



The Kitchen class is defined as follows:

```
public class Kitchen{

    private int sizeInSQM;

    private String[] utensils;

    private String chefName;

    private int chefAge;

    private boolean michelinAward;

    public Kitchen(int size, String chefName, int age){

        this.sizeInSQM = size;

        this.chefName = chefName;

        this.utensils = new String[10];

        this.chefAge = age

        this.michelinAward = false;

    }

    //sample accessor method

    public String getChefName(){

        return chefName;

    }

    //sample mutator method

    public void setChefAge(){

        chefAge += 1;

    }

    public int getNameLength(){

        return chefName.length()

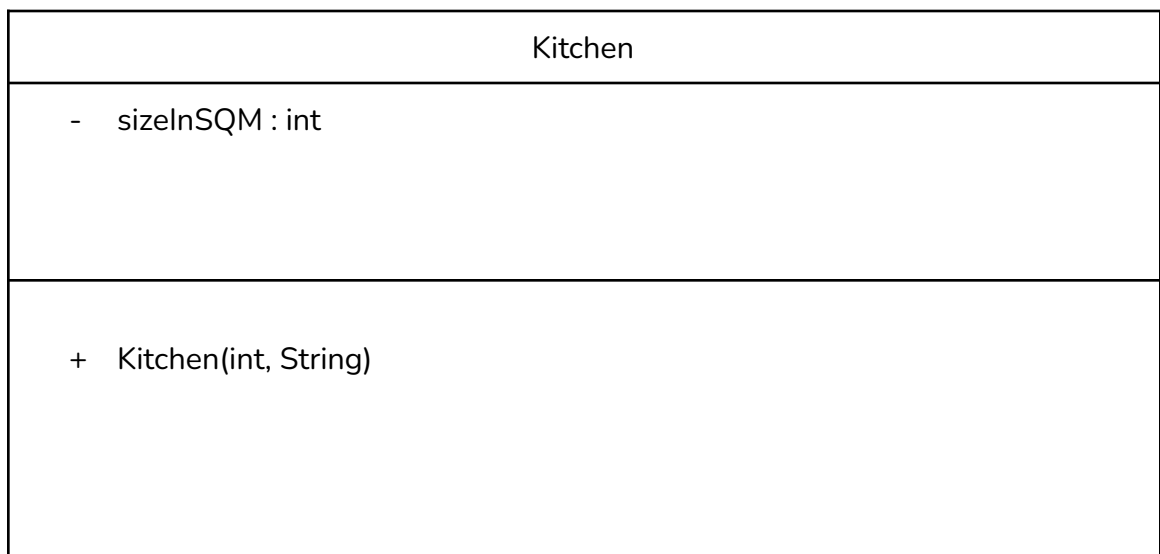
    }

}
```

(a) Write a sample accessor method for the `sizeInSQM` **class attribute**.

(b) Write a sample mutator method for the `michelinAward` **class attribute**.

(c) Including the two methods you wrote in parts (a) and (b), complete the UML diagram for the `Kitchen` class.



(d) Write a line of code that will **instantiate** (create an instance of) the `Kitchen` class in another class eg a `Main` class

.

The above code is an example of how to store and manage data in a digital object. The design is an example of **abstraction**, where only essential details are included/visible and non-essential details are hidden.

Look again at the Kitchen class and highlight every attribute relating to a Chef (hint: there should be 3 Chef-related attributes - *utensils don't belong to the Chef, they belong to the Kitchen*).

(e) Now, using your list of attributes, **complete** the full Chef class:

```
public class Chef{

    private String name;
    private int age;
    private boolean michelin;

    public Chef(String name, int age, boolean michelin){
        this.name =
        this.age =
        this.michelin =
    }

    public String getName(){

    }

    public int _____(){

    }

    public boolean _____(){

    }

    public void _____(){

        this.age = this.age + 1;

    }

    public void setMichelin(boolean michelin){

    }

}
} //end of class
```

Oh! Now we have a Chef class which describes Chefs - we know their attributes/properties and their methods/behaviour.

- (f) Let's return to the Kitchen class. How can we redesign the Kitchen class to include this new class?

```
public class Kitchen{  
  
    private int sizeInSQM;  
    private String[] utensils;  
    private _____ chef;  
  
    public Kitchen(_____ chef, int sizeInSQM) {  
        this._____ = chef;  
        this.sizeInSQM = sizeInSQM;  
    }  
  
    public int getSizeInSQM(){  
        return this.sizeInSQM;  
    }  
  
    public _____ getChef() {  
        return this.chef;  
    }  
  
    public _____ setChef(_____ newChef) {  
        this._____ = newChef;  
    }  
}
```

Oh! Interesting. Take a long look at this talk about it with your classmate(s) if you have any.

What questions/problems do you have or lines of code don't you get. Make notes here:

Scenario

A famous Kitchen in northern Italy, Uga Montalto, has a Chef, Villane Mustarde, who is 37 years old and has a Michelin award.

While Villane is under 45 years of age, the Kitchen is happy to employ him.

However, on Villane's 45th birthday, he will be replaced with a new Chef, Michelle Vinaugre, who is 24 and hoping to earn a Michelin award in the future by working at this famous Kitchen.

Using your knowledge of the new Kitchen class and the Chef class, design a Main program which will simulate the above scenario.

```
public class Main{
```

```
}
```

AGGREGATION

We can now see that a Kitchen **HAS-A** Chef.

Without a Chef, there cannot really be a Kitchen. However, the Chef can continue to exist without the Kitchen.

This relationship is known as **Aggregation**.

To express this relationship as a UML diagram, we can connect the 2 classes with a diamond to express the **HAS-A** relationship.

