package com.javarush.task.task21.task2101;  
  
/\*   
  
Определяем адрес сети  
1. Даны IP-адрес и маска подсети, необходимо вычислить адрес сети — реализуй метод getNetAddress.  
Используйте операцию поразрядной конъюнкции (логическое И).  
  
Пример:  
IP-адрес: 11000000 10101000 00000001 00000010 (192.168.1.2)  
Маска подсети: 11111111 11111111 11111110 00000000 (255.255.254.0)  
Адрес сети: 11000000 10101000 00000000 00000000 (192.168.0.0)  
  
2. Реализовать метод print, который выведет в консоль данные в двоичном коде. Для IP-адреса(192.168.1.2)  
должна быть выведена строка «11000000 10101000 00000001 00000010»  
3. Метод main не участвует в тестировании  
\*/  
public class Solution {  
 public static void main(String[] args) {  
 byte[] ip = new byte[]{(byte) 192, (byte) 168, 1, 2};  
 byte[] mask = new byte[]{(byte) 255, (byte) 255, (byte) 254, 0};  
 byte[] netAddress = *getNetAddress*(ip, mask);  
 *print*(ip); //11000000 10101000 00000001 00000010  
 *print*(mask); //11111111 11111111 11111110 00000000  
 *print*(netAddress); //11000000 10101000 00000000 00000000  
 }  
  
 public static byte[] getNetAddress(byte[] ip, byte[] mask) {  
 return new byte[4];  
 }  
  
 public static void print(byte[] bytes) {  
 }  
}

package com.javarush.task.task21.task2102;  
  
import java.lang.reflect.Method;  
import java.lang.reflect.Modifier;  
  
/\*   
Сравниваем модификаторы  
Реализовать логику метода isAllModifiersContainSpecificModifier, который проверяет, содержит ли переданный параметр allModifiers значение конкретного модификатора specificModifier.  
  
P.S. Перед выполнением задания ознакомься с классом Modifier и реализацией методов isPublic, isStatic и т.п.  
\*/  
public class Solution {  
 public static void main(String[] args) {  
 int modifiersOfThisClass = Solution.class.getModifiers();  
 System.*out*.println(*isAllModifiersContainSpecificModifier*(modifiersOfThisClass, Modifier.*PUBLIC*)); //true  
 System.*out*.println(*isAllModifiersContainSpecificModifier*(modifiersOfThisClass, Modifier.*STATIC*)); //false  
  
 int modifiersOfMethod = *getMainMethod*().getModifiers();  
 System.*out*.println(*isAllModifiersContainSpecificModifier*(modifiersOfMethod, Modifier.*STATIC*)); //true  
 }  
  
 public static boolean isAllModifiersContainSpecificModifier(int allModifiers, int specificModifier) {  
 return false;  
 }  
  
 private static Method getMainMethod() {  
 Method[] methods = Solution.class.getDeclaredMethods();  
 for (Method method : methods) {  
 if (method.getName().equalsIgnoreCase("main")) return method;  
 }  
  
 return null;  
 }  
}

package com.javarush.task.task21.task2103;  
  
/\*   
Все гениальное - просто!  
Упростить. Переменные не переименовывать, комментарии не оставлять.  
\*/  
public class Solution {  
 public static boolean calculate(boolean a, boolean b, boolean c, boolean d) {  
 return (a && b && c && !d) || (!a && c) || (!b && c) || (c && d);

}  
  
 public static void main(String[] args) {  
  
 }  
}

package com.javarush.task.task21.task2104;  
  
import java.util.HashSet;  
import java.util.Set;  
  
