package com.javarush.test.level13.lesson02.task01;  
  
/\* Пиво  
1. Подумай, какой из двух интерфейсов нужно реализовать в классе Beer.  
2. Добавь к классу Beer этот интерфейс и реализуйте все нереализованные методы.  
3. Подумай, как связаны переменная READY\_TO\_GO\_HOME и метод isReadyToGoHome.  
4. Верни значение переменной READY\_TO\_GO\_HOME в методе isReadyToGoHome.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 }  
  
 public interface Drink  
 {  
 void askMore(String message);  
  
 void sayThankYou();  
  
 boolean isReadyToGoHome();  
 }  
  
 public interface Alcohol extends Drink  
 {  
 boolean *READY\_TO\_GO\_HOME* = false;  
  
 void sleepOnTheFloor();  
 }  
  
 public static class Beer  
 {  
  
 }  
}

package com.javarush.test.level13.lesson02.task02;  
  
/\* Пиво: возвращение  
Добавь к классу AlcoholicBeer интерфейс Drink и реализуй все нереализованные методы.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 Drink beer = new AlcoholicBeer();  
 System.*out*.println(beer.toString());  
 }  
  
 public interface Drink  
 {  
 boolean isAlcoholic();  
 }  
  
 public static class AlcoholicBeer  
 {  
 @Override  
 public String toString()  
 {  
 if (isAlcoholic()) {  
 return "Напиток алкогольный";  
 }  
 else {  
 return "Напиток безалкогольный";  
 }  
  
 }  
  
 }  
}

package com.javarush.test.level13.lesson02.task03;  
  
/\* Пиво и кола  
Реализуй интерфейс Drink в классах Beer и Cola.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
  
 *print*(new Beer());  
 *print*(new Cola());  
  
 }  
  
 private static void print(Drink drink)  
 {  
 System.*out*.println(drink.getClass().getSimpleName());  
 }  
  
 public interface Drink  
 {  
 boolean isAlcoholic();  
 }  
  
 public static class Beer  
 {  
 }  
  
 public static class Cola  
 {  
 }  
}

package com.javarush.test.level13.lesson02.task04;  
  
/\* Selectable и Updatable  
1. Создай класс Screen.  
2. Реализуй в нем интерфейсы Selectable с методом onSelect, Updatable с методом refresh.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 }  
  
 interface Selectable  
 {  
 void onSelect();  
 }  
  
 interface Updatable  
 {  
 void refresh();  
 }  
  
 //напишите тут ваш класс  
}

package com.javarush.test.level13.lesson02.task05;  
  
/\* 4 ошибки  
Исправь 4 ошибки в программе, чтобы она компилировалась.  
Объявление интерфейсов не изменять.  
\*/  
  
public class Solution  
{  
  
 public static void main(String[] args) throws Exception  
 {  
  
 System.*out*.println(new Dream().HOBBIE.toString());  
 System.*out*.println(new Hobbie().toString());  
  
 }  
  
 interface Desire  
 {  
 }  
  
 interface Dream  
 {  
 private static Hobbie *HOBBIE* = new Hobbie();  
 }  
  
 class Hobbie extends Desire implements Dream  
 {  
 static int *INDEX* = 1;  
  
 @Override  
 public String toString()  
 {  
 *INDEX*++;  
 return "" + *INDEX*;  
 }  
 }  
  
}

package com.javarush.test.level13.lesson02.task06;  
  
/\* Баг в initializeIdAndName  
1. Подумать, что в программе неправильно.  
2. Вынести реализацию метода initializeIdAndName в класс User.  
3. initializeIdAndName в классе User должен возвращать тип User.  
4. Поправить программу, чтобы компилировалась и работала.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 System.*out*.println(Matrix.*NEO*);  
 System.*out*.println(Matrix.*TRINITY*);  
 }  
  
 static class Matrix  
 {  
 public static DBObject *NEO* = new User().initializeIdAndName(1, "Neo");  
 public static DBObject *TRINITY* = new User().initializeIdAndName(2, "Trinity");  
 }  
  
 interface DBObject  
 {  
 DBObject initializeIdAndName(long id, String name) {  
 this.id = id;  
 this.name = name;  
 return this;  
 }  
 }  
  
 static class User implements DBObject  
 {  
 long id;  
 String name;  
  
 @Override  
 public String toString()  
 {  
 return String.*format*("User has name %s, id = %d", name, id);  
 }  
 }  
  
}

package com.javarush.test.level13.lesson02.task07;  
  
