package com.javarush.test.level16.lesson03.task01;  
  
/\* My first thread  
Создать public static class TestThread - нить с помощью интерфейса Runnable.  
TestThread должен выводить в консоль "My first thread".  
\*/  
public class Solution{  
 public static void main(String[] args) {  
 TestThread task = new TestThread();  
 new Thread(task).start();  
 }  
   
}

package com.javarush.test.level16.lesson03.task02;  
  
/\* My second thread  
1. Создать public static класс TestThread унаследовавшись от класса Thread.  
2. Создать статик блок внутри TestThread, который выводит в консоль "it's static block inside TestThread".  
3. Метод run должен выводить в консоль "it's run method".  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 TestThread thread = new TestThread();  
 thread.start();  
 }  
   
}

package com.javarush.test.level16.lesson03.task03;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\* Список и нити  
В методе main добавить в статический объект list 5 нитей SpecialThread - различных объектов.  
\*/  
  
public class Solution {  
 public static volatile List<Thread> *list* = new ArrayList<Thread>(5);  
  
 public static void main(String[] args) {  
 //Add your code here - добавьте свой код тут  
 }  
  
 public static class SpecialThread implements Runnable {  
 public void run() {  
 System.*out*.println("it's run method inside SpecialThread");  
 }  
 }  
}

package com.javarush.test.level16.lesson03.task04;  
  
/\* Вывод стек-трейса  
1. Создать таск - класс public static SpecialThread - на основании интерфейса Runnable  
2. SpecialThread должен выводить в консоль свой стек-трейс.  
Подсказка: main thread уже выводит в консоль свой стек-трейс.  
\*/  
  
public class Solution {  
 public static void main(String[] args) throws InterruptedException {  
 Thread thread = new Thread(new SpecialThread());  
 thread.start();  
  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
  
 for (StackTraceElement element : Thread.*currentThread*().getStackTrace()) {  
 System.*out*.println(element);  
 }  
 }  
}

package com.javarush.test.level16.lesson03.task05;  
  
import java.util.Date;  
  
/\* Поговорим о музыке?  
1. Измените класс Violin так, чтоб он стал таском для нити. Используйте интерфейс MusicalInstrument  
2. Реализуй необходимый метод в нити Violin. Реализация должна быть следующей:  
2.1. Считай время начала игры - метод startPlaying().  
2.2. Подожди 1 секунду - метод sleepNSeconds(int n), где n - количество секунд.  
2.3. Считай время окончания игры - метод stopPlaying().  
2.4. Выведи на консоль продолжительность игры в миллисекундах. Пример "Playing 1002 ms".  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 Thread violin = new Thread(new Violin("Player"));  
 violin.start();  
 }  
  
 public static class Violin implements Runnable{  
 private String owner;  
  
 public Violin(String owner) {  
 this.owner = owner;  
 }  
  
 public Date startPlaying() {  
 System.*out*.println(this.owner + " starts playing");  
 return new Date();  
 }  
  
 public Date stopPlaying() {  
 System.*out*.println(this.owner + " stops playing");  
 return new Date();  
 }  
  
 @Override  
 public void run()  
 {  
   
   
 }  
 }  
  
 public static int *delay* = 1000;  
  
 public static void sleepNSeconds(int n) {  
 try {  
 Thread.*sleep*(n \* *delay*);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public static interface MusicalInstrument extends Runnable {  
 Date startPlaying();  
  
 Date stopPlaying();  
 }  
}

package com.javarush.test.level16.lesson05.task01;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\* join  
Подумайте, в каком месте и для какого объекта нужно вызвать метод join, чтобы результат выводился по-порядку  
сначала для firstThread, а потом для secondThread.  
Вызовите метод join в нужном месте.  
Пример вывода:  
firstThread : String 1  
firstThread : String 2  
...  
firstThread : String 19  
firstThread : String 20  
secondThread : String 1  
...  
secondThread : String 20  
\*/  
  
public class Solution {  
 public static void main(String[] args) throws InterruptedException {  
 PrintListThread firstThread = new PrintListThread("firstThread");  
 PrintListThread secondThread = new PrintListThread("secondThread");  
 firstThread.start();  
   
 secondThread.start();  
 }  
  
 public static class PrintListThread extends Thread {  
 public PrintListThread(String name) {  
 super(name);  
 }  
  
 public void run() {  
 *printList*(*getList*(20), getName());  
 }  
 }  
  
 public static void printList(List<String> list, String threadName) {  
 for (String item : list) {  
 System.*out*.println(String.*format*("%s : %s", threadName, item));  
 }  
 }  
  
 public static List<String> getList(int n) {  
 List<String> result = new ArrayList<String>();  
 if (n < 1) return result;  
  
 for (int i = 0; i < n; i++) {  
 result.add(String.*format*("String %d", (i + 1)));  
 }  
 return result;  
 }  
}

package com.javarush.test.level16.lesson05.task02;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\* Horse Racing  
Разберись, что делает программа.  
Реализуй метод calculateHorsesFinished. Он должен:  
1. Посчитать количество финишировавших лошадей и возвратить его. Используй метод isFinished().  
2. Если лошадь еще не пришла к финишу (!isFinished()), то:  
2.1. Вывести в консоль "Waiting for " + horse.getName().  
2.2. Подождать, пока она завершит гонку. Подумай, какой метод нужно использовать для этого.  
\*/  
  
public class Solution {  
 public static int *countHorses* = 10;  
  
