package com.javarush.test.level14.lesson02.task01;  
  
/\* Bingo  
Исправь строчку 'Object o = new Pet();' в методе main так, чтобы программа вывела "Bingo!"  
\*/  
  
public class Solution  
{  
 public static void main(String[] args)  
 {  
 Object o = new Pet();  
 boolean isCat = o instanceof Cat;  
 boolean isTiger = o instanceof Tiger;  
 boolean isPet = o instanceof Pet;  
  
 *printResults*(isCat, isTiger, isPet);  
 }  
  
 private static void printResults(boolean cat, boolean tiger, boolean pet)  
 {  
 if (cat && tiger && pet) System.*out*.println("Bingo!");  
 }  
  
 static class Pet  
 {  
 }  
  
 static class Cat extends Pet  
 {  
 }  
  
 static class Tiger extends Cat  
 {  
 }  
  
}

package com.javarush.test.level14.lesson02.task02;  
  
/\* Bingo-2  
Исправь строчку 'Cat o = new Cat();' так, чтобы программа вывела "Bingo!"  
\*/  
  
public class Solution  
{  
 public static void main(String[] args)  
 {  
 Cat o = new Cat();  
  
 boolean isCat = o instanceof Cat;  
 boolean isMoveable = o instanceof Moveable;  
 boolean isTom = o instanceof TomCat;  
  
 if (isCat && isMoveable && isTom) System.*out*.println("Bingo!");  
  
 }  
  
 interface Moveable  
 {  
 }  
  
 static class Cat  
 {  
 }  
  
 static class TomCat extends Cat implements Moveable  
 {  
  
 }  
}

package com.javarush.test.level14.lesson04.task01;  
  
/\* Building и School  
1. Расставь правильно наследование между Building(здание) и School(здание школы).  
2. Подумай, объект какого класса должны возвращать методы getSchool и getBuilding.  
3. Измени null на объект класса Building или School.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args)  
 {  
 Building school = *getSchool*();  
 Building shop = *getBuilding*();  
  
 System.*out*.println(school);  
 System.*out*.println(shop);  
 }  
  
 public static Building getSchool()  
 {  
 //измените null на объект класса Building или School  
 return null;  
 }  
  
 public static Building getBuilding()  
 {  
 //измените null на объект класса Building или School  
 return null;  
 }  
  
 static class School /\*Add your code here\*/  
 {  
 @Override  
 public String toString()  
 {  
 return "School";  
 }  
 }  
  
 static class Building /\*Add your code here\*/  
 {  
 @Override  
 public String toString()  
 {  
 return "Building";  
 }  
 }  
}

package com.javarush.test.level14.lesson04.task02;  
  
import java.io.BufferedReader;  
import java.io.InputStreamReader;  
  
/\* Коты  
1. Считывать строки(параметры) с консоли, пока пользователь не введет пустую строку(Enter).  
2. Каждый параметр соответствует имени кота.  
Для каждого параметра:  
3. Создать объект cat класса Cat, который равен коту из getCatByKey(String параметр).  
4. Вывести на экран cat.toString().  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 //Add your code here  
 }  
  
 static class CatFactory  
 {  
  
 static Cat getCatByKey(String key)  
 {  
 Cat cat = null;  
 if ("vaska".equals(key))  
 {  
 cat = new MaleCat("Василий");  
 } else if ("murka".equals(key))  
 {  
 cat = new FemaleCat("Мурочка");  
 } else if ("kiska".equals(key))  
 {  
 cat = new FemaleCat("Кисюлька");  
 } else  
 {  
 cat = new Cat(key);  
 }  
  
 return cat;  
 }  
 }  
  
 static class Cat  
 {  
 private String name;  
  
 protected Cat(String name)  
 {  
 this.name = name;  
 }  
  
 public String getName()  
 {  
 return this.name;  
 }  
  
 public String toString()  
 {  
 return "Я уличный кот " + getName();  
 }  
 }  
  
 static class MaleCat extends Cat  
 {  
 MaleCat(String name)  
 {  
 super(name);  
 }  
  
 public String toString()  
 {  
 return "Я - солидный кошак по имени " + getName();  
 }  
 }  
  
 static class FemaleCat extends Cat  
 {  
 FemaleCat(String name)  
 {  
 super(name);  
 }  
  
 public String toString()  
 {  
 return "Я - милая кошечка по имени " + getName();  
 }  
 }  
}

package com.javarush.test.level14.lesson04.task03;  
  