/\*   
Equals and HashCode  
В классе Solution исправить пару методов equals/hashCode в соответствии с правилами реализации этих методов(детали уточни у своего любимого поисковика).  
Обе строки first и last должны принимать участие в сравнении с помощью метода equals и вычислении hashcode.  
Метод main не участвует в тестировании.  
\*/  
public class Solution {  
 private final String first, last;  
  
 public Solution(String first, String last) {  
 this.first = first;  
 this.last = last;  
 }  
  
 public boolean equals(Solution n) {  
 return n.first.equals(first) && n.last.equals(last);  
 }  
  
 public int hashCode() {  
 return 31 \* first.hashCode() + last.hashCode();  
 }  
  
 public static void main(String[] args) {  
 Set<Solution> s = new HashSet<>();  
 s.add(new Solution("Donald", "Duck"));  
 System.*out*.println(s.contains(new Solution("Donald", "Duck")));  
 }  
}

package com.javarush.task.task21.task2105;  
  
import java.util.HashSet;  
import java.util.Set;  
  
/\*   
Исправить ошибку. Сравнение объектов  
Сравнение объектов Solution не работает должным образом. Найти ошибку и исправить.  
Метод main не участвует в тестировании.  
\*/  
public class Solution {  
 private final String first, last;  
  
 public Solution(String first, String last) {  
 this.first = first;  
 this.last = last;  
 }  
  
 public boolean equals(Object o) {  
 if (!(o instanceof Solution))  
 return false;  
 Solution n = (Solution) o;  
 return n.first.equals(first) && n.last.equals(last);  
 }  
  
 public static void main(String[] args) {  
 Set<Solution> s = new HashSet<>();  
 s.add(new Solution("Mickey", "Mouse"));  
 System.*out*.println(s.contains(new Solution("Mickey", "Mouse")));  
 }  
}

package com.javarush.task.task21.task2107;  
  
import java.util.LinkedHashMap;  
import java.util.Map;  
  
/\*   
Глубокое клонирование карты  
Обеспечь возможность клонирования объекта класса Solution используя глубокое клонирование.  
Данные в карте users также должны быть клонированы.  
Не забудь о методах equals и hashCode для корректного добавления элементов типа User в HashMap.  
\*/  
public class Solution {  
  
 public static void main(String[] args) {  
 Solution solution = new Solution();  
 solution.users.put("Hubert", new User(172, "Hubert"));  
 solution.users.put("Zapp", new User(41, "Zapp"));  
 Solution clone = null;  
 try {  
 clone = solution.clone();  
 System.*out*.println(solution);  
 System.*out*.println(clone);  
  
 System.*out*.println(solution.users);  
 System.*out*.println(clone.users);  
  
 } catch (CloneNotSupportedException e) {  
 e.printStackTrace(System.*err*);  
 }  
 }  
  
 protected Map<String, User> users = new LinkedHashMap();  
  
 public static class User {  
 int age;  
 String name;  
  
 public User(int age, String name) {  
 this.age = age;  
 this.name = name;  
 }  
 }  
}

package com.javarush.task.task21.task2108;  
  
/\*   
Клонирование растений

Класс Plant не должен реализовывать интерфейс Cloneable  
Реализуй механизм глубокого клонирования для Tree.

\*/  
public class Solution {  
 public static void main(String[] args) {  
 Tree tree = new Tree("willow", new String[]{"s1", "s2", "s3", "s4"});  
 Tree clone = null;  
 try {  
 clone = tree.clone();  
 } catch (CloneNotSupportedException e) {  
 e.printStackTrace();  
 }  
  
 System.*out*.println(tree);  
 System.*out*.println(clone);  
  
 System.*out*.println(tree.branches);  
 System.*out*.println(clone.branches);  
 }  
  
 public static class Plant{  
 private String name;  
  
 public Plant(String name) {  
 this.name = name;  
 }  
  
 public String getName() {  
 return name;  
 }  
 }  
  
 public static class Tree extends Plant {  
 private String[] branches;  
  
 public Tree(String name, String[] branches) {  
 super(name);  
 this.branches = branches;  
 }  
  
 public String[] getBranches() {  
 return branches;  
 }  
 }  
}

package com.javarush.task.task21.task2109;  
  
