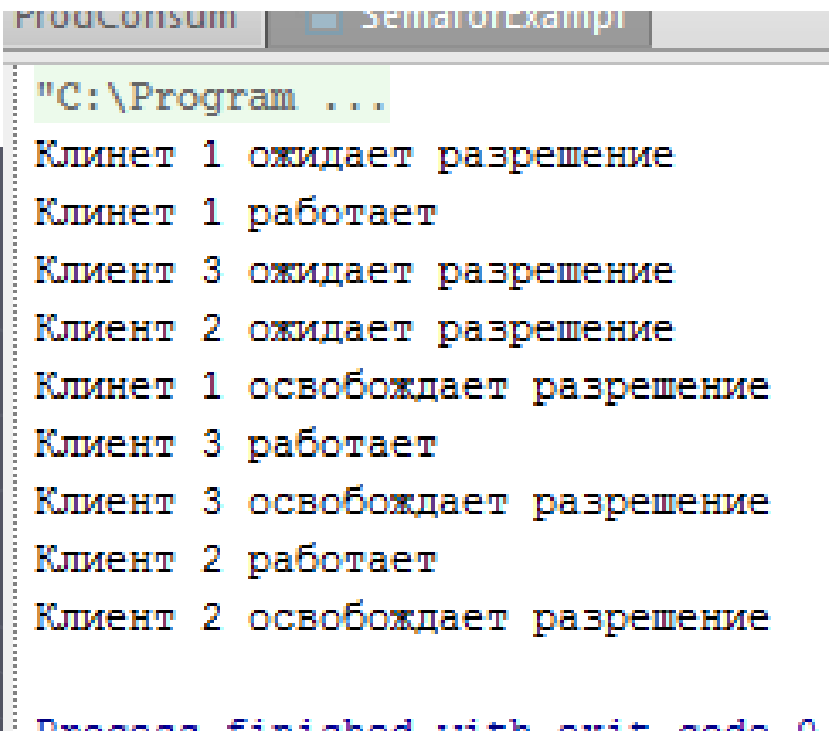
    public void run(){
        try{
            System.out.println(name + " ожидает разрешение");
            sem.acquire();
            {System.out.println(name + " работает");
                Thread.sleep(1000);
            }
        }
        catch (InterruptedException e)
            {System.out.println(e.getMessage());}

        System.out.println(name + " освобождает разрешение");
        sem.release();
    }
}
```

```
import java.util.concurrent.Semaphore;

public class SemaforExamp1 {
    public static void main(String[] args) {

        Semaphore sem = new Semaphore(1); // 1 разрешение
        CommonResource res = new CommonResource();
        (new CountThread(res, sem, "Клинет 1")).start();
        (new CountThread(res, sem, "Клиент 2")).start();
        (new CountThread(res, sem, "Клиент 3")).start();
    }
}
```



```
Process Console | SemaforExamp1
"C:\Program ...
Клинет 1 ожидает разрешение
Клинет 1 работает
Клиент 3 ожидает разрешение
Клиент 2 ожидает разрешение
Клинет 1 освобождает разрешение
Клиент 3 работает
Клиент 3 освобождает разрешение
Клиент 2 работает
Клиент 2 освобождает разрешение
Process finished with exit code 0
```

► Задача с пулом ресурсов: есть число огранич. ресурса (соединений) и клиенты – их много (ждут и уходят)

```
public class ConnectionPool <T> {  
  
    private final static int POOL_SIZE = 3; // размер пула  
    private final Semaphore semaphore = new Semaphore(POOL_SIZE, true);  
    private final Queue<T> resources = new LinkedList<T>();  
    public ConnectionPool(Queue<T> source) {  
        resources.addAll(source);  
    }  
    public T getResource(long maxWaitMillis) throws Exception {  
        try {  
            if (semaphore.tryAcquire(maxWaitMillis, TimeUnit.MILLISECONDS)) {  
                T res = resources.poll();  
                return res;  
            }  
        } catch (InterruptedException e) {  
            throw new Exception(e);  
        }  
        throw new Exception(":превышено время ожидания");  
    }  
    public void returnResource(T res) {  
        resources.add(res); // возвращение экземпляра в пул  
        semaphore.release();  
    }  
}
```

```
class Connect {  
    private int connectID;  
    public Connect(int id) {  
        super();  
        this.connectID = id;  
    }  
    public int getConnectID() {  
        return connectID;  
    }  
    public void setConnectID(int id) {  
        this.connectID = id;  
    }  
    public void using() {  
        try {  
            // использование соединения  
            Thread.sleep(new Random().nextInt(500));  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```

class Client extends Thread {
    private boolean reading = false;
    private ConnectionPool<Connect> pool;
    public Client (ConnectionPool<Connect> pool) {
        this.pool = pool;
    }
    public void run() {
        Connect connect = null;
        try {
            connect = pool.getResource(500); // изменить на 100
            reading = true;
            System.out.println("Соединение Клиент #" + this.getId()
                               + " соединение #" + connect.getConnectID());
            connect.using();
        } catch (Exception e) {
            System.out.println("Клиент #" + this.getId() +
                               " отказано в соединении ->"
                               + e.getMessage());
        } finally {
            if (connect != null) {
                reading = false;
                System.out.println("Соединение Клиент #" + this.getId() + " : "
                                   + connect.getConnectID() + " отсоединился");
                pool.returnResource(connect);
            }
        }
    }
    public boolean isReading() { return reading; }
}

```



```

public class Runner {
    public static void main(String[] args) {
        LinkedList<Connect> list = new LinkedList<Connect>() {
            {
                this.add(new Connect(1));
                this.add(new Connect(2));
                this.add(new Connect(3));
            }
        };
        ConnectionPool<Connect> pool = new ConnectionPool<>(list);
        for (int i = 0; i < 10; i++) {
            new Client(pool).start();
        }
    }
}

```

"C:\Program ...

```

Соединение Клиент #11  соединение #1
Соединение Клиент #17  соединение #2
Соединение Клиент #16  соединение #3
Соединение Клиент #16 : 3 отсоединился
Соединение Клиент #15  соединение #3
Соединение Клиент #11 : 1 отсоединился
Соединение Клиент #19  соединение #1
Соединение Клиент #19 : 1 отсоединился
Соединение Клиент #13  соединение #1
Соединение Клиент #17 : 2 отсоединился
Соединение Клиент #12  соединение #2
Соединение Клиент #15 : 3 отсоединился
Клиент #20 отказано в соединении
->:превышено время ожидания
Соединение Клиент #14  соединение #3
Клиент #18 отказано в соединении
->:превышено время ожидания
Соединение Клиент #12 : 2 отсоединился
Соединение Клиент #13 : 1 отсоединился
Соединение Клиент #14 : 3 отсоединился

```

Барьеры

- Задачи разделены на подзадачи и выполняются параллельно. По достижении некоторой данной точки всеми параллельными потоками подводится итог и определяется общий результат.

CyclicBarrier

```
int await()  
int await(long timeout, TimeUnit unit)
```

останавливают поток, использующий барьер до тех пор,
пока число потоков достигнет заданного числа в классе-барьере

```
boolean isBroken() // проверяет состояние барьера  
reset() // сбрасывает состояние барьера к моменту инициализации  
int getNumberWaiting() // позволяет определить число ожидаемых  
                        // барьером потоков до его снятия
```

Задача – аукцион, предложения, при достижении заданного числа потоков запускается барьер – тоже поток, который определяет победителя

```
class Auction{  
    private ArrayList<Bid> bids;  
    private CyclicBarrier barrier;  
    public final int BIDS_NUMBER = 5;  
  
    public Auction() {  
        this.bids = new ArrayList<Bid>();  
        this.barrier = new CyclicBarrier  
            (this.BIDS_NUMBER, new Runnable() {  
                public void run() {  
                    Bid winner = Auction.this.defineWinner();  
                    System.out.println("Ставка #" +  
                        winner.getBidId() +  
                        ", цена:" + winner.getPrice()  
                        + " победил!");  
                }  
            });  
    }  
}
```

```
public CyclicBarrier getBarrier() {  
    return barrier;  
}  
public boolean add(Bid e) {  
    return bids.add(e);  
}  
public Bid defineWinner() {  
    return Collections.max(bids, new Comparator<Bid>() {  
        @Override  
        public int compare(Bid ob1, Bid ob2) {  
            return ob1.getPrice() - ob2.getPrice();  
        }  
    });  
}  
}
```

```
class Bid extends Thread {  
    private Integer bidId;  
    private int price;  
    private CyclicBarrier barrier;  
    public Bid(int id, int price, CyclicBarrier barrier) {  
        this.bidId = id;  
        this.price = price;  
        this.barrier = barrier;  
    }  
    public Integer getBidId() {  
        return bidId;  
    }  
    public int getPrice() {  
        return price;  
    }  
}
```

@Override

public void run() {

try {

System.out.println("Клиент " + this.bidId
+ " определил цену.");

Thread.sleep(new Random().nextInt(3000)); // время на раздумье
// определение уровня повышения цены

int delta = new Random().nextInt(50);

price += delta;

System.out.println("Ставка " + this.bidId + " : " + price);

this.barrier.await(); // остановка у барьера

System.out.println("Подождите..."); // проверить кто выиграл
// и оплатить в случае победы ставки

} catch (BrokenBarrierException e) {

e.printStackTrace();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

