PPT Praktikum PBO Week 10

Inner Class & Outer Class

Inner Class

- Di Java, kita bisa membuat class di dalam class, atau disebut dengan Inner Class
- Salah satu kasus kita membuat inner class biasanya ketika kita butuh membuat beberapa class yang saling berhubungan, dimana sebuah class tidak bisa dibuat tanpa class lain
- Misal kita perlu membuat class Employee, dimana membutuhkan class Company, maka kita bisa membuat class Employee sebagai inner class Company
- Cara membuat inner class, cukup membuatnya di dalam blok class outer class nya

Mengakses Outer Class

- Keuntungan saat kita membuat inner class adalah, kemampuan untuk mengakses outer class nya
- Inner class bisa membaca semua private member yang ada di outer class nya
- Untuk mengakses object outer class nya, kita bisa menggunakan nama class outer nya diikuti dengan kata kunci this, misal Company.this
- Dan untuk mengakses super class outer class nya, kita bisa menggunakan nama class outer nya diikuti dengan kata kunci super, misal Company.super

Kelas Company

```java
package praktikum.pbo.data;

public class Company {  8 usages

  private String name;  3 usages

  public String getName() { return name; }

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

  public class Employee {  4 usages

    private String name;  2 usages

    public String getCompany() { return Company.this.name; }

    public String getName() { return name; }

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

Kelas CompanyApp

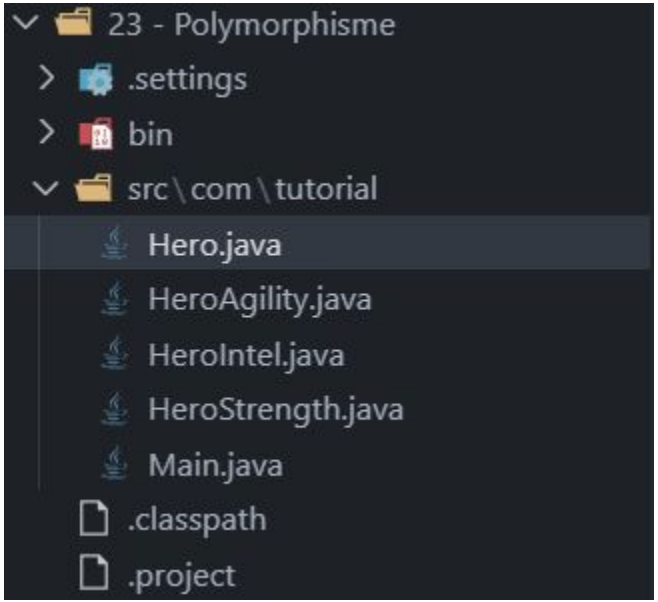
```java
package praktikum.pbo.application;
import praktikum.pbo.data.Company;

public class CompanyApp {
  public static void main(String[] args) {
    Company company = new Company();
    company.setName("Praktikum PBO");

    Company.Employee employee = company.new Employee();
    employee.setName("Farhan");

    System.out.println(employee.getName());
    System.out.println(employee.getCompany());

    Company company2 = new Company();
    company2.setName("Belum Ada");

    Company.Employee employee2 = company2.new Employee();
    employee2.setName("Budi");

    System.out.println(employee2.getName());
    System.out.println(employee2.getCompany());

  }
}
```

Polymorphism

Polymorphism

- Polymorphism berasal dari bahasa Yunani yang berarti banyak bentuk.
- Dalam OOP, Polymorphism adalah kemampuan sebuah object berubah bentuk menjadi bentuk lain
- Polymorphism erat hubungannya dengan Inheritance

Hirarki Kelas



Class Hero

```java
package com.tutorial;

Codeium: Refactor | Explain
public class Hero {
    String nama;

    Hero(String nama){
        this.nama = nama;
    }

    Codeium: Refactor | Explain | Generate Javadoc | ×
    void display(){
        System.out.println("\nName \t: " + this.nama);
    }

