Nesne Yönelimli Programlama

Overloading, Overriding, Covariant Dönüş Tipi

Hüseyin Ahmetoğlu

Metot Aşırı Yükleme(Overloading)

- Bir sınıfın aynı ada sahip ancak parametreleri farklı birden fazla metodu varsa, bu duruma **Overloading** denir.
- Benzer işlemler için aynı isme sahip metotlar kullanmak kodun okunabilirliğini artırır.
- Verilen sayıların toplanması gerektiğini varsayalım, ancak iki parametre için (int, int) ve üç parametre için (int, int, int) gibi bir metot yazabilirsiniz. Herhangi bir sayıda bağımsız değişken olabilir.

Metot Aşırı Yükleme(Overloading)

- **Metot aşırı yüklemenin avantajı**
  - Metot aşırı yüklenme *, programın okunabilirliğini artırır* .
- **Metot aşırı yüklemenin farklı yolları**
  - Argüman sayısını değiştirerek
  - Veri türünü değiştirerek
- **Java'da, Yöntem Aşırı Yükleme yalnızca yöntemin dönüş türünü değiştirerek gerçekleştirilemez.**

1) Overloading: argüman sayılarını değiştirme

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

Output:

```
22
33
```

2) Overloading : bağımsız değişkenlerin veri türünü değiştirme

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){return a+b;}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

Output:

```
22
24.9
```

Neden yöntem aşırı yükleme sadece yöntemin dönüş tipini değiştirerek gerçekleştirilemez?

- Cevap: Belirsizlik

```
class Adder{
static int add(int a,int b){return a+b;}
static double add(int a,int b){return a+b;}
}
class TestOverloading3{
public static void main(String[] args){
System.out.println(Adder.add(11,11));//ambiguity
}}
```

System.out.println (Adder.add (11,11)); // Burada, java hangi sum () yönteminin çağrılması gerektiğini nasıl belirleyebilir?

Output:

```
Compile Time Error: method add(int,int) is already defined in class Adder
```

Not: Derleme Zamanı Hatası, Çalışma Zamanı Hatası'ndan daha iyidir. Dolayısıyla, aynı yöntemin aynı parametrelere sahip olduğunu bildirirseniz, java derleyici zamanı hatası oluşturur.

Java main () metodu aşırı yükleyebilir miyiz?

- Aşırı metot yükleyerek bir sınıfta istediğiniz sayıda ana metot kullanabilirsiniz. Ancak JVM, yalnızca string dizisini bağımsız değişken olarak alan main () metodunu çağırır.

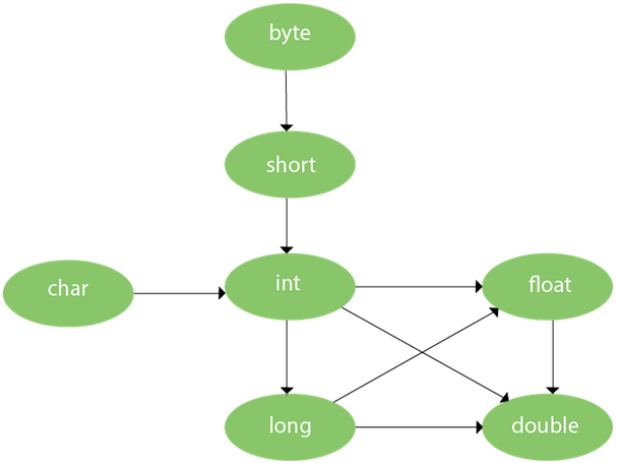
```
class TestOverloading4{
public static void main(String[] args){System.out.println("main with String[]");}
public static void main(String args){System.out.println("main with String");}
public static void main(){System.out.println("main without args");}
}
```

Output:

```
main with String[]
```

Metot Aşırı Yükleme ve Tip Tanıtımı

- Eşleşen bir veri türü bulunmazsa, bir veri türü dolaylı olarak diğerine yükseltilir.
- Şemada gösterildiği gibi, bayt, short, int, long, float veya double yükseltilebilir. short veri tipi int, long, float veya double olarak yükseltilebilir. char veri türü int, long, float veya double vb. olarak yükseltilebilir.



