***Condițiile lucrării: (fiecare desrie varianta proprie cu urmatoarele cerinte)***

1. *De descris o ierarhie de clase (minimum 4). Clasa de baza sa fie abstractă cu minimum o funcție abstractă.*
2. *De descris clasa Main cu funcția main în afara ierarhiei.*
3. *De descris minimum 2 interfețe, fiecare sa fie implementata minimum de o clasa din ierarhie.*
4. *Fiecare clasa moștenita trebuie sa aiba metode și cimpuri unice.*
5. *De a apela constructorii claselor părinți cu parametri și fără.*
6. *De a apela metodele simple din clasele părinți.*
7. *De a redefini în fiecare clasa Randomize în constructorul fără parametri.*
8. *De a redefini în fiecare clasa funcția de introducere a datelor de la tastatura.*
9. *De a redefini în fiecare clasa funcția de a înscriere în fișier.*
10. *De a crea minimum cite o variabila, functie, clasa* ***final****.*
11. *De a crea minimum cite o variabila, functie* ***static****.*
12. *De a folosi* ***instanceof*** *și conversia tipurilor obiectelor*
13. *În* ***main*** *de a crea minimum 2 vectori de obiecte de tipul: clasei de baze && interfeței.*
14. *În funcția main, pentru obiectele create, de a chema funcțiile unice din clasele corespunzătoare, și de a afișa obiectele create la ecran.*
15. *De descris in dare de seama conditiile variantei proprie asa, cum este aratat in Exemple de mai jos*
16. *de spus “****URRRRRRAAAAAAA ! ! !”*** *și de prezintat profesorului al doilea laborator. :)*

***Exemple pentru inspiratie ))) :***

***Stea:***

De descris în limbajul Java ierarhia de clase **Stea**  → **Temperatura** → **Culoare** → Magniudine

De descris interfata**.** De implementat interfața în cîteva clase din ierarhie.

**clasa Stea**

*câmpuri:*

* variabilă statică, care arată categoriile = {”stea”, ”temperatura”, ”culoare”, ”magnitudine” }
* câmp de tip întreg, care arată înălțimea Omului
* câmp șir de caractere, care descrie culoarea ochilor Omului
* câmp de tip real, care păstrează greutatea Omului
* câmp de tip logic, care descrie sexul Omului

*funcții:*

* câțiva constructori (implicit (completarea campurilor cu valori aleatoare) și cu parametri)
* funcțiile de acces la câmpurile clasei, și funcțiile de setare a lor
* funcția de afisare a câmpurilor clasei la ecran
* funcția abstracta void saveObjectToFile (String nameFile)

**class Temperatura** extends Stea

*câmpuri:*

* câmp de tip întreg, care arată numărul de variabile a temperaturii
* vector dinamic, în care se păstrează informația despre cele mai mari temperature inregistrate vreodata

*funcții:*

* câțiva constructori (implicit și cu parametri) cu apelarea la constructorii din clasa parinte.
* funcțiile de acces la câmpurile clasei, și funcțiile de setare a lor
* redescrierea a functiei de afisare

**class Culoarea** extends Temperatura

*câmpuri:*

* câmp de tip real, care arată coeficientul
* 2 câmpuri de tip real, care arată IQ Administratorului și salariul Administratorului

*funcții:*

* câțiva constructori (implicit (completarea campurilor cu valori aleatoare) și cu parametri)
* funcțiile de acces la câmpurile clasei, și funcțiile de setare a lor
* redescrierea a functiei de afisare
* funcția de limitare a drepturilor de acces ale Userului, transmis ca parametru în fucnție. (adică nulificarea vectorului Userului)
* funcția de adunare (de tip final) a doi Administratori (noul Administrator va avea media aritmetică dintre lenea ambilor operanzi, media aritmetică IQurilor lor și reuniunea drepturilor de acces ale lor)

**class Șefu\_Mare** extends Admin

*câmpuri:*

* câmp de tip real, care arată numărul de subalterni ai Șefului\_Mare
* vector dimanic, în care se păstrează Iqurile subalternilor Șefului\_Mare