/\* Food  
1. Реализовать интерфейс Selectable в классе Food.  
2. Метод onSelect() должен писать в консоль "food is selected".  
3. Подумай, какие методы можно вызвать для переменной food и какие для selectable.  
4. В методе foodMethods вызови методы onSelect, eat, если это возможно.  
5. В методе selectableMethods вызови методы onSelect, eat, если это возможно.  
6. Явное приведение типов не использовать.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args)  
 {  
 Food food = new Food();  
 Selectable selectable = new Food();  
 Food newFood = (Food) selectable;  
  
 *foodMethods*(food);  
 *selectableMethods*(selectable);  
 }  
  
 public static void foodMethods(Food food)  
 {  
 //тут добавьте вызов методов для переменной food  
 }  
  
 public static void selectableMethods(Selectable selectable)  
 {  
 //тут добавьте вызов методов для переменной selectable  
 }  
  
 interface Selectable  
 {  
 void onSelect();  
 }  
  
 static class Food  
 {  
 public void eat()  
 {  
 System.*out*.println("food is eaten");  
 }  
 }  
}

package com.javarush.test.level14.lesson04.task04;  
  
/\* Без ошибок  
Инициализировать объект obj таким классом, чтобы метод main выполнился без ошибок.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args)  
 {  
 Object obj = //Add your code here  
  
 Mouse mouse = (Mouse) obj;  
 GreyMouse greyMouse = (GreyMouse) mouse;  
 Jerry jerry = (Jerry) greyMouse;  
  
 *printClasses*(obj, mouse, greyMouse, jerry);  
  
 }  
  
 public static void printClasses(Object obj, Mouse mouse, GreyMouse greyMouse, Jerry jerry)  
 {  
 System.*out*.println(jerry.getClass().getSimpleName());  
 System.*out*.println(greyMouse.getClass().getSimpleName());  
 System.*out*.println(mouse.getClass().getSimpleName());  
 System.*out*.println(obj.getClass().getSimpleName());  
 }  
  
 static class Mouse  
 {  
 }  
  
 static class GreyMouse extends Mouse  
 {  
 }  
  
 static class Jerry extends GreyMouse  
 {  
 }  
}

package com.javarush.test.level14.lesson04.task05;  
  
import java.io.BufferedReader;  
import java.io.InputStreamReader;  
  
/\* Player and Dancer  
1. Подумать, что делает программа.  
2. Изменить метод haveRest так, чтобы он вызывал метод  
- play, если person имеет тип Player  
- dance, если person имеет тип Dancer  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 BufferedReader reader = new BufferedReader(new InputStreamReader(System.*in*));  
  
 Person person = null;  
 String key;  
 while (!(key = reader.readLine()).equals("exit"))  
 {  
 if ("player".equals(key))  
 {  
 person = new Player();  
 } else if ("dancer".equals(key))  
 {  
 person = new Dancer();  
 }  
 *haveRest*(person);  
 }  
 }  
  
 public static void haveRest(Person person)  
 {  
 //Add your code here  
 }  
  
 interface Person  
 {  
 }  
  
 static class Player implements Person  
 {  
 void play()  
 {  
 System.*out*.println("playing");  
 }  
 }  
  
 static class Dancer implements Person  
 {  
 void dance()  
 {  
 System.*out*.println("dancing");  
 }  
 }  
}

package com.javarush.test.level14.lesson06.home01;  
  
/\* Куриная фабрика  
Написать Фабрику(Factory) по производству кур(Hen)  
1. Создать класс Hen  
1.1. Сделать его абстрактным  
1.2. Добавить в класс абстрактный метод int getCountOfEggsPerMonth()  
1.3. Добавить в класс метод String getDescription(), который возвращает строку "Я курица."  
  
