Nesne Yönelimli Programlama

Polimorfizm (Polymorphism)

Hüseyin Ahmetoğlu

Java'da çok biçimlilik

- **Java'da polimorfizm,** *tek bir eylemi farklı şekillerde* gerçekleştirebileceğimiz bir kavramdır.
- Polimorfizm 2 Yunanca kelimeden türetilmiştir: poli ve morf. "Poli" kelimesi birçok anlamına gelir ve "morf" form anlamına gelir. Yani polimorfizm birçok form anlamına gelir.
- Java'da iki tür polimorfizm vardır: derleme zamanı polimorfizmi ve çalışma zamanı polimorfizmi.
- Java'da metot aşırı yüklenmesi ve metot geçersiz kılma ile polimorfizm yapabiliriz.
- Java'da statik bir metodu aşırı yüklerseniz, derleme zamanı polimorfizmine bir örnektir.

Java'da Çalışma Zamanı Polimorfizmi

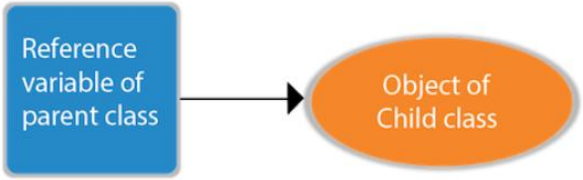
- **Çalışma zamanı polimorfizmi** veya **Dinamik metot dağıtımı** , geçersiz kılınmış bir metoda yapılan çağrının derleme zamanı yerine çalışma zamanında çözüldüğü bir işlemdir.
- Bu işlemde, bir üst sınıfın referans değişkeni üzerinden geçersiz kılınmış bir yöntem çağrılır. Çağrılacak yöntemin belirlenmesi, referans değişkeni tarafından atıfta bulunulan nesneye dayanmaktadır.

Upcasting

- Ana sınıfın başvuru değişkeni, Child sınıfının nesnesine başvuruyorsa, bu upcasting olarak tanımlanır.
- Upcasting için, sınıf tipi veya bir arabirim tipi referans değişkenini kullanabiliriz.

```
class A{}
class B extends A{}
```

```
A a=new B();//upcasting
```




```
interface I{}
class A{}
class B extends A implements I{}
```

Burada B sınıfı ilişkisi şöyle olur:

```
B IS-A A
B IS-A I
B IS-A Object
```

Object, Java'daki tüm sınıfların kök sınıfı olduğundan, B IS-A Object yazabiliriz.

Java Runtime Polimorfizmi Örneği

```
class Bike{
  void run(){System.out.println("running");}
}
class Splendor extends Bike{
  void run(){System.out.println("running safely with 60km");}

  public static void main(String args[]){
    Bike b = new Splendor();//upcasting
    b.run();
  }
}
```

Output:

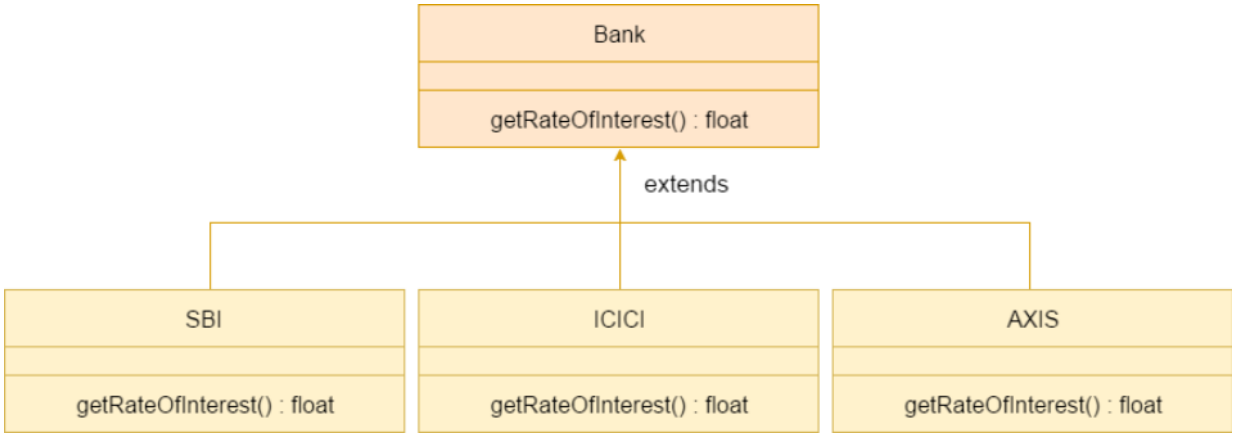
```
running safely with 60km.
```

Java Runtime Polimorfizmi Örneği

- Bu örnekte, iki sınıf Bike ve Splendor yaratıyoruz.
- Splendor sınıfı, Bike sınıfını genişletir ve run () metodunu geçersiz kılar.
- Parent sınıfının başvuru değişkeni tarafından run metodunu çağırıyoruz.
- Alt sınıf nesnesine başvurduğundan ve alt sınıf metodu Ana sınıf metodunu geçersiz kıldığından, alt sınıf metodu çalışma zamanında çağrılır.
- Metod çağırma derleyici değil JVM tarafından belirlendiğinden,bu durum çalışma zamanı polimorfizmi olarak bilinir.