*funcții:*

* câțiva constructori (implicit (completarea campurilor cu valori aleatoare) și cu parametri)
* funcțiile de acces la câmpurile clasei, și funcțiile de setare a lor
* redescrierea a functiei de afisarea *a* câmpurilor clasei la ecran
* funcția, care setează salariul nou unui Administrator concret, transmis ca parametru în funcție (salariul se mărește, dacă raportul dintre lenea si IQ Administratorului e mai mică ca 1, altfel – se micșorează)
* funcția de concediere **Administratorului** concret în cazul, dacă raportul dintre lenea si IQ **Administratorului** e mai mare ca 100 (în același timp schimbînd numărul de subalterni ai Șeful\_Mare)
* funcția de înmulțire :) (noul Șef va avea numărul subalternilor egal cu produsul subalternilor ambilor operanzi)

**interface**

*funcții:*

* funcția de afisare a câmpurilor clasei la ecran

**main**

* de creat vectorul dinamic cu exemplarele ”viețuitoarelor” de oficiu descrise mai sus, dând ca tipul de bază – clasa origine, și de inițializat aceste elemente cu obiectele tuturor claselor moștenite. *Dupa ce de utilizat in main numai obiectele din acest vector.*
* de afișat la ecran câmpurile tuturor obiectelor create
* de limitat drepturile de acces unui user folosind funcțiile unui admin. Drepturile noi de afișat la ecran.
* un Șef trebuie să controleze toți adminii la raportul dintre IQ și lene, și sau să concedieze, sau să facă ceva cu salariul fiecaruia :). Salariile noi ale adminilor de afișat la ecran.
* de ”adunat” câțiva admini. De ”înmulțit” câțiva Șefi.
* de comparat câțiva șefi la deșteptăciunea totală a subalternilor fiecăruia. Rezultatul de afișat la ecran și de a-l salva în fișier textual.

***Exemplu 1:***

De descris în limbajul Java ierarhia de clase **Om**  →  **User** →  **Administrator** →  **Șefu\_Mare**

De descris interfata**.** De implementat interfața în cîteva clase din ierarhie.

**clasa Om**

*câmpuri:*

* variabilă statică, care arată statutul social = {”om”, ”user”, ”admin”, ”boss” }
* câmp de tip întreg, care arată înălțimea Omului
* câmp șir de caractere, care descrie culoarea ochilor Omului
* câmp de tip real, care păstrează greutatea Omului
* câmp de tip logic, care descrie sexul Omului

*funcții:*

* câțiva constructori (implicit (completarea campurilor cu valori aleatoare) și cu parametri)
* funcțiile de acces la câmpurile clasei, și funcțiile de setare a lor
* funcția de afisare a câmpurilor clasei la ecran
* funcția abstracta void saveObjectToFile (String nameFile)

**class User** extends Om

*câmpuri:*

* câmp de tip întreg, care arată numărul de drepturi de acces pentru User
* vector dinamic, în care se păstrează informația despre fiecare drept de acces al Userului

*funcții:*

* câțiva constructori (implicit și cu parametri) cu apelarea la constructorii din clasa parinte.
* funcțiile de acces la câmpurile clasei, și funcțiile de setare a lor
* redescrierea a functiei de afisare

**class Admin** extends User

*câmpuri:*

* câmp de tip real, care arată coeficientul de lene al Adminului(de la 0 la 1)
* 2 câmpuri de tip real, care arată IQ Administratorului și salariul Administratorului

*funcții:*

* câțiva constructori (implicit (completarea campurilor cu valori aleatoare) și cu parametri)
* funcțiile de acces la câmpurile clasei, și funcțiile de setare a lor
* redescrierea a functiei de afisare
* funcția de limitare a drepturilor de acces ale Userului, transmis ca parametru în fucnție. (adică nulificarea vectorului Userului)
* funcția de adunare (de tip final) a doi Administratori (noul Administrator va avea media aritmetică dintre lenea ambilor operanzi, media aritmetică IQurilor lor și reuniunea drepturilor de acces ale lor)

**class Șefu\_Mare** extends Admin

*câmpuri:*

* câmp de tip real, care arată numărul de subalterni ai Șefului\_Mare
* vector dimanic, în care se păstrează Iqurile subalternilor Șefului\_Mare