/\* Параметризованый интерфейс  
В классе StringObject реализуй интерфейс SimpleObject с параметром типа String.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 }  
  
 interface SimpleObject<T>  
 {  
 SimpleObject<T> getInstance();  
 }  
  
 class StringObject //допишите здесь ваш код  
 {  
 }  
}

package com.javarush.test.level13.lesson02.task08;  
  
/\* Жив или нет  
1. Создай интерфейс Person.  
2. Добавь в него метод isAlive(), который проверяет, жив человек или нет.  
3. Подумай, какой тип должен возвращать этот метод.  
4. Создай интерфейс Presidentable.  
5. Унаследуй интерфейс Presidentable от интерфейса Person.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 }  
}

package com.javarush.test.level13.lesson02.task09;  
  
/\* Наследование интерфейса  
1. Создать интерфейс Movable с методом speed.  
2. Метод speed возвращает значение типа Double и не принимает параметров.  
3. Создать и унаследовать интерфейс Runable от интерфейса Movable.  
4. Добавить в интерфейс Runable метод speed.  
5. Метод speed возвращает значение типа Double и принимает один параметр типа Runable.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 }  
}

package com.javarush.test.level13.lesson02.task10;  
  
/\* CleverMan и SmartGirl  
1. Добавь интерфейсы Secretary и Boss к классам CleverMan и SmartGirl. По одному на каждый. Подумай, кому какой.  
2. Унаследуй интерфейсы Secretary и Boss от интерфейсов Person и Workable так,  
чтобы все методы у классов CleverMan и SmartGirl оказались объявленными в каком-то интерфейсе.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 }  
  
 interface Person  
 {  
 void use(Person person);  
 void startToWork();  
 }  
  
 interface Workable  
 {  
 boolean wantGetExtraWork();  
 }  
  
 interface Secretary  
 {  
 }  
  
 interface Boss  
 {  
 }  
  
 class CleverMan  
 {  
 public void use(Person person)  
 {  
 person.startToWork();  
 }  
  
 public void startToWork()  
 {  
 }  
  
 public boolean wantGetExtraWork()  
 {  
 return true;  
 }  
 }  
  
 class SmartGirl  
 {  
 public void use(Person person)  
 {  
 }  
  
 public void startToWork()  
 {  
 }  
 }  
}

package com.javarush.test.level13.lesson04.task01;  
  
/\* Переводчик с английского  
1. Создать класс EnglishTranslator, который наследуется от Translator.  
2. Реализовать все абстрактные методы.  
3. Подумай, что должен возвращать метод getLanguage.  
4. Программа должна выводить на экран "Я переводчик с английского".  
5. Метод main менять нельзя.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 EnglishTranslator englishTranslator = new EnglishTranslator();  
 System.*out*.println(englishTranslator.translate());  
 }  
  
 public static abstract class Translator  
 {  
 public abstract String getLanguage();  
  
 public String translate()  
 {  
 return "Я переводчик с " + getLanguage();  
 }  
 }  
  
}

package com.javarush.test.level13.lesson04.task02;  
  
/\* Исправление кода  
Исправить код так, чтобы программа выполнялась и выводила "Я переводчик с английского".  
Метод main менять нельзя!  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 Translator translator = new Translator();  
 System.*out*.println(translator.translate());  
 }  
  
 public abstract static class Translator  
 {  
 public String translate()  
 {  
 return "Я переводчик с английского";  
 }  
 }  
  
  
}

package com.javarush.test.level13.lesson04.task03;  
  
import java.awt.\*;  
  
/\* Интерфейс Animal  
1. Унаследовать Fox от интерфейса Animal.  
2. Поменяй код так, чтобы в классе Fox был только один метод - getName.  
3. Создавать дополнительные классы и удалять методы нельзя!  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 }  
  
 public interface Animal  
 {  
 Color getColor();  
 }  
  
 public static class Fox  
 {  
 public String getName()  
 {  
 return "Fox";  
 }  
 }  
}

package com.javarush.test.level13.lesson04.task04;  
  
import java.awt.\*;  
  
/\* Класс BigFox  
1. Исправь класс BigFox так, чтобы программа компилировалась.  
2. Метод main менять нельзя.  
\*/  
  
public class Solution  
{  
  
 public static void main(String[] args) throws Exception  
 {  
  
 Fox bigFox = new BigFox();  
 System.*out*.println(bigFox.getName());  
 System.*out*.println(bigFox.getColor());  
  
 }  
  
 public interface Animal  
 {  
 Color getColor();  
 }  
  
 public static abstract class Fox implements Animal  
 {  
 public String getName() {  
 return "Fox";  
 }  
 }  
  
