package com.javarush.task.task22.task2201;  
  
/\*   
Строки нитей или строковые нити? Вот в чем вопрос  
1. Метод getPartOfString должен возвращать подстроку между первой и последней табуляцией.  
2. На некорректные данные getPartOfString должен бросить исключение:  
а) TooShortStringFirstThreadException, если имя трэда FIRST\_THREAD\_NAME.  
б) TooShortStringSecondThreadException, если имя трэда SECOND\_THREAD\_NAME.  
в) RuntimeException в других случаях.  
3. Реализуйте логику трех protected методов в ThisUncaughtExceptionHandler используя вызовы соответствующих методов согласно следующим шаблонам:  
a) 1# : TooShortStringFirstThreadException : java.lang.StringIndexOutOfBoundsException: String index out of range: -1  
б) java.lang.StringIndexOutOfBoundsException: String index out of range: -1 : TooShortStringSecondThreadException : 2#  
в) RuntimeException : java.lang.StringIndexOutOfBoundsException: String index out of range: -1 : 3#  
\*/  
public class Solution {  
 public static void main(String[] args) {  
 new Solution();  
 }  
  
 public static final String *FIRST\_THREAD\_NAME* = "1#";  
 public static final String *SECOND\_THREAD\_NAME* = "2#";  
  
 private Thread thread1;  
 private Thread thread2;  
 private Thread thread3;  
  
 public Solution() {  
 initThreads();  
 }  
  
 protected void initThreads() {  
 this.thread1 = new Thread(new Task(this, "A\tB\tC\tD\tE\tF\tG\tH\tI"), *FIRST\_THREAD\_NAME*);  
 this.thread2 = new Thread(new Task(this, "J\tK\tL\tM\tN\tO\tP\tQ\tR\tS\tT\tU\tV\tW\tX\tY\tZ"), *SECOND\_THREAD\_NAME*);  
 this.thread3 = new Thread(new Task(this, "\t\t"), "3#");  
  
 Thread.*setDefaultUncaughtExceptionHandler*(new ThisUncaughtExceptionHandler());  
  
 this.thread1.start();  
 this.thread2.start();  
 this.thread3.start();  
 }  
  
 public synchronized String getPartOfString(String string, String threadName) {  
 return null;  
 }  
}

package com.javarush.task.task22.task2202;  
  
/\*   
Найти подстроку  
Метод getPartOfString должен возвращать подстроку начиная с символа после 1-го пробела и до конца слова,  
которое следует после 4-го пробела.  
  
Пример:  
«JavaRush — лучший сервис обучения Java.»  
  
Результат:  
«— лучший сервис обучения»  
  
На некорректные данные бросить исключение TooShortStringException (сделать исключением).  
\*/  
public class Solution {  
 public static void main(String[] args) {  
 System.*out*.println(*getPartOfString*("JavaRush - лучший сервис обучения Java."));  
 }  
  
 public static String getPartOfString(String string) {  
 return null;  
 }  
  
 public static class TooShortStringException {  
 }  
}

package com.javarush.task.task22.task2203;  
  
/\*   
Между табуляциями  
Метод getPartOfString должен возвращать подстроку между первой и второй табуляцией.  
На некорректные данные бросить исключение TooShortStringException.  
Класс TooShortStringException не менять.  
\*/  
public class Solution {  
 public static String getPartOfString(String string) {  
 return null;  
 }  
  
 public static class TooShortStringException extends Exception {  
 }  
  
 public static void main(String[] args) throws TooShortStringException {  
 System.*out*.println(*getPartOfString*("\tJavaRush - лучший сервис \tобучения Java\t."));  
 }  
}

package com.javarush.task.task22.task2204;  
  
/\*   
Форматирование строки  
Исправить метод getFormattedString так, чтобы он возвращал строку с параметрами для форматирования.  
Для перевода каретки не используйте n.  
  
