package com.javarush.task.task24.task2401;  
  
/\*   
  
Создание своего интерфейса-маркера  
1. Создай интерфейс-маркер SelfInterfaceMarker.  
2. Создай класс SelfInterfaceMarkerImpl, который реализует SelfInterfaceMarker.  
3. Добавь в SelfInterfaceMarkerImpl минимум 2 любых public метода.  
4. Создай исключение UnsupportedInterfaceMarkerException, унаследуй его от класса Exception.  
5. В методе testClass класса Util: если параметр == null, то выбросьте UnsupportedInterfaceMarkerException.  
\*/  
public class Solution {  
 public static void main(String[] args) throws UnsupportedInterfaceMarkerException {  
 SelfInterfaceMarkerImpl obj = new SelfInterfaceMarkerImpl();  
 Util.*testClass*(obj);  
 }  
  
}

package com.javarush.task.task24.task2401;  
  
*/\*\*  
 \* Created by nik on 21.03.2017.  
 \*/*public interface SelfInterfaceMarker {  
}

package com.javarush.task.task24.task2401;  
  
*/\*\*  
 \* Created by nik on 21.03.2017.  
 \*/*public class SelfInterfaceMarkerImpl implements SelfInterfaceMarker{  
 public void job(){  
  
 }  
 public void work(){}  
}

package com.javarush.task.task24.task2401;  
  
*/\*\*  
 \* Created by nik on 21.03.2017.  
 \*/*public class UnsupportedInterfaceMarkerException extends Exception {  
}

package com.javarush.task.task24.task2401;  
  
import java.lang.reflect.Method;  
  
public class Util {  
 //пример того, как можно использовать интерфейс-маркер  
 //this method is available only for a SelfInterfaceMarker implementation  
 public static void testClass(SelfInterfaceMarker interfaceMarker) throws UnsupportedInterfaceMarkerException {  
 if (interfaceMarker==null){  
 throw new UnsupportedInterfaceMarkerException();  
 }  
 for (Method method : interfaceMarker.getClass().getDeclaredMethods()) {  
 System.*out*.println(method);  
 }  
 }  
  
}

package com.javarush.task.task24.task2402;  
  
/\*   
Cloneable  
Добавьте java-код, чтобы метод main отработал без исключений.  
\*/  
public class Solution {  
  
 public static void main(String[] args) throws CloneNotSupportedException {  
 Test1 test1 = new Test1();  
 test1.clone();  
  
 Test2 test2 = new Test2();  
 test2.clone(new Test2());  
  
 Test3 test3 = new Test3();  
 test3.c1one();  
  
 Test4 test4 = new Test4();  
 test4.clone();  
 }  
  
  
 public static class Test1 {  
 protected Test1 clone() throws CloneNotSupportedException {  
 return (Test1)super.clone();  
 }  
 }  
  
 public static class Test2 extends Test1 {  
 public Test2 clone(Test2 test2) throws CloneNotSupportedException {  
 return (Test2)test2.clone();  
 }  
 }  
  
 public static class Test3 {  
 protected Object c1one() throws CloneNotSupportedException {  
 return new Test3();  
 }  
 }  
  
 public static class Test4 extends Test3 {  
 protected Object c1one() throws CloneNotSupportedException {  
 return super.c1one();  
 }  
 }  
}

package com.javarush.task.task24.task2403;  
  
import java.io.Serializable;  
import java.rmi.Remote;  
import java.rmi.server.RemoteObject;  
import java.util.EventListener;  
  
/\*   
Так-с... сопоставим  
  
Исправь ошибки: перемести методы clone в те классы, в которых они должны быть реализованы.  
Лишние методы удали.  
Не удаляй метод main.  
\*/  
public class Solution {  
 public static class A implements Serializable {  
 }  
  
 public static class B implements Remote {  
 }  
  
 public static class C extends ArrayDeque {  
 }  
  
 public static class D implements EventListener {  
 }  
  
 protected A clone() throws CloneNotSupportedException {  
 return (A) super.clone();  
 }  
  
 protected Thread clone() throws CloneNotSupportedException {  
 return (Thread) super.clone();  
 }  
  
 protected final B clone() throws CloneNotSupportedException {  
 return (B) super.clone();  
 }  
  
 protected ArrayDeque clone() throws CloneNotSupportedException {  
 return super.clone();  
 }  
  
 public C clone() {  
 return (C) super.clone();  
 }  
  
 protected D clone() throws CloneNotSupportedException {  
 return (D) super.clone();  
 }  
  
 public static void main(String[] args) {  
  
 }  
}

package com.javarush.task.task24.task2404;  
  
/\*   
Рефакторинг Rectangle  
  
В классе Rectangle:  
1. Измени методы getHeight и getWidth, чтобы они возвращали объекты типов HasHeight и HasWidth соответственно.  
2. Для этого внутри методов getHeight и getWidth создай локальные классы — реализации интерфейсов.  
3. Переименуй getHeight в castToHasHeight, getWidth в castToHasWidth (на имени метода нажми Shift+F6).  
4. Убери наследование интерфейсов в классе Rectangle.  
  
P.S. Ожидается, что после внесения требуемых изменений, закомментированный код в методе станет рабочим и должен быть раскомментирован.  
\*/  
public class Solution {  
 public static void main(String[] args) {  
 Rectangle rectangle = new Rectangle(1, 2, 3, 4);  
 System.*out*.println(*getHeight*(rectangle));  
 System.*out*.println(*getWidth*(rectangle));  
 /////////////////////expected//////////////////  
 //System.out.println(getHeight(rectangle.castToHasHeight()));  
 //System.out.println(getWidth(rectangle.castToHasWidth()));  
 }  
  
 public static double getHeight(HasHeight rectangle) {  
 return rectangle.getHeight();  
 }  
  
 public static double getWidth(HasWidth rectangle) {  
 return rectangle.getWidth();  
 }  
  
  
 public static class Rectangle implements HasHeight, HasWidth{  
 private Point point1;  
 private Point point2;  
  
 public Rectangle(double x1, double y1, double x2, double y2) {  
 point1 = new Point(x1, y1);  
 point2 = new Point(x2, y2);  
 }  
  
 public double getHeight() {  
 return Math.*abs*(point1.getY() - point2.getY());  
 }  
  
 public double getWidth() {  
 return Math.*abs*(point1.getX() - point2.getX());  
 }  
 }  
}

package com.javarush.task.task24.task2404;  
  
public interface HasHeight {  
 double getHeight();  
}

package com.javarush.task.task24.task2404;  
  
public class Point {  
 private double x;  
 private double y;  
  
 public Point(double x, double y) {  
 this.x = x;  
 this.y = y;  
 }  
  
 public double getX() {  
 return x;  
 }  
  
 public double getY() {  
 return y;  
 }  
}

package com.javarush.task.task24.task2404;  
  
public interface HasWidth {  
 double getWidth();  
}

package com.javarush.task.task24.task2405;  
  
