NESNEYE YÖNELİK PROGRAMLAMA

Java ve C++

Emir Öztürk

JAVA - CPP

- C++ dilinde bulunan bazı özellikleri Java desteklememektedir.
  - Önişlemciler
  - Pointerlar
  - Çoklu kalıtım
  - Operatör aşırı yükleme
  - Gizli dönüşümler
  - Goto
  - delete
  - Typedef
  - struct - union
  - template
  - Global değişken ve fonksiyonlar
  - Çok boyutlu diziler

ÖNİŞLEMCİLER

- Derlenmeden önceki aşama
- # karakteri
  - #include
  - #define
  - #if

```
#include <iostream>
#define x 3520


int main(int argc, const char * argv[]) {
    int a = x;
    std::cout << a;
    return 0;
}
```

Önişlemciler C++ diline C dilinden alınmıştır. Kodun derlenmesinden önce çalışırlar. Önişlemcilerin dikkate aldığı direktifler # karakteri ile başlar. Bu direktiflerin en bilinenleri include, define ve if'tir.
Include başka başlık dosyalarını koda dahil etmek amacı ile kullanılmaktadır.
Define ile tanımlanmış ifadeler tüm kodun içerisinde kullanıldığı yerde değiştirilirler.
If ile verilen koşul sağlandığı sürece arada kalan kod bloğu derleyiciye verilir.

POINTERLAR

- Adres gösterimi
- * karakteri
  - Int *
- Java - Referans türleri
- new

```
class sinif{

};
int main(int argc, const char * argv[]) {
    sinif s;
    sinif *refS = new sinif();
```

```
class sinif{

}
public class Main {

    public static void main(String[] args) {
        sinif s = new sinif();
    }
}
```

C ve C++ dillerinde pointerlar adres saklamaktadırlar. C++'ta bellekten dinamik olarak alınan alanlar bir adres değerine atanırlar ve bu adres değeri hangi türden veriyi işaret ediyorsa o türün isminden sonra bir * karakteri ile tanımlanırlar. Java'da da küme alanından alınan bölge dinamik alınsa da referans türündeki değişkenler için açıkça bir karakter ile belirtme durumu yoktur. C++'ta bir alan new ile alınmazsa stack'te, new ile alınırsa heap'te oluşturulurken javada tüm nesneler heap'te oluşturulur ve tüm nesneleri almak için new anahtar kelimesinin kullanılması gerekmektedir.

ÇOKLU KALITIM

- C++ destekler / Java için arayüzler
- Çok üst sınıf bir alt sınıf
- Miras erişimi

```
class A{

};
class B:A{

};
class C:A{

};
class D:B,C{

};
int main(int argc, const char * argv[]) {
    D d;
```

```
interface A{
}
interface B extends A{
}
interface C extends A{
}
class D implements A,B{
}
public class Main {
    public static void main(String[] args) {
        D d = new D();
```

Java'nın aksine C++ çoklu kalıtımı desteklemektedir. Java ile bu problemin üstesinden gelebilmek adına arayüzler kullanılmaktadır. Çoklu kalıtımda bir sınıf birden fazla sınıftan miras alabilmektedir. Ayrıca C++'ta miras alınan sınıfın miras türü (public, protected private) da belirtilebilmektedir. Bu sayede çok seviyeli kalıtımda bir alt sınıfa nelerin aktarılıp aktarılamayacağı da belirtilebilmektedir.

ÇOKLU KALITIM

- Diamond problem
- virtual

```
class A{
public:
    void metot(){

    }
};
class B: public A{
};
class C: public A{
};
class D: B, C{
};
int main(int argc, const char * argv[]) {
    D d;
    d.metot();
}
```

Non-static member 'metot' found in multiple base-class subobjects of type 'A':

```
class A{
public:
    void metot(){

    }
};
class B: public virtual A{
};
class C: public virtual A{
};
class D: public B, public C{
};
int main(int argc, const char * argv[]) {
    D d;
    d.metot();
}
```

Çoklu kalıtımın sebep olduğu bir problem de bir sınıftan miras alan iki farklı sınıfın bir başka sınıf için üst sınıf olma durumudur. Böyle bir durumda hiyerarşinin en altındaki sınıfın hangi sınıf üzerinden metot çağıracağı belirsiz olacaktır. Bu problemin çözümü için C++ virtual kelimesinin kullanılmasına izin vermektedir. Böylece virtual olarak kalıtılan sınıf en üstteki sınıfın (virtual olmayan ilk sınıfın) metoduna erişim sağlayacaktır.