Java Runtime Polymorphism Example: Bank

- Bankanın faiz oranını elde etmek için bir metot barındıran bir sınıf olduğu bir senaryo düşünün. Ancak, faiz oranı bankalara göre değişebilir. Örneğin, SBI, ICICI ve AXIS bankaları% 8,4,% 7,3 ve% 9,7 faiz oranı sağlamaktadır.



Java Runtime Polymorphism Example: Bank

```
class Bank{
float getRateOfInterest(){return 0;}
}
class SBI extends Bank{
float getRateOfInterest(){return 8.4f;}
}
class ICICI extends Bank{
float getRateOfInterest(){return 7.3f;}
}
class AXIS extends Bank{
float getRateOfInterest(){return 9.7f;}
}
class TestPolymorphism{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("SBI Rate of Interest: "+b.getRateOfInterest());
b=new ICICI();
System.out.println("ICICI Rate of Interest: "+b.getRateOfInterest());
b=new AXIS();
System.out.println("AXIS Rate of Interest: "+b.getRateOfInterest());
}
}
```

Output:

```
SBI Rate of Interest: 8.4
ICICI Rate of Interest: 7.3
AXIS Rate of Interest: 9.7
```

Java Runtime Polimorfizmi Örnek: Şekil

```
class Shape{
void draw(){System.out.println("drawing...");}
}
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle...");}
}
class Circle extends Shape{
void draw(){System.out.println("drawing circle...");}
}
class Triangle extends Shape{
void draw(){System.out.println("drawing triangle...");}
}
class TestPolymorphism2{
public static void main(String args[]){
Shape s;
s=new Rectangle();
s.draw();
s=new Circle();
s.draw();
s=new Triangle();
s.draw();
}
}
```

Output:

```
drawing rectangle...
drawing circle...
drawing triangle...
```

Java Runtime Polymorphism Example: Animal

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
}
class Cat extends Animal{
void eat(){System.out.println("eating rat...");}
}
class Lion extends Animal{
void eat(){System.out.println("eating meat...");}
}
class TestPolymorphism3{
public static void main(String[] args){
Animal a;
a=new Dog();
a.eat();
a=new Cat();
a.eat();
a=new Lion();
a.eat();
}}
```

Output:

```
eating bread...
eating rat...
eating meat...
```

Veri Üyesi ile Java Runtime Polimorfizmi

- Aşağıda verilen örnekte, her iki sınıfın da bir veri üyesi hız sınırı vardır. Veri üyesine, alt sınıf nesnesine başvuran Parent sınıfının başvuru değişkeni ile erişiyoruz. Geçersiz kılınmamış veri üyesine eriştiğimiz için, Ana sınıfın veri üyesine her zaman erişecektir.
- **Kural: Çalışma zamanı polimorfizmi veri üyeleri tarafından gerçekleştirilemez.**

```
class Bike{
 int speedlimit=90;
}
class Honda3 extends Bike{
 int speedlimit=150;

 public static void main(String args[]){
  Bike obj=new Honda3();
  System.out.println(obj.speedlimit);//90
}
```

Output:

```
90
```

Çok Düzeyli Kalıtım ile Java Runtime Polimorfizmi

```
class Animal{
void eat(){System.out.println("eating");}
}
class Dog extends Animal{
void eat(){System.out.println("eating fruits");}
}
class BabyDog extends Dog{
void eat(){System.out.println("drinking milk");}
public static void main(String args[]){
Animal a1,a2,a3;
a1=new Animal();
a2=new Dog();
a3=new BabyDog();
a1.eat();
a2.eat();
a3.eat();
}
}
```

Output:

```
eating
eating fruits
drinking Milk
```

Örnek

```
class Animal{
void eat(){System.out.println("animal is eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("dog is eating...");}
}
class BabyDog1 extends Dog{
public static void main(String args[]){
Animal a=new BabyDog1();
a.eat();
}}
```

Output:

```
Dog is eating
```

BabyDog, eat () metodunu geçersiz kılmadığından, Köpek sınıfının eat () metodu çağrılır.

Statik Bağlanma ve Dinamik Bağlanma

- Bir metot çağrısının metot gövdesine bağlanmasına ***binding*** denir.
- İki tür bağlanma vardır
  - Statik Bağlanma (Erken (early binding) Bağlanma olarak da bilinir).
  - Dinamik Bağlanma (Geç (late binding) Bağlanma olarak da bilinir).

Türü Anlama

- **1) değişkenlerin bir türü vardır**

```
int data=30;
```

- **2) Referansların bir türü vardır**

```
class Dog{
 public static void main(String args[]){
  Dog d1;//Here d1 is a type of Dog
 }
}
```

- **3) Nesnelerin bir türü vardır**
  - Bir nesne, belirli bir java sınıfının örneğidir, ancak aynı zamanda üst sınıfının bir örneğidir.
  - Burada d1, Dog sınıfının bir örneğidir, ancak aynı zamanda Animal sınıfının da bir örneğidir.