/\*   
Black box  
1. Восстанови логику метода someAction для поля solutionAction.  
2. Пример вывода смотри в комментарии к методу main.  
3. Подсказка: метод someAction анонимного класса поля solutionAction должен вызвать метод сабкласса FirstClass, если param > 0, иначе вызвать метод сабкласса SecondClass.  
  
Не изменяй метод main!  
  
\*/  
public class Solution implements Action {  
 public static int *countActionObjects*;  
  
 private int param;  
  
 private Action solutionAction = new Action() {  
 //!!!!! Changes can be here  
 //!!!!! Изменения могут быть тут  
  
 public void someAction() {  
 //!!!!! All changes have to be here  
 //!!!!! Все изменения должны быть только тут  
 }  
 };  
  
  
 public Solution(int param) {  
 this.param = param;  
 }  
  
 @Override  
 public void someAction() {  
 solutionAction.someAction();  
 }  
  
 */\*\*  
 \* 5  
 \* 4  
 \* 3  
 \* 2  
 \* 1  
 \* class FirstClass, method someAction  
 \* class SecondClass, method someAction  
 \* Specific action for anonymous SecondClass, param = 0  
 \* Count of created Action objects is 2  
 \* class SecondClass, method someAction  
 \* Specific action for anonymous SecondClass, param = -1  
 \* Count of created Action objects is 3  
 \*/* public static void main(String[] args) {  
 Solution solution = new Solution(5);  
 solution.someAction();  
 System.*out*.println("Count of created Action objects is " + *countActionObjects*);  
  
 solution = new Solution(-1);  
 solution.someAction();  
 System.*out*.println("Count of created Action objects is " + *countActionObjects*);  
 }  
}

package com.javarush.task.task24.task2405;  
  
public interface Action {  
 void someAction();  
}

package com.javarush.task.task24.task2405;  
  
public abstract class FirstClass implements Action { //first implementation  
  
 protected FirstClass() {  
 Solution.*countActionObjects*++;  
 }  
  
 public void someAction() {  
 System.*out*.println("class FirstClass, method someAction");  
 }  
  
 public abstract Action getDependantAction();  
}

package com.javarush.task.task24.task2405;  
  
public class SecondClass implements Action { //second implementation  
 public static String *SPECIFIC\_ACTION\_FOR\_ANONYMOUS\_SECOND\_CLASS\_PARAM* = "\nSpecific action for anonymous SecondClass, param = ";  
  
 private static String *SECOND\_CLASS\_METHOD\_STRING* = "class SecondClass, method someAction";  
  
 public SecondClass() {  
 Solution.*countActionObjects*++;  
 }  
 protected StringBuilder sb = new StringBuilder(*SECOND\_CLASS\_METHOD\_STRING*);  
  
 public void someAction() {  
 System.*out*.println(sb.toString());  
 }  
}

package com.javarush.task.task24.task2406;  
  
import java.math.BigDecimal;  
  
/\*   
Наследование от внутреннего класса  
Внутри класса Solution создай 2 внутренних public класса Apt3Bedroom, BigHall.  
Унаследуй их от Apartments и Hall.  
\*/  
public class Solution {  
 public class Building {  
 public class Hall {  
 private BigDecimal square;  
  
 public Hall(BigDecimal square) {  
 this.square = square;  
 }  
 }  
  
 public class Apartments {  
 }  
 }  
  
 public static void main(String[] args) {  
  
 }  
}

package com.javarush.task.task24.task2407;  
  
import java.util.List;  
  
/\*   
Реализация интерфейса используя локальный класс  
В классе Cat реализуй логику метода toSayable, которая описана в джавадоке.  
\*/  
public class Solution {  
 public static void main(String[] args) {  
 List<Pet> pet = Util.*getPets*();  
 List<Sayable> pets = Util.*convertPetToSayable*(pet);  
 Util.*printDialog*(pets);  
 }  
}

package com.javarush.task.task24.task2407;  
  
/\*  
В работе вам иногда будет нужно закастить класс к какому-нибудь интерфейсу (тут Sayable),  
который не реализован в текущем классе  
 \*/  
public class Cat implements Pet {  
 private String name;  
  
 public Cat(String name) {  
 this.name = name;  
 }  
  
 */\*\*  
 \* Это - механизм адаптирования к другому интерфейсу - Sayable  
 \* Внутри метода toSayable создайте class CatPet, реализующий интерфейс Sayable  
 \* Логика метода say:  
 \* Если i <= 0, то вывести на экран, что кот спит. Пример, "Васька спит."'  
 \* Иначе вывести фразу: "имя\_кота говорит мяу!". Пример для i=3, "Васька говорит мяяяу!"  
 \* <p/>  
 \* <b>Пример вывода:</b>  
 \* Мурзик спит.  
 \* Васька говорит мяяу!  
 \* Кошка говорит мяяяяяу!  
 \* Мышь пищит.  
 \* Томас говорит мяу!  
 \* <p/>  
 \** ***@param*** *i количество букв 'я' в слове мяу  
 \** ***@return*** *экземпляр класса CatPet  
 \*/* public Sayable toSayable(final int i) {  
 return null;  
 }  
}

package com.javarush.task.task24.task2407;  
  
