পাঠ ৫.১: ইনহেরিট্যান্স

ইনহেরিট্যান্স-

এবার আমরা অবজেক্ট ওরিয়েন্টেড কনসেপ্ট-এর আরও ভেতরে প্রবেশ করবো। শুরুতেই আমরা ইনহেরিটেন্স নিয়ো আলোচনা করি।

ইনহেরিটেন্স নিয়ে কথা বলতে হলে এর সাথে আরেকটি বিষয় চলে আসে সেটি হলো অবজেক্ট কম্পোজিশান। এটি মোটামুটিভাবে একটু কঠিন অন্যান্য টপিক থেকে। তাই এই টপিকটি পড়ার সময় একটু ধৈর্য্য নিয়ে পড়তে হবে।

তো শুরু করার যাক-

প্রথমেই আমরা কথা বলবো Is - A এবং Has – A নিয়ে।

যেহেতু আমরা জাভা প্রোগ্রামিং শুরু করেছি, তো আমরা যতই এর ভেতরে প্রবেশ করতে শুরু করবো, ততই বুঝতে শুরু করবো যে ক্লাস আসলে একটা স্ট্যান্ড এলোন কম্পোনেন্ট নয়, বরং এটি অন্যান্য ক্লাসের উপর নির্ভর করে। অর্থাৎ ক্লাস গুলো একটি রিলেশন মেইনটেইন করে চলে। এই রিলেশন গুলো সাধারণত দুই ধরণের হয়- Is - A এবং Has – A।

আমাদের বাস্তব জগৎ থেকে একটা এনালজি দেয়া যাক । যেমন একটি বিড়াল, কিংবা কার অথবা বাস। বিড়াল হচ্ছে একটি প্রাণি। কার এর থাকে চাকা এবং ইঞ্জিন। বাস এরও থাকে চাকা এব ইঞ্জিন। আবার কার এবং বাস দুটিই ভেহিকল বা যান।

এখানে যে উদাহরণ গুলো দেয়া হয়েছে এর সবগুলো মূলত Is - A অথবা Has – A রিলেশনশিপ মেইনটেইন করে । যেমন -

A cat is an Animal (বিড়াল একটি প্রাণি ।) A car has wheels (কার এর চাকা আছে ।) A car has an engine (কার এর একটি ইঞ্জিন আছে ।)

তো ব্যাপারটি একদম সহজ। ঠিক এই ব্যপারটিকে আমরা আমাদের অবজেক্ট ওরিয়েন্টেট কনসেপ্ট এর মাধ্যমে বলতে পারি। যখন কোন অবজেক্ট এর মাঝে Is – A এই সম্পর্কটি দেখবো তাকে বলবো ইনহেরিটেন্স। আবার যখন কোন অবজেক্ট এর মাঝে Has – A এই সম্পর্কটি দেখবো তখন সেই ব্যপারটিকে বলবো অবজেক্ট কম্পোজিশান।

ইনহেরিটেন্স মূলত একটি ট্রি-রিলেশানশিপ। অর্থাৎ এটি একটি অবজেক্ট থেকে ইনহেরিট করে আসে।

আর যখন আমরা অনেকগুলো অবজেক্ট নিয়ে আরেকটি অবজেক্ট তৈরি করবো তখন সেই নতুন অবজেক্ট হলো মেইড-আপ বা নতুন তৈরি করা অবজেক্ট এই ঘটনাটি হলো কম্পোজিশান।

এর সবই আসলে একটি কনসেপ্ট এবং আইডিয়া থেকে এসেছে, সেটি হলো কোড রিইউজ করা এবং সিম্পল করা। যেমন দুটি অবজেক্ট এর কোড এর কিছু অংশ যদি কমন থাকে তাহলে আমরা সেই অংশটিকে দুইটি ক্লাসের মধ্যে পুনরায় না লিখে বরং তাকে ব্যবহার করতে পারি।