OPERATÖR AŞIRI YÜKLEME

- Operatörlerin farklı kullanımı
  - +, -, *, /
- Atama operatörünün aşırı yüklenmesi
  - =
- Aritmetik, lojik, karşılaştırma, bit tabanlı

```
class Sayi{
private:
    int sayi;
public:
    Sayi(int _sayi){sayi = _sayi;}
    int getSayi(){return sayi;}
    void setSayi(int _sayi){sayi = _sayi;}
    Sayi operator+(Sayi &S){return Sayi(sayi + S.sayi);}
};
int main(int argc, const char * argv[]) {
    Sayi s1(5);
    Sayi s2(3);
    Sayi s3 = s1+s2;
    std::cout<<s3.getSayi();
}
```

Java ve C++ dillerinde operatörler varsayılan (primitif) değişkenler için kullanılabilir durumdadırlar. +, -, *, / gibi operatörler temel tipler üzerinde kullanılabilmektedir. Fakat bu sınıflar (+ için stringler dışında) primitif türler dışında kullanılamamaktadırlar.
C++'ın operatörleri aşırı yüklemesine izin vermesi sayesinde bu operatörler primitif dışındaki türler için de kullanılabilmektedirler. Aşırı yükleme işleminden sonra tanımlanan metotlar sayesinde tüm operatörler belirtilen işleri gerçekleştirecek şekilde kullanıma sunulurlar. Atama operatörü (=) dahil tüm operatörler C++'ta aşırı yüklenebilirler.

GİZLİ DÖNÜŞÜMLER

- Gizli dönüşüm
  - Hata oluşmaz
  - Veri kaybı
- Java - Dönüşüm metotları

```
int main(int argc, const char * argv[]) {
    double d = 234.533;
    int a = d;
    std::cout << a;
}
```

```
public class Main {
    public static void main(String[] args) {
        double d = 234.533;
        int a = d;
    }
}
```

Required type: int
Provided: double
Cast to 'int' ⌥⌘⏎ More actions... ⌥⏎

C++'ta türler veri kaybı kontrolü gerçekleştirmeden diğer türlere dönüştürülebilmektedirler. Örneğin bir double değer int bir değere atanabilmektedir. Fakat değer virgülden sonraki kısmını kaybedecektir. Java'da ise iki türü birbirine dönüştürmek için dönüşüm metotlarının kullanılması gerekmektedir.

GOTO

- Etiketler
- Kodun belirli bir kısmına yönlenme
- Spagetti kod
- Kod kontrolünün kaybı
- Optimizasyon

```
int main(int argc, const char * argv[]) {
    double d = 234.533;
    std::cout << d;
etiket:
    int a = 123;
    std::cout << a;
    goto etiket;
}
```

C++ dilinde, C dilinde olduğu gibi goto ifadesi mevcuttur. Assembler'da olduğu gibi kod bloklarının belirli kısımlarına etiketler atanabilmektedir. Goto deyimi ile bu etiketlere yönlenme işlemi gerçekleştirilebilir. Genellikle goto kullanımında kodun hangi kısmından etikete gelindiğinin takibinin yapılması önemli olduğundan uygunsuz kullanımda kodun takip edilebilirliği zorlaşmaktadır (spaghetti kod). Ayrıca kodun bir bölümüne iki farklı yerden gelindiğinde sonuç işleminde göre tekrar goto ile başka yere yönlendirme işlemi gerçekleştirilmesi gerekmekte ve kodun okunurluğu oldukça azalmaktadır. Buna ragmen bazı işlemlerin veya kontrollerin tekrarlanmasına gerek kalmadan kodun başka bir

bölümüne yönlenme gibi performans gerektiren durumlarda hala kullanılabilmektedir. Örneğin iç içe döngülerden en içtekinin bir koşulu sağlaması durumunda tüm döngülerden çıkılması istendiğinde goto kullanılmadığı takdirde her döngü için bir kontrol (if) ve break ifadesi kullanılması gerekmektedir.