2. Создать класс RussianHen, который наследуется от Hen  
3. Создать класс UkrainianHen, который наследуется от Hen  
4. Создать класс MoldovanHen, который наследуется от Hen  
5. Создать класс BelarusianHen, который наследуется от Hen  
  
6. В каждом из четырех последних классов написать свою реализацию метода getCountOfEggsPerMonth.  
Методы должны возвращать количество яиц в месяц от данного типа куриц.  
  
7. В каждом из четырех последних классов написать свою реализацию метода getDescription.  
Методы должны возвращать строку вида:  
<getDescription() родительского класса> + <" Моя страна - Sssss. Я несу N яиц в месяц.">  
где Sssss - название страны  
где N - количество яиц в месяц  
  
8. В классе HenFactory реализовать метод getHen, который возвращает соответствующую стране породу кур  
9. Все созданные вами классы должны быть в отдельных файлах  
\*/  
  
public class Solution  
{  
 public static void main(String[] args)  
 {  
 Hen hen = HenFactory.*getHen*(Country.*BELARUS*);  
 hen.getCountOfEggsPerMonth();  
 }  
  
 static class HenFactory {  
  
 static Hen getHen(String country) {  
 Hen hen = null;  
 //add your code here  
 return hen;  
 }  
 }  
  
  
}

package com.javarush.test.level14.lesson06.home01;  
  
public interface Country  
{  
 String *UKRAINE* = "Ukraine";  
 String *RUSSIA* = "Russia";  
 String *MOLDOVA* = "Moldova";  
 String *BELARUS* = "Belarus";  
}

package com.javarush.test.level14.lesson08.home01;  
  
/\* Мосты  
1. Создать интерфейс Bridge с методом int getCarsCount().  
2. Создать классы WaterBridge и SuspensionBridge, которые реализуют интерфейс Bridge.  
3. Метод getCarsCount() должен возвращать любое захардкоженое значение типа int  
4. Метод getCarsCount() должен возвращать различные значения для различных классов  
5. В классе Solution создать публичный метод println(Bridge bridge).  
6. В методе println вывести на консоль значение getCarsCount() для объекта bridge.  
7. Каждый класс и интерфейс должны быть в отдельных файлах.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args)  
 {  
 println(new WaterBridge());  
 println(new SuspensionBridge());  
 }  
  
 //add println method here  
}

package com.javarush.test.level14.lesson08.home02;  
  
/\* Дегустация вин  
1. Создать абстрактный класс Drink с реализованным методом public void taste(), который выводит в консоль "Вкусно"  
2. Создать класс Wine, который наследуется от Drink, с реализованным методом public String getHolidayName(), который возвращает строку "День рождения"  
3. Создать класс BubblyWine, который наследуется от Wine, с реализованным методом public String getHolidayName(), который возвращает строку "Новый год"  
4. Написать реализацию методов getDeliciousDrink, getWine, getBubblyWine  
5. Каждый класс и интерфейс должны быть в отдельных файлах  
6. Метод main менять нельзя!  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 *getDeliciousDrink*().taste();  
 System.*out*.println(*getWine*().getHolidayName());  
 System.*out*.println(*getBubblyWine*().getHolidayName());  
 System.*out*.println(*getWine*().getHolidayName());  
 }  
  
 public static Drink getDeliciousDrink() {  
  
 }  
  
 public static Wine getWine() {  
  
 }  
  
 public static Wine getBubblyWine() {  
  
 }  
}

package com.javarush.test.level14.lesson08.home03;  
  
import java.io.BufferedReader;  
import java.io.InputStreamReader;  
  
/\* User, Looser, Coder and Proger  
1. Ввести [в цикле] с клавиатуры несколько строк (ключей).  
Строки(ключи) могут быть такими: "user", "looser", "coder", "proger".  
Ввод окончен, когда строка не совпадает ни с одной из выше указанных.  
  