ধরা যাক, আমরা দুটি অবজেক্ট তৈরি করতে চাই- Animal এবং Cat

আমরা জানি যে সব Animal খায়, ঘুমায়। সুতরাং আমরা এই ক্লাসে এই দুটি বৈশিষ্ট্য আমরা এই ক্লাসে লিখতে পারি। আবার যেহেতু আমরা জানি যে Cat হচ্ছে একটি Animal। সুতরাং আমরা যদি এমন ভাবে কোড লিখতে পারি, যাতে করে এই Cat ক্লাসের মধ্যে নতুন করে আর সেই দুটি বৈশিষ্ট্যের কোড আর লিখতে হচ্ছে না, বরং আমর এই Animal ক্লাসটিকে রিইউজ করলাম, তাহলে যে ঘটনাটি ঘটবে তাকেই মূলত ইনহেরিটেন্স বলা হয়।

এইভাবে আমরা আরও অন্যান্য Animal যেমন, Dog, Cow ইত্যাদি ক্লাস লিখতে পারি।

কম্পোজিশান তুলনামূলক ভাবে একটু সহজ।

যেমন আমরা একটি Car তৈরি করতে চাই। Car তৈরি করতে হলে আমাদের লাগবে Wheel এবং Engine. সুতরাং আমরা Wheel এবং Engine এই দুটি ক্লাসকে নিয়ে নতুন আরেকটি ক্লাস লিখবো।

এবার তাহলে একটি উদাহরণ দেখা যাক।

```
public class Bicycle {

    // the Bicycle class has three fields
    public int cadence;
    public int gear;
    public int speed;

    // the Bicycle class has one constructor
    public Bicycle(int startCadence, int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }

    // the Bicycle class has four methods
    public void setCadence(int newValue) {
```

```
            cadence = newValue;
        }

        public void setGear(int newValue) {
            gear = newValue;
        }

        public void applyBrake(int decrement) {
            speed -= decrement;
        }

        public void speedUp(int increment) {
            speed += increment;
        }
    }
```

উপরের Bicycle ক্লাসটিতে তিনটি ফিল্ড এবং চারটি মেথড আছে। এবার এই Bicycle থেকে আমরা এর একটি সাব-ক্লাস লিখবো-

```
    public class MountainBike extends Bicycle {

        // the MountainBike subclass adds one field
        public int seatHeight;

        // the MountainBike subclass has one constructor
        public MountainBike(int startHeight,
                            int startCadence,
                            int startSpeed,
                            int startGear) {
            super(startCadence, startSpeed, startGear);
            seatHeight = startHeight;
        }

        // the MountainBike subclass adds one method
        public void setHeight(int newValue) {
            seatHeight = newValue;
        }
    }
```

এই MountainBike ক্লাসটি উপরে Bicycle এর সব ফিল্ড এবং মেথড গুলো ইনহেরিট করে এবং এতে নতুন করে শুধু একটি ফিল্ড এবং একটি মেথড লেখা হয়েছে। তাহলে আমাদের MountainBike ক্লাসটিতে Bicycle ক্লাসটির সব প্রোপ্রাটি এবং মেথড অটোম্যাটিক্যালি পেয়ে গেলো।

এখানে এ Bicycle হচ্ছে সুপার ক্লাস(Super Class) এবং MountainBike হচ্ছে সাব-ক্লাস(Sub Class) । অর্থাৎ যে ক্লাস থেকে ইনহেরিট করা হয় তাকে বলা হয় সুপার ক্লাস এবং যে ক্লাস সাব ক্লাস থেকে ইনহেরিট করে

মেথড অভাররাইডিং(Method Overriding)

যদিও সাব-ক্লাস সুপার-ক্লাসের সব গুলো প্রোপ্রার্টি এবং মেথড ইনহেরিটর করে, তবে সাব-ক্লাসে সুপার ক্লাসের যে কোন প্রোপ্রার্টি বা মেথড কে অভাররাইড করা যায়।

একটি উদাহরণ দেখা যাক-

```
public class Circle {
    double radius;
    String color;

    public Circle(double radius, String color) {
        this.radius = radius;
        this.color = color;
    }

    public Circle() {
        radius = 1.0;
        color = "RED";
    }

    public double getArea() {
        return radius * radius * Math.PI;
    }
}
```