C++'ta da Java'da olduğu gibi dinamik olarak bellek alanı tahsisi mümkündür. Dinamik olarak alınan bu alanlar ihtiyaç duyulmadığı takdirde iade edilmelidirler. Aksi takdirde kullanılmayan alanların bellekte kapladığı gereksiz alan kaynak tüketimine sebep olacaktır (bkz. memory leak). C dilinde malloc veya calloc fonksiyonları ile alınan dinamik alanlar free fonksiyonu ile iade edilirken C++ için new ile alınan alanlar delete fonksiyonu ile iade edilmektedirler. Java dilinde C++'ın aksine bir iade işlemi gerçekleştirilmesi gerekmemektedir. Finalize metodu ile alanların iade edileceği durumda ne yapılması gerektiği tanımlanabilmektedir fakat iade işleminin gerçekleştirileceği zaman çöp toplayıcıya bağlıdır. Çöp toplayıcı belirli zamanlarda silinecek alanları kullanıcıdan bağımsız olarak belleğe iade eder. Böylece gereksiz kaynak kullanımının önüne geçilmiş olur.

TYPEDEF

- Kullanıcı tanımlı tür
- İsim değişikliği
- Java - primitif / sınıf
-

```
typedef Sayi * sayiPointer;
int main(int argc, const char * argv[]) {
    sayiPointer s = new Sayi(234);

    delete s;
}
```

Typedef anahtar kelimesi ile C++ dilinde belirli bir ifade yeniden adlandırılabilmektedir. Örneğin on elemanlı bir dizi için Onluk isminde bir yeniden adlandırma yapıp daha sonra bu Onluk ismi ile değişkenler tanımlanabilmektedir.
Java için ise türler ya primitif ya da sınıf olarak tanımlanmaktadır. Tür tanımı bulunmamaktadır.

STRUCT - UNION

- struct
  - Kullanıcı tanımlı yapılar
  - Stack içerisinde
  - Alanlarına erişime açık
- Union
  - Birden fazla değişken aynı alan

```
struct Sayi{
    int sayi;
};
```

```
class Sayi{
public:
    int sayi;
};
```

C++ dilinde C dilinde olduğu gibi struct'lar bulunmaktadır. Structlar ile kullanıcı tanımlı yapılar oluşturulabilmektedir. Dilde bulunan diğer yapıların bir araya getirilmesiyle tanımlanmaktadırlar. Struct değişkenleri stack içerisinde saklanırlar ve tüm alanları public olan ve sadece özellikler içeren bir sınıfa benzetilebilirler. Union'da ise bir alanı ortak olarak birden fazla değişken paylaşmaktadır. Bu iki yapı da Java'da kullanılamamaktadır.

TEMPLATE

- Birden fazla veri tipi için bir sınıf
  - STL
- Derleme anında
- Farklı veri türleri için farklı örneklerin alınması

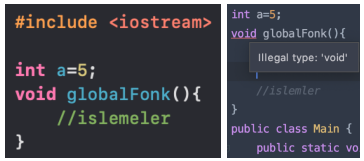
```
template <class T>
T BuyukOlan(T n1, T n2)
{
    return (n1 > n2) ? n1 : n2;
}

int main(int argc, const char * argv[]) {
    BuyukOlan(3, 5);
    BuyukOlan(1.3, 7.2);
    BuyukOlan("a", "b");
}
```

C++'ta template yapısı ile bir sınıf tüm veri türleri için yazılabilmektedir. Bu sayede bir sınıf tanımı verilen bir veri türü için kullanılabilmektedir. STL kütüphanesi içerisinde sıklıkla kullanılmaktadır. Derleme anında kullanılan değişken tipine göre sınıf hazırlanmaktadır. Aynı sınıf farklı veri türleri için farklı örneklerde derlenip hazırlanabilmektedir. Java dilinde template'e benzer en yakın yapı generic'lerdir.