 public static void main(String[] args) throws InterruptedException {  
 List<Horse> horses = *prepareHorsesAndStart*();  
 while (*calculateHorsesFinished*(horses) != *countHorses*) {  
 }  
 }  
  
 public static int calculateHorsesFinished(List<Horse> horses) throws InterruptedException {  
 int countFinished = 0;  
 //add your implementation here - добавь свою реалзацию тут  
 return countFinished;  
 }  
  
 public static class Horse extends Thread {  
  
 public Horse(String name) {  
 super(name);  
 }  
  
 private boolean isFinished;  
  
 public boolean isFinished() {  
 return isFinished;  
 }  
  
 public void run() {  
 String s = "";  
 for (int i = 0; i < 1001; i++) { //delay  
 s += new String("" + i);  
 if (i == 1000) {  
 s = " has finished the race!";  
 System.*out*.println(getName() + s);  
 isFinished = true;  
 }  
 }  
 }  
 }  
  
 public static List<Horse> prepareHorsesAndStart() {  
 List<Horse> horses = new ArrayList<Horse>(*countHorses*);  
 String number;  
 for (int i = 1; i < *countHorses* + 1; i++) {  
 number = i < 10 ? ("0" + i) : "" + i;  
 horses.add(new Horse("Horse\_" + number));  
 }  
  
 for (int i = 0; i < *countHorses*; i++) {  
 horses.get(i).start();  
 }  
 return horses;  
 }  
}

package com.javarush.test.level16.lesson05.task03;  
  
/\* Продвижение на политических дебатах  
1. Разберитесь, что делает программа.  
2. Нужно сделать так, чтобы Иванов сказал больше всего речей на политических дебатах.  
3. Подумай, какой метод можно вызвать у объекта ivanov, чтобы Иванов разговаривал, пока не завершится всё свободное время.  
\*/  
  
public class Solution {  
 public static int totalCountSpeeches = 200;  
 public static int soundsInOneSpeech = 1000000;  
  
 public static void main(String[] args) throws InterruptedException {  
 Politic ivanov = new Politic("Иванов");  
 Politic petrov = new Politic("Петров");  
 Politic sidorov = new Politic("Сидоров");  
  
 while (ivanov.getCountSpeaches() + petrov.getCountSpeaches() + sidorov.getCountSpeaches() < totalCountSpeeches) {  
 }  
  
 System.out.println(ivanov);  
 System.out.println(petrov);  
 System.out.println(sidorov);  
 }  
  
 public static class Politic extends Thread {  
 private int countSounds;  
  
 public Politic(String name) {  
 super(name);  
 start();  
 }  
  
 public void run() {  
 while (countSounds < totalCountSpeeches \* soundsInOneSpeech) {  
 countSounds++;  
 }  
 }  
  
 public int getCountSpeaches() {  
 return countSounds / soundsInOneSpeech;  
 }  
  
 @Override  
 public String toString() {  
 return String.format("%s сказал речь %d раз", getName(), getCountSpeaches());  
 }  
 }  
}

package com.javarush.test.level16.lesson05.task04;  
  
/\* Справедливость  
1. Разберитесь, что делает программа.  
2. Нужно сделать так, чтобы все мыши ели одновременно.  
3. Подумай, какой метод позволяет альфа-самцу мыши есть первым, и почему остальные мыши ждут.  
4. Удали вызов этого метода.  
\*/  
  
public class Solution {  
 public static void main(String[] args) throws InterruptedException {  
 Mouse alpha = new Mouse("#1");  
 alpha.join();  
 Mouse mouse1 = new Mouse("#2");  
 Mouse mouse2 = new Mouse("#3");  
 }  
  
 public static class Mouse extends Thread {  
 public Mouse(String name) {  
 super(name);  
 start();  
 }  
  
 public void run() {  
 System.*out*.println(getName() + " starts eating");  
 *eating*();  
 System.*out*.println(getName() + " finished");  
 }  
 }  
 private static void eating() {  
 try {  
 Thread.*sleep*(2000);  
 } catch (InterruptedException e) {  
 }  
 }  
}

package com.javarush.test.level16.lesson05.task05;  
  
/\* Расставь вызовы методов join()  
1. Разберитесь, что делает программа.  
2. Расставь вызовы методов join() так, чтобы для каждой кошки выполнялось следующее:  
2.1. Сначала кошка рожает котят.  
2.2. Потом все котята вылазят из корзинки в произвольном порядке.  
2.3. В конце кошка собирает их назад в корзинку.  
2.4. Все события для одной кошки могут быть перемешаны с событями для другой кошки.  
2.5. Добавить сон котят (200 мс) в investigateWorld  
\*/  
  
public class Solution {  
 public static void main(String[] args) throws InterruptedException {  
 Cat cat1 = new Cat("Мурка");  
 Cat cat2 = new Cat("Пушинка");  
 }  
  
 public static class Cat extends Thread {  
 protected Kitten kitten1;  
 protected Kitten kitten2;  
  
 public Cat(String name) {  
 super(name);  
 kitten1 = new Kitten("Котенок 1, мама - " + getName());  
 kitten2 = new Kitten("Котенок 2, мама - " + getName());  
 start();  
 }  
  
 public void run() {  
 System.*out*.println(getName() + " родила 2 котенка");  
 try {  
 initAllKitten();  
 } catch (InterruptedException e) {  
 }  
 System.*out*.println(getName() + ": Все котята в корзинке. " + getName() + " собрала их назад");  
 }  
  
 private void initAllKitten() throws InterruptedException {  
 kitten1.start();  
 kitten2.start();  
 }  
 }  
  
 public static class Kitten extends Thread {  
 public Kitten(String name) {  
 super(name);  
 }  
  
 public void run() {  
 System.*out*.println(getName() + ", вылез из корзинки");  
 *investigateWorld*();  
 }  
 }  
  
 private static void investigateWorld() {  
  
 }  
}

package com.javarush.test.level16.lesson07.task01;  
  