এই ক্লাসটিতে `getArea()` মেথড একটি বৃত্তের ক্ষেত্রফল রিটার্ন করে।
এখন আমরা এই ক্লাসটিকে এক্সটেন্ড(`extends`) করে নতুন আরেকটি ক্লাস লিখবো-

```
public class Cylinder extends Circle {
    double height;

    public Cylinder() {
        this.height = 1.0;
    }

    public Cylinder(double radius, String color, double height) {
        super(radius, color);
        this.height = height;
    }

  @Override
    public double getArea() {
        return 2 * Math.PI * radius * height + 2 * super.getArea();
    }
}
```

এই ক্লাসটিতে `Circle` এর মেথডটি আমরা সাধারণ ভাবেই পেয়ে যাবো। `Cylinder` এর ক্ষেত্রফল নির্ধারণ করতে হলে `getArea()` কল করলেই হয়ে যাচ্ছে। কিন্তু আমরা জানি যে `Circle` এবং `Cylinder` এর ক্ষেত্রফল একভাবে নির্ধারণ করা যায় না। এক্ষেত্রে আমরা যদি `Circle` এর মেথডটি কে ব্যবহার করি তাহলে আমাদের ক্ষেত্রফলের মান ভুল আসবে। এই সমস্যা সমধাণ করার জন্যে আমরা আমাদের `Cylinder` ক্লাসটিতে `getArea()` মেথডটিকে পুনারায় লিখেছি।

এখানে লক্ষ্য রাখতে হবে যে, দুটি মেথড এর সিগনেচার, রিটার্ন-টাইপ এবং প্যারামিটার লিস্ট একই রকম হতে হবে।

এখন আমরা যদি `Cylinder` ক্লাস-এর `getArea()` মেথড কল করি, তাহলে অভারাইডেড মেথডটি কল হবে।

অ্যানোটেশান(Annotation) @Override

`@Override` এই অ্যানোটেশানটি জাভা 1.5 ভার্সনে প্রথম নিয়ে আসা হয়। কোন মেথডকে যদি আমরা অভাররাইড করি তাহলে সেই মেথড এর উপরে `@Override` দেয়া হয়। এটি কম্পাইলারকে ইনফর্ম করে যে, এই মেথটি সুপার ক্লাসের অভাররাইডেড মেথড। তবে এটি অপশনাল হলেও অবশ্যই ভাল যদি ব্যবহার করা হয়।

`super` কিওয়ার্ড

আমরা যদি সাব ক্লাস থেকে সুপার ক্লাসের কোন মেথড বা ভেরিয়েবল একসেস করতে চাই তাহলে আমরা এই কিওয়ার্ডটি ব্যবহার করি। কিন্তু আমরা জানি যে সাব ক্লাসে অটোমেটিক্যালি সুপার ক্লাসের সব প্রোপ্রার্টি চলে আসে তাহলে এর প্রয়োজনিয়তা নিয়ে প্রশ্ন হতে পারে।

আমরা আবার উপরের Cylinder ক্লাসটি আবার দেখি। এই ক্লাসটিতে আমরা নতুন আরেকটি মেথড লিখতে চাই।

```
      public double getVolume() {
            return getArea() * height;
        }
```

এই মেথডটি-তে `getArea() * height` এই স্ট্যাটমেন্টটি লক্ষ করি। এখানে `getArea()` এই মেথটি কল করা হয়েছে। আমাদের এই `Cylinder` ক্লাসটিতে `getArea()` মেথডটিকে আমরা `Circle` ক্লাস এর `getArea()` মেথড-কে অভারাইড করেছি। সুতরাং আমরা যখন এই `Cylinder` ক্লাস থেকে `getArea()` মেথডটি কল করবো তখন আসলে `Cylinder` ক্লাস এর মেথডটি কল হবে।
কিন্তু এক্ষেত্রে আমাদের একটি সমস্যা হচ্ছে যে - আমরা জানি সিলিন্ডারের আয়তন

```
V   = Pi * r^2 * h
    =  (Pi * r^2) * h
    =  Area of Circle * h
```

সুতরাং Cylinder ক্লাসের `getArea()` মেথডটি ব্যবহার করা যাচ্ছে না। কারণ সিলিন্ডারের ক্ষেত্রফল-

```
        A = (2 * Pi  * r * h)  +  (2 * Pi * r^2)
```

