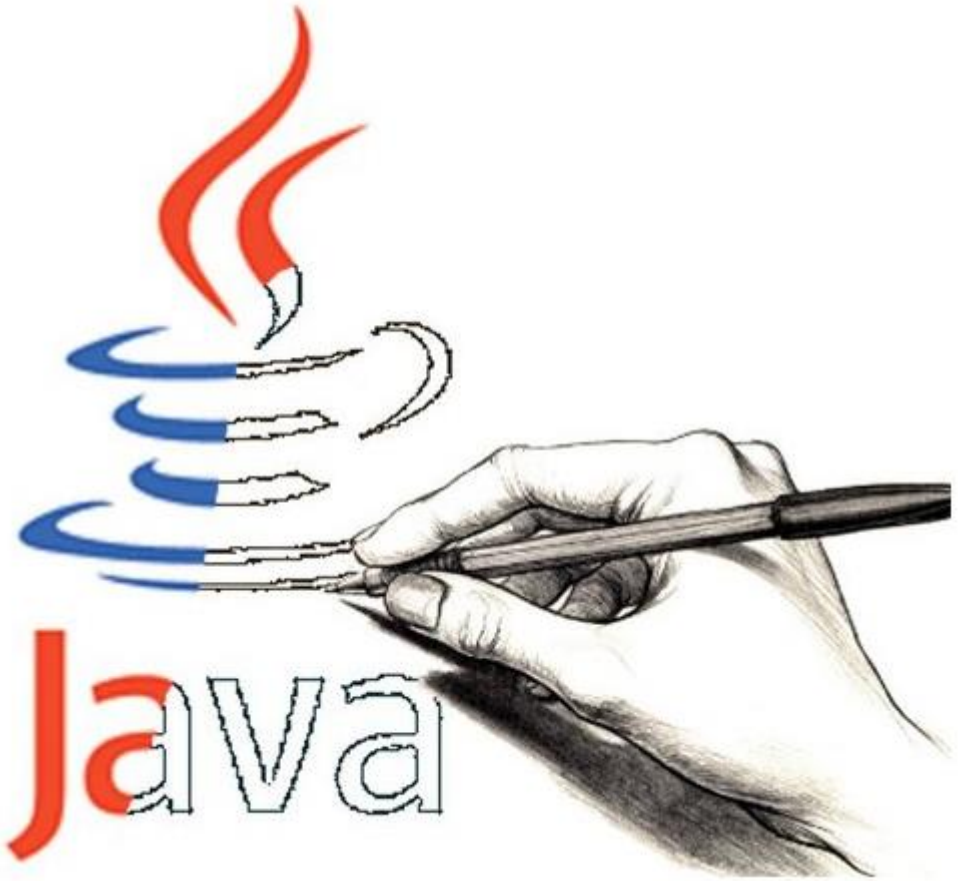
YMT 112
Algoritma ve Programlama II

Fatih Özkaynak

İçerik

- Değişkenler
- Operatörler
- Kontrol Yapıları
- Döngüler
- Metotlar - Fonksiyonlar



Metotlar

- Her proje metotlardan meydana gelmektedir.
- Bir görevi yerine getirmek için grup kod bloğunun bir araya gelmesidir.
- Programı küçük parçalara bölmek amacıyla metotları kullanırız.
- Metotlar projelerimizin daha performanslı çalışmasını sağlar.
  - Projeyi hatalardan arındırma,
  - projenin zamanla gelişse bile kolayca değişiklikler yapılmasını
  - Anlaşabilir kod yazımını arttırır.
- İsimlendirmesinde fiil içermesine dikkat edilmelidir.
- OOP (nesne yönelimli programlarda) programlar birbirleri ile etkileşimde olmalıdır.

Metot Nedir ?