 //add your code below  
 public static class BigFox  
 {  
 public Color () {  
 return Color.*GRAY*;  
 }  
 }  
  
}

package com.javarush.test.level13.lesson06.task01;  
  
/\* Dog, Cat и Mouse  
1. Создать классы Dog, Cat и Mouse.  
2. Реализовать интерфейсы в добавленных классах, учитывая что:  
- Кот(Cat) может передвигаться, может кого-то съесть и может быть сам съеден.  
- Мышь(Mouse) может передвигаться и может быть съедена.  
- Собака(Dog) может передвигаться и съесть кого-то.  
\*/  
  
public class Solution  
{  
  
 public interface Moveable //может двигаться  
 {  
 void move();  
 }  
  
 public interface Eatable //может быть съеден  
 {  
 void eaten();  
 }  
  
 public interface Eat //может кого-нибудь съесть  
 {  
 void eat();  
 }  
}

package com.javarush.test.level13.lesson11.home01;  
  
/\* Некорректные строки  
Удали все некорректные строки в интерфейсе Button.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 System.*out*.println(SimpleObject.*NAME*);  
 System.*out*.println(Button.*NAME*);  
 }  
  
 interface SimpleObject  
 {  
 String *NAME* = "SimpleObject";  
 void onPress();  
 }  
  
 interface Button extends SimpleObject  
 {  
  
 final String *NAME* = "Submit";  
 public void onPress();  
 protected void onPress();  
 void onPress();  
 private void onPress();  
  
 protected String onPress(Object o);  
 String onPress(Object o);  
 private String onPress(Object o);  
  
 }  
}

package com.javarush.test.level13.lesson11.home02;  
  
/\* Погода  
1. В классе Today реализовать интерфейс Weather.  
2. Подумай, как связан параметр String type с методом getWeatherType().  
3. Интерфейсы Weather и WeatherType уже реализованы.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args)  
 {  
 System.*out*.println(new Today(WeatherType.*CLOUDY*));  
 System.*out*.println(new Today(WeatherType.*FOGGY*));  
 System.*out*.println(new Today(WeatherType.*FROZEN*));  
 }  
  
 static class Today  
 {  
 private String type;  
  
 Today(String type)  
 {  
 this.type = type;  
 }  
  
 @Override  
 public String toString()  
 {  
 return String.format("%s for today", this.getWeatherType());  
 }  
 }  
}

package com.javarush.test.level13.lesson11.home03;  
  
/\* Чтение файла  
1. Считать с консоли имя файла.  
2. Вывести в консоль(на экран) содержимое файла.  
3. Не забыть освободить ресурсы. Закрыть поток чтения с файла и поток ввода с клавиатуры.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args)  
 {  
 //add your code here  
 }  
}

package com.javarush.test.level13.lesson11.home04;  
  
/\* Запись в файл  
1. Прочесть с консоли имя файла.  
2. Считывать строки с консоли, пока пользователь не введет строку "exit".  
3. Вывести абсолютно все введенные строки в файл, каждую строчку с новой стороки.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args)  
 {  
 }  
}

package com.javarush.test.level13.lesson11.home05;  
  
/\* Neo  
1. Реализовать интерфейс DBObject в классе User.  
2. Реализовать метод initializeIdAndName так, чтобы программа работала и выводила на экран "User has name Neo, id = 1".  
3. Метод toString и метод main менять нельзя.  
4. Подумай, что должен возвращать метод initializeIdAndName в классе User.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 System.*out*.println(Matrix.*NEO*);  
 }  
  
  
 static class Matrix  
 {  
 public static DBObject *NEO* = new User().initializeIdAndName(1, "Neo");  
 }  
  
 interface DBObject  
 {  
 DBObject initializeIdAndName(long id, String name);  
 }  
  
 static class User  
 {  
 long id;  
 String name;  
  
 @Override  
 public String toString()  
 {  
 return String.*format*("User has name %s, id = %d", name, id);  
 }  
 }  
  
}

package com.javarush.test.level13.lesson11.home06;  
  
/\* Исправление ошибок  
1. Переделай наследование в классах и интерфейсах так, чтобы программа компилировалась и продолжала делать то же самое.  
2. Класс Hobbie должен наследоваться от интерфейсов Desire, Dream.  
\*/  
  
public class Solution  
{  
  
 public static void main(String[] args) throws Exception  
 {  
 System.*out*.println(Dream.*HOBBIE*.toString());  
 System.*out*.println(new Hobbie().*INDEX*);  
 }  
  
 interface Desire  
 {  
 }  
  
 interface Dream implements Hobbie  
 {  
 static Hobbie *HOBBIE* = new Hobbie();  
 }  
  
 static class Hobbie extends Desire, Dream  
 {  
 static int *INDEX* = 1;  
  
