## Object Oriented Programming with Java

It's a modern software development methodology supported by most programming languages. We should always use it whenever we create a simple utility or a complex database system. The OOP is not just a technology or a recommended program structure, it's mainly a new way of thinking. A new perspective from which we could analyze problems and a new era in software development.

Between structured programming and the object-oriented programming, there was one more intermediate approach called modular programming. It involved the encapsulation of specific functionality into modules

OOP is a philosophy and a way of thinking, designing and implementing solutions that focus on **reusability**. This approach is inspired by the industrial revolution - the invention of basic components. Making a "component program" is smarter and cheaper. Components do not fail, if there's a problem, it is most likely in the code you have written **in one specific section**. OOP consists of four different concepts:

**Abstraction.** Abstraction means using simple things to represent complexity. In Java, abstraction means simple things like objects, classes, and variables represent more complex underlying code and data. This is important because it lets avoid repeating the same work multiple times.

**Encapsulation.** This is the practice of keeping fields within a class private, then providing access to them via public methods. It’s a protective barrier that keeps the data and code safe within the class itself. This way, we can re-use objects like code components or variables without allowing open access to the data system-wide.

**Inheritance.** This is a special feature of Object Oriented Programming in Java. It lets programmers create new classes that share some of the attributes of existing classes. This lets us build on previous work without reinventing the wheel.

**Polymorphism.** This Java OOP concept lets programmers use the same word to mean different things in different contexts. One form of polymorphism in Java is methodoverloading. That’s when different meanings are implied by the code itself. The other form is methodoverriding. That’s when the different meanings are implied by the values of the supplied variables.

1. **How OOP Concepts in Java Work**

OOP, concepts in Java work by letting programmers create components that can be re-used in different ways, but still maintain security.

**Abstraction** as an OOP concept in Java works by letting programmers create useful, reusable tools. For example, a programmer can create several different types of objects. These can be variables, functions, or data structures. Programmers can also create different classes of objects. These are ways to define the objects.

**Encapsulation** lets us re-use functionality without jeopardizing security. It’s a powerful OOP concept in Java because it helps us save a lot of time. It may be useful to reuse that code with other databases or processes. Encapsulation lets us do that while keeping our original data private. It also lets us alter our original code without breaking it for others who have adopted it in the meantime.

**Inheritance** is another labor-saving Java OOP concept. It works by letting a new class adopt the properties of another. We call the inheriting class a subclass or a childclass. The original class is often called the parent. We use the keyword extends to define a new class that inherits properties from an old class.

**Polymorphism** in Java works by using a reference to a parent class to affect an object in the child class. We might create a class called “horse” by extending the “animal” class. That class might also implement the “professional racing” class. The “horse” class is “polymorphic,” since it inherits attributes of both the “animal” and “professional racing” class.

Two more examples of polymorphism in Java are method overriding and method overloading. In method overriding, the child class can use the OOP polymorphism concept to override a method of its parent class. That allows a programmer to use one method in different ways depending on whether it’s invoked by an object of the parent class or an object of the child class. In methodoverloading**,** a single method may perform different functions depending on the context in which it’s called. That is, a single method name might work in different ways depending on what arguments are passed to it.

## Object Oriented Programming with Java

Java is known as an Object-Oriented language. It means that the **foundations** of any kind of **program constructed in Java** might be imagined in terms of **Objects**. Imagine that we are tasked with constructing a software program intended to keep track of an actual **public library system**. This system must keep track of each of the **branches** associated with the libraries, the whole set of **materials** that can be contained in the branches, and additionally all of the **people** that may need to access **books** from the library’s branch.

**All these words represent Objects in Java**. This really is, in essence, Object Oriented programming (generally known as O-O programming). In a programming language, encapsulation is used to refer to one of two related but distinct notions, and sometimes to the combination thereof:

* A language mechanism for restricting access to some of the object’s components.
* A language construct that facilitates the bundling of data with the methods (or other functions) operating on that data.

## Object Oriented Programming with Java

Object-oriented programming is the successor of procedural (structural) programming. Proceduralprogramming describes programs as groups of reusable code units (procedures) which define input and output parameters. Procedural programs consist of procedures, which invoke each other.

The problem with procedural programming is that codereusabilityishard and limited – only procedures can be reused and it is hard to make them generic and flexible. There is no easy way to work with abstract data structures with different implementations.

The object-oriented approach relies on the paradigm that each and every program works with data that describes entities (objects or events) from real life. This is how objects came to be. The main advantages and goals of OOP are to make complex software faster to develop and easier to maintain. OOP enables the easy reuse of code by applying simple and widely accepted rules (principles).

In order for a programming language to be object-oriented, it has to enable working with classes and objects as well as the implementation and use of the fundamental object-oriented principles and concepts: inheritance, abstraction, encapsulation and polymorphism.

# **Understanding Java’s Object-Oriented Programming (OOP)**

Java is object-oriented - which focus on giving the computer imperative “Do this/Do that” commands. Object-oriented programs tell the computer what to do. They start, however, by organizing the data, and the commands come later.

Object-oriented languages are better than “Do this/Do that” languages because they organize data in a way that lets people do all kinds of things with it. To modify the data, you can build on what you already have, rather than scrap everything you’ve done and start over each time you need to do something new. In an object-oriented language, you use objects *and* classes to organize your data.

**Research and write about the differences between checked and unchecked exceptions**

**1) Checked:** are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using *throws* keyword. Checked Exception is required to be handled by compile time while Unchecked Exception doesn't. Checked Exception is direct sub-Class of Exception while Unchecked Exception are of Runtime Exception. Checked Exception represent scenario with higher failure rate while Unchecked Exception are mostly programming mistakes. Checked exceptions denote error scenarios which are outside the immediate control of the program. They occur usually interacting with outside resources/ network resources e.g. database problems, network connection errors, missing files etc.

**2) Unchecked** are the exceptions that are not checked at compiled time. In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to be civilized and specify or catch the exceptions.  
In Java exceptions under Error and Runtime Exception classes are unchecked exceptions, everything else under throwable is checked. Java also provides Unchecked Exceptions, the **occurrences of which are not checked by the compiler**. They will come into life/occur into your program, once any buggy code is executed. Unchecked exceptions are usually **result of bad code in your system**. A method is not forced by compiler to declare the unchecked exceptions thrown by its implementation and methods almost always do not declare them.

**References:**

<https://howtoprogramwithjava.com/object-oriented-programming-java-tutorial/>

<https://stackify.com/oops-concepts-in-java/>

<https://www.ict.social/java/oop/introduction-to-object-oriented-programming-in-java>

<http://www.introprogramming.info/english-intro-csharp-book/read-online/chapter-20-object-oriented-programming-principles/>

<https://www.dummies.com/programming/java/understanding-javas-object-oriented-programming-oop/>

<https://www.geeksforgeeks.org/checked-vs-unchecked-exceptions-in-java/>

<https://javarevisited.blogspot.com/2011/12/checked-vs-unchecked-exception-in-java.html>

<https://howtodoinjava.com/core-java/exception-handling/checked-vs-unchecked-exceptions-in-java/>