BÖLÜM 2

2. Nesneler ve Sınıflar

Java, Nesne Yönelimli bir dildir. Nesne Yönelimli özelliği olan bir dil olarak Java, aşağıdaki temel kavramları destekler:

- Çok Biçimlilik (Polymorphism)
- Kalıtım (Inheritance)
- Kapsülleme (Encapsulation)
- Soyutlama (Abstraction)
- Sınıflar (Classes)
- Nesneler (Objects)
- Oluşum (Instance)
- Metot (Method)
- Mesajı Ayrıştırma (Message Parsing)

Bu bölümde, Nesneler ve Sınıflar kavramlarını inceleyeceğiz.

- **Nesne –** Nesneler durumlar ve davranışlara sahiptir. Örnek: Bir köpek; renk, isim, doğurganlık gibi niteliklere sahip olmakla birlikte havlama, sallama ve yemek yeme gibi davranışlara da sahiptir. Nesne, bir sınıfın oluşumudur(instance).
- **Sınıf -** Bir sınıf, kendi türünde desteklenen bir nesnenin, niteliklerini/davranışlarını betimleyen bir şablon olarak tanımlanabilir.

2.1 Java Nesneleri

Şimdi de nesnelerin neler olduklarını daha detaylı olarak inceleyelim. Eğer gerçek dünyayı göz önünde bulundurursak, çevremizde bir çok nesne bulabiliriz: Arabalar, Köpekler, İnsanlar gibi. Bütün bu nesneler, nitelik ve davranışlara sahiptirler.

Bir köpek düşünelim, o halde onun nitelikleri – isim, doğurganlık, renk ve davranışları ise – havlama, sallama ve koşma olur.

Eğer bir yazılım nesnesi ile gerçek dünya nesnesini karşılaştırırsanız, birbirine çok benzer özelliklerinin olduğunu görürsünüz.

Yazılım nesneleri de nitelik ve davranışlara sahiptir. Bir yazılım nesnesinin niteliği alanlarda depolanır ve davranışları, metotlar aracılığıyla gösterilir.

Yani, yazılım geliştirme yöntemleri bir nesnenin iç durumu üzerinde çalışır ve nesneden nesneye iletişim metotlar aracılığıyla yapılır.

2.2 Java Sınıfları

Sınıf, nesnelerin tek tek oluşturulduğu bir şablondur. Aşağıda bir sınıf örneği verilmiştir:

```
public class Dog{
   String breed;
   int age;
   String color;

   void barking(){
   }

   void hungry(){
   }

   void sleeping(){
   }
}
```

Bir sınıf aşağıdaki değişkenlerden herhangi birini içerebilir.

- **Yerel değişkenler .** metotların, yapıcıların veya blokların içinde tanımlanmış değişkenlere yerel değişkenler adı verilir. Değişken metot içinde deklare edilip, ilk değeri atanır ve metot tamamlandığında değişken yok edilir.
- **Oluşum(Instance) değişkenleri .** Oluşum(instance) değişkenleri, bir sınıf içinde ama herhangi bir metotun dışında tanımlanan değişkenlerdir. Sınıfı yüklendiğinde bu değişkenler oluşturulur. Oluşum değişkenlerine, belirli bir sınıfın herhangi bir metodu, yapıcısı veya bloğu içinden erişilebilir.
- **Sınıf değişkenleri .** Bir sınıfın içinde , herhangi bir metodun dışında ve static anahtar sözcüğüyle deklare edilmiş değişkenlere sınıf değişkenleri adı verilir.

Bir sınıf, metotların birçok türde değerine erişebilmek için herhangi bir sayıda metoda sahip olabilir. Yukarıdaki örnekte, barking(), hungry() ve sleeping() metotlardır.

Aşağıda, Java Dili sınıflarını gözden geçirirken, tartışılması gereken bazı önemli konulardan bahsedilmiştir.

2.3 Constructor'lar (Yapıcı Metotlar)

Sınıfların en önemli alt konularından biri de Constructor'lardır. Her sınıfın bir constructor'ı vardır. Eğer, bir sınıf için açıkça bir constructor yazmazsak, java derleyicisi bu sınıf için varsayılan bir constructor oluşturur.