Должен быть вывод:  
20 / 7 = 2,86  
Exp = 3,33e+00  
\*/  
public class Solution {  
 public static void main(String[] args) {  
 System.*out*.println(String.*format*(*getFormattedString*(), 20.0 / 7.0, 10.0 / 3.0));  
 //должен быть вывод  
 //20 / 7 = 2,86  
 //Exp = 3,33e+00  
 }  
  
 public static String getFormattedString() {  
 return "20 / 7 = % %Exp = %";  
 }  
}

package com.javarush.task.task22.task2205;  
  
/\*   
МНЕ нравится курс JavaRush  
Исправить метод getFormattedString так, чтобы он возвращал строку с параметрами для форматирования.  
Параметры должны меняться местами.  
  
Должен быть вывод:  
МНЕ нравится курс JavaRush  
\*/  
public class Solution {  
 public static void main(String[] args) {  
 System.*out*.println(String.*format*(*getFormattedString*(), "JavaRush", "курс", "мне", "нравится"));  
 //должен быть вывод  
 //"МНЕ нравится курс JavaRush"  
 }  
 public static String getFormattedString() {  
 return "";  
 }  
}

package com.javarush.task.task22.task2206;  
  
import java.util.Date;  
  
/\*   
Форматирование даты  
Исправить метод getFormattedString так, чтобы он возвращал строку с параметрами для форматирования.  
  
Должен быть вывод аналогичный следующему:  
31:10:13 15:59:59  
\*/  
public class Solution {  
 public static void main(String[] args) {  
 Date date = new Date();  
 System.*out*.println(String.*format*(*getFormattedString*(), date, date, date, date, date, date));  
 //должен быть вывод аналогичный следующему  
 //31:10:13 15:59:59 - dd:MM:yy hh:mm:ss  
 }  
  
 public static String getFormattedString() {  
 return "";  
 }  
}

package com.javarush.task.task22.task2207;  
  
import java.util.LinkedList;  
import java.util.List;  
  
/\*   
Обращенные слова\  
В методе main с консоли считать имя файла, который содержит слова, разделенные пробелами.  
Найти в тексте все пары слов, которые являются обращением друг друга. Добавить их в result.  
Использовать StringBuilder.  
  
Пример содержимого файла  
рот тор торт о  
о тот тот тот  
  
Вывод:  
рот тор  
о о  
тот тот  
\*/  
public class Solution {  
 public static List<Pair> *result* = new LinkedList<>();  
  
 public static void main(String[] args) {  
  
 }  
  
 public static class Pair {  
 String first;  
 String second;  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
  
 Pair pair = (Pair) o;  
  
 if (first != null ? !first.equals(pair.first) : pair.first != null) return false;  
 return second != null ? second.equals(pair.second) : pair.second == null;  
  
 }  
  
 @Override  
 public int hashCode() {  
 int result = first != null ? first.hashCode() : 0;  
 result = 31 \* result + (second != null ? second.hashCode() : 0);  
 return result;  
 }  
  
 @Override  
 public String toString() {  
 return first == null && second == null ? "" :  
 first == null && second != null ? second :  
 second == null && first != null ? first :  
 first.compareTo(second) < 0 ? first + " " + second : second + " " + first;  
  
 }  
 }  
  
}

package com.javarush.task.task22.task2209;  
  
/\*   
Составить цепочку слов  
В методе main считай с консоли имя файла, который содержит слова, разделенные пробелом.  
В методе getLine используя StringBuilder расставь все слова в таком порядке, чтобы последняя буква данного слова совпадала с первой буквой следующего не учитывая регистр.  
Каждое слово должно участвовать 1 раз.  
Метод getLine должен возвращать любой вариант.  
Слова разделять пробелом.  
Вывести полученную строку на экран.  
  
Пример тела входного файла:  
Киев Нью-Йорк Амстердам Вена Мельбурн  
  
Результат:  
Амстердам Мельбурн Нью-Йорк Киев Вена  
\*/  
public class Solution {  
 public static void main(String[] args) {  
 //...  
 StringBuilder result = *getLine*();  
 System.*out*.println(result.toString());  
 }  
  
 public static StringBuilder getLine(String... words) {  
 return null;  
 }  
}

package com.javarush.task.task22.task2210;  
  
/\*   
StringTokenizer  
Используя StringTokenizer разделить query на части по разделителю delimiter.  
  