*funcții:*

* câțiva constructori (implicit (completarea campurilor cu valori aleatoare) și cu parametri)
* funcțiile de acces la câmpurile clasei, și funcțiile de setare a lor
* redescrierea a functiei de afisarea *a* câmpurilor clasei la ecran
* funcția, care setează salariul nou unui Administrator concret, transmis ca parametru în funcție (salariul se mărește, dacă raportul dintre lenea si IQ Administratorului e mai mică ca 1, altfel – se micșorează)
* funcția de concediere **Administratorului** concret în cazul, dacă raportul dintre lenea si IQ **Administratorului** e mai mare ca 100 (în același timp schimbînd numărul de subalterni ai Șeful\_Mare)
* funcția de înmulțire :) (noul Șef va avea numărul subalternilor egal cu produsul subalternilor ambilor operanzi)

**interface**

*funcții:*

* funcția de afisare a câmpurilor clasei la ecran

**main**

* de creat vectorul dinamic cu exemplarele ”viețuitoarelor” de oficiu descrise mai sus, dând ca tipul de bază – clasa origine, și de inițializat aceste elemente cu obiectele tuturor claselor moștenite. *Dupa ce de utilizat in main numai obiectele din acest vector.*
* de afișat la ecran câmpurile tuturor obiectelor create
* de limitat drepturile de acces unui user folosind funcțiile unui admin. Drepturile noi de afișat la ecran.
* un Șef trebuie să controleze toți adminii la raportul dintre IQ și lene, și sau să concedieze, sau să facă ceva cu salariul fiecaruia :). Salariile noi ale adminilor de afișat la ecran.
* de ”adunat” câțiva admini. De ”înmulțit” câțiva Șefi.
* de comparat câțiva șefi la deșteptăciunea totală a subalternilor fiecăruia. Rezultatul de afișat la ecran și de a-l salva în fișier textual.

***Exemplu 2:***

De descries în limbajul Java ierarhia de clase **Snusmumrel → Salivel → Hrumel → Marfufica**

De descris interfata**.** De implementat interfața în cîteva clase din ierarhie.

**clasa Snusmumrel**

*câmpuri:*

* câmp de tip întreg, care arată numărul de lăbuțe Snusmumrelului

*funcții:*

* câțiva constructori (implicit (completarea campurilor cu valori aleatoare) și cu parametri)
* funcțiile de acces la câmpurile clasei, și funcțiile de setare a lor
* funcția de afisare a câmpurilor clasei la ecran
* funcția virtuală ”ghemuit”. Fiecare exemplar al clasei care se va ghemui, își va micșora în jumate valorile tuturor cîmpurilor clasei

**class Salivel** extends Snusmumrel

*câmpuri:*

* câmp de tip întreg, care arată numărul de hrunicele Salivelului

*funcții:*

* câțiva constructori (implicit și cu parametri) cu apelarea la constructorii din clasa parinte.
* funcțiile de acces la câmpurile clasei, și funcțiile de setare a lor
* redescrierea a functiei de afisare

**class Hrumel** extends Salivel

*câmpuri:*

* vector dinamic, în care se păstrează informația despre fiecare hrunica Hrumelului
* câmp de tip real, care arată câte faricele are Hrumelul

*funcții:*

* câțiva constructori (implicit (completarea campurilor cu valori aleatoare) și cu parametri)
* funcțiile de acces la câmpurile clasei, și funcțiile de setare a lor
* redescrierea a functiei de afisare
* funcția de înfucierea Salivelului de Hrumel, Salivelul fiind transmis în calitate de parametru.(Adică nulificarea vectorului dinamic Salivelului) – asta se întâmplă în cazul dacă numărul de hrunicele Salivelului e mai mic. Altfel – atunci salivelul insuși va înfucia Hrumelul care l-a apelat(adică îi ia toate hrunicelele – nulifică vectorul Hrumelului)
* funcția de înmulțirea Hrumeilor (Hrumelul nou-născut va avea ca valorile câmpurilor -rezultatul produsului câmpurilor respective ai operanzilor)

**class Marfufica** extends Hrumel

*câmpuri:*

* câmp de tip real, care arată lungimea cozii Marfuficăi

*funcții:*

* câțiva constructori (implicit (completarea campurilor cu valori aleatoare) și cu parametri)
* funcțiile de acces la câmpurile clasei, și funcțiile de setare a lor
* redescrierea a functiei de afisarea *a* câmpurilor clasei la ecran
* funcția care schimbă numărul de hrunicele Hrumelului concret, transmis în calitate de parametru (dacă numărul de faricele la hrumel e mai mare, marfufica îi adaugă hrunicelele, dacă mai mic sau egal – le scade)
* funcția statica de adunare a Marfuficelor (Marfufica nou-născută va avea lungimea cozii egală cu suma lungimilor cozilor ai ambilor operanzi)