/\*  
обратите внимание, как именно Mouse отличается от Cat  
Этот класс - привычный для вас.  
\*/  
public class Mouse implements Pet, Sayable {  
 @Override  
 public Sayable toSayable(int i) {  
 return this;  
 }  
  
 @Override  
 public String say() {  
 return "Мышь пищит.";  
 }  
}

package com.javarush.task.task24.task2407;  
  
public interface Pet {  
 public Sayable toSayable(int i);  
}

package com.javarush.task.task24.task2407;  
  
public interface Sayable {  
 String say();  
}

package com.javarush.task.task24.task2407;  
  
import java.util.LinkedList;  
import java.util.List;  
  
public class Util {  
 //Util отлично работает со всеми классами Cat, Mouse. А ведь Mouse отличается от Cat.  
 public static void printDialog(List<Sayable> pets) {  
 for (int i = 0; i < pets.size(); i++) {  
 System.*out*.println(pets.get(i).say());  
 }  
 }  
  
 public static List<Pet> getPets() {  
 List<Pet> pets = new LinkedList<>();  
 pets.add(new Cat("Мурзик"));  
 pets.add(new Cat("Васька"));  
 pets.add(new Cat("Кошка"));  
 pets.add(new Mouse());  
 pets.add(new Cat("Томас"));  
 return pets;  
 }  
  
 public static List<Sayable> convertPetToSayable(List<Pet> pets) {  
 List<Sayable> result = new LinkedList<>();  
 for (Pet pet : pets) {  
 int i = (int) (Math.*random*() \* 7) - 2;  
 result.add(pet.toSayable(i));  
 }  
 return result;  
 }  
}

package com.javarush.task.task24.task2407;  
  
import java.util.List;  
  
/\*   
Реализация интерфейса используя локальный класс  
В классе Cat реализуй логику метода toSayable, которая описана в джавадоке.  
\*/  
public class Solution {  
 public static void main(String[] args) {  
 List<Pet> pet = Util.*getPets*();  
 List<Sayable> pets = Util.*convertPetToSayable*(pet);  
 Util.*printDialog*(pets);  
 }  
}

package com.javarush.task.task24.task2407;  
  
/\*  
В работе вам иногда будет нужно закастить класс к какому-нибудь интерфейсу (тут Sayable),  
который не реализован в текущем классе  
 \*/  
public class Cat implements Pet {  
 private String name;  
  
 public Cat(String name) {  
 this.name = name;  
 }  
  
 */\*\*  
 \* Это - механизм адаптирования к другому интерфейсу - Sayable  
 \* Внутри метода toSayable создайте class CatPet, реализующий интерфейс Sayable  
 \* Логика метода say:  
 \* Если i <= 0, то вывести на экран, что кот спит. Пример, "Васька спит."'  
 \* Иначе вывести фразу: "имя\_кота говорит мяу!". Пример для i=3, "Васька говорит мяяяу!"  
 \* <p/>  
 \* <b>Пример вывода:</b>  
 \* Мурзик спит.  
 \* Васька говорит мяяу!  
 \* Кошка говорит мяяяяяу!  
 \* Мышь пищит.  
 \* Томас говорит мяу!  
 \* <p/>  
 \** ***@param*** *i количество букв 'я' в слове мяу  
 \** ***@return*** *экземпляр класса CatPet  
 \*/* public Sayable toSayable(final int i) {  
 return null;  
 }  
}

package com.javarush.task.task24.task2407;  
  
/\*  
обратите внимание, как именно Mouse отличается от Cat  
Этот класс - привычный для вас.  
\*/  
public class Mouse implements Pet, Sayable {  
 @Override  
 public Sayable toSayable(int i) {  
 return this;  
 }  
  
 @Override  
 public String say() {  
 return "Мышь пищит.";  
 }  
}

package com.javarush.task.task24.task2407;  
  
public interface Sayable {  
 String say();  
}

package com.javarush.task.task24.task2407;  
  
import java.util.LinkedList;  
import java.util.List;  
  
public class Util {  
 //Util отлично работает со всеми классами Cat, Mouse. А ведь Mouse отличается от Cat.  
 public static void printDialog(List<Sayable> pets) {  
 for (int i = 0; i < pets.size(); i++) {  
 System.*out*.println(pets.get(i).say());  
 }  
 }  
  
 public static List<Pet> getPets() {  
 List<Pet> pets = new LinkedList<>();  
 pets.add(new Cat("Мурзик"));  
 pets.add(new Cat("Васька"));  
 pets.add(new Cat("Кошка"));  
 pets.add(new Mouse());  
 pets.add(new Cat("Томас"));  
 return pets;  
 }  
  
 public static List<Sayable> convertPetToSayable(List<Pet> pets) {  
 List<Sayable> result = new LinkedList<>();  
 for (Pet pet : pets) {  
 int i = (int) (Math.*random*() \* 7) - 2;  
 result.add(pet.toSayable(i));  
 }  
 return result;  
 }  
}

package com.javarush.task.task24.task2408;  
  
import java.util.List;  
  
/\*   
Как избежать Copy+Paste  
В классе Dog реализуй логику метода toSayable, которая описана в джавадоке.  
\*/  
public class Solution {  
 public static void main(String[] args) {  
 List<Pet> pet = Util.*getPets*();  
 List<Sayable> pets = Util.*convertPetToSayable*(pet);  
 Util.*printDialog*(pets);  
 }  
}

package com.javarush.task.task24.task2408;  
  