GLOBAL DEĞİŞKEN VE FONKSİYONLAR

- Global değişken ve fonksiyonlar C++ dilinde bulunmaktadır.
- Java dilinde her değişken ve fonksiyon bir sınıfın parçası olmak zorundadır.



```cpp
#include <iostream>

int a=5;
void globalFonk(){
    //islemeler
}
```

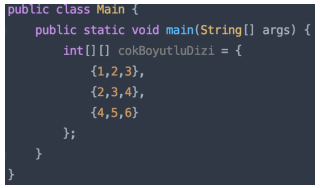


```
int a=5;
void globalFonk(){
    Illegal type: 'void'
    //islemler
}
public class Main {
    public static vo
```

C++ dilinde sınıfların dışında global olarak her yerde değişkenler ve metotlar tanımlanabilmektedirler. Java için ise değişken veya metotlar mutlaka bir sınıfın içerisinde olmak zorundadırlar.

JAVA - CPP

- Java'da çok boyutlu dizi bulunmamaktadır.
  - Dizilerin dizisi
- C++'ta çok boyutlu dizi desteği bulunmakta ve bellekte tek boyutta saklanmaktadır.



```java
public class Main {
    public static void main(String[] args) {
        int[][] cokBoyutluDizi = {
            {1,2,3},
            {2,3,4},
            {4,5,6}
        };
    }
}
```

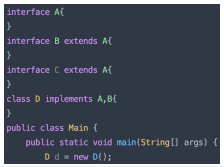
Java'da çok boyutlu diziler, dizilerin dizisi şeklinde tanımlanmaktadır. Yani [10,10]'luk bir matris tanımlanmak istendiğinde aslında 10 elemanlı 10 adet dizi olarak tanımlama işlemi gerçekleştirilmektedir. C++ dilinde ise iki boyutlu diziler tanımlanabilmekte ve bu iki boyutlu dizi bellekte satır sıralı olarak saklanmaktadır.

JAVA - CPP

- Bulunmayan özelliklerin aksine Java, C++ ile desteklenmeyen bazı özelliklere sahiptir
  - Arayüzler
  - Çöp toplayıcısı (GC)
  - ~~Dil tanımlı çoklu işlem - threading~~ (C++ 11)
  - Dönüşüm kontrolü
  - Paketler
  - Sanal makine (JVM) ve bytecode

ARAYÜZLER

- Interface
- Birden fazla interface kullanımı
- C++
  - Class
  - Virtual



```
interface A{
}
interface B extends A{
}
interface C extends A{
}
class D implements A,B{
}
public class Main {
    public static void main(String[] args) {
        D d = new D();
```

Java'da çoklu kalıtım desteklenmemektedir. Bu yüzden javada arayüzler kullanılabilmektedirler. Bir sınıf birden fazla arayüzü implement edebilir. Bu sayede dolaylı olarak çoklu kalıtım sağlanmış olur. Arayüzler herhangi bir implementasyon içermezler. Sadece arayüzü implement eden bir sınıfın hangi özellik ve metotlara sahip olması gerektiğini tanımlarlar. C++'ta interface anahtar kelimesi bulunmaz. Bunun yerine virtual olarak tanımlanmış sınıflar kullanılabilmektedir.

