Object-Oriented Programming (OOP) in Java

Gusti Alfian M. P

About Me...

- Web
  - Backend
    - php (CI, Laravel, lumen)
    - Node.js (express, sails)
  - Frontend
    - Bootstrap
    - JQuery
    - Angular.js
    - ReactJS
    - D3
- Android
- Unity
- Blender 3D

Why OOP ?

- Breaking complex problems into more manageable ones.
- Create an architecture that can scale up.
- Many programming language using OOP.

OOP Key Feature

- Class
- Atributte
- Method
- Object

- Public
- Protcted
- Private

- Final
- Static

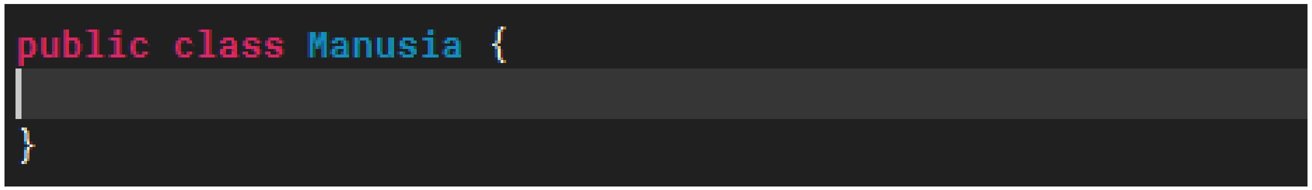
- Encapsulation
- Inheritance
- Polymorphism

- Abstract class
- Interface

- Overwrite
- Override

You and OOP for the first time probably like…

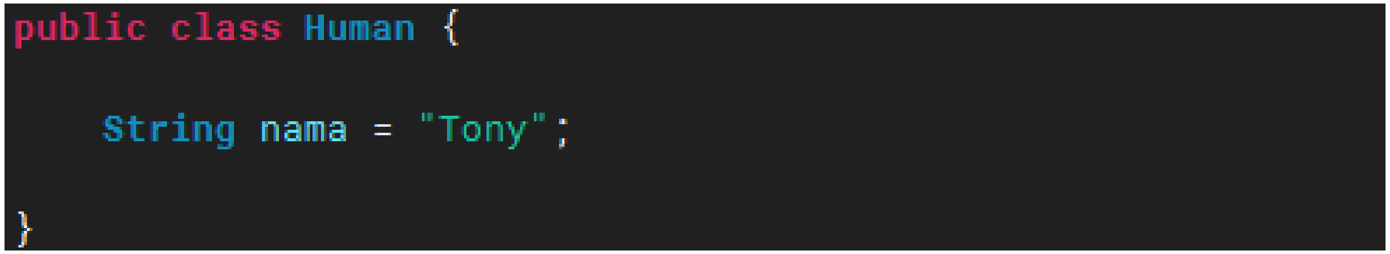
Class



```
public class Manusia {

}
```

- Blueprint

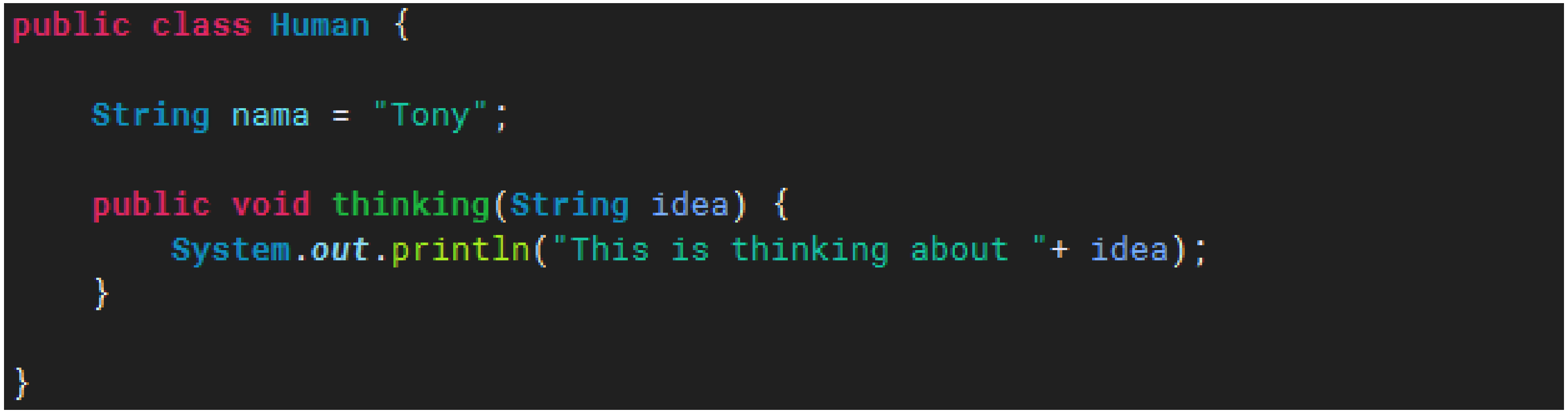
Atributte



```
public class Human {

    String nama = "Tony";

}
```

