Лекция 7

Многопоточность

Содержание

- Возможности и преимущества многопоточности
- Способы реализации
- Механизмы синхронизации

Возможности и преимущества многопоточности

- Одновременное обслуживание нескольких клиентов в конфигурации клиент-сервер
- Реализация систем с активным участием пользователя в процессе вычислений (например, игры)
- Оптимизация скорости работы за счет распараллеливания работы с «медленными» периферийными устройствами

Многопоточность не ускоряет систему

Приоритеты и типы потоков

- Приоритет потока определяет долю квантов времени, выделяемых ему.

  потоки низкого приоритета все равно продолжают исполняться
- Потоки-демоны (daemon) – потоки специального назначения, чаще всего – обслуживающие
- Приложение исполняется, пока существует хотя бы один «не-daemon» поток

Поточная модель Java

- Поток – экземпляр класса Thread
- Методы класса Thread:

`public static Thread currentThread()` – возвращает ссылку на поток из которого вызывается метод;

`final String getName()` – получить имя потока;

`final void setName(String s)` – задать имя потока;

`final int getPriority()` – приоритет потока (+ `setPriority(int n)`, `MIN_PRIORITY = 1`, `MAX_PRIORITY = 10`, `NORM_PRIORITY = 5`);

`final boolean isAlive()` – позволяет выяснить исполняется поток или нет;

`final void join() throws InterruptedException` – ожидание завершения потока;

`static void sleep(long n) throws InterruptedException` – приостанавливает выполнение потока на n миллисекунд;

`void run()` – определяет точку входа в поток;

`void start()` – запускает поток, вызывая его метод `run()`

Поточная модель Java

- Конструкторы класса Thread:

```
Thread(Runnable threadOb);
Thread(Runnable threadOb, String name);
…
```

при запуске программы начинает выполняться главный поток, в котором уже могут порождаться дочерние. Главный поток создается автоматически. В идеале программа начинает выполняться с главного потока и завершается с завершением главного потока.

Как создать поток?

- Поток в Java – экземпляр класса Thread
  - Реализуем класс-наследник Thread
  - Переопределяем метод `void run()`
  - Создаем экземпляр класса
  - Вызываем метод… **`start()`**
  - Виртуальная машина Java принимает решение о моменте запуска потока, производит его инициализацию и сама вызывает метод `run()`

Как создать поток?

```
public class MyThread extends Thread {
   public void run() {
      // вычисления
   }
   // метод start() реализовывать нельзя!
}

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

Как создать поток?

- Наследование от Thread может привести к конфликту
- Реализуем интерфейс Runnable
  - Создаем класс, реализующий интерфейс Runnable
  - Реализуем метод `void run()`
  - Создаем экземпляр класса
  - Создаем экземпляр класса Thread, передавая в виде параметра ссылку на созданный экземпляр Runnable
  - Вызываем метод **`start()`** у класса Thread

Как создать поток?

```
public class MyThread
          implements Runnable {
   public void run() {
      // вычисления
   }
}

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

Методы управления потоком

- Изнутри
  - `static void sleep(int mseconds)` – приостановка работы на указанное число миллисекунд
  - `static void yield()` – приостановка работы и передача управления другим потокам (если они есть)
- Снаружи
  - `interrupt()` – прерывание работы потока, у которого этот метод вызван. Порождает `InterruptedException` «внутрь» `run()`

Синхронизация

- При одновременной работе с общими переменными результат непредсказуем:
  - Изменение переменной = чтение; вычисления; запись (т.е. делается в несколько этапов)
  - Примеры: банковский счет, продажа билетов

Блокировка

- Блокировка устанавливается на объект
- Блокировка объекта может быть установлена только одним потоком
  - Прочие действия с объектом остаются доступными
  - Все другие потоки, попытавшиеся установить блокировку, ждут освобождения объекта
- При выполнении блокировки локальная память потока полностью синхронизируется с общей; при снятии – аналогично (в обратную сторону)

Блокировка используется для обеспечения предсказуемости изменений объекта.

Модификатор **synchronized**

- Объявление synchronized-блока

```
synchronized (object) {
    …
}
```

Устанавливается блокировка на `object`

- Объявление synchronized-метода

```
public void synchronized process() {
    …
}
```

Устанавливается блокировка на весь объект, содержащий synchronized-метод

Deadlock

- Взаимная блокировка потоков
  - После блокировки одного объекта поток пытается установить блокировку на второй;
  - Второй поток установил блокировку второго объекта и пытается заблокировать первый;
  - Оба потока находятся в режиме ожидания друг друга.
- В Java отсутствуют средства предотвращения или распознавания deadlock
- Также отсутствует проверка, заблокирован ли объект другим потоком

Вопросы синхронизации должны внимательно решаться на этапе проектирования