FEATURES · 8 minute read

Top Features of OOPS

December 3, 2021

Table Of Contents

Introduction

Object-oriented programming (OOP) is a basic programming paradigm that almost every developer has utilized at some time in their career. Object-oriented programming (OOP) is the most prevalent programming paradigm.

Today, we'll go through the fundamentals and features of OOPS so you can start using it in your projects.

What is OOPS?

Object-Oriented Programming (OOP) is a programming model that uses classes and objects. It's utilized to break down a software program into reusable code blueprints (called classes) that you may use to build specific instances of things. Object-oriented programming languages include JavaScript, C++, Java, and Python, to name a few.

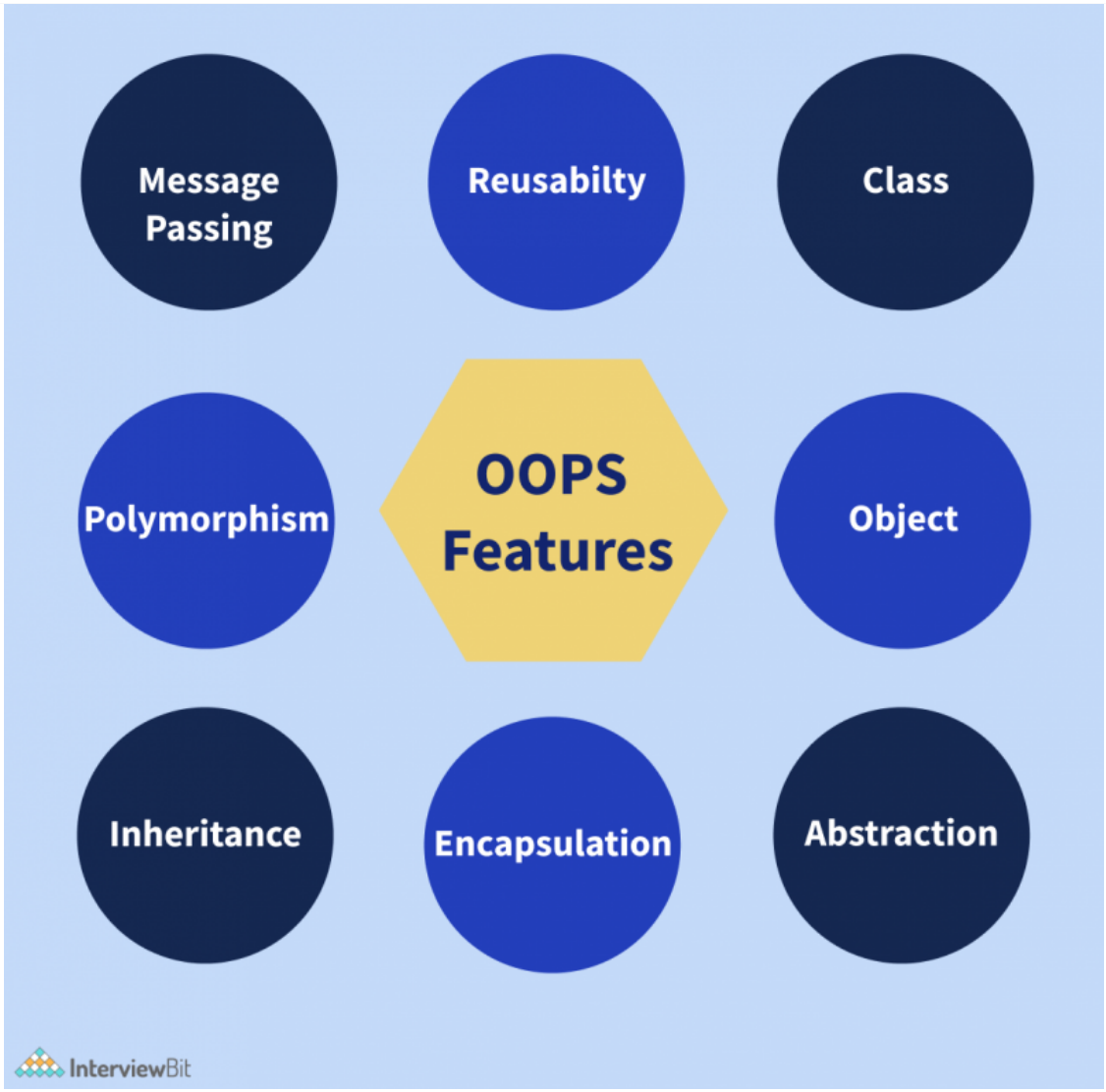
Expand in New Tab

Individual objects are created using class templates as a blueprint. For example, MyCar and goldenRetriever are two particular instances of the abstract class. The attributes specified in the class may have unique values for each object.