**interface**

*funcții:*

* funcția de afisare a câmpurilor clasei la ecran

**main**

* de creat vectorul dinamic cu exemplarele ”viețuitoarelor” ciudate, dând ca tipul de bază – clasa Snusmumrel, și de inițializat aceste elemente cu obiectele tuturor claselor moștenite. *Dupa ce de utilizat in main numai obiectele din acest vector.*
* de afișat la ecran câmpurile tuturor obiectelor create
* macară un exemplar al fiecărei clase(în afară de Snusmumrelelor, desigur) trebuie să se ghemuiască. După ce de afișat caracteristicile noi al ecran.
* de găsit din toate Marfuficelele create pe aceea cu coada cea mai lungă coadă și de a obliga-o să schimbe numărul de hrunicele la câțiva Hrumei. De afișat la ecran Hrumeii schimbați.
* de ”înmulțit” o pereche de Hrumei de ”adunat” câteva marfuficele. Rezultatele adunării și înmulțirii creaturilor – de salvat într-un fișier textual.
* *și niciodata pe urmă de a nu folosi nimic ce ar putea să Vă pună să Vă gândiți la Hrumei sau Marfuficele:)*

***Exemplu 3:***

**Ierbivor** **Peștilor** **de** **aur**

De descries în limbajul Java ierarhia de clase **Pește**

**Prădător**

De descris interfata**.** De implementat interfața în cîteva clase din ierarhie.

**clasa Pește**

*câmpuri:*

* câmp de tip șir de caractere, care păstreaza numele Peștelui
* câmp de tip real, care indică greutatea Peștelui
* câmp de tip întreg, care arată viteza Peștelui

*funcții:*

* câțiva constructori (implicit (completarea campurilor cu valori aleatoare) și cu parametri)
* funcțiile de acces la câmpurile clasei, și funcțiile de setare a lor
* funcția de afisare a câmpurilor clasei la ecran
* funcția abstracta void saveObjectToFile (String nameFile)

**class Ierbivor** extends Pește

*câmpuri:*

* câmp de tip întreg, care indică numărul de cicatrici ai Ierbivorului

*funcții:*

* câțiva constructori (implicit și cu parametri) cu apelarea la constructorii din clasa parinte.
* funcțiile de acces la câmpurile clasei, și funcțiile de setare a lor
* redescrierea a functiei de afisare
* funcția în care pe obiectul care a apelat funcția se ațâță un Prădător (în calitate de parametru), și care returnează viteza fugarului vegetarian în dependență de coeficientul de accelerare și numărul dinților al atacantului, și care adaugă Ierbivorului curent atâtea cicatrici, câți dinți are atacantul.
* funcția, de tip final, de încrucișare (adunare) a două obiecte, și care returnează un Ierbivor nou(rezultatul se salvează în obiect nou), cu caracteristicile calculate din media aritmetica a câmpurilor operanzilor.

**class Peștilor de aur** extends Ierbivor

*câmpuri:*

* câmp de tip șir de caractere, care indică naționalitatea Peștișorului de aur
* câmp de tip întreg, care indică numărul de dorințe pe care Peștișorul de aur este gata să înfăptuiască fiind prinsă

*funcții:*

* câțiva constructori (implicit (completarea campurilor cu valori aleatoare) și cu parametri)
* funcțiile de acces la câmpurile clasei, și funcțiile de setare a lor
* redescrierea a functiei de afisare
* funcția ”prăjeală”, care mărește cu 50% numărul dorințelor pe care Peștișorul dat este gata să le înfăptuiască.

**class Prădător** extends Pește

*câmpuri:*

* câmp de tip întreg, care indică numărul de dinți ai Prădătorului
* câmp de tip real al voracității, care indică câtă carne poate sa mănânce Prădătorul
* câmp de tip real, final, care indică coeficientul de accelerare al Ierbivorului când apare un Prădător

*funcții:*

* câțiva constructori (implicit (completarea campurilor cu valori aleatoare) și cu parametri)
* funcțiile de acces la câmpurile clasei, și funcțiile de setare a lor
* redescrierea a functiei de afisarea *a* câmpurilor clasei la ecran
* funcția logica care află dacă el se va sătura de la consumul Ierbivor transmis ca parametru(se compară voracitatea Prădătorului cu greutatea Ierbivorului).
* funcția statica de coliziune a doi pești, care în dependență de la viteza totală, dezbate fiecărei un număr oarecare de dinți

