Java ile Programlama

IDE

- İstediğiniz IDE'yi kullanmakta özgürsünüz. Ancak derste biz Eclipse IDE kullanacağız.
- Bilindik IDE'lerin hepsi güçlüdür ve işinizi görecektir.

İlk Program

```
class Ornek {
    public static void main(String[] args) {
        System.out.println("Java!");
    }
}
```

- Java programlama dilinde programa verdiğiniz **dosya_adi** önemlidir. Örneğin şimdi gidip, adı `Ornek.java` olmayan bir programın içine bunu kopyalarsanız çalışmaz. Sınıf adı ile dosya adı aynı olmalıdır.
- Sınıf adları büyük harfle başlar.

Yorum

```
// bu bir yorum satırıdır
/*
bu birden fazla yorum yapmaya izin verir
*/
```

Main, Public, Private

- Aynı C ya da C++ dilinde yapıldığı gibi, burada da bir Main fonksiyonu var. `public static void main` diyerek aslında bu Main fonksiyonunu çağırıyoruz.
- **public** ve **private**, *access modifier* olarak geçer. Bunlar, o sınıfa nerelerden erişim sağlanacağını kontrol ederler.
    - Eğer public olarak tanımlarsak, o zaman o sınıfın dışındaki sınıflar da erişebilir.
    - Eğer private olarak tanımlarsak, ancak o sınıf içindeki metodlar erişebilir.

Static, Void

- Main için **static** diyoruz, çünkü o sınıf içinde hiç obje yaratmadan Main'i çağırmak gerekiyor.
- Static buna olanak sağlıyor.
- Void, return ile alakalı. Return nasıl olacaksa öyle yazarız.
  - Örneğin int dönecekse, `int` yazacağız.

String[] args

- Bu aslında alınan parametrelerle ilgili. Args isminde bir String array var ve input olarak alınabilir.
- Şu an almıyor, ancak bu şekilde declare ediliyor method.

Örnek

```
class Ornek2 {
    public static void main(String[] args) {
        int myVar1;

        myVar1 = 100;
        System.out.println(myVar1);
        System.out.println("myVar1 değişkeni:" + myVar1);

        int myVar2 = 200;
        System.out.println(myVar2 / 2);

    }
}
```

- Variables are very similar to C++.
- `tip değişken_adı` şeklinde tanımlanıyor. `int myVar`, `char myChar`.

Veri tipleri

- Java *strongly typed* language. Yani, Python gibi dinamik değil.
- Java'nın içerdiği veri tipleri ikiye ayrılıyor: nesne tabanlı olanlar ve olmayanlar.
  - Nesne tabanlı olanlar sınıflarla tanımlanıyor.
- Primitive Types (İlkel tipler)
  - boolean, byte, char, double, float, int, long, short
- Java'daki tüm diğer veri tipleri bu yukarıdakiler kullanılarak oluşturuluyor.

Primitive Tipler

- Bu tiplerin hepsi bir aralığı tanımlıyor.
- Tam sayılar (**Integer**) için şu tipleri kullanıyoruz:

Tip	Boyut	Aralık
byte	8	-128 - 127
short	16	-32768 - 32767
int	32	-2,147,483,648 - 2,147,483,647
long	64	-9,223,372,036,854,775,808 - 9,223,372,036,854,775,807

Tipler

- En yaygın kullanılan tip **int**. Ancak eğer bu belirtilen aralıktan daha büyük sayılar kullanılacaksa o zaman **long** kullanmak gerekir.
- Floa için ise **float** ve **double** var. Float virgülden sonra 1 basamak, double ise 2 basamak kullanılacaksa kullanılır.
    - Float 32-bit, double ise 64 bittir.
- **Double** daha çok kullanılmaktadır.
    - İhtiyacımız olduğunda onu kullanacağız.

