Теория и технология программирования

Программирование на языке Java

Лекция 13. Основы многопоточного программирования (продолжение)

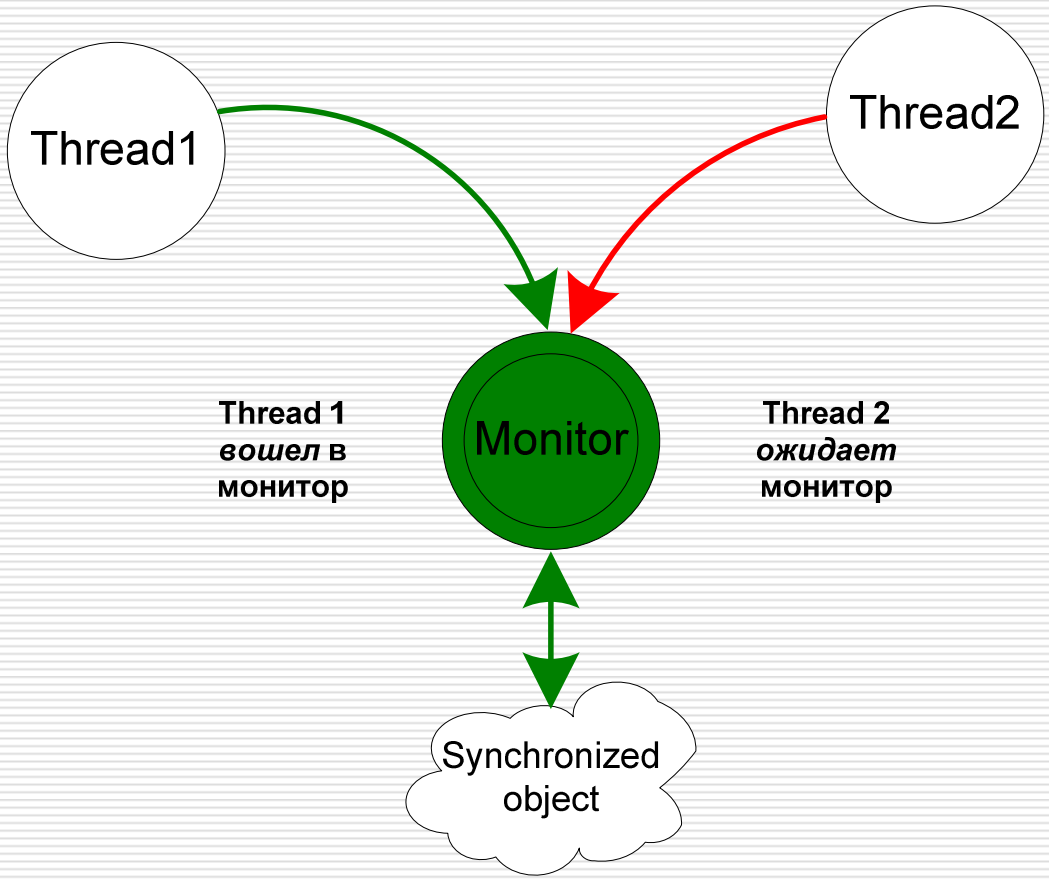
Глухих Михаил Игоревич, к.т.н., доц.
mailto: glukhikh@mail.ru

Синхронизация потоков

- Имеется две схемы синхронизации
  - Синхронизация по ресурсам (управление ресурсами)
    - Объекты совместно используют ресурсы данных
  - Синхронизация по событиям (управление временем)
    - Для совместной работы потоки уступают процессорное время друг другу и/или уведомляют друг друга о возможности продолжения работы

Синхронизация по ресурсам

- Если один поток работает с некоторым объектом, второй поток не может работать с ним в это же время



Пример с банковским счетом

- Есть денежный счет
- С ним одновременно работают клиент, забирающий деньги из банкомата, и банк, переводящий на него деньги

Класс Deposit

```
public class Deposit {
    private int balance;
    public Deposit( int startBalance ) {
        balance = startBalance;
    }
    public void setBalance( int newBalance )
        throws InterruptedException {
        Thread.sleep(10);
        balance = newBalance;
    }
    public int getBalance() throws InterruptedException {
        Thread.sleep(10);
        return balance;
} }
```

