package com.javarush.test.level17.lesson02.task01;  
  
 import java.util.ArrayList;  
 import java.util.List;  
  
/\* Заметки  
1. Класс Note будет использоваться нитями.  
2. Создай public static нить NoteThread (Runnable не является нитью),  
которая в методе run 1000 раз (index = 0-999) сделает следующие действия:  
2.1. используя метод addNote добавит заметку с именем [getName() + "-Note" + index], например, при index=4  
"Thread-0-Note4"  
2.2. используя метод removeNote удалит заметку  
2.3. в качестве первого параметра в removeNote передай имя нити - метод getName()  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
   
 }  
  
 public static class Note {  
  
 public static final List<String> *notes* = new ArrayList<String>();  
  
 public static void addNote(String note) {  
 *notes*.add(0, note);  
 }  
  
 public static void removeNote(String threadName) {  
 String note = *notes*.remove(0);  
 if (note == null) {  
 System.*out*.println("Другая нить удалила нашу заметку");  
 } else if (!note.startsWith(threadName)) {  
 System.*out*.println("Нить [" + threadName + "] удалила чужую заметку [" + note + "]");  
 }  
 }  
  
 }  
   
  
}

package com.javarush.test.level17.lesson02.task02;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
  
/\* Вместе быстрее? Ща проверим :)  
1. Разберись, что и как работает  
2. Создай public static нить SortThread, которая в методе run отсортирует  
статический массив testArray используя метод sort  
\*/  
  
public class Solution {  
 public static int *countThreads* = 10;  
 public static int[] *testArray* = new int[1000];  
   
 static {  
 for (int i = 0; i < Solution.*testArray*.length; i++) {  
 *testArray*[i] = i;  
 }  
 }  
  
 public static void main(String[] args) throws InterruptedException {  
 *initThreads*();  
 }  
  
 public static void initThreads() throws InterruptedException {  
 List<Thread> threads = new ArrayList<Thread>(*countThreads*);  
 for (int i = 0; i < *countThreads*; i++) threads.add(new SortThread());  
 for (Thread thread : threads) thread.start();  
 for (Thread thread : threads) thread.join();  
 }  
  
 public static void sort(int[] array) {  
 for (int i = 0; i < array.length - 1; i++) {  
 for (int j = i + 1; j < array.length; j++) {  
 if (array[i] < array[j]) {  
 int k = array[i];  
 array[i] = array[j];  
 array[j] = k;  
 }  
 }  
 }  
 }  
}

package com.javarush.test.level17.lesson04.task01;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\* Синхронизированные заметки  
1. Класс Note будет использоваться нитями.   
Поэтому сделай так, чтобы обращения к листу notes блокировали мютекс notes, не this  
2. Все System.out.println не должны быть заблокированы (синхронизированы), т.е. не должны находиться в блоке synchronized  
\*/  
  
public class Solution {  
 public static class Note {  
  
 public final List<String> notes = new ArrayList<String>();  
  
 public void addNote(int index, String note) {  
 System.*out*.println("Сейчас будет добавлена заметка [" + note + "] На позицию " + index);  
 notes.add(index, note);  
 System.*out*.println("Уже добавлена заметка [" + note + "]");  
 }  
  
 public void removeNote(int index) {  
 System.*out*.println("Сейчас будет удалена заметка с позиции " + index);  
 String note = notes.remove(index);  
 System.*out*.println("Уже удалена заметка [" + note + "] с позиции " + index);  
 }  
 }  
}

package com.javarush.test.level17.lesson04.task02;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\* Синхронизированные заметки 2  
Класс Note будет использоваться нитями. Поэтому сделай так, чтобы вcе методы были синхронизированы  
\*/  
  
public class Solution {  
 public static class Note {  
  
 public final List<String> notes = new ArrayList<String>();  
  
 public void addNote(int index, String note) {  
 System.*out*.println("Сейчас будет добавлена заметка [" + note + "] На позицию " + index);  
 notes.add(index, note);  
 System.*out*.println("Уже добавлена заметка [" + note + "]");  
 }  
  
