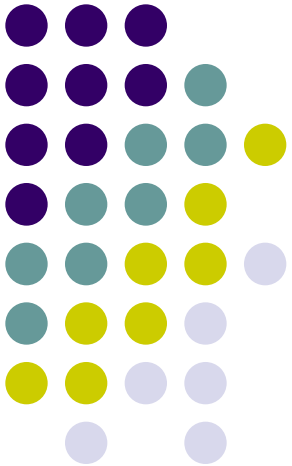
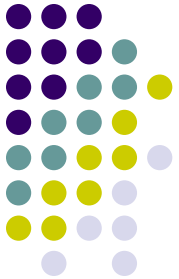
Java

Параллельное выполнение



Терминология

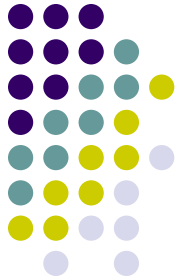


При переводе на русский ДВА английских термина имеют одинаковое значение – поток:

- **stream**
- **thread**

Thread имеет еще одно значение - нить

Поток и процесс



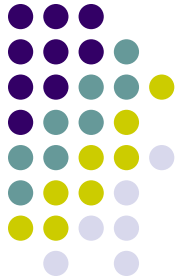
Процесс – это задача операционной системы.

- У него собственное адресное пространство.
- С ним может быть проассоциировано несколько потоков.

Поток – это гораздо более мелкая единица.

- Все потоки разделяют адресное пространство породившего их процесса и имеют доступ к одним данным.

Threads



Потоки – средство, которое помогает организовать одновременное выполнение нескольких задач, каждую в независимом потоке.

Потоки представляют собой классы, каждый из которых запускается и функционирует самостоятельно, автономно (или относительно автономно) от главного потока выполнения программы.

Существуют два способа создания и запуска потока:

- расширение класса **Thread**
- реализация интерфейса **Runnable**.

example01 : расширение класса Thread: Talk.java

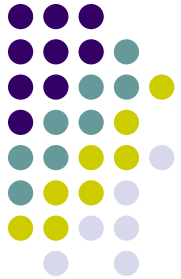
реализация интерфейса Runnable:

Walk.java,

WalkTalk.java

eck01.

Планировщик и сборщик мусора



Часть потокового механизма Java, обеспечивающая переключение процессора между потоками.

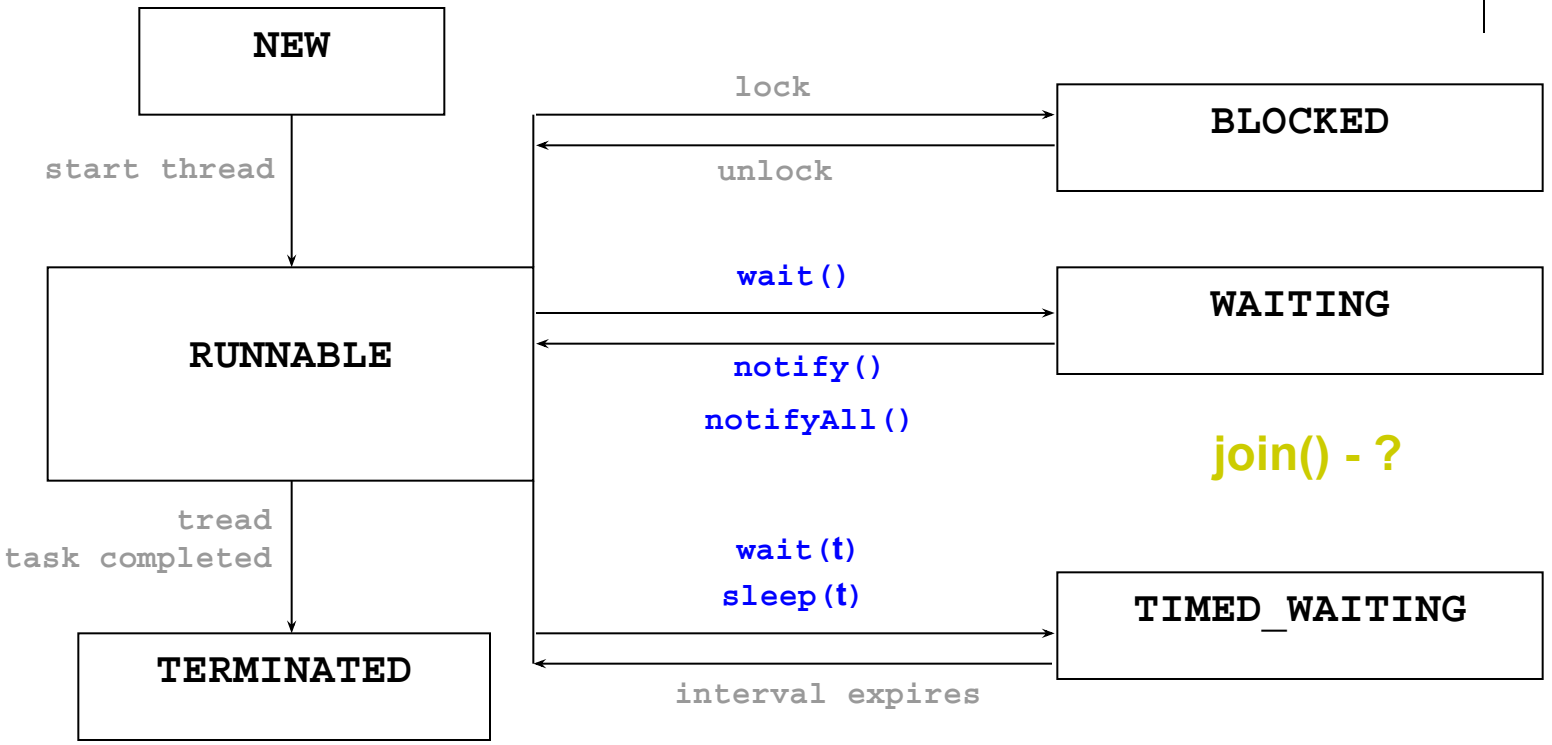
Если на компьютере установлено несколько процессоров, планировщик потоков автоматически распределяет потоки между разными процессорами.