/\*   
Запретить клонирование  
Разреши клонировать класс А  
Запрети клонировать класс B  
Разреши клонировать класс C  
Не забудь о методах equals и hashCode!  
\*/  
public class Solution {  
 public static class A implements Cloneable {  
 private int i;  
 private int j;  
  
 public A(int i, int j) {  
 this.i = i;  
 this.j = j;  
 }  
  
 public int getI() {  
 return i;  
 }  
  
 public int getJ() {  
 return j;  
 }  
 }  
  
 public static class B extends A {  
 private String name;  
  
 public B(int i, int j, String name) {  
 super(i, j);  
 this.name = name;  
 }  
  
 public String getName() {  
 return name;  
 }  
 }  
  
 public static class C extends B {  
 public C(int i, int j, String name) {  
 super(i, j, name);  
 }  
 }  
  
 public static void main(String[] args) {  
  
 }  
}

package com.javarush.task.task21.task2110;  
  
import java.io.BufferedWriter;  
import java.io.IOException;  
import java.nio.charset.Charset;  
import java.nio.charset.StandardCharsets;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.nio.file.Paths;  
import java.util.Enumeration;  
import java.util.zip.ZipEntry;  
import java.util.zip.ZipFile;  
  
/\*   
Рефакторинг методов  
Отрефакторите метод writeZipEntriesToFile в соответствии с java7 try-with-resources.  
\*/  
public class Solution {  
 public static void writeZipEntriesToFile(String zipFileName, String outputFileName) {  
 Charset charset = StandardCharsets.*UTF\_8*;  
 Path outputFilePath = Paths.*get*(outputFileName);  
  
 BufferedWriter writer = null;  
 ZipFile zip = null;  
 try {  
 zip = new ZipFile(zipFileName);  
 writer = Files.*newBufferedWriter*(outputFilePath, charset);  
 String newLine = System.*getProperty*("line.separator");  
 for (Enumeration entries = zip.entries(); entries.hasMoreElements(); ) {  
 // Берем имя файла из архива и записываем его в результирующий файл  
 // Get the entry name and write it to the output file  
 String zipEntryName = ((ZipEntry) entries.nextElement()).getName() + newLine;  
 writer.write(zipEntryName, 0, zipEntryName.length());  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 } finally {  
 if (writer != null) {  
 try {  
 writer.close();  
 } catch (IOException e1) {  
 e1.printStackTrace();  
 }  
 }  
 if (zip != null) {  
 try {  
 zip.close();  
 } catch (IOException e1) {  
 e1.printStackTrace();  
 }  
 }  
 }  
 }  
  
 public static void main(String[] args) {  
  
 }  
}

package com.javarush.task.task21.task2112;

/\*  
AutoCloseable и try-with-resources  
 В классе FakeConnection реализуй интерфейс AutoCloseable, чтобы объекты этого типа можно было использовать в try-with-resources.  
 Метод close() должен выводить на экран фразу «Closing database connection…»  
 В блоке try последовательно вызови методы usefulOperation() и unsupportedOperation().  
  
 Вывод на экран должен быть следующим:  
 Creating database connection...  
 Entering the body of try block.  
 Executing useful operation.  
 Operation is not supported yet!  
 Closing database connection...  
  
 Обрати внимание на то, что ресурсы были освобождены автоматически несмотря на исключение брошенное методом unsupportedOperation.

