# Упражнения: Абстрактни класове

## Фигури

Създайте йерархия от класове, започвайки с **абстрактен** клас **Shape**:

* **Абстрактни методи:**
  + **calculatePerimeter(): doulbe**
  + **calculateArea(): double**
* **Виртуални методи:**
  + **Draw(): string**

Наследете класа **Shape** с двата дъщерни класа:

* **Rectangle**
* **Circle**

Всеки от тях трябва да има:

* **Полета:** 
  + **Дължина и ширина за Rectangle**
  + **Радиус за Circle**
* **Капсулация за полетата**
* **Публичен конструктор**
* **Конкретни методи за изчисления (обиколка и лице)**
* **Презаписани методи за рисуване**

## Цветна фигура

Създайте абстрактен клас **ColoredFigure**, който притежава:

* Поле **color** за отбелязване на цвета (като низ)
* Поле **size** за отбелязване на размер на фигурата
* Конструктор, който приема за параметри цвят и размер
* Метод **Show()**, който отпечатва цвета и размера на обекта.
* Абстрактен метод **GetName()**, който връща името на фигурата
* Абстрактен метод **GetArea()**, който връща лицето на фигурата

Създайте клас **Triangle**, който наследява **ColoredFigure**, като този клас има:

* Конструктор, който извиква конструктора на суперкласа
* Дефиниция за абстрактния метод **GetName()**, като този метод връща низа **"Triangle"**.
* Дефиниция за абстрактния метод **GetArea()**, като този метод връща лицето на триъгълника, като триъгълникът се приема за **равностранен**, със страна **size**. Използвайте формулата: 

Създайте клас **Square**, който наследява **ColoredFigure**, като този клас има:

* Конструктор, който извиква конструктора на суперкласа
* Дефиниция за абстрактния метод **GetName()**, като този метод връща низа **"Square"**.
* Дефиниция за абстрактния метод **GetArea()**, като този метод връща лицето на квадрата със страна **size**.

Създайте клас **Circle**, който наследява **ColoredFigure**, като този клас има:

* Конструктор, който извиква конструктора на суперкласа
* Дефиниция за абстрактния метод **GetName()**, като този метод връща низа **"Circle"**.
* Дефиниция за абстрактния метод **GetArea()**, като този метод връща лицето на кръга, с радиус **size**.

### Вход

На първия ред на входа има единствено цяло число N – брой заявки. От следващите N реда се подава заявка в един от следните формати:

* **Triangle <цвят> <размер>**
* **Circle <цвят> <размер>**
* **Square <цвят> <размер>**

### Изход

За всяка заявка трябва да създаде обект от съответния клас, след което трябва да изпечатате 4 реда:

<име на фигурата>:

Color: <цвят>

Size: <размер>

Area: <лице>

Отпечатвайте лицето с точно два знака след запетаята.

### Примери

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 3 Circle blue 1 Square red 2 Triangle green 3 | Circle: Color: blue Size: 1 Area: 3.14 Square: Color: red Size: 2 Area: 4.00 Triangle: Color: green Size: 3 Area: 3.90 |

## Превозни средства

Напишете програма, която моделира 2 превозни средства (**Car** и **Truck**). Трябва да може да симулирате **шофиране** и **презареждане** на превозните средства. **Car** и **truck** имат **количество гориво**, **консумация на гориво в литър за км** и могат да бъдат **управлявани на дадено разстояние** и **презаредени с определено количество гориво.** Но през лятото и двете превозни средства използват климатик и тяхната **консумация** за км е завишена с **0.9** литра за **Car** и с **1.6** литра за **Truck**. Също така **камионът** има малка дупка в резервоара и когато се **зарежда** получава само **95%** от **горивото**. **Колата** няма проблеми със зареждането и получава всичкото гориво. Ако превозното средство не може да измине даденото разстояние, горивото му не се променя.