```
class Animal{}

class Dog extends Animal{
 public static void main(String args[]){
  Dog d1=new Dog();
 }
}
```

statik bağlanma

- Nesnenin türü derleme zamanında (derleyici tarafından) belirlenirse, bu durum statik bağlanma olarak bilinir.
- Bir sınıfta herhangi bir private, final veya static yöntem varsa statik bağlanma vardır.

```
class Dog{
 private void eat(){System.out.println("dog is eating...");}

 public static void main(String args[]){
  Dog d1=new Dog();
  d1.eat();
 }
}
```

Dinamik bağlanma

- Nesnenin türü çalışma zamanında belirlenirse bu, dinamik bağlama olarak bilinir.
- Aşağıdaki örnekte nesne türü derleyici tarafından belirlenemez, çünkü Dog örneği aynı zamanda Animal'in bir örneğidir. Bu nedenle derleyici türünü bilemez, yalnızca taban türünü bilir.

```
class Animal{
 void eat(){System.out.println("animal is eating...");}
}

class Dog extends Animal{
 void eat(){System.out.println("dog is eating...");}

 public static void main(String args[]){
  Animal a=new Dog();
  a.eat();
 }
}
```

```
Output:dog is eating...
```

Java instanceof

- Bir nesnenin belirtilen bir tür (sınıf veya alt sınıf veya arayüz) bir örneği olup olmadığını test etmek için kullanılır.
- instanceof, tür ile tür karşılaştırdığı için tür *karşılaştırma operatörü* olarak da bilinir .
- True veya False döner.
- instanceof operatörünü null değeri olan herhangi bir değişkenle uygularsak, false değerini döndürür.

Örnek

```
class Simple1{
 public static void main(String args[]){
 Simple1 s=new Simple1();
 System.out.println(s instanceof Simple1);//true
 }
}
```

```
Output:true
```

Örnek

- Alt sınıf türünde bir nesne aynı zamanda bir üst sınıf türüdür.

```
class Animal{}
class Dog1 extends Animal{//Dog inherits Animal

 public static void main(String args[]){
 Dog1 d=new Dog1();
 System.out.println(d instanceof Animal);//true
 }
}
```

```
Çıktı: true
```

null örneği

- Null değerine sahip bir değişkenle instanceof operatörünü uygularsak, false değerini döndürür.

```
class Dog2{
 public static void main(String args[]){
  Dog2 d=null;
  System.out.println(d instanceof Dog2);//false
 }
}
```

```
Output:false
```

Java instanceof operatörü ile downcasting

- Alt sınıf türü, Üst sınıf nesnesine başvurduğunda, bu durum **downcasting** olarak bilinir.
- Doğrudan yaparsak derleyici Derleme hatası verir.
- Typecasting ile gerçekleştirirseniz, ClassCastException çalışma zamanında atılır.
- Ancak instanceof operatörünü kullanırsak, indirgeme mümkündür.

```
Dog d=new Animal();//Compilation error
```

```
Dog d=(Dog)new Animal();
//Compiles successfully but ClassCastException is thrown at runtime
```

instanceof ile downcasting

```
class Animal { }

class Dog3 extends Animal {
  static void method(Animal a) {
    if(a instanceof Dog3){
      Dog3 d=(Dog3)a;//downcasting
      System.out.println("ok downcasting performed");
    }
  }

  public static void main (String [] args) {
    Animal a=new Dog3();
    Dog3.method(a);
  }

 }
```

```
Output:ok downcasting performed
```

instanceof kullanmadan downcasting

```
class Animal { }
class Dog4 extends Animal {
  static void method(Animal a) {
      Dog4 d=(Dog4)a;//downcasting
      System.out.println("ok downcasting performed");
  }
   public static void main (String [] args) {
    Animal a=new Dog4();
    Dog4.method(a);
  }
}
```

```
Output:ok downcasting performed
```

```
Animal a=new Animal();
Dog.method(a);
//Now ClassCastException but not in case of instanceof operator
```

instanceof'in gerçek kullanımı

```
interface Printable{}
class A implements Printable{
public void a(){System.out.println("a method");}
}
class B implements Printable{
public void b(){System.out.println("b method");}
}

class Call{
void invoke(Printable p){//upcasting
if(p instanceof A){
A a=(A)p;//Downcasting
a.a();
}
if(p instanceof B){
B b=(B)p;//Downcasting
b.b();
}

}
}//end of Call class
```

```
class Test4{
public static void main(String args[]){
Printable p=new B();
Call c=new Call();
c.invoke(p);
}
}
```

```
Output: b method
```

KAYNAKLAR

- Java Tutorial | Learn Java - javatpoint. (2021, March 21). Retrieved from https://www.javatpoint.com/java-tutorial