/\*  
В работе вам иногда будет нужно закастить класс к какому-нибудь другому классу, не интерфейсу :)))  
Класс DogPet использует 2 класса - SuperDog и Dog, разберись с getName в классе DogPet  
Так намного лучше, чем Copy+Paste!  
 \*/  
public class Dog implements Pet {  
 private String name;  
  
 public Dog(String name) {  
 this.name = name;  
 }  
  
 */\*\*  
 \* Если так получилось, что есть готовый класс А (тут SuperDog) с логикой, которую вы хотите использовать.  
 \* То возможны 3 способа:  
 \* 1) из класса А скопировать логику себе (это плохо, т.к. поддерживать несколько копий одного и того же кода проблематично)  
 \* 2) создать экземпляр класса А внутри нашего класса и использовать его, связь has-a (не всегда подходит, т.к. класс А сам по себе)  
 \* 3) создать внутренний класс, который наследуется от А, использовать его методы вместе с  
 \* методами и полями нашего класса, т.к. внутренний класс имеет доступ к своему родителю как наследник,  
 \* а также ко всем полям и методам включая private того класса, в который он вложен.  
 \* <p/>  
 \* Итак, разбираемся с п.3:  
 \* Внутри метода toSayable создайте class DogPet, который наследуется от SuperDog и реализует интерфейс Sayable  
 \* создайте метод private String getName(), который будет использовать логику двух классов - Dog и SuperDog.  
 \* Пусть этот метод возвращает имя собаки (класс Dog), которое с обеих сторон выделено getSuperQuotes (класс SuperDog)  
 \* Пример, " \*\*\* Барбос \*\*\* "  
 \* Логика метода say:  
 \* Если i < 1, то используя метод getName вывести на экран, что собака спит. Пример, " \*\*\* Шарик \*\*\* спит."'  
 \* Иначе вывести фразу: "имя\_собаки лает гааав! сегодняшняя\_дата". Пример для i=3, " \*\*\* Тузик \*\*\* лает гааав! 13-ноя-2013 Ср"  
 \* Для форматирования даты используйте formatter из класса SuperDog.  
 \* <p/>  
 \* <b>Пример вывода:</b>  
 \* \*\*\* Барбос \*\*\* лает гааааав! 13-ноя-2013 Ср  
 \* \*\*\* Тузик \*\*\* лает гаав! 13-ноя-2013 Ср  
 \* \*\*\* Бобик \*\*\* лает гааав! 13-ноя-2013 Ср  
 \* Мыша пищит.  
 \* \*\*\* Шарик \*\*\* спит.  
 \*  
 \** ***@param*** *i количество букв 'а' в слове гав  
 \** ***@return*** *экземпляр класса DogPet  
 \*/* public Sayable toSayable(final int i) {  
 return null;  
 }  
}

package com.javarush.task.task24.task2408;  
  
/\*  
обратите внимание, как именно Mouse отличается от Dog  
Этот класс - привычный для вас.  
\*/  
public class Mouse implements Pet, Sayable {  
 @Override  
 public Sayable toSayable(int i) {  
 return this;  
 }  
  
 @Override  
 public String say() {  
 return "Мышь пищит.";  
 }  
}

package com.javarush.task.task24.task2408;  
  
public interface Pet {  
 public Sayable toSayable(int i);  
}

package com.javarush.task.task24.task2408;  
  
public interface Sayable {  
 String say();  
}

package com.javarush.task.task24.task2408;  
  
import java.text.SimpleDateFormat;  
  
public abstract class SuperDog {  
 protected String getSuperQuotes() {  
 //some logic here  
 return " \*\*\* ";  
 }  
  
 protected SimpleDateFormat formatter = new SimpleDateFormat("dd-MMM-yyyy EEE");  
}

package com.javarush.task.task24.task2408;  
  
import java.util.LinkedList;  
import java.util.List;  
  
public class Util {  
 //Util отлично работает с классами Dog, Mouse.  
 public static void printDialog(List<Sayable> pets) {  
 for (int i = 0; i < pets.size(); i++) {  
 System.*out*.println(pets.get(i).say());  
 }  
 }  
  
 public static List<Pet> getPets() {  
 List<Pet> pets = new LinkedList<>();  
 pets.add(new Dog("Барбос"));  
 pets.add(new Dog("Тузик"));  
 pets.add(new Dog("Бобик"));  
 pets.add(new Mouse());  
 pets.add(new Dog("Шарик"));  
 return pets;  
 }  
  
 public static List<Sayable> convertPetToSayable(List<Pet> pets) {  
 List<Sayable> result = new LinkedList<>();  
 for (Pet pet : pets) {  
 int i = (int) (Math.*random*() \* 7) - 2;  
 result.add(pet.toSayable(i));  
 }  
 return result;  
 }  
}

package com.javarush.task.task24.task2409;  
  
/\*   
Интернет-магазин продажи джинсов  
1. Создай 2 интерфейса в отдельных файлах:  
1.1) Item с методами int getId(), double getPrice(), String getTM()  
1.2) Jeans extends Item с методами int getLength() и int getSize()  
  
2. В классе Util в методе getAllJeans добавь пропущенную часть java-кода:  
2.1) разберись в том, что уже есть в методе getAllJeans класса Util  
2.2) создай абстрактный class AbstractJeans от интерфейса Jeans с одним абстрактным методом, реализуй остальные методы  
2.3) создай классы Levis и Denim от AbstractJeans, реализуй оставшийся метод  
2.4) в классе AbstractJeans реализуй метод toString() (можешь воспользоваться Alt+Insert -> toString())  
2.5) метод toString класса AbstractJeans должен начинаться с имени сабкласса, например, Levis{id=1, length=34, size=6, price=150.0}  
\*/  
public class Solution {  
  
 public static List<Jeans> *allJeans* = Util.*getAllJeans*();  
  
 public static void main(String[] args) {  
 for (Jeans jeans : *allJeans*) {  
 System.*out*.println(jeans);  
 }  
 }  
  
}

package com.javarush.task.task24.task2409;  
  
*/\*\*  
 \* Created by nik on 27.03.2017.  
 \*/*public class Item {  
}

package com.javarush.task.task24.task2410;  
  
import java.util.LinkedList;  
import java.util.List;  
  
/\*   
Рефакторинг, анонимные классы  
Рефакторинг, анонимные классы  
Сделайте так, чтобы метод getIterator возвращал анонимный класс.  
Перенесите логику LocalIterator в анонимный класс.  
Меняйте только тело метода getIterator.  
\*/  
public class Solution {  
 public static List<Iterator> *iterators* = new LinkedList<>();  
  
 private int countItems;  
  
 public Iterator getIterator(final String name) {  
 class LocalIterator implements Iterator {  
 public LocalIterator() {  
 countItems++;  
 System.*out*.println(name + " item " + countItems);  
 }  
  
 public Iterator next() {  
 return new LocalIterator();  
 }  
 }  
 return new LocalIterator();  
 }  
  
 public static void main(String[] args) {  
 Solution solution = new Solution();  
  