 @Override  
 public String toString()  
 {  
 *INDEX*++;  
 return "" + *INDEX*;  
 }  
 }  
  
}

package com.javarush.test.level13.lesson11.home07;  
  
/\* Интерфейс SimpleObject  
1. Создай класс StringObject.  
2. В классе StringObject реализуй интерфейс SimpleObject с параметром типа String.  
3. Программа должна компилироваться.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 SimpleObject<String> stringObject = new StringObject<Object>();  
 }  
  
 interface SimpleObject<T>  
 {  
 SimpleObject<T> getInstance();  
 }  
  
}

package com.javarush.test.level13.lesson11.home08;  
  
/\* Интерфейс Updatable в классе Screen  
Реализовать интерфейс Updatable в классе Screen.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 }  
  
 interface Selectable  
 {  
 void onSelect();  
 }  
  
 interface Updatable extends Selectable  
 {  
 void refresh();  
 }  
  
 class Screen  
 {  
  
 }  
}

package com.javarush.test.level13.lesson11.home09;  
  
import java.awt.\*;  
  
/\* Один метод в классе  
1. Унаследовать Fox от интерфейса Animal.  
2. Поменяй код так, чтобы в классе Fox был только один метод - getName.  
3. Методы удалять нельзя!  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 }  
  
 public interface Animal  
 {  
 Color getColor();  
  
 Integer getAge();  
 }  
  
 public static class Fox  
 {  
 public String getName()  
 {  
 return "Fox";  
 }  
 }  
}

package com.javarush.test.level13.lesson11.home10;  
  
import java.awt.\*;  
  
/\* Компиляция программы  
1. Исправь классы Fox и BigFox так, чтобы программа компилировалась. Задача не предполагает создания экземпляров базового класса.  
2. Метод main менять нельзя.  
\*/  
  
public class Solution  
{  
  
 public static void main(String[] args) throws Exception  
 {  
 Fox bigFox = new BigFox();  
 System.*out*.println(bigFox.getName());  
 System.*out*.println(bigFox.getColor());  
 }  
  
 public interface Animal  
 {  
 Color getColor();  
 }  
  
 public static class Fox implements Animal  
 {  
 public String getName() {  
 return "Fox";  
 }  
 }  
  
 public abstract static class BigFox  
 {  
  
 }  
  
}

package com.javarush.test.level13.lesson11.bonus01;  
  
/\* Сортировка четных чисел из файла  
1. Ввести имя файла с консоли.  
2. Прочитать из него набор чисел.  
3. Вывести на консоль только четные, отсортированные по возрастанию.  
Пример ввода:  
5  
8  
11  
3  
2  
10  
Пример вывода:  
2  
8  
10  
\*/  
  
public class Solution  
{  
 public static void main(String[] args)  
 {  
 // напишите тут ваш код  
 }  
}

package com.javarush.test.level13.lesson11.bonus02;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\* Репка  
Сказка Репка:  
1. Реализовать интерфейс RepkaItem в классе Person.  
2. В классе Person реализовать метод pull(Person person), который выводит фразу типа '<name> за <person>'. Пример:  
Бабка за Дедку  
Дедка за Репку  
3. Исправить логическую ошибку цикла в методе tell класса RepkaStory.  
4. Выполнить метод main и наслаждаться сказкой!  
\*/  
  
public class Solution  
{  
 public static void main(String[] args)  
 {  
 List<Person> plot = new ArrayList<Person>();  
 plot.add(new Person("Репка", "Репку"));  
 plot.add(new Person("Дедка", "Дедку"));  
 plot.add(new Person("Бабка", "Бабку"));  
 plot.add(new Person("Внучка", "Внучку"));  
 RepkaStory.*tell*(plot);  
 }  
}

package com.javarush.test.level13.lesson11.bonus02;  
  
import java.util.List;  
  
public class RepkaStory  
{  
 static void tell(List<Person> items) {  
  
 Person first;  
 Person second;  
 for (int i = items.size() - 1; i > 0; i--) {  
 first = items.get(i - 1);  
 second = items.get(i);  
 first.pull(second);  
 }  
 }  
}

package com.javarush.test.level13.lesson11.bonus02;  
  
public class Person  
{  
 private String name;  
 private String namePadezh;  
  
 public Person(String name, String namePadezh)  
 {  
 this.name = name;  
 this.namePadezh = namePadezh;  
 }  
}

package com.javarush.test.level13.lesson11.bonus02;  
  
public interface RepkaItem  
{  
 public String getNamePadezh();  
}

package com.javarush.test.level13.lesson11.bonus03;  
  