/\* Часы  
1. Разберись, что делает программа.  
2. Реализуйте логику метода printTikTak:  
2.1. Через первые полсекунды должна выводиться в консоль фраза: Tik.  
2.2. Через вторые полсекунды должна выводиться в консоль фраза: Tak.  
\*/  
  
public class Solution {  
 public static volatile boolean *isStopped* = false;  
  
 public static void main(String[] args) throws InterruptedException {  
 Clock clock = new Clock();  
 Thread.*sleep*(2000);  
 *isStopped* = true;  
 System.*out*.println("Clock has to be stopped");  
 Thread.*sleep*(1000);  
 System.*out*.println("Double-check");  
 }  
  
 public static class Clock extends Thread {  
 public Clock() {  
 setPriority(*MAX\_PRIORITY*);  
 start();  
 }  
  
 public void run() {  
 try {  
 while (!*isStopped*) {  
 printTikTak();  
 }  
 } catch (InterruptedException e) {  
 }  
 }  
  
 private void printTikTak() throws InterruptedException {  
 //add your code here - добавь код тут  
 }  
 }  
}

package com.javarush.test.level16.lesson07.task02;  
  
/\* Stopwatch (Секундомер)  
1. Разберись, что делает программа.  
2. Реализуй логику метода doSeveralSteps так, чтобы учитывалась скорость бегуна.  
2.1. Метод getSpeed() в классе Runner показывает, сколько шагов в секунду делает бегун.  
Нужно, чтобы бегун действительно делал заданное количество шагов в секунду.  
Если Иванов делает 4 шага в секунду, то за 2 секунды он сделает 8 шагов.  
Если Петров делает 2 шага в секунду, то за 2 секунды он сделает 4 шага.  
2.2. Метод sleep в классе Thread принимает параметр типа long.  
\*/  
  
public class Solution {  
 public static volatile boolean *isStopped* = false;  
  
 public static void main(String[] args) throws InterruptedException {  
 Runner ivanov = new Runner("Ivanov", 4);  
 Runner petrov = new Runner("Petrov", 2);  
 //на старт!  
 //внимание!  
 //марш!  
 ivanov.start();  
 petrov.start();  
 Thread.*sleep*(2000);  
 *isStopped* = true;  
 Thread.*sleep*(1000);  
 }  
  
 public static class Stopwatch extends Thread {  
 private Runner owner;  
 private int stepNumber;  
  
 public Stopwatch(Runner runner) {  
 this.owner = runner;  
 }  
  
 public void run() {  
 try {  
 while (!*isStopped*) {  
 doSeveralSteps();  
 }  
 } catch (InterruptedException e) {  
 }  
 }  
  
 private void doSeveralSteps() throws InterruptedException {  
 stepNumber++;  
 //add your code here - добавь код тут  
 System.*out*.println(owner.getName() + " делает шаг №" + stepNumber + "!");  
 }  
 }  
  
 public static class Runner {  
 private String name;  
 private double speed;  
 Stopwatch stopwatch;  
  
 public Runner(String name, double speed) {  
 this.name = name;  
 this.speed = speed;  
 this.stopwatch = new Stopwatch(this);  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public double getSpeed() {  
 return speed;  
 }  
  
 public void start() {  
 stopwatch.start();  
 }  
 }  
}

package com.javarush.test.level16.lesson07.task03;  
  
/\* Big Ben clock  
1. Разберись, что делает программа.  
2. Реализуй логику метода printTime так, чтобы каждую секунду выдавалось время начиная с установленного в конструкторе  
Пример:  
В г. Лондон сейчас 23:59:58!  
В г. Лондон сейчас 23:59:59!  
В г. Лондон сейчас полночь!  
В г. Лондон сейчас 0:0:1!  
\*/  
  
public class Solution {  
 public static volatile boolean *isStopped* = false;  
  
 public static void main(String[] args) throws InterruptedException {  
 Clock clock = new Clock("Лондон", 23, 59, 57);  
 Thread.*sleep*(4000);  
 *isStopped* = true;  
 Thread.*sleep*(1000);  
 }  
  
 public static class Clock extends Thread {  
 private String cityName;  
 private int hours;  
 private int minutes;  
 private int seconds;  
  
 public Clock(String cityName, int hours, int minutes, int seconds) {  
 this.cityName = cityName;  
 this.hours = hours;  
 this.minutes = minutes;  
 this.seconds = seconds;  
 start();  
 }  
  
 public void run() {  
 try {  
 while (!*isStopped*) {  
 printTime();  
 }  
 } catch (InterruptedException e) {  
 }  
 }  
  
 private void printTime() throws InterruptedException {  
 //add your code here - добавь код тут  
  
 if (hours == 0 && minutes == 0 && seconds == 0) {  
 System.*out*.println(String.*format*("В г. %s сейчас полночь!", cityName));  
 } else {  
 System.*out*.println(String.*format*("В г. %s сейчас %d:%d:%d!", cityName, hours, minutes, seconds));  
 }  
 }  
 }  
}