2. Для каждой введенной строки нужно:  
2.1. Создать соответствующий объект [см Person.java], например, для строки "user" нужно создать объект класса User.  
2.2. Передать этот объект в метод doWork.  
  
3. Написать реализацию метода doWork, который:  
3.1. Вызывает метод live() у переданного обекта, если этот объект (person) имеет тип User.  
3.2. Вызывает метод doNothing(), если person имеет тип Looser.  
3.3. Вызывает метод coding(), если person имеет тип Coder.  
3.4. Вызывает метод enjoy(), если person имеет тип Proger.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 BufferedReader reader = new BufferedReader(new InputStreamReader(System.*in*));  
 Person person = null;  
 String key = null;  
  
 //тут цикл по чтению ключей, пункт 1  
 {  
 //создаем объект, пункт 2  
  
 *doWork*(person); //вызываем doWork  
  
 }  
 }  
  
 public static void doWork(Person person)  
 {  
 // пункт 3  
 }  
}

package com.javarush.test.level14.lesson08.home04;  
  
/\* Реализовать метод printMainInfo  
1. Напиши реализацию метода printMainInfo, чтобы:  
1.1. Если в метод передают объект типа Drawable, у этого объекта вызывался метод draw.  
1.2. Если в метод передают объект типа Movable, у этого объекта вызывался метод move.  
2. Метод main менять нельзя.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args)  
 {  
 Object obj = new Circle();  
 Movable movable = (Movable) obj;  
 Drawable drawable = new Rectangle();  
  
 *printMainInfo*(drawable);  
 *printMainInfo*(movable);  
 }  
  
 public static void printMainInfo(Object object)  
 {  
 //Add your code here  
 }  
  
 static interface Movable  
 {  
  
 void move();  
 }  
  
 static class Circle implements Movable  
 {  
  
 public void draw()  
 {  
 System.*out*.println("can be drawn");  
 }  
  
 public void move()  
 {  
 System.*out*.println("can be moved");  
 }  
  
 }  
  
 static interface Drawable  
 {  
 void draw();  
 }  
  
 static class Rectangle implements Drawable  
 {  
 public void draw()  
 {  
 System.*out*.println("can be drawn");  
 }  
  
 public void move()  
 {  
 System.*out*.println("can be moved");  
 }  
 }  
}

package com.javarush.test.level14.lesson08.home05;  
  
/\* Computer  
1. Создай интерфейс CompItem.  
2. Добавь в него метод String getName().  
3. Создай классы Keyboard, Mouse, Monitor, которые реализуют интерфейс CompItem.  
4. Метод getName() должен возвращать имя класса, например, для класса Keyboard будет "Keyboard".  
5. Создай класс Computer.  
6. В класс Computer добавь приватное поле типа Keyboard.  
7. В класс Computer добавь приватное поле типа Mouse.  
8. В класс Computer добавь приватное поле типа Monitor.  
9. Создай конструктор в классе Computer используя комбинацию клавиш Alt+Insert внутри класса (команда Constructor).  
10 Внутри конструктора инициализируйте все три поля (переменных) класса  
11. Создай геттеры для полей класса Computer (в классе используй комбинацию клавиш Alt+Insert и выбери команду Getter).  
12. Все созданные классы и интерфейс должны быть в отдельных файлах.  
13. Класс Solution менять нельзя.  
\*/  
  
public class Solution  
{  
  
 public static void main(String[] args)  
 {  
 Computer computer = new Computer();  
 if (*isWork*(computer.getKeyboard()) &&  
 *isWork*(computer.getMonitor()) &&  
 *isWork*(computer.getMouse()))  
 {  
 System.*out*.println("Work!");  
 }  
 }  
  
 public static boolean isWork(CompItem item)  
 {  
 System.*out*.println(item.getName());  
 return item.getName() != null && item.getName().length() > 4;  
 }  
  
