package com.javarush.test.level15.lesson02.task01;  
  
/\* ООП - Расставить интерфейсы  
1. Добавить все возможные интерфейсы из Movable, Sellable, Discountable в класс Clothes.  
2. Реализовать их методы.  
\*/  
  
public class Solution {  
 public static interface Movable {  
 boolean getAllowedAction(String name);  
 }  
  
 public static interface Sellable {  
 Object getAllowedAction(String name);  
 }  
  
 public static interface Discountable {  
 Object getAllowedAction();  
 }  
  
 public static class Clothes {  
 }  
}

package com.javarush.test.level15.lesson02.task02;  
  
/\* ООП - наследование животных  
1. Создать public static класс Goose(Гусь).  
2. Создать public static класс Dragon(Дракон).  
3. Унаследовать класс Goose от BigAnimal или SmallAnimal, подумать, какой логически больше подходит.  
4. Унаследовать класс Dragon от BigAnimal или SmallAnimal, подумать, какой логически больше подходит.  
5. В классах Goose и Dragon переопределить метод String getSize(), расширить видимость до максимальной.  
6. В классе Goose метод getSize должен возвращать строку "Гусь маленький, " + [getSize родительского класса].  
7. В классе Dragon метод getSize должен возвращать строку "Дракон большой, " + [getSize родительского класса].  
\*/  
  
public class Solution {  
 //добавьте классы Goose и Dragon тут  
  
 public static class BigAnimal {  
 protected String getSize() {  
 return "как динозавр";  
 }  
 }  
  
 public static class SmallAnimal {  
 String getSize() {  
 return "как кошка";  
 }  
 }  
}

package com.javarush.test.level15.lesson02.task03;  
  
/\* ООП - машинки  
1. Для вывода использовать можно только переменные из класса Constants.  
2. В классе Ferrari реализуйте метод printlnDesire, чтобы он выводил на экран "Я хочу ездить на Феррари".  
3. В классе Lanos реализуйте метод printlnDesire, чтобы он выводил на экран "Я хочу ездить на Ланосе".  
4. Создайте public static класс LuxuriousCar(РоскошнаяМашина).  
5. Создайте public static класс CheapCar(ДешеваяМашина).  
6. Унаследуйте Ferrari и Lanos от CheapCar и LuxuriousCar, подумайте, какой класс для кого.  
7. В классе LuxuriousCar реализуйте метод printlnDesire, чтобы он выводил на экран "Я хочу ездить на роскошной машине".  
8. В классе CheapCar реализуйте метод printlnDesire, чтобы он выводил на экран "Я хочу ездить на дешевой машине".  
9. В класах LuxuriousCar и CheapCar для метода printlnDesire расставьте различными способами модификаторы доступа так,  
чтобы в классах Ferrari и Lanos выполнялось расширение видимости.  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 new Solution.LuxuriousCar().printlnDesire();  
 new Solution.CheapCar().printlnDesire();  
 new Solution.Ferrari().printlnDesire();  
 new Solution.Lanos().printlnDesire();  
 }  
  
 public static class Ferrari {  
 public void printlnDesire() {  
 //add your code here  
 }  
 }  
  
 public static class Lanos {  
 public void printlnDesire() {  
 //add your code here  
 }  
 }  
  
 public static class Constants {  
 public static String *WANT\_STRING* = "Я хочу ездить на ";  
 public static String *LUXURIOUS\_CAR* = "роскошной машине";  
 public static String *CHEAP\_CAR* = "дешевой машине";  
 public static String *FERRARY\_NAME* = "Феррари";  
 public static String *LANOS\_NAME* = "Ланосе";  
 }  
}

package com.javarush.test.level15.lesson02.task04;  
  
/\* ООП - книги  
1. Создайте public static класс MarkTwainBook, который наследуется от Book. Имя автора [Mark Twain]. Параметр конструктора - имя книги.  
2. В классе MarkTwainBook реализуйте все абстрактные методы.  
3. Для метода getBook расширьте тип возвращаемого результата.  
4. Создайте по аналогии AgathaChristieBook. Имя автора [Agatha Christie].  
5. В классе Book реализуйте тело метода getOutputByBookType так, чтобы он возвращал:  
5.1. agathaChristieOutput для книг Агаты Кристи;  
5.2. markTwainOutput для книг Марка Твена.  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 List<Book> books = new LinkedList<Book>();  
 books.add(new MarkTwainBook("Tom Sawyer"));  
 books.add(new AgathaChristieBook("Hercule Poirot"));  
 System.*out*.println(books);  
 }  
  
 public abstract static class Book {  
 private String author;  
  
 public Book(String author) {  
 this.author = author;  
 }  
  
 public abstract Book getBook();  
  
 public abstract String getName();  
  
 private String getOutputByBookType() {  
 String agathaChristieOutput = author + ", " + getBook().getName() + " is a detective";  
 String markTwainOutput = getBook().getName() + " book was written by " + author;  
  