package com.javarush.test.level16.lesson07.task04;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\* Обратный отсчет  
1. Разберись, что делает программа.  
2. Реализуй логику метода printCountdown так, чтобы каждые полсекунды выводился объект из  
переменной list в обратном порядке - от переданного индекса до нуля.  
Пример: Передан индекс 3  
Пример вывода в консоль:  
Строка 2  
Строка 1  
Строка 0  
\*/  
  
public class Solution {  
 public static volatile List<String> *list* = new ArrayList<String>(5);  
  
 static {  
 for (int i = 0; i < 5; i++) {  
 *list*.add("Строка " + i);  
 }  
 }  
  
 public static void main(String[] args) throws InterruptedException {  
 Thread t = new Thread(new Countdown(3));  
 t.start();  
 }  
  
 public static class Countdown implements Runnable {  
 private int countFrom;  
  
 public Countdown(int countFrom) {  
 this.countFrom = countFrom;  
 }  
  
 public void run() {  
 try {  
 while (countFrom > 0) {  
 printCountdown();  
 }  
 } catch (InterruptedException e) {  
 }  
 }  
  
 public void printCountdown() throws InterruptedException {  
 //add your code here - добавь код тут  
 }  
 }  
}

package com.javarush.test.level16.lesson07.task05;  
  
/\* Аэропорт  
1. Разберись, что делает программа.  
2. Исправь метод takingOff(взлет) - сейчас он работает оооочень долго. Взлет должен занимать 100 миллисекунд.  
3. Реализуй метод waiting по аналогии с методом takingOff. Время ожидания не должно превышать время взлета.  
\*/  
public class Solution {  
 public static volatile Runway *RUNWAY* = new Runway(); //1 взлетная полоса  
  
 public static void main(String[] args) throws InterruptedException {  
 Plane plane1 = new Plane("Самолет #1");  
 Plane plane2 = new Plane("Самолет #2");  
 Plane plane3 = new Plane("Самолет #3");  
 }  
  
 public static class Plane extends Thread {  
 public Plane(String name) {  
 super(name);  
 start();  
 }  
  
 public void run() {  
 boolean isAlreadyTakenOff = false;  
 while (!isAlreadyTakenOff) {  
 if (*RUNWAY*.getTakingOffPlane() == null) { //если взлетная полоса свободна  
 *RUNWAY*.setTakingOffPlane(this);  
 System.*out*.println(getName() + " взлетает");  
 *takingOff*();//взлетает  
 System.*out*.println(getName() + " уже в небе");  
 isAlreadyTakenOff = true;  
 *RUNWAY*.setTakingOffPlane(null);  
 } else if (!this.equals(*RUNWAY*.getTakingOffPlane())) { //если взлетная полоса занята самолетом  
 System.*out*.println(getName() + " ожидает");  
 *waiting*(); //ожидает  
 }  
 }  
 }  
 }  
  
 private static void waiting() {  
 //add your code here - добавь код тут  
 }  
  
 private static void takingOff() {  
 //fix this method - исправь этот метод  
 try {  
 Thread.*sleep*(10000);  
 } catch (InterruptedException e) {  
 }  
 }  
  
 public static class Runway { //взлетная полоса  
 private Thread t;  
  
 public Thread getTakingOffPlane() {  
 return t;  
 }  
  
 public void setTakingOffPlane(Thread t) {  
 synchronized (this) {  
 this.t = t;  
 }  
 }  
 }  
}

package com.javarush.test.level16.lesson10.task01;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
  
/\* Считаем секунды  
1. Напиши реализацию метода run в нити Stopwatch (секундомер).  
2. Stopwatch должен посчитать количество секунд, которое прошло от создания нити до ввода строки.  
3. Выведи количество секунд в консоль.  
\*/  
  
public class Solution {  
 public static void main(String[] args) throws IOException {  
 InputStreamReader in = new InputStreamReader(System.*in*);  
 BufferedReader reader = new BufferedReader(in);  
 //create and run thread  
 Stopwatch stopwatch = new Stopwatch();  
 stopwatch.start();  
 //read a string  
 reader.readLine();  
 stopwatch.interrupt();  
 //close streams  
 reader.close();  
 in.close();  
 }  
  
 public static class Stopwatch extends Thread {  
 private int seconds;  
  
 public void run() {  
 try {  
 //add your code here - добавьте код тут  
 } catch (InterruptedException e) {  
 System.*out*.println(seconds);  
 }  
 }  
 }  
}

package com.javarush.test.level16.lesson10.task02;  
  
/\* Отсчет на гонках  
1. Разберись, что делает программа.  
2. Реализуй логику метода run так, чтобы каждую секунду через пробел  
выдавался отсчет начиная с countSeconds до 1, а потом слово [Марш!] (см примеры).  
3. Если нить работает 3.5 секунды или более, прерви ее методом interrupt и внутри нити выведи в консоль слово [Прервано!].  
Пример для countSeconds=4 : [4 3 2 1 Прервано!]  
4. Если нить работает менее 3.5 секунд, она должна завершиться сама.  
Пример для countSeconds=3 : [3 2 1 Марш!]  
PS: метод sleep выбрасывает InterruptedException.  
\*/  
  
public class Solution {  
 public static volatile int *countSeconds* = 3;  
  
 public static void main(String[] args) throws InterruptedException {  
 RacingClock clock = new RacingClock();  
 //add your code here - добавь код тут  
 }  
  
 public static class RacingClock extends Thread {  
 public RacingClock() {  
 start();  
 }  
  
 public void run() {  
 //add your code here - добавь код тут  
 }  
 }  
}

package com.javarush.test.level16.lesson10.task03;  
  
