**Задание 1.**

Реализовать ThreadPool

**public interface** ThreadPool {  
 **void** start(); *// запускает потоки. Потоки бездействуют, до тех пор пока не появится новое задание в очереди (см. execute)* **void** execute(Runnable runnable); *// складывает это задание в очередь. Освободившийся поток должен выполнить это задание. Каждое задание должны быть выполнено ровно 1 раз*}

Сделать 2 реализации ThreadPool

1) FixedThreadPool - Количество потоков задается в конструкторе и не меняется.

2) ScalableThreadPool в конструкторе задается минимальное и максимальное(int min, int max) число потоков,

количество запущенных потоков может быть увеличено от минимального к максимальному, если при добавлении нового задания в очередь нет свободного потока для исполнения этого задания. При отсутствии задания в очереди, количество потоков опять должно быть уменьшено до значения min

**Задание 2.**

Ваша задача реализовать класс Task имеющий один метод get():

**public class** Task<T> {  
 …  
  
 **public** Task(Callable<? **extends** T> callable) {  
 *//...* }  
  
 **public** T get() {  
 ….. *//* ***todo implement me*** }  
}

Данный класс в конструкторе принимает экземпляр java.util.concurrent.Callable. Callable похож на Runnuble, но результатом его работы является объект (а не void).

Ваша задача реализовать метод get() который возвращает результат работы Callable. Выполнение callable должен начинать тот поток, который первый вызвал метод get(). Если несколько потоков одновременно вызывают этот метод, то выполнение должно начаться только в одном потоке, а остальные должны ожидать конца выполнения (не нагружая процессор).

Если при вызове get() результат уже просчитан, то он должен вернуться сразу, (даже без задержек на вход в синхронизированную область).

Если при просчете результата произошел Exception, то всем потокам при вызове get(), надо кидать этот Exception, обернутый в ваш RuntimeException (подходящее название своему ексепшену придумайте сами)

**Задание 3.**

Ваша задача реализовать интерфейс ExecutionManager

**public interface** ExecutionManager {  
 Context execute(Runnable callback, Runnable... tasks);  
}

Метод execute принимает массив тасков, это задания которые ExecutionManager должен выполнять параллельно (в вашей реализации пусть будет в своем пуле потоков). После завершения всех тасков должен выполниться callback (ровно 1 раз).

Метод execute – это неблокирующий метод, который сразу возвращает объект Context. Context это интерфейс следующего вида:

**public interface** Context {  
 **int** getCompletedTaskCount();  
  
 **int** getFailedTaskCount();  
  
 **int** getInterruptedTaskCount();  
  
 **void** interrupt();  
  
 **boolean** isFinished();  
}

Метод getCompletedTaskCount() возвращает количество тасков, которые на текущий момент успешно выполнились.

Метод getFailedTaskCount() возвращает количество тасков, при выполнении которых произошел Exception.

Метод interrupt() отменяет выполнения тасков, которые еще не начали выполняться.

Метод getInterruptedTaskCount() возвращает количество тасков, которые не были выполены из-за отмены (вызовом предыдущего метода).

Метод isFinished() вернет true, если все таски были выполнены или отменены, false в противном случае.