## Encapsulation

## 01 Class Box

You are given a geometric figure box with parameters **length**, **width** and **height**. Model a class **Box** that can be instantiated by the same three parameters. Expose to the outside world only methods for its surface area, lateral surface area and its volume (formulas: <http://www.mathwords.com/r/rectangular_parallelepiped.htm>).

A box’s side **should not** be zero or a negative number. Make a private setter that performs **data validation internally**.

On the first three lines you will get the length, width and height. On the next three lines print the **surface area**, **lateral surface area** and the **volume** of the box:

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  3  4 | Surface Area - 52.00  Lateral Surface Area - 40.00  Volume – 24.00 |
| 1.3  1  6 | Surface Area - 30.20  Lateral Surface Area - 27.60  Volume - 7.80 |

## 02 Animal Farm

You should be familiar with encapsulation already. For this problem, you’ll be working with the **Animal Farm project**. It contains a class **Chicken**. Chicken contains several **fields**, a **constructor**, several **properties** and several **methods**. Your task is to encapsulate or hide anything that is not intended to be viewed or modified from outside the class.

Chicken lives for **15 years**. Chicken have **name** for sure, at least **1 symbol** long. Chicken producing eggs:

* First 6 years it produces 2 eggs per day
* Next 6 years it produces 1 egg per day
* And after that it produces 0.75 eggs per day

### Step 1. Encapsulate Fields

Fields should be **private**. Leaving fields open for modification from outside the class is potentially dangerous. Make all fields in the Chicken class private.

In case the value inside a field is needed elsewhere, use **getters** to reveal it.

### Step 2. Ensure Classes Have a Correct State

Having **getters and setters** is useless if you don’t actually use them. The Chicken constructor modifies the fields directly which is wrong when there are suitable setters available. Modify the constructor to fix this issue.

### Step 3. Validate Data Properly

Validate the chicken’s **name** (it cannot be null, empty or whitespace). In case of **invalid name**, print exception message "Name cannot be empty."

Validate the **age** properly, minimum and maximum age are provided, make use of them. In case of **invalid age**, print exception message "Age should be between 0 and 15."

### Step 4. Hide Internal Logic

If a method is intended to be used only by descendant classes or internally to perform some action, there is no point in keeping them **public**. The **calculateProductPerDay()** method is used by the **productPerDay()** public getter. This means the method can safely be hidden inside the Animal class by declaring it **private**.

|  |  |
| --- | --- |
| **Input** | **Output** |
| Mara  10 | Chicken Mara (age 10) can produce 1 eggs per day. |
| Mara  17 | Age should be between 0 and 15. |
| Gosho  6 | Chicken Gosho (age 6) can produce 1 eggs per day. |

## 03 Shopping Spree

Create two classes: class **Person** and class **Product**. Each person should have a **name**, **money** and a **bag of products**. Each product should have **name** and **cost**. Name cannot be an empty string. Money cannot be a negative number.

Create a program in which each command corresponds to a person buying a product. If the person **can afford** a product **add it** to his bag. If a person **doesn’t have** enough money, **print** an appropriate message ("[Person name] can't afford [Product name]").

On the first two lines you are given all people and all products. After all purchases print every person in the order of appearance and all products that he has bought also in order of appearance. If nothing is bought, print the name of the person followed by "Nothing bought".

Read commands till you find line with **"END"** command. In case of invalid input (negative money exception message: "**Money cannot be negative**") or empty name: (empty name exception message "**Name cannot be empty**") break the program with an appropriate message. See the examples below:

|  |  |
| --- | --- |
| **Input** | **Output** |
| Pesho=11;Gosho=4  Bread=10;Milk=2  Pesho Bread  Gosho Milk  Gosho Milk  Pesho Milk  END | Pesho bought Bread  Gosho bought Milk  Gosho bought Milk  Pesho can't afford Milk  Pesho - Bread  Gosho - Milk, Milk |
| Mimi=0  Kafence=2  Mimi Kafence  END | Mimi can't afford Kafence  Mimi – Nothing bought |
| Jeko=-3  Chushki=1  Jeko Chushki  END | Money cannot be negative |