    Codeium: Refactor | Explain | Generate Javadoc | ×
    void attack(Hero enemy){
        System.out.println(x:"Menyerang");
    }
}
```

Class HeroAgility

```java
package com.tutorial;

Codeium: Refactor | Explain
public class HeroAgility extends Hero{
    String type = "Agility";

    HeroAgility(String nama){
        super(nama);
    }

    Codeium: Refactor | Explain | Generate Javadoc | ×
    @Override
    void display(){
        super.display();
        System.out.println("Type \t: " + this.type);
    }

    Codeium: Refactor | Explain | Generate Javadoc | ×
    void showoff(){
        System.out.println(x:"Saya hero Agility!!");
    }
}
```

Class HeroIntel

```java
package com.tutorial;

Codeium: Refactor | Explain
public class HeroIntel extends Hero{
    String type = "intel";

    HeroIntel(String nama){
        super(nama);
    }

    Codeium: Refactor | Explain | Generate Javadoc | ×
    @Override
    void display(){
        super.display();
        System.out.println("Type \t: " + this.type);
    }
}
```

Class HeroStrength

```
package com.tutorial;

Codeium: Refactor | Explain
public class HeroStrength extends Hero{
    String type = "Strength";

    HeroStrength(String nama){
        super(nama);
    }

    Codeium: Refactor | Explain | Generate Javadoc | ×
    @Override
    void display(){
        super.display();
        System.out.println("Type \t: " + this.type);
    }
}
```

Class Main (1)

```java
package com.tutorial;

Codeium: Refactor | Explain
public class Main {
    Run | Debug | Codeium: Refactor | Explain | Generate Javadoc |
    public static void main(String[] args) {
        Hero hero1 = new Hero(nama:"Ucup");
        HeroStrength hero2 = new HeroStrength(nama:"Otong");
        hero1.display();
        hero2.display();

        // Polymorphic
        Hero hero3 = new HeroAgility(nama:"Maria");
        hero3.display();

        HeroAgility hero4 = new HeroAgility(nama:"Mahmud");
        hero4.display();
        hero4.showoff();
        // tidak bisa
        // HeroIntel hero4 = new Hero("Mahmud");
        // hero4.display();
```

Class Main (2)

```
        // Array list
        Hero[] kumpulanHero = new Hero[4];
        kumpulanHero[0] = hero1;
        kumpulanHero[1] = hero2;
        kumpulanHero[2] = hero3;
        kumpulanHero[3] = hero4; // casting

        kumpulanHero[0].display();
        kumpulanHero[1].display();
        kumpulanHero[2].display();

        // method calls
        // kumpulanHero[3].showoff(); // ini tidak bisa
        hero4.showoff();

        // aplikasi
        hero1.attack(hero2);
        hero1.attack(hero3);
        hero1.attack(hero4);