/\* Снова interrupt  
Создай нить TestThread.  
В методе main создай экземпляр нити, запусти, а потом прерви ее используя метод interrupt().  
\*/  
  
public class Solution {  
 public static void main(String[] args) throws InterruptedException {  
 //Add your code here - добавь код тут  
 }  
  
 //Add your code below - добавь код ниже  
 public static class TestThread {  
 }  
}

package com.javarush.test.level16.lesson10.task04;  
  
/\* А без interrupt слабо?  
Разберись, как работает программа.  
Сделай так, чтобы в методе ourInterruptMethod можно было сделать так, чтобы нить TestThread завершилась сама.  
Нельзя использовать метод interrupt.  
\*/  
  
public class Solution {  
 public static void main(String[] args) throws InterruptedException {  
 Thread t = new Thread(new TestThread());  
 t.start();  
 Thread.*sleep*(3000);  
 *ourInterruptMethod*();  
 }  
  
 public static void ourInterruptMethod() {  
  
 }  
  
 public static class TestThread implements Runnable {  
 public void run() {  
 while(true) {  
 try {  
 System.*out*.println("he-he");  
 Thread.*sleep*(500);  
 } catch (InterruptedException e) {  
 }  
 }  
 }  
 }  
}

package com.javarush.test.level16.lesson10.task05;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\* Один для всех, все - для одного  
1. Разберись, как работает программа.  
1.1. Обрати внимание, что объект Water - один для всех нитей.  
  
2. Реализуй метод ourInterruptMethod, чтобы он прерывал все нити из threads.  
3. В методе run исправь значения переменных:  
3.1. isCurrentThreadInterrupted - должна равняться значению метода isInterrupted у текущей нити.  
3.2. threadName - должна равняться значению метода getName (реализовано в классе Thread) у текущей нити.  
\*/  
  
public class Solution {  
 public static byte *countThreads* = 3;  
 static List<Thread> *threads* = new ArrayList<Thread>(*countThreads*);  
  
 public static void main(String[] args) throws InterruptedException {  
 *initThreadsAndStart*();  
 Thread.*sleep*(3000);  
 *ourInterruptMethod*();  
 }  
  
 public static void ourInterruptMethod() {  
 //add your code here - добавь код тут  
 }  
  
 private static void initThreadsAndStart() {  
 Water water = new Water("water");  
 for (int i = 0; i < *countThreads*; i++) {  
 *threads*.add(new Thread(water, "#" + i));  
 }  
  
 for (int i = 0; i < *countThreads*; i++) {  
 *threads*.get(i).start();  
 }  
 }  
  
 public static class Water implements Runnable {  
 private String commonResource;  
  
 public Water(String commonResource) {  
 this.commonResource = commonResource;  
 }  
  
 public void run() {  
 //fix 2 variables - исправь 2 переменных  
 boolean isCurrentThreadInterrupted = false;  
 String threadName = "";  
  
 try {  
 while (!isCurrentThreadInterrupted) {  
  
 System.*out*.println("Объект " + commonResource + ", нить " + threadName);  
 Thread.*sleep*(1000);  
 }  
 } catch (InterruptedException e) {  
 }  
 }  
 }  
}

package com.javarush.test.level16.lesson13.home01;  
  
/\* Thread.currentThread - всегда возвращает текущую нить  
1. В методе printMsg присвой переменной t текущую нить.  
2. В методе printMsg после всех действий поставь задержку в 1 миллисекунду.  
\*/  
  
public class Solution {  
 static int *count* = 5;  
  
 public static void main(String[] args) {  
 NameOfDefferentThreads tt = new NameOfDefferentThreads();  
 tt.start();  
 for (int i = 0; i < *count*; i++) {  
 tt.printMsg();  
 }  
 }  
  
 public static class NameOfDefferentThreads extends Thread {  
 public void run() {  
 for (int i = 0; i < *count*; i++) {  
 printMsg();  
 }  
 }  
 public void printMsg() {  
 Thread t = null;//присвой переменной t текущую нить  
 String name = t.getName();  
 System.*out*.println("name=" + name);  
 //add sleep here - добавь sleep тут  
 }  
 }  
}

package com.javarush.test.level16.lesson13.home02;  
  
/\* Последовательные выполнения нитей  
1. В методе run после всех действий поставь задержку в 10 миллисекунд. Выведи "Нить прервана", если нить будет прервана.  
2. Сделай так, чтобы все нити выполнялись последовательно: сначала для нити №1 отсчет с COUNT до 1, потом для нити №2 с COUNT до 1 и т.д.  
Пример:  
#1: 4  
#1: 3  
...  
#1: 1  
#2: 4  
...  
\*/  
  
public class Solution {  
 public volatile static int *COUNT* = 4;  
  
 public static void main(String[] args) throws InterruptedException {  
 for (int i = 0; i < *COUNT*; i++) {  
 new SleepingThread();  
 //напишите тут ваш код  
 }  
 }  
  
 public static class SleepingThread extends Thread {  
 private volatile int countDownIndex = *COUNT*;  
 private static volatile int *threadCount* = 0;  
  
 public SleepingThread() {  
 super(String.*valueOf*(++*threadCount*));  
 start();  
 }  
  
 public void run() {  
 while (true) {  
 System.*out*.println(this);  
 if (--countDownIndex == 0) return;  
 //add sleep here - добавь sleep тут  
 }  
 }  
  
 public String toString() {  
 return "#" + getName() + ": " + countDownIndex;  
 }  
 }  
}

package com.javarush.test.level16.lesson13.home03;  
  
