Java dilində obyekt yönümlü proqramlaşdırma (OOP-Object-Oriented Programming) anlayışının öyrədilməsi metodikası

Fərid Yusif oğlu Əhmədov
Azərbaycan Dövlət Pedoqoji Universiteti
E-mail: feridehmedov1999@gmail.com

Rəyçilər: r.ü.f.d., dos.Z.Ə. Tağıyeva,
p.ü.f.d., dos.S.C.-C. Cəbrayılzadə

Açar sözlər: OYP – obyekt yönümlü proqramlaşdırma, Object, Class, Inheritance, Interface, Encapsulation, Field, Property, Method, Instantiation, Initialization

Ключевые слова: ООП - Объектно-ориентированное программирование, объект, класс, наследование, интерфейс, инкапсуляция, поле, свойство, метод, реализация, инициализация

Key words: OOP - Object-oriented programming, Object, Class, Inheritance, Interface, Encapsulation, Field, Property, Method, Instantiation, Initialization

Hazırda bir çox obyekt yönümlü proqramlaşdırma dilləri mövcuddur. Əvvəllər proqramların yazılmasına və sazlanmasına proqramçının nə qədər vaxt sərf etməsi o qədər də önəmli deyildi. Fərdi kompüterlərin meydana çıxması ilə proqramçılar vaxtları hədər getməsin deyə proqramlaşdırma mühitini təkmilləşdirməyə başladılar. İlk növbədə koddan təkrar istifadə prinsipi yarandı. Bu prinsipə görə, kimlərnsə bir dəfə yaratmış olduqları toplanıb saxlanmalı və başqa proqramçılar tərəfindən hazır bloklar şəklində istifadə olunmalıdır. Belə proqram bloklarını obyektlər (OBJECT) adlandırdılar. Yeni proqram hazırlamaq lazım gəldikdə əvvəlki proqramlardan obyektlər götürülür və onlar sadəcə, yeni tələblərə uyğun dəyişdirilir.

Java dilində Obyekt Yönümlü Proqramlaşdırma anlayışının ali məktəbdə tələbələrə tədrisi metodikası belədir. Belə ki, ardıcıllıq olaraq Obyekt Yönümlü Proqramlaşdırmanın (OYP) mahiyyəti izah edilir. Sonra isə OYP-nin elementərinin hər biri tələbələr üçün real həyat nümunələri ilə əlaqələndirilərək izah edilir. Tədris prosesi zamanı Java dilində OYP-nin əhəmiyyəti qabardılır.

İndi isə Java dilində OYP anlayışının nə olduğuna baxaq. Java dilində OYP anlayışl nədir? Bunlara *object, class, inheritance, interface, encapsulation*… kimi anlayışlar aiddir. Onları nə demək olduğunu danışarkən, bu anlayışların real həyat nümunələri ilə əlaqələndirilərək yaxşı başa düşülmələrini təmin etməyi hədəfləyirəm. Əlbəttə, nümunələr Java sintaksisindən istifadə edilərək veriləcəkdir. Terminləri izah edərkən onu Azərbaycan dilinə çevirməməyi üstün tuturam.

OOP məntiqini anlamaq üçün əsas *object* anlayışını anlamaqdır. *Object* ən sadə şəkildə düşünün. Proqramlaşdırmanı unudun. Ətrafınıza baxın. Gördüyünüz şeylərin hamısı *object*'dir. Stolunuzdakı qələm, bağdakı ağac, yolda maşın, hamısı *object*'dir. Bunların ortaq 2 nöqtəsi var. Bu ortaq nöqtələrə diqqət edəcəyik.

İlk ortaq nöqtə, hər bir *object*'in öz xüsusiyyətlərinə sahib olmasıdır. Məsələn, qələmin rəngi, ağacın hündürlüyü, avtomobilin markası ... Hər bir *object*'in xüsusiyyətləri var. Başqa bir ümumi nöqtə, hər bir *object*'in davranış və bir hərəkətə sahib olmasıdır. Məsələn, qələm yazır, ağac böyüyür, avtomobil hərəkət edir... Onları çoxaltmaq mümkündür, məsələn, avto-

mobil üçün əyləc, yanacaq doldurma, farları yandırmaq kimi hərəkətlərdən danışmaq olar.

Bu nöqtələrə baxaraq OOP dünyasına girə bilərsiniz. Proqram *object*‘lərinin həqiqi dünya objectləri kimi xüsusiyyətlərə və davranışlara malikdir. Proqram *object*‘lərinin xüsusiyyətləri field adlanan sahələrdə (bəzi dillərdə property adlanır) və method adlanan quruluşlarda (bəzi dillərdə function da deyilir) göstərilmişdir.

*Method*lar *object*‘in xüsusiyyətlərini idarə etmək və *object*‘in xarici *object*‘lərlə əlaqə qura bilməsini təmin edir. Bu şəkildə *object*‘in daxili quruluşu digər *object*‘lərdən gizlidir. Başqa sözlə, digər *object*‘lər xüsusiyyətlərə çata bilməzlər, yalnız *method*lar idarə olunan bir şəkildə əlaqə qururlar. Buna data encapsulation deyilir. Bu *object* yönümlü proqramlaşdırmanın əsaslarından biridir.