}

package com.javarush.test.level14.lesson08.home06;  
  
/\* MovieFactory  
Расширение функционала по аналогии, чтение с консоли:  
1. Разобраться, что программа умеет делать.  
2. Все классы должны быть внутри класса Solution.  
3. Добавить классы Cartoon, Thriller.  
4. Разобраться, как мы получаем объект класса SoapOpera по ключу "soapOpera".  
Аналогично получению объекта SoapOpera сделать:  
5. Добавить в MovieFactory.getMovie получение объекта Cartoon для ключа "cartoon".  
6. Добавить в MovieFactory.getMovie получение объекта Thriller для ключа "thriller".  
  
7. Считать с консоли несколько ключей (строк).  
7.1. Ввод заканчивается, как только вводится строка не совпадающая с одной из: "cartoon", "thriller", "soapOpera".  
8. Создать переменную movie класса Movie и для каждой введенной строки(ключа):  
8.1. Получить объект используя MovieFactory.getMovie и присвоить его переменной movie.  
8.2. Вывести на экран movie.getClass().getSimpleName().  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 //ввести с консоли несколько ключей (строк), пункт 7  
  
 /\*  
8 Создать переменную movie класса Movie и для каждой введенной строки(ключа):  
8.1 получить объект используя MovieFactory.getMovie и присвоить его переменной movie  
8.2 вывести на экран movie.getClass().getSimpleName()  
 \*/  
  
 }  
  
 static class MovieFactory  
 {  
  
 static Movie getMovie(String key)  
 {  
 Movie movie = null;  
  
 //создание объекта SoapOpera (мыльная опера) для ключа "soapOpera"  
 if ("soapOpera".equals(key))  
 {  
 movie = new SoapOpera();  
 }  
  
 //напишите тут ваш код, пункты 5,6  
  
 return movie;  
 }  
 }  
  
 static abstract class Movie  
 {  
 }  
  
 static class SoapOpera extends Movie  
 {  
 }  
  
 //Напишите тут ваши классы, пункт 3  
}

package com.javarush.test.level14.lesson08.home07;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\* Клининговый центр  
Клининговый центр  
1. Реализовать метод cleanAllApartaments.  
Для каждого объекта из apartaments:  
2. Для однокомнатных квартир (Apt1Room) вызвать метод clean1Room.  
т.е. если объект типа Apt1Room, то вызвать у него метод clean1Room.  
3. Для двухкомнатных квартир (Apt2Room) вызвать метод clean2Rooms  
т.е. если объект типа Apt2Room, то вызвать у него метод clean2Rooms.  
4. Для трехкомнатных квартир (Apt3Room) вызвать метод clean3Rooms  
т.е. если объект типа Apt3Room, то вызвать у него метод clean3Rooms.  
\*/  
public class Solution  
{  
 public static void main(String[] args)  
 {  
 List<Apartament> apartaments = new ArrayList<Apartament>();  
 apartaments.add(new Apt1Room());  
 apartaments.add(new Apt2Room());  
 apartaments.add(new Apt3Room());  
  
 *cleanAllApartaments*(apartaments);  
 }  
  
 public static void cleanAllApartaments(List<Apartament> apartaments)  
 {  
 //написать тут вашу реализацию пунктов 1-4  
 }  
  
 static interface Apartament  
 {  
 }  
  
 static class Apt1Room implements Apartament  
 {  
 void clean1Room()  
 {  
 System.*out*.println("1 room is cleaned");  
 }  
 }  
  
 static class Apt2Room implements Apartament  
 {  
 void clean2Rooms()  
 {  
 System.*out*.println("2 rooms are cleaned");  
 }  
 }  
  
 static class Apt3Room implements Apartament  
 {  
 void clean3Rooms()  
 {  
 System.*out*.println("3 rooms are cleaned");  
 }  
 }  
}

package com.javarush.test.level14.lesson08.home08;  
  