 public void removeNote(int index) {  
 System.*out*.println("Сейчас будет удалена заметка с позиции " + index);  
 String note = notes.remove(index);  
 System.*out*.println("Уже удалена заметка [" + note + "] с позиции " + index);  
 }  
 }  
}

package com.javarush.test.level17.lesson04.task03;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\* Сад-огород  
1. Создайте метод public void addFruit(int index, String fruit) - который добавляет параметр fruit в лист fruits на позицию index  
2. Создайте метод public void removeFruit(int index) - который удаляет из fruits элемент с индексом index  
3. Создайте метод public void addVegetable(int index, String vegetable) - который добавляет параметр vegetable в лист vegetables на позицию index  
4. Создайте метод public void removeVegetable(int index) - который удаляет из vegetables элемент с индексом index  
5. Класс Garden будет использоваться нитями. Поэтому сделай так, чтобы все методы блокировали мютекс this  
6. Реализуй это минимальным количеством кода  
\*/  
  
public class Solution {  
 public static class Garden {  
  
 public final List<String> fruits = new ArrayList<String>();  
 public final List<String> vegetables = new ArrayList<String>();  
  
 }  
}

package com.javarush.test.level17.lesson04.task04;  
  
/\* Синхронизированный президент  
И снова Singleton паттерн - синхронизация в статическом блоке  
Внутри класса OurPresident в статическом блоке создайте синхронизированный блок.  
Внутри синхронизированного блока инициализируйте president.  
\*/  
  
public class Solution {  
 public static class OurPresident {  
  
 private static OurPresident *president*;  
  
 private OurPresident() {  
 }  
  
 public static OurPresident getOurPresident() {  
 return *president*;  
 }  
 }  
}

package com.javarush.test.level17.lesson04.task05;  
  
/\* МВФ  
Singleton паттерн - синхронизация в методе  
IMF - это Международный Валютный Фонд  
Внутри метода getFund создайте синхронизированный блок  
Внутри синхронизированного блока инициализируйте переменную imf так, чтобы метод getFund всегда возвращал один и тот же объект  
\*/  
  
public class Solution {  
 public static class IMF {  
  
 private static IMF *imf*;  
  
 public static IMF getFund() {  
 //add your code here - добавь код тут  
 return *imf*;  
 }  
  
 private IMF() {  
 }  
 }  
}

package com.javarush.test.level17.lesson06.task01;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\* Заметки для всех  
Класс Note будет использоваться нитями.  
Поэтому сделай так, чтобы лист notes находился в общей памяти  
\*/  
  
public class Solution {  
  
 public static class Note {  
  
 public List<String> notes = new ArrayList<String>();  
  
 public void addNote(int index, String note) {  
 System.*out*.println("Сейчас будет добавлена заметка [" + note + "] На позицию " + index);  
 notes.add(index, note);  
 System.*out*.println("Уже добавлена заметка [" + note + "]");  
 }  
  
 public void removeNote(int index) {  
 System.*out*.println("Сейчас будет удалена заметка с позиции " + index);  
 String note = notes.remove(index);  
 System.*out*.println("Уже удалена заметка [" + note + "] с позиции " + index);  
 }  
 }  
}

package com.javarush.test.level17.lesson06.task02;  
  
/\* Предложения  
Не используя synchronized сделайте так, чтобы количество сделанных и принятых предложений было одинаковым.  
\*/  
  
public class Solution {  
 public static int *proposal* = 0;  
  
 public static void main(String[] args) {  
 new MakeProposal().start();  
 new AcceptProposal().start();  
 }  
  
 public static class MakeProposal extends Thread {  
 @Override  
 public void run() {  
 int thisProposal = *proposal*;  
  
 while (*proposal* < 10) {  
 System.*out*.println("Сделано предложение №" + (thisProposal + 1));  
 *proposal* = ++thisProposal;  
 try {  
 Thread.*sleep*(100);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
  
 public static class AcceptProposal extends Thread {  
 @Override  
 public void run() {  
 int thisProposal = *proposal*;  
  
 while (thisProposal < 10) {  
 if (thisProposal != *proposal*) {  
 System.*out*.println("Принято предложение №" + *proposal*);  
 thisProposal = *proposal*;  
 }  
 }  
 }  
 }  
}

package com.javarush.test.level17.lesson10.home01;  
  
import java.util.\*;  
  