Вход

* На първи ред – информация за колата във формат {Car {fuel quantity} {liters per km}}
* На втори ред – информация за камиона във формат {Truck {fuel quantity} {liters per km}}
* На трети ред – брой команди N, които ще бъдат подадени на следващите N реда
* На следващите N реда – команди във формат:
* Drive Car {distance}
* Drive Truck {distance}
* Refuel Car {liters}
* Refuel Truck {liters}

Изход

След всяка Drive команда отпечатайте дали колата/камионът може да пропътува разстоянието, като използвате следния формат при успех:

Car/Truck travelled {distance} km

Или при неуспех:

Car/Truck needs refueling

Накрая изпечатайте оставащото гориво за колата и камиона закръгледно до 2 знака след запетаята във формат:

Car: {liters}

Truck: {liters}

**Примери**

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| Car 15 0.3  Truck 100 0.9  4  Drive Car 9  Drive Car 30  Refuel Car 50  Drive Truck 10 | Car travelled 9 km  Car needs refueling  Truck travelled 10 km  Car: 54.20  Truck: 75.00 |
| Car 30.4 0.4  Truck 99.34 0.9  5  Drive Car 500  Drive Car 13.5  Refuel Truck 10.300  Drive Truck 56.2  Refuel Car 100.2 | Car needs refueling  Car travelled 13.5 km  Truck needs refueling  Car: 113.05  Truck: 109.13 |

## Превозни средства II

Използвайте решението на предната задача като стартова точка и добавете нова функционалност. Добавете ново превозно средство – **Bus**. Сега всяко превозно средство има **капацитет на резервоара** и количество на горивото, което **не може да падне под 0** (ако количеството гориво падне под 0, **отпечатайте** на конзолата **“Fuel must be a positive number”**).

**Колата** и **автобуса не могат да се заредят** с гориво **повече от техния капацитет на резервоара**. Ако се **опитате да сложите повече гориво** в резервоара от **наличното място,** отпечатайте **“Cannot fit fuel in tank”** и **не добавяйте гориво** в резервоара.

Добавете **нова команда** за автобуса. **Автобусът** може да **пътува със или без хора**. Ако автобусът пътува **с хора**, то **климатикът трябва да е включен** и неговата **консумация на гориво** за километър се **увеличава с 1.4 литра**. Ако в автобуса **няма хора**, то климатикът ще е **изключен** и **консумацията не се увеличава**.

### Вход

* На първите три реда въвеждате информация за превозните средства във формат:

**Vehicle {initial fuel quantity} {liters per km} {tank capacity}**

* На четвъртия ред – броят на командите N, които ще бъдат подадени на следващите N реда
* На следващите N реда – команди във формата
  + Drive Car {distance}
  + Drive Truck {distance}
  + Drive Bus {distance}
  + DriveEmpty Bus {distance}
  + Refuel Car {liters}
  + Refuel Truck {liters}
  + Refuel Bus {liters}

### Изход

* След всяка Drive команда изпечатайте дали колата/камионът/автобусът може да пропътува това разстояние във формат при успех:

Car/Truck/Bus travelled {distance} km

* Или при неуспех:

Car/Truck/Bus needs refueling

* Ако даденото гориво е **≤ 0** изпечатайте **“Fuel must be a positive number”.**
* Ако даденото гориво, не може да се вмести в резервоара, отпечатайте “Cannot fit in tank”
* Накрая, отпечатайте оставащото гориво за колата, камиона и автобуса, закръглени до 2 знака след запетаята във формат:

Car: {liters}

Truck: {liters}

Bus: {liters}

## Ферма за животни

Вашата задача е да създадете йерархия от класове като тази на диаграмата по-долу. Всички класове освен Vegetable, Meat, Mouse, Tiger, Cat & Zebra трябва да са абстрактни. Презаписвайте метод ToString().



Входът трябва да се прочете от конзолата. Всеки **четен** ред ще съдържа информация за животно в следния формат:

**{AnimalType} {AnimalName} {AnimalWeight} {AnimalLivingRegion} [{CatBreed}** *= Only if its cat***]**

На **нечетните** редове ще получите информация за храната, която трябва да дадете на животното. Редът ще съдържа **FoodType** и **количество** разделено от интервал.