Yeni bir nesne oluşturulan her zaman, en azından bir constructor çalıştırılacaktır. Constructor'ların ana kuralı, sınıfı ile aynı ismi taşımasıdır. Bir sınıfı birden çok constructor'ı içerebilir. Constructor örneği aşağıda verilmiştir:

```
public class Puppy{
   public puppy(){
   }

   public puppy(String name){
      // This constructor has one parameter, name.
   }
}
```

2.4 Nesne Oluşturma

Daha önceden de belirtildiği gibi; sınıflar, nesneler için şablonlar sağlamaktadır. Yani temelde bir nesne, bir sınıftan oluşturulur. Java da, new anahtar sözcüğü, yeni nesneler oluştururken kullanılmaktadır.

Bir sınıftan bir nesne oluştururmanın 3 adımı vardır:

- **Declaration(Deklarasyon).** Nesne türü ve değişen adıyla değişken deklare etme.
- **Instantiation(Örnekleme).** Nesne oluşturmak için 'new' anahtar sözcüğü kullanılır.
- **Initialization(İlk değerini atama).** 'new' anahtar sözcüğü constructor'ın çağrılması tarafından takip edilir. Bu çağrılma, new nesnesine ilk değerini atar.

Aşağıda nesne oluşturma örneği verilmiştir:

```
public class Puppy{

   public Puppy(String name){
      // This constructor has one parameter, name.
      System.out.println("Passed Name is :" + name );
   }
}
```

```
public class testclass {
    public static void main(String []args){
    // Following statement would create an object myPuppy
    Puppy myPuppy = new Puppy( "tommy" );
   }
}
```

Yukarıdaki programı derler ve çalıştırırsak, aşağıdaki sonuç üretilecektir:

```
Passed Name is :tommy
```

2.5 Oluşum(Instance) Değişkenlerine ve Metotlara Erişim

Oluşum değişkenlerine ve metotlara, oluşturulmuş nesneler aracılığıyla erişilir. Bir oluşum değişkenine erişmenin yolu aşağıdaki gibi olmalıdır:

```
/* First create an object */
ObjectReference = new Constructor();

/* Now call a variable as follows */
ObjectReference.variableName;

/* Now you can call a class method as follows */
ObjectReference.MethodName();
```

Örnek:

Bu örnek oluşum(instance) değişkenlere ve metotlara nasıl erişildiğini göstermektedir:

```
public class Puppy{

   int puppyAge;
```

```
   public Puppy(String name){
      // This constructor has one parameter, name.
      System.out.println("Passed Name is :" + name );
   }
   public void setAge( int age ){
       puppyAge = age;
   }

   public int getAge( ){
       System.out.println("Puppy's age is :" + puppyAge );
       return puppyAge;
   }
}
```

```
public class testclass {
    public static void main(String []args){
      /* Object creation */
      Puppy myPuppy = new Puppy( "tommy" );

      /* Call class method to set puppy's age */
      myPuppy.setAge( 2 );

      /* Call another class method to get puppy's age */
      myPuppy.getAge( );

      /* You can access instance variable as follows as well */
      System.out.println("Variable Value :" + myPuppy.puppyAge );
   }
}
```

Yukarıdaki programı derler ve çalıştırırsak, aşağıdaki sonuç üretilecektir:

```
Passed Name is :tommy
Puppy's age is :2
Variable Value :2
```

2.6 Kaynak Dosya Deklarasyon Kuralları

Bu bölümün son kısmı, kaynak dosya deklarasyon kurallarını incelememizi sağlamaktadır. Bu kurallar , sınıf deklarasyonları ve kaynak dosyasının içindeki import ve paket ifadeleri için gereklidir.

- Her kaynak dosyası için sadece bir tane public sınıf olabilir.
- Bir kaynak dosyası bir çok non public sınıfa sahip olabilir.
- Public sınfın ismi, kaynak dosyasının ismiyle aynı olmalı ve sonuna .java uzantısı eklenmiş olmalıdır. Örneğin: Sınıfın ismi ; *public class Employee{}* ise kaynak dosyasının ismide Employee.java olmalıdır.