**interface**

*funcții:*

* funcția de afisare a câmpurilor clasei la ecran

**main**

* de creat vectorul dinamic cu exemplarele tuturor peștișorilor descriși mai sus , dând ca tipul de bază – clasa origine, și de inițializat aceste elemente cu obiectele tuturor claselor moștenite. *Dupa ce de utilizat in main numai obiectele din acest vector.*
* de afișat la ecran câmpurile tuturor obiectelor create
* de a prăji toate pestesori de aur gasite in vector.
* De a ațâța un Prădător asupra unui Ierbivor. Rezultatele de afișat la ecran.
* De a afla satisfacția Prădătorului de la consumul unui ierbivor. De a îcnrucișa câțiva Ierbivor supraviețuiți
* De a coliziona(fără colaider) doi Prădători(preferabil flămânzi). Rezultatul de afișat la ecran și de a-l salva în fișier textual

***Постановка задачи:***

1. *Описать иерархию классов (минимум 4). Базовый класс абстрактный с минимум 1 абстрактной функцией*
2. *Описать class Main с функцией main вне иерархии*
3. *Описать как минимум 2 интерфейса, каждый подключить минимум в 1 класс иерархии*
4. *Каждый наследный класс должен иметь уникальные поля и методы*
5. *Вызвать конструкторы классов родителей с параметрами и без*
6. *Вызвать обычные функции из классов родителей*
7. *Переопределить в каждом классе Randomize в дефолтном конструкторе*
8. *Переопределить в каждом классе функцию ввода полей с клавиатуры*
9. *Переопределить в каждом классе функцию записи в файл*
10. *Создать минимум одну final : переменную && функцию && класс*
11. *Создать минимум одну static : переменную && функцию*
12. *Использовать instanceof и преобразование типов объектов*
13. *В main создать вектора объектов с типами: базовый класс && интерфейс*
14. *В функции main для созданных объектов вызвать уникальные функции их классов, вывести их на экран*
15. *Описать в отчете собственный вариант работы так, как показано в примерах ниже.*
16. *После чего - сказать “****УРРРРАААА ! ! ! !****” и сдать уже вторую лабораторную работу преподавателю. :)*

***Примеры для вдохновения ))) :***

***Пример 1:***

Описать на языке Java иерархию классов **Человек → Юзер → Админ → Босс.**

Описать Интерфейс **Публичность.** Подключить Публичность к Человеку.

**Класс Человек**

*поля:*

* статическая переменная, показывающая социальный статус = {«чел», «юзер», «админ», «босс»}
* целочисленное поле, показывающее рост Человека
* символьное поле, показывающее цвет глаз Человека
* дробное поле, показывающее вес Человека
* логическое поле, показывающее пол Человека

*методы:*

* конструкторы (по умолчанию (с заполнением полей случайными значениями) и с параметрами)
* функции доступа к полям класса, функции изменения полей класса
* функции вывода на экран полей Человека
* абстрактная функция сохранения данных Человека в файл

**Класс Юзер** extends Человек

*поля:*

* целочисленное поле, показывающее количество прав доступа для Юзера
* динамический вектор, в котором хранится информация о каждом праве доступа Юзера

*методы:*

* конструкторы (по умолчанию и с параметрами) с вызовом конструкторов класса родителя
* функции доступа к полям класса, функции изменения полей класса
* переопределение функции вывода на экран

**Класс Админ** extends Юзер

*поля:*

* дробное поле, показывающее коэффициент лени Админа(от 0 до 1)
* 2 дробных поля, показывающие IQ Админа (от 0 до 1) и зарплату Админа

*методы:*

* конструкторы (по умолчанию и с параметрами) с вызовом конструкторов класса родителя
* функции доступа к полям класса, функции изменения полей класса
* переопределение функции вывода на экран
* функция урезания админом прав доступа для конкретного Юзера, переданного как параметр (т.е. обнуление динамического вектора Юзера)
* функция сложения (типа final) двух админов (при этом у нового админа будет среднее арифметическое лени обоих операндов, среднее арифметическое их IQ и суммарные права доступа)

**Класс Босс** extends Админ

*поля:*

* дробное поле, показывающее количество подчиненных **Босса**
* динамический вектор, в котором хранятся IQ всех подчиненных **Босса**