A class is a generic template that you may use to create more specialized, concrete things. Classes are commonly used to indicate large groupings with similar characteristics. Classes may also have functions known as methods that are exclusively accessible to objects of that kind. These functions are specified inside the class and execute an action beneficial to that particular object type.

Top Features of OOPS

Here are the top features of OOPS



Inheritance

In layman's terms, the attributes that you inherit from your parents are a simple illustration of inheritance. Classes may inherit characteristics from other classes thanks to inheritance. Parent classes, in other words, extend properties and behaviors to child classes. Reusability is aided via inheritance. Prototyping is another name for inheritance in JavaScript. A prototype object serves as a base from which another object may derive its features and actions. Thus, you may use multiple prototype object templates to form a prototype chain. Inheritance is passed down from one generation to the next. parent

Consider the application Polygon, which represents several Shapes. We're expected to make two distinct sorts of polygons: a Rectangle and a Triangle.

Encapsulation

Encapsulation is the process of enclosing all critical information inside an object and only revealing a subset of it to the outside world. For example, code inside the class template defines attributes and behaviors.

The data and methods are then enclosed in the object when it is created from the class. Inside a class, encapsulation conceals the underlying software code implementation and the internal data of the objects. Encapsulation necessitates designating certain fields as private while others are made public.

- Methods and attributes only available from other methods in the same class make up the private/internal interface.
- Methods and attributes that are available from outside the class are known as the public / external interface.

Encapsulation Demonstration in Real-Time

One of the most practical examples of encapsulation is a school bag. Our books, pencils, and other items may be kept in our school bag.

The following are some of the advantages of encapsulation:

- **Data Hiding:** In this case, the user will be unaware of the class's internal implementation. Even the user will have no idea how the class stores data in variables. He or she will only be aware that the values are sent to a setter method and that variables are initialised with that value.
- **Increased Flexibility:** Depending on our needs, we may make the variables of the class read-only or write-only. If you want to make the variables read-only, remove the setter methods like setName(), setAge(), and so on from the above programme. If you want to make the variables write-only, remove the get methods like getName(), getAge(), and so on from the above programme.
- It also promotes **reusability** and makes it simple to alter to meet new needs.

Abstraction refers to the user's interaction with just a subset of an object's characteristics and operations. To access a complicated item, abstraction uses simpler, high-level techniques.

- Simple items are used to show complexity.
- Keep complicated information hidden from the user.

Simple classes are used to indicate complexity in abstraction. Encapsulation is an extension of abstraction.

A Real-Life Example of Abstraction

Abstraction reveals just the most significant facts to the user while hiding the underlying intricacies. For example, when we ride a bike, we only know how to ride it but not how it works. We also have no idea how a bike works on the inside.

Advantages of Abstraction

- It simplifies the process of seeing things in their entirety.
- Code duplication is avoided, and reusability is increased.
- Because just the most necessary information is shown to the user, it helps to enhance the security of an application or software.

Polymorphism

Polymorphism refers to the creation of items that have similar behavior. For example, objects may override common parent behaviors with particular child behaviors through inheritance. Method overriding and method overloading are two ways that polymorphism enables the same method to perform various actions.

Examine how Polymorphism and the actual world are interconnected with examples.

Take, for example, your mobile phone. It has the capability of storing your Contacts. Consider the following scenario: you wish to store two numbers for one individual. You may do this by storing the second number under the same name as the first.

Consider the following scenario: you wish to store two numbers for the same individual in an object-oriented language such as Java. Create a function that will accept as arguments two integers and the name of the individual to some function void createContact that will be defined later (String name, int number1, int number2).

Method Overriding

Method overriding is used in runtime polymorphism. When a child class overrides a parent class's method, the child class might offer an alternative implementation.