Работа планировщика потоков недетерминирована.

Когда метод main() создает объекты-потоки Thread, он не сохраняет на них ссылки.

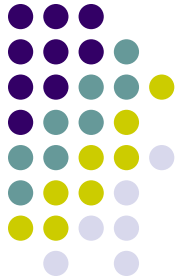
Каждый поток (Thread) самостоятельно «регистрирует» себя, то есть на самом деле ссылка на него где-то существует, и сборщик мусора не вправе удалить его объект.

Жизненный цикл потока



Метод **interrupt()** успешно завершает поток, если он **RUNNABLE**.
Иначе метод генерирует исключительные ситуации разного типа в зависимости от способа остановки потока.

Состояния потока

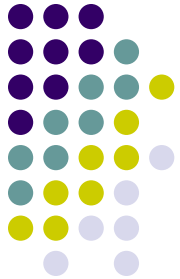


Поток может находиться в одном из состояний, соответствующих элементам статически вложенного перечисления **Thread.State**:

- **NEW** – поток создан, но еще не запущен;
- **RUNNABLE** – поток выполняется;
- **BLOCKED** – поток блокирован (ожидает входа в синхронизированный блок/метод);
- **WAITING** – поток ждет окончания работы другого потока;
- **TIMED_WAITING** – поток некоторое время ждет окончания другого потока;
- **TERMINATED** — поток завершен.

Получить значение состояния потока можно вызовом метода **getState()**.

Остановка потока



В Java НЕТ средств для принудительной остановки потока. (кроме **deprecated** метода **stop()**).

В Java принят уведомительный порядок остановки потока:

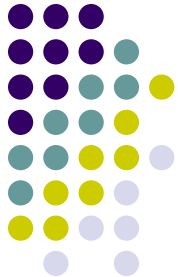
- либо при помощи существующих механизмов;
- либо созданных вручную.

Метод **interrupt()** класса **Thread** выставляет некоторый флаг.

Состояние этого флага можно проверить с помощью метода **isInterrupted()**.

Статический метод **interrupted()** производит проверку для текущего потока и **сбрасывает** флаг.
Если у потока были вызваны методы **sleep()** или **wait()** - ожидание прервется и будет выброшено исключение **InterruptedException**. Флаг в этом случае **не выставляется**.