\*/

public class Solution {  
 public static void main(String[] args) {  
 DBConnectionManager dbConnectionManager = new DBConnectionManager();  
 try (FakeConnection fakeConnection = dbConnectionManager.getFakeConnection()) {  
 System.*out*.println("Entering the body of try block.");  
 } catch (Exception e) {  
 }  
 }  
}

package com.javarush.task.task21.task2112;  
  
public class FakeConnection {  
  
 public FakeConnection() {  
 System.*out*.println("Creating database connection...");  
 }  
  
 public void unsupportedOperation() {  
 System.*out*.println("Operation is not supported yet!");  
 throw new RuntimeException("UnsupportedOperation!");  
 }  
  
 public void usefulOperation() {  
 System.*out*.println("Executing useful operation.");  
 }  
}

package com.javarush.task.task21.task2112;  
  
public class DBConnectionManager {  
 public FakeConnection getFakeConnection() {  
 return new FakeConnection();  
 }  
}

Большая задача

|  |
| --- |
|  |
| Задание 16 |
|  | Теперь уже точно все. |
|  | Добавь вызов метода printWinner в конец метода main. |
|  | Запускай и любуйся своей первой компьютерной игрой :) |
|  |  |
|  |  |
|  | Задание 15 |
|  | Добавим определение победителя. |
|  | В классе Hippodrome сделаем два метода: |
|  | public Horse getWinner() и public void printWinner() |
|  |  |
|  | Метод getWinner должен возвращать лошадь пробежавшую самую большую дистанцию. |
|  | Метод printWinner выводит на экран имя победителя в виде: |
|  | Winner is <NAME>! |
|  | Пример: |
|  | Winner is Lucky! |
|  |  |
|  |  |
|  |  |
|  | Задание 14 |
|  | Запускаем и любуемся. |
|  | У нас каждые полсекунды отображается новый кадр с ситуацией на ипподроме. |
|  | Мышкой уменьши размер консоли так, чтобы был виден только один "кадр" и на том же самом месте. |
|  | Тогда можно наблюдать забег в живую и даже покомментировать: |
|  | - Старт. |
|  | - Гомер неожиданно обходит Лаки. |
|  | - Слевин вырывается вперед. |
|  | - Вперед Лаки! |
|  | - 10 баксов на Слевина. |
|  | - Похоже Лаки сбросил жокея и расслабился. |
|  | - Гомер уверенно вырывается вперед. |
|  | - Неожиданно для всех побеждает Гомер. Вот это номер! |
|  |  |
|  |  |
|  |  |
|  | Задание 13 |
|  | Теперь вернемся к методу print класса Horse. |
|  | Т.к. мы работаем с консолью, то все лошади на бегах будут выглядеть примерно так: |
|  | ........Sleven <- лошадь Слевин |
|  | ....Lucky <- лошадь Лаки |
|  | ..........Gomer <- лошадь Гомер |
|  |  |
|  | Другими словами, в методе print надо вывести на экран строку состоящую из точек и имени лошади. |
|  | Количество точек равно distance, округленному до целого числа. |
|  |  |
|  |  |
|  |  |
|  | Задание 12 |
|  | Осталось совсем немного - дописать класс Horse. |
|  | Каждый ход у лошади будет вызываться метод move. |
|  | Когда у лошади вызывают метод move, лошадь должна пробежать некоторую дистанцию. |
|  | Дистанция зависит от скорости лошади (speed). В самом простом варианте, выглядеть этот метод должен примерно так: |
|  | distance += speed; |
|  |  |
|  | Но, чтобы было интереснее, давай сделаем так, чтобы скорость все время немного менялась. |
|  | Для этого умножь speed на случайное число. |
|  | Случайное число можно получить с помощью метода Math.random() |
|  |  |
|  |  |
|  |  |
|  | Задание 11 |
|  | С классом Hippodrome почти закончили. |
|  | Добавь в конец метода main вызов run(). |
|  |  |
|  | Подсказка: |
|  | run() - это нестатический метод, поэтому вызвать его можно только у объекта. |
|  | А где взять объект? |
|  |  |