Трябва да направите логиката, която определя дали животното ще яде предоставената му храна. Мишката и зебрата трябва да проверят дали храната им е Зеленчук. Ако е – те ще я ядат. В противен случай, трябва да отпечатате съобщение в следния формат:

**{AnimalType} are not eating that type of food!**

**Котките** ядат **каква да е** храна, но **тигрите** приемат **само месо**. Ако се даде **зеленчук** на **тигъра**, отпечатайте съобщение като това отгоре на конзолата.

Презапишете **ToString** метода, така че да отпечата информация за животното във формата:

**{AnimalType} [{AnimalName}, {CatBreed}, {AnimalWeight}, {AnimalLivingRegion}, {FoodEaten}]**

След като въведете информация за животно и храна, извикайте **MakeSound** метода за текущото животно и след това го нахранете. В края изпечатайте целия обект и продължете да четете информация за следващото животно/храна. Входът ще продължи докато не получите команда “**End**”.

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| Cat Gray 1.1 Home Persian  Vegetable 4  End | Meowwww  Cat[Gray, Persian, 1.1, Home, 4] |
| Tiger Typcho 167.7 Asia  Vegetable 1  End | ROAAR!!!  Tigers are not eating that type of food!  Tiger[Typcho, 167.7, Asia, 0] |
| Zebra Doncho 500 Africa  Vegetable 150  End | Zs  Zebra[Doncho, 500, Africa, 150] |
| Mouse Jerry 0.5 Anywhere  Vegetable 0  End | SQUEEEAAAK!  Mouse[Jerry, 0.5, Anywhere, 0] |

## Работници

Създайте абстрактен клас **BaseEmployee**, който притежава:

* Поле **employeeID**, което пази идентификационния номер на работника като низ.
* Поле **employeeName** за отбелязване на името на работника
* Поле **employeeAddress** за отбелязване на адреса по местоживеене на работника
* Конструктор, който приема три параметъра и ги присвоява на съответните полета, изброени по-горе
* Метод **Show()**, който отпечатва информация за работника.
* Абстрактен метод **CalculateSalary(int workingHours)**, който ще изчислява заплатата за работника, като се приема параметър – брой изработени часове
* Абстрактен метод **GetDepartment()**, който връща името на звеното от фирмата, в което работи работника

Създайте клас **FullTimeEmployee**, който наследява **BaseEmployee**, като този клас има:

* Поле **employeePosition**, което пази позицията, на която е назначен работника
* Поле **employeeDepartment**, което пази отдела, в който е назначен работника
* Конструктор с пет параметъра – **employeeID, employeeName, employeeAddress, employeePosition, employeeDepartment**, който извиква конструктора на суперкласа, а след това присвоява стойностите за двете полета от този клас
* Презаписан метод **Show()**, който извиква метод **Show()** от базовия клас, а след това отпечатва допълнително два реда – за позицията и отдела
* Дефиниция за абстрактния метод **CalculateSalary(int workingHours)**, като този метод връща сума, според следната формула: 250 + **workingHours**\*10.80.
* Дефиниция за абстрактния метод **GetDepartment()**, като този метод връща стойността записана в **employeeDepartment**

Създайте клас **ContractEmployee**, който наследява **BaseEmployee**, като този клас има:

* Поле **employeeTask**, което пази задачата, с която този работник е назначен като контрактьор
* Поле **employeeDepartment**, което пази отдела, в който е назначен работника
* Конструктор с пет параметъра – **employeeID, employeeName, employeeAddress, employeeTask, employeeDepartment**, който извиква конструктора на суперкласа, а след това присвоява стойностите за двете полета от този клас
* Презаписан метод **Show()**, който извиква метод **Show()** от базовия клас, а след това отпечатва допълнително ред – за задачата, с която е назначен работника
* Дефиниция за абстрактния метод **CalculateSalary(int workingHours)**, като този метод връща сума, според следната формула: 250 + **workingHours**\*20.
* Дефиниция за абстрактния метод **GetDepartment()**, като този метод връща стойността записана в **employeeDepartment**