Consider a family of three, consisting of the father, mother, and son. The father makes the decision to teach his kid to shoot. As a result, he brings him to the range with his favorite rifle and teaches him how to aim and fire at targets. The father, on the other hand, is right-handed, while the kid is left-handed. So they each have their own way of handling the pistol! Because of their differing orientations, the father was concerned that he may not be able to teach his son how to shoot.

The son, on the other hand, was astute and chose to flip his father's hands, putting his dominant hand on the trigger rather than the father's. Specifically, the right hand. By significantly changing the learning process, the son was able to grasp the skill of shooting!

Method overriding is the term used in programming to describe this idea.

Method Overloading

Method overloading is used in Compile Time Polymorphism. Although two methods or functions may have the same name, the number of arguments given into the method call may vary. Therefore, depending on the number of parameters entered, you may obtain different results.

With the help of a simple example, it may be comprehended in simple words. A class addition contains two add() methods, one with arguments int a and int b and the other with three integer parameters, int a, int b, and int c. As a result, the add() function is considered overloaded.

The amount of arguments given in the method calling statement determines which method is performed. For example, add(20,30) calls the two-parameter add() function, whereas add(10,20,30) calls the three-parameter add method

Objects

An object is a self-contained segment with the attributes and processes needed to make data usable in programming terms. From an object-oriented perspective, objects are the main building pieces of programs. In each application you create, you may employ a variety of objects of various sorts. Each kind of object is derived from a specific class of that type. Consider an object to be a sculpt of the real-world perceptions, processes, or objects that are important to the application you're designing.

A variable, function, or data structure may all be considered an object. The term "object" in object-oriented programming refers to a specific instance of a class. Objects are used in software development to combine data components with methods that alter them, allowing for the usage of abstract data structures. Objects in object-oriented programming are answers to the idea of inheritance, resulting in improved program dependability, simpler software maintenance, library administration, and task division in programmer teams. Of basic terms, "Objects" are the fundamental data types in object-oriented programming languages and are used to build object-oriented programming.

In the oops concept, a class is a construct that is used to describe an individual type. The class is instantiated into instances of itself – referred to as class instances or simply objects. A class defines ingredient members that allow its instances to have position and behavior. Member variables or instance variables facilitate a class instance to maintain its position. On the other hand, other kinds of members, especially methods, allow the behavior of class instances. Simply classes consequently define the type of their instances. A class usually represents a person, place or thing, or something.

For example, a “Bird” class would symbolize the properties and functionality of birds. A single, particular bird would be an instance of the “Bird” class, an object of the type “Bird”. There is a set of access specifiers in classes. private (or class-private) specifiers restrict the entrance to the class itself. Only the methods that are elements of a similar class only can access private members. protected (or class-protected) specifies enables the class itself and all classes under it (sub-classes) to access the member and public means that member can be accessed by its name using any code.

Constructors and Destructors

Constructors in most object-oriented languages have the same name as the class and are public. Constructors may be overloaded, which means that multiple argument lists can be used with the same name. The function Object() { [native code] } in PHP 5.0 is the function _construct (). Normally, attribute values would be initialised in a function Object() { [native code] }. The _destruct() method is optional, although it might be used to implement code that cleans up once an object is destroyed, such as shutting files or database connections.

OOP Advantages

- Complex things are modeled as repeatable, basic structures in OOP.
- Thus, OOP objects are reusable and may be utilized in several applications.
- Modularity for easier troubleshooting.
- Classes are easier to debug since they generally include all relevant information.

Conclusion

Object-oriented programming necessitates planning and thinking about the program's structure before starting to code and examining how to decompose the requirements into basic, reusable classes that you may utilize to create object instances. Overall, using OOP provides for more reusable data structures and saves time in the long run.