- Bir işlemi bir kere yazıp, program içerisinde istediğimiz yerde ve sıklıkta kullandığımız yapılardır.
- Fonksiyon olarak da adlandırılırlar ve diğer dillerde genellikle bu tabirle kullanılırlar.
- Java'da "methods"(metotlar) olarak isimlendirilir ve bu şekilde bilinirler.
- Metotlar oluşturulma şekillerine göre ikiye ayrılırlar :
  - Önceden Tanımlanmış Metotlar :
    - Java kütüphanelerinde bulunan hazır metotlardır. Örneğin pow(x,y) metodu. Math sınıfına ait olan bu metot, x ve y olarak iki değer alır ve bir değer döndürür. Görevi ilk değerin(x), ikinci değer(y) üssünü almaktır.
  - Kullanıcı Tanımlı Metotlar: Bizim yazdığımız ve oluşturduğumuz metotlardır.

Metotların Deklarasyonu

- erisimBelirteci durumBelirteci donusTipi metotAdi(parametreList) {
- ifadeler;
- }
- erisimBelirteci, bu metodun hangi alanlar tarafından kapsandığını ve ulaşılabilme sınırını belirtir.
- durumBelirteci, metotun durumunu belirtmemize yarayan belirteçtir.
- donusTipi, geri dönüş tipidir. Döndüreceği veri tipi buraya yazılır.
- metotAdi, metodumuzun adıdır.
- parametreList, parametrelerin yazılacağı alandır.
- Süslü parantezler
- Return

```java
package Dersler;

public class ders06 {

    //örnek metodumuz
    public void ornekMetot() {
        System.out.println("Bu örnek bir metottur.");
    }

    public static void main(String[] args) {
        ders06 nesne = new ders06();    //Nesne oluşturma

        nesne.ornekMetot();             //Metota ulaşma

        System.out.println("-- Programın Sonu --");
    }
}
```

Metotlarda Parametre Kavramı

- Metotlar parametre alabilirler ve bu değerleri metot içerisinde döndürebiliriz.
- Parametresiz Metotlar
  - Değer döndürmeyen ve parametreleri olmayan metotlardır.

```java
package Dersler;

public class ders06 {

    public int topla(int a, int b) {
        return a+b;
    }

    public static void main(String[] args) {
        ders06 nesne = new ders06();

        int a = 10;
        a = nesne.topla(15,30);

        System.out.println("a'nın değeri : "+a);
    }
}
```

```
public class Program {

    static int totalLength(String a, String b) {
        // Add up lengths of two strings.
        return a.length() + b.length();
    }

    static int averageLength(String a, String b) {
        // Divide total length by 2.
        return totalLength(a, b) / 2;
    }

    public static void main(String[] args) {

        // Call methods.
        int total = totalLength("Golden", "Bowl");
        int average = averageLength("Golden", "Bowl");

        System.out.println(total);
        System.out.println(average);
    }
}
```

Ana Metodumuz : main()

- Java’da program, ana metot olan main’den başlar.

- Metot Adı -> main
- Erişim Belirteci -> public
- Durum Belirteci -> static
- Dönüş Tipi -> void
- Parametre -> String dizisi

Diğer Özel Metotlar

- Yapılandırıcı Metodlar – Constructors
  - Sınıf ismi ile aynı ismi taşıyan parametreli ya da parametresiz metotlardır.
  - Yapılandırıcı metotların dönüş tipi yoktur.
  - Yapılandırıcı metot kullanmak zorunlu değildir.
  - Derleyici default olarak yapılandırıcı olduğunu varsayar.