/\* Рекурсивное создание нитей  
1. Измени класс GenerateThread так, чтобы он стал нитью.  
2. Создай конструктор GenerateThread, который должен:  
2.1. Вызвать конструктор суперкласса с параметром String - номером созданной нити. Используй countCreatedThreads.  
2.2. Запустить текущую нить.  
2.3. Номер первой нити должен начинается с 1.  
3. Переопредели метод toString, для этого внутри GenerateThread нажми Alt+Insert -> Override Methods. Начни печатать toString.  
3.1. Метод toString должен возвращать № текущей нити и слово " created". Используй getName(). Пример: [8 created]  
4. Пока количество созданных нитей меньше Solution.count метод run должен:  
4.1. создать новую нить типа GenerateThread.  
4.2. Вывести в консоль созданную в пункте 4.1. нить.  
5. В итоге должно быть выведено в консоль 15 строк.  
\*/  
  
public class Solution {  
 static int *count* = 15;  
 static volatile int *countCreatedThreads*;  
  
 public static void main(String[] args) {  
 System.*out*.println(new GenerateThread());  
 }  
  
 public static class GenerateThread {  
 }  
}

package com.javarush.test.level16.lesson13.home04;  
  
/\* Последовательные выполнения нитей  
1. Разберись, что делает программа.  
2. Сделай так, чтоб программа сначала выводила результат нити, а когда нить завершится - продолжила метод main.  
3. Пример выходных данных:  
inside MyThread 0  
inside MyThread 1  
...  
inside MyThread 9  
inside main 0  
inside main 1  
...  
inside main 9  
\*/  
  
public class Solution {  
 public static MyThread *t* = new MyThread();  
 static String *message* = "inside main ";  
  
 public static void main(String a[]) throws Exception {  
 *t*.start();  
 for (int i = 0; i < 10; i++) {  
 System.*out*.println(*message* + i);  
 *sleep*();  
 }  
 }  
  
 static class MyThread extends Thread {  
 String message = "inside MyThread ";  
  
 public void run() {  
 for (int i = 0; i < 10; i++) {  
 System.*out*.println(message + i);  
 Solution.*sleep*();  
 }  
 }  
 }  
  
 public static void sleep() {  
 try {  
 Thread.*sleep*(10);  
 } catch (InterruptedException e) {  
 }  
 }  
}

package com.javarush.test.level16.lesson13.home05;  
  
/\* Взаимная блокировка  
1. Разберись, как работает программа.  
2. Не меняя классы T1 и T2 сделай так, чтобы они завершились, не обязательно успешно.  
3. метод sleep не использовать.  
\*/  
  
public class Solution {  
 static Thread *t1* = new T1();  
 static Thread *t2* = new T2();  
  
 public static void main(String[] args) throws InterruptedException {  
 *t1*.start();  
 *t2*.start();  
 }  
  
 public static class T1 extends Thread {  
 @Override  
 public void run() {  
 try {  
 *t2*.join();  
 System.*out*.println("T1 finished");  
 } catch (InterruptedException e) {  
 System.*out*.println("T1 was interrupted");  
 }  
 }  
 }  
  
 public static class T2 extends Thread {  
 @Override  
 public void run() {  
 try {  
 *t1*.join();  
 System.*out*.println("T2 finished");  
 } catch (InterruptedException e) {  
 System.*out*.println("T2 was interrupted");  
 }  
 }  
 }  
}

package com.javarush.test.level16.lesson13.home06;  
  
/\* Создание по образцу  
Разберись, как работает программа.  
По образу и подобию CountDownRunnable создай нить CountUpRunnable, которая выводит значения в нормальном порядке - от 1 до number  
\*/  
  
public class Solution {  
 public static int *number* = 5;  
  
 public static void main(String[] args) {  
 new CountDownRunnable("было");  
 new CountUpRunnable("стало");  
 }  
  
 //Add your code below - добавьте код ниже  
 public static class CountUpRunnable {  
 public CountUpRunnable(String name) {  
 }  
 }  
  
 public static class CountDownRunnable implements Runnable {  
 private int countIndexDown = Solution.*number*;  
 private Thread t;  
  
 public CountDownRunnable(String name) {  
 t = new Thread(this, name);  
 t.start();  
 }  
  
 public void run() {  
 try {  
 while (true) {  
 System.*out*.println(toString());  
 countIndexDown -= 1;  
 if (countIndexDown == 0) return;  
 Thread.*sleep*(500);  
 }  
 } catch (InterruptedException e) {  
 }  
 }  
  
 public String toString() {  
 return t.getName() + ": " + countIndexDown;  
 }  
 }  
}

package com.javarush.test.level16.lesson13.home07;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\* Поиграем?  
Три человека играют в игру. Каждый игрок(Gamer) характеризуется двумя параметрами: фамилией(name) и количеством действий в секунду (rating).  
Нужно вывести в консоль ход игры и определить победителя и проигравших.  
Итак...  
1. Разберись, что делает программа.  
1.1. List<String> steps хранит последовательность действий, которое каждый игрок выполняет от 0 до последнего.  
1.2. isWinnerFound показывает, найден победитель или нет.  
1.3. метод sleep выбрасывает InterruptedException и принимает параметр типа long.  
1.4. Игроки играют независимо друг от друга.  
  