    }
}
```

Type Check & Casts

Type Check & Casts

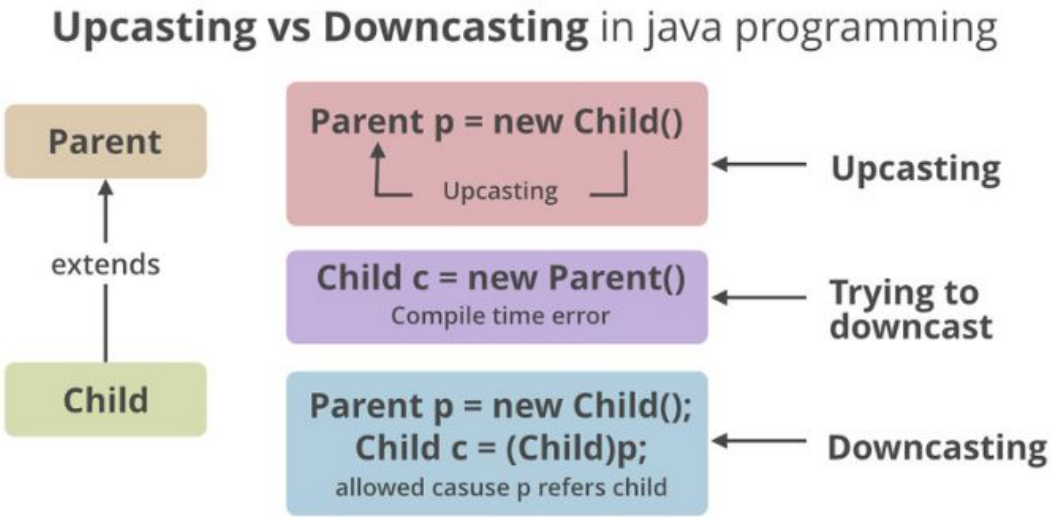
- Sebelumnya kita sudah tau cara melakukan konversi tipe data (casts) di tipe data primitif
- Casts juga bisa digunakan untuk tipe data bukan primitif
- Namun agar aman, sebelum melakukan casts, pastikan kita melakukan type check (pengecekan tipe data), dengan menggunakan kata kunci instanceof
- Hasil operator instanceof adalah boolean, true jika tipe data sesuai, false jika tidak sesuai

Object Casting

Object Casting

- Object casting adalah proses mengubah tipe referensi objek menjadi tipe lain dalam hierarki kelasnya. Ini memungkinkan akses ke metode dan atribut yang spesifik untuk tipe target.
- Casting digunakan untuk memanfaatkan polimorfisme dan memungkinkan penggunaan metode yang lebih spesifik pada subclass. Hal ini juga membantu dalam penanganan koleksi objek dengan tipe yang beragam.
- Up casting adalah proses mengonversi subclass menjadi superclassnya. Ini dilakukan secara implisit dan digunakan untuk menyederhanakan kode saat memproses objek dalam konteks yang lebih umum.
- Down casting adalah proses mengonversi superclass menjadi subclassnya. Ini harus dilakukan secara eksplisit dan memerlukan pengecekan tipe dengan instanceof untuk menghindari ClassCastException.

Upcasting Vs Downcasting



Class Employee

```
class Employee {  5 usages  2 inheritors

  String name;  7 usages

  Employee(String name) { this.name = name; }

  void sayHello(String name) { System.out.println("Hi " + name + ", My Name Is Employee " + this.name); }
}
```

Class Manager

```
class Manager extends Employee {  7 usages  1 inheritor

  String company;  1 usage

  Manager(String name) { super(name); }

  Manager(String name, String company) {  no usages
    super(name);
    this.company = company;
  }

  void sayHello(String name) { System.out.println("Hi " + name + ", My Name Is Manager " + this.name); }

}
```

Class VicePresident

```
class VicePresident extends Manager {  6 usages

  VicePresident(String name) { super(name); }

  void sayHello(String name) { System.out.println("Hi " + name + ", My Name Is VP " + this.name); }

}
```

Class PolymorphismApp (1)

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


    Employee employee = new Employee( name: "EKo");
    employee.sayHello( name: "Budi");

    employee = new Manager( name: "Eko");
    employee.sayHello( name: "Budi");

    employee = new VicePresident( name: "Eko");
    employee.sayHello( name: "Budi");

    sayHello(new Employee( name: "Eko"));
    sayHello(new Manager( name: "Joko"));
    sayHello(new VicePresident( name: "Budi"));
  }
```

Class PolymorphismApp (2)

```
  static void sayHello(Employee employee) {  3 usages
    if (employee instanceof VicePresident) {
      VicePresident vicePresident = (VicePresident) employee;
      System.out.println("Hello VP " + vicePresident.name);
    } else if (employee instanceof Manager) {
      Manager manager = (Manager) employee;
      System.out.println("Hello Manager " + manager.name);
    } else {
      System.out.println("Hello " + employee.name);
    }
  }
}
```

Latihan

Latihan 1

Anda diminta untuk mengembangkan aplikasi manajemen kendaraan di sebuah perusahaan penyewaan. Perusahaan tersebut memiliki berbagai jenis kendaraan, termasuk mobil dan truk. Semua kendaraan memiliki atribut umum seperti merek, model, dan tahun. Mobil memiliki atribut tambahan seperti jumlah kursi, sedangkan truk memiliki atribut tambahan seperti kapasitas muatan.

