**4 FUNDAMENTAL CONCEPTS (PILLARS) OF OBJECT-ORIENTED PROGRAMMING**

**POLYMORPHISM**

- poly: many ; morph: change or form

- polymorphism can be achieved thru Overriding and Overloading.

- (**Overloading**) ability of a function/ method to take on multiple forms. Meaning, you can create multiple functions with same name but each having different parameters and might provide diff. functionality.

- (**Overriding**) when a method in Parent Class is redefined by one of its Child classes, thus modifying the behaviour of that method

**Non-technical explanation (real world example):**

For example an Animal, a Dog is a parent of Puppy, thus all the characteristics and behaviours of the Dog can or will be acquired by Puppy. But then, a puppy is too young therefore it cannot eat the food a Dog can eat. So there will be another way a puppy eats its food or let’s say another food to eat but still its behaviour of a Dog (overriding).

(for overloading) when a dog walks, it may vary depending on the hours or the distance it will walk. so therefore we can say that there are 2 kinds of walk a dog has , thru number of hours and by distance.

**Code example:**

\*example of overloading

**public** **class** Dog {

**public** **void** doEat(String foodName) {}

**public** **void** doEat(Integer foodQuantity) {}

}

\* example of overriding

**public** **class** Puppy **extends** Dog {

@Override

**public** **void** doEat(String foodName) {}

@Override

**public** **void** doEat(Integer foodQuantity) {}

}

**INHERITANCE**

- is a concept wherein a class acquires some/all properties and methods of another class. In other words, the subclass (child) uses the states and behaviours of the super class (parent)

**Non-technical explanation (real world example):**

● For example for this is a Father and its child. A Father can have characteristics such as being short, having a dark skin, and big eyes. These characteristics might be then acquired by its child. Same applies with Fathers behaviour like knowing how to dance or sing.

● Another example, in a software company Software Engineers, Sr. Software Engineers, Module Lead, Technical Lead, Project Lead, Project Manager, Program Manager, Directors all are the employees of the company but their work, perks, roles, responsibilities differs. So in OOP, the Employee base class would provide the common behaviours of all types/level of employee and also some behaviours properties that all employee must have for that company. The particular sub class or child class of the employee would implement behaviours specific to that level of the employee. So by above example you can notice that the main concept behind inheritance are extensibility and code reuse (in this case you are extending the Employee class and using its code into sub class or derived class).

**Code example:**

**public** **class** Animal {

**public** **void** sleep() {}

**public** **void** eat() {}

}

**public** **class** Dog **extends** Animal {

**public** **void** bark() {}

}

**public** **class** InheritanceExample {

**public** **static** **void** main(String[] args) {

Dog d = **new** Dog();

// methods inherit from Animal (Parent class)

d.eat();

d.sleep();

// method from Dog (Current / Child Class)

d.bark();

}

}

**ENCAPSULATION**

- is the principle of information/ data hiding. But actually we didn’t purposely hide the stuff to prevent from seeing it because that’s a different thing, but instead we wrapped up all the data members, internal structure of an objects as well as the implementation of its methods into a single unit.

- answers: how does object do it?

- why use encapsulation:

1. to provide a simplified and understandable way to use your object without the need to understand the complexity inside.

2. to easily manage change.

3. The main benefit of encapsulation is the ability to modify our implemented code without breaking the code of others who use our code

**Non-technical explanation (real world example):**

A car is having multiple parts like steering wheel, gear stick, engine, etc., which binds together to form a single object that is car. So, here multiple parts of cars encapsulates itself together to form a single object that is Car.

**Code example:**

**public** **class** Ship {

**public** **void** move() {

// Code that moves the ship

}

**public** **void** fire() {

// Code that makes the ship fire a bullet

}

}

\* To have the ship fire a bullet, all you would have to do is call **ship.fire().** How the code implements firing a bullet is not important, because all we care about is firing a bullet. This way, if you want to change the code to fire a laser blast instead, you just have to change the method **fire()** and not every call to **ship.fire().**

**ABSTRACTION**

- focused on the representation of essential information ignoring all other details. In simpler words, showing only the “relevant” data and “hide” unnecessary irrelevant and complex details of an object. So you are aware only of what the application does, not how it does it.

- Abstraction provides the interface of applications, like the tasks the program is capable to execute and their respective input and output data. Then, Encapsulation is the one who hides the details of that implementation from end user and treats them as a secret.

-answers: what does the object do?

**Non-technical explanation (real world example):**

● Remote is an interface between user and TV, which has buttons like 0 to 10, on/ of, etc., but we don’t know circuits inside remote. User does not need to know how it works internally due to its complexity. Just he is using essential thing that is remote.