/\* Общий список  
1. Изменить класс Solution так, чтобы он стал списком. (Необходимо реализовать интерфейс java.util.List).  
2. Список Solution должен работать только с целыми числами Long.  
3. Воспользуйтесь полем original.  
4. Список будет использоваться нитями, поэтому позаботьтесь, чтобы все методы были синхронизированы.  
\*/  
  
public class Solution {  
 private ArrayList<Long> original = new ArrayList<Long>();  
}

package com.javarush.test.level17.lesson10.home02;  
  
/\* Comparable  
Реализуйте интерфейс Comparable<Beach> в классе Beach, который будет использоваться нитями.  
\*/  
  
public class Beach {  
 private String name; //название  
 private float distance; //расстояние  
 private int quality; //качество  
  
 public Beach(String name, float distance, int quality) {  
 this.name = name;  
 this.distance = distance;  
 this.quality = quality;  
 }  
  
 public synchronized String getName() {  
 return name;  
 }  
  
 public synchronized void setName(String name) {  
 this.name = name;  
 }  
  
 public synchronized float getDistance() {  
 return distance;  
 }  
  
 public synchronized void setDistance(float distance) {  
 this.distance = distance;  
 }  
  
 public synchronized int getQuality() {  
 return quality;  
 }  
  
 public synchronized void setQuality(int quality) {  
 this.quality = quality;  
 }  
   
}

package com.javarush.test.level17.lesson10.home03;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\* Аптека  
Реализуй интерфейс Runnable в классах Apteka и Person.  
Все нити должны работать пока не isStopped  
Логика для Apteka: drugsController должен сделать закупку случайного лекарства (getRandomDrug) в количестве (getRandomCount) и подождать 300 мс  
Логика для Person: drugsController должен сделать продажу случайного лекарства (getRandomDrug) в количестве (getRandomCount) и подождать 100 мс  
Расставь synchronized там, где это необходимо  
\*/  
  
public class Solution {  
 public static DrugsController *drugsController* = new DrugsController();  
 public static boolean *isStopped* = false;  
  
 public static void main(String[] args) throws InterruptedException {  
 Thread apteka = new Thread(new Apteka());  
 Thread man = new Thread(new Person(), "Мужчина");  
 Thread woman = new Thread(new Person(), "Женщина");  
  
 apteka.start();  
 man.start();  
 woman.start();  
  
 Thread.*sleep*(1000);  
 *isStopped* = true;  
 }  
  
 public static class Apteka {  
  
 }  
  
 public static class Person {  
  
 }  
  
 public static int getRandomCount() {  
 return (int) (Math.*random*() \* 3) + 1;  
 }  
  
 public static Drug getRandomDrug() {  
 int index = (int) ((Math.*random*() \* 1000) % (*drugsController*.*allDrugs*.size()));  
 List<Drug> drugs = new ArrayList<>(*drugsController*.*allDrugs*.keySet());  
 return drugs.get(index);  
 }  
  
 private static void waitAMoment() {  
 try {  
 Thread.*sleep*(100);  
 } catch (InterruptedException e) {  
 }  
 }  
}

package com.javarush.test.level17.lesson10.home03;  
  
import java.util.HashMap;  
import java.util.Map;  
  
public class DrugsController {  
 public static Map<Drug, Integer> *allDrugs* = new HashMap<Drug, Integer>(); // <Лекарство, Количество>  
  
 static {  
 Drug panadol = new Drug();  
 panadol.setName("Панадол");  
 *allDrugs*.put(panadol, 5);  
  
 Drug analgin = new Drug();  
 analgin.setName("Анальгин");  
 *allDrugs*.put(analgin, 18);  
  