 Iterator iterator = solution.getIterator("iterator");  
 for (int i = 1; i < 5; i++) {  
 *iterators*.add(iterator.next());  
 }  
 }  
}

package com.javarush.task.task24.task2410;  
  
public interface Iterator {  
 Iterator next();  
}

package com.javarush.task.task24.task2411;  
  
/\*   
Вспомним наследование  
Исправить наследование во всех классах, чтобы вывод метода main был CCBAYS.  
\*/  
public class Solution {  
 private class A {  
 protected String value = "A";  
  
 public A() {  
 System.*out*.print(value);  
 }  
 }  
  
 private A a = new A() {  
 { //у анонимных классов нет своих конструкторов, но что-то можно сделать в блоке инициализации класса  
 value = "Y";  
 if (super.getClass().getName().contains(".Solution$")) {  
 System.*out*.print(value);  
 }  
 }  
 };  
  
 public Solution() {  
 System.*out*.print("S");  
 }  
  
 public static void main(String[] args) {  
 new Solution();  
 }  
}  
  
package com.javarush.task.task24.task2411;  
  
public interface JustAnInterface {  
 public static final B *B* = new B();  
  
 class B {  
 public B() {  
 System.*out*.print("B");  
 }  
 }  
}

package com.javarush.task.task24.task2411;  
  
public class C {  
 public C() {  
 System.*out*.print("C");  
 B localB = B;  
 }  
}

package com.javarush.task.task24.task2412;  
  
import java.text.ChoiceFormat;  
import java.text.Format;  
import java.text.MessageFormat;  
import java.text.SimpleDateFormat;  
import java.util.\*;  
  
/\*   
Знания - сила!  
1. В методе sort написать компаратор для Stock:  
1.1. Первичная сортировка по name в алфавитном порядке  
1.2. Вторичная сортировка по дате без учета часов, минут, секунд (сверху самые новые), потом по прибыли от положительных к отрицательным  
  
… open 125,64 and last 126,74 — тут прибыль = 126,74-125,64  
… open 125,64 and last 123,43 — тут прибыль = 123,43-125,64  
  
2. Разобраться с \*Format-ами и исправить IllegalArgumentException.  
Подсказка — это одна строчка.  
  
Пример вывода:  
Fake Apple Inc. AAPL | 17-11-2025 open 125,64 and last 123,43  
Fake Oracle Corporation ORCL | 21-08-1989 closed 0,15  
\*/  
public class Solution {  
 public static void main(String[] args) {  
 List<Stock> stocks = *getStocks*();  
 *sort*(stocks);  
 Date actualDate = new Date();  
 *printStocks*(stocks, actualDate);  
 }  
  
 public static void printStocks(List<Stock> stocks, Date actualDate) {  
 SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy");  
  
 double[] filelimits = {0d, actualDate.getTime()};  
 String[] filepart = {"closed {4}", "open {2} and last {3}"};  
  
 ChoiceFormat fileform = new ChoiceFormat(filelimits, filepart);  
 Format[] testFormats = {null, null, dateFormat, fileform};  
 MessageFormat pattform = new MessageFormat("{0} {1} | {5} {6}");  
 pattform.setFormats(testFormats);  
  
 for (Stock stock : stocks) {  
 String name = ((String) stock.get("name"));  
 String symbol = (String) stock.get("symbol");  
 double open = !stock.containsKey("open") ? 0 : ((double) stock.get("open"));  
 double last = !stock.containsKey("last") ? 0 : ((double) stock.get("last"));  
 double change = !stock.containsKey("change") ? 0 : ((double) stock.get("change"));  
 Date date = (Date) stock.get("date");  
 Object[] testArgs = {name, symbol, open, last, change, date, date.getTime()};  
 System.*out*.println(pattform.format(testArgs));  
 }  
 }  
  
 public static void sort(List<Stock> list) {  
 Collections.*sort*(list, new Comparator<Stock>() {  
 public int compare(Stock stock1, Stock stock2) {  
 return 0;  
 }  
 });  
 }  
  
 public static class Stock extends HashMap {  
 public Stock(String name, String symbol, double open, double last) {  
 put("name", name);  
 put("symbol", symbol);  
 put("open", open);  
 put("last", last);  
 put("date", *getRandomDate*(2020));  
 }  
  
 public Stock(String name, String symbol, double change, Date date) {  
 put("name", name);  
 put("symbol", symbol);  
 put("date", date);  
 put("change", change);  
 }  
 }  
  
 public static List<Stock> getStocks() {  
 List<Stock> stocks = new ArrayList();  
  
 stocks.add(new Stock("Fake Apple Inc.", "AAPL", 125.64, 123.43));  
 stocks.add(new Stock("Fake Cisco Systems, Inc.", "CSCO", 25.84, 26.3));  
 stocks.add(new Stock("Fake Google Inc.", "GOOG", 516.2, 512.6));  
 stocks.add(new Stock("Fake Intel Corporation", "INTC", 21.36, 21.53));  
 stocks.add(new Stock("Fake Level 3 Communications, Inc.", "LVLT", 5.55, 5.54));  
 stocks.add(new Stock("Fake Microsoft Corporation", "MSFT", 29.56, 29.72));  
  
 stocks.add(new Stock("Fake Nokia Corporation (ADR)", "NOK", .1, *getRandomDate*()));  
 stocks.add(new Stock("Fake Oracle Corporation", "ORCL", .15, *getRandomDate*()));  
 stocks.add(new Stock("Fake Starbucks Corporation", "SBUX", .03, *getRandomDate*()));  
 stocks.add(new Stock("Fake Yahoo! Inc.", "YHOO", .32, *getRandomDate*()));  
 stocks.add(new Stock("Fake Applied Materials, Inc.", "AMAT", .26, *getRandomDate*()));  
 stocks.add(new Stock("Fake Comcast Corporation", "CMCSA", .5, *getRandomDate*()));  
 stocks.add(new Stock("Fake Sirius Satellite", "SIRI", -.03, *getRandomDate*()));  
  
 return stocks;  
 }  
  
 public static Date getRandomDate() {  
 return *getRandomDate*(1970);  
 }  
  
 public static Date getRandomDate(int startYear) {  
 int year = startYear + (int) (Math.*random*() \* 30);  
 int month = (int) (Math.*random*() \* 12);  
 int day = (int) (Math.*random*() \* 28);  
 GregorianCalendar calendar = new GregorianCalendar(year, month, day);  
 return calendar.getTime();  
 }  
}