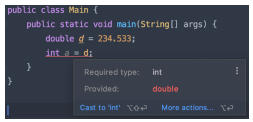
ÇÖP TOPLAYICISI (GC)

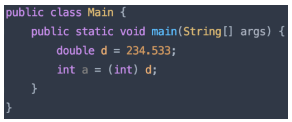
- Garbage Collector
- Heap
- Nesiller
  - Eden
  - Survivor
  - Tenured
  - Permanent
- C++
  - Delete

GC (Garbage collector), bellekten alınan nesnelerin iadesi görevini üstlenir. GC, heap alanına bakıp kullanılmayan nesnelerin silinmesi prensibine göre çalışır.
Java'da heap nesillere bölünmüştür
Eden space: İlk oluşturulan nesneler buraya yerleştirilir. Daha sonra burası dolduğunda minor GC çalışır ve kullanılmayanları temizler.
Survivor space: Eden'da silinmeyen nesneler buraya yerleşir.
Tenured space: GC'nin survivor space üzerinde belirli sayıda tekrarda gezmesinden sonra (threshold) hala silinmeyen nesneler buraya yerleştirilir. Burası dolduğunda major garbage collection çalışır
Permanent generation: Vm ile ilgili tüm kalıcı nesneler burada saklanır.
C++ dilinde bir çöp toplayıcı bulunmaz. Bellekten dinamik olarak alınan yerler delete anahtar kelimesi ile iade edilmelidir.

DÖNÜŞÜM KONTROLÜ

- Primitif türler
  - Otomatik dönüşüm
- Dönüşüm gerektiren türler
  - Örn. Int -> bool

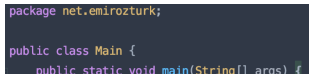




C++ temel türler arasında otomatik dönüşüm gerçekleştirmektedir. Ayrıca tanımlandığı takdirde kullanıcı tanımlı tipler de birbirlerine dönüştürülebilmektedirler. Java'da ise temel tipler arasındaki dönüşümler gerçekleştirilebilir fakat bunun dışındaki dönüşümler açıkça yapılmalıdır. Örneğin C++ dilinde if içerisinde bir atama işlemi ya da bir int değer yazılabilmektedir (if(i=5) veya if(3)). Java'da ise bu kod int'i bool'a dönüştüremeyeceği için hata verecektir. Bu dönüşümün yapılmaması hataların engellenmesi için bir avantaj sağlamaktadır.

PAKETLER

- Sınıfları bir arada tutmak
- Sınıf isimlerinin ayrımı
- Hiyerarşik
- import



Paketler sınıfların bir araya getirilip organize edilmesi için uygun bir ortam sunmaktadır.
Bir paketin içerisinde diğer paketlerle oluşacak bir isim çakışmasını düşünmeden sınıf tanımı yapılabilmektedir.
Paketler hiyerarşik bir yapıda kullanılabilirler ve bir paketin kullanılabilmesi için kullanılacağı sınıfın içerisine dahil edilmelidirler.
Bir paketin sisteme dahil edilmesi için import anahtar kelimesi kullanılmaktadır.
C++'ta paketler bulunmamaktadır.

JVM

- Sanal makine
- javac.exe
- bytecode
- Multiplatform

Java ile yazılan bir program JVM sayesinde bütün platformlarda çalıştırılabilmektedir. Böylece tek bir geliştirme dili / ortamı ile tüm platformlara yazılım geliştirmek mümkün olmaktadır.
Java derleyicisi, java dilini okur ve JVM'de çalıştırmak üzere bytecode'a çevirir. .JVM, Java derleyicisi tarafından derlenip bytecode'a dönüştürülmüş dosyayı bilgisayar üzerinde çalıştıran sanal makinedir.

ERİŞİM BELİRLEYİCİLERİ

- Java
  - private, public, protected, default
- C++
  - private, public, protected

Java'da dört farklı erişim belirleyicisi bulunmaktadır.
Private anahtar kelimesi ile yalnızca sınıfın içerisinden erişim mümkündür.
Hiç bir anahtar kelime belirtilmediğinde (default) sadece paket içerisinden erişim sağlanabilmektedir.
Protected ile default'tan farklı olarak başka paketlerde sınıftan türetilmiş alt sınıflardan da erişim mümkündür.
Public ise her yerden erişime açıktır.
C++ içerisinde ise hiç bir anahtar kelime kullanılmadığında özellikler private olur.
Private ve public java'da olduğu gibi kullanılmaktadır. Protected için ise paket sistemi olmadığı için yalnızca alt sınıfların erişimi sağlanmaktadır.