কিন্তু আমরা যদি `Circle` ক্লাস এর মেথডটি ব্যবহার করি তাহলে আমাদের সমস্যা সমাধান হয়ে যায়। এখন যদি আমরা সুপার ক্লাস(Circle) এর মেথডটি কল করে এই মেথডটি লিখতে চাই তাহলে -

```
     public double getVolume() {
```

```
            return super.getArea() * height;
     }
```

অর্থাৎ সাব ক্লাসে যদি মেথড অভাররাইড করা হয় এবং তারপরেও কোন কারণে যদি আমাদের সুপার ক্লাসের মেথড কে কল করার প্রয়োজন হয় তাহলে আমরা সুপার(`super`) কিওয়ার্ডটি ব্যবহার করি।
ইনহেরিটেন্স এর ক্ষেত্রে মনে রাখতে হবে –

জাভা মাল্টিপল ইনহেরিটেন্স সাপোর্ট করে না। এর মানে হচ্ছে আমার একটি ক্লাস শুধুমাত্র একটি ক্লাসকেই ইনহেরিট করতে পারে।

কনস্ট্রাক্টর অভাররাইডিং(Constructor Overriding)

মেথডের মত কনস্ট্রাকটরও অভাররাইড এবং ওভারলোড (Overload) করা যায় । অভাররাইড করা যায় বলতে অনেক ক্ষেত্রে কনস্ট্রাকটর অভাররাইড ম্যান্ডেটরি । প্যারেন্ট ক্যালে যদি এমন কোন কনস্ট্রাক্টর থাকে যেটি প্রাইভেট নয় এবং যেটি এক বা একাধিক ইনপুট প্যারামিটার নিয়ে থাকে তবে চাইল্ড ক্লাসে অবশ্যই সেই কনস্ট্রাক্টর অবশ্যই অভাররাইড করতে হবে । এটা বাধ্যতামূলক । একটা ছোট্ট উদাহরন দিয়ে বিষয়টি আমরা পরিষ্কার করে নিতে পারি ।

```
class Animal {

    String animalName ;
    String animalColor ;

    public Animal(String animalName, String animalColor){

        this.animalName = animalName;
        this.animalColor = animalColor;
    }

    public void showName(){
        System.out.println("Animal Name is: "+this.animalName);
    }
    public void showColor(){
        System.out.println("Animal Color is: "+this.animalColor);
    }
}

class Cow extends Animal{

    private String work ;
    public Cow(String animalName, String animalColor) {
//        this.work = "No Work";//This is not valid
        super(animalName, animalColor);//super in constructor have to be on
top
        this.work = "Gives Milk";//This is valid
    }

    @Override
    public void showColor() {
```

```
        System.out.println("Before showColor in child");
        super.showColor();
    }

    @Override
    public void showName() {
        super.showName();
        System.out.println("After showName in child");
    }
    public void showDescription(){
        this.showName();
        System.out.println("Animal Work is: "+this.work);
        this.showColor();
    }
}


public class Main {

    public static void main(String[] args) {

        Cow cow = new Cow("White Cow", "White");
        cow.showDescription();
    }
}
```