- boolean
- int
- double
- String

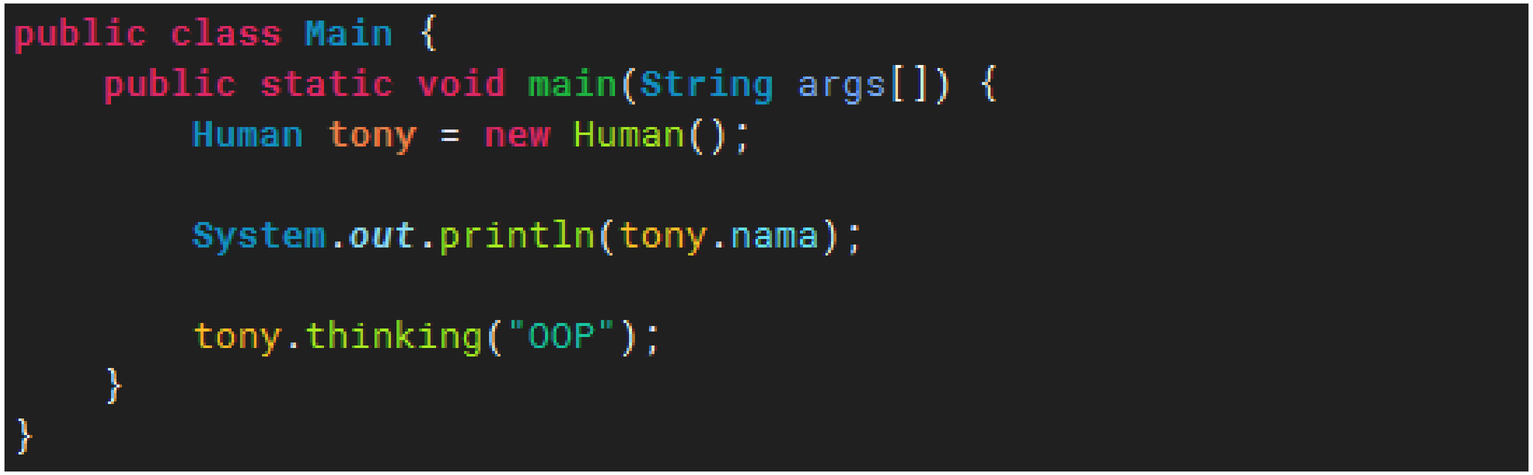
Method



```
public class Human {

    String nama = "Tony";

    public void thinking(String idea) {
        System.out.println("This is thinking about "+ idea);
    }

}
```

- Access modifier
- Return type
- Name of method
- Params

Object



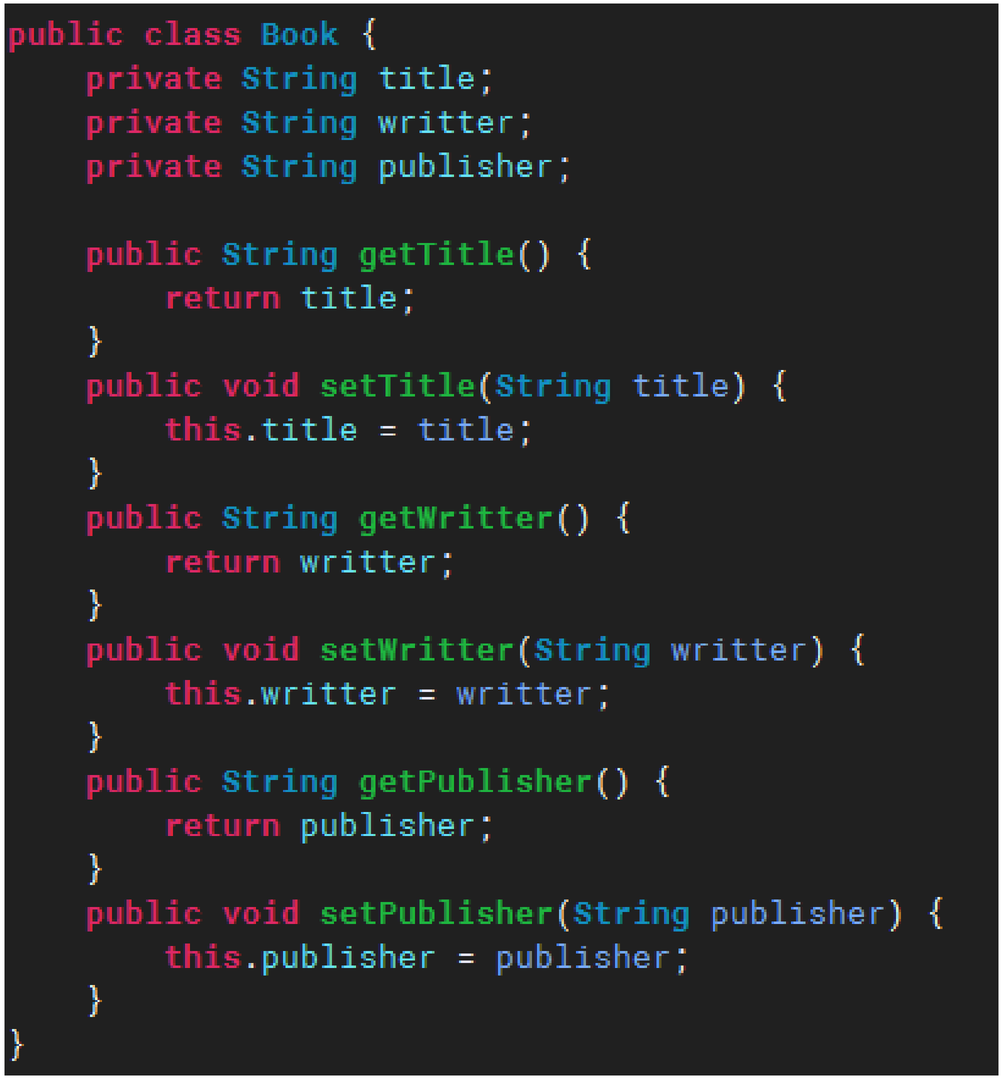
```
public class Main {
    public static void main(String args[]) {
        Human tony = new Human();

        System.out.println(tony.nama);

        tony.thinking("OOP");
    }
}
```

