পাঠ ৫.২: পলিফরফিজম

পলিমরফিজম (Polymorphism)

এবার আমরা কথা বলবো পলিমরফিজম নিয়ে। শব্দটির মধ্যেই একটি বিশেষ গাম্ভীর্য আছে যা কিনা একটি সাধারণ কথোপকথনকে অনেক গুরুত্বপূর্ণ করে তুলতে পারে। তবে এটি অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর একটি বহুল ব্যবহৃত কৌশল । এই শব্দটির সহজ মানে হচ্ছে যার একাধিক রূপ আছে অর্থাৎ বহুরূপিতা।

সহজ কথায় পলিমরফিজম হল এমন একটি টেকনিক বা পদ্ধতি যেখানে আমরা একটি ক্লাস, অ্যাবস্ট্রাক্ট ক্লাস বা ইন্টারফেসের অবজেক্ট ক্রিয়েট করি তার চাইল্ড ক্লাসের কনস্ট্রাক্টরের মাধ্যমে । অর্থাৎ আমরা একটি ক্লাসের অবজেক্ট ক্রিয়েট করবো অন্য একটি ক্লাসের কনস্ট্রাক্টর কল করে । সহজ ভাষায় এটিই হল পলিমরফিজম ।

মনে করা যাক,

```
public class Liquid {
    public void swirl(boolean clockwise) {
        // Implement the default swirling behavior for liquids
        System.out.println("Swirling Liquid");
    }
}
```

এখন এর একটি অবজেক্ট তৈরি করতে চাইলে – আমাদের new অপারেটর ব্যবহার করে তা একটি ভেরিয়েবল এ রাখতে হবে।

```
    Liquid myFavoriteBeverage = new Liquid ();
```

এখানে `myFavoriteBeverage` হচ্ছে আমাদের ভেরিয়েবল যা `Liquid` অবজেক্ট এর রেফারেন্স। আমরা এখন পর্যন্ত যা যা শিখেছি সে অনুযায়ী এই স্টেন্টমেন্টটি যথার্থ। তবে আমরা এর আগের অধ্যায়ে `Is-A` সম্পর্কে জেনে এসেছি।
আমাদের জাভা প্রোগ্রামিং পলিমরফিজম সাপোর্ট করায় আমরা `myFavoriteBeverage` এই রেফারেন্সের যায়গায় `Is-A` সম্পর্কিত যে কোন টাইপ রাখতে পারি। যেমন –

```
Liquid myFavoriteBeverage = new Coffee();
Liquid myFavoriteBeverage = new Milk();
```

এখানে `Coffee` এবং `Milk` হচ্ছে`Liquid`এর সাব- ক্লাস বা টাইপ এবং `Liquid` এদের সুপার ক্লাস বা টাইপ।
পলিমরফিজম নিয়ে আরও একটু আশ্চর্য হতে চাইলে আমরা এখন একটি বিষয় জানবো যা দিয়ে আমরা কোন একটি অবজেক্ট এর কোন মেথড কল করবো তবে তা কোন ক্লাসের অবজেক্ট সেটি না জেনেই। আরেকটু পরিষ্কার করে বলি, আমরা যখন সুপার ক্লাসের এর রেফারেন্স ধরে কোন এর মেথড কল করবো তখন কিন্তু আমরা জানি না যে এটি আসলে কোন অবজেক্ট এর মেথড। যেমন-

```
    Liquid myFavoriteBeverage = // ….
```

এখানে আমাদের myFavoriteBeverage এই রেফারেন্স এ `Liquid`, `Coffee`, `Milk` এর যেকোন একটির অবজেক্ট হতে পারে। উদাহরণ -

```
    public class Coffee extends Liquid {
        @Override
        public void swirl(boolean clockwise) {
            System.out.println("Swirling Coffee");
        }
    }

    public class Milk extends Liquid{
        @Override
        public void swirl(boolean clockwise) {
            System.out.println("Swirling Milk");
        }
    }

    public class CoffeeCup {
        private Liquid innerLiquid;

        void addLiquid(Liquid liq) {
            innerLiquid = liq;
            // Swirl counterclockwise
            innerLiquid.swirl(false);
        }
    }
```