FAQs

Q1: Which are the best features of OOPs, and why explain?
Ans: Encapsulation, inheritance, and polymorphism are three fundamental elements of object-oriented programming that distinguish it from non-OOP languages. **Encapsulation** is the process of creating self-contained modules that connect processing processes to data. Classes are organized into hierarchies, and **inheritance** enables one class's structure and functions to be transferred down the hierarchy. Finally, object-oriented programming enables the creation of procedures regarding objects whose precise type is unknown until runtime is known as **polymorphism.**

Q2: What is an object in OOPs?
Ans: A class instance is referred to as an object. A real-world object is something like a pen, a laptop, a phone, a bed, a keyboard, a mouse, or a chair. A physical entity is referred to as an object. In Java, there are many methods for creating objects, including the new keyword, the newInstance() method, the clone() method, the factory method, and deserialization. Here is the real-life example of an object in OOPS is: Class: Human, Object: Man, Woman

Class: Fruit, Object: Apple, Banana, Mango, Guava, etc.

Q3: What are the basic principles of OOPs?
Ans: Abstraction, Encapsulation, Inheritance, and Polymorphism are the four core ideas of OOP. There are also classes and objects.

Additional Resources

- OOPs Interview Questions

Features of OOPS | OOPS

PREVIOUS POST

COMPARE

Difference Between HTML and JavaScript

August 23, 2022

NEXT POST

COMPARE

Difference Between Product and Service Based Company

August 25, 2022

Categories

- Applications
- Architecture
- Books
- Careers
- Characteristics
- Coding Problems
- Commands
- Compare
- Components
- Courses
- Features
- Frameworks
- IDE
- Interview Question
- IT Companies
- Job Roles
- Libraries
- Methodologies
- Model
- Principles
- Projects
- Resume
- Salaries
- Skills
- Technologies

Types

FEATURES · 8 minute read

Top Java 8 Features With Examples

August 10, 2022

Table Of Contents

On March 18, 2014, Java 8 was released. Even though it was a long time ago, Java 8 is still used in many applications because Java 8 got a significant update

What is Java 8?

Oracle’s Java 8 release was a watershed moment in the history of the world’s most popular development platform. In addition to a significant improvement to the Java programming paradigm as a whole, the JVM, Java language, and libraries were all updated in a coordinated manner. Many new features were included in this update, including enhanced simplicity of use, increased productivity and security, and overall better system performance.

Top Java 8 Features With Examples

The following Java 8 features will be discussed briefly, with each being explained via the use of basic and simple examples.

Q1 :

Functional Interfaces And Lambda Expressions

In Java 8, a new notion called functional interfaces was introduced. A Functional Interface is an interface that has exactly one abstract method. To designate an interface as a Functional Interface, we don’t need to use the @FunctionalInterface annotation.

The @FunctionalInterface annotation prevents abstract methods from being accidentally added to functional interfaces. It’s similar to a @Override annotation

and it's recommended that you use it. java.lang. Runnable is a fantastic example of a functional interface since it has one abstract method, run ().

One of the most appealing features of the functional interface is creating objects using lambda expressions. We can create an interface using an anonymous class, but the code is cumbersome. For example:

```
@FunctionalInterface
public interface FunctionalInterface_one
{
    public void firstInt_method();

    @Override
    public String toString(); //Overridden from Object class

    @Override
    public boolean equals(Object obj); //Overridden from Object clas
}
```

An anonymous function may be defined as a Lambda Expression (or function) (a function with no name and an identifier). Lambda Expressions are defined precisely where they are required, often as a parameter to another function.

Lambda Expressions, on the other hand, express instances of Functional Interfaces from a different viewpoint. Lambda Expressions implement functional interfaces by implementing the single abstract function provided in the functional interface.

A basic example of the Lambda Expression is:
(x,y) → x+y

The formula above accepts two parameters, x, and y, and returns their total, x+y. The procedure may be used numerous times in different locations depending on the data type of x and y. As a result, the arguments x and y will match int or

(if the parameters are int) or concatenate the two strings (when parameters are a string).

Let's implement a program that demonstrates Lambda Expressions.

```
interface MyInterface
{
     void abstract_func(int x,int y);

     default void default_Fun()
    {
         System.out.println("This is default method");
    }
}

class Main
{
     public static void main(String args[])
    {
        //lambda expression
        MyInterface fobj = (int x, int y)->System.out.println(x+y);

        System.out.print("The result = ");
        fobj.abstract_func(5,5);
        fobj.default_Fun();
    }
}
```

forEach() Method In Iterable Interface

In Java 8, the Java.lang interface now supports a "forEach" function. Iterable that can iterate over the collection's items. The Iterable interface has a default method called "forEach." Collection classes use it to iterate items, which extends

the Iterable interface.You may send Lambda Expression as an argument to the “forEach” method, which accepts the Functional Interface as a single parameter.

```
importjava.util.ArrayList;
importjava.util.List;
public class Main {
     public static void main(String[] args) {
        List<String> subList = new ArrayList<String>();
        subList.add("Carrot");
        subList.add("Potato");
        subList.add("Cauliflower");
        subList.add("LadyFinger");
        subList.add("Tomato");
        System.out.println("------------Vegetable List-------------
        subList.forEach(sub -> System.out.println(sub));
  }
}
```

