NESNEYE YÖNELİK PROGRAMLAMA

Overloading ve Overriding

Emir Öztürk

ÇOK BİÇİMLİLİK

- Aşırı yükleme (Overloading)
- Çiğneme* (Overriding)

**http://bilgisayarkavramlari.sadievrenseker.com/2008/11/22/cignemek-overriding/*

Aşırı yükleme, aynı isme sahip birden fazla metot tanımlayabilme yetisidir (Farklı imzalara sahip metotlar). Aşırı yüklenmiş metotlar derlenme anında bağlanır. Çiğneme ise bir üst sınıfın metodunun alt sınıfta yeniden yazılıp kullanılabilmesidir. Çiğnenmiş metotlar ise nesnelerin çağırılma durumuna göre çalışma anında belirlenir.

METOT İMZASI

- Dönüş türü
- Adı
- Aldığı parametreler

Fonksiyon:
```
void imzasiOlanFonksiyon(int a,double b, char c){
}
```

Prototip:
```
void imzasiOlanFonksiyon(int,double,char);
```

İmza:
```
imzasiOlanFonksiyon(int,double,char)
```

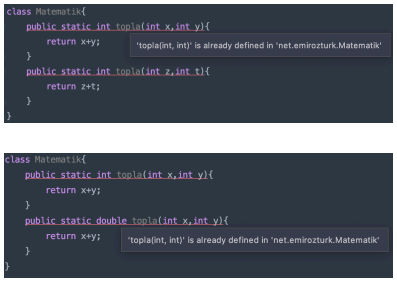
Bir metodun dönüş türü, adı ve aldığı parametrelerin tümü metodun imzasını oluşturur.
Parametrelerin türü ve sayısının değişmesi ile metodun imzası da değişmektedir.

AŞIRI YÜKLEME

- İmza farklılıkları
  - Aynı isim
  - ~~Parametre adı farklılıkları~~
  - ~~Dönüş türü farklılıkları~~
  - Parametre türü farklılıkları
  - Parametre sayısı farklılıkları

Aşırı yükleme (overloading), farklı imzalara sahip metotların aynı isimle tanımlanabilmesidir. Bir sınıf içerisinde aynı isimli metot birden fazla farklı şekilde tanımlanabilir.
Metodun aldığı parametrelerin türü önemlidir. Aldığı parametrelerin türü aynı fakat dönüş türü farklıysa bu geçerli değildir. Derleyici dönüş türüne göre hangi metodun çağırılacağını belirleyemez.
Aşırı yüklemenin geçerli olabilmesi için parametre sayısının ya da parametre türlerinin farklı olması gerekmektedir.

AŞIRI YÜKLEME



Aşırı yüklenen metotların aynı isimli olması gerekmektedir.
Parametre adı farklılıkları veya dönüş türü farklılıkları geçerli değildir. Örneğin iki int alan metotların aldıkları parametre isimleri x,y veya z,t olmak üzere farklı olsa da iki int aldıkları için imzaları aynıdır.
Alınan parametrelerin aynı fakat dönüş türünün farklı olması durumunda bu metot çağırıldığında çözümleme yapılamayacağı için dönüş türünün farklı olması önemli değildir.

AŞIRI YÜKLEME

```java
class Matematik{
    public static int topla(int x,int y){
        return x+y;
    }
    public static int topla(double x,int y){
        return (int)x +y;
    }
    public static int topla(int x,double y){
        return x+(int)y;
    }
    public static int topla(int x,int y,int z){
        return x+y+z;
    }
}
```

Parametre türleri farklı olmalıdır ya da parametre sayısı farklı olmalıdır.
Örneğin iki farklı int değeri alan veya 3 farklı int değeri alan aynı isimli metot tanımlamak mümkündür.
Ayrıca parametrelerin yerleri ile de tanım değişmektedir. İlk parametresi double ikincisi int olan metotla ilk parametresi int ikincisi double olan metot aynı imzaya sahip değillerdir.

GENERICS

```
class Ekran{
    void EkranaGoster(int a){
        System.out.println(a);
    }
    void EkranaGoster(float a){
        System.out.println(a);
    }
    void EkranaGoster(double a){
        System.out.println(a);
    }
    void EkranaGoster(String a){
        System.out.println(a);
    }
}
```

```
class Ekran<T>{
    void EkranaGoster(T a){
        System.out.println(a);
    }
}
```

Bir metodun aldığı parametreler aynı sayıda ve farklı türler için ise overloading yerine generic'ler de kullanılabilir.
Genericlerin kullanılmasının avantajı metot implementasyonunun tekrarlamasını önlemek olsa da T türü her türden parametreyi kabul edeceği için kontrol edilmesi hata almamak adına önemli olacaktır. Genericlerin kullanılmasında türe göre kontroller ile (if) işlem yapılma ihtiyacı duyuluyorsa overloading kullanılması daha uygun olmaktadır.
Genericler overloading alternatifi değildir. Arayüz ve sınıf ile de kullanılabilmektedirler.