MVC (1)

Привет! Эта задача будет на паттерн **MVC** — **Model-View-Controller**.  
Мы вместе построим архитектуру используя **MVC**. Разберись подробно, что и почему нужно реализовывать так, как я тебе покажу.  
Прочти дополнительную литературу, которую дает профессор в конце уровня.  
Тебя, скорее всего, на собеседовании спросят об этом паттерне либо дадут задание, в котором нужно будет его реализовать.

Итак…  
У тебя есть два пакета: bean, содержащий единственный класс **User**, и **dao**,  
в котором хранится эмуляция базы данных в пакете **mock** и **UserDao**. **UserDao** — это уровень **ДАО**, т.е. уровень доступа к базе.  
В нем размещают различные методы по **сохранению** и **получению** объектов из базы данных.  
В реальном приложении строку private DataSource dataSource = DataSource.getInstance() не встретить.  
Я реализовал **DataSource** в виде синглтона. В действительности, у тебя будет что-то такое:  
@Autowired  
private DataSource dataSource;

Фреймворк, которым ты будешь пользоваться, сам создаст объект базы данных и инициализирует поле dataSource.

Запомни, с **ДАО** уровнем работают се рвисы. Никакие другие классы в **ДАО** не лезут. В сервисах описана бизнес логика.  
Сервисы забирают данные из базы используя **ДАО**, обрабатывают их и отдают тому, кто данные запросил.  
Однако не все данные хранятся в базе. Например, залогиненый пользователь будет храниться в специальном объекте — Модели.  
Объект, который содержит в себе данные, необходимые для отображения информации на клиенте, называется Моделью.  
Также этот объект Модель содержит ссылки на все необходимые сервисы.  
Если данных для отображения очень много, то их выделяют в отдельный объект.

Напишем приложение, которое будет показывать список пользователей и что-то делать с ними, например, обновлять их данные и удалять.

1. Создай пакет **model**, в котором создай класс **ModelData**.  
**ModelData** — это объект, который будет хранить необходимые данные для отображения на клиенте.  
Создай поле с геттером и сеттером **List**<User> users — это будет список пользователей для отображения.

2. Используя любую модель должна быть возможность получить все необходимые данные для отображения. Поэтому в пакете **model** создай интерфейс **Model**, который должен содержать метод ModelData getModelData().

3. В пакете model создай класс **FakeModel**, реализующий **Model**. Он нам понадобится на начальном этапе.  
В нем создай поле **ModelData** modelData, которое инициализируй объектом.

4. В интерфейсе **Model** создай метод void loadUsers().  
Реализуй его в FakeModel: инициализируй список пользователей **modelData** любыми данными. Они не влияют на тестирование.

**У меня такие данные:**  
User{name=’A’, id=1, level=1}  
User{name=’B’, id=2, level=1}

Думаю, ты помнишь, что все методы интерфейса являются **public**-ами, поэтому модификатор указывать не нужно.  
Программисты часто мОкают данные на начальном этапе. Получение реальных данных реализуется на последних этапах.  
Мокать — это подменять реальные объекты на хардкоженные, тестовые данные.

### MVC (2)

1. Создай пакет **controller**, в котором создай класс **Controller**.  
Этот класс будет получать запрос от клиента, оповещать Модель об этом, а Модель, в свою очередь, будет обновлять **ModelData**.

2. Добавь в контроллер поле Model model вместе с сеттером.

3. В контроллере создай публичный метод void onShowAllUsers(), который должен обратиться к модели и инициировать загрузку пользователей.

4. Создай пакет **view**. В нем создай интерфейс **View**.

5. В интерфейс **View** добавь два метода: void refresh(ModelData modelData) и void setController(Controller controller)

**Требования:**  
1. Класс Controller должен быть создан в пакете controller.  
2. В классе Controller должно быть создано приватное поле Model model и сеттер для этого поля.  
3. В классе Controller должен быть создан публичный метод void onShowAllUsers(), в котором у модели должен вызываться метод loadUsers().  
4. Интерфейс View должен быть создан в пакете view.  
5. В интерфейсе View должны быть объявлены два метода: void refresh(ModelData modelData) и void setController(Controller controller).

### MVC (3)

Чтобы понимать, в правильном ли направлении ты движешься, тебе надо видеть данные. Поэтому:  
1. В пакете **view** создай класс **UsersView**, реализующий **View**. Он будет отображать список пользователей в консоль.

2. В **UsersView** создай поле-контроллер, также создай ему сеттер.

3. Реализуй логику метода refresh:  
3.1. Выведи в консоль фразу «**All users:**«.  
3.2. Выведи в консоль всех пользователей, которые есть в **modelData**.  
Перед каждым пользователем сделай отступ в виде табуляции.  
3.3. В конце выведи визуальный разделитель данных  
==================================================

4. Уже интересно посмотреть, что же получилось.  
Добавь в **UsersView** публичный метод void fireEventShowAllUsers(), который будет эмулировать событие клиента.  
Обратись к контроллеру и вызови его нужный метод для отображения всех пользователей.

5. Класс **Solution** будет эмулятором пользователя. Открой класс **Solution**, стань на красный метод,  
с помощью горячих клавиш Идеи создай проперти(**поле**) для **usersView**.  
Нужен только сеттер. Если у тебя создался геттер, то удали его.