Tugas Anda adalah:

1. Membuat kelas Vehicle sebagai superclass dengan atribut dan metode dasar.
2. Membuat kelas Car dan Truck yang mewarisi kelas Vehicle.
3. Menyusun metode untuk memeriksa tipe objek menggunakan instanceof dan melakukan casting yang sesuai.
4. Menyusun metode untuk menghitung biaya sewa berdasarkan tipe kendaraan dan atribut spesifiknya.

Class Vehicle (1)

```java
public class Vehicle {  4 usages  2 inheritors
    private String brand;  3 usages
    private String model;  3 usages
    private int year;  3 usages

    public Vehicle(String brand, String model, int year) {  2 usages
        this.brand = brand;
        this.model = model;
        this.year = year;
    }

    public String getBrand() {  no usages
        return brand;
    }

    public String getModel() {  no usages
        return model;
    }
```

Class Vehicle (2)

```
    public int getYear() {  no usages
        return year;
    }

    public void displayInfo() {  1 usage
        System.out.println("Brand: " + brand + ", Model: " + model + ", Year: " + year);
    }

    public double calculateRentalCost() {  3 usages  2 overrides
        return 50; // Biaya dasar sewa kendaraan
    }
}
```

Class Car

```
public class Car extends Vehicle {  no usages
    private int numberOfSeats;  3 usages

    public Car(String brand, String model, int year, int numberOfSeats) {  no usages
        super(brand, model, year);
        this.numberOfSeats = numberOfSeats;
    }

    public int getNumberOfSeats() {  no usages
        return numberOfSeats;
    }

    @Override  1 usage
    public double calculateRentalCost() {
        return super.calculateRentalCost() + (numberOfSeats * 2); // Biaya tambahan berdasarkan jumlah kursi
    }
}
```

Class Truck

```java
public class Truck extends Vehicle {  no usages
    private double loadCapacity;  3 usages

    public Truck(String brand, String model, int year, double loadCapacity) {  no usages
        super(brand, model, year);
        this.loadCapacity = loadCapacity;
    }

    public double getLoadCapacity() {  no usages
        return loadCapacity;
    }

    @Override  2 usages
    public double calculateRentalCost() {
        return super.calculateRentalCost() + (loadCapacity * 0.5); // Biaya tambahan berdasarkan kapasitas muatan
    }
}
```

Class Main

```
public class Main {
    public static void main(String[] args) {
        Vehicle[] vehicles = {
                new Car( brand: "Toyota",  model: "Corolla",  year: 2020,  numberOfSeats: 5),
                new Truck( brand: "Ford",  model: "F-150",  year: 2018,  loadCapacity: 2.5),
                new Car( brand: "Honda",  model: "Civic",  year: 2019,  numberOfSeats: 4),
                new Truck( brand: "Chevrolet",  model: "Silverado",  year: 2021,  loadCapacity: 3.0)
        };

        for (Vehicle vehicle : vehicles) {
            vehicle.displayInfo();
            if (vehicle instanceof Car) {
                Car car = (Car) vehicle;
                System.out.println("Number of Seats: " + car.getNumberOfSeats());
            } else if (vehicle instanceof Truck) {
                Truck truck = (Truck) vehicle;
                System.out.println("Load Capacity: " + truck.getLoadCapacity() + " tons");
            }
            System.out.println("Rental Cost: $" + vehicle.calculateRentalCost());
            System.out.println();
        }
    }
}
```

Thank You