 Drug placebo = new Drug();  
 placebo.setName("Плацебо");  
 *allDrugs*.put(placebo, 1);  
 }  
  
 public void sell(Drug drug, int count) {  
 String name = Thread.*currentThread*().getName();  
 if (!*allDrugs*.containsKey(drug)) {  
 System.*out*.println("Нет в наличии");  
 }  
 Integer currentCount = *allDrugs*.get(drug);  
 if (currentCount < count) {  
 System.*out*.println(String.*format*("%s хочет %s %d шт. В наличии - %d", name, drug.getName(), count, currentCount));  
 } else {  
 *allDrugs*.put(drug, (currentCount - count));  
 System.*out*.println(String.*format*("%s купил(а) %s %d шт. Осталось - %d", name, drug.getName(), count, (currentCount - count)));  
 }  
 }  
  
 public void buy(Drug drug, int count) {  
 System.*out*.println("Закупка " + drug.getName() + " " + count);  
 if (!*allDrugs*.containsKey(drug)) {  
 *allDrugs*.put(drug, 0);  
 }  
 Integer currentCount = *allDrugs*.get(drug);  
 *allDrugs*.put(drug, (currentCount + count));  
 }  
}

package com.javarush.test.level17.lesson10.home03;  
  
public class Drug {  
 private String name;  
 private String description;  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getDescription() {  
 return description;  
 }  
  
 public void setDescription(String description) {  
 this.description = description;  
 }  
}

package com.javarush.test.level17.lesson10.home04;  
  
/\* Синхронизированные методы  
Установить модификатор synchronized только тем методам, которым необходимо.  
Объект класса Solution будет использоваться нитями.  
\*/  
  
public class Solution {  
 private double param = Math.*random*();  
 private StringBuilder sb = new StringBuilder();  
  
 private void method0() {  
 double i = method3();  
 }  
  
 protected void method1(String param1) {  
 Solution solution = new Solution();  
 solution.method0();  
 }  
  
 public void method2(int param1) {  
 param1++;  
 }  
  
 double method3() {  
 double random = Math.*random*();  
 param += 40.7;  
 return random + param;  
 }  
  
 private void method4() {  
 sb.append(1).append(1).append(1).append(1);  
 }  
  
 protected void method5(String param2) {  
 new StringBuffer().append(param2).append(param2).append(param2);  
 }  
  
 public String method6(int param2) {  
 System.*out*.println("Thinking....");  
 method7(5e-2);  
 sb = new StringBuilder("Got it!.");  
 return sb.toString();  
 }  
  
 String method7(double param2) {  
 return "" + param2;  
 }  
  
}

package com.javarush.test.level17.lesson10.home05;  
  
/\* Лишняя синхронизация  
synchronized существенно замедляет программу, поэтому убери избыточность synchronized внутри методов  
\*/  
  
public class Solution {  
 char[] value;  
 int count;  
  
 public Solution append(CharSequence s) {  
 synchronized (Solution.class) {  
 if (s == null) {  
 synchronized (this) {  
 s = "null";  
 }  
 }  
  
 if (s instanceof String) {  
 synchronized (this) {  
 return this.append((String) s);  
 }  
 }  
  
 if (s instanceof Solution) {  
 synchronized (this) {  
 return this.appendThis((Solution) s);  
 }  
 }  
 }  
 return this.append(s);  
 }  
  
 public synchronized Solution appendThis(Solution s) {  
 //do something here....  
 return this;  
 }  
  
 private static final java.io.ObjectStreamField[] *serialPersistentFields* =  
 {  
 new java.io.ObjectStreamField("value", char[].class),  
 new java.io.ObjectStreamField("count", Integer.*TYPE*),  
 new java.io.ObjectStreamField("shared", Boolean.*TYPE*),  
 };  
  
 private synchronized void writeObject(java.io.ObjectOutputStream s) throws java.io.IOException {  
 java.io.ObjectOutputStream.PutField fields = s.putFields();  
 synchronized (fields) {  
 fields.put("value", value);  
 fields.put("count", count);  
 fields.put("shared", false);  
 }  
 synchronized (s) {  
 s.writeFields();  
 }  
 }  
  
 private void readObject(java.io.ObjectInputStream s) throws java.io.IOException, ClassNotFoundException {  
 synchronized (new java.io.IOException()) {  
 java.io.ObjectInputStream.GetField fields = s.readFields();  
 value = (char[]) fields.get("value", null);  
 count = fields.get("count", 0);  
 }  
 }  
}

package com.javarush.test.level17.lesson10.home06;  
  
