Многопоточность

- Многопоточность:
  - процесс меет минимум один главнй поток
  - Все потоки разделяют адресное пространство процесса
  - Стек у каждого потока свой
- Синхронизация потоков:
  - Совместный доступ к ресурсам
  - Ожидание результата вычислений
- Два способа создания потока:
  - Создать класс, наследующий Thread и переопределить метод run()
  - Создать класс, реализующий интерфейс Runnable и передать его экземпляр в конструткор Thread
- Запуск потока – метод start()
- Приостановка – метод sleep()
- Класс называется *потокобезопасный*, если его поведение корректно вне зависимости от способа использования в многопоточной среде
- Классы проблем многопоточных приложений:
  - Thread interference – атомарность чтения и записи, неопределенность параллелизма
  - Memory consistency – Синхронизация изменений в данных между разными потоками: happens-before
- Как бороться с этими проблемами:
  - не использовать общие переменные в разных потоках
  - обеспечить неизменяемость общих переменных (immutability)
  - Синхронизировать доступ к общим переменным, так что одновременно с ними будет работать только один поток
- Критическая секция – блок инструкций, в котором производится доступ к общим ресурсам, который не должен одновременно исполняться более, чем одним потоком
- Race conditions → most common → check then act → by the time you take the action the observation could have become invalid
- У каждого потока есть своя “рабочая” память, с который он взаимодействует. В целях оптимизации этот блок памяти не всегда переносится в основную память.

- *Synchronized* – метод устранить гонку и обеспечить memory consistency. Применим к:
  - методам, в том числе и статическим
  - отдельным блокам инструкций
- synchronized обеспечивает сериализацию (эксклюзивное исполнение) критических секций различными потоками.
- Synchronized упорядочивает блоки в контексте конкретных экземпляров (или экземпляра CharRunnable.class в случае статических методов)
- Общий контекст синхронизации
- Thread Contention:
  - Deadlock
  - Livelock
- Операции чтения и записи атомарны для всех типов кроме double и long. Чтение и запись атомарны для всех переменных, помеченных ключевым словом volatile. Гарантирует выполнение happens-before что повзоляет устранить memory consistency.
- Использование переменной состояния для завершения потока.
- Ключевое слово volatile гарантирует видимость изменений между потоками. Гарантирует когеррентность кешей ядер каждого потока.
- Thread.interrupt() сигнализирует потоку о необходимости завершиться. Нужно:
  - обрабатывать InterruptedException и завершать работуэ
  - Периодически проверять статус Thread.currentThread().isInterrupted()
- Важные особенности ^|^
- ReentrantLock – альтернатива syncronised. Возможность прирывистой блокировки на отдельном участке. Более гибкий вариант.