Lab: Polymorphism

This document defines the lab for the "Java Advanced" course @ Software University. Please submit your solutions (source code) to all below-described problems in Judge.

1. Math Operation

Create a class **MathOperation**, which should have method **add()**. Method **add()** has to be invoked with **two, three,** or **four Integers.**

You should be able to use the class like this:

Main.java

```
public static void main(String[] args) throws IOException {
    MathOperation math = new MathOperation();
    System.out.println(math.add(2, 2));
    System.out.println(math.add(3, 3, 3));
    System.out.println(math.add(4, 4, 4, 4));
}
```

Examples

Input	Output
	4 9 16

Solution

Class **MathOperation** should look like this:

```
public class MathOperation {

    public int add(int a, int b) {
        return a + b;
    }
    public int add(int a, int b, int c) {
        return a + b + c;
    }
    public int add(int a, int b, int c, int d) {
        return a + b + c + d;
    }

}
```

2. Shapes

Create class hierarchy, starting with abstract class **Shape**:

- **Fields:**
  - **perimeter: Double**
  - **area: Double**

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

- **Encapsulation for these fields**
- **Abstract methods:**
  - **`calculatePerimeter()`**
  - **`calculateArea()`**

Extend Shape class with two children:

- **`Rectangle`**
- **`Circle`**

Each of them needs to have:

- **Fields:**

  For **`Rectangle`**
  - **`height: Double`**
  - **`width: Double`**

  For **`Circle`**
  - **`radius: Double`**
- **Encapsulation for these fields**
- **Public constructor**
- **Concrete methods for calculations (perimeter and area)**

3. Animals

Create a class **`Animal`**, which holds two fields:

- **`name: String`**
- **`favouriteFood: String`**

The **`Animal`** has one abstract method **`explainSelf(): String`.**
You should add two new classes - **Cat** and **Dog. Override** the **`explainSelf()`** method by adding concrete animal sound on a new line. (Look at examples below)

You should be able to use the class like this:

Main

```
public static void main(String[] args) {
    Animal cat = new Cat("Oscar", "Whiskas");
    Animal dog = new Dog("Rocky", "Meat");
    System.out.println(cat.explainSelf());
    System.out.println(dog.explainSelf());
}
```

Examples

Input	Output
	I am Oscar and my favourite food is Whiskas MEEOW I am Rocky and my favourite food is Meat DJAAF

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

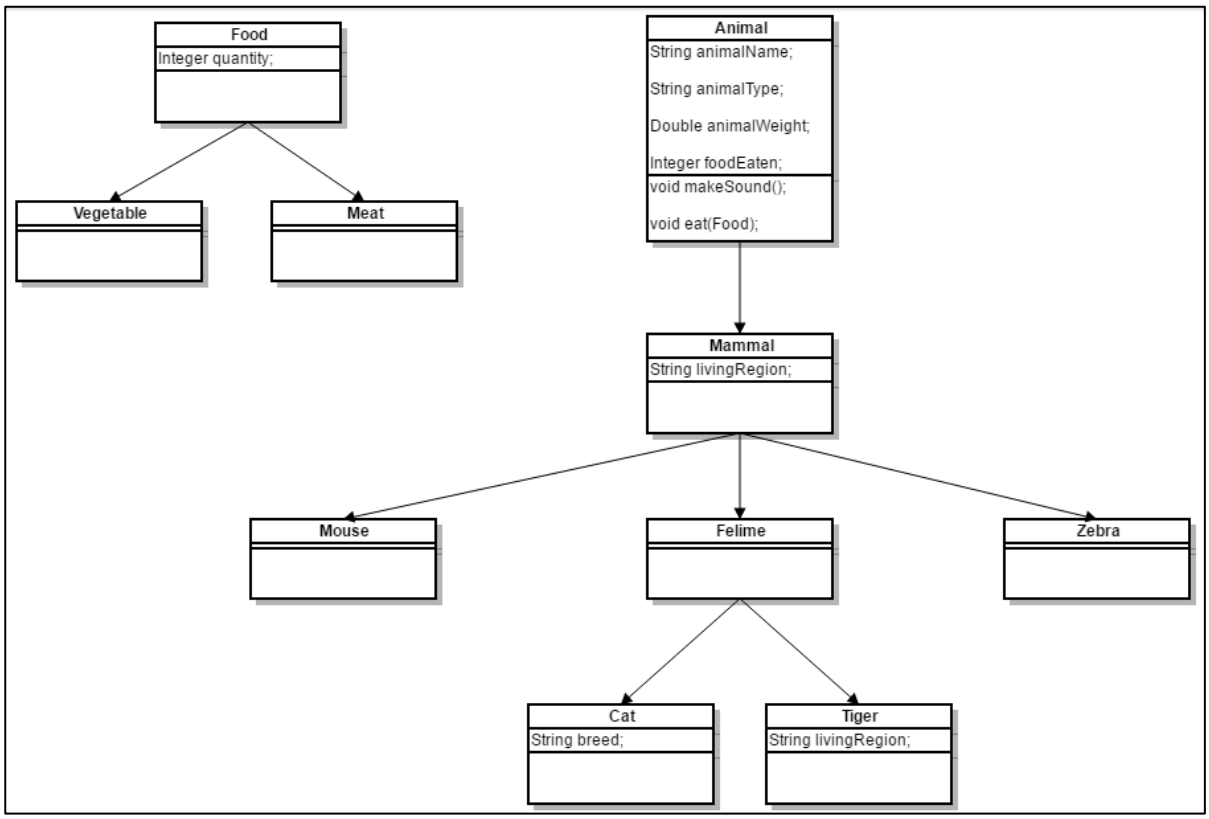
Solution

```
public abstract class Animal {
    private String name;
    private String favouriteFood;

    protected Animal(String name, String favouriteFood) {
        this.setName(name);
        this.setFavouriteFood(favouriteFood);
    }

    public String explainSelf() {
        return String.format("I am %s and my favourite food is %s",
                this.getName(),
                this.getFavouriteFood());
    }
```

```
public class Cat extends Animal {
    public Cat(String name, String favouriteFood) {
        super(name, favouriteFood);
    }

    @Override
    public String explainSelf() {
        return String.format("%s%nMEEOW", super.explainSelf());
    }
}
```