```

public class AuctionRunner {
    public static void main(String[ ] args) {
        Auction auction = new Auction();

        int startPrice = new Random().nextInt(100);

        for (int i = 0; i < auction.BIDS_NUMBER; i++) {
            Bid thread =
                new Bid(i, startPrice, auction.getBarrier());

            auction.add(thread);
            thread.start();
        }
    }
}

```

```

Клиент 0 определил цену.
Клиент 1 определил цену.
Клиент 4 определил цену.
Клиент 2 определил цену.
Клиент 3 определил цену.
Ставка 1 : 47
Ставка 0 : 41
Ставка 4 : 46
Ставка 2 : 34
Ставка 3 : 64
Ставка #3, цена:64 победил!
Подождите...
Подождите...
Подождите...
Подождите...
Подождите...

```

```
public class AuctionRunner {  
    public static void main(String[] args) {  
        Auction auction = new Auction();  
        int startPrice = new Random().nextInt(100);  
        for (int i = 0; i < 2; i++) {  
            Bid thread =  
                new Bid(i, startPrice, auction.getBarrier());  
            auction.add(thread);  
            thread.start();  
        }  
    }  
}
```

"C:\Program ...

Клиент 0 определил цену.

Клиент 1 определил цену.

Ставка 1 : 44

Ставка 0 : 46


```
public class AuctionRunner {  
    public static void main(String[] args) {  
        Auction auction = new Auction();  
        int startPrice = new Random().nextInt(100);  
        for (int i = 0; i < 10; i++) {  
            Bid thread =  
                new Bid(i, startPrice, auction);  
            auction.add(thread);  
            thread.start();  
        }  
    }  
}
```

```
"C:\Program ...  
Клиент 0 определил цену.  
Клиент 2 определил цену.  
Клиент 4 определил цену.  
Клиент 6 определил цену.  
Клиент 8 определил цену.  
Клиент 1 определил цену.  
Клиент 9 определил цену.  
Клиент 5 определил цену.  
Клиент 3 определил цену.  
Клиент 7 определил цену.  
Ставка 2 : 96  
Ставка 5 : 54  
Ставка 7 : 72  
Ставка 1 : 89  
Ставка 9 : 79  
Ставка #2, цена:96 победил!  
Подождите...  
Подождите...  
Подождите...  
Подождите...  
Подождите...  
Ставка 8 : 61  
Ставка 0 : 65  
Ставка 3 : 51  
Ставка 6 : 98  
Ставка 4 : 77  
Ставка #6, цена:98 победил!  
Подождите...  
Подождите...  
Подождите...  
Подождите...  
Подождите...
```

Щеколда - CountdownLatch

инициализируется начальным значением числа ожидающих потоков

```
await() //останавливает поток без  
countDown() //значение счетчика снижается «щеколда»  
//сдвигается на единицу.
```

- 1) Набор заданий (тестов)
- 2) Студенту предлагается для выполнения набор заданий. Он выполняет их и переходит в режим ожидания оценок по всем заданиям, чтобы вычислить среднее значение оценки.
- 3) Преподаватель (Tutor) проверяет задание и после каждого проверенного задания сдвигает «щеколду» на единицу. Когда все задания студента проверены, барьер снимается, производятся вычисления в классе Student.

Обмен блокировками

► класс Exchanger

T exchange (T ob)

► Производитель – Потребитель

Сколько продали

Сколько произвели

Цена изменяется в зависимости от разницы

```
class Item {  
  
}  
  
class Subject {  
    protected static Exchanger<Item> exchanger = new Exchanger<>();  
    protected Item item;  
}  
  
class Producer extends Subject implements Runnable {  
  
    public void run() {  
        try {  
            item = exchanger.exchange(item);  
            if (proposedNumber <= item.getNumber()) {  
                // ...  
            }  
        }  
    }  
}  
  
class Consumer extends Subject implements Runnable {  
    public void run() {  
        try {  
            item = exchanger.exchange(item); // обмен  
        }  
    }  
}
```

Интерфейс Lock

- ▶ провести опрос о блокировании
- ▶ установить время ожидания блокировки
- ▶ условия ее прерывания

ReentrantLock пакет *java.util.concurrent.locks*

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**.

не ожидает получения блокировки, если она недоступна

void unlock(): снимает блокировку