/\* Глажка  
И снова быт...  
Поставьте один synchronized, чтобы diana и igor гладили по-очереди, ведь утюг всего один!  
Подсказка: использовать блокировку на уровне класса.  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 Person diana = new Person("Diana");  
 Person igor = new Person("Igor");  
 }  
  
 public static class Person extends Thread { //Человек  
  
 public Person(String name) {  
 super(name);  
 start();  
 }  
  
 @Override  
 public void run() {  
 Iron iron = takeIron();  
 Clothes clothes = takeClothes();  
 ironing(iron, clothes);  
 returnIron();  
 }  
  
 protected Iron takeIron() {  
 System.*out*.println("Taking an Iron");  
 return new Iron();  
 }  
  
 protected Iron returnIron() {  
 System.*out*.println("Returning the Iron");  
 return new Iron();  
 }  
  
 protected Clothes takeClothes() {  
 return new Clothes("T-shirt");  
 }  
  
 protected void ironing(Iron iron, Clothes clothes) {  
 System.*out*.println(getName() + "'s ironing the " + clothes.name);  
 }  
 }  
  
 public static class Iron {  
 } //Утюг  
  
 public static class Clothes {//Одежда  
 String name;  
  
 public Clothes(String name) {  
 this.name = name;  
 }  
 }  
}

package com.javarush.test.level17.lesson10.home07;  
  
import java.util.HashMap;  
import java.util.Map;  
  
/\* ApplicationContext  
ApplicationContext будет доступен множеству нитей.  
Сделать так, чтобы данные не терялись: подумай, какое ключевое слово необходимо поставить и где.  
\*/  
  
public abstract class ApplicationContext<GenericsBean extends Bean> {  
 private Map<String, GenericsBean> container = new HashMap<String, GenericsBean>();  
 //Map<Name, some class implemented Bean interface>  
  
  
 protected ApplicationContext() {  
 parseAllClassesAndInterfaces();  
 }  
  
 public GenericsBean getByName(String name) {  
 return container.get(name);  
 }  
  
 public GenericsBean removeByName(String name) {  
 return container.remove(name);  
 }  
  
 protected abstract void parseAllClassesAndInterfaces();  
}

package com.javarush.test.level17.lesson10.home07;  
  
public interface Bean { //это интерфейс-маркер  
}

package com.javarush.test.level17.lesson10.home08;  
  
/\* Банкомат  
Разберись, как работает программа  
Во время тестирования лог содержит следующее:  
.....  
Добавляем 100, на счету 1100  
Добавляем 100, на счету 1200  
Тратим 1000, на счету 100  
Недостаточно денег  
.....  
  
Создан баг: При списании денег со счета теряются деньги  
Найти и исправить ошибку  
\*/  
  
public class Bankomat {  
  
 static BankAccount *account* = new BankAccount("Amigo");  
  
 public static volatile boolean *isStopped*;  
  
 public static void main(String[] args) throws InterruptedException {  
 *addMoney*.start();  
 new SpendThread();  
 new SpendThread();  
 new SpendThread();  
 Thread.*sleep*(4000);  
 *isStopped* = true;  
 }  
  
 private static Thread *addMoney* = new Thread() {  
 @Override  
 public void run() {  
 while (!*isStopped*) {  
 *account*.deposit("1000"); //кладем на счет  
 try {  
 Thread.*sleep*(1000);  
 } catch (InterruptedException e) {  
 break;  
 }  
 }  
 }  
 };  
  
  
 public static class SpendThread extends Thread {  
 public SpendThread() {  
 start();  
 }  
  
 @Override  
 public void run() {  
 while (!*isStopped*) {  
 try {  
 *account*.withdraw("100"); //снимаем со счета  
 } catch (NotEnoughMoneyException e) {  
 System.*out*.println("Недостаточно денег");  
 }  
 try {  
 Thread.*sleep*(100);  
 } catch (InterruptedException e) {  
 break;  
 }  
 }  
 }  
 }  
  