Пример  
getTokens("level22.lesson13.task01", ".")  
  
Возвращает  
{"level22", "lesson13", "task01"}  
\*/  
public class Solution {  
 public static void main(String[] args) {  
  
 }  
 public static String [] getTokens(String query, String delimiter) {  
 return null;  
 }  
}

package com.javarush.task.task22.task2211;  
  
import java.io.\*;  
import java.nio.charset.Charset;  
  
/\*   
Смена кодировки  
В метод main первым параметром приходит имя файла, тело которого в кодировке Windows-1251.  
В метод main вторым параметром приходит имя файла, в который необходимо записать содержимое первого файла в кодировке UTF-8.  
  
\*/  
public class Solution {  
 static String *win1251TestString* = "РќР°СЂСѓС€РµРЅРёРµ РєРѕРґРёСЂРѕРІРєРё РєРѕРЅСЃРѕР»Рё?"; //only for your testing  
  
 public static void main(String[] args) throws IOException {  
  
 }  
}

package com.javarush.task.task22.task2212;  
  
/\*   
Проверка номера телефона  
Проверка номера телефона  
Метод checkTelNumber должен проверять, является ли аргумент telNumber валидным номером телефона.  
  
Критерии валидности:  
1) если номер начинается с ‘+‘, то он содержит 12 цифр  
2) если номер начинается с цифры или открывающей скобки, то он содержит 10 цифр  
3) может содержать 0-2 знаков ‘—‘, которые не могут идти подряд  
4) может содержать 1 пару скобок ‘(‘ и ‘)‘ , причем если она есть, то она расположена левее знаков ‘-‘  
5) скобки внутри содержат четко 3 цифры  
6) номер не содержит букв  
7) номер заканчивается на цифру  
  
Примеры:  
+380501234567 - true  
+38(050)1234567 - true  
+38050123-45-67 - true  
050123-4567 - true  
+38)050(1234567 - false  
+38(050)1-23-45-6-7 - false  
050ххх4567 - false  
050123456 - false  
(0)501234567 - false  
\*/  
public class Solution {  
 public static boolean checkTelNumber(String telNumber) {  
 return false;  
 }  
  
 public static void main(String[] args) {  
  
 }  
}

Змейка(1)

А теперь напишем игру «***змейка***«.

**Для этого нам понадобятся:**  
а) собственно сама змея (класс **Snake**)  
б) комната, в которой она будет ползать (класс **Room**)  
в) мыши, которых она будет есть (класс **Mouse**)

Создай классы **Snake**, **Room**, **Mouse**

Не забудь создать метод public static void main(String[] **args**) в классе **Room**.

### Змейка(2)

Теперь самое интересное — как будет устроена змея.

**Что можно сказать о змее:**  
а) По сути — змея это **набор кусочков — клеточек**. Но точно известно, что один из них — **голова**.  
б) У змеи есть **направление движения**.

**Так что нам еще понадобятся классы:**  
**SnakeSection** — кусочек змеи.  
**SnakeDirection** — класс (enum) будет описывать направление движения змеи.

Создай классы **SnakeSection** и **SnakeDirection**.

### Змейка(3)

Отлично. Я еще добавил класс **KeyboardObserver** на будущее.  
Изучи его внимательно. Пригодится.

Начнем с самого простого — класса **Mouse**.  
Мышь не будет двигаться и у нее будут только координаты и все.  
Добавь два приватных поля в класс **Mouse**: x и y типа int.

### Змейка(4)

Координаты x и y ты уже создал в классе **Mouse**, а как получить к ним доступ?  
Создай корректные геттеры для этих полей и конструктор с двумя параметрами типа int (x и y соответственно).