*методы:*

* конструкторы (по умолчанию и с параметрами) с вызовом конструкторов класса родителя
* функции доступа к полям класса, функции изменения полей класса
* переопределение функции вывода на экран
* функция, которая устанавливает новую зарплату конкретному Админу, переданному как параметр (зарплата повышается, если отношение лени к IQ Админа < 1, иначе — понижается.)
* функция увольнения конкретного Админа в случае, если отношение его лени к IQ > 100 (при этом меняется количество подчиненных Босса)
* статическая функция умножения двух Боссов :) (у нового босса число подчиненных = произведение подчиненных обоих операндов)

**Интерфейс Публичность**

*методы:*

* функция вывода на экран данных

**main**

* понасоздавать динамический вектор экземпляров всей вышеописанной офисной «живности», задав как тип элементов вектора — базовый класс и заполнив эти элементы — объектами всех наследованных классов. *Далее работать только с элементами этого вектора, не создавая дополнительных отдельных объектов.*
* вывести на экран поля всех созданных человек
* урезать права одного из юзеров с помощью функции одного из админов. Новые права урезанного юзера вывести на экран
* Один из боссов должен проверить всех админов на отношение лени к IQ, и либо уволить, либо что-то сделать с зарплатой каждого :) Новые зарплаты админов вывести на экран.
* «сложить» пару админов и «перемножить» пару боссов
* сравнить нескольких боссов на суммарную мозговитость их подчиненных. Результат вывести на экран и сохранить в текстовый файл.

***Пример 2:***

Описать на языке Java иерархию классов **Снусмумрик → Слюник → Хрямзик → Марфуфочка.**

Описать Интерфейс **Публичность.** Подключить Публичность к Снусмумрику.

**Класс Снусмумрик**

*поля:*

* целочисленное поле, показывающее число лап Снусмумрика

*методы:*

* конструкторы (по умолчанию (с заполнением полей случайными значениями) и с параметрами)
* функции доступа к полям класса, функции изменения полей класса
* функции вывода на экран всех полей класса
* абстрактная функция “кукожиться”. Любой экземпляр класса, который будет кукожиться, уменьшит значение всех своих полей вдвое.

**Класс Слюник** extends Снусмумрик

*поля:*

* целочисленное поле, показывающее количество хрунечек у Слюника

*методы:*

* конструкторы (по умолчанию и с параметрами) с вызовом конструкторов класса родителя
* функции доступа к полям класса, функции изменения полей класса
* переопределение функции вывода на экран

**Класс Хрямзик** extends Слюник

*поля:*

* динамический вектор, в котором хранится информация о каждой хрунечке у Хрямзика
* дробное поле, показывающее, сколько фариков у Хрямзика

*методы:*

* конструкторы (по умолчанию и с параметрами) с вызовом конструкторов класса родителя
* функции доступа к полям класса, функции изменения полей класса
* переопределение функции вывода на экран
* функция нафучивания хрямзиком конкретного Слюника, переданного в качестве параметра в функцию. (т.е. зануление динамического вектора Слюника) — это происходит в том случае, если у принятого как параметр слюника меньше хрунечек. Если больше — то слюник сам нафучивает вызвавшего его хрямзика (т.е. забирает все его хрунечки — обнуляет динамический вектор Хрямзика)
* статическая функция умножения Хрямзиков (при этом у нового хрямзика все поля будут произведением соответствующих полей обоих операндов)

**Класс Марфуфочка** extends Хрямзик

*поля:*

* дробное поле, показывающее длину хвоста Марфуфочки

*методы:*

* конструкторы (по умолчанию и с параметрами) с вызовом конструкторов класса родителя
* функции доступа к полям класса, функции изменения полей класса
* переопределение функции вывода на экран
* функция, которая меняет число хрунечек конкретному Хрямзику, переданному в качестве параметра в функцию (если число фариков у хрямзика больше, марфуфочка добавляет хрямзику хрунечки, если меньше или равно — уменьшает их число...)
* статическая функция сложения Марфуфочек (новая марфуфочка будет иметь длину хвоста суммарную от длин хвостов обоих операндов)