/\* Битва роботов  
1 Разобраться в том, что тут написано.  
2 http://info.javarush.ru/uploads/images/00/00/07/2014/08/12/50f3e37f94.png  
3 Смириться со своей участью и продолжить разбираться в коде.  
4 ...  
5 Порадоваться, что мы все поняли.  
  
6 Изменить код согласно новой архитектуре и добавить новую логику:  
6.1 Сделать класс AbstractRobot абстрактным, вынести логику атаки и защиты из Robot в AbstractRobot.  
6.2 Отредактировать класс Robot учитывая AbstractRobot.  
6.3 Расширить класс BodyPart новой частью тела "грудь".  
6.4 Добавить новую часть тела в реализацию интерфейсов Attackable и Defensable (в классе AbstractRobot).  
  
7 http://info.javarush.ru/uploads/images/00/00/07/2014/08/12/3b9c65580b.png  
\*/  
  
public class Solution  
{  
 public static void main(String[] args)  
 {  
 Robot amigo = new Robot("Амиго");  
 Robot enemy = new Robot("Сгибальщик-02");  
  
 *doMove*(amigo, enemy);  
 *doMove*(amigo, enemy);  
 *doMove*(enemy, amigo);  
 *doMove*(amigo, enemy);  
 *doMove*(enemy, amigo);  
 *doMove*(amigo, enemy);  
 *doMove*(enemy, amigo);  
 *doMove*(amigo, enemy);  
 }  
  
 public static void doMove(AbstractRobot robotFirst, AbstractRobot robotSecond) {  
 BodyPart attacked = robotFirst.attack();  
 BodyPart defenced = robotFirst.defense();  
 System.*out*.println(String.*format*("%s атаковал робота %s, атакована %s, защищена %s",  
 robotFirst.getName(), robotSecond.getName(), attacked, defenced));  
 }  
}

package com.javarush.test.level13.lesson11.bonus03;  
  
public final class BodyPart  
{  
 final static BodyPart *LEG* = new BodyPart("нога");  
 final static BodyPart *HEAD* = new BodyPart("голова");  
 final static BodyPart *ARM* = new BodyPart("рука");  
  
 private String bodyPart;  
  
 private BodyPart(String bodyPart)  
 {  
 this.bodyPart = bodyPart;  
 }  
  
 @Override  
 public String toString()  
 {  
 return this.bodyPart;  
 }  
}

package com.javarush.test.level13.lesson11.bonus03;  
  
public interface Attackable  
{  
 BodyPart attack();  
}

package com.javarush.test.level13.lesson11.bonus03;  
  
public interface Defensable  
{  
 BodyPart defense();  
}

package com.javarush.test.level13.lesson11.bonus03;  
  
public class Robot  
{  
 private static int *hitCount*;  
 private String name;  
  
 public Robot(String name)  
 {  
 this.name = name;  
 }  
  
 public String getName()  
 {  
 return name;  
 }  
  
 public BodyPart attack()  
 {  
 BodyPart attackedBodyPart = null;  
 *hitCount* = *hitCount* + 1;  
  
 if (*hitCount* == 1)  
 {  
 attackedBodyPart = BodyPart.*ARM*;  
 } else if (*hitCount* == 2)  
 {  
 attackedBodyPart = BodyPart.*HEAD*;  
 } else if (*hitCount* == 3)  
 {  
 *hitCount* = 0;  
 attackedBodyPart = BodyPart.*LEG*;  
 }  
 return attackedBodyPart;  
 }  
  
 public BodyPart defense()  
 {  
 BodyPart defencedBodyPart = null;  
 *hitCount* = *hitCount* + 1;  
  
 if (*hitCount* == 1)  
 {  
 defencedBodyPart = BodyPart.*HEAD*;  
 } else if (*hitCount* == 2)  
 {  
 defencedBodyPart = BodyPart.*LEG*;  
 } else if (*hitCount* == 3)  
 {  
 *hitCount* = 0;  
 defencedBodyPart = BodyPart.*ARM*;  
 }  
 return defencedBodyPart;  
 }  
}

package com.javarush.test.level13.lesson11.bonus03;  
  
public class AbstractRobot  
{  
}