Змейка(5)

Теперь доделаем enum **SnakeDirection** так, чтобы он содержал возможные направления движения змейки.

Добавь **4** значения **UP**, **DOWN**, **LEFT**, **RIGHT**.

**Должно получиться примерно так:**  
public enum SnakeDirection  
{  
 UP,  
 RIGHT,  
 DOWN,  
 LEFT  
}

### Змейка(6)

Теперь займемся классом **SnakeSection**.  
Он будет описывать один кусочек змеи.

А у каждого кусочка что есть?  
Правильно! **Координаты**.

Добавь два поля в класс **SnakeSection**: x и y типа int.  
Добавь для них **getter’ы** и конструктор с двумя параметрами типа int.

### Змейка(7)

«***Кусочки змеи***» мы описали, теперь займемся самой змеей.

У змеи должен быть список кусочков(**SnakeSection**) и голова.  
Пусть головой будет просто самый первый кусочек (с номером **0**).

У змеи также есть направление движения — сторона, куда она движется по умолчанию.  
А еще есть состояние — жива змея или мертва.  
Игра заканчивается тогда, когда змея мертва.

**Надо:**  
а) Добавить в класс **Snake** поле sections типа **List**<SnakeSection>.  
б) Добавить поле isAlive типа **boolean**.  
в) Добавить поле direction типа **SnakeDirection**.  
г) Для всех полей добавить **getter’ы**, а для direction еще и **setter**.

**Примечание:**  
Для полей логического типа геттер принято писать не как getValue(), а как isValue() — это улучшает читабельность кода.  
Только не пиши isIsAlive(). Одного **is** будет вполне достаточно.

**Примечание 2:**  
Все поля создаваемые в этом задании должны быть приватными, а методы — публичными.

Змейка(8)

Настала очередь конструктора класса **Snake**.

Змея у нас будет поначалу состоять из одного кусочка — головы.  
А что для него требуется?  
Что необходимо передать в конструктор?  
Координаты змеи, конечно же.

**Надо:**  
а) Передать в конструктор координаты головы змеи (x и y)  
б) создать в нем первый «***кусочек змеи***» (*голову*) и добавить его в коллекцию **sections**.  
в) isAlive выставить в **true**  
г) не забудь в конструкторе инициализировать переменную sections. В **null** не много-то и добавишь!  
д) создать и реализовать метод int **getX()**. Метод должен вернуть координату Х головы змеи.  
е) создать и реализовать метод int **getY()**. Метод должен вернуть координату Y головы змеи.  
ж) еще добавить классу метод move()— он нам пригодится попозже.  
з) созданный конструктор должен быть публичным.

### Змейка(9)

Еще остался самый главный класс — **Room**.  
Что нам нужно для его описания?  
Размеры комнаты (width и **height**) — раз.  
**Змея** — **два**  
**Мышь** — **три**.

**Надо:**  
а) создать в классе **Room** переменные width и height типа int.  
б) создать в классе **Room** переменную snake типа **Snake**.  
в) создать в классе **Room** переменную mouse типа **Snake**. Шучу. Типа **Mouse**, конечно же :).  
г) создать для них всех **геттеры** и **сеттеры**.  
д) создать конструктор. Конструктор должен принимать три параметра: width, height и … snake!

### Змейка(10)

Займемся методом main.

Для начала надо создать в классе **Room** статическое поле game типа **Room**. Доступ к переменной должен быть из всех классов.

**Затем в методе**main**:**  
а) Создать змею — объект типа **Snake**.  
б) Создать комнату — объект типа **Room** и передать в него ширину, высоту и змею.  
в) Установить змее direction равным SnakeDirection.DOWN

Чего еще не хватает? Собственно метода, в котором будет идти вся игровая логика.  
**И еще нужен метод, который бы отрисовывал все это на экране:**  
а) создать метод void run()  
б) создать метод void print()

### Змейка(11)

Теперь логика управления мышью.

С мышью у нас будут происходить две вещи.

Первая - змея съедает мышь.