 String output = "output";  
 //Add your code here  
  
 return output;  
 }  
  
 public String toString() {  
 return getOutputByBookType();  
 }  
 }  
}

package com.javarush.test.level15.lesson02.task05;  
  
import java.util.ArrayList;  
import java.util.List;  
  
/\* ООП - исправь ошибки в наследовании  
Исправь метод containsBones и всю связанную с ним логику так, чтобы:  
1. Поведение программы осталось прежним, т.е. она должна выдавать то же самое, что и выдает сейчас  
2. Метод containsBones должен возвращать тип Object и значение "Yes" вместо true, "No" вместо false  
\*/  
  
public class Solution {  
 public static interface Alive {  
 boolean containsBones();  
 }  
  
 public static class BodyPart implements Alive {  
 private String name;  
  
 public BodyPart(String name) {  
 this.name = name;  
 }  
  
 public boolean containsBones() {  
 return true;  
 }  
  
 public String toString() {  
 return containsBones() ? name + " содержит кости" : name + " не содержит кости";  
 }  
 }  
  
 public static class Finger extends BodyPart {  
 private boolean isFoot;  
 public Finger(String name, boolean isFoot) {  
 super(name);  
 this.isFoot = isFoot;  
 }  
  
 public boolean containsBones() {  
 return super.containsBones() && !isFoot;  
 }  
 }  
 public static void main(String[] args)  
 {  
 *printlnFingers*();  
 *printlnBodyParts*();  
 *printlnAlives*();  
 }  
  
 private static void printlnAlives() {  
 System.*out*.println(new BodyPart("Рука").containsBones());  
 }  
  
 private static void printlnBodyParts() {  
 List<BodyPart> bodyParts = new ArrayList<BodyPart>(5);  
 bodyParts.add(new BodyPart("Рука"));  
 bodyParts.add(new BodyPart("Нога"));  
 bodyParts.add(new BodyPart("Голова"));  
 bodyParts.add(new BodyPart("Тело"));  
 System.*out*.println(bodyParts.toString());  
 }  
  
 private static void printlnFingers() {  
 List<Finger> fingers = new ArrayList<Finger>(5);  
 fingers.add(new Finger("Большой", true));  
 fingers.add(new Finger("Указательный", true));  
 fingers.add(new Finger("Средний", true));  
 fingers.add(new Finger("Безымянный", false));  
 fingers.add(new Finger("Мизинец", true));  
 System.*out*.println(fingers.toString());  
 }  
}

package com.javarush.test.level15.lesson04.task01;  
  
/\* Что-то лишнее  
1. Программа должна выводить следующее:  
Это double  
Это Object  
Это double  
Это Integer  
Это double  
  
2 Удалите реализации всех лишних методов  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 *print*((short) 1);  
 *print*((Number) 1);  
 *print*(1);  
 *print*((Integer) 1);  
 *print*((int) 1);  
 }  
  
 public static void print(Integer i) {  
 System.*out*.println("Это Integer");  
 }  
  
 public static void print(int i) {  
 System.*out*.println("Это Integer");  
 }  
  
 public static void print(Short i) {  
 System.*out*.println("Это Object");  
 }  
  
 public static void print(Object i) {  
 System.*out*.println("Это Object");  
 }  
  
 public static void print(double i) {  
 System.*out*.println("Это double");  
 }  
  
 public static void print(Double i) {  
 System.*out*.println("Это double");  
 }  
  
 public static void print(float i) {  
 System.*out*.println("Это Double");  
 }  
}

package com.javarush.test.level15.lesson04.task02;  
  
