

Object Oriented can help your life.

How does it work?

Powered by :
Anatasya Dwijayanti Chang
Riana Hasna Muthiah
Shafira Ayu Maharani

Lecturer : Dr. Fariz Darari
TA : Jonathan Christian

OBJECT ORIENTED DESIGN

Single Responsibility

A class should have one and only one reason to change, meaning that a class should have only one job

Open Closed

Objects or entities should be open for extension, but closed for modification

Liskov Substitution

Let $q(x)$ be a property provable about objects of x of type T . Then $q(y)$ should be provable for objects

Dependency Inversion

Entities must depends on abstractions not on concretions. It states that the high level module must not depend on the low level module, but they should depend on abstractions

Interface Segregation

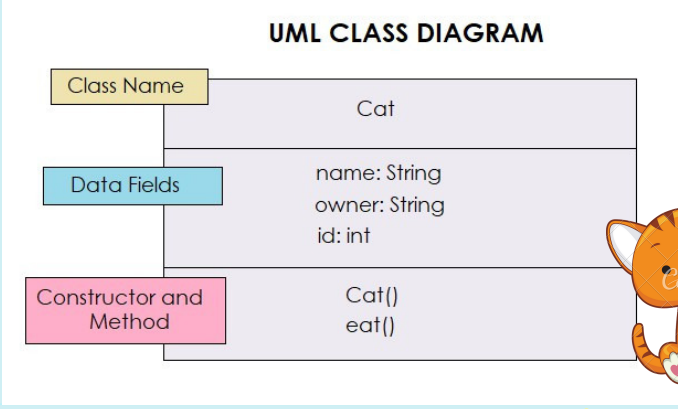
A client should never be forced to implement an interface that it doesn't be forced to depend on methods they do not use

Setter & Getter

UML Class Diagram

Recall that the data fields (instance variables) of variable are private. We can access them from within the Cat class, but if we try to access them from another class, the compiler generates an error.

This technique can be used to maintain the confidentiality of personal data so that only a few people can access the data



Access Modifier

ACCESS MODIFIERS	CLASS	PACKAGE	SUBCLASS	WORLD
PUBLIC	✓	✓	✓	✓
PROTECTED	✓	✓	✓	✗
DEFAULT	✓	✓	✗	✗
PRIVATE	✓	✗	✗	✗

Advantages of Encapsulation

Data Hiding



It will not be visible to the user that how the class is storing values in the variables

Increased Flexibility



Can make the variables of the class as read-only or write-only depending on our requirement

Reusability



Improves the re-usability and easy to change with new requirements.

Testing code is easy



Encapsulated code is easy to test for unit testing.

SIMPLE CODE ENCAPSULATION

Without setter & getter

```
public class CatHouse {
    public static void main(String[] args) {
        Cat cat1 = new Cat();
        cat1.name = "Milky";
        cat1.owner = "Steve";
        cat1.Id = 1833;

        Cat cat2 = new Cat();
        cat2.name = "Boo";
        cat2.owner = "Sarah";
    }
}

class Cat {
    String name;
    String owner;
    int Id;

    public Cat() {
        this.name = "Cat";
    }

    public void eat() {
        System.out.println("I'm Eating...");
    }
}
```

With setter & getter

```
public class CatHouse {
    public static void main(String[] args) {
        Cat cat1 = new Cat();
        cat1.name = "Milky";
        cat1.owner = "Steve";
        cat1.Id = 1833;

        Cat cat2 = new Cat();
        cat2.name = "Boo";
        cat2.owner = "Sarah";
    }
}

class Cat {
    String name;
    String owner;
    int Id;
}
```

```
public Cat() {
    this.name = "Cat";
}

public void setName(String name) {
    this.name = name;
}

public void setOwner(String owner) {
    this.owner = owner;
}

public void setId(int Id) {
    this.Id = Id;
}

public String getName() {
    return name;
}

public String getOwner() {
    return owner;
}

public int getId() {
    return Id;
}

public void eat() {
    System.out.println("I'm Eating...");
}
}
```