- Initiation, with method constructor
- Calling atributte
- Calling method

DEMO

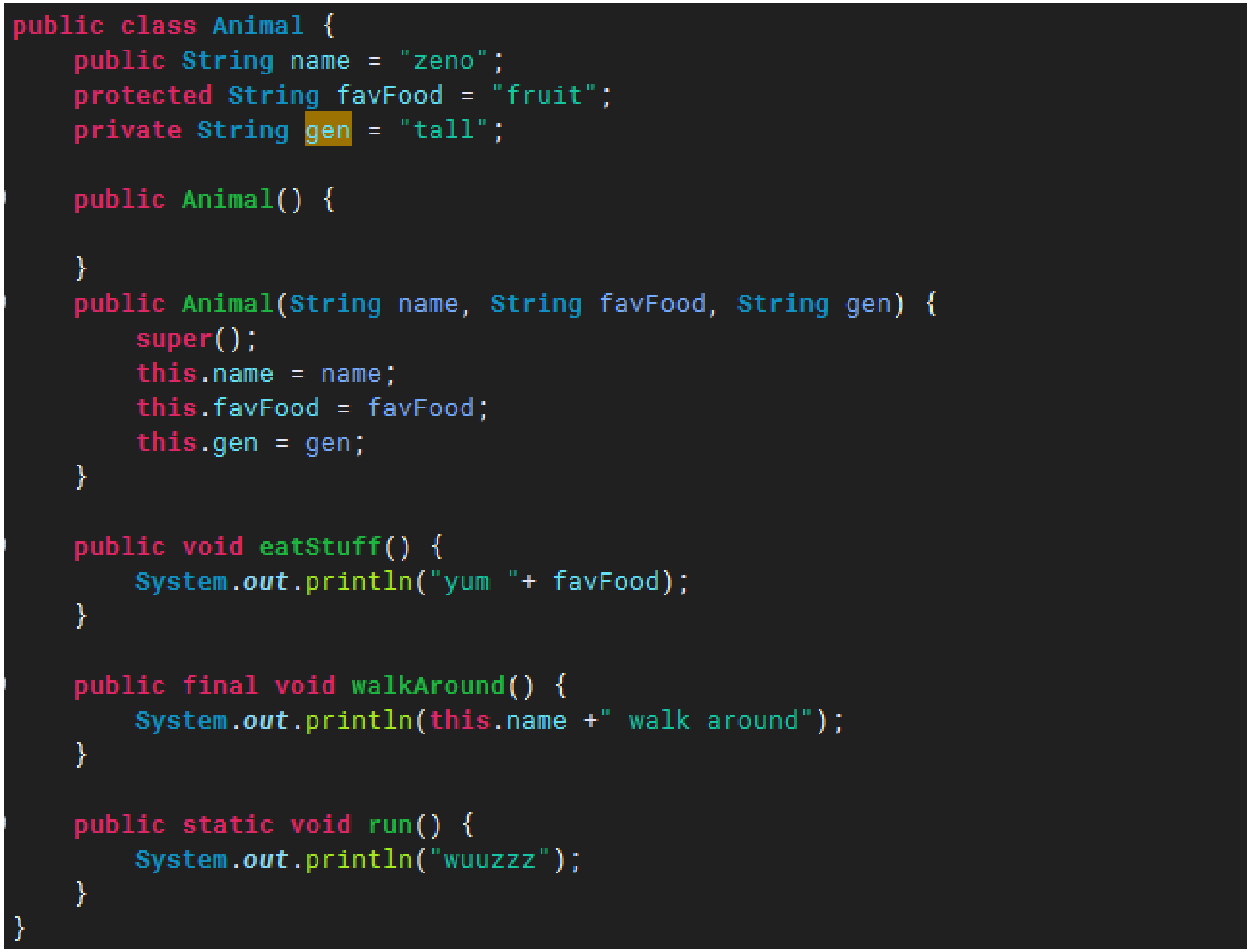
5
BEACH

Encapsulation



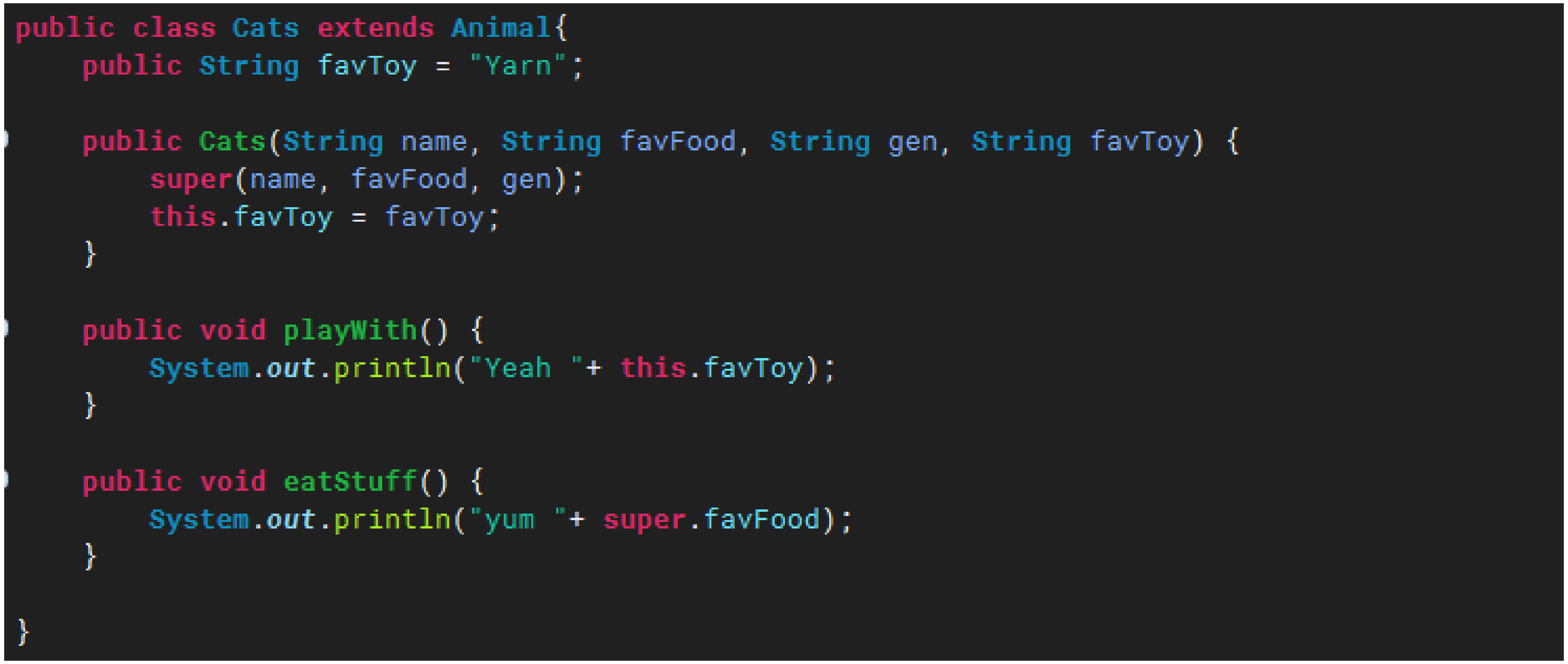
```java
public class Book {
    private String title;
    private String writter;
    private String publisher;

    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getWritter() {
        return writter;
    }
    public void setWritter(String writter) {
        this.writter = writter;
    }
    public String getPublisher() {
        return publisher;
    }
    public void setPublisher(String publisher) {
        this.publisher = publisher;
    }
}
```