/\* ООП - Перегрузка  
Перегрузите метод printMatrix 8 различными способами. В итоге должно получиться 10 различных методов printMatrix.  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 *printMatrix*(2, 3, "8");  
 }  
  
 public static void printMatrix(int m, int n, String value) {  
 System.*out*.println("Заполняем объектами String");  
 *printMatrix*(m, n, (Object) value);  
 }  
  
 public static void printMatrix(int m, int n, Object value) {  
 for (int i = 0; i < m; i++) {  
 for (int j = 0; j < n; j++) {  
 System.*out*.print(value);  
 }  
 System.*out*.println();  
 }  
 }  
}

package com.javarush.test.level15.lesson04.task03;  
  
/\* ООП - Перегрузка - убираем лишнее  
1. Подумайте, какая из реализаций метода print будет вызвана.  
2. Удалите все лишние реализации метода print.  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 *print*(1);  
 }  
  
 public static void print(byte b) {  
 System.*out*.println("Я буду Java прогером!");  
 }  
  
 public static void print(double d) {  
 System.*out*.println("Я буду Java прогером!");  
 }  
  
 public static void print(long l) {  
 System.*out*.println("Я буду Java прогером!");  
 }  
  
 public static void print(float f) {  
 System.*out*.println("Я буду Java прогером!");  
 }  
  
 public static void print(char c) {  
 System.*out*.println("Я буду Java прогером!");  
 }  
}

package com.javarush.test.level15.lesson04.task04;  
  
/\* Мужчина или женщина?  
1. В методе main создать 2 человека man и woman. man с типом Man, woman с типом Woman.  
2. Изменить метод printName так, чтобы он выполнялся для man и woman.  
3. Реализация метода printName должна быть одна.  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 man = new Man();  
 woman = new Woman();  
  
 *printName*(man);  
 *printName*(woman);  
 }  
  
 public static void printName(Man man) {  
  
 }  
  
 public static class Human {  
  
 }  
  
 public static class Man extends Human {  
  
 }  
  
 public static class Woman extends Human {  
  
 }  
}

package com.javarush.test.level15.lesson04.task05;  
  
/\* Все лишнее - прочь!  
Убрать в методе main лишние строки, для которых метод add нереализован.  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 *add*((short) 1, 2f);  
 *add*(1, 2);  
 *add*(2d, 2);  
 add("1",2d);  
 *add*((byte) 1, 2d);  
 }  
  
 public static void add(int i, int j) {  
 System.*out*.println("Integer addition");  
 }  
  
 public static void add(int i, double j) {  
 System.*out*.println("Integer and double addition");  
 }  
  
 public static void add(double i, double j) {  
 System.*out*.println("Double addition");  
 }  
}

package com.javarush.test.level15.lesson06.task01;  
  
import java.io.Serializable;  
  
/\* Максимально простой код 1  
Упрости код - убери все наследования и реализации, которые и так будут добавлены автоматически при компиляции  
PS: Взаимосвязь между объектами me и zapp - Has-a (использует): http://ru.wikipedia.org/wiki/Has-a  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 JuniorJavaDev me = new JuniorJavaDev();  
 System.*out*.println(me.askHubert("What do you think about level15.lesson06.task01?"));  
 System.*out*.println(me.askZapp("When will be the next update?"));  
 }  
  
 public interface SpecificSerializable extends Serializable {  
 }  
  
 public static class JavaDev extends Object implements SpecificSerializable {  
 String answerQuestion(String question) {  
 return String.*format*("I'll be thinking of [%s]", question);  
 }  
 }  
  
 public static class JuniorJavaDev extends Object, JavaDev implements SpecificSerializable {  
 JavaDev zapp = new JavaDev();  
 JavaDev hubert = new JavaDev();  
  
 String askZapp(String question) {  
 return zapp.answerQuestion(question);  
 }  
  
 String askHubert(String question) {  
 return hubert.answerQuestion(question);  
 }  
 }  
}

package com.javarush.test.level15.lesson06.task02;  
  
/\* Максимально простой код 2  
Посмотри, что, связанное с конструкторами, может быть добавлено при компиляции, но в этой программе уже присутствует.  
Упрости код - удали все конструкторы и вызовы конструкторов супер-классов, которые создаются и добавляются автоматически.  
  