Karakterler

- Java'da **char** tipi mevcuttur.

```
char ch;
ch = 'X';
```

- Dikkat etmemiz gereken önemli şeylerden biri **char** oluştururken `'` kullanırız, `"` değil.
- Char 16-bit uzunluğundadır ve yalnızca **tek** karakter tutabilir.
- Char karakteri üzerinde değişik işlemler yapılabilir:

```
char ch;
ch = 'X';
ch++;
ch = 90;
```

Unicode

- Java, evrensel olarak kullanılması amaçlandığı için unicode kullanır.
- ASCII karakterler 8bit ile tanımlanabilir ancak bunlar birkaç alfabe dışında etkisizdir.
  - O yüzden bazı dillerde Türkçe karakterler sorun çıkarır. Java'da çıkarmaz.

Boolean

- Doğru / Yanlış tutan bir değerdir.
- `boolean b = true`, `boolean c = false` şeklinde kullanılır.

String

- String diğer tiplerden daha farklı, aslında tam olarak bir tip değil, bir sınıf.
- `String a = new String()` şeklinde de kullanılabilse de, çok sık kullanılan bir şey olduğu için Java String'e torpil geçmiştir.
- `String a = "selam";` çalışır.
- String içine `\n` gibi şeyleri yine diğer dillerde olduğu gibi yerleştirebiliriz.

Scope

- Bir değişkeni nerede tanımladığımız çok önemlidir.

```
public class Main {

                public static void main(String[] args) {
            int x = 10;
            if(x == 10) {
                int y=5;
                System.out.println("x: " + x; + " y:" + y);
            }
            y = 4;
            System.out.println(y);
        }
}
```

- Örneğin bu kod çalışmayacaktır, çünkü `main` fonksiyonu içinde `y` tanımlanmadı. O `if` içinde tanımlandı ve orada çağrılabilir.

Operatörler

- +: toplama
- -: çıkarma
- *: çarpma
- /: bölme
- %: mod
- ++: Bir ekleme
- --: Bir eksiltme

Örnek

```
public static void main(String[] args) {
        int bolum;
        int kalan;

        bolum = 10 / 3;
        kalan = 10 % 3;

        System.out.println("10 ile 3 ün bölümünden \n");
        System.out.println("Bölum: " + bolum);
        System.out.println("Kalan: " + kalan);
        }
```

Örnek a++, ++a

- Bu iki gösterim de sıklıkla uygulanır ancak farkı çok fazla bilinmez.
- Bir örnek ile görelim.

```
public static void main(String[] args) {
    int a = 0;
    int b = 10;

    System.out.println("a: " + a);
    System.out.println("b: " + b);

    a = b++;

    System.out.println("a: " + a); //10
    System.out.println("b: " + b); //11
}
```

Örnek a++, ++a

```
public static void main(String[] args) {
    int a = 0;
    int b = 10;

    System.out.println("a: " + a);
    System.out.println("b: " + b);

    a = ++b;

    System.out.println("a: " + a); //11
    System.out.println("b: " + b); //11
}
```

İlişkisel operasyonlar

- `==` : eşit
- `!=` : eşit değil
- `>` : büyük
- `<` : küçük
- `>=` : büyük eşittir
- `<=` : küçük eşittir

Mantıksal operasyonlar

- `&` : ve (and)
- `|` : veya (or)
- `^` : xor
- `||` : short-circuit (conditional) OR
- `&&` : short-circuit (conditional) AND
- `!` : not

Mantıksal operasyonlar

- If içerisinde `&, | , &&, ||` kullanabiliyoruz. Bunların farkı şu şekilde:
  - Eğer kısa-devre OR ya da AND kullanırsak, if içerisindeki iki duruma birden bakmak yerine, öncelikle ilk olana bakıyor.
  - Eğer o durumu sağlamıyorsa, ikinciye direkt bakmıyor bile.
- O halde `& ve |` a gerek yok gibi duruyor ancak yine de bunlar kullanışlı olabilir.

Tip değişimi

- Diğer dillerde olduğu gibi bazı tipler birbirine dönüştürülebilir.

```
double a = 5;
int b = (int) a;
```

- Soru:

```
double c = 10 / 3;
System.out.println(c);
```

- Sonuç ne olur?