Inheritance



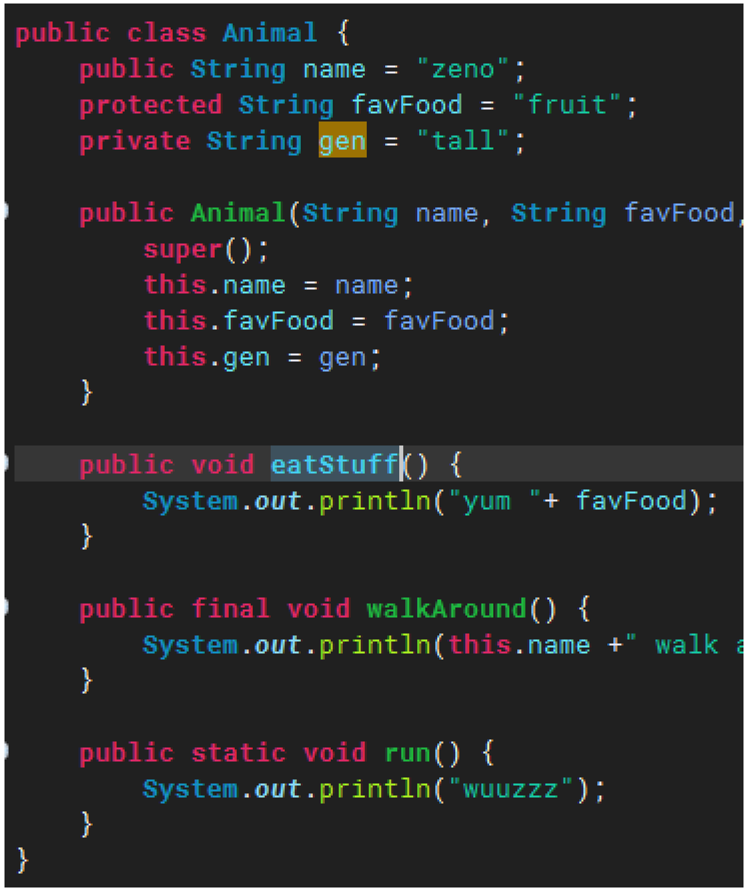
```java
public class Animal {
    public String name = "zeno";
    protected String favFood = "fruit";
    private String gen = "tall";

    public Animal() {

    }
    public Animal(String name, String favFood, String gen) {
        super();
        this.name = name;
        this.favFood = favFood;
        this.gen = gen;
    }

    public void eatStuff() {
        System.out.println("yum "+ favFood);
    }

    public final void walkAround() {
        System.out.println(this.name +" walk around");
    }

    public static void run() {
        System.out.println("wuuzzz");
    }
}
```

Inheritance

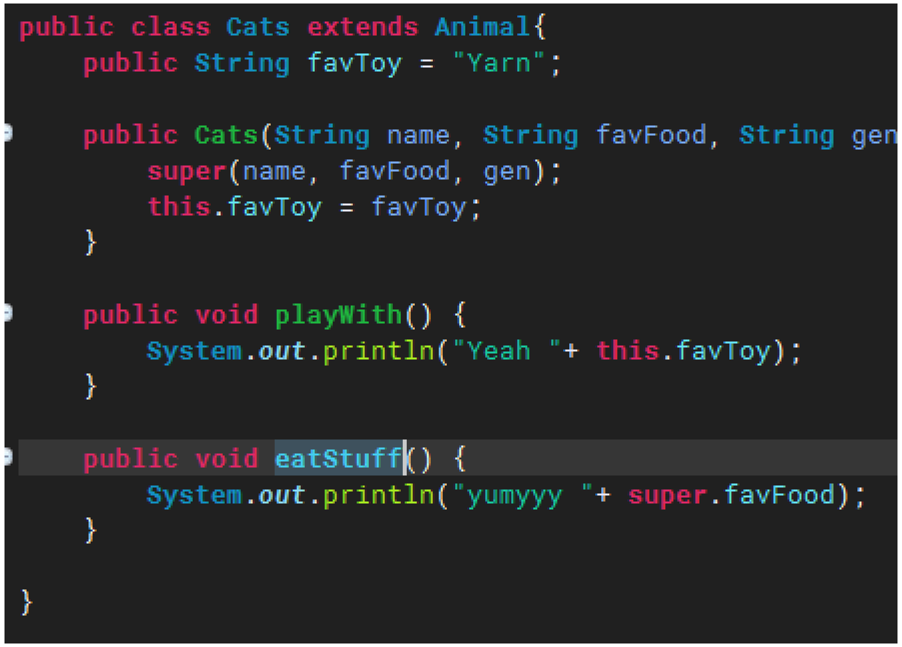


```
public class Cats extends Animal{
    public String favToy = "Yarn";

    public Cats(String name, String favFood, String gen, String favToy) {
        super(name, favFood, gen);
        this.favToy = favToy;
    }

    public void playWith() {
        System.out.println("Yeah "+ this.favToy);
    }

    public void eatStuff() {
        System.out.println("yum "+ super.favFood);
    }

}
```