● A Laptop consists of many things such as processor, motherboard, RAM, keyboard, LCD screen, wireless antenna, web camera, USB ports, battery, speakers etc. To use it, you don't need to know how internally LCD screens, keyboard, web camera, battery, wireless antenna, speakers works. You just need to know how to operate the laptop by switching it on. The intrinsic details are invisible. So here the Laptop is an object that is designed to hide its complexity.

**Code example:**

● Abstraction can be implemented in two ways. Abstract Class and Interface

Parent Class = Super Class = Base Class = Ancestor Class

Child Class = Sub Class = Derived Class = Descendant Class

Attributes/States/Characteristics/Properties= Variable/Fields

Behaviour/Action = Method/Function

**Constructor**

Default Constructor – If you do not define any constructor in your class, java compiler will generate one by default. It is not visible in source code.

No-arg Constructor – Same with default constructor, however it is visible in the code and should have at least a body (code inside)

Parameterized Constructor – constructor with arguments

Note: If your parent class lacks a no-argument constructor, your subclass/ child class is required to override at least one of the parameterized constructors of the parent class, if there’s one.

**ACCESS MODIFIERS**

**Public** – can be accessed outside the class; if it is being accessed in a diff package, then you need to import it.

**Private** - only accessible within the class

**Protected** – only accessible within the package and a class that inherits it (subclasses)

**Default (no modifier)** - only accessible within the package

**NON ACCESS MODIFIERS**

**Abstract**

Abstract Class – class that was declared as abstract cannot be instantiated. It can only be extended

Abstract Method – when at least one of the methods is declared as abstract, class should be abstract as well. There is no default implementation for it, the implementing class (child class) is required to override/implement the abstract method and will provide the details.

**Final**

Final Class – class cannot be extended but can be instantiated.

Final Variable – value cannot be changed

Final Method – cannot be overridden

**Static**

- static keyword is used to create variable/method that is part of the class rather than the instances of the class.

- having a static variable can be used to refer to the common property of all objects (that is not unique for each object).

- static can be applied to:

* Variable/Class Variable
* Method/ Class Method
* Block
* Nested Class

**Synchronized**

Synchronized Method – can be accessed by only one thread at a time

**Native**

Native Method – is applied to a method to indicate that the method is implemented in native code using JNI(Java Native Interface). It marks a method, that it will be implemented in other languages, not in Java. It works together with JNI (Java Native Interface).

**Strictfp**

Strictfp Class/Method - is used to force the precision of floating point calculations (float or double) in Java conform to IEEE’s 754 standard, explicitly. Without using strictfp keyword, the floating point precision depends on target platform’s hardware, i.e. CPU’s floating point processing capability. In other words, using strictfp ensures result of floating point computations is always same on all platforms.

**Transient**

Transient Variable – if instance variable is declared as transient, then its value will not be serialized

**Volatile**

Volatile Variable - tells the compiler that the volatile variable can be changed unexpectedly by other parts of your program. Volatile variables are used in case of multithreading program

**SUPER and THIS keywords**

super – access the members of the parent class

this – used to access the members of the child/ current class ; this cannot be used in static context

**EXTENDS and IMPLEMENTS keywords**

* implements means that you're using the elements of a Java Interface in your class, and extends means that you are creating a subclass of the class you are extending.
* For a class, you can only extend one class, but can implement multiple interfaces
* For an interface, you can’t use implements keyword, but can extend multiple interfaces

**Accessor (Getter) and Mutator (Setter)**

- purpose of these two is mainly for returning and setting the values in an object

Accessor - is used to return the value of a private field. It follows a naming scheme prefixing the word "get" to the start of the method name.

Mutator - is used to set a value of a private field. It follows a naming scheme prefixing the word "set" to the start of the method name.

**ABSTRACT CLASS and INTERFACE**

* All methods in an interface are implicitly abstract. On the other hand, an abstract class may contain both abstract and non-abstract methods.
* In Abstract class, modifiers you can use for abstract methods are public, protected, and default. While in Interface, you can only use public and default.
* When a subclass is declared as abstract it does not required to override/implement the abstract methods of both abstract class and interface. Opposite happens when a subclass is non-abstract
* An interface is absolutely abstract and cannot be instantiated. An abstract class also cannot be instantiated, but can be invoked if it contains a main method.
* In class implementing interface, if the interface extends another interfaces, the subclass will be able to override also the abstract methods of the extended interfaces.

**INTEFACE**

They are used to design the requirements for a set of classes to implement. The interface sets everything up, and the class or classes that implement the interface do all the work.