PS: Взаимосвязь между объектами классов NakedCat и NormalCat, SiamCat - Is-a (наследование): http://en.wikipedia.org/wiki/Is-a  
http://cs7002.vk.me/c7007/v7007577/1411a/\_dAiEola310.jpg  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 SiamCat simka = new SiamCat("Simka");  
 NakedCat nakedSimka = simka.shave();  
 }  
  
 public static class NakedCat {  
 public NakedCat() {  
 super();  
 }  
 }  
  
 public static class NormalCat extends NakedCat {  
 public NormalCat() {  
 super();  
 }  
  
 public NormalCat(String name) {  
 System.*out*.println("My name is " + name);  
 }  
  
 public NakedCat shave() {  
 return this;  
 }  
 }  
  
 public static class SiamCat extends NormalCat {  
 public SiamCat(String name) {  
 super(name);  
 }  
 }  
}

package com.javarush.test.level15.lesson06.task03;  
  
/\* Максимально простой код 3  
Упрости код:  
- убери всё то, что будет генерироваться автоматически при компиляции  
- убери все наследования классов, которые и так будут добавлены автоматически при компиляции  
- убери все конструкторы, которые создаются и добавляются автоматически.  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 }  
  
 public interface Runnable {  
  
 }  
  
 public class Machine extends Object implements Runnable {  
 public Machine() {  
 super();  
 }  
 }  
  
 public class Car extends Machine implements Runnable {  
 public Car() {  
 super();  
 }  
 }  
}

package com.javarush.test.level15.lesson09.task01;  
  
import java.util.HashMap;  
import java.util.Map;  
  
/\* Статики 1  
В статическом блоке инициализировать labels 5 различными парами.  
\*/  
  
public class Solution {  
 public static Map<Double, String> *labels* = new HashMap<Double, String>();  
  
 public static void main(String[] args) {  
 System.*out*.println(*labels*);  
 }  
}

package com.javarush.test.level15.lesson09.task02;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
  
/\* Статики 2  
1. В статическом блоке считайте две переменные с консоли А и В с типом int.  
2. Не забыть про IOException, который надо обработать в блоке catch.  
3. Закрыть поток ввода методом close().  
\*/  
  
public class Solution {  
 public static int *A*;  
 public static int *B*;  
  
 public static final int *MIN* = *min*(*A*, *B*);  
  
 public static void main(String[] args) {  
 System.*out*.println(*MIN*);  
 }  
  
 public static int min(int a, int b) {  
 return a < b ? a : b;  
 }  
}

package com.javarush.test.level15.lesson09.task03;  
  
/\* Статики 3  
1. Создать 7 public полей класса. Убедитесь, что они инициализируются дефолтными значениями.  
intVar с типом int  
doubleVar с типом double  
DoubleVar с типом Double  
booleanVar с типом boolean  
ObjectVar с типом Object  
ExceptionVar с типом Exception  
StringVar с типом String  
2. В методе main вывести их значения в заданном порядке.  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 }  
}

package com.javarush.test.level15.lesson09.task04;  
  
/\* Статики и исключения  
В статическом блоке выбросьте RuntimeException  
В результате класс не загрузится, и вы увидите сообщение об ошибке вместо значения переменной B  
  
Exception in thread "main" java.lang.ExceptionInInitializerError  
at java.lang.Class.forName0(Native Method)  
at java.lang.Class.forName(Class.java:186)  
at com.intellij.rt.execution.application.AppMain.main(AppMain.java:113)  
Caused by: java.lang.RuntimeException:  
at com.javarush.test.level15.lesson09.task04.Solution.clinit(Solution.java:22)  
  
Hint: Нужно погуглить причину, если получилось следующее:  
java: initializer must be able to complete normally  
java: unreachable statement  
\*/  
  
public class Solution {  
 public static int *A* = 0;  
  
 static {  
 //throw an exception here - выбросьте эксепшн тут  
 }  
  
 public static int *B* = 5;  
  
 public static void main(String[] args) {  
 System.*out*.println(*B*);  
 }  
}

package com.javarush.test.level15.lesson09.task05;  
  
/\* Статики 4  
Внутри статического блока:  
1. Создайте класс Cat  
2. Внутри Cat создайте поле класса String name = "Пушок"  
3. Создайте объект класса Cat - myCat  
4. Присвойте статическому объекту Object cat объект, созданный в п.3  
5. Выведите в консоль myCat.name  
\*/  
  
public class Solution {  
 public static Object *cat*;  
}

package com.javarush.test.level15.lesson12.home01;  
  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.util.ArrayList;  
  