- Eğer sınıf bir paketin içinde tanınlanmışsa, paket ifadesi kaynak dosyasının içindeki ilk ifade olmalıdır.
- Eğer import ifadeleri varsa, paket ifadesi ve sınıf deklarasyonu arasına yazılmalıdırlar. Eğer hiçbir paket ifadesi yoksa, import ifadesi kaynak kodunun ilk satırında olmalıdır.

2.7 Java Paketleri (package)

Basit olarak sınıflar ve arayüzlerin kategorize edilme yoludur. Java da uygulama geliştirirken, yüzlerce sınıf ve arayüz yazılacaktır, sonuç olarak bu sınıfları kategorize etmek büyük rahatlık sağlar.

2.8 Import İfadeleri

Javada, daha sonra derleyici kaynak kodunu ve sınıfların yerini kolay bir şekilde bulabilsin diye için paket ve sınıf ismini içeren bir tam nitelikli isim kullanılır. Import ifadesi; belli bir sınıfın, uygun konumunu derleyiciye verme işlemidir.

Örneğin aşağıdaki satır, derleyiciden java_installation/java/io dizinindeki bütün sınıfların yüklenmesini istemektedir.

```
import java.io.*;
```

2.9 Örnek Çalışma

Bu örnek çalışma 2 adet sınıftan oluşmaktadır. Bunlar Employee ve EmployeeTest sınıflarıdır.

Öncelikle, notepad'i açın ve aşağıdaki kodu ekleyin. Unutmayın ki, bu Employee sınıfıdır ve bu sınıf public bir sınıftır. Şimdi, kaynak dosyayı Employee.java ismiyle kaydedin.

Employee sınıfı; name, age, designation ve salary adında 4 değişkene sahiptir. Sınıf, bir adet parametre almış constructor'a sahiptir.

```
import java.io.*;
public class Employee{
   String name;
   int age;
```

```
   String designation;
   double salary;

   // This is the constructor of the class Employee
   public Employee(String name){
      this.name = name;
   }
   // Assign the age of the Employee  to the variable age.
   public void empAge(int empAge){
      age =  empAge;
   }
   /* Assign the designation to the variable designation.*/
   public void empDesignation(String empDesig){
      designation = empDesig;
   }
   /* Assign the salary to the variable    salary.*/
   public void empSalary(double empSalary){
      salary = empSalary;
   }
   /* Print the Employee details */
   public void printEmployee(){
      System.out.println("Name:"+ name );
      System.out.println("Age:" + age );
      System.out.println("Designation:" + designation );
      System.out.println("Salary:" + salary);
   }
}
```

Önceki çalışmalarda belirtildiği gibi, işlem main metodundan itibaren başlamaktadır. Sonuç olarak, *Employee* sınıfını çalıştırmak için bir main metodu olmalı ve nesneler oluşturulmalıdır. Bu görevler için ayrı bir sınıf oluşturmalıyız.

Aşağıda verilen *EmployeeTest* sınıfı, *Employee* sınıfından 2 adet instance(oluşum) oluşturur ve her bir değişkene değerini atamak için her nesnenin metodunu çalıştırır.

Aşağıdaki kodu, *EmployeeTest.java* dosyası içine kaydedin.

```
import java.io.*;
public class EmployeeTest{

   public static void main(String args[]){
      /* Create two objects using constructor */
      Employee empOne = new Employee("James Smith");
      Employee empTwo = new Employee("Mary Anne");

      // Invoking methods for each object created
      empOne.empAge(26);
      empOne.empDesignation("Senior Software Engineer");
      empOne.empSalary(1000);
      empOne.printEmployee();

      empTwo.empAge(21);
      empTwo.empDesignation("Software Engineer");
      empTwo.empSalary(500);
      empTwo.printEmployee();
   }
}
```

Şimdi her iki sınıfı da derleyin ve ardından aşağıdaki sonucu görmek için *EmployeeTest* dosyasını çalıştırın.

```
C :> javac Employee.java
C :> vi EmployeeTest.java
C :> javac  EmployeeTest.java
C :> java EmployeeTest
Name:James Smith
Age:26
Designation:Senior Software Engineer
Salary:1000.0
Name:Mary Anne
Age:21
Designation:Software Engineer
Salary:500.0
```