উপরের কোডটি মন দিয়ে লক্ষ করুন । প্যারেন্ট ক্লাস `Animal` এর মাঝে একটি পাবলিক কনস্ট্রাক্ট আছে যেটি দুটি প্যারামিটার নিয়ে থাকে । তাই এটার চাইল্ড ক্লাসেও আমাদের অবশ্যই একটি কনস্ট্রাক্টর থাকতে হবে যেটির মাঝে প্যারেন্ট ক্লাসের ওই কনস্ট্রাকটর ইনভোক করতে হবে । এটা ম্যান্ডেটরি । এটি না করলে কোড কম্পাইলেশন এরর শো করবে এবং কম্পাইলই হবেনা । আরো একটি বিষয় চাইল্ড ক্লাসের কনস্ট্রাক্টরের মাঝ থেকে প্যারেন্ট ক্লাসের কনস্ট্রাক্টরকে `super` কিয়াওর্ড দিয়ে কল করতে হবে তবে , `super` কিয়াওর্ড অবশ্যই সবার উপর থাকতে হবে । এমনকি একটি প্রিন্ট স্টেটমেন্টও থাকতে পারবে না । `super` এর পর যা খুশি থাকতে পারে কোন সমস্যা নাই । এছাড়া অন্য একাধিক কনস্ট্রাক্টর ডিক্লেয়ার করার প্রয়োজন হলে সেটাও করতে পারবেন , এটাকে বলা হবে কনস্ট্রাক্টর ওভারলোডিং । যথারীতি এর মাঝেও `super` বাবাজি অধিপত্য বিরাজ করে বসে থাকবে ।
তবে মেথড আর কনস্ট্রাক্টরের ওভাররাইডিং এর মাঝে এটা বড় একটা পার্থক্য যে মেথডের ক্ষেত্রে সুপার আপনারা কাজের সুবিধার জন্য যেকোন যায়গার ব্যাবহার করতে পারবেন । তবে কনস্ট্রাক্টরের ক্ষেতে আঈন খুবই কঠিন ।

অভারলোডিং অফ মেথড & কনস্ট্রাক্টর (Overloading of method and constructor):

অভারলোডিং বলতে খুব সাধারন ভাষায় বোঝায় একই নামের এবং একই রিটার্ন টাইপের (নাও হতে পারে) একাধিক মেথড বা কনস্ট্রাক্টর (এক্ষেত্রে কোন রিটার্ন টাইপ থাকবে না) বিদ্যমান থাকা । তার অর্থ দাড়ালো একই ক্লাসের মাঝে একই নামের এবং রিটার্ন টাইপের একাধিক মেথড বা কনস্ট্রাক্টর বিদ্যমান থাকবে । বিষয়টা একটু গোলমেলে মনে হচ্ছে তাইনা ? আসলে তেমন কিছুই নয় , বরং বিষয়টি অন্য অনেক কিছুর থেকেও অনেক বেশি সরল । একই নামের মেথড বা কনস্ট্রাক্ট থাকলেও তাদের ইনপুট প্যারামিটার কিন্তু একই হবেনা । ইনপুট টাইপ হয়ত ভিন্ন

টাইপের হবে নাহয় একটি মেথড থেকে অন্যা মেথডের ইনপুট প্যারামিটার সংখ্যা ভিন্ন হবে । উদাহরন সহকারে আমরা আমাদের কনফিউশন দুর করতে পারি । চলুন একটি উদাহরন দেখে নেওয়া যাকঃ

```
public class Main {

    private int initialNumber;
    private int terminalNumber;

    public Main(int initialNumber, int terminalNumber) {

        this.initialNumber = initialNumber;
        this.terminalNumber = terminalNumber;
    }

    public Main(int terminalNumber) {

        this(0, terminalNumber);
    }

    public Main() {

        this(0, 100);
    }

    public void showNumbers() {

        System.out.println("First Number: " + this.initialNumber + ", Second
Number: " + this.terminalNumber);
    }

    public static void main(String[] args) {

        Main m = new Main(1, 5);
        m.showNumbers();
        Main m2 = new Main(5);
        m2.showNumbers();
        Main m3 = new Main();
        m3.showNumbers();
    }
}
```

উপরের কোড সেগমেন্টটিতে আরা দেখতে পারছি যে একই `Main` ক্লাসে একই নামের কনস্ট্রাক্টর ৩ টি । খেয়াল করলে দেখা যাবে যে ৩ টি কনস্ট্রাক্টর প্রায় একই কাজ করলেও তাদের ইনপুট প্যারামিটার কিন্তু একই নয় । একেক জন একেক রকম ইনপুট নিয়ে কাজ করছে । এভাবে একই ক্লাসের মাঝে একাধিক কাজের জন্য একাধিক কনস্ট্রাক্টর ব্যাবহার করাকে বলাহয় কনস্ট্রাক্টর অভারলোডিং । যেখানে একই কনস্ট্রাক্টরের লোড অভার হয়ে গিয়েছে :P ওকে, এবার আসা যাক মেথড অভারলোডিং বিষয়ে । কনস্ট্রাক্টরের মত মেথড অভারলোডিংও সেম ম্যাকানিজম ফলো করে । একটি উদাহরন দিলেই বিষয়টি পরিষ্কার হয়ে যাবেঃ

```
public class Main {
```

```
    private int sum(int a, int b){
        return a+b;
    }
    private int sum(int a, int b, int c){
        return a+b+c;
    }
    private int sum(int ... a){

        int result = 0;
        for(int x : a){
            result+=x;
        }
        return result;
    }

    public static void main(String[] args)  {

        Main m = new Main();
        System.out.println(m.sum(3,  5));
        System.out.println(m.sum(3,  5,  7));
        System.out.println(m.sum(3,  5,  7,  17));
        System.out.println(m.sum(3,  5,  7,  17,  23));
    }
}
```