আমরা এখানে একটি `CoffeeCup` ক্লাস লিখেছি যার মাঝে `addLiquid()` নামে একটি মেথড আছে যা কিনা একটি `Liquid` টাইপ parameter নেয়, এবং সেই `Liquid` এর `swirl()` মেথড-কে কল করে।

কিন্তু আমরা আমাদের সত্যিকারের জগতে একটি কফি-কাপ এ শুধুমাত্র কফি-ই এড করতে পারি তা নয়, আমরা চাইলে যে কোন ধরণের লিকুইড এড করতে পারি, সেটি মিল্ক ও হতে পারে। তাহলে এই `addLiquid` মেথড তো শুধুমাত্র `Liquid` টাইপ parameter নেয়, তাহলে আমাদের সত্যিকারের জগতের সাথে এই প্রোগ্রামিং মডেল এর সাদৃশ্য থাকলো কোথায় ?

তবে মজার ব্যপার এখানেই, আমাদের এই `CoffeeCu`p ক্লাসটি পলিমরফিজমের ম্যজিক ব্যাবহার করে সত্যিকার অর্থেই আমাদের সত্যিকারের জগতের `CoffeeCup` এর মতোই কাজ করে।

```
    public class MainApp {
        public static void main(String[] args) {
            // First you need a coffee cup
            CoffeeCup myCup = new CoffeeCup();

            // Next you need various kinds of liquid
            Liquid genericLiquid = new Liquid();
            Coffee coffee = new Coffee();
            Milk milk = new Milk();

            // Now you can add the different liquids to the cup
            myCup.addLiquid(genericLiquid);
            myCup.addLiquid(coffee);
            myCup.addLiquid(milk);
```

```
        }
    }
```

উপরের কোড গুলোতে দেখা যাচ্ছে যে আমরা একটি `CoffeeCup` এর একটি অবজেক্ট তৈরি করে সেটি তে বিভিন্ন রকম `Liquid` এড করতে পারছি।
আরেকটু লক্ষ্য করি,

```
    void addLiquid(Liquid liq) {
            innerLiquid = liq;
            // Swirl counterclockwise
            innerLiquid.swirl(false);
      }
```

এই মেথডটিতে innerLiquid.swirl(false) যখন কল করি তখন কিন্তু আমরা জানি না যে এই innerLiquid আসলে কোন অবজেক্ট এর রেফারেন্স। এটি লিকুইড বা এর যে কোন সাব-টাইপ হতে পারে।

কিছু প্রয়োজনীয় তথ্য-

১. একটি সাব ক্লাস এর অবজেক্টকে আমরা এর সুপার ক্লাসের রেফারেন্স এ এসাইন করতে পারি। ২. সাব ক্লাসের অবজেক্টকে সুপার ক্লাসের রেফারেন্স-এ এসাইন করলে, মেথড কল করার সময় শুধু মাত্র সুপার ক্লাসের মেথড গুলোকেই কল করতে পারি। ৩. তবে সাব ক্লাস যদি সুপার ক্লাসের মেথড অভাররাইড করে, তাহলে যদিও আমরা সুপার ক্লাস এর রেফারেন্স ধরে মেথড কল করছি, কিন্তু রানটাইম-এ সাব ক্লাসের মেথডটি কল হবে। মনে রাখতে হবে এটি শুধুমাত্র মেথড অভাররাইড করা হলেই সত্য হবে।

আপ-কার্স্টিং(Upcasting) এবং ডাউনকার্স্টিং (Downcasting)

```
        Liquid liquid = new Coffee ();
```

এখানে সাব ক্লাসের অবজেক্টকে সুপার ক্লাসের রেফারেন্স এ এসাইন করা হয়ছে। একে বলা হয় আপ-কাস্টিং। এই কাস্টিং সবসময় সেইফ ধরা হয় কারণ আপকাস্টং এর ক্ষেত্রে সাব ক্লাস সবসময়ই সুপার ক্লাসের সবকিছু ইনহেরিট করে এবং কম্পাইলার কম্পাইল করার সময়-ই এ কার্স্টিং করা সম্ভব কিনা তা চেক করে থাকে।

```
        Liquid liquid = new String();
```