2. Реализуйте логику метода run так, чтобы для каждого игрока:  
2.1. За 1 секунду через равные интервалы времени выводились в консоль действия, описанные в steps. Количество выведенных действий должно равняться rating.  
2.2. Любой текст должен начинаться с фамилии игрока (метод getName()), потом следовать двоеточие, а затем сам текст. Пример: [Ivanov:Начало игры].  
2.3. Когда игрок выполнит все действия из steps, то он считается победителем. Выведите [getName() + ":победитель!"].  
2.4. Когда найден победитель, то игра останавливается, и остальные игроки считаются побежденными. Выведите для них [getName() + ":проиграл"].

P.s победитель может быть и не один   
\*/  
  
public class Solution {  
 public static void main(String[] args) throws InterruptedException {  
 OnlineGame onlineGame = new OnlineGame();  
 onlineGame.start();  
 }  
  
 public static class OnlineGame extends Thread {  
 public static volatile boolean *isWinnerFound* = false;  
  
 public static List<String> *steps* = new ArrayList<String>();  
  
 static {  
 *steps*.add("Начало игры");  
 *steps*.add("Сбор ресурсов");  
 *steps*.add("Рост экономики");  
 *steps*.add("Убийство врагов");  
 }  
  
 protected Gamer gamer1 = new Gamer("Ivanov", 3);  
 protected Gamer gamer2 = new Gamer("Petrov", 1);  
 protected Gamer gamer3 = new Gamer("Sidorov", 5);  
  
 public void run() {  
 gamer1.start();  
 gamer2.start();  
 gamer3.start();  
  
 while (!*isWinnerFound*) {  
 }  
 gamer1.interrupt();  
 gamer2.interrupt();  
 gamer3.interrupt();  
 }  
 }  
  
 public static class Gamer extends Thread {  
 private int rating;  
  
 public Gamer(String name, int rating) {  
 super(name);  
 this.rating = rating;  
 }  
  
 @Override  
 public void run() {  
 //Add your code here - добавь код тут  
 }  
 }  
}

package com.javarush.test.level16.lesson13.home08;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.util.ArrayList;  
import java.util.List;  
  
/\* Кто первый встал - того и тапки  
1. Разберись, что делает программа.  
1.1. Каждая нить должна читать с консоли слова. Используйте готовый static BufferedReader reader.  
1.2. Используй static byte countReadStrings, чтобы посчитать, сколько слов уже считано с консоли всеми нитями.  
  
2. Реализуйте логику метода run:  
2.1. Пока нить не прервана (!isInterrupted) читайте с консоли слова и добавляйте их в поле List<String> result.  
2.2. Используй countReadStrings для подсчета уже считанных с консоли слов.  
\*/  
  
public class Solution {  
 public static volatile byte *countReadStrings*;  
 public static volatile BufferedReader *reader* = new BufferedReader(new InputStreamReader(System.*in*));  
  
 public static void main(String[] args) throws IOException {  
 //read count of strings  
 int count = Integer.*parseInt*(*reader*.readLine());  
  
 //init threads  
 ReaderThread consolReader1 = new ReaderThread();  
 ReaderThread consolReader2 = new ReaderThread();  
 ReaderThread consolReader3 = new ReaderThread();  
  
 while (count > *countReadStrings*) {  
 }  
  
 consolReader1.interrupt();  
 consolReader2.interrupt();  
 consolReader3.interrupt();  
 System.*out*.println("#1:" + consolReader1);  
 System.*out*.println("#2:" + consolReader2);  
 System.*out*.println("#3:" + consolReader3);  
  
 *reader*.close();  
 }  
  
 public static class ReaderThread extends Thread {  
 private List<String> result = new ArrayList<String>();  
  
 public ReaderThread() {  
 start();  
 }  
  
 public void run() {  
 //add your code here - добавьте код тут  
 }  
  
 @Override  
 public String toString() {  
 String s = result.toString();  
 return s.substring(1, s.length()-1);  
 }  
 }  
}

package com.javarush.test.level16.lesson13.home09;  
  
/\* Только по-очереди!  
1. В классе Solution создать нить public static Read3Strings унаследовавшись от Thread.  
2. В методе run реализовать чтение с консоли трех строк.  
3. Три подряд введенных строки должны считываться в одной нити и объединяться в одну строку через пробел.  
4. В методе main вывести результат для каждой нити.  
5. Используйте join  
  
Пример:  
Входные данные  
a  
b  
c  
d  
e  
f  
Выходные данные:  
a b c  
d e f  
\*/  
  
public class Solution {  
 public static volatile BufferedReader *reader* = new BufferedReader(new InputStreamReader(System.*in*));  
  
 public static void main(String[] args) throws InterruptedException {  
 Read3Strings t1 = new Read3Strings();  
 Read3Strings t2 = new Read3Strings();  
 t1.start();  
 t2.start();  
 System.*out*.println(/\*print t1 result here\*/);  
 System.*out*.println(/\*print t2 result here\*/);  
 }  
}

package com.javarush.test.level16.lesson13.home10;  
  
import java.io.BufferedReader;  
import java.io.FileNotFoundException;  
import java.io.FileReader;  
import java.io.IOException;  
  