Класс Client

```
public class Client extends Thread {
    private final Deposit deposit;
    private final int change;
    Client( Deposit deposit, int change ) {
        super( "Bank client" );
        this.deposit = deposit;
        this.change = change;
    }
    // ...
}
```

Класс Client

```
public void run() {
    try {
        int balance = deposit.getBalance();
        System.out.println(
  "Client: balance before transaction: " + balance );
        balance += change;
        deposit.setBalance(balance);
        System.out.println( "Client: balance: " +
            balance );
    } catch( InterruptedException e ) {
        System.out.println(
          "Interrupting client thread");
    }
}
```

Главная функция

```
Deposit deposit = new Deposit( 300 );
Client client = new Client( deposit, -100 );
Client bank = new Client( deposit, 1000 );
bank.start();
client.start();
try {
    bank.join();
    client.join();
    int balance = deposit.getBalance();
    System.out.println(
       "Balance at termination: " + balance );
} catch( InterruptedException e ) {
    System.out.println( "Enexpected");
}
```

Итог

- Как вы думаете, каким получится результат?

Проблема

- Участок между getBalance() и setBalance() не должен одновременно выполняться несколькими потоками
- Иначе говоря, участок должен быть атомарным

Вариант решения

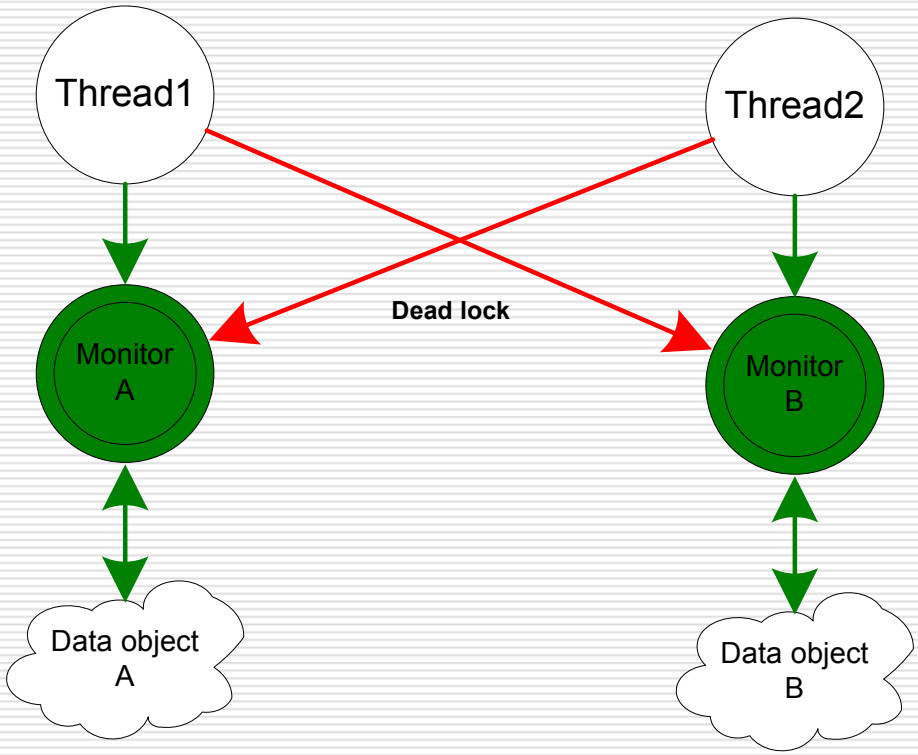
```
public void run() {
    try {
        synchronized(deposit) {
            int balance = deposit.getBalance();
            System.out.println( "Client: balance before
                    transaction: " + balance );
            balance += change;
            deposit.setBalance(balance);
        }
        System.out.println( "Client: balance: " + balance );
    } catch( InterruptedException e ) {
        System.out.println( "Interrupting client thread");
    }
}
```