**Интерфейс Публичность**

*методы:*

* функция вывода на экран данных

**main**

* понасоздавать динамический вектор экземпляров всей вышеописанной странной «живности», задав как тип элементов вектора — Снусмумриков и заполнив эти элементы — объектами всех наследованных классов. *Далее работать только с элементами этого вектора, не создавая дополнительных отдельных объектов.*
* вывести на экран поля всех созданных объектов
* хотя бы по один экземпляр каждого класса (кроме снусмумриков, конечно) должен покукожиться. После чего надо вывести его новые характеристики на экран.
* понафучивать несколько пар хрямзиков и слюников. Новые характеристики каждого вывести на экран.
* найти из всех созданных марфуфочек самую длиннохвостую и заставить ее изменить число хрунечек у нескольких хрямзиков. Изменившихся хрямзиков вывести на экран.
* «перемножить» пару хрямзиков и «сложить» пару марфуфочек. Результаты сложения и умножения живности – сохранить в текстовый файл.
* больше не употреблять ничего такого, что заставило бы Вас думать о хрямзиках и марфуфочках :)

***Пример 3:***

**Травоед Золотая Рыбка**

Описать на языке Java иерархию классов **Рыба**

**Хищник**

Описать Интерфейс **Публичность.** Подключить Публичность к Рыбе.

**Класс Рыба**

*поля:*

* строковое поле, хранящее кличку Рыбы
* дробное поле, показывающее массу Рыбы
* целочисленное поле, показывающее скорость Рыбы

*методы:*

* конструкторы (по умолчанию (с заполнением полей случайными значениями) и с параметрами)
* функции доступа к полям класса, функции изменения полей класса
* функция вывода на экран всех полей класса
* абстрактная функция сохранения данных Человека в файл.

**Класс Травоед** extends Рыба

*поля:*

* целочисленное поле, показывающее количество шрамов на теле Травоеда
* дробное поле, типа final, показывающее коефициент ускорения Травоеда при виде Хищника

*методы:*

* конструкторы (по умолчанию и с параметрами) с вызовом конструкторов класса родителя
* функции доступа к полям класса, функции изменения полей класса
* переопределение функции вывода на экран
* функция, в которой на вызывающий объект натравливается Хищник (в виде параметра), и которая возвращает скорость улепетывающего вегетарианца в зависимости от коефициента ускорения и количества зубов нападающего, и добавляющее текущему Травоеду столько шрамов сколько зубов у нападающего.
* Функция (типа final) скрещивания (сложения) двух объектов, возвращающую нового Травоеда (результат сохраняется в новом объекте), с полями средне-арифметическими от значений полей операндов.

**Класс ЗолотаяРыбка** extends Травоед

*поля:*

* строковое поле, хранящее национальность Золотой Рыбки
* целочисленное поле, показывающее, сколько желаний готова исполнить загнанная в угол Золотая Рыбка

*методы:*

* конструкторы (по умолчанию и с параметрами) с вызовом конструкторов класса родителя
* функции доступа к полям класса, функции изменения полей класса
* переопределение функции вывода на экран
* «поджаривание», которая увеличивает в полтора раза количество желаний которые данная рыбка готова исполнить.

**Класс Хищник** extends Рыба

*поля:*

* целочисленное поле, показывающее количество зубов Хищника
* дробное поле прожорливости, показывающее, сколько мяса может съесть Хищник

*методы:*

* конструкторы (по умолчанию и с параметрами) с вызовом конструкторов класса родителя
* функции доступа к полям класса, функции изменения полей класса
* переопределение функции вывода на экран
* логическая функция, которая узнает, насытится ли Хищник при поглощении скормленным как параметр Травоедом (сравниваются поля прожорливости Хищника и массы Травоеда)
* статическая функция столкновения двух рыбин, которая в зависимости от суммарной скорости обеих, выбивает каждой некоторое количество зубов

**Интерфейс Публичность**

*методы:*

* функция вывода на экран данных

**main**

* понасоздавать динамический вектор экземпляров всего вышеописанного «рая Ихтиандра» задав как тип элементов вектора — базовый класс и заполнив эти элементы — объектами всех наследованных классов.. *Далее работать только с элементами этого вектора, не создавая дополнительных отдельных объектов.*
* вывести на экран поля всех созданных рыбин
* Поджарить всех золотых рыбок из вектора
* Натравить на Травоеда Хищника. Узнать результат трапезы Хищника заданым Травоедом.
* Скрестить несколько выживших травоедов.
* Разогнать и столкнуть двух Хищников(желательно голодных).
* Результат всех махинаций вывести на экран и сохранить в текстовый файл.

