# Homework: OOP Inheritance

Tasks with **\*** are considered hard or involve something we still haven't mentioned in the lectures. Try to solve them nevertheless. Do not worry about constraints how big the numbers can be, length of strings and so on. As we know JavaScript handles automatically the type and length of values. All tasks are submitted to GitHub.

## Person

You are asked to model an application for storing data about people. You should be able to have a person and a child. The child is derived of the person. Your task is to model the application. The only constraints are:

* **Person** – represents the base class by which all others are implemented
  + People should **not** be able to have **negative age**
* **Child** - represents a class which is derived by the class **Person.**
  + Children should **not** be able to have age **greater than 15**

### Note

Your classes' names **MUST** be the same as the names shown above!!!

Create a new empty class and name it **Person**. Every person has a name, and age.

|  |
| --- |
| **Sample Code** |
| **class** Person {    *// 1. Add the Properties  // 2. Add the Constructor  // 3. Add the Methods* } |

### Step 1. Define the Properties of a Person

Define the **name** and **age** properties of a Person. Ensure that they can only be **changed by the class itself or its descendants** (pick the most appropriate access modifier).

### Step 2. Define the Constructor

Define a constructor that accepts **name** and **age** arguments.

### Step 3. Perform Validations

After you have created the **properties** (e.g. **name** and **age**). Next step is to **perform validations** for each one. The **getter should return the corresponding property’s value** and the **setter should validate** the input data before setting it. Do this for each property.

### Constraints

* If the age of a person is negative – exception’s message is: **"Age must be positive!"**
* If the age of a child is bigger than 15 – exception’s message is: **"Child’s age must be less than 16!"**
* If the name of a child or a person is no longer than 3 symbols – exception’s message is: **"Name’s length should not be less than 3 symbols!"**

### Step 5. Create a Child

Create a **Child** class that inherits **Person** and has the same constructor definition. However, do not copy the code from the Person class - **reuse the Person class’s constructor**.

## Book Shop

You are working in a library. And you are pissed off by writing descriptions for books by hand, so you wanted to use the computer to make them faster. So the task is simple. Your program should have two classes – one for the ordinary books – **Book**, and another for the special ones – **GoldenEditionBook**. So let’s get started! We need two classes:

* **Book** - represents a book that holds **title**, **author** and **price**. A book should offer **information** about itself in the format shown in the output below.
* **GoldenEditionBook** - represents a special book holds the same properties as any **Book**, but its **price** is always **30% higher**.

### Input

The input comes from the console. It will hold four lines. On the first you will be given the name, on the second the title and third will be the standard price. Fourth line will hold the type of book - **GOLD** or **STANDARD**

### Output

On the first line print OK, if all inputs are in the correct format. If not print every error message on a single line.

If input is valid print a second line with the price of the book.

### Constraints

* If the author’s second name is starting with a digit– exception’s message is: **"Author not valid!"**
* If the title’s length is less than 3 symbols – exception’s message is: **"Title not valid!"**
* If the price is zero or it is negative – exception’s message is: **"Price not valid!"**
* If book type is invalid – exception’s message is: **"Type is not valid!"**

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| Ivo 4ndonov  Under Cover  9999999999999999999  STANDARD | Author not valid! |
| Georgi Petrov  Summer Moon  22  STANDARD | OK  22 |
| Vasil Ivanov  UML Basics  10  GOLD | OK  13 |

### Step 1. Create a Book Class

Create a new class and name it **Book**.

|  |
| --- |
| **Sample Code** |
| **class** Book {   *// 1. Add the Properties  // 2. Add the Constructor  // 3. Add the Methods* } |

### Step 2. Define the Properties of a Book

Define the **title**, **author** and **price** properties of a Book. Ensure that they can only be **changed by the class itself or its descendants** (pick the most appropriate access modifier).

### Step 3. Define a Constructor

Define a constructor that accepts **author, title** and **price** arguments.

### Step 4. Perform Validations

Create the **properties** (**price**, **title** and **author**) and **perform validations** for each one. The **getter should return the corresponding property** and the **setter should validate** the input data before setting it. Do this for every property.

### Step 5. Define method toString

The toString() method allows a class to decide how it will react when it is treated like a string.

And voila! If everything is correct, we can now create **Book objects** and display information about them.

### Step 6. Create a GoldenEditionBook

Create a **GoldenEditionBook** class that inherits **Book** and has the same constructor definition. However, do not copy the code from the Book class - **reuse the Book class constructor**.

There is **no need** to rewrite the **price**, **title** and **author** properties since **GoldenEditionBook** inherits **Book** and by default has them.

### Step 7. Override the Price Method

Golden edition books should return a **30%** higher **price** than the original price. In order for the getter to return a different value, we need to override the method.

Back to the **GoldenEditionBook** class, let's override the getter method.

|  |
| --- |
| **Sample Code** |
| **function** getPrice() {  **return super.**getPrice() \* 1.3; } |

## Mankind

Your task is to model an application. It is very simple. The mandatory models of our application are 3: Human, Worker and Student.

The parent class – Human should have **first name** and **last name**. Every student has a **faculty number**. Every worker has a **week salary** and **work hours per day**. It should be able to calculate the money he earns by hour. You can see the constraints below.

### Input

On the first input line you will be given info about a single student - first name, last name and faculty number.