6. Запусти main. Упс, ничего не вывело : (  
Это получилось потому, что данные пришли с сервера, обновились в **ModelData**, но Вью ничего о них не знает.  
Вью сама не умеет себя обновлять. Это делает Контроллер.  
Пойди в контроллер и добавь обновление данных во Вью.  
Напомню, данные хранятся в Модели.

7. Запусти main. У меня теперь такой вывод:  
All users:  
User{name='A', id=1, level=1}  
User{name='B', id=2, level=1}  
==================================================  
Ура, идем дальше.

**Требования:**  
1. Класс UsersView, реализующий интерфейс View, должен быть создан в пакете view.  
2. В классе UsersView должно быть создано приватное поле Controller controller и сеттер для этого поля.  
3. Метод refresh класса UsersView должен быть реализован согласно условию.  
4. В классе UsersView должен быть создан публичный метод void fireEventShowAllUsers(), в котором у контроллера должен вызываться метод onShowAllUsers().  
5. В классе Controller должно быть создано приватное поле UsersView usersView и сеттер для этого поля. Геттера для этого поля не должно быть создано.  
6. В методе onShowAllUsers() класса Controller должен вызываться метод refresh у объекта usersView с параметром model.getModelData() после вызова метода loadUsers().  
7. Вызов метода main должен выводить на экран информацию о всех пользователях, которых ты добавляешь в методе loadUsers() класса FakeModel.

### MVC (4)

Пора заменять нашу фейковую Модель на реальную, которая будет получать данные из **DataSource**.  
В пакет **model.service** я добавил сервис для работы с пользователями.  
Также в корне этой задачи ты найдешь утильный класс **Util**.

1. Аналогично **FakeModel** создай модель **MainModel**.

2. Т.к. Модель обращается к сервисам, то в **MainModel** создай поле UserService userService, инициализируй объектом.

3. Реализуй логику метода loadUsers:  
3.1. Достань всех пользователей между **1** и **100** уровнями. (Метод getUsersBetweenLevels(1, 100)).  
3.2. Положи всех пользователей в **modelData**.

4. Обнови **Solution.main**: замени **FakeModel** на **MainModel**.  
Преимущества **MVC** в том, что в любой момент времени легко можно заменить любую часть паттерна.

**Требования:**  
1. Класс MainModel, реализующий интерфейс Model, должен быть создан в пакете model.  
2. В классе MainModel должно быть создано и инициализировано приватное поле ModelData modelData.  
3. В классе MainModel должно быть создано и инициализировано приватное поле UserService userService.  
4. В классе MainModel необходимо реализовать метод getModelData.  
5. Метод loadUsers класса MainModel должен быть реализован согласно условию.  
6. В методе main класса Solution должен создаваться объект класса MainModel вместо объекта класса FakeModel.

### MVC (5)

В сервисе есть метод, который возвращает всех удаленных пользователей. Давай их отобразим.

1. Распредели методы по классам **MVC**:

public void fireEventShowDeletedUsers() {  
…onShowAllDeletedUsers();  
}

public void onShowAllDeletedUsers() {  
…loadDeletedUsers();  
}

public void loadDeletedUsers() {  
List<User> users = userService.getAllDeletedUsers();  
}

Не забудь, что данные, полученные с сервера, необходимо положить в **ModelData**. А потом обновить view.  
Добавь это самостоятельно в нужные методы.

2. Добавь в **Solution.main** вызов нового метода, который ты поместил в view.

3. Добавь в интерфейс Model метод, который ты поместил в **Модель**, реализуй его в **FakeModel**: выброси UnsupportedOperationException.

**Требования:**  
1. Необходимо определить правильное расположение метода fireEventShowDeletedUsers() и реализовать этот метод.  
2. Необходимо определить правильное расположение метода onShowAllDeletedUsers() и реализовать этот метод.  
3. Необходимо определить правильное расположение метода loadDeletedUsers() и реализовать этот метод.  
4. В методе main класса Solution необходимо вызвать метод, который ты ранее реализовал в классе UsersView.  
5. Интерфейс Model должен содержать объявление метода, который ты ранее реализовал в классе MainModel.  
6. В классе FakeModel в теле метода, помещенного в интерфейс Model, необходимо бросить UnsupportedOperationException.\

### MVC (6)

Функционал отображения удаленных пользователей есть, а самих таких пользователей нет. Давай это исправим.  
Давай сделаем новую Вью, которая будет отвечать за редактирование одного конкретного пользователя.  
**UsersView** отображает список пользователей.  
**EditUserView** будет отображать данные о редактировании конкретного пользователя.  
Для этого нам сначала нужен этот выбранный пользователь.  
Как и любые данные его поместим в **ModelData**.

1. Создай в **ModelData** поле User activeUser с геттером и сеттером (Alt+Insert -> Getter and Setter).

2. Аналогично **UsersView** создай **EditUserView**.  
Логика метода refresh:  
2.1. Вывести в консоль «***User to be edited:***«.  
2.2. С новой строки вывести табуляцию и активного пользователя.  
2.3. С новой строки вывести разделитель «===================================================».

3. Создай в контроллере поле EditUserView editUserView с сеттером.

Когда наши данные выводятся в консоль, то совсем не понятно, список каких пользователей — удаленных или нет — выводится.  
Давай сделаем так, чтобы Вью отображала эту информацию. Все данные для отображения хранятся в Моделе.

**Поэтому:**  
4. создай в **ModelData** поле boolean displayDeletedUserList с **геттером** и **сеттером**.

5. Измени метод refresh в **UsersView** так, чтобы он отображал «***All users:***» либо «***All deleted users:***»  
в зависимости от того, какие пользователи находятся в списке.  
Добавь в необходимые методы модели изменение **displayDeletedUserList**.

**Требования:**  
1. В классе ModelData должно быть создано приватное поле User activeUser, геттер и сеттер для этого поля.  
2. Класс EditUserView должен быть создан аналогично классу UsersView: он должен поддерживать интерфейс View, содержать приватное поле Controller controller и сеттер этого поля.  
3. Метод refresh класса EditUserView должен быть реализован согласно условию.  
4. В классе Controller должно быть создано приватное поле EditUserView editUserView и сеттер этого поля.  
5. В классе ModelData должно быть создано приватное поле boolean displayDeletedUserList, геттер и сеттер для этого поля.  
6. Метод refresh в классе UsersView должен быть изменен согласно условию.  
7. Необходимо добавить в некоторые методы класса MainModel вызов метода setDisplayDeletedUserList(boolean) с правильным флагом.