Kullanıcıdan veri alma

- Bunun için farklı kaynaklar farklı şeyler gösterse de, standard `Scanner` sınıfını kullanmaktır.
- Bunun için onu import etmek gerekiyor. O yüzden en tepeye `import java.util.Scanner;` demek gerek.

```
public static void main(String[] args) {
    Scanner girdi = new Scanner(System.in);
        System.out.println("Adın nedir?");
        String kullaniciAdi = girdi.nextLine();
        System.out.println("Kullanıcı: " + kullaniciAdi);
        }
```

- Burada nextLine() dedik çünkü String istiyoruz. Ne alacaksak ona göre bir metod çağırmalıyız. `nextInt()` .

IF

- Diğer dillerdekiyle aynı şekilde kullanılıyor.

```
Scanner girdi = new Scanner(System.in);
System.out.println("Sayı gir\n");
int sayi = girdi.nextInt();
if (sayi == 4) {
    System.out.println("dört");
}
else if(sayi == 5) {
    System.out.println("beş");
}
else {
    System.out.println("dört ya da beş değil");
}
```

Switch-Case

- Yine kullanımı benzer şekilde. Ancak bu basit hali, geliştirilmiş bir versiyonu daha var Java içinde.

```
for(int i=0;i<5;i++) {
    switch(i) {
        case 0:
        System.out.println("i 0");
        break;
        case 1:
        System.out.println("i 1");
        break;
        default:
        System.out.println("i 0 ya da 1 değil");
    }
}
```

Örnek uygulama

- Bu aşamaya kadar kullanıcıdan veri almayı ve if komutunu öğrendik.
- O halde çok basit bir kullanıcı adı ve şifresini kontrol eden uygulama yapalım.

```
Scanner sc = new Scanner(System.in);
System.out.println("Kullanici adi girin\n");
String kullanici = sc.nextLine();

//db'den cektik diyelim
String dbKullanici = "einstein";
String dbSifre = "eins123";

if(kullanici.equals(dbKullanici) && sifre.equals(dbSifre)) {
    System.out.println("Dogru");
}
else {
    System.out.println("Yanlis");
}
```

For döngüsü

- C ve C++ gibi kullanılıyor.

```
for(int i=0;i<10;i++) {
    System.out.println(i);
}
```

```
for(int i=0;i<10;i+=2) {
    System.out.println(i);
}
```

For döngüsü

```
for(int i=0;i<5;i++){
    for(int j=10; j>5; j--) {
        System.out.println("i ve j:" + i + j);
    }
}
```

```
int i,j;

for(i=0, j=10; i < j; i++, j--) {
    System.out.println("i ve j:" + i + " " + j);
}
```

For döngüsü

```
for(int i=0;i<10;) {
    System.out.println("Case #:" + i);
    i++;
}
```

- Bu çalışır mı?

Sonsuz döngü

```
for(;;) {
    //
}
```

Scope

- Daha önce verdiğimiz Scope ile alakalı örneğe For döngüsü içinde bakabiliriz.
- For içerisinde döndüreceğimiz sayıyı da tanımlamak gerekir. Ancak bunu **for** içinde yapabiliriz.

```
for(int i=0;i<5;i++){
    \\
}
```

- Ancak şimdi `i`, for dışında anlamsız hale gelecektir.

While

```
int a = 4;
while(a < 10) {
    System.out.println(a);
    a++;
}
char ch = 'a';
while(ch <= 'z') {
    System.out.print(ch);
    ch++;
}
```

Do While

- Burada while içindeki şey eğer doğruysa, do içindeki çalışsın isteriz.
- Ancak burada while içindeki doğru olmasa da do en az bir kez çalışır.

```
int a = 5;
do {
    System.out.println(a);
    a++;
} while (a < 10);
```

Break

- Döngülerin içinden çıkmak için kullanılır.

```
for(int i=0;i<5;i++) {
    if(i == 3) {
        break;
    }
    System.out.println(i);
}
```

Continue

- Bir loop içerisinde bir sonraki iterasyona geçmek istersek `continue` kullanırız.

```
for(int i=0;i<5;i++) {
    if(i<2) {
        continue;
    }
    System.out.println("0 ya da 1 değilim ben:" + i + "");
}
```