Object istifadə edərək kod yazmağın çox təsirli faydaları var;

• Modulluq: tətbiqlərinizdən asılı olmayaraq *object*‘ləri dizayn edə və kodlaşdıra bilərsiniz. *Object*‘i yaratdıqdan sonra tətbiqinizin hər hansı bir yerinə daxil olub istifadə edə bilərsiniz.

• Məxfilik: *Object*‘in içərisində nə baş verdiyini bilmirsiniz. Komandadakı başqa bir proqram tərtibçisi bir *object* tərtib etmiş ola bilər. Bu *object*‘in xüsusiyyətlərini bilmirsiniz, onun metodlarının necə işlədiyini bilməlisiniz.

• Kod təkrarlanmasının qarşısının alınması. Bir *object* yazılıbsa, onu başqası da yaza bilər. Bunu hər tətbiqdə istifadə edə bilərsiniz. Məsələn, tətbiqinizdə bir fayl *object*‘i istifadə etmisinizsə, digər tətbiqlərdə də istifadə edə bilərsiniz. Yenidən yazmağa ehtiyac yoxdur.

• Problemlərin həlli: Sistemdə bir səhv baş verdikdə, bir *object*, ehtimal ki, öz işini düzgün yerinə yetirə bilmir. Bu vəziyyətdə, sadəcə həmin *object* düzəltmək və dəyişdirməklə bütün sistemin düzgün işləməsini təmin edə bilərsiniz.

Beləliklə, eyni xüsusiyyətlərə və eyni hərəkətlərə sahib bir çox maşın var. Hər avtomobildə rəng, sürət, ötürücü ... və s. Tək bir avtomobilə *object* deyilirsə, bütün avtomobillər üçün ümumi bir dizayn hazırlanırsa, bu bir *class* (sinif) adlanır. Başqa sözlə, *object* bir classdan törəmiş (instance) bir nümunədir. Avtomobil üçün java’da belə bir *class* yazıla bilər.

```
1  class Car {
2
3      string color = "black";
4       int gear = 1;
5      void changeColor(string newValue) {
6          color = newValue;
7      }
8      void changeGear(int newValue) {
9          gear = newValue;
10     }
11     }
```

Həqiqi həyat mirası kimi, uşaq da valideynlərinin bəzi xüsusiyyətlərini alır, həm də öz xüsusiyyətlərinə malikdir. Proqramda bir *class*‘ın alt *class*‘lari ola bilər. Bu alt *class*‘lar yuxarı *class*‘da müəyyən edilmiş *field* və *method*‘ları *inheritance* alır və öz xüsusi *field* və *method*‘ları da tətbiq edə bilər. Bir üst *class*‘ın *field* və *method*‘ları daha aşağı *class inheritance* ilə köçürülməsi üçün önündə private olmamalıdır. Bu vəziyyətdə, bütün *field* və *method*‘lar *class*‘da yazılmasa belə istifadə edilə bilər.

• Bir *class* alt *class*'larinin sayında məhdudiyyət yoxdur.

• Bir *class*'da yalnız bir yuxarı *class* ola bilər. Java çox *inheritance* xüsusiyyətini dəstəkləmir.

Java-da alt *class* tətbiqetmə aşağıdakı kimi aparılır;

```
1    class Truck extends Car {
2
3    // Truck spesifik field ve method'lar burada təyin olunur.
4
5    }
```

Bir *interface* yaratarkən bu *interface*'in aid classların olmalı olduğu methodları yazırıq. Ancaq biz bunu implement etmirik, sadəcə olaraq adlarını yazarıq. Avtomobil classı üçün bir *interface* olsaydı;

```
1
2     interface ICar {
3
4        void changeColor(string newValue);
5
6        void changeGear(int newValue);
7     }
```

kimi təyin edirik. İzləyici heç bir detal görmədən, sadəcə bir avtomobil class'ından necə istifadə edəcəyini başa düşür. Bu üsulların avtomobil class'ında olacağına əmin olur. Əlbəttə ki, bu *interface* avtomobil class'ınıda implement etdiyinizi göstərməlisiniz. Burada bunu edirsiniz.

```
1    class Car implements ICar {
2       // Daha əvvəl Car class'ını yazdıq. Car class'ını daha önce yazmıştık.
3       // Interface'de göstərilən methodlar burada implement olunur.
4    }
```

Bir *class* yazdıqdan sonra həmin *class'dan* istifadə edən objectlərin yaradılması instantiation adlanır. Beləliklə, siz o *class'dan* bir nümunə yaradırsınız. Bunun üçün new açar söz istifadə olunur. New açar söz yaddaşda kifayət qədər yer ayırır və bu *class'ın* bir nümunəsinə uyğun gəlir və constructor methodunu başlanğıc üçün çağırır.