Polymorphism

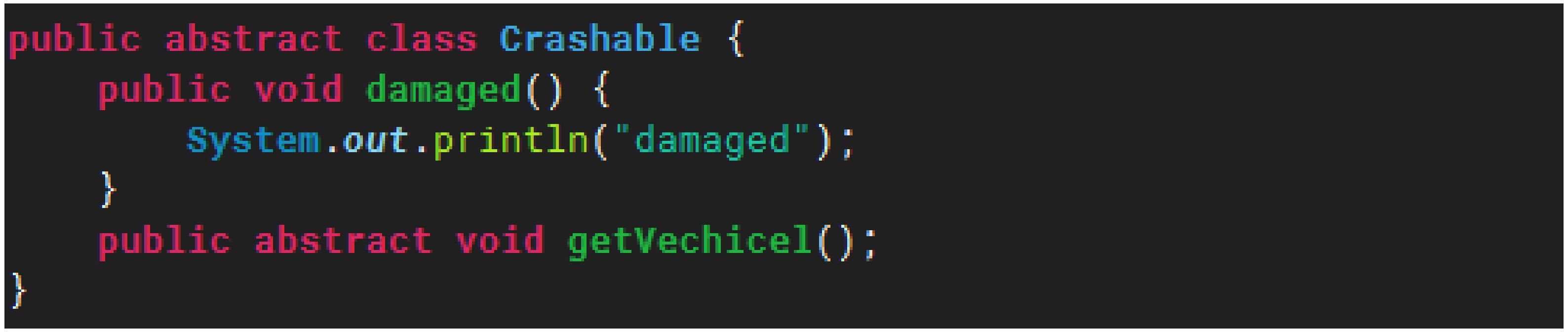


```java
public class Animal {
    public String name = "zeno";
    protected String favFood = "fruit";
    private String gen = "tall";

    public Animal(String name, String favFood,
        super();
        this.name = name;
        this.favFood = favFood;
        this.gen = gen;
    }

    public void eatStuff() {
        System.out.println("yum "+ favFood);
    }

    public final void walkAround() {
        System.out.println(this.name +" walk a
    }

    public static void run() {
        System.out.println("wuuzzz");
    }
}
```



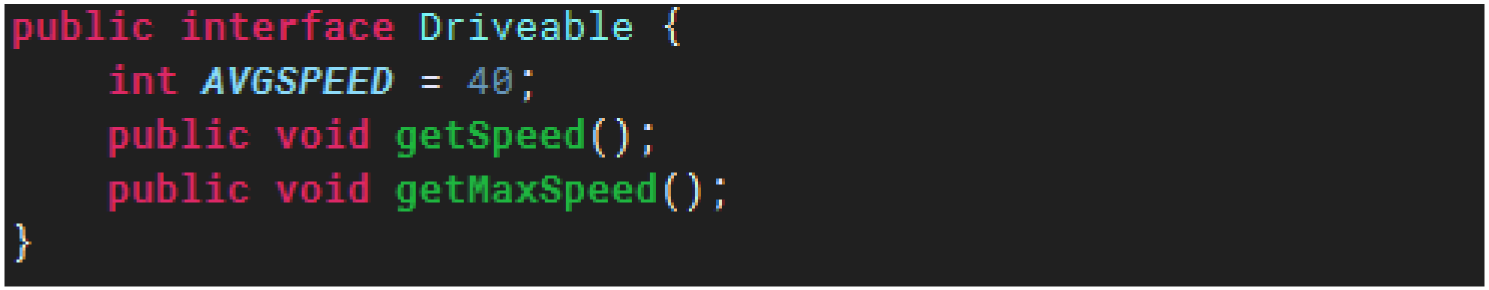
```java
public class Cats extends Animal{
    public String favToy = "Yarn";

    public Cats(String name, String favFood, String gen
        super(name, favFood, gen);
        this.favToy = favToy;
    }

    public void playWith() {
        System.out.println("Yeah "+ this.favToy);
    }

    public void eatStuff() {
        System.out.println("yumyyy "+ super.favFood);
    }

}
```