```
class CommonResource{
```

```
    int x=0;
```

```
}
```

```
class CountThread extends Thread{
```

```
    CommonResource res;
```

```
    ReentrantLock locker;
```

```
    public void run() {
```

```
        try{
```

```
            locker.lock(); // устанавливаем блокировку
```

```
            res.x=1;
```

```
            Thread.sleep(100);
```

```
        }
```

```
        catch (InterruptedException e) {
```

```
            System.out.println(e.getMessage());
```

```
        }
```

```
        finally{
```

```
            locker.unlock(); // снимаем блокировку
```

```
        }
```

```
    }
```

```
}
```

```
public static void main(String[] args) {  
  
    CommonResource commonResource= new CommonResource();  
    ReentrantLock locker = new ReentrantLock(); // создаем заглушку  
    for (int i = 1; i < 6; i++){  
  
        CountThread t = new CountThread(commonResource, locker);  
        t.setName("Поток "+ i);  
        t.start();  
    }  
}
```

"C:\Program ...

Поток 1 1

Поток 3 1

Поток 5 1

Поток 2 1

Поток 4 1

Класс ExecutorService

```
execute(Runnable thread) // запускает потоки
submit(Callable<T> task) // запускает потоки
                        //с возвращаемым значением
shutdown() //останавливает все запущенные
newSingleThreadExecutor() //позволяет
                        //исполнителю запускать только один поток
newFixedThreadPool(int numThreads) //не более
                        //чем указано в параметре
```


Phaser

позволяет синхронизировать потоки, представляющие отдельную фазу или стадию выполнения общего действия.

- ▶ Phaser определяет объект синхронизации, который ждет, пока не завершится определенная фаза. Затем Phaser переходит к следующей стадии или фазе и снова ожидает ее завершения.

```
Phaser()  
Phaser(int parties)  
Phaser(Phaser parent)  
Phaser(Phaser parent, int parties)
```

```
int register() //регистрирует поток - номер текущей фазы  
int arrive() // сообщает, что поток завершил фазу  
int arriveAndAwaitAdvance(): // аналогичен - заставляет phaser  
// ожидать завершения фазы всеми остальными потоками  
int arriveAndDeregister() //сообщает о завершении всех фаз  
// и снимает ее с регистрации  
int getPhase() // возвращает номер текущей фазы
```

- 1) Создается объект
- 2) Репегистрировать все потоки `register()`
- 3) Потоки работают - выполняют фазы
- 4) синхронизатор `Phaser` ждет, пока все потоки не завершат выполнение фазы.
- 5) поток должен вызвать метод `arrive()` или `arriveAndAwaitAdvance()`.
- 6) После этого синхронизатор переходит к следующей фазе.

```
class PhaseThread extends Thread{
```

```
    Phaser phaser;
```

```
    String name;
```

```
    PhaseThread(Phaser p, String n){
```

```
        this.phaser=p;
```

```
        this.name=n;
```

```
        phaser.register();
```

```
    }
```

```
    public void run() {
```

```
        try {
```

```
            System.out.println(name + " выполняет фазу 0"); // Process finished with exit cc
```

```
            phaser.arriveAndAwaitAdvance(); // сообщаем, что первая фаза достигнута
```

```
            System.out.println(name + " выполняет фазу " + phaser.getPhase());
```

```
            phaser.arriveAndAwaitAdvance(); // сообщаем, что вторая фаза достигнута
```

```
            Thread.sleep(200);
```

```
            System.out.println(name + " выполняет фазу " + phaser.getPhase());
```

```
            phaser.arriveAndDeregister(); // сообщаем о завершении фаз и удаляем
                                           //с регистрации объекты
```

```
        } catch (InterruptedException e) {
```

```
            e.printStackTrace();
```

```
        }
```

```
    }
```

```
}
```

```
поток 2 выполняет фазу 0
```

```
поток 1 выполняет фазу 0
```

```
фаза 1 завершена
```

```
поток 1 выполняет фазу 1
```

```
поток 2 выполняет фазу 1
```

```
фаза 2 завершена
```

```
поток 2 выполняет фазу 2
```

```
поток 1 выполняет фазу 2
```

```
фаза 3 завершена
```

```
public static void main(String[] args) {

    Phaser phaser = new Phaser(1);
    (new PhaseThread(phaser, "поток 1")).start();
    ( new PhaseThread(phaser, "поток 2")).start();

    // ждем завершения фазы 0
    phaser.arriveAndAwaitAdvance();
    System.out.println("фаза " + phaser.getPhase() + " завершена");

    // ждем завершения фазы 1
    phaser.arriveAndAwaitAdvance();
    System.out.println("фаза " + phaser.getPhase() + " завершена");

    // ждем завершения фазы 2
    phaser.arriveAndAwaitAdvance();
    System.out.println("фаза " + phaser.getPhase() + " завершена");

    phaser.arriveAndDeregister();
}
}
```