Вторая - появляется новая мышь в случайной точке комнаты.

Надо написать и реализовать метод createMouse() в классе Room.

В этом методе мы просто должны создавать новую мышь со случайными координатами в комнате.

Как получить случайные координаты?

Это ты уже должен был знать. На всякий случай даю подсказку:

int x = (int) (Math.random() \* width);

Еще понадобится метод - eatMouse(), на случай, если мышь все-таки кто-то съест :)

Пока сложной логики в этом методе не будет - просто будем вызывать метод createMouse и все.

### Змейка(12)

Змея у нас есть, пусть и мышь будет.

**Добавить в метод main:**  
a) вызов метода createMouse().  
б) вызов метода run().  
Без него ничего работать не будет. В нем вся основная логика.  
Неплохо получилось, но я все-таки внесу пару правок.  
Кстати, как насчет написать метод sleep?  
Ты уже понял из предыдущих задач, что в методе run нужна пауза.  
Но насколько я помню, скорость в змейке должна расти при росте ее длины.  
Значит чем длиннее змея, тем выше скорость и меньше пауза.

### Змейка(13)

Предлагаю тебе в этот раз написать специальный метод sleep(), который будет делать паузу в зависимости от длины змеи (количества элементов в **sections**).  
Придумай какой-нибудь хитрый алгоритм. Чтобы на первом уровне пауза была **500** **миллисекунд**,  
а к **10** **уровню** постепенно уменьшилась до **300**. А к **15** до **200**. И дальше оставалась постоянной.

### Змейка(14)

Теперь поработаем над методом print().

**Надо:**  
а) вывести на экран прямоугольник из точек размером width x height.  
б) тело змеи отметить символом «***x***«-**английское**  
в) голову змеи нарисовать символом «***X***«-**английское**.

**Подсказка:**  
а) удобно сначала создать матрицу типа **int[][]** с размером (height x width)  
б) затем пройтись по всем объектам и отрисовать их в матрицу.  
Например, тело змеи — **1**, голова змеи — **2**, мышь — **3**.

### Змейка(15)

Теперь осталось допилить змею.

**Вот что я предлагаю насчет движения змеи:**  
Змея состоит из кусочков. Давай каждый ход просто добавлять один кусочек со стороны головы,  
а самый последний — удалять. Тогда получится, что **змея ползет**.

Давай добавим два метода move, один без параметров, а другой с двумя параметрами типа int.

**В методе** move **без параметров необходимо:**  
а) прекратить движение если змея умерла(isAlive == **false**)  
б) вызвать метод move(**0**, **-1**) если направление движения равно SnakeDirection.UP  
в) вызвать метод move(**1**, **0**) если направление движения равно SnakeDirection.RIGHT  
г) вызвать метод move(**0**, **1**) если направление движения равно SnakeDirection.DOWN  
д) вызвать метод move(**-1**, **0**) если направление движения равно SnakeDirection.LEFT

Метод move с параметрами int, int пока оставим пустым.

### Змейка(16)

Ничто не вечно. Так и змея **должна умирать**, если она врезается в стену или саму себя.

Для определения, не пересекается ли змея сама с собой, можно сделать очень простую проверку:  
содержит ли список sections «***новую голову змеи***«.

**Код для этого будет выглядеть примерно так:**  
if (sections.contains(head))

При этом head должен быть еще не добавлен в список sections, иначе будет всегда **true**.  
Но чтобы этот код работал, надо реализовать методы сравнения объектов (equals и hashCode) в классе **SnakeSection**.

**Подсказка:**  
ИспользуйAlt+Insertв***Intellij IDEA***для автоматической генерации методовequalsиhashCode.

**Задание:**  
а) реализуй методы equals и hashCode в классе SnakeSection.  
б) реализуй метод checkBorders(SnakeSection **head**): если голова змеи за границами комнаты — змея умирает (isAlive = **false**)  
в) реализуй метод checkBody(SnakeSection **head**): если голова змеи пересекается с ее телом — змея умирает (isAlive = **false**)