GENERICS

```
class Ekran{
    void EkranaGoster(int a){
        System.out.println(a);
    }
    void EkranaGoster(float a){
        System.out.println(a);
    }
    void EkranaGoster(double a){
        System.out.println(a);
    }
    void EkranaGoster(String a){
        System.out.println(a);
    }
}
```

```
class A{

}
public class Main {

    public static void main(String[] args) {
        Ekran t = new Ekran();
        A a = new A();
        t.EkranaGoster( a: 3);
        t.EkranaGoster( a: 3.0);
        t.EkranaGoster( a: a);
    }
}
```

Cannot resolve method 'EkranaGoster(net.emirozturk.A)'

Create method 'EkranaGoster' in 'Ekran' ⌥⏎ More actions... ⌥⏎

Soldaki metotların overload edildiği örnekte yalnızca belirli parametreler kabul edilmektedir.

GENERICS

```
class Ekran<T>{
    void EkranaGoster(T a){
        System.out.println(a);
    }
}
```

```
class A{

}
public class Main {

    public static void main(String[] args) {
        Ekran t = new Ekran();
        A a = new A();
        t.EkranaGoster( a: 3);
        t.EkranaGoster( a: 3.0);
        t.EkranaGoster(a);
    }
}
```

Genericler ile yapılmış metotta ise istenilen türde parametreler verilebilmekte ve çalışma zamanında hata oluşturma ihtimali artmaktadır.

AŞIRI YÜKLENEBİLEN METOTLAR

- Eleman fonksiyonlar (Metotlar)
- Yapıcılar
- main metodu
- ~~Yıkıcı~~
- CPP için operatörler

Örneklerde verildiği üzere eleman fonksiyonlar aşırı yüklenebileceği gibi yapıcılar ve main metotları da aşırı yüklenebilmektedirler. Yapıcı metotlar parametre alabildikleri için farklı parametre türleri veya sayıları ile aşırı yüklenebilirken yıkıcı metotlar parametre alamadıkları ve değer döndürmedikleri için aşırı yüklenmeleri mümkün değildir.

AŞIRI YÜKLEME

```
class Zaman{
    public Zaman(int baslangic){

    }
    public Zaman(int baslangic,int bitis){

    }
    public Zaman(double baslangic, double bitis){

    }
}
```

Yapıcı metotların aşırı yüklenmesi

OPERATÖR AŞIRI YÜKLEME

- Dil üzerinde önceden tanımlı operatörlerin farklı iş yapması için kullanılır.
- Kod okunurluğu artar.
- Kod anlaşılabilirliğinin azalma ihtimali bulunmaktadır.

Örneklerde verildiği üzere eleman fonksiyonlar aşırı yüklenebileceği gibi yapıcılar ve main metotları da aşırı yüklenebilmektedirler. Yapıcı metotlar parametre alabildikleri için farklı parametre türleri veya sayıları ile aşırı yüklenebilirken yıkıcı metotlar parametre alamadıkları ve değer döndürmedikleri için aşırı yüklenmeleri mümkün değildir.

OPERATÖR AŞIRI YÜKLEME

```
class Kisi{
    std::string ad;
    int yas;
public:
    void setYas(int yas){
        this->yas = yas;
    }
};

int main(int argc, const char * argv[]) {
    Kisi k1;
    Kisi k2;
    k1.setYas(30);
    k2.setYas(20);
    std::cout << k1+k2 << std::endl;
    return 0;
}
```

Invalid operands to binary expression ('Kisi' and 'Kisi')

Örneklerde verildiği üzere eleman fonksiyonlar aşırı yüklenebileceği gibi yapıcılar ve main metotları da aşırı yüklenebilmektedirler. Yapıcı metotlar parametre alabildikleri için farklı parametre türleri veya sayıları ile aşırı yüklenebilirken yıkıcı metotlar parametre alamadıkları ve değer döndürmedikleri için aşırı yüklenmeleri mümkün değildir.