Другой вариант решения

```
public class Deposit {
    private int balance;
    public Deposit( int startBalance ) {
        balance = startBalance;
    }
    synchronized public int changeBalance(int change)
            throws InterruptedException {
        Thread.sleep(10);
        return balance += change;
    }
    public int getBalance() throws InterruptedException {
        Thread.sleep(10);
        return balance;
} }
```

Взаимная блокировка потоков

- Два потока имеют циклическую зависимость от пары синхронизированных объектов
- Трудна для отладки
  - происходит редко
  - может включать больше двух потоков



Взаимная блокировка потоков – пример

```
public void run() {
    // ...
    synchronized (listA) {
        System.out.println("Second thread locks listA");
        i++;
        listA.add(i);
        synchronized (listB) {
            System.out.println("Second thread locks listB");
            i++;
            listB.add(i);
        }
    }
}
```

Взаимная блокировка потоков – пример

```
synchronized (listB) {
    System.out.println("Second thread locks listB");
    j++;
    listB.add(j);
    synchronized (listA) {
        System.out.println("Second thread locks listA");
        j++;
        listA.add(j);
    }
}
```

Синхронизация "ожидание-уведомление"

- Для этой цели имеется несколько методов класса Object; их можно вызывать из synchronized методов и блоков
- void wait() – уступить монитор и перейти в режим ожидания
- void notify() – пробудить один из потоков, вызвавший wait() на данном объекте
- void notifyAll() – пробудить все потоки, вызвавшие wait() на данном объекте

Пример

- Разделим операцию "изменение баланса" на две:
  - получение денег (getMoney)
  - добавление денег (putMoney)
- Запретим иметь отрицательный баланс
- Если при получении денег на счету недостаточно, можно подождать, пока их добавят

Класс Deposit, метод getMoney

```
public synchronized int getMoney(int requested)
        throws InterruptedException {
    Thread.sleep(10);
    if (requested > balance) {
        wait(5000);
        if (requested > balance) {
            int oldBalance = balance;
            balance = 0;
            return oldBalance;
        }
    }
    balance -= requested;
    return requested;
}
```

Класс Deposit, метод putMoney

```
public synchronized int putMoney(int sum)
        throws InterruptedException {
    Thread.sleep(10);
    balance += sum;
    notifyAll();
    return balance;
}
```

Главная функция

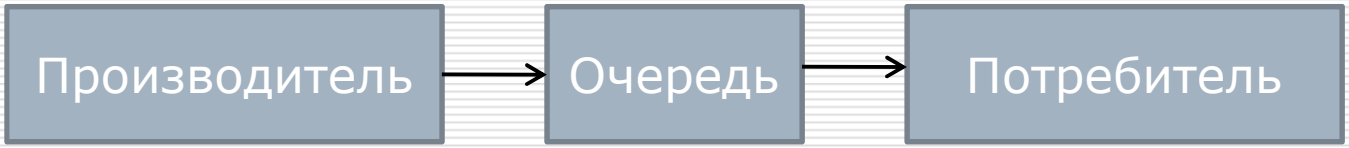
```
Deposit deposit = new Deposit( 300 );
GetMoneyThread client = new GetMoneyThread(
    "Client", deposit, 700 );
PutMoneyThread bank = new PutMoneyThread(
    "Bank", deposit, 500 );
try {
    client.start();
    Thread.sleep(1000);
    bank.start();
    bank.join();
    client.join();
    int balance = deposit.getBalance();
    System.out.println(
        "Balance at termination: " + balance );
} // ...
```

Ping-Pong

- Пример с двумя нитями, передающими друг другу некоторый ресурс (шарик, Ball)
- Пока шарик принадлежит одной нити, другая ждёт
- Отдавая шарик, нить сообщает об этом другой нити и начинает ждать сама

Шаблон Producer - Consumer

- Производитель – потребитель
- Один поток генерирует данные (или, например, получает их из сети) – производитель
- Второй поток обрабатывает эти данные – потребитель
- Сами данные при этом содержатся в некотором контейнере, обычно – в очереди



Элементарный пример

- Поток-производитель читает строчки из файла и записывает их в очередь
- Поток-потребитель ищет строчки, все символы которых различны, и выводит их в выходной файл

Элементарный пример

- См.