**CONCRETE CLASS AND METHOD**

A concrete method have complete definition just like a normal method. And it can be overridden in the inherited classes if they are not final. The methods which are not abstract methods are all nothing but concrete methods in java. And Abstract methods are those which are declared by using keywords public & abstract. Abstract methods doesn’t contains body.

Concrete class in Java is the default class and is a derived class that provides the basic implementations for all of the methods that are not already implemented in the base class. The opposite of the concrete class is the abstract class, which does not provide implementations for all of its methods.

**VARIABLE TYPES**

Local Variable – are declared in methods, constructors, or blocks

Instance Variable – declared in a class, but outside a method, constructor, or any block

Class/ Static Variable – is like an instance variable but declared as static. There would only be one copy of each class variable per class, regardless of how many objects are created from it

**PROS AND CONS OF STATIC VARIABLE**

Variables that are declared static stays in the memory till the program executes thus taking extra space.

The use of static may be beneficial if you want to use/retain a value for a long period of time, however declaring all variables as static is not recommended and is not a good practice. If you make a habit of declaring all value as static your program will eat unnecessary memory.

Other than that, static variable doesn't comply with OOPS concept where scopes, abstraction and encapsulation are defined along with the effervescence object. Through which you can call and delete variables at will.

The biggest disadvantage of using static variables will appear if you are working in limited memory space (such as mobile applications) in that case you application will crash if it is overhogged by variables and less memory space.

If you want to store a value permanently there are other ways around like database, files etc makes the job easier and cleaner. Just my 2 cents.

|  |
| --- |
| **P**: Use static variables only when you want to share some value between multiple instances of the same class.  **C:** A static variable is declared for the class. If you need individual values per class instance, you can't use a static variable.  You do not know when a static variables is instantiated and whether or not it will be instantiated before another static variable  Static members are part of class and thus remain in memory till application terminates and can’t be ever garbage collected. |

**CLASS AND OBJECT**

- Class is a blueprint or template to build a specific type of object

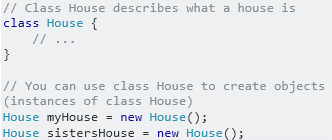
- Object is a self-contained component that contains properties and methods needed to make a certain type of data useful

|  |  |  |
| --- | --- | --- |
| No. | Object | Class |
| 1) | Object is an instance of a class. | Class is a blueprint or template from which objects are created. |
| 2) | Object is a real world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc. | Class is a group of similar objects. |
| 3) | Object is a physical entity. | Class is a logical entity. |
| 4) | Object is created through new keyword mainly e.g. Student s1=new Student(); | Class is declared using class keyword e.g. class Student{} |
| 5) | Object is created many times as per requirement. | Class is declared once. |
| 6) | Object allocates memory when it is created. | Class doesn't allocated memory when it is created. |
| 7) | There are many ways to create object in java such as new keyword, newInstance() method, clone() method, factory method and deserialization. | There is only one way to define class in java using class keyword. |

**Code example:**

A **class** is a blueprint which you use to create **objects**. An object is an **instance** of a class - it's a concrete 'thing' that you made using a specific class. So, 'object' and 'instance' are the same thing, but the word 'instance' indicates the relationship of an object to its class.

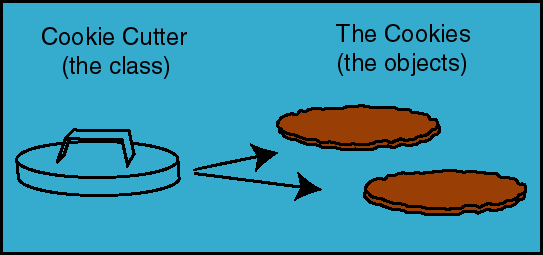
This is easy to understand if you look at an example. For example, suppose you have a class House. Your own house is an object and is an instance of class House. Your sister's house is another object (another instance of class House).



The class House describes the concept of what a house is, and there are specific, concrete houses which are objects and instances of class House.

Note: This is exactly the same in Java as in all object oriented programming languages.

**Non-technical explanation (real world example):**



● An architect will have the blueprints for a house....those blueprints will be plans that explain exactly what properties the house will have and how they are all laid out. However it is just the blueprint, you can't live in it. Builders will look at the blueprints and use those blueprints to make a physical house. They can use the same blueprint to make as many houses as they want....each house will have the same layout and properties. Each house can accommodate its own families...so one house might have the Smiths live in it, one house might have the Jones live in it.

The blueprint is the class...the house is the object. The people living in the house are data stored in the object's properties.

● Consider for a moment a cell phone:

1. When you look at a cell phone, you will see it has certain properties of attributes.

2. For example, it has buttons. Some have your basic "phone" keys (numbers, #, etc.). The fancier ones have complete keyboards.