On the second input line you will be given info about a single worker - first name, last name, salary and working hours.

### Output

You should first print the info about the student following a single blank line and the info about the worker in the given formats:

* Print the student info in the following format:

**First Name: {student's first name}**

**Last Name: {student's last name}**

**Faculty number: {student's faculty number}**

* Print the worker info in the following format:

**First Name: {worker's first name}**

**Last Name: {worker's second name}**

**Week Salary: {worker's salary}**

**Hours per day: {worker's working hours}**

**Salary per hour: {worker's salary per hour}**

Print exactly two digits after every double value's decimal separator (e.g. 10.00)

### Constraints

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Constraint** | **Exception Message** |
| Human first name | Should start with a capital letter | "Expected upper case letter!Argument: firstName" |
| Human first name | Should be at least 4 symbols | "Expected length at least 4 symbols!Argument: firstName" |
| Human last name | Should start with a capital letter | "Expected upper case letter!Argument: lastName" |
| Human last name | Should be at least 3 symbols | "Expected length at least 3 symbols!Argument: lastName " |
| Faculty number | Length should be in range [5..10] digits | "Invalid faculty number!" |
| Worker last name | Should be more than 3 symbols | "Expected length more than 3 symbols!Argument: lastName" |
| Week salary | Should be more than 10 | "Expected value mismatch!Argument: weekSalary" |
| Working hours | Should be in the range [1..12] | "Expected value mismatch!Argument: workHoursPerDay" |

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| Ivan Ivanov 08  Pesho Kirov 1590 10 | Invalid faculty number! |
| Stefo Mk321 0812111  Ivcho Ivancov 1590 10 | First Name: Stefo  Last Name: Mk321  Faculty number: 0812111  First Name: Ivcho  Last Name: Ivancov  Week Salary: 1590.00  Hours per day: 10.00  Salary per hour: 22.71 |

## \*Animals

Create a hierarchy of **Animals**. Your task is simple: there should be a base class which all others derive from. Your program should have 3 different animals – **Dog**, **Frog** and **Cat**. Let’s go deeper in the hierarchy and create two additional classes – **Kitten** and **Tomcat**. **Kittens are female and Tomcats are male!** We are ready now, but the task is not complete. Along with the animals, there should be a class which classifies its derived classes as sound producible. You may guess that all animals are sound producible. The only one mandatory functionality of all sound producible objects is to **produceSound()**. For instance, the dog should bark.

Your task is to model the hierarchy and test its functionality. Create an animal of all kinds and make them produce sound.

On the console, you will be given some lines of code. Each two lines of code, represents animals and their names, age and gender. On the first line there will be the kind of animal, you should instantiate. And on the next line, you will be given the name, the age and the gender. Stop the process of gathering input, when the command **“Beast!”** is given.

### Output

* On the console, print for each animal you’ve instantiated, its info on two lines. On the first line, print:  
   {**Kind of animal**} {**name**} {**age**} {**gender**}
* On the second line, print: {**produceSound()**}

### Constraints

* Each **Animal** should have **name**, **age** and **gender**
* **All** **properties**’ values should **not be blank** (e.g. name, age and so on…)
* If you enter invalid input for one of the properties’ values, throw exception with message: **“Invalid input!”**
* Each animal should have a functionality to **produceSound()**
* Here is example of what each kind of animal should produce when, **produceSound()** is called
  + **Dog: “BauBau”**
  + **Cat: “MiauMiau”**
  + **Frog: “Frogggg”**
  + **Kittens: “Miau”**
  + **Tomcat: “Give me one million b\*\*\*h”**
  + **Message from the Animal class: "Not implemented!"**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Cat  Macka 12 Female  Dog  Sharo 132 Male  Beast! | Cat Macka 12 Female  MiauMiau  Dog Sharo 132 Male  BauBau |
| Frog  Sashky 12 Male  Beast! | Frog Sashky 12 Male  Frogggg |
| Frog  Sashky -2 Male  Beast! | Invalid input! |

## \*\*Cat Lady

Ginka has many cats in her house of various breeds, since some breeds have specific characteristics, Ginka needs some way to catalogue the cats, help her by creating a class hierarchy with all her breeds of cats so that she can easily check on their characteristics. Ginka has 3 specific breeds of cats “**Siamese**”, “**Cymric**” and the very famous bulgarian breed “**Street Extraordinaire**”, each breed has a specific characteristic about which information should be kept. For the Siamese cats their ear size should be kept, for Cymric cats - the length of their fur in milimeters and for the Street Extraordinaire the decibels of their meowing during the night.

From the console you will receive lines of cat information until the command “**End**” is received, the information will come in one of the following formats:

* “**Siamese <name> <earSize>”**
* “**Cymric <name> <furLength>”**
* “**StreetExtraordinaire <name> <decibelsOfMeows>”**

On the last line after the “**End**” command you will receive the name of a cat, you should print that cat’s information in the same format in which you received it.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| StreetExtraordinaire Maca 85  Siamese Sim 4  Cymric Tom 28  End  Tom | Cymric Tom 28 |
| StreetExtraordinaire Koti 80  StreetExtraordinaire Maca 100  Cymric Tim 31  End  Maca | StreetExtraordinaire Maca 100 |

### Hint

Use class inheritance to represent the cat hierarchy and define toString methods of concrete breeds to allow for easy printing of the cat, regardless what breed it is.