/\* Разные методы для разных типов  
1. Считать с консоли данные, пока не введено слово "exit".  
2. Для каждого значения, кроме "exit", вызвать метод print. Если значение:  
2.1. содержит точку '.', то вызвать метод print для Double;  
2.2. больше нуля, но меньше 128, то вызвать метод print для short;  
2.3. больше либо равно 128, то вызвать метод print для Integer;  
2.4. иначе, вызвать метод print для String.  
\*/  
  
public class Solution {  
 public static void main(String[] args) throws IOException {  
 //напиште тут ваш код  
 }  
  
 public static void print(Double value) {  
 System.*out*.println("Это тип Double, значение " + value);  
 }  
  
 public static void print(String value) {  
 System.*out*.println("Это тип String, значение " + value);  
 }  
  
 public static void print(short value) {  
 System.*out*.println("Это тип short, значение " + value);  
 }  
  
 public static void print(Integer value) {  
 System.*out*.println("Это тип Integer, значение " + value);  
 }  
}

package com.javarush.test.level15.lesson12.home02;  
  
/\* Тренировка мозга  
Найти логическую ошибку: метод doAction в реализации интерфейса Movable должен выдавать "moving".  
Исправьте ошибку, при необходимости измените (отрефакторите) интерфейс Movable.  
Результат вывода в консоль должен быть:  
flying  
moving  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 Duck duck = new Duck();  
 Util.*fly*(duck);  
 Util.*move*(duck);  
 }  
  
 public static class Duck implements Flyable, Movable {  
 @Override  
 public void doAction() {  
 System.*out*.println("flying");  
 }  
 }  
  
 public static class Util {  
 static void fly(Flyable animal) {  
 animal.doAction();  
 }  
  
 static void move(Movable animal) {  
 animal.doAction();  
 }  
 }  
  
 public static interface Flyable {  
 void doAction();  
 }  
  
 public static interface Movable {  
 void doAction();  
 }  
}

package com.javarush.test.level15.lesson12.home03;  
  
import java.math.BigDecimal;  
  
/\* ООП - перегрузка  
1.В классе Tree перегрузи метод info(Object s) два раза так, чтобы получилось три метода info(Object s), info(Number s), info(String s).  
1.1. Разберись в методе info(Object s). Сделай по аналогии функционал новых методов.  
1.2. Например, для метода info(Number s) результат может быть таким "Дерево № 123 , метод Number, параметр Short".  
2. В блоке 2 должен вызываться метод info(Object s).  
3. В блоке 3 должен вызываться метод info(Number s).  
4. В блоке 4 должен вызываться метод info(String s).  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 //Блок 2.  
 //Вызов для Object (задание 2)  
 new Tree().info((Object)new Integer("4"));  
 new Tree().info((Object)new Short("4"));  
 new Tree().info((Object)new BigDecimal("4"));  
  
 //Блок 3.  
 //Вызов для Number (Задание 3)  
 new Tree().info(new Integer("4"));  
 new Tree().info(new Short("4"));  
 new Tree().info(new BigDecimal("4"));  
  
 //Блок 4.  
 //Вызов для String (Задание 4)  
 new Tree().info(new String("4"));  
 new Tree().info(new Integer("4").toString());  
 new Tree().info(new Short("4").toString());  
 new Tree().info(new BigDecimal("4").toString());  
 }  
}

package com.javarush.test.level15.lesson12.home03;  
  
public class Tree {  
 public static int *globalNumber*;  
 public int number;  
  
 public Tree() {  
 this.number = ++*globalNumber*;  
 }  
  
 public void info(Object s) {  
 System.*out*.println(String.*format*("Дерево № %d , метод Object, параметр %s", number, s.getClass().getSimpleName()));  
 }  
}

package com.javarush.test.level15.lesson12.home04;  
  
/\* Закрепляем Singleton pattern  
1. Найти в гугле пример для - Singleton pattern Lazy initialization.  
2. По образу и подобию в отдельных файлах создать три синглтон класса Sun, Moon, Earth.  
3. Реализовать интерфейс Planet для классов Sun, Moon, Earth.  
4. В статическом блоке класса Solution вызвать метод readKeyFromConsoleAndInitPlanet.  
5. Реализовать функционал метода readKeyFromConsoleAndInitPlanet:  
5.1. С консоли считать один параметр типа String.  
5.2. Если параметр равен одной из констант интерфейса Planet, то создать соответствующий объект и присвоить его Planet thePlanet, иначе обнулить Planet thePlanet.  
5.3. Сравнивать введенный параметр можно только с константами из Planet, нельзя создавать свои строки.  
\*/  
  
public class Solution {  
 public static Planet *thePlanet*;  
  