উপরোক্ত কোডটিতে দেখুন `sum` মেথডটি ৩ বার লেখা হয়েছে । মেথডের আইডেন্টিফায়ার, রিটার্ন টাইপ সবই ঠিক আছে তবু কাজ করছে ! হ্যা কারন আপনার ইনপুর প্যারামিটার ভিন্ন দিয়েছি । প্রথম `sum` মেথড কেবল ২ টি নাম্বারের যোগ করে দিতে পারে । দ্বিতীয়টি পারে ৩ টি নাম্বারের , আর শেষেরটি পারে যত সংখ্যক ইন্টিজার নাম্বারই দেওয়া হোক না কেন সে যোগ করে রেজাল্ট দিবে । সিম্পলি এটাকেই বলা হয় মেথড ওভারলোডিং । যেখানে একই নামের একাধিক মেথড থাকে যাদের নাম এক হলেও ইনপুট প্যারামিটার বা কাজের ধরন সম্পূর্ন আলাদা হয় ।

অ্যাবস্ট্রাক্ট ক্লাস (Abstract Class):

অ্যাবস্ট্রাক্ট ক্লাস হল বিশেষ এক ধরনের ক্লাস যেটির মাঝে কমপক্ষে একটি অ্যাবস্ট্রাক্ট মেথড থাকবে । তাহলে প্রশ্ন হল অ্যাবস্ট্রাক্ট মেথড আসলে কি জিনিস । অ্যাবস্ট্রাক্ট মেথড হল এমন এক ধরনের মেথড যেটার কোন বডি নেই। সোজা কথায় যে মেথডের অ্যাক্সেস মডিফায়ার আছে, রিটার্ন টাইপ আছে, মেথডের নাম আছে, ইনপুট প্যারামিটার আছে কিন্তু কোন বডি ডিফাইন করা নেই । বডি এম্পটি বা অ্যাবস্ট্রাক্ট সেজন্য এই মেথডকে অ্যাবস্ট্রাক্ট মেথড বলা হয়েছে । চলুন ছোট্ট একটা উদাহরন দেখে পরে ব্যাখ্যার দিকে যাই ।

```
public abstract class Animal{

    public abstract String color();
    public void name(){
        System.out.println(“Tiger”);
    }
}
```