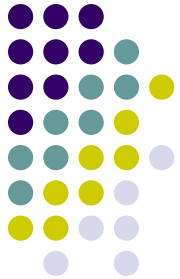
Приостановка потока



Методы класса **Thread**:

- унарные
  - **sleep(t)** – приостановить (задержать) выполнение потока на **t** миллисекунд.
  - **yield()** – может сделать некоторую паузу и позволить другим потокам начать выполнение своей задачи.
- для синхронизации с другими потоками
  - **join()** блокирует работу потока, в котором он вызван, до тех пор, пока не будет закончено выполнение вызывающего метод потока.
  - **wait(t)** – помещает поток в состояние ожидания на время t
  - **wait()** – помещает поток в состояние ожидания пока другой поток не вызовет **notify()** или **notifyAll()** для ожидающего объекта

Совместное использование ресурсов



В условиях многозадачности есть сразу несколько потоков, которые стремятся получить доступ к одному и тому же ограниченному ресурсу.

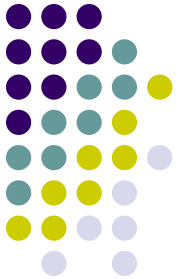
Поэтому фактически все многопоточные схемы ***синхронизируют доступ к разделяемым ресурсам***.

Доступ к разделяемому ресурсу в один момент времени может получить только один поток.

В Java есть встроенная поддержка для предотвращения конфликтов в виде ключевого слова **synchronized**.

Синхронизации не требуют только атомарные процессы по записи/чтению, не превышающие по объему 32 бит.

ненадежна и опасна



Атомарные операции

Атомарная операция – это операция, которую не может прервать планировщик потоков

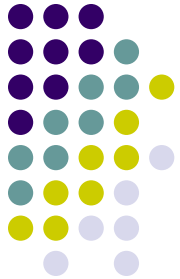
Ключевое слово **volatile** обеспечивает видимость в рамках приложения.

Если поле объявлено как **volatile**, это означает, что сразу же после записи в поле изменение будет отражено во всех последующих операциях чтения.

В Java SE5 появились специальные классы для выполнения атомарных операций с переменными.

Синхронизация влияет на производительность

Методы synchronized



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

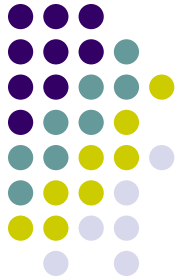
Изоляция снимается:

- когда поток полностью выполнит соответствующий метод
- если есть вызов метода **wait()** из изолированного метода

example04: синхронизация записи информации в файл :

MyThread.java
Synchro.java
SynchroThreads.java

Блоки synchronized

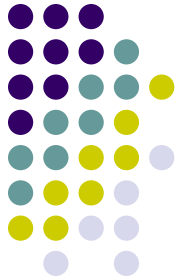


Синхронизировать объект можно при помощи синхронизированного блока кода.

В этом случае происходит блокировка объекта, указанного в инструкции **synchronized**, и он становится недоступным для других синхронизированных методов и блоков.

example05: блокировка объекта потоком: TwoThread.java

Мониторы



- то, что обеспечивает работу **synchronized-методов** и **synchronized-блоков**
- это средство обеспечения контроля за доступом к ресурсу

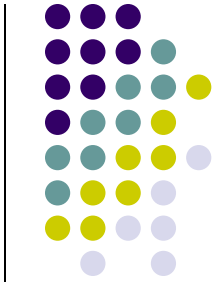
У монитора может быть максимум один владелец в каждый текущий момент времени.

В Java у **каждого экземпляра** объекта есть монитор.

Используется монитор так:

- любой *нестатический* synchronized-метод при своем вызове прежде всего пытается захватить монитор того объекта, у которого он вызван
  (на который он может сослаться как на **this**).
- Если это удалось – метод исполняется.
- Если нет – поток останавливается и ждет, пока монитор будет отпущен.

Статические методы



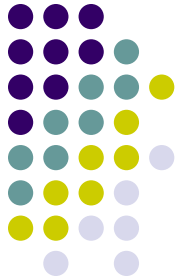
У статического метода нет ссылки **this**.

При синхронизации статического метода блокируется монитор соответствующего объекта класса **Class**.

Объекты класса Class существуют в единственном экземпляре только в пределах одного ClassLoader-а.

Резюме: Если два нестатических метода объявлены как synchronized, то в каждый момент времени *из разных потоков* **на одном объекте** может быть вызван только один из них. Поток, который вызывает метод первым, захватит монитор, и второму потоку придется ждать.

Взаимные блокировки deadlock



Предположим, что ***один поток*** уже захватил монитор на некотором объекте **x** и для продолжения работы ему нужно захватить монитор на объекте **y**.

Другой поток уже захватил монитор на объекте **y** и ему нужен монитор объекта **x**.

В результате оба потока будут ждать, пока нужный монитор освободится.