/\* Исправление ошибок  
1. Подумать, как связаны интерфейсы Swimable(способен плавать) и Walkable(способен ходить) с классом OceanAnimal(животное океана).  
2. Расставить правильно наследование интерфейсов и класса OceanAnimal.  
3. Подумать, как могут быть связаны классы Orca(Косатка), Whale(Кит), Otter(Выдра) с классом OceanAnimal.  
4. Расставить правильно наследование между классами Orca, Whale, Otter и классом OceanAnimal.  
5. Подумать, какой класс должен реализовать интерфейс Walkable и добавить интерфейc этому классу.  
6. Подумать, какое животное еще не умеет плавать и добавить ему интерфейс Swimable.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args)  
 {  
/\*  
 Swimable animal = new Orca();  
 animal.swim();  
 animal = new Whale();  
 animal.swim();  
 animal = new Otter();  
 animal.swim();  
\*/  
 }  
  
 public static void test(Swimable animal)  
 {  
 animal.swim();  
 }  
  
 static interface Walkable  
 {  
 void walk();  
 }  
  
 static interface Swimable  
 {  
 void swim();  
 }  
  
 static abstract class OceanAnimal  
 {  
 public void swim()  
 {  
 OceanAnimal currentAnimal = (OceanAnimal) getCurrentAnimal();  
 currentAnimal.swimming();  
 }  
  
 private void swimming()  
 {  
 System.*out*.println(getCurrentAnimal().getClass().getSimpleName() + " is swimming");  
 }  
  
 abstract Swimable getCurrentAnimal();  
 }  
  
 static class Orca  
 {  
 }  
  
 static class Whale  
 {  
  
 }  
  
 static class Otter  
 {  
  
 }  
}

package com.javarush.test.level14.lesson08.home09;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\* Валюты  
1. Реализуй метод getAmount в классе Money:  
1.1. Подумай, какого типа нужно создать приватную переменную, если метод getAmount будет ее возвращать.  
1.2. Создай приватную переменную этого типа и верни ее в методе getAmount.  
1.3. В конструкторе присвой ей значение, полученное параметром.  
2. В отдельном файле создай класс Hrivna.  
3. Наследуй класс Hrivna от класса Money.  
4. В классе Hrivna реализуй метод getCurrencyName, который возвращает "HRN".  
5. В отдельном файле создай класс USD.  
6. Наследуй класс USD от класса Money.  
7. В классе USD реализуй метод getCurrencyName, который возвращает "USD".  
8. Подумай, объекты каких классов можно добавить в список(лист) allMoney.  
9. Добавь в конструктор класса Person заполнение листа allMoney объектами всех возможных классов.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args)  
 {  
 Person ivan = new Person("Иван");  
 for (Money money : ivan.getAllMoney())  
 {  
 System.*out*.println(ivan.name + " имеет заначку в размере " + money.getAmount() + " " + money.getCurrencyName());  
 }  
 }  
  
 static class Person  
 {  
 public String name;  
  
 Person(String name)  
 {  
 this.name = name;  
 this.allMoney = new ArrayList<Money>();  
 //Add your code here  
 }  
  
 private List<Money> allMoney;  
  
 public List<Money> getAllMoney()  
 {  
 return allMoney;  
 }  
 }  
}

package com.javarush.test.level14.lesson08.home09;  
  
public abstract class Money  
{  
 public Money(double amount)  
 {  
 }  
  
 public abstract double getAmount();  
  
 public abstract String getCurrencyName();  
}

package com.javarush.test.level14.lesson08.home10;  
  
import java.util.LinkedList;  
import java.util.List;  
  