উপরোক্ত কোডটিতে `Animal` একটি অ্যাবস্ট্রাক্ট ক্লাস । অ্যাবস্ট্রাক্ট ডিক্লেয়ার করার সময় `class` কিওয়ার্ডের আগে `abstract` কিওয়ার্ডটি লিখতে হবে । অ্যাবস্ট্রাক্ট ক্লাস পাবলিক বা ডিফল্ট যেকোনটিই হতে পারে । এই ক্লাস অ্যাবস্ট্রাক্ট ডিক্লেয়ার করার কারন এর মাঝে আমরা একটি অ্যাবস্ট্রাক্ট মেথড ডিক্লেয়ার করেছি যেটির নাম `color` । সহজেই আমরা বুঝতে পারছি যে অ্যাবস্ট্রাক্ট মেথড ডিক্লেয়ার করতে গেলে তার আগে `abstract` কিওয়ার্ডটি ব্যাবহার করতে হবে । লক্ষ করে দেখুন `color` মেথডের অ্যাক্সেস মডিফায়ার, রিটার্ন টাইপ (ইনপুট প্যারামিটার দিলে দেওয়া সম্ভব) সবই আছে কিন্তু কোন বডি নেই । এজন্য এই মেথডকে বলা হয়েছে অ্যাবস্ট্রাক্ট মেথড । একটি অ্যাবস্ট্রাক্ট ক্লাসে যেমন একাধিক অ্যাবস্ট্রাক্ট মেথড থাকতে পারে তেমন সাধারন মেথডও থাকতে পারে প্রচুর পরিমান । প্রশ্ন হল কেন এই অ্যাবস্ট্রাক্ট মেথড ? অ্যাবস্ট্রাক্ট ক্লাসের কোন ইন্সট্যান্স ক্রিয়েট করা যায়না যতক্ষন না সবগুলা অ্যাবস্ট্রাক্ট মেথডকে ওভাররাইড করা হচ্ছে । অ্যাবস্ট্রাক্ট মেথড হল একটা রুলের বা নিয়মের মত । এই মেথডের মাধ্যমে বলে দেওয়া হচ্ছে যে , যে ক্লাসই এই `Animal` ক্লাসকে এক্সটেন্ড করবে তাকে অবশ্যই `color` মেথডটি ওভাররাইড করতে হবে এবং নিজস্ব কাজের উপর ভিত্তি করে তাকে । যদি `Animal` ক্লাসকে `Bird` ক্লাস এক্সটেন্ড করে কিন্তু `color` মেথডটি ওভাররাইড না করে তবে `Bird` ক্লাসটিকেও অবশ্যই অ্যাবস্ট্রাক্ট ক্লাস হতে হবে ।

```
abstract class Animal{

    abstract void color();
}

abstract class Bird extends Animal{
    //abstract void color(); is present by default
}

class Crow extends Bird{

    @Override
    void color() {
        System.out.println("Black");
    }
}
public class Main {

    public static void main(String[] args) {

        //Animal animal = new Animal(); //This is not possible
        //Bird animal = new Bird(); //This is not possible
        Crow bird = new Crow();
        bird.color();
    }
}
```

উপরোক্ত কোডটিতে আমরা `Bird` ক্লাসের অবজেক্ট কোনভাবেই ক্রিয়েট করতে পারবো না যতক্ষন পর্যন্ত না আমরা এর সব অ্যাবস্ট্রাক্ট মেথড ইম্প্লিমেন্ট করছি । সে কাজটি করা সম্ভব এই ক্লাসটিকে যদি অন্য কোন ক্লাস এক্সটেন্ড করে এবং সব অ্যাবস্ট্রাক্ট মেথড ইম্লিমেন্ট করে অথবা অবজেক্ট ক্রিয়েট করার সময় আমরা সব অ্যাবস্ট্রাক্ট মেথড ইমপ্লিমেন্ট করে দেই । প্রথম কাজটি আপনারা পারেন । এখানে ২য় উপায়টি দেখানো হলঃ

```
public class Main {

```

```
    public static void main(String[] args) {

        Bird bird = new Bird() {
            @Override
            void color() {
                System.out.println("White");
            }
        };
        bird.color();
    }
}
```

ইন্টারফেস (Interface):