- Method Overloading – Metot Aşırı Yükleme
  - Metotların genel olarak isimleri farklı olmalıdır.
  - Uygulama karışıklığı olmasın diye aynı isimde bir metot daha oluşturabiliriz.
  - Metot isim-işlev benzerliği de işe karışınca aynı işlevi yapacak metotları yönetmek zor olacaktır.
  - Böyle durumlarda overloading gerçekleşir.
  - Metotları isimleri aynı olmalı ancak parametre tipleri ya da sayıları farklı olmalıdır.
  - Yani isimleri aynı dönüş tipleri farklı olan metotlar oluşturarak çalışmaya zorlarız.

```java
package Dersler;

public class ders06 {
    public void kutu(int en, int boy) {
        System.out.println("En : "+en);
        System.out.println("Boy : "+boy);
        System.out.println("Alan : "+(en*boy));
    }

    public void kutu(int en, int boy, int kalinlik) {
        System.out.println("En : "+en);
        System.out.println("Boy : "+en);
        System.out.println("Kalınlık : "+boy);
        System.out.println("Hacim : "+(en*boy*kalinlik));
    }

    public static void main(String[] args) {
        ders06 nesne = new ders06();

        nesne.kutu(5, 10);
        System.out.println("----------");
        nesne.kutu(5, 10, 5);
    }
}
```

```
public class Program {

    public static void action(int value) {
        System.out.println("Int = " + Integer.toString(value));
    }

    public static void action(String value) {
        System.out.println("Length = " + value.length());
    }

    public static void main(String[] args) {

        // Call with Integer argument.
        action(1);

        // Call with String argument.
        action("cat");
    }
}
```

```
class Item {
    public int size() { // An instance method.
        return 10;
    }
}

public class Program {
    public static void main(String[] args) {

        // The size method can only be accessed from an instance.
        Item item = new Item();
        int value = item.size();
        System.out.println(value);
    }
}
```

```
class Test {
    public void apply() {
        System.out.println("Apply called");
        this.validate();
    }

    private void validate() {
        System.out.println("Validate called");
    }
}

public class Program {
    public static void main(String[] args) {

        // Create new Test and call public method.
        Test t = new Test();
        t.apply();

        // Cannot call a private method:
        // t.validate();
    }
}
```

```java
public class Program {

    static String getName() {
        // This method must return a String.
        return "Augusta";
    }

    public static void main(String[] args) {

        String name = getName();
        System.out.println(name);
    }
}
```

```java
public class Program {

    static void test(int value) {
        if (value == 0) {
            // A void method has no return value.
            // ... We use an empty return statement.
            return;
        }
        System.out.println(value);
        // A return statement is automatically added here.
    }

    public static void main(String[] args) {

        // No variables can be assigned to a void method call.
        test(0); // No effect.
        test(1);
    }
}
```

```
class Cat {
    public boolean isFuzzy() {
        return true;
    }
}

public class Program {
    public static void main(String[] args) {

        Cat c = new Cat();

        // Call boolean instance method.
        if (c.isFuzzy()) {
            System.out.println(true);
        }
    }
}
```

```
public class Program {

    static void displayAll(int... test) {
        // The argument is an int array.
        for (int value : test) {
            System.out.println(value);
        }
        System.out.println("DONE");
    }

    public static void main(String[] args) {
        // Pass variable argument lists to displayAll method.
        displayAll(10, 20, 30);
        displayAll(0);
        displayAll();
    }
}
```

```
public class Program {

    static int computeSize(int height, int width) {
        // Return an expression based on two arguments (variables).
        return height * width;
    }

    public static void main(String[] args) {

        // Assign to the result of computeSize.
        int result = computeSize(10, 3);
        System.out.println(result);
    }
}
```

```
public class Program {

    static int cube(int value) {
        // Return number to the power of 3.
        return (int) Math.pow(value, 3);
    }

    static int getVolume(int size) {
        // Return cubed number.
        return cube(size);
    }

    public static void main(String[] args) {

        // Assign to the return value of getVolume.
        int volume = getVolume(2);
        System.out.println(volume);
    }
}
```

```
public class Program {

    static void displayPassword(String password) {
        // Write the password to the console.
        System.out.println("Password: " + password);
        // Return if our password is long enough.
        if (password.length() >= 5) {
            return;
        }
        System.out.println("Password too short!");
        // An implicit return is here.
    }

    public static void main(String[] args) {
        displayPassword("furball");
        displayPassword("cat");
    }
}
```

```java
public class Program {

    static boolean isValid(String name, boolean exists) {
        // Return a boolean based on the two arguments.
        return name.length() >= 3 && exists;
    }

    public static void main(String[] args) {

        // Test the results of the isValid method.