Abstract class

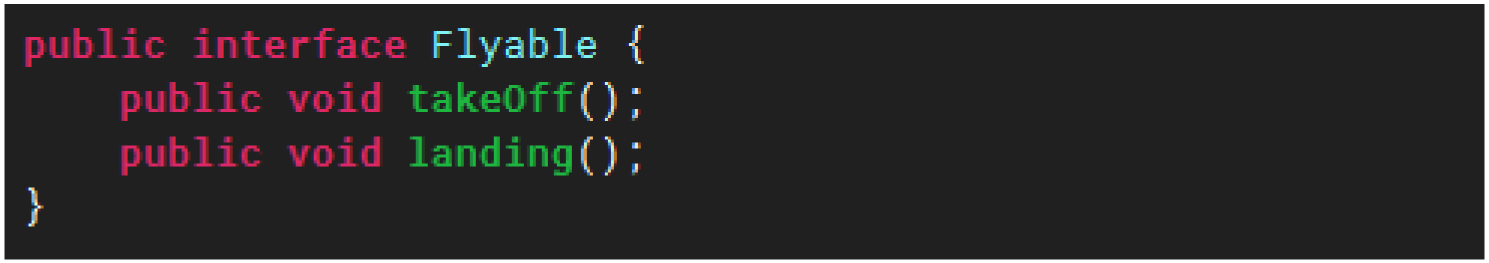


```java
public abstract class Crashable {
    public void damaged() {
        System.out.println("damaged");
    }
    public abstract void getVechicel();
}
```

Interface

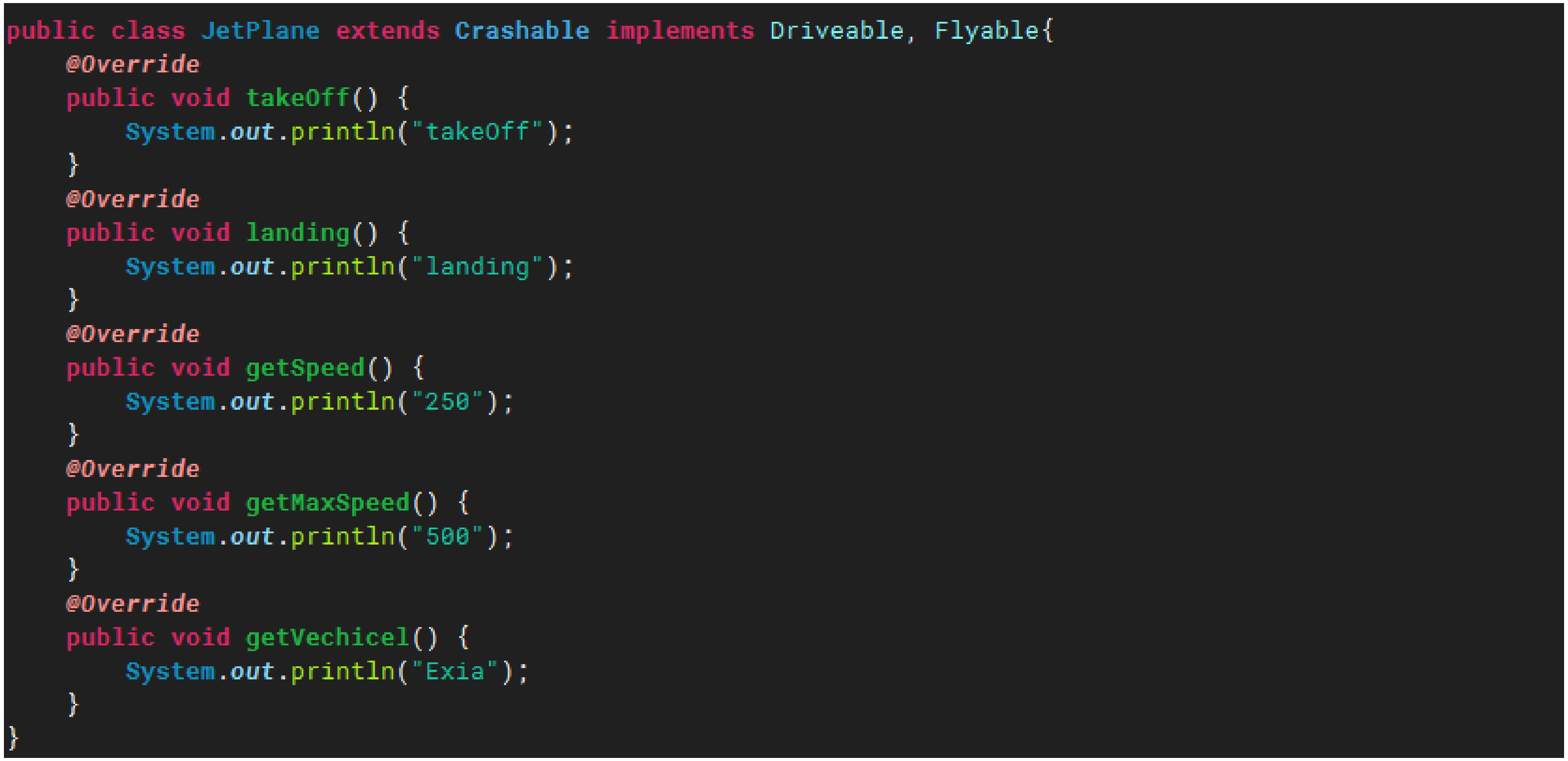


```java
public interface Driveable {
    int AVGSPEED = 40;
    public void getSpeed();
    public void getMaxSpeed();
}
```



