Прекрасный мир Java

Часть 11.

Многопоточное программирование

Потоками называются средства, позволяющие нескольким операциям сосуществовать в рамках одного процесса.

Любая программа Java использует хотя бы один поток — главный поток.

При запуске программы JVM создает главный поток и вызывает метод main() внутри этого потока.

Зачем нужны потоки?

- Повышение отзывчивости интерфейса пользователя
- Использование преимущества многопроцессорных (многоядерных) систем
- Упрощение моделирования
- Выполнение асинхронной или фоновой обработки

Многопоточность != Параллельность

Класс Thread

Создание и запуск потока — не одно и то же.

start()
join()
sleep()
interrupt()
setDaemon()
setPriority()
isAlive()

Основные проблемы многопоточности

- Совместное одновременное использование разделяемых ресурсов из нескольких потоков
- Синхронизация потоков
- Взаимная блокировка потоков

```
synchronized (<объект синхронизации>) {
... // защищаемый код
}
```

1. проверяется, есть ли в данный момент поток, работающий внутри (*возможно другой*) секции synchronized с *тем же объектом* синхронизации.
2. если такой поток есть, то выполнение текущего потока останавливается до тех пор, пока не останется *ни одного потока*, работающего внутри секции synchronized *с тем же объектом* синхронизации.
3. продолжается выполнение защищаемого кода.

Все три шага работают как одна атомарная операция

```
synchronized void doSomething() {
... // защищаемый код
}
```

Эквивалент:

```
void doSomething() {
   synchronized(this) {
      // защищаемый код
   }
}
```

```
class SomeClass {
   static synchronized void doSomething() {
      // защищаемый код
   }
}
```

Эквивалент:

```
class SomeClass {
   static void doSomething() {
      synchronized(SomeClass.class) {
         // защищаемый код
      }
   }
}
```

Никакой синхронизации для различных объектов

```
Object objSync1 = new Object();
Object objSync2 = new Object();
// первый блок
synchronized(objSync1) {
   ...
}
// второй блок
synchronized(objSync2) {
   ...
}
```

void wait(long timeout) - останавливает работу текущего потока на заданное время (timeout) или до получения оповещения через объект синхронизации, у которого вызван метод wait, если оно поступит до истечения периода ожидания. Период ожидания равный 0 означает бесконечное ожидание.
void wait() - эквивалентен вызову wait(0).
void notify() - послать оповещение какому-то одному из потоков, находящемуся в методе wait() данного объекта синхронизации.
void notifyAll() - послать оповещение всем потокам, находящимся в методе wait() данного объекта синхронизации.

Правила многопоточных приложений

1. Блоки synchronized должны быть максимально компактными.
2. Чрезмерное использование блоков synchronized снижает быстродействие программы.
3. Вложенные блокировки должны быть согласованы, чтобы избегать взаимной блокировки потоков.

Пример взаимной блокировки потоков

```
static Object lock1 = new Object();
static Object lock2 = new Object();
static void method1() {
   synchronized(lock1) {
      synchronized(lock2) {...}
   }
}
static void method2() {
   synchronized(lock2) {
      synchronized(lock1) {...}
   }
}
```