3. Most have some sort of screen. Again, these range from basic LCD text screens to fancy touch screens.

4. Cell phones can also perform tasks. That is they have behaviors or methods that allow them to take some action(s) using the attributes.

5. For example, you can make and receive phone calls. On some, you can take and send pictures, etc.

6. So, a cell phone, as an object, is a collection of both its attributes and its methods.

**OOP**

- is a programming model where Programs are organized around object and data rather than action and logic

- is a programming paradigm that uses "objects" and their interactions to design applications and computer programs.

It is not just a programming language, but a paradigm (An example or model used to explain a concept or theory). OOP does not tell you how to program, rather it tells you how to go about designing your software. There are many languages that implement/help you in implementing OOP. Java and C++ are some of them. When you are developing Object-oriented program/software the emphasis is more on how you think about and design the software rather than on actual implementation (code) of it.

1. Object - instance of class

2. Class - blue print of Object

3. Polymorphism - different behaviours at diff. instances

4. Inheritance - one property of object is acquiring to another property of object

5. Encapsulation - protecting our data by hiding complexity

6. Abstraction - only expose what’s necessary

**EXPLAINING THE WHOLE OOP CONCEPTS IN ONE REAL WORLD EXAMPLE**

You have a clock, and it tells the time - well, in programming you put all the code and stuff you have to do all together (sounds pretty obvious, but people didn't used to do this way back in the early days). Anyway, that's called encapsulation.

Now you've got a clock thing, you might want an alarm clock - well, once you've got all the stuff together you can add things to it to make it do more - like set the alarm and make it ring. This is called inheritance.

Also, you look at the clock I have on my wrist, but you can other clocks that look different like a grandfather clock or a digital clock. It appears different, but it's still a clock - well, that's called polymorphism.

Mobile

<http://www.c-sharpcorner.com/UploadFile/cda5ba/object-oriented-programming-with-real-world-scenario/>

Piano

<http://www.c-sharpcorner.com/blogs/real-life-examples-of-object-oriented-programming1>

**ADVANTAGES:**

1. Improved software-development productivity: Object-oriented programming is modular, as it provides separation of duties in object-based program development. It is also extensible, as objects can be extended to include new attributes and behaviours. Objects can also be reused within an across applications. Because of these three factors – modularity, extensibility, and reusability – object-oriented programming provides improved software-development productivity over traditional procedure-based programming techniques.

2. Improved software maintainability: For the reasons mentioned above, object oriented software is also easier to maintain. Since the design is modular, part of the system can be updated in case of issues without a need to make large-scale changes.

3. Faster development: Reuse enables faster development. Object-oriented programming languages come with rich libraries of objects, and code developed during projects is also reusable in future projects.

4. Lower cost of development: The reuse of software also lowers the cost of development. Typically, more effort is put into the object-oriented analysis and design, which lowers the overall cost of development.

5. Higher-quality software: Faster development of software and lower cost of development allows more time and resources to be used in the verification of the software. Although quality is dependent upon the experience of the teams, object oriented programming tends to result in higher-quality software.

**DISADVANTAGES:**

1. Steep learning curve: The thought process involved in object-oriented programming may not be natural for some people, and it can take time to get used to it. It is complex to create programs based on interaction of objects. Some of the key programming techniques, such as inheritance and polymorphism, can be challenging to comprehend initially.

2. Larger program size: Object-oriented programs typically involve more lines of code than procedural programs.

3. Slower programs: Object-oriented programs are typically slower than procedure based programs, as they typically require more instructions to be executed.

4. Not suitable for all types of problems: There are problems that lend themselves well to functional-programming style, logic-programming style, or procedure-based programming style, and applying object-oriented programming in those situations will not result in efficient programs.

**Advantages of OOP**

1. **simplicity**: software objects model real world objects, so the complexity is reduced and the program structure is very clear;
2. **modularity**: each object forms a separate entity whose internal workings are decoupled from other parts of the system;
3. **modifiability**: it is easy to make minor changes in the data representation or the procedures in an OO program. Changes inside a class do not affect any other part of a program, since the only public interface that the external world has to a class is through the use of methods;
4. **extensibility**: adding new features or responding to changing operating environments can be solved by introducing a few new objects and modifying some existing ones;
5. **maintainability**: objects can be maintained separately, making locating and fixing problems easier;
6. **re-usability**: objects can be reused in different programs.

**DISAdvantages of OOP**

1. **Size:**  OO programs are much larger than other programs. In the early days of computing, space on hard drives, floppy drives and in memory was at a premium. Today we do not have these restrictions.
2. **Effort:** OO programs require a lot of work to create. For example, a great deal of planning goes into an object oriented program well before a single piece of code is ever written.
3. **Speed