OPERATÖR AŞIRI YÜKLEME

```
class Kisi{
    std::string ad;
    int yas;
public:
    void setYas(int yas){
        this->yas = yas;
    }
    int getYas(){
        return this->yas;
    }
    int operator+(Kisi &k){
        return this->yas+k.getYas();
    }
};

int main(int argc, const char * argv[]) {
    Kisi k1;
    Kisi k2;
    k1.setYas(30);
    k2.setYas(20);
    std::cout << k1+k2 << std::endl;
    return 0;
}
```

Örneklerde verildiği üzere eleman fonksiyonlar aşırı yüklenebileceği gibi yapıcılar ve main metotları da aşırı yüklenebilmektedirler. Yapıcı metotlar parametre alabildikleri için farklı parametre türleri veya sayıları ile aşırı yüklenebilirken yıkıcı metotlar parametre alamadıkları ve değer döndürmedikleri için aşırı yüklenmeleri mümkün değildir.

ÇİĞNEME (OVERRIDING)

- Temel Sınıf
- Kalıtım
- Temel Sınıfın metot özelliklerinin değişimi
- Sınıfların üst sınıflara olan bağlantısı

Overriding temel sınıfta bulunan bir metodun alt sınıflarda yeniden tanımlanması ile gerçekleştirilir. Amaç alt sınıflarda metodun ayrıca tanımlanması ile özelleştirme sağlamaktır. Eğer alt sınıf metotları içermiyorsa üst sınıfın metotları kullanılacaktır.

(ÇİĞNEME) OVERRIDING

- Dönüş türünün aynı olması
- Aldığı parametre değişirse overloading
- Final anahtar kelimesi

Override edilen metotların üst sınıf metodu ile aynı imzaya sahip olması gerekmektedir. Eğer farklı imzaya sahip bir metot yazılırsa bu yeni bir metot olmaktadır. Aynı sınıf içerisinde farklı imzaya sahip metotlar ise aşırı yükleme tanımına uyacaktır. Final ile tanımlanmış metotlar ise override edilemez.

(ÇİĞNEME) OVERRIDING

```
class A{
    void metot(){
        System.out.println("A metodu");
    }
}
class B extends A{
    void metot(){
        System.out.println("B metodu");
    }
}
class C extends A{

}
```

```
public class Main {
    public static void main(String[] args) {
        A a = new A();
        a.metot();

        B b = new B();
        b.metot();

        C c = new  C();
        c.metot();
    }
}
```

A metodu
B metodu
A metodu

A sınıfından bir nesne tanımlandığında A metodu çağırılırken B sınıfından bir nesne tanımlandığında B sınıfının içerisindeki metot override olduğu için çağırılacaktır. Eğer C sınıfından bir nesne oluşturulursa bu metod override edilmediği için yine A sınıfındaki metot çağırılacaktır.

(ÇİĞNEME) OVERRIDING

```
class A{
    void metot(){
        System.out.println("A metodu");
    }
}
class B extends A{
    void metot(){
        System.out.println("B metodu");
    }
}
class C extends A{

}
```

```
public class Main {
    public static void fonksiyon(A nesne){
        nesne.metot();
    }
    public static void main(String[] args) {
        A b = new B();
        b.metot();
        fonksiyon(b);
    }
}
```

B metodu
B metodu

Nesnenin türü A olarak belirlenip b olarak örnek alınsa bile durum değişmeyecektir. Çalışma anında nesnenin türü çözümlenip gerekli metot çağırılacaktır.

(ÇİĞNEME) OVERRIDING

```java
class A{
    void metot(){
        System.out.println("A metodu");
    }
}
class B extends A{
    void metot(){
        System.out.println("B metodu");
    }
}
class C extends B{

}
```

```java
public class Main {
    public static void fonksiyon(A nesne){
        nesne.metot();
    }
    public static void main(String[] args) {
        A c = new C();
        c.metot();
        fonksiyon(c);
    }
}
```

B metodu

B metodu

Çok seviyeli kalıtımda da metodun son override edildiği noktada çağırım gerçekleştirilecektir. Örneğin B sınıfından kalıtılmış bir C sınıfı B sınıfından metot() isimli metodu çağıracaktır. C sınıfı nesnesinin A türünde tanımlanması da durumu değiştirmeyecektir.

ÇİĞNENEMEYEN METOTLAR

➤ final anahtar kelimesinin kullanıldığı metotlar

Herhangi bir metotta final anahtar kelimesi kullanıldığında bu metot alt sınıflar tarafından çiğnenememektedir.