/\* Последовательный вывод файлов  
1. Разберись, что делает программа.  
2. В статическом блоке считай 2 имени файла firstFileName и secondFileName.  
3. Внутри класса Solution создай нить public static ReadFileThread, которая реализует  
интерфейс ReadFileInterface (Подумай, что больше подходит - Thread или Runnable).  
3.1. Метод setFileName должен устанавливать имя файла, из которого будет читаться содержимое.  
3.2. Метод getFileContent должен возвращать содержимое файла.  
3.3. В методе run считай содержимое файла, закрой поток. Раздели пробелом строки файла.  
4. Подумай, в каком месте нужно подождать окончания работы нити, чтобы обеспечить последовательный вывод файлов.  
4.1. Для этого добавь вызов соответствующего метода.  
Ожидаемый вывод:  
[все тело первого файла]  
[все тело второго файла]  
\*/  
  
public class Solution {  
 public static String firstFileName;  
 public static String secondFileName;  
  
 public static void main(String[] args) throws InterruptedException {  
 systemOutPrintln(firstFileName);  
 systemOutPrintln(secondFileName);  
 }  
  
 public static void systemOutPrintln(String fileName) throws InterruptedException {  
 ReadFileInterface f = new ReadFileThread();  
 f.setFileName(fileName);  
 f.start();  
 System.out.println(f.getFileContent());  
 }  
  
 public static interface ReadFileInterface {  
  
 void setFileName(String fullFileName);  
  
 String getFileContent();  
  
 void join() throws InterruptedException;  
  
 void start();  
 }  
}

package com.javarush.test.level16.lesson13.bonus01;  
  
import com.javarush.test.level16.lesson13.bonus01.ImageReaderFactory;  
import com.javarush.test.level16.lesson13.bonus01.common.ImageReader;  
import com.javarush.test.level16.lesson13.bonus01.common.ImageTypes;  
  
/\* Factory method pattern  
1. Внимательно посмотри, какие классы у тебя есть.  
2. В отдельных файлах в пакете common создай классы JpgReader, PngReader, BmpReader, которые реализуют интерфейс ImageReader.  
3. В отдельном файле в пакете bonus01 создай класс ImageReaderFactory с одним методом.  
3.1. Подумай, как он должен называться.  
3.2. Подумай, какие модификаторы должны быть у этого метода.  
4. Этот метод должен:  
4.1. для каждого значения из ImageTypes возвращать соответствующий Reader, например, для ImageTypes.JPG - JpgReader;  
4.2. если передан неправильный параметр, то выбросить исключение IllegalArgumentException("Неизвестный тип картинки").  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 ImageReader reader = ImageReaderFactory.getReader(ImageTypes.*JPG*);  
 }  
}

package com.javarush.test.level16.lesson13.bonus01.common;  
  
public enum ImageTypes {  
 *BMP*,  
 *JPG*,  
 *PNG*;  
}

package com.javarush.test.level16.lesson13.bonus01.common;  
  
public interface ImageReader {  
}

package com.javarush.test.level16.lesson13.bonus02;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\* Клубок  
1. Создай 5 различных своих нитей c отличным от Thread типом:  
1.1. нить 1 должна бесконечно выполняться;  
1.2. нить 2 должна выводить "InterruptedException" при возникновении исключения InterruptedException;  
1.3. нить 3 должна каждые полсекунды выводить "Ура";  
1.4. нить 4 должна реализовать интерфейс Message, при вызове метода showWarning нить должна останавливаться;  
1.5. нить 5 должна читать с консоли цифры пока не введено слово "N", а потом вывести в консоль сумму введенных цифр.  
2. В статическом блоке добавь свои нити в List<Thread> threads в перечисленном порядке.  
3. Нити не должны стартовать автоматически.  
Подсказка: Нить 4 можно проверить методом isAlive()  
\*/  
  
public class Solution {  
 public static List<Thread> *threads* = new ArrayList<Thread>(5);  
}

package com.javarush.test.level16.lesson13.bonus02;  
  
public interface Message {  
 void showWarning();  
}

package com.javarush.test.level16.lesson13.bonus03;  
  
/\* Отдебажим все на свете  
Разобраться, что делает програма.  
Почитать про UncaughtExceptionHandler - это важно.  
Еще раз внимательно посмотреть программу.  
Разобраться - продебажить - почему наш OurUncaughtExceptionHandler не срабатывает.  
Исправить ошибку, т.е. все должно работать. :)  
  
Ожидаемый результат в произвольном порядке:  
Нить 1: My exception message  
Нить 2: My exception message  
\*/  
  
public class Solution {  
 public static Thread.UncaughtExceptionHandler *handler* = new OurUncaughtExceptionHandler();  
  
 public static void main(String[] args) {  
 TestedThread commonThread = new TestedThread(*handler*);  
  
 Thread threadA = new Thread(commonThread, "Нить 1");  
 Thread threadB = new Thread(commonThread, "Нить 2");  
  
 threadA.start();  
 threadB.start();  
  
 threadA.interrupt();  
 threadB.interrupt();  
 }  
  
 public static class TestedThread extends Thread {  
 public TestedThread(UncaughtExceptionHandler handler) {  
 setUncaughtExceptionHandler(handler);  
 start();  
 }  
  
 public void run() {  
 try {  
 Thread.*sleep*(3000);  
 } catch (InterruptedException x) {  
 throw new RuntimeException("My exception message");  
 }  
 }  
 }  
  
 public static class OurUncaughtExceptionHandler implements Thread.UncaughtExceptionHandler {  
 @Override  
 public void uncaughtException(Thread t, Throwable e) {  
 System.*out*.println(t.getName() + ": " + e.getMessage());  
 }  
 }  
}