 //add static block here - добавьте статический блок тут  
  
 public static void readKeyFromConsoleAndInitPlanet() {  
 // implement step #5 here - реализуйте задание №5 тут  
 *thePlanet* = Sun.getInstance();  
 }  
}

package com.javarush.test.level15.lesson12.home04;  
  
public interface Planet {  
 static String *SUN* = "sun";  
 static String *MOON* = "moon";  
 static String *EARTH* = "earth";  
}

package com.javarush.test.level15.lesson12.home05;  
  
/\* Перегрузка конструкторов  
1. В классе Solution создайте по 3 конструктора для каждого модификатора доступа.  
2. В отдельном файле унаследуйте класс SubSolution от класса Solution.  
3. Внутри класса SubSolution создайте конструкторы командой Alt+Insert -> Constructors.  
4. Исправьте модификаторы доступа конструкторов в SubSolution так, чтобы они соответствовали конструкторам класса Solution.  
\*/  
  
public class Solution {  
}

package com.javarush.test.level15.lesson12.home06;  
  
/\* Порядок загрузки переменных  
Разобраться, что в какой последовательности инициализируется.  
Исправить порядок инициализации данных так, чтобы результат был следующим:  
static void init()  
Static block  
public static void main  
non-static block  
static void printAllFields  
0  
null  
Solution constructor  
static void printAllFields  
6  
First name  
\*/  
  
public class Solution {  
 static {  
 System.*out*.println("Static block");  
 }  
  
 {  
 System.*out*.println("non-static block");  
 *printAllFields*(this);  
 }  
  
 public int i = 6;  
  
 public String name = "First name";  
 static {  
 *init*();  
 }  
  
 public Solution() {  
 System.*out*.println("Solution constructor");  
 *printAllFields*(this);  
 }  
  
 public static void init() {  
 System.*out*.println("static void init()");  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println("public static void main");  
 Solution s = new Solution();  
 }  
  
 public static void printAllFields(Solution obj) {  
 System.*out*.println("static void printAllFields");  
 System.*out*.println(obj.name);  
 System.*out*.println(obj.i);  
 }  
}

package com.javarush.test.level15.lesson12.home07;  
  
import java.io.BufferedReader;  
import java.io.FileNotFoundException;  
import java.io.FileReader;  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.List;  
  
/\* Файл в статическом блоке  
1. Инициализируй константу Constants.FILE\_NAME полным путем к файлу с данными, который содержит несколько строк.  
2. В статическом блоке считай из файла с именем Constants.FILE\_NAME все строки и добавь их по-отдельности в List lines.  
3. Закрой поток ввода методом close().  
\*/  
  
public class Solution {  
 public static List<String> *lines* = new ArrayList<String>();  
  
 public static void main(String[] args) {  
 System.*out*.println(*lines*);  
 }  
}

package com.javarush.test.level15.lesson12.home07;  
  
public class Constants {  
 public static String *FILE\_NAME* = /\* add your path to source file here\*/;  
}

package com.javarush.test.level15.lesson12.home08;  
  
/\* Дебаг, дебаг, и еще раз дебаг  
Программа выводит 0 9, а должна 6 9. Найди одну! ошибку и исправь.  
Используй дебаг. Для этого поставь breakpoint-ы(Ctrl+F8), потом зайди в меню Run -> Debug.  
F9 - выполнение кода до следующего breakpoint-а  
F8 - переход к следующей строке кода  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
   
 new B(6);  
  
 }  
  
 public static class A {  
 private int f1 = 7;  
  
 public A(int f1) {  
 this.f1 = f1;  
 initialize();  
 }  
  
 protected void initialize() {  
  
 System.*out*.println(f1);  
 }  
 }  
  
 public static class B extends A {  
 protected int f1 = 3;  
  
 public B(int f1) {  
 super(f1);  
 this.f1 += f1;  
 initialize();  
 }  
  
 protected void initialize() {  
 System.*out*.println(f1);  
 }  
 }  
}

package com.javarush.test.level15.lesson12.home09;  
  
/\* Парсер реквестов  
Считать с консоли URl ссылку.  
Вывести на экран через пробел список всех параметров (Параметры идут после ? и разделяются &, например, lvl=15).  
URL содержит минимум 1 параметр.  
Если присутствует параметр obj, то передать его значение в нужный метод alert.  
alert(double value) - для чисел (дробные числа разделяются точкой)  
alert(String value) - для строк  
  