***Пример 4:***

**Грифиндор** 

Описать на языке Java иерархию классов **МагХогвардса Коктевран**

**Слизерин**

**Пуфендуй**

Описать интерфейсы “**Пожиратели смерти”** подключить к слизеринуи “**Орден феникса**” подключить к грифиндору***.***

**Класс МагХогвардса**

*поля:*

* строковое поле, хранящее имя Мага
* дробное поле, показывающее силу Мага
* целочисленное поле, показывающее здоровье Мага
* целочисленное поле, показывающее возраст Мага
* строковое поле, хранящее статус Мага { “студент”, “профессор”}

*методы:*

* конструкторы (по умолчанию (с заполнением полей случайными значениями) и с параметрами)
* функции доступа к полям класса, функции изменения полей класса
* функция вывода на экран всех полей класса
* абстрактная функция \*\*\*.

**Класс Грифиндор** extends МагХогвардса

*поля:*

* целочисленное поле, показывающее храбрость Грифиндорца

*методы:*

* конструкторы (по умолчанию и с параметрами) с вызовом конструкторов класса родителя
* функции доступа к полям класса, функции изменения полей класса
* переопределение функции вывода на экран
* функция, в которой на вызывающий объект натравливается опасное магическое существо (в виде 2 параметров: Название и сила существа ), и которая возвращает результат нападения

**Класс Слизерин** extends МагХогвардса

*поля:*

* целочисленное поле, показывающее степень хитрости Слизеринца

*методы:*

* конструкторы (по умолчанию и с параметрами) с вызовом конструкторов класса родителя
* функции доступа к полям класса, функции изменения полей класса
* переопределение функции вывода на экран
* функция «пакость», которая принимает как параметр МагаХогвардса и меняет ему характеристики в зависимости от его статуса и силы, а также принадлежности к факультету (слизеринцы своим не пакостят, а грифиндорцам пакостят особенно сильно).

**Класс Ревенкло** extends МагХогвардса

*поля:*

* целочисленное поле, показывающее IQ мага из Ревенкло

*методы:*

* конструкторы (по умолчанию и с параметрами) с вызовом конструкторов класса родителя
* функции доступа к полям класса, функции изменения полей класса
* переопределение функции вывода на экран
* статическая функция «примирение», которая принимает как параметры двух студентов Хогвардса (двух магов, у которых статус “студент”) и заставляет их обменяться силой.

**Класс Пуфендуй** extends МагХогвардса

*поля:*

* дробное поле, показывающее преданность Пуфендуйца
* целочисленное поле, показывающее количество друзей Пуфендуйца

*методы:*

* конструкторы (по умолчанию и с параметрами) с вызовом конструкторов класса родителя
* функции доступа к полям класса, функции изменения полей класса
* переопределение функции вывода на экран

**Интерфейс Пожиратели Смерти**

*методы:*

* функция Авадакедавра, принимающая как параметр МагаХогвардса и обнуляющая ему силу и здоровье
* функция Круциатус, принимающая как параметр МагаХогвардса и уменьшающая ему вдвое здоровье
* функция Империо, принимающая как параметр МагаХогвардса и стирающая ему имя

**Интерфейс Орден Феникса**

*методы:*

* функция Экспелиармус, принимающая как параметр МагаХогвардса и обнуляющая ему силу
* функция ЭкспектоПатронум, выводящая на экран вид патронуса, одного из {“лань”, “феникс”, “медведь”, “сова”, “жаба”}

**main**

* создать динамический вектор “Хогвардс” объектов типа МаговХогвардса и заполнить его студентами и преподавателями всех факультетов. Создать вектор “Плохие парни” типа ПожирателиСмерти и вектор “Хорошие парни” типа ОрденФеникса. *Далее работать только с элементами этих векторов, не создавая дополнительных отдельных объектов.*
* вывести на экран поля всех созданных магов
* заставить всех грифиндорцев из “Хогвардса” сразиться со случайными магическими существами
* заставить всех найденных в “Хогвардсе” слизеринцев напакостить случайному магу из этого же вектора.
* Найти и вывести на экран самого дружелюбного пуфендуйца из “Хогвардса” (у кого больше всего друзей).
* заставить первого найденного в “Хогвардсе” студента Ревенкло помирить пару из магов из “Плохих парней” и “Хороших парней”.
* Заставить всех “Плохих парней” и “Хороших парней” побиться друг с другом методами из их интерфейсов.