|  | Подсказка 2: |
|  | game.run(); |
|  |  |
|  |  |
|  |  |
|  | Задание 10 |
|  | Еще нужно написать метод print класса Hippodrome. |
|  | В нем тоже все просто: в цикле для каждой лошади вызываем ее метод print. |
|  | Ну, и еще выведи после цикла пару пустых строк: System.out.println() - чтобы было красивее. |
|  |  |
|  |  |
|  |  |
|  | Задание 9 |
|  | Теперь вернемся к методам move и print. Начнем с move. |
|  | В методе move класса Hippodrome в цикле у каждой лошади мы вызываем метод move. |
|  |  |
|  | Да ты прав, его еще нет у класса Horse. |
|  | Поэтому в класс Horse надо добавить свой метод move :) |
|  | И метод print, кстати тоже. |
|  | Если я не говорю ничего насчет параметров метода, значит метод без параметров. |
|  | Делай все методы public, если явно не указано обратное. |
|  |  |
|  |  |
|  |  |
|  | Задание 8 |
|  | В методе run сделай цикл от 1 до 100. Это и будет наш забег. |
|  | В теле цикла вызываем сначала move, затем print. |
|  | Чтобы весь цикл не отработал за долю секунды - добавь в него еще Thread.sleep(200); |
|  |  |
|  |  |
|  |  |
|  | Задание 7 |
|  | Но и это еще не все - надо чтобы лошади бежали. |
|  | Добавь в класс Hippodrome методы run, move и print. Без параметров. |
|  | Метод move будет управлять движением всех лошадей. |
|  | Метод print отрисовывать их на экран. |
|  | А метод run - управлять всем этим. |
|  |  |
|  |  |
|  |  |
|  | Задание 6 |
|  | Теперь перейдем к методу main. |
|  | Нам нужно создать в нем объект класса Hippodrome и добавить ему несколько лошадей. |
|  |  |
|  | Для начала: |
|  | Создай статическую переменную game типа Hippodrome. |
|  | Должно получиться что-то вроде: |
|  | public static Hippodrome game; |
|  |  |
|  | В методе main надо: |
|  | а) Создать объект типа Hippodrome и сохранить его в переменную game. |
|  | б) Создать три объекта "лошадь". Имена придумай сам. Начальные скорость у всех лошадей - 3, дистанция - 0. |
|  | в) Добавить созданных лошадей в список лошадей ипподрома (horses). Используйте для этого геттер. |
|  |  |
|  |  |
|  |  |
|  | Задание 5 |
|  | Закончим написание класса "лошадь" |
|  | Добавь конструктор с параметрами (name, speed, distance). |
|  | Добавь getter'ы и setter'ы для всех полей класса Horse. |
|  | Делай все методы public, если явно не указано обратное. |
|  |  |
|  |  |
|  | Задание 4 |
|  | Теперь вернемся к лошадям. У каждой лошади на скачках должны быть известны имя(name) и скорость(speed). |
|  | Наши лошади будут бежать просто определенное время (100 секунд/"шагов"). |
|  | Будем определять победителя, как лошадь, пробежавшая наибольшую дистанции. |
|  |  |
|  | Поэтому нам понадобится хранить расстояние(distance), которое лошадь уже пробежала. |
|  | Добавь в класс Horse переменные name (String), speed (double), distance(double). |
|  |  |
|  |  |
|  |  |
|  | Задание 3 |
|  | Раз это ипподром, то на нем должны быть лошади. |
|  | А, значит наш ипподром должен хранить список всех его лошадей. |
|  | Добавь переменную horses типа ArrayList<Horse> в класс Hippodrome. |
|  | Добавь getter для этого поля. |
|  |  |
|  |  |
|  |  |
|  | Задание 2 |
|  | Любая программа начинается с выполнения метода main. |
|  | Добавь метод main в класс Hippodrome. |
|  |
|  |
|  |
|  | Задание 1 |
|  | Сегодня мы напишем небольшую игру под названием "Ипподром". |
|  | Когда я говорю мы - я имею ввиду тебя. Я же буду работать наставником. |
|  |  |
|  | Для начала нам понадобятся классы "ипподром" и "лошадь". |
|  | Создай классы Hippodrome (ипподром), Horse(лошадь) |
|  |  |