Output:

————Vegetable List————
Carrot
Potato
Cauliflower
LadyFinger
Tomato

Optional Class

In Java 8, the “java.util” package included an optional class. The public final class “Optional” is used to handle NullPointerException in a Java program. You may give other code or values to execute using Optional. Thus, optional reduces the number of null checks required to avoid a nullPointerException.

You may use the Optional class to prevent the application from crashing and terminating unexpectedly. The Optional class has methods for checking the existence of a value for a given variable.

The following program demonstrates the use of the Optional class.

```
import java.util.Optional;
public class Main{

   public static void main(String[] args) {
        String[] str = new String[10];
        Optional<String>checkNull =
                       Optional.ofNullable(str[5]);
        if (checkNull.isPresent()) {
            String word = str[5].toLowerCase();
            System.out.print(str);
         } else
           System.out.println("string is null");
    }
}
```

Output:
String is null

To verify whether the string is null in this application, we utilize the Optional class's "ofNullable" attribute. If it is, the user receives the relevant message.

Default And Static Methods In Interfaces

In Java 8, you may add non-abstract methods to interfaces, allowing you to create interfaces with method implementation. To construct interfaces with method implementation, use the Default and Static keywords. Lambda Expression functionality is mostly enabled through default approaches.

You may extend the functionality of your libraries' interfaces by using default methods. This ensures that the code created for previous versions is compatible with the newer interfaces (binary compatibility).

Let's understand the Default Method with an example:

```
import java.util.Optional;
interface interface_default {
     default void default_method(){
         System.out.println("We are default method of interface");
    }
}
class derived_class implements interface_default{

}
class Main{
   public static void main(String[] args){
        derived_class obj1 = new derived_class();
        obj1.default_method();
    }
}
```

Output:
We are the default method of interface

We have an interface called "interface default" with a default implementation of the function default method(). Next, we create a class called "derived class" that implements the "interface default" interface.

In this class, we haven't implemented any interface functions. Then, in the main function, we construct a "derived class" object and invoke the interface's "default method" without specifying it in the class.

The usage of default and static methods in the interface is an example of this. If

implementation.

Java Stream API For Bulk Data Operations On Collections

Another significant feature in Java 8 is the Stream API. The Stream API is used to handle a collection of items and allows many iterations. A Stream is a collection of items (elements) that enables you to combine multiple techniques to achieve your goals.

The java.util.stream package in Java 8 introduces a new Streams API that enables you to process components of Java Collections in parallel. Because Java is inherently sequential, and there is no direct way to implement parallel processing at the library level, the stream API will fill that void. You may use Java's Stream API to filter items of a collection based on a defined condition. If you have a list of orders, for example, you may combine purchase and sell orders, filter orders by amount and price, and so on.

Java Date Time API

Under the package java.time, Java 8 offers a new date-time API. The following are the most prominent classes among them:

- Local: Simplified date-time API with no timezone management complexity.
- Zoned: specialized date-time API that can handle several time zones.

For Example:

Class	Description
LocalDate	Represents a date (year, month, day (yyyy-MM-dd))

Class	Description
LocalDateTime	Represents both a date and a time (yyyy-MM-dd-HH-mm-ss-ns)
DateTimeFormatter	Formatter for displaying and parsing date-time objects

Dates

In Java 8, the Date class has been deprecated. The following are the new courses that have been introduced:

- A date is defined by the LocalDate class. It is devoid of any indication of time or time zone.
- A time is defined by the LocalTime class. It doesn't have a date or time-zone representation.
- A date-time is defined by the LocalDateTime class. It doesn't have a time-zone representation

To combine time-zone information with date functions, you may utilize Lambda's OffsetDate, OffsetTime, and OffsetDateTime classes. Another class – "ZoneId" – is used to express the timezone offset here.