Пример 1  
Ввод:  
http://javarush.ru/alpha/index.html?lvl=15&view&name=Amigo  
Вывод:  
lvl view name  
  
Пример 2  
Ввод:  
http://javarush.ru/alpha/index.html?obj=3.14&name=Amigo  
Вывод:  
obj name  
double 3.14  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 //add your code here  
 }  
  
 public static void alert(double value) {  
 System.*out*.println("double " + value);  
 }  
  
 public static void alert(String value) {  
 System.*out*.println("String " + value);  
 }  
}

package com.javarush.test.level15.lesson12.home10;  
  
/\* ООП - наследование  
Исправить класс Hrivna так, чтоб избежать ошибку StackOverflowError, класс Money менять нельзя.  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 System.*out*.println(new Hrivna().getAmount());  
 }  
  
 public static abstract class Money {  
 abstract Money getMoney();  
  
 public Object getAmount() {  
 return getMoney().getAmount();  
 }  
 }  
  
 //add your code below - добавь код ниже  
 public static class Hrivna extends Money {  
 public double amount = 123d;  
  
 public Hrivna getMoney() {  
 return this;  
 }  
 }  
}

package com.javarush.test.level15.lesson12.bonus01;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
  
/\* Осваивание статического блока  
1. В отдельных файлах создать классы Plane и Helicopter, реализующие интерфейс Flyable.  
2. Класс Plane должен иметь 1 конструктор с параметром int - количество перевозимых пассажиров.  
3. В статическом методе reset класса Solution:  
3.1. Считать с консоли параметр типа String. Параметр может быть "plane" или "helicopter".  
3.2. Если параметр равен "helicopter", то статическому объекту Flyable result присвоить объект класса Helicopter.  
3.3. Если параметр равен "plane", то считать второй параметр типа int, статическому объекту Flyable result присвоить объект класса Plane.  
4. В статическом блоке инициализировать Flyable result вызвав метод reset.  
5. Закрыть поток ввода методом close().  
\*/  
  
public class Solution {  
 static {  
 //add your code here - добавьте код тут  
 }  
  
 public static Flyable *result*;  
  
 public static void reset() {  
 //add your code here - добавьте код тут  
 }  
}

package com.javarush.test.level15.lesson12.bonus01;  
  
public interface Flyable {  
 void fly();  
}

package com.javarush.test.level15.lesson12.bonus02;  
  
/\* Template pattern  
1. В отдельном файле создать класс DrinkMaker с тремя абстрактными методами:  
- void getRightCup() - выбрать подходящую чашку  
- void putIngredient() - положить ингредиенты  
- void pour() - залить жидкостью  
2. В классе DrinkMaker создать и реализовать метод void makeDrink(), который готовит напиток в такой последовательности: выбирает чашку, кладет ингредиенты, заливает жидкостью.  
3. В отдельных файлах создать классы LatteMaker и TeaMaker, которые наследуются от DrinkMaker.  
4. Распредели следующие фразы между всеми методами в классах LatteMaker и TeaMaker, различные фразы для различных методов.  
5. Каждый метод должен выводить в консоль свою фразу не пересекаясь с другими методами.  
6. Фразы:  
"Заливаем водой"  
"Берем чашку для латте"  
"Насыпаем чай"  
"Берем чашку для чая"  
"Заливаем молоком с пенкой"  
"Делаем кофе"  
\*/  
  
public class Solution {  
 public static void main(String[] args) {  
 }  
}

package com.javarush.test.level15.lesson12.bonus03;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
  
/\* Факториал  
Написать метод, который вычисляет факториал - произведение всех чисел от 1 до введенного числа включая его.  
Пример: 4! = factorial(4) = 1\*2\*3\*4 = 24  
1. Ввести с консоли число меньше либо равно 150.  
2. Реализовать функцию factorial.  
3. Если введенное число меньше 0, то вывести 0.  
0! = 1  
\*/  
  
public class Solution {  
 public static void main(String[] args) throws IOException {  
 BufferedReader reader = new BufferedReader(new InputStreamReader(System.*in*));  
  
 int input = Integer.*parseInt*(reader.readLine());  
 reader.close();  
  
 System.*out*.println(*factorial*(input));  
 }  
  
 public static String factorial(int n) {  
 //add your code here  
  
 return "";  
 }  
}