/\* Исправить 4 ошибки  
Исправить 4 ошибки в конструкторе NotIncapsulatedClass и отрефактори код  
1. В класе NotIncapsulatedClass создать private методы initList(List<Number> list), printListValues, processCastedObjects.  
2. Метод initList должен заполнять значениями входящий параметр list:  
- найди нужный блок кода в конструкторе, в котором list заполняется значениями  
- перенеси его в метод initList  
- верни заполненный list.  
3. Метод printListValues должен принимать параметр list и вывести в консоль все элементы из списка list:  
- метод ничего не возвращает  
- найди нужный блок кода в конструкторе, в котором в цикле из списка list выводятся в консоль все значения  
- перенеси его в метод printListValues  
- исправь 2 ошибки в этом методе.  
4. Метод processCastedObjects:  
- входящий параметр метода имеет тип List<Number> list  
- метод ничего не возвращает  
- найди нужный блок кода в конструкторе, в котором в цикле для каждого объекта из списка list проверяется его тип  
- перенеси этот блок в метод processCastedObjects  
- исправь 2 ошибки в этом методе  
- учти, что для объекта типа Float нужно вывести "Is float value defined? " + [Float\_object].isNaN()  
- учти, что для объекта типа Double нужно вывести "Is double value infinite? " + [Double\_object].isInfinite().  
\*/  
  
public class Solution  
{  
 public static void main(String[] args)  
 {  
 new NotIncapsulatedClass();  
 }  
  
 public static class NotIncapsulatedClass  
 {  
 public NotIncapsulatedClass()  
 {  
 List<Number> list = new LinkedList<Number>();  
 //1  
 list.add(new Double(1000f));  
 list.add(new Double("123e-445632"));  
 list.add(new Float(-90 / -3));  
 list.remove(new Double("123e-445632"));  
  
 //2 - Исправь 2 ошибки  
 for (int i = 0; i <= list.size(); i--)  
 {  
 System.*out*.println(list.get(i));  
 }  
  
 //3  
 for (Number object : list)  
 {  
 //Исправь 2 ошибки  
 if (object instanceof Float)  
 {  
 Double a = (Double) object;  
 System.*out*.println("Is float value defined? " + a.isNaN());  
 } else if (object instanceof Double)  
 {  
 Float a = (Float) object;  
 System.*out*.println("Is double value infinite? " + a.isInfinite());  
 }  
 }  
 }  
 }  
}

package com.javarush.test.level14.lesson08.bonus01;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\* Нашествие эксепшенов  
Заполни массив exceptions 10 различными эксепшенами.  
Первое исключение уже реализовано в методе initExceptions.  
\*/  
  
public class Solution  
{  
 public static List<Exception> *exceptions* = new ArrayList<Exception>();  
  
 public static void main(String[] args)  
 {  
 *initExceptions*();  
  
 for (Exception exception : *exceptions*)  
 {  
 System.*out*.println(exception);  
 }  
 }  
  
 private static void initExceptions()  
 { //it's first exception  
 try  
 {  
 float i = 1 / 0;  
  
 } catch (Exception e)  
 {  
 *exceptions*.add(e);  
 }  
  
 //Add your code here  
  
 }  
}

package com.javarush.test.level14.lesson08.bonus02;  
  
/\* НОД  
Наибольший общий делитель (НОД).  
Ввести с клавиатуры 2 целых положительных числа.  
Вывести в консоль наибольший общий делитель.  
\*/  
  
public class Solution  
{  
 public static void main(String[] args) throws Exception  
 {  
 }  
}

package com.javarush.test.level14.lesson08.bonus03;  
  
/\* Singleton  
Класс является синглтоном (реализует паттерн(шаблон) Singleton), если позволяет создать всего один объект своего типа.  
  
Реализовать Singleton pattern:  
1. Создай класс Singleton в отдельном файле.  
2. Добавь в него статический метод getInstance().  
3. Метод getInstance должен возвращать один и тот же объект класса Singleton при любом вызове метода getInstance.  
4. Подумай, каким образом можно запретить создание других объектов этого класса.  
5. Сделай все конструкторы в классе Singleton приватными (private).  
  
6. В итоге должна быть возможность создать объект (экземпляр класса) ТОЛЬКО используя метод getInstance.  
\*/  
public class Solution  
{  
 public static void main(String[] args)  
 {  
 }  
}