উপরের স্টেটমেন্টটি কম্পাইলার কম্পাইল করবে না, কারণ `String` মোটেই `Liquid` ক্লাসের সাব ক্লাস নয়। এক্ষেত্রে কম্পাইলার incompatible types ইরর দেখাবে।

হোমেজিনিয়াস কালেকশন (Homogeneous Collection):

হোমোজিনিয়াস কালেকশন হল একই ক্লাসের কিছু সংখ্যক অবজেক্টের কালেকশন । একটি উদাহরন দিয়ে বিষয়টি একটু সুরাহা করা যাকঃ

```
interface Animal {

    public abstract void name(String animalName);
}

class Cow implements Animal {

    private String animalName;

    public void work(String animalWork) {
        System.out.println("Work of " + this.animalName + " is " +
animalWork);
    }

    @Override
    public void name(String animalName) {
        this.animalName = animalName;
        System.out.println("Name of the animal is: " + this.animalName);
    }
}

public class Main {

    public static void main(String[] args) {

        Animal[] collection1 = new Cow[3];
        collection1[0] = new Cow();
        collection1[1] = new Cow();
        collection1[2] = new Cow();

        Cow[] collection2 = new Cow[3];
        collection2[0] = new Cow();
        collection2[1] = new Cow();
        collection2[2] = new Cow();
    }
}
```

লক্ষ করুন । এখানে `Cow` ক্লাসটি `Animal` ইন্টারফেসের চাইল্ড । এবং `Main` ক্লাসের `main` মেথড এর মাঝে ২ টি অবজেক্টের অ্যারে ডিক্লেয়ার করা হয়েছে । একটি `Animal` ক্লাসের অবজেক্টের অ্যারে যেটির সবগুলা অবজেক্ট `Cow` ক্লাসের কনস্ট্রাক্টর দিয়ে ইন্সট্যানশিয়েট করা হয়েছে । এখানে পলিমরফিজম স্পষ্ট । এবং অন্যটি অবজেক্ট অ্যারেটি চীরাচরিত অবজেক্ট অ্যারে । এই দুই অ্যারেই হল হোমোজিনিয়াস কালেকশনের উদাহরন । বোঝা যায়নি ? ওকে, এখানে `collection1` অ্যারেটির প্রতিটি অবজেক্টই `Cow` ক্লাসের কনস্ট্রাক্টর দিয়ে ইন্সট্যানশিয়েট করা হয়েছে । তার মানে `collection1` এর মাঝে সবগুলা অবজেক্টই একই ধরনের । যেহেতু এই অ্যারেটির সবগুলা এলিমেন্ট একই ধরনের/ক্লাসের অবজেক্ট সুতরাং এটিকে বলা হবে হোমোজিনিয়াস কালেকশন । একই কথা `collection2` এর ক্ষেত্রেও প্রোযোজ্য ।

হেটারোজিনিয়াস কালেকশন (Heterogeneous Collection):

ভিন্নধর্মী অবজেক্টের কালেকশনকেই বলা হয় হেটারোজিনিয়াস কালেকশন । হেটারোজিনিয়াস কালেকশন বুঝতে হলে আমাদের একটি উদাহরন দেখে নেওয়া উত্তমঃ

```
class Animal {

    String animalName ;
    public Animal(String animalName){
        this.animalName = animalName;
    }
    public void name(){
        System.out.println("Animal name is: "+this.animalName);
    }
}

class Cow extends Animal {

    public Cow(String animalName) {
        super(animalName);
    }

    public void work(String animalWork) {
        System.out.println("Work of " + this.animalName + " is " +
animalWork);
    }
}

class Dog extends Animal {

    public Dog(String animalName) {
        super(animalName);
    }

    public void work(String animalWork) {
        System.out.println("Work of " + this.animalName + " is " +
animalWork);
    }
}

class Cat extends Animal {

    public Cat(String animalName) {
        super(animalName);
    }

    public void work(String animalWork) {
        System.out.println("Work of " + this.animalName + " is " +
animalWork);
    }
}

public class Main {

    public static void main(String[] args) {

        Animal[] animals = new Animal[4];
        animals[0] = new Animal("Dolphin");
        animals[1] = new Cow("Big Cow");
        animals[2] = new Dog("Red Dog");
        animals[3] = new Cat("White Cat");
    }
```

```
}
```