TypePromotion ile Yöntem Aşırı Yüklemesi Örneği

```
class OverloadingCalculation1{
  void sum(int a,long b){System.out.println(a+b);}
  void sum(int a,int b,int c){System.out.println(a+b+c);}

  public static void main(String args[]){
  OverloadingCalculation1 obj=new OverloadingCalculation1();
  obj.sum(20,20);//now second int literal will be promoted to long
  obj.sum(20,20,20);

  }
}
```

```
Output:40
       60
```

Eşleşme bulunursa Tür Tanıtımı ile Metot Aşırı Yüklemesi Örneği

```
class OverloadingCalculation3{
  void sum(int a,long b){System.out.println("a method invoked");}
  void sum(long a,int b){System.out.println("b method invoked");}

  public static void main(String args[]){
  OverloadingCalculation3 obj=new OverloadingCalculation3();
  obj.sum(20,20);//now ambiguity
  }
}
```

```
Output:Compile Time Error
```

Java'da Metot Geçersiz Kılma (Overriding)

- Alt sınıf, üst sınıfta bildirilenle aynı metoda sahipse, bu durum **Overriding** olarak adlandırılır.
- Bir alt sınıf, üst sınıfından herhangi biri tarafından bildirilen metodun özel bir uygulamasını sağlarsa Overriding gerçekleşmiş olur.
- Metot geçersiz kılma, üst sınıf tarafından zaten sağlanan bir metodun spesifik uygulamasını sağlamak için kullanılır.
- Çalışma zamanı polimorfizmi için metot geçersiz kılma kullanılır

Overriding Kuralları

- Metot, üst sınıftakiyle aynı ada sahip olmalıdır
- Metodun, üst sınıftaki parametrelerle aynı olması gerekir.
- Bir IS-A ilişkisi (kalıtım) olmalıdır.

Örnek

```
//Java Program to demonstrate why we need method overriding
//Here, we are calling the method of parent class with child
//class object.
//Creating a parent class
class Vehicle{
  void run(){System.out.println("Vehicle is running");}
}
//Creating a child class
class Bike extends Vehicle{
  public static void main(String args[]){
  //creating an instance of child class
  Bike obj = new Bike();
  //calling the method with child class instance
  obj.run();
  }
}
```

Output:

```
Vehicle is running
```

Yöntem geçersiz kılma örneği

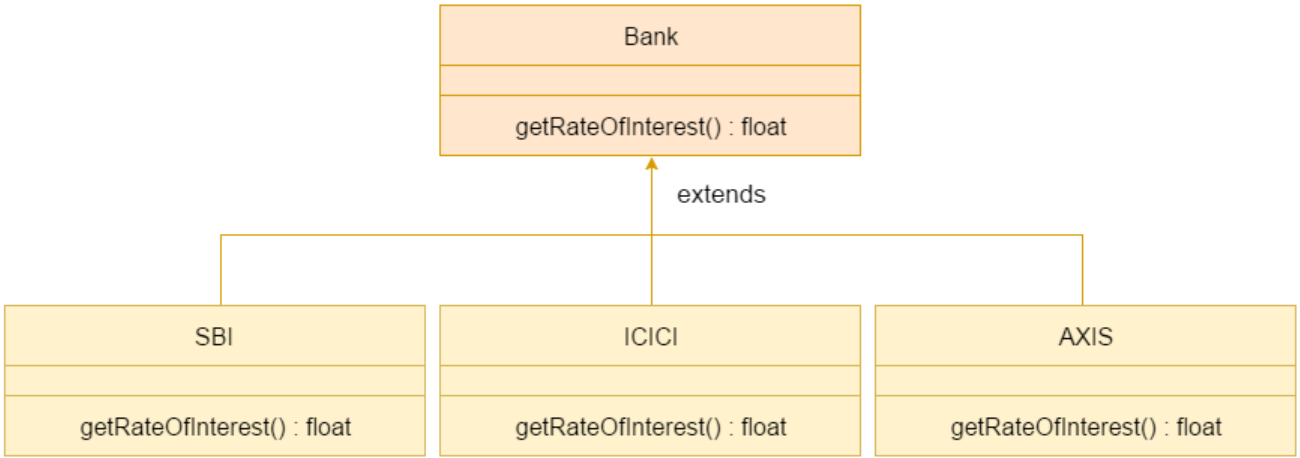
```
//Java Program to illustrate the use of Java Method Overriding
//Creating a parent class.
class Vehicle{
  //defining a method
  void run(){System.out.println("Vehicle is running");}
}
//Creating a child class
class Bike2 extends Vehicle{
  //defining the same method as in the parent class
  void run(){System.out.println("Bike is running safely");}

  public static void main(String args[]){
  Bike2 obj = new Bike2();//creating object
  obj.run();//calling method
  }
}
```