Hər *class* üçün xüsusi bir method var. Bu methodu digər methodlardan fərqləndirən cəhət odur ki, *class* ilə eyni ada malikdir və heç bir dəyər qaytarmır. Ayrıca bu method *class*'dan *object* yaratdıqda, yəni instantiation mərhələsində çağrılır. *Class*'ınızda bir *constructor* yoxdursa, Java Compiler açıq şəkildə yaradır və heç bir parametr götürməyən bir *constructor* çağırır. Bu *constructor* içərisində də *class*'ınız *üst class*'ına aid boş *constructor* çağırılır. Əgər bir üst *class* yoxdursa Java'da build-in olaraq mövcud olan Object adlı *class*'ın boş *constructor'u* çağırılır (Object *class*'ı təməl bir *class*'dır və biz yazmasaqda əslində hər *class* Object *class*'ından inherit edər). Burada *constructor*'larınızı yazmağı təklif edirəm. Kompilyatorun bu işi sizin üçün həll etməsinə icazə verməyin, əks halda problemlərlə üzləşə bilərsiniz.

Dedik ki, *instantiation* mərhələsində object üçün yaddaşda yer ayrılır və *constructor* çağırılır. *Constructor* bu sahədəki field'lərə ilkin dəyərlər təyin edir, bu proses initialization adlanır. Təcrübə edərək son 3 başlıqda dediklərimi göstərim;

```
1       class Car {
2
3           private string color;
4           private int speed;
5           private int gear;
6
7           public Car () {
8               color = "black";
9               speed = 0;
10              gear  = 1;
11          }
12
13          public Car (string newColor, int newSpeed, int newGear) {
14              color = newColor;
15              speed = newSpeed;
16              gear  = newGear;
17          }
18      }
```

Yuxarıdakı Car class'ını yenidən yazdım. Bu classın objectlərini yaradaq;

```
1     public class CarFactory {
2
3         public static void main(String[] args) {
4             //* c1 ve c2 adlı 2 Car object'i yaradaq
5             Car c1 = new Car();
6             Car c2 = new Car("white", 80, 2);
7         }
8     }
```

Buradakı kodun Car c1 hissəsi decleration kimi tanınır və hər hansı bir primitive variable yaratmağa bənzəyir. Məsələn, int say=5; Dediyiniz kimi, objectiniz üçün dəyişən yaradırsınız. Ancaq burada bir fərq var, bu dəyişən (c1 və c2) objectin özünü saxlamır, objectin yerləşdiyi memory sahəsinə, yəni bir pointer'dir. Sahəni new ilə ayırırsınız, sahənin adresini dəyişənə təyin edir və *constructor* ilə başlatdırırsınız.

Məqalənin aktuallığı. Hazırda bir çox obyekt yönümlü proqramlaşdırma dilləri mövcuddur. Fərdi kompüterlərin meydana çıxması ilə proqramçıların vaxtını hədər yerə sərf etməmək üçün proqramlaşdırma mühiti getdikcə təkmilləşdi. Məqalə də Java dilində obyekt yönümlü proqramlaşdırma anlayışının ali məktəbdə tələbələrə tədrisinin əsas istiqamətlərinin təhlili baxımından aktual əhəmiyyət kəsb edir..

Məqalənin elmi yeniliyi. Elmi yenilik ondan ibarətdir ki, məqalədə Java dilində obyekt yönümlü proqramlaşdırma anlayışının ali məktəbdə tələbələrə tədrisi metodikasının təsnifatı verilməklə, həmçinin ayrı-ayrı anlayışlar geniş təhlili olunur.

Məqalənin praktik əhəmiyyəti və tətbiqi. Məqalədən orta ixtisas məktəblərinin müəllimləri, tələbə və magistrantlar istifadə edə bilər.

Ədəbiyyat

1. Java Programlama Dili ve Yazılım Tasarımı. Altuğ B.Altıntaş (Java dilinə başlayanlar üçün əla bir kitab) 2010.
2. Jon Byous. "Java Technology" 2005 the early years.
3. https://www.w3schools.com/java/java_oop.asp

Ф.Ю. Ахмедов

Методика обучения концепции объектно-ориентированного программирования на Java

Резюме

В данной статье рассматривается методика преподавания и обучения концепции объектно-ориентированного объектного программирования на Java для студентов университетов. Таким образом, в статье представлена классификация методологии обучения концепции объектно-ориентированного программирования на Java учащимся старших классов средней школы, дан комплексный анализ отдельных концепций.

F.Y. Ahmadov

Methodology of teaching the concept of Object-Oriented Programming in Java

Summary

This article covers the methodology for teaching and teaching the concept of Object Oriented Programming in Java for university students. Thus, the article provides a classification of the methodology of teaching the concept of Object Oriented Programming in Java to students in high school, provides a comprehensive analysis of individual concepts.

Redaksiyaya daxil olub: 11.09.2020