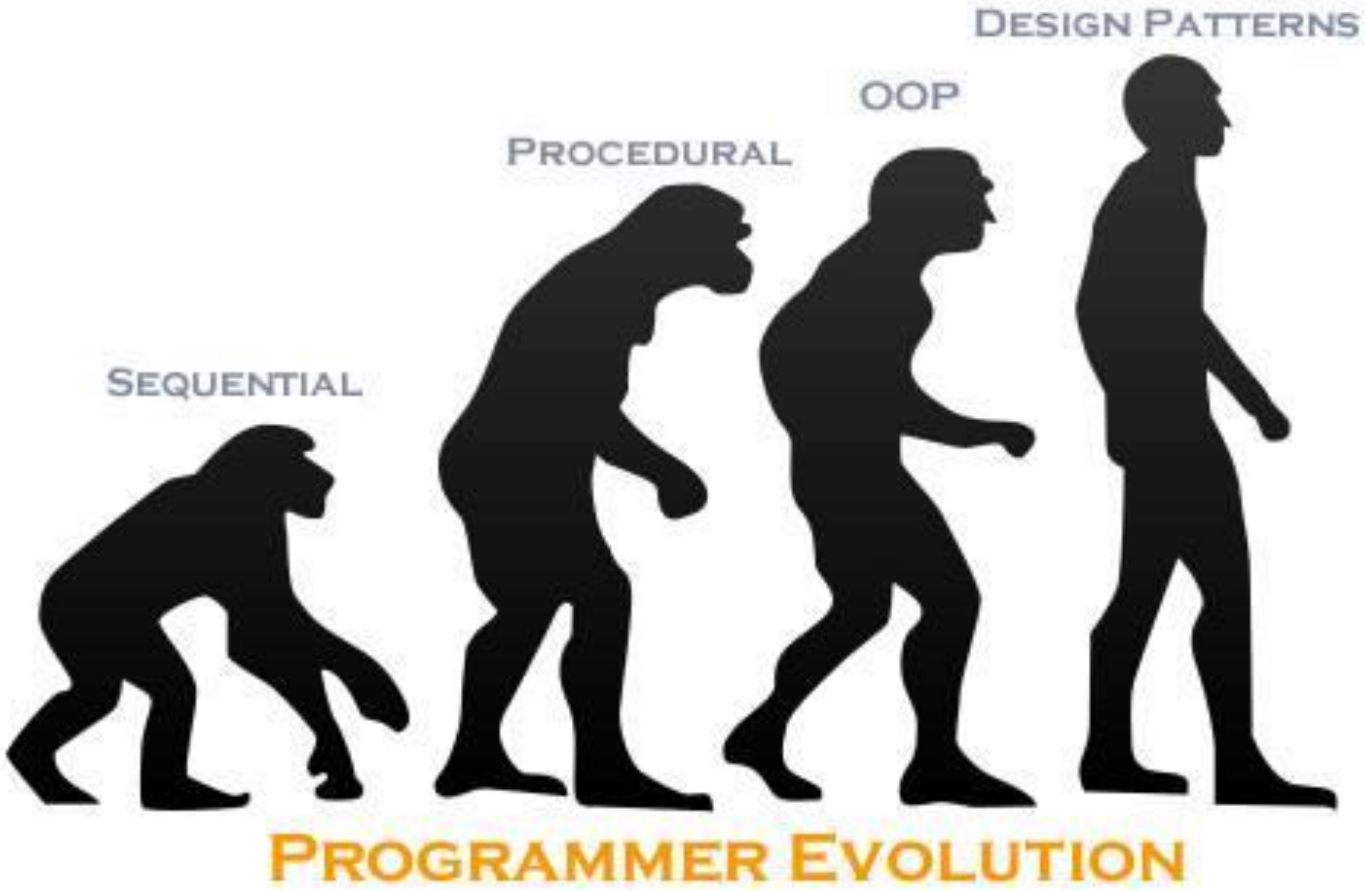
```java
public interface Flyable {
    public void takeOff();
    public void landing();
}
```

How to use them all



```
public class JetPlane extends Crashable implements Driveable, Flyable{
    @Override
    public void takeOff() {
        System.out.println("takeOff");
    }
    @Override
    public void landing() {
        System.out.println("landing");
    }
    @Override
    public void getSpeed() {
        System.out.println("250");
    }
    @Override
    public void getMaxSpeed() {
        System.out.println("500");
    }
    @Override
    public void getVechicel() {
        System.out.println("Exia");
    }
}
```

DEMO


Sequential
Procedural
OOP
Design Patterns
Programmer Evolution

?

Stay in Touch…

- FB : Gusti Alfian Miftah Pratama
- Twitter : @AlfinKima