4. *Wild Farm

Your task is to create a class **hierarchy** like the picture below. All the classes except **Vegetable**, **Meat**, **Mouse**, **Tiger**, **Cat** & **Zebra** should be **abstract**.



© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

Input should be read from the console. Every **even** line will contain information about the **Animal** in following format:

"{AnimalType} {AnimalName} {AnimalWeight} {AnimalLivingRegion}".
If the animai is a **cat**: **"{AnimalType} {AnimalName} {AnimalWeight} {AnimalLivingRegion} {CatBreed}"**.

On the **odd** lines, you will receive information about the food that you should give to the **Animal**. The line will consist of **FoodType** and **quantity** separated by whitespace.

You should build the logic to determine if the animal is going to eat the provided food. The **Mouse** and **Zebra** should check if the food is **Vegetable**. If it is they will eat it. Otherwise, you should print a message in the format:

"{AnimalType} are not eating that type of food!". AnimalType to be in the **plural**.

Cats eat **any** kind of food, but **Tigers** accept **only Meat**. If a **Vegetable** is provided to a **tiger** message like the one above should be printed on the console.

After you read information about the **Animal** and **Food** then invoke **makeSound()** method of the current animal and then feed it. In the end, print the whole object in the format:

"{AnimalType} [{AnimalName}, {AnimalWeight}, {AnimalLivingRegion}, {FoodEaten}]".

If the animal is a **cat**: **"{AnimalType} [{AnimalName}, {CatBreed}, {AnimalWeight}, {AnimalLivingRegion}, {FoodEaten}]".**

Proceed to read information about the next animal/food. The input will continue until you receive "**End**".

Print all **AnimalWeight** with two digits after the decimal separator. Use the **DecimalFormat** class.

Note: consider overriding **toString()** method.

Example

Input	Output
Cat Gray 1.1 Home Persian Vegetable 4 End	Meowwww Cat[Gray, Persian, 1.1, Home, 4]
Tiger Tom 167.7 Asia Vegetable 1 End	ROAAR!!! Tigers are not eating that type of food! Tiger[Tom, 167.7, Asia, 0]
Zebra Jaguar 500 Africa Vegetable 150 End	Zs Zebra[Jaguar, 500, Africa, 150]
Mouse Jerry 0.5 Anywhere Vegetable 0 End	SQUEEEAAAK! Mouse[Jerry, 0.5, Anywhere, 0]

SoftUni
© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us: