AZƏRBAYCAN RESPUBLİKASI TƏHSİL NAZİRLİYİ

BAKI MÜHƏNDİSLİK UNİVERSİTETİ

Kompüter və İnformasiya Texnologiyaları kafedrası

"*Object Oriented Programming*" fənnindən

K U R S İ Ş İ

Mövzu: *Object Oriented Programming* dilləri

_____

_____

İxtisas İnformasiya texnologiyaları

Tələbə Məmmədov Fərid

Rəhbər Etibar Seyidzadə

XIRDALAN – 2019

TABLE OF CONTENT

OOP LANGUAGES

Object-oriented programming (**OOP**) is a programming paradigm based on the concept of “objects”, which may contain data, in the form of fields, often known as attributes*;* and code, in the form of procedures, often known as methods. A feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of “this” or "self"). In OOP, computer programs are designed by making them out of objects that interact with one another There is significant diversity of OOP languages, but the most popular ones are class-based, meaning that objects are instances of classes, which typically also determine their type.
Object Oriented Programming provides a means of structuring programs so that properties and behaviors are bundled into individual object. For instance, an object could represent a person with a name property, age, address, etc., with behaviors like walking, talking, breathing, and running. Or an email with properties like recipient list, subject, body, etc., and behaviors like adding attachments and sending.

Put another way, object-oriented programming is an approach for modeling concrete, real-world things like cars as well as relations between things like companies and employees, students and teachers, etc. OOP models real-world entities as software objects, which have some data associated with them and can perform certain functions.

Object-oriented programming (OOP) languages are designed to overcome these problems.

1. The basic unit of OOP is a *class*, which encapsulates both the *static properties* and *dynamic operations* within a "box", and specifies the public interface for using these boxes. Since classes are well-encapsulated, it is easier to reuse these classes. In other words, OOP combines the data structures and algorithms of a software entity inside the same box.
2. OOP languages permit *higher level of abstraction* for solving real-life problems. The traditional procedural language (such as C and Pascal) forces you to think in terms of the structure of the computer (e.g. memory bits and bytes, array, decision, loop) rather than thinking in terms of the problem you are trying to solve. The OOP languages (such as Java, C++ and C#) let you think in the problem space, and use software objects to represent and abstract entities of the problem space to solve the problem.

Historically, "OOP" has been one of the most influential developments in computer programming, gaining widespread use in the mid-1980s. Originally heralded for its facility for managing complexity in ever-growing software systems, OOP quickly developed its own set of difficulties. Fortunately, the ever evolving programming landscape gave us "interface" programming, design patterns, generic programming, and other improvements paving the way for more contemporary Multi-Paradigm programming.

Object oriented programming can be traced back to a language called Simula, and in particular 67, Simula is considered the first object-oriented programming language. The programming paradigm where everything is represented as an object is known as a truly object-oriented programming language. Smalltalk is considered the first truly object-oriented programming language. Simula first instituted "classes" and "objects," leading to the term "object oriented"

programming. By the early 1990s, enough experience had been gained with large OOP projects to discover some limitations. Languages such as Self, ideas like interface

programming (also known as component or component-oriented programming), and methodologies such as generic programming were being developed in response to these difficulties. Although often derided by OOP purists, it was the standardization of C++ in 1998 —including generic programming facilities — that really ushered in the modern era of OOP, which we also refer to as Multi-Paradigm programming. This is largely due to the popularity of C++ and the genius of the Standard Template Library (STL) demonstrating the utility of the new methodology to such a large audience.

Many of the most widely used programming languages (such as C++, Object Pascal, Java, Python etc.) are multi-paradigm programming languages that support object-oriented programming to a greater or lesser degree, typically in combination with imperative, procedural programming. Significant object-oriented languages include Java, C++, C#, Python, JavaScript, PHP, Ruby, Perl,

Object Pascal, Dart, Swift, Scala, Common Lisp, Smalltalk and etc.

Languages called "pure" OOP languages, because everything in them is treated consistently as an object, from primitives such as characters and punctuation, all the way up to whole classes, prototypes, blocks, modules, etc. They were designed specifically to facilitate, even enforce, OO methods. Examples: Python, Ruby, Scala, Smalltalk, Eiffel, JADE, Self. **P.S**. Java is not a pure object oriented language. Because to trigger main function we don't need objects as well as the primitives like int, float, double, etc. still exists. Though there are classes for the same but we can write code without creating objects.

float, double, etc. still exists. Though there are classes for the same but we can write code without creating objects.

Languages designed mainly for OO programming, but with some procedural elements. Examples: Java, C++, C#, Delphi, VB.NET.

Languages that are historically procedural languages, but have been extended with some OO features. Examples: PHP, Perl, Visual Basic (derived from BASIC), Matlab, Fortran 2003, ABAP, Pascal.

CONCEPS OF OBJECT ORIENTED PROGRAMMING LANGUAGES

An object-based application is based on declaring classes, creating objects from them and interacting between these objects. Classes and Objects which is also a part of object-oriented programming concepts. we will understand the below core concepts of Object Oriented Programming in the following sequence:

1. Inheritance
2. Encapsulation
3. Abstraction
4. Polymorphism

What is Inheritance

Inheritance is the OOP ability that allows classes to be derived from other classes. The parent class is called a superclass and the derivatives are called subclasses. Subclasses inherit fields and methods from their superclasses.

Inheritance is one of the four major concepts behind object oriented programming(OOP). OOP questions are very common on job interviews, so you may expect questions about inheritance on your next Java job interview. The “mother of all classes” in Java is the Object class. Each and every class in Java inherits from Object. At the top of the hierarchy, Object is the most general of all classes. Classes near the bottom of the hierarchy provide more specialized behavior. Java has a single inheritance model, which means every class has one and only one direct superclass.

In object oriented programming languages there are two classes:

1. Parent class (Super or Base class)

2. Child class (Subclass or Derived class)

A class which inherits the properties is known as Child Class whereas a class whose properties are inherited is known as Parent class.

It’s quite simple to achieve inheritance as an OOP concept in Java. Inheritance can be as easy as using the **extends** keyword:

```
class ParentClass{
  //Parent class constructor
  ParentClass(){
            System.out.println("Constructor of Parent");
  }
  void disp(){
            System.out.println("Parent Method");
  }
}
class JavaExample extends ParentClass{
  JavaExample(){
            System.out.println("Constructor of Child");
```

```
  }
  void disp(){
        System.out.println("Child Method");
     //Calling the disp() method of parent class
        super.disp();
  }
  public static void main(String args[]){
        //Creating the object of child class
        JavaExample obj = new JavaExample();
        obj.disp();
  }
}
```

What is Encapsulation

Encapsulation simply means binding object state(fields) and behaviour(methods) together. If you are creating class, you are doing encapsulation. In this guide we will see how to do encapsulation in java program, if you are looking for a real-life example of encapsulation then refer this guide: OOPs features explained using real-life examples. The whole idea behind encapsulation is to hide the implementation details from users. If a data member is private, it means it can only be accessed within the same class. No outside class can access private data member (variable) of other class. his way data can only be accessed by public methods thus making the private fields and their implementation hidden for outside classes. That's why encapsulation is known as **data hiding.** Let's see an example to understand this concept better.

How to implement encapsulation in java:
1) Make the instance variables private so that they cannot be accessed directly from outside the class. You can only set and get values of these variables through the methods of the class.

2) Have getter and setter methods in the class to set and get the values of the fields.

public: The class/variable/method is accessible and available to *all* the other objects in the system.

private: The class/variable/method is accessible and available *within this class only*.

Here is Java Encapsulation example that used **private** and **public**:

```
public class Student {

 private String name;

 public String getName() {

  return name; }

 public void setName(String name) {

 this.name = name; }

}

package com.javatpoint;

class Test {

 public static void main(String[] args) {

  Student s = new Student();
```

```
 s.setName(“Emir”);

 System.out.println(s.getName()); }

}
```

What is Abstraction

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it. A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

```
//Example of an abstract class that has abstract and non-abstract methods
abstract class Bike{
  Bike(){System.out.println("bike is created");}
  abstract void run();
  void changeGear(){System.out.println("gear changed");}
}
//Creating a Child class which inherits Abstract class
class Honda extends Bike{
void run(){System.out.println("running safely..");}
}
//Creating a Test class which calls abstract and non-abstract methods
```

```
class TestAbstraction2{
public static void main(String args[]){
 Bike obj = new Honda();
 obj.run();
 obj.changeGear();
 }
}
```

What is Polymorphism

Polymorphism is a concept by which we can perform a *single action in different ways*. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding. If you overload a static method in Java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.Runtime polymorphism or Dynamic Method Dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile-time.In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

```
class Overload
{
   void demo (int a){
      System.out.println ("a: " + a);
   }
   void demo (int a, int b){
      System.out.println ("a and b: " + a + "," + b);
```

```
    }
    double demo(double a) {
        System.out.println("double a: " + a);
        return a*a;
    }
}
class MethodOverloading{
    public static void main (String args [])
    {
        Overload Obj = new Overload();
        double result;
        Obj .demo(10);
        Obj .demo(10, 20);
        result = Obj .demo(5.5);
        System.out.println("O/P : " + result);}
    }
```

In today's life programming are widely used. One main part of programming is object oriented programming. Some of the OOP languages are not used know, but some of them are really popular and also widely used in the world. Java, Swift, Ruby, Python and etc. Are the most popular object oriented languages of present time. In below some information are given about these modern object oriented programming languages:

JAVA PROGRAMMING LANGUAGE

Java is a general-purpose computer-programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture. As of 2016, Java is one of the most popular programming languages in use, particularly for client-server web applications, with a reported 9 million developers.

James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. Java was originally designed for interactive television, but it was too advanced for the digital cable television industry at the time. The language was initially called Oak after an oak tree that stood outside Gosling's office. Later the project went by the name *Green*land was finally renamed *Java*, from Java coffee. Java was originally developed by James

Gosling at Sun Microsystems (which has since been acquired by Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform

The original and reference implementation Java compilers, virtual machines, and class libraraies were originally released by Sun under proprietary licenses. As of May 2007, in compliance with the specifications of the Java Community Process, Sun relicensed most of its Java technologies under the GNU General Public License. Others have also developed alternative implementations of these Sun technologies, such as the GNU Compiler for Java (bytecode compiler), GNU Class path (standard libraries), and IcedTea-Web (browser plugin for applets).

The latest version is Java 11, released on September 25, 2018, which follows Java 10 after only six months in line with the new release schedule. Java 8 is still supported but there will be no more security updates for Java 9. Versions earlier than Java 8 are supported by companies on a commercial basis; e.g. by Oracle back to Java 6 as of October 2017 (while they still "highly recommend that you uninstall" pre-Java 8 from at least Windows computers).

Principles

There were five primary goals in the creation of the Java language:

1. It must be "simple, object-oriented, and familiar".
2. It must be "robust and secure".
3. It must be "architecture-neutral and portable".
4. It must execute with "high performance".
5. It must be "interpreted, threaded, and dynamic".

Versions

As of 20 March 2018, both Java 8 and 11 are officially supported. Major release versions of Java, along with their release dates:

- JDK 1.0 (January 23, 1996)
- JDK 1.1 (February 19, 1997)
- J2SE 1.2 (December 8, 1998)
- J2SE 1.3 (May 8, 2000)
- J2SE 1.4 (February 6, 2002)
- J2SE 5.0 (September 30, 2004)
- Java SE 6 (December 11, 2006)
- Java SE 7 (July 28, 2011)
- Java SE 8 (March 18, 2014)
- Java SE 9 (September 21, 2017)
- Java SE 10 (March 20, 2018)
- Java SE 11 (September 25, 2018)

Syntaxis

The syntax of Java is largely influenced by C++. Unlike C++, which combines the syntax for structured, generic, and object-oriented programming, Java was built almost exclusively as an object-oriented language. All code is written inside classes, and every data item is an object, with the exception of the primitive data types, (*i.e.* Integers, floating-point numbers, Boolean values, and characters), which are not objects for performance reasons. Java reuses some popular aspects of C++ (such as the `printf` method).

Unlike C++, Java does not support operator overloading or multiple inheritance for *classes*, though multiple inheritance is supported for interfaces.

Java uses comments similar to those of C++. There are three different styles of comments: a single line style marked with two slashes (`//`), a multiple line style opened with `/*` and closed with `*/`, and the Javadoc commenting style opened with `/**` and closed with `*/`. The Javadoc style of commenting allows the user to run the Javadoc executable to create documentation for the program and can be read by some integrated development environments (IDEs) such as Eclipse to allow developers to access documentation within the IDE.

The traditional "Hello, world!" program can be written in Java as:

```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Prints the string to the console.
    }
}
```

PYTHON PROGRAMMING LANGUAGE

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. In July 2018, Van Rossum stepped down as the leader in the language community.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library. Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of Python's other implementations. Python and CPython are managed by the non-profit Python Software Foundation.

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL), capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum's long influence on Python is reflected in the title given to him by the Python community: Benevolent Dictator For Life (BDFL) – a post from which he gave himself permanent vacation on July 12, 2018.

Python is the reference implementation of Python. It is written in C, meeting the C89 standard with several select C99 features. It compiles Python programs into an intermediate bytecode which is then executed by its virtual machine. Python is distributed with a large standard library written in a mixture of C and native

Python. It is available for many platforms, including Windows and most modern Unix-like systems. Platform portability was one of its earliest priorities.

Versions

Python 2.0 was released on 16 October 2000 with many major new features, including a cycle-detecting garbage collector and support for Unicode.

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible. Many of its major features were backported to Python 2.6.x and 2.7.x version series. Releases of Python 3 include the 2to3 utility, which automates (at least partially) the translation of Python 2 code to Python 3.

Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3. In January 2017, Google announced work on a Python 2.7 to Go transcompiler to improve performance under concurrent workloads.

Features, philosophy and syntaxis

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming. Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

- Uses an elegant syntax, making the programs you write easier to read.
- Is an easy-to-use language that makes it simple to get your program working. This makes Python ideal for prototype development and other ad-hoc programming tasks, without compromising maintainability.
- Comes with a large standard library that supports many common programming tasks such as connecting to web servers, searching text with regular expressions, reading and modifying files.
- Python's interactive mode makes it easy to test short snippets of code. There's also a bundled development environment called IDLE.
- Is easily extended by adding new modules implemented in a compiled language such as C or C++.
- Can also be embedded into an application to provide a programmable interface.
- Runs anywhere, including Mac OS X, Windows, Linux and Unix, with unofficial builds also available for Android and iOS.
- Is free software in two senses. It doesn't cost anything to download or use Python, or to include it in your application. Python can also be freely modified and re-distributed, because while the language is copyrighted it's available under an open source license.

While offering choice in coding methodology, the Python philosophy rejects exuberant syntax (such as that of Perl) in favor of a simpler, less-cluttered grammar. As Alex Martelli put it: "To describe something as 'clever' is not considered a compliment in the Python culture." Python's philosophy rejects the Perl "there is more than one way to do it" approach to language design in favor of "there should be one—and preferably only one—obvious way to do it".

A common neologism in the Python community is *pythonic*, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called *unpythonic*.

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

The traditional “Hello, world!” program can be written in Python as:

```
# This program prints Hello, world!

print('Hello, world!')
```

Here is another example that written in Python:

```
class Parrot:

    # instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # instance method
    def sing(self, song):
        return “{} sings {}”.format(self.name, song)

    def dance(self):
        return “{} is now dancing”.format(self.name)

# instantiate the object
blu = Parrot("Blu", 10)

# call our instance methods
print(blu.sing("'Happy'"))
print(blu.dance())
```

Sources and sites

https://en.wikipedia.org/
https://www.upwork.com/
https://www.javatpoint.com/java-oops-concepts
https://javatutorial.net/
https://realpython.com/