 ;  
}

package com.javarush.test.level17.lesson10.home08;  
  
public class NotEnoughMoneyException extends Exception {  
}

package com.javarush.test.level17.lesson10.home08;  
  
import java.math.BigDecimal;  
  
public class BankAccount {  
 private BigDecimal balance;  
 private String owner;  
  
 public BankAccount(String owner) {  
 this(BigDecimal.*ZERO*, owner);  
 }  
  
 public BankAccount(BigDecimal balance, String owner) {  
 this.balance = balance;  
 this.owner = owner;  
 }  
  
 public void deposit(BigDecimal money) {  
 BigDecimal newBalance = balance.add(money);  
 System.*out*.println("Добавляем " + money + ", на счету " + newBalance);  
 balance = newBalance;  
 }  
  
 public synchronized void withdraw(BigDecimal money) throws NotEnoughMoneyException {  
 BigDecimal newBalance = balance.subtract(money);  
  
 if (newBalance.compareTo(BigDecimal.*ZERO*) < 0) throw new NotEnoughMoneyException();  
  
 balance = newBalance;  
 System.*out*.println("Тратим " + money + ", на счету " + balance);  
 }  
  
 public void deposit(String money) {  
 deposit(new BigDecimal(money));  
 }  
  
 public void withdraw(String money) throws NotEnoughMoneyException {  
 withdraw(new BigDecimal(money));  
 }  
}

package com.javarush.test.level17.lesson10.home09;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\* Транзакционность  
Сделать метод joinData транзакционным, т.е. если произошел сбой, то данные не должны быть изменены.  
1. Считать с консоли 2 имени файла  
2. Считать построчно данные из файлов. Из первого файла - в allLines, из второго - в forRemoveLines  
В методе joinData:  
3. Если список allLines содержит все строки из forRemoveLines, то удалить из списка allLines все строки, которые есть в forRemoveLines  
4. Если список allLines НЕ содержит каких-либо строк, которые есть в forRemoveLines, то  
4.1. очистить allLines от данных  
4.2. выбросить исключение CorruptedDataException  
Метод joinData должен вызываться в main. Все исключения обработайте в методе main.  
\*/  
  
public class Solution {  
 public static List<String> *allLines* = new ArrayList<String>();  
 public static List<String> *forRemoveLines* = new ArrayList<String>();  
  
 public static void main(String[] args) {  
 }  
  
 public void joinData () throws CorruptedDataException {  
  
 }  
}

package com.javarush.test.level17.lesson10.home09;  
  
import java.io.IOException;  
  
public class CorruptedDataException extends IOException {  
}

package com.javarush.test.level17.lesson10.home10;  
  
/\* Посчитаем  
1. Сделай так, чтобы результат успел посчитаться для всех элементов массива values НЕ используя Thread.sleep  
2. Исправь synchronized блок так, чтобы массив values заполнился значением 1  
\*/  
  
public class Solution {  
 public static void main(String[] args) throws InterruptedException {  
 Counter counter1 = new Counter();  
 Counter counter2 = new Counter();  
 Counter counter3 = new Counter();  
 Counter counter4 = new Counter();  
  
 counter1.start();  
 counter2.start();  
 counter3.start();  
 counter4.start();  
  
 for (int i = 1; i <= 100; i++) {  
 if (*values*[i] != 1) {  
 System.*out*.println("Массив values содержит элементы неравные 1");  
 break;  
 }  
 }  
 }  
  
 public static Integer *count* = 0;  
 public static int[] *values* = new int[105];  
  
 static {  
 for (int i = 0; i < 105; i++) {  
 *values*[i] = 0;  
 }  
 }  
  
 public static void incrementCount() {  
 *count*++;  
 }  
  
 public static int getCount() {  
 return *count*;  
 }  
  
 public static class Counter extends Thread {  
 @Override  
 public void run() {  
 do {  
 synchronized (this) {  
 *incrementCount*();  
 *values*[*getCount*()]++;  
 }  
  
 try {  
 Thread.*sleep*(1);  
 } catch (InterruptedException e) {  
 }  
 } while (*getCount*() < 100);  
 }  
 }  
}