Collection API Improvements

The Collection API in Java 8 now includes the following new methods. A few of them are listed below.

- This is a default method for the Iterator. forEachRemaining (Consumer action): This is a default method for the Iterator. It repeats the "action" for the remaining items until all of them have been processed, or the "action" throws an exception.
- The default technique for removing items from a collection is removed (Predicate filter). This removes all objects from the collection that satisfy the

- Spliterator () This collection method returns a spliterator object that may be used to traverse the items sequentially or in parallel.
- ReplaceAll (), calculate(), and merge() are methods in the Map collection.
- The performance of the HashMap class with Key collisions has been enhanced etc.

Java IO Improvements

The following are some of the IO enhancements made in Java 8:

- Files.list (Path dir): Returns a lazily filled stream, each element of which represents a directory entry.
- Files.lines (Path path): Reads all the lines from a stream.
- Files.find (): Returns a stream filled by a path after searching for files in the file tree rooted at a provided beginning file and many more.
- BufferedReader.lines (): Returns a stream containing all of the elements of BufferedReader's lines and much more
- Buffered

Miscellaneous Core API Improvements

- ThreadLocal's static function with initial (Supplier supplier) allows you to build an instance quickly.
- The default and static methods for natural ordering, reverse order, and other operations have been added to the "Comparator" interface.
- The min (), max (), and sum () methods are available in the Integer, Long, and Double wrapper classes.
- The logicalAnd (), logicalOr (), and logicalXor () methods have been added to the Boolean class.
- The Math class introduces a number of useful techniques.
- The JDBC-ODBC Bridge has been deactivated.
- The memory space used by PermGen is no longer available.

For Base64 encoding, Java 8 has built-in encode and decode functions. The Base64 encoding class in Java.util.Base64.

Three Base64 encoders and decoders are provided in this class:

- The output is mapped to a set of characters between A-Za-z0-9+/ in this version. The encoder does not add a line feed to the output, and the decoder rejects any character other than the above.
- The filename safe is mapped to the set of characters between A-Za-z0-9+/, and the output is the URL.
- The output of this sort of encoder is mapped to a MIME-friendly format.

Conclusion

As we can see, Java 8 offers a number of important features. Other programming languages, such as Scala, have inspired some of the features. Java must improve in order to avoid becoming outdated, and although certain features are rudimentary in comparison to other programming languages.

FAQs

Q1: What are the characteristics of Java 8?
Ans: Following are the characteristics of Java 8:

- Lambda expressions
- Method references
- Functional interfaces
- Stream API
- Default methods
- Base64 Encode Decode
- Static methods in interface
- Optional class and many more

Q2: Why is Java 8 still popular?
Ans: The fact that Java 8 is an LTS (Long-Term Support) version is one of the main reasons for its continued popularity. Regrettably, not all Java versions are LTS versions! Only Java 8 (2014) and Java 11 (2018) have been recognized as having LTS since the policy was implemented. All subsequent versions, including Java 14 and the anticipated Java 15 (September 2020), will be LTS-free.

Q3: What changed in Java 8?
Ans: Some of the significant productivity enhancements are lambda expressions, the Streams API, and new methods on existing classes.

Additional Resources

- Learn Java
- Online Java Compiler
- Java MCQ
- Advance Java MCQ
- Practice Coding
- Java Interview Questions
- How To Become A Java Developer
- Best Java IDE
- Java Projects
- Java 9 Features
- Java 11 New Features
- Features of Java
- Java Frameworks
- Java Developer Salary
- Java Developer Skills

java | Java 8 | Java 8 Features

PREVIOUS POST

COMPARE

Stateful vs Stateless: Full Difference

February 23, 2022

NEXT POST

COMPARE

Gradle Vs Maven: What's The Difference? [2022]

August 19, 2022

Categories

Applications

Architecture

Books

Careers

Characteristics

Coding Problems

Commands

Compare

Components

Courses

Features

Frameworks

IDE

Interview Question

IT Companies

Job Roles

Libraries

Methodologies

Model

Principles

Projects

Resume

Salaries

Skills

Technologies

Types