খুব ভালোভাবে লক্ষ করুন । আমরা `Animal` ক্লাসের অবজেক্টের একটু অ্যারে ডিক্লেয়ার করেছি যার সাইজ ৪ । কিন্তু ইন্সট্যানশিয়েট করার সমস আমরা পলিমরফিজম মেকানিজম ব্যাবহার করে এর চাইল্ড ক্লাসের ভিন্ন ভিন্ন কনস্ট্রাক্টর দিয়ে ইন্সট্যানশিয়েট করেছি । অর্থাৎ `animals` অ্যারেটির প্রতিটি অবজেক্টই আলাদা আলাদা কনস্ট্রাক্ট দিয়ে ইন্সট্যানশিয়েট করা এবং তাদের বিহ্যাভিয়েরাল পার্থ্য আছে । এধরনের কালেকশনকে বলা হয় হেটারোজিনিয়াস কালেকশন ।

এবার একটু ভিন্ন পন্থায় এগোন যাক । মেইন ক্লাসটিকে আমরা একটু মডিফাই করবো । বাকী সবই ঠিক থাকবে আগের মত ।

```
public class Main {

    public static void main(String[] args) {

        Animal animal = new Cat("Cute Cat");
        animal.name();
        //animal.work("Some Work");//Not possible
        Cat cat = new Cat("Preety Cat");
        cat.name();
        cat.work("It plays");
    }
}
```

খেয়াল করে দেখুন আমরা `Animal` এবং `Cat` এর অবজেক্ট ক্রিয়েট করার সময় কনস্ট্রাক্টর ব্যাবহার করেছি `Cat` এর কিন্তু `Animal` এর অবজেক্ট থেকে আমরা `work` মেথডটি কোন ভাবেই কল করতে পারছি না বা পারবো না কিন্তু `Cat` এর অবজেক্ট থেকে ঠিকই পারছি । কারনটা কি ? কারন হল `Animal` ক্লাসের মাঝে ঠিক যে যে মেথড আছে সেগুলাকেই আমরা অ্যাক্সেস করতে পারব তবে `Cat` এর ইমপ্লিমেন্টেশন দিয়ে । `Animal` এর মাঝে নেই কিন্তু `Cat` ক্লাসে বাড়তি আছে এমন কোন মেথডকে আমরা অ্যাক্সেস করতে পারবো না । এমনকি `Animal` ক্লাসের অবজেক্টে `Cat` ক্লাসের `work` মেথডের কোন রেফারেন্সই ক্রিয়েট হবেনা ।

তাহলে এটা করি কেন আমরা ? এটা করার পেছনে বেশ কিছু কারন থাকতে পারে । প্রথমত আমরা প্যারেন্ট ক্লাস এবং চাইল্ড ক্লাসের ইমপ্লিমেন্টশন নিয়ে কাজ করতে চাইলে পলিমরফিজমের এই সুবিধাটি নেওয়া হয় । অন্য কারনটি হল মেমোরি কনজাম্পশন । ভেবে দেখুন যদি `Animal` ক্লাসে ৩ টি মেথড থাকে যেগুলার জন্য আপনি `Cat` ক্লাসের ইমপ্লিমেন্টেশন ব্যাবহার করতে চান , কিন্তু `Cat` ক্লাসের মাঝে ১৫ টির মত মেথড আছে এবং অনেক অ্যাট্রিবিউট । আপনি যদি `Cat` এর অবজেক্ট ক্রিয়েট করেন তবে মেমোরি থেকে প্রচুর স্পেস কনজিউম করবে উক্ত অবজেক্ট । অন্যদিকে আপনি যদি `Animal` এর অবজেক্ট ক্রিয়েট করেন `Cat` এর কনস্ট্রাক্টর ব্যাবহার করে তাহলে `Cat` ক্লাসের ইমপ্লিমেন্টেশন ব্যাবহার করতে পারছেন এবং মেমোরি থেকে খুব কম মেমোরি কনজিউম করছে (`Animal` মেথডগুলার জন্য প্রয়োজনীয় মোমোরি মাত্র) । কোনটি বেশি সুবিধাজনক ? এছাড়া আরো কারন আছে । পরবর্তীতে সেগুলা নিয়েও আলোচনা করা হবে ।