### MVC (7)

1. Распредели методы по классам **MVC**:

public void onOpenUserEditForm(long userId) {  
…loadUserById(userId);  
…refresh(…getModelData());  
}

public void fireEventOpenUserEditForm(long id) {  
…onOpenUserEditForm(id);  
}

public void loadUserById(long userId) {  
User user = userService.getUsersById(userId);  
…setActiveUser(user);  
}

!!!! Пользователь видит Вью со списком пользователей, нажимает на одного из них, запрос идет на сервер, выгребаем новые данные и отображаем другую Вью, которая относится к одному выбранному пользователю.  
Учти это при реализации этого задания.

2. Добавь в метод main открытие формы редактирования для пользователя с id**=126** перед вызовом метода fireEventShowDeletedUsers().

3. Добавь в интерфейс Model метод, который ты поместил в Модель, реализуй его в **FakeModel**: выброси **UnsupportedOperationException**.

**Требования:**  
1. Необходимо определить правильное расположение метода onOpenUserEditForm(long) и реализовать этот метод.  
2. Необходимо определить правильное расположение метода fireEventOpenUserEditForm(long) и реализовать этот метод.  
3. Необходимо определить правильное расположение метода loadUserById(long) и реализовать этот метод.  
4. В методе main класса Solution необходимо вызвать метод открытия формы редактирования для пользователя с id=126 у объекта класса UsersView перед вызовом метода fireEventShowDeletedUsers().  
5. Интерфейс Model должен содержать объявление метода, который ты ранее реализовал в классе MainModel.  
6. В классе FakeModel в теле метода, помещенного в интерфейс Model, необходимо бросить UnsupportedOperationException.

### MVC (8)

1. Следуя принципу **MVC** аналогично реализации предыдущих методов сделай следующее:  
напиши логику удаления пользователя. После удаления должен отображаться список пользователей.

Распредели методы по классам **MVC**, используя следующие сигнатуры публичных методов:  
void fireEventUserDeleted(long id)  
void onUserDelete(long id)  
void deleteUserById(long id)

**Примечание:** метод, который ты собираешься добавить в Вью нужно добавить в**EditUserView**.

2. Добавь в main вызов fireEventUserDeleted(**124L**) перед вызовом метода fireEventShowDeletedUsers().

3. Выполни рефакторинг класса **MainModel**. Теперь, когда есть удаленные пользователи, часть методов стала работать неправильно.  
Почти во всех методах, где требуется список пользователей, нужно работать только с активными(живыми) пользователями.  
Вынеси в отдельный приватный метод**List**<User> getAllUsers() получение списка всех пользователей.  
Фильтрация активных пользователей у тебя уже есть — метод **List**<User> filterOnlyActiveUsers(List<User> **allUsers**).  
Отрефактори все методы, которые используют список пользователей. Они должны использовать список живых пользователей.

4. Добавь в интерфейс **Model** метод, который ты поместил в Модель, реализуй его в **FakeModel**: выброси **UnsupportedOperationException**.

**Требования:**  
1. Необходимо определить правильное расположение метода fireEventUserDeleted(long) и реализовать этот метод.  
2. Необходимо определить правильное расположение метода onUserDelete(long) и реализовать этот метод.  
3. Необходимо определить правильное расположение метода deleteUserById(long) и реализовать этот метод.  
4. В методе main класса Solution необходимо вызвать метод fireEventUserDeleted(124L) у объекта класса EditUserView перед вызовом метода fireEventShowDeletedUsers().  
5. В методе main вызов методов должен происходить в такой последовательности: fireEventShowAllUsers(), fireEventOpenUserEditForm(126L), fireEventUserDeleted(124L), fireEventShowDeletedUsers().  
6. Необходимо реализовать приватный метод List getAllUsers() в классе MainModel.  
7. Необходимо выполнить рефакторинг: все методы класса MainModel, которые используют список пользователей, должны использовать список живых пользователей.  
8. Интерфейс Model должен содержать объявление метода, который ты ранее реализовал в классе MainModel.  
9. В классе FakeModel в теле метода, помещенного в интерфейс Model, необходимо бросить UnsupportedOperationException.

### MVC (9)

Это последнее задание по **MVC**.

1. Следуя принципу **MVC** аналогично реализации предыдущих методов сделай следующее:  
напиши логику обновления пользователя. После обновления должен отображаться список пользователей.

Распредели методы по классам **MVC**, используя следующие сигнатуры публичных методов:  
void fireEventUserChanged(String name, long id, int level)  
void onUserChange(String name, long id, int level)  
void changeUserData(String name, long id, int level)

где **name** и **level** — это новые значения для пользователя с выбранным **id**.

**Примечание:** метод, который ты собираешься добавить в Вью нужно добавить в **EditUserView**.

2. Добавь в main вызов fireEventUserChanged перед вызовом метода fireEventShowDeletedUsers().

3. Добавь в интерфейс **Model** метод, который ты поместил в Модель, реализуй его в **FakeModel**: выброси **UnsupportedOperationException**.

**Требования:**  
1. Необходимо определить правильное расположение метода fireEventUserChanged(String name, long id, int level) и реализовать этот метод.  
2. Необходимо определить правильное расположение метода onUserChange(String name, long id, int level) и реализовать этот метод.  
3. Необходимо определить правильное расположение метода void changeUserData(String name, long id, int level) и реализовать этот метод.  
4. В методе main необходимо вызвать метод fireEventUserChanged(String, long, int) перед вызовом метода fireEventShowDeletedUsers().  
5. Интерфейс Model должен содержать объявление метода, который ты ранее реализовал в классе MainModel.  
6. В классе FakeModel в теле метода, помещенного в интерфейс Model, необходимо бросить UnsupportedOperationException.