        System.out.println(isValid("green", true));
        System.out.println(isValid("", true));
        System.out.println(isValid("orchard", false));
    }
}
```

```
public class Program {

    static int getResult(String id) {
        // This method does not compile.
        // ... It must return an int.
        if (id.length() <= 4) {
            System.out.println("Short");
        }
    }

    public static void main(String[] args) {

        int result = getResult("cat");
        System.out.println(result);
    }
}
```

```java
class Data {
    public String name;
    public int size;
}

public class Program {

    static void getTwoValues(Data data) {
        // This method returns two values.
        // ... It sets values in a class.
        data.name = "Java";
        data.size = 100;
    }

    public static void main(String[] args) {

        // Create our data object and call getTwoValues.
        Data data = new Data();
        getTwoValues(data);
        System.out.println(data.name);
        System.out.println(data.size);
    }
}
```

```
public class Program {

    public static void main(String[] args) {

        System.out.println("Hello");
        return;
        System.out.println("World");
    }
}
```

```
public class Program {

    static void test() {
        System.out.println(123);
    }

    public static void main(String[] args) {

        // This does not compile.
        // ... We cannot assign to a void method.
        int result = test();
    }
}
```

```
public class Program {

    public static boolean isPrime(int candidate) {
        // All even numbers except 2 are not primes.
        if ((candidate & 1) == 0) {
            if (candidate == 2) { // Two is prime.
                return true;
            } else {
                return false;
            }
        }
        // Search for prime numbers of the candidate.
        // ... Primes are odd, smaller than the candidate.
        // ... And a modulo division returns 0.
        for (int i = 3; (i * i) <= candidate; i += 2) {
            if ((candidate % i) == 0) {
                return false;
            }
        }
        return candidate != 1;
    }
```

```
    public static void main(String[] args) {

        // Test first 100 numbers for primes.
        for (int test = 0; test < 100; test++) {
            if (isPrime(test)) {
                System.out.println(test);
            }
        }
        // Test larger numbers.
        for (int test = 10000; test < 10100; test++) {
            if (isPrime(test)) {
                System.out.println(test);
            }
        }
    }
}
```

```java
public class Program {

    static int add(int value1, int value2) {
        // This method handles two int arguments.
        return value1 + value2;
    }

    static int add(int value1, int value2, String value3) {
        // This overload handles three arguments.
        return value1 + value2 + value3.length();
    }

    public static void main(String[] args) {

        // Call overloaded add methods.
        int total = add(5, 5);
        total += add(10, 10, "carrot");
        System.out.println(total);
    }
}
```

```
public class Program {

    static int add(int value, int value2, String value3) {
        // This method handles a null String with a branch.
        // ... It could be overloaded.
        if (value3 == null) {
            return value + value2;
        } else {
            return value + value2 + value3.length();
        }
    }

    public static void main(String[] args) {

        // Call add() method twice.
        int total = add(5, 5, null);
        total += add(10, 10, "carrot");
        System.out.println(total);
    }
}
```

```java
class Element {
    int size;
    String color;

    public Element(int size, String color) {
        this.size = size;
        this.color = color;
    }

    public Element(int size) {
        this.size = size;
        this.color = "blank";
    }

    public Element(String color) {
        this.size = 0;
        this.color = color;
    }

    public String toString() {
        // This puts field values into a String.
        return "size = " + this.size + ", color = " + this.color;
    }
}
```

```java
public class Program {
    public static void main(String[] args) {

        // Create with two arguments.
        Element e = new Element(10, "blue");
        System.out.println(e);

        // Use just one argument (size or color).
        e = new Element(50);
        System.out.println(e);

        e = new Element("red");
        System.out.println(e);
    }
}
```

```
import java.lang.Math;

public class Program {
    public static void main(String[] args) {

        // Use exact methods.
        int result1 = Math.addExact(1, 1);
        int result2 = Math.addExact(100, 1);
        int result3 = Math.subtractExact(100, 1);
        int result4 = Math.multiplyExact(5, 4);
        int result5 = Math.incrementExact(2);
        int result6 = Math.decrementExact(1000);
        int result7 = Math.negateExact(100);
        // Display our results.
        System.out.println(result1 + " " + result2 + " " + result3 + " "
                + result4 + " " + result5 + " " + result6 + " " + result7);

        // An ArithmeticException is thrown if a number overflows.
        try {
            int invalid = Math.multiplyExact(Integer.MAX_VALUE, 100);
        } catch (Exception ex) {
            System.out.println(ex);
        }
    }
```