Technology of Using Java Program in Teaching Computer Science Course Justification of the Choice of the Topic of the Dissertation

Assos. prof. Sevinj Jabrayilzada Azerbaijan/Baku

Keywords: OYP - Object Oriented Programming, Object, Class, Inheritance, Interface, Encapsulation, Field, Property, Method, Instantiation, Initialization.

Annotation. Many Object Oriented Programming languages are currently available. Previously, it did not matter how much time a programmer spent writing and setting up programs. With the advent of personal computers, programmers began to improve the programming environment so as not to waste time. First of all, the principle of reusing the code was born. According to this principle, what someone once created must be collected and stored and used by other programmers in the form of ready-made blocks. Such program blocks are called objects (OBJECT). When a new program needs to be developed, objects are removed from previous programs and simply modified to meet the new requirements.

Practical significance: The concept of Object Oriented Programming in Java The methodology of teaching students in higher education is as follows: Thus, the essence of Object Oriented Programming (OOP) is explained sequentially. During the process, the importance of OYP in Java is emphasized.
Now let's look at the concept of OYP in Java. What is OYP in Java? These include concepts such as Object, class, inheritance, interface, encapsulation…. Speaking of what they mean, I aim to ensure that these concepts are well understood by relating them to real life examples. Of course, examples will be given using Java syntax. When explaining the terms, I prefer not to translate it into Azeri.

Object: The key to understanding OOP logic is to understand the concept of Object. Think of the object in the simplest way. Forget programming. Look around you. Everything you see is an object. The pen on your desk, the tree in the garden, the car on the road, it's all an object. They have 2 points in common. We will focus on these common points.
The first common point is that each object has its own characteristics. For example, the color of the pen, the height of the tree, the make of the car ... Each object has its own characteristics. Another common point is that each object has a behavior and an action. For example, a pen writes, a tree grows, a car moves ... They can be multiplied, for example, we can talk about actions such as braking a car, refueling, turning on the headlights.
You can enter the world of OOP by looking at these points. Software objects have the same characteristics and behavior as real-world objects. The properties of program objects are shown in fields called fields (sometimes called property) and structures called methods (also called functions in some languages).
Methods allow you to control the properties of an object and allow the object to interact with external objects. In this way, the internal structure of the object is hidden from other objects. In other words, other objects cannot access properties, only communicate in a method-controlled way. This is called data encapsulation. This is one of the basics of object-oriented programming.
There are many effective benefits to writing code using an object;

• **Modularity**: you can design and encode objects regardless of your application. After creating an object, you can access and use it anywhere in your application.

• **Privacy:** You do not know what is going on inside the Object. Another programmer in the team may have compiled an object. You don't know the properties of this object, you need to know how its methods work.

• **Prevent code duplication**: If one object is written, it can be written by another. You can use it in any application. For example, if you used one file object in your application, you can use it in other applications. No need to rewrite.

• **Troubleshooting**: When an error occurs in the system, an object may not be able to do its job properly. In this case, you can ensure the proper functioning of the entire system by simply correcting and replacing that object.

Class: So there are many machines with the same features and the same actions. Each car has its own color, speed, transmission ... etc. If a single car is called an object, and a common design is made for all cars, it is called a class. In other words, an object is an instance of a class. Such a class can be written in java for the car.

```
class Car {

    string color = "black";
     int gear = 1;
    void change Color(string new Value) {
        color = new Value;
    }
    void change Gear(int new Value) {
        gear = new Value;
    }
    }
```

Inheritance: Like a real-life legacy, a child acquires some of the characteristics of his parents, but also has his own characteristics. A program can have subclasses of a class. These subclasses inherit the fields and methods defined in the upper class and can also apply their own custom fields and methods. The fields and methods of an upper class must not be private in order to be inherited by a lower class inheritance. In this case, all fields and methods can be used even if they are not written in the class.
• There is no limit to the number of subclasses in a class.
• There can be only one upper class in a class. Java does not support many inheritance features.
In Java, a subclass application is implemented as follows;

```
class Truck extends Car {

//Truck spesifik field ve method'lar burada təyin olunur.

}
```

Interface: When creating an interface, we write the methods by which this interface must have classes. But we do not implement it, we just write their names. If there was an interface for the car class;

```
interface ICar {

    void change Color(string new Value);

    void change Gear(int new Value);
  }
```

as we define. The viewer understands how to use a car class without seeing any details. Make sure these methods are in the car class. Of course, you must show that you implement this interface in the car class. You do it here.

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

Instantiation: After writing a class, creating objects that use that class is called instantiation. So you create an example from that class. A new keyword is used for this. The new keyword allocates enough space in memory and corresponds to an instance of this class and calls the constructor method to start.

Constructor: There is a specific method for each class. What distinguishes this method from other methods is that it has the same name as the class and does not return any value. This method is also called when creating an object from a class, that is, during the instantiation phase. If your class does not have a constructor, the Java Compiler explicitly creates it and calls a constructor that does not take any parameters. This constructor also calls an empty constructor belonging to the upper class of your class. If there is no top class, the empty constructor of the Object class, which exists as a build in Java, is called (the Object class is the base class, and if we do not write, in fact, each class inherits from the Object class). I suggest you write your constructors here. Do not allow the compiler to solve this problem for you, otherwise you may face problems.

Initialization: We said that in the instantiation phase, space is allocated in memory for the object and the constructor is called. The constructor assigns initial values to the fields in this field, a process called initialization. Let me show what I said in the last 3 headings by experimenting;

```
class Car {

    private string color;
    private int speed;
    private int gear;

    public Car () {
        color = "black";
        speed = 0;
        gear  = 1;
    }

    public Car (string new Color, int new Speed, int new Gear) {
        color = new Color;
        speed = new Speed;
        gear  = new Gear;
    }
}
```

I rewrote the Car class above. Let's create objects of this class;

```
public class Car Factory {

    public static void main(String[] args) {
        //* c1 ve c2 adlı 2 Car object'i yaradaq
        Car c1 = new Car();
        Car c2 = new Car("white", 80, 2);
    }
}
```

The Car c1 part of the code here is known as decleration and is similar to creating any primitive variable. For example, int number = 5; As you said, you create a variable for your object. However, there is a difference, this variable (c1 and c2) does not store the object itself, but the memory area where the object is located, ie a pointer. You separate the field with new, assign the field address to the variable, and start with the constructor.

REFERENCE

[1]. Java Programming Language and Writing Design. Altug B. Altintas (an excellent book for beginners in Java) 2010.

[2]. Jon Byous "Java Technology" 2005 the early years.

[3]. https://www.w3schools.com/java/java_oop.asp

Assos. prof. Sevinj Jabrayilzada Azerbaijan/Baku