ইনহেরিট্যান্সের খুব গুরুত্বপূর্ন এবং কার্যকরী একটি টার্মস হল ইন্টারফেস । ইন্টারফেস ডিক্লেয়ার করতে হয় `interface` কিওয়ার্ডটি দিয়ে । এটি `public` বা ডিফল্ট যেকোনটিই হতে পারে ক্লাসের মত । আমরা ইতোমধ্যে জেনে ফেলেছি `Abstract` ক্লাস এবং অ্যাবস্ট্রাক্ট মেথড কি জিনিস । অ্যাবস্ট্রাক্ট ক্লাস এবং মেথড বুঝে থাকলে ইন্টারফেস বুঝতে পারা খুব বেশি কঠিন কিছুই নয় । ইন্টারফেস এমন একটি ক্লাস যেখানে সবগুলা মেথডই অ্যাবস্ট্রাক্ট । অর্থাৎ ১০০% অ্যাবস্ট্রাক্ট ক্লাসকে ইন্টারফেস বলা যায় । অ্যাবস্ট্রাক্ট ক্লাসে অ্যাবস্ট্রাক্ট মেথড এবং রেগুলার মেথড দুটিই ছিল কিন্তু ইন্টারফেসে কোন প্রকার রেগুলার মেথড থাকবে না । ইন্টারফেসে কেবল অ্যাবস্ট্রাক্ট মেথডই থাকবে । চলুন আমরা একটি উদাহরন দেখে নেইঃ

```
interface Animal{

    public abstract void name(String animalName);
    String color();
}

interface Cow{
    void work();
}

public class Main implements Animal, Cow{

    public static void main(String[] args) {

        Main m = new Main();
        m.name("I don't know this :P");
        System.out.println(m.color());
        m.work();
    }

    @Override
    public void name(String animalName) {
        System.out.println(animalName);
    }

    @Override
    public String color() {
        return "Red";
    }

    @Override
```

```
    public void work() {
        System.out.println("Gives Milk");
    }
}
```

লক্ষ করুন এখানে `interface` কিওয়ার্ডটি দিয়ে দুটি ইন্টারফেস ডিক্লেয়ার করা হয়েছে যথাক্রমে `Animal` এবং `Cow` । `Animal` ইন্টারফেসের মধ্য দুটি মেথড আছে যাদের একজনে `public` এবং `abstract` ডিক্লেয়ার করা হয়েছে কিন্তু অন্য মেথডটি কেবল রিটার্নটাইপ দেওয়া হয়েছে । এটির কারন হল ইন্টারফেসের মাঝে আপনি যদি কোন মেথডের পূর্বে `public` এবং `abstract` ডিক্লেয়ার নাও করেন তবু তারা বাই ডিফল্ট পাবলিক এবং অ্যাবস্ট্রাক্ট । এবার আসি `Main` ক্লাসে । এতক্ষন আমরা যেনে এসেছি যে জাভা মাল্টিপল ইনহেরিট্যান্স সাপোর্ট করেনা তাহলে এখানে কেন দুটি ইন্টারফেস ইম্লিমেন্ট করছে ? হ্যা সেটাই করবে কারন পরে ব্যাখ্যা করা হবে । এখানে লক্ষনীয় বিষয় হল ইন্টারফেসকে কিন্তু `implements` কিওয়ার্ড দিয়ে ইমপ্লিমেন্ট করতে হয় । এখানে কিন্তু এক্সটেন্ড হবেনা । একটি ক্লাস কেবল অন্য একটি ক্লাসকে এক্সটেন্ড করতে পারবে তবে একই সাথে অন্য শুন্য , এক বা একাধিক ইন্টারফেসকেও ইমপ্লিমেন্ট করতে পারবে । একাধিক ইন্টারফেস ইমপ্লিমেন্ট করার প্রয়োজন হলে কমা (,) দিয়ে একটির পর আরেকটি যোগ করতে হবে । তবে অবশ্যই অ্যাবস্ট্রাক্ট মেথড ইমপ্লিমেন্ট করতে ভুলবেন না । :P

কয়েকটি বিষয় জেনে রাখা ভালোঃ

১) একটি ক্লাস একটি মাত্র ক্লাস বা অ্যাবস্ট্রাক্ট ক্লাসকে এক্সটেন্ড করতে পারবে । ২) একটি ক্লাস বা অ্যাবস্ট্রাক্ট ক্লাস এক বা একাধিক ইন্টারফেসকে ইমপ্লিমেন্ট করতে পারবে । ৩) একটি ইন্টারফেস এক বা একাধিক ইন্টারফেসকে এক্সটেন্ড (ইমপ্লিমেন্ট নয় কিন্তু) করতে পারবে ।

উল্লেখ্য অ্যাবস্ট্রাক্ট মেথডের মত ইন্টারফেসেরও কোন অবজেক্ট ক্রিয়েট করা যায়না । যায়না সেটা বলা ভুল তবে সরাসরি যায়না । কিভাবে যায় সেটা বোঝার জন্য আপনাদের পলিমরফিজম পর্যন্ত যাওয়া লাগবে । ;)