Output:

```
Bike is running safely
```

Gerçek Bir Örnek

- Bankanın faiz oranını elde etmek için her durumda farklı sonuçlar sağlayan bir sınıfın olduğu bir senaryo düşünün. Her faiz oranı bankalara göre değişmektedir. Örneğin, SBI, ICICI ve AXIS bankaları %8, %7 ve %9 faiz sağlayabilir.



```java
//Java Program to demonstrate the real scenario of Java Method Overriding
//where three classes are overriding the method of a parent class.
//Creating a parent class.
class Bank{
int getRateOfInterest(){return 0;}
}
//Creating child classes.
class SBI extends Bank{
int getRateOfInterest(){return 8;}
}

class ICICI extends Bank{
int getRateOfInterest(){return 7;}
}
class AXIS extends Bank{
int getRateOfInterest(){return 9;}
}
```

```java
//Test class to create objects and call the methods
class Test2{
public static void main(String args[]){
SBI s=new SBI();
ICICI i=new ICICI();
AXIS a=new AXIS();
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
}
}
```

```
Output:
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

Sorular-Cevaplar

- Statik bir metodu geçersiz kılabilir miyiz?
  - Hayır, statik bir metot geçersiz kılınamaz.
- Statik metot neden geçersiz kılamıyoruz?
  - Bunun nedeni, statik metodun sınıfa, örnek metodun ise bir nesneye bağlı olmasıdır. Statik sınıf alanına aittir ve bir örnek yığın alanına aittir.
- Java main metodu geçersiz kılabilir miyiz?
  - Hayır, çünkü main statik bir yöntemdir.

Kovaryant Geri Dönüş Tipi

- Kovaryant dönüş türü, dönüş türünün alt sınıfla aynı yönde değişebileceğini belirtir.
- Java5'ten önce, dönüş türünü değiştirerek herhangi bir yöntemi geçersiz kılmak mümkün değildi. Ancak şimdi, Java5'ten beri, alt sınıf, **dönüş türü ilkel olmayan** herhangi bir metodu geçersiz kılarsa yani dönüş türünü alt sınıf türüne değiştirirse, dönüş türünü değiştirerek metodu geçersiz kılmak mümkündür.

Örnek

A sınıfının get () metodunun dönüş türü A, ancak B sınıfının get () metodunun dönüş türü B'dir. Her iki metodun da farklı dönüş türü vardır ve bu bir metot geçersiz kılmadır. Bu durum kovaryant dönüş tipi olarak bilinir.

```
class A{
A get(){return this;}
}

class B1 extends A{
B1 get(){return this;}
void message(){System.out.println("welcome to covariant return type");}

public static void main(String args[]){
new B1().get().message();
}
}
```

```
Output:welcome to covariant return type
```

Kovaryant dönüş türleri nasıl uygulanır?

- Java, dönüş tipi tabanlı aşırı yüklemeye izin vermez, fakat JVM için bu kural esnetilebilir.
- JVM, arama için bir metodun tam imzasını kullanır. Tam imza, argüman türlerine ek olarak dönüş türünü içerdiği anlamına gelir. Bir sınıfın yalnızca dönüş türüne göre değişen iki veya daha fazla metodu olabilir.
- Javac bu durumu kovaryant dönüş tiplerini uygulamak için kullanır.
- ÖZETLE: Kovaryant dönüş, bir yöntem geçersiz kılındığında, geçersiz kılan yöntemin dönüş türünün, geçersiz kılınan yöntemin dönüş türünün bir alt türü olmasına izin verildiği anlamına gelir.

overloading ve overriding arasındaki fark

- Method Overloading

1. Programın *okunabilirliğini artırmak için* yöntem aşırı yüklenmesi kullanılır
2. Yöntem aşırı yüklemesi *sınıf içinde* yapılır .
3. Yöntemin aşırı yüklenmesi durumunda *parametre farklı olmalıdır*
4. Yöntem aşırı yüklenmesi, *derleme zamanı polimorfizminin bir* örneğidir
5. Java'da yöntem aşırı yüklenmesi, yalnızca yöntemin dönüş türü değiştirilerek gerçekleştirilemez. *Dönüş tipi* yöntem aşırı yüklemesinde *aynı veya farklı olabilir* . Ancak parametreyi değiştirmeniz gerekir.

- Method Overriding

1. Yöntem geçersiz kılma, süper sınıfı tarafından zaten sağlanan yöntemin *özel uygulamasını sağlamak için* kullanılır .
2. Yöntem geçersiz kılma, IS-A (kalıtım) ilişkisi *olan iki sınıfta* gerçekleşir .
3. Yöntemin geçersiz kılınması durumunda *parametre aynı olmalıdır*
4. Metot geçersiz kılma, *çalışma zamanı polimorfizminin bir* örneğidir .
5. *Dönüş tipi* , yöntem geçersiz *kılmada aynı veya kovaryant olmalıdır* .

overloading ve overriding arasındaki fark

- Method Overloading

```
class OverloadingExample{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
```

- Method Overriding

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

KAYNAKLAR

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