Homework

Exercise 1. University

Create a class named `University`.

- It should contain two properties
  - `teachers` (`[]` as default)
  - `students` (`[]` as default)
- It should contain 3 methods
  - `addMember(member`
  - `removeMember(member)`
  - `startLesson()` (description below)

For members you should create two classes, Teacher and `Student` which will be inherited from the class `UniversityMember`.

Members should contain 4 properties

- `name`
- `age`
- `role`
- `energy` (`24` as default)

And one method named `info()` which will return all 4 properties of it.

After calling the `University` class `startLesson()` method, students get `-2` of energy and teachers get `-5` of energy.

Exercise 2. CoffeeShop

Create a constructor `CoffeeShop`, which has three properties:

- **name** : a string (basically, of the shop)
- **menu** : an array of items (of object type), with each item containing the item (name of the item), type (whether food or a drink) and price.
- **orders** : an empty array

and seven methods:

- **addOrder**: adds the name of the item to the end of the orders array if it exists on the menu. Otherwise, return "This item is currently unavailable!"

- **fulfillOrder**: if the `orders` array is not empty, return `"The {item} is ready!"`. If the `orders` array is empty, return `"All orders have been fulfilled!"`
- **listOrders**: returns the list of orders taken, otherwise, an empty array.
- **dueAmount**: returns the total amount due for the orders taken.
- **cheapestItem**: returns the name of the cheapest item on the menu.
- **drinksOnly**: returns only the item names of type `drink` from the menu.
- **foodOnly**: returns only the item names of type food from the menu.

IMPORTANT:
Orders are fulfilled in a FIFO (first-in, first-out) order.

Notes:
You can create another constructor for menu items.

Example:

```
tcs.addOrder("hot cocoa") ➞ "This item is currently unavailable!"
// Tesha's coffee shop does not sell hot cocoa

tcs.addOrder("iced tea") ➞ "This item is currently unavailable!"
// specifying the variant of "iced tea" will help the process

tcs.addOrder("cinnamon roll") ➞  "Order added!"
tcs.addOrder("iced coffee") ➞ "Order added!"

tcs.listOrders ➞ ["cinnamon roll", "iced coffee"]
// the list of all the items in the current order

tcs.dueAmount() ➞ 2.17

tcs.fulfillOrder() ➞ "The cinnamon roll is ready!"
tcs.fulfillOrder() ➞ "The iced coffee is ready!"
tcs.fulfillOrder() ➞ "All orders have been fulfilled!"
// all orders have been presumably served
tcs.listOrders() ➞ []
// an empty array is returned if all orders have been exhausted

tcs.dueAmount() ➞ 0.0
// no new orders taken, expect a zero payable

tcs.cheapestItem() ➞ "lemonade"

tcs.drinksOnly() ➞ ["orange juice", "lemonade", "cranberry juice",
"pineapple juice", "lemon iced tea", "vanilla chai latte", "hot
chocolate", "iced coffee"]

tcs.foodOnly() ➞ ["tuna sandwich", "ham and cheese sandwich", "bacon
and egg", "steak", "hamburger", "cinnamon roll"]
```

Exercise 3 Abstract class

Abstract classes are base classes from which other classes may be derived. In other words, you can't create an instance with that class, you can only extend from it.

Create a class, which will throw an error if you try to create an instance with it.

Exercise 4.1. Class hierarchy

Implement the described class hierarchy: the `Character` class is the parent class for all the others. 6 child classes `Bowerman`, `Swordsman`, `Magician`, `Daemon`, `Undead`, and `Zombie` inherit from it, setting their own characteristics.

1. Properties that instances of the Character class should have:

`name` - name
`type` - type
`health` - standard of living
`level` - character level
`attack` - attack
`defense` - protection

2. The class constructor must meet the following requirements:

`name` - string, min - 2 characters, max - 10
`type` - one of the types (string): `Bowman, Swordsman, Magician, Daemon, Undead, Zombie`

If incorrect values are passed, an error should be thrown

```
throw new Error(...)
```

3. Your function should automatically set the following fields:

`health`: 100
`level`: 1
`Attack / Defense`:

- `Bowman`: 25/25
- `Swordsman`: 40/10

- `Magician`: 10/40
- `Undead`: 25/25
- `Zombie`: 40/10
- `Daemon`: 10/40

Exercise 4.2. Class hierarchy

Implement the `levelUp` method in the `Character` class, which works like this:

- Raises the level field by 1
- Increases attack and defense by 20%
- Sets health to 100

The method should work only if the life indicator is not equal to 0. Otherwise, an error is generated (you cannot increase the level of the deceased).

Implement the `damage(points)` method in the `Character` class, which changes the internal state of the object (`points` is the damage done to the character). The `damage(points)` method does not return anything and calculates the final change in the character's health (`health`) according to the formula:

`health -= points * (1 - defense / 100)`,
given that the value of `health >= 0`.

Exercise 5. OOP Structure

Let's imagine that we have the following classes

- `City`
- `Building`
- `Hospital`
- `PoliceDepartment`
- `Car`
- `PoliceCar`
- `AmbulanceCar`

Please decide on the inheritance hierarchy.
You have to decide which class will be a parent class and which should be a child class by correctly extending classes from each other.

Example:

```
PoliceCar extends Car {...}
```

Also please implement the following conditions.

Condition 1.

The `City` has many buildings. It may be an array property of the `City` class named buildings. It can include different Objects of the `Building` type

Example:

```
const nairiHospital = new Hospital();
...
const erebuniHospital = new Hospital();
...
const centralPoliceDepartment = new PoliceDepartment()

city.buildings = [nairiHospital, erebuniHospital,
centralPoliceDepartment]
```

Condition 2.

Is similar to **Condition 1**. Buildings can have many cars.
For example, a `Hospital` may have 10 `AmbulanceCars` or a police department may have police cars.

Exercise 6. TV Class

Create a `TV` class with properties like `brand`, `channel` and `volume`.

- Specify brand in a constructor parameter. Channel should be 1 by default. Volume should be 50 by default.
- Add methods to increase and decrease volume. Volume can never be below 0 or above 100.
- Add a method to set the channel. Let's say the TV has only 50 channels so if you try to set channel 60 the TV will stay at the current channel.
- Add a method to reset TV so it goes back to channel 1 and volume 50. (Hint: consider using it from the constructor).
- It's useful to write a status that returns info about the TV status like: **`"Panasonic at channel 8, volume 75"`**.

Exercise 7. Shopping Cart

Create a class `Product` with properties `name`, `type`, and `price`

Create a class `ShoppingCart`

- In `ShoppingCart`, define a method `addProduct(product)`
  In `ShoppingCart`, define a method `removeProduct(product)`

- In `ShoppingCart`, define a method `totalPrice()`, that returns the total amount of the products it contains.

- Now let's say that, if you buy 5 products or more, you have a 10% discount. Change `totalPrice` to reflect this calculation.

- Besides the previous discount, if you buy 3 items of the same type (e.g. 3 pencils) you may get one more for free. So, if you buy 4 pencils you only pay 3, if you buy 8 pencils you only pay 6, etc. Change `totalPrice` so it considers the free items you get. Notice that if you buy 3 pencils you just pay for the 3 of them.

- Sometimes a product is sold out and has to be replaced by a new one. Add a method `replace(productName, replacementProduct)` that looks for products with `productName` and replaces them with new instances of the product like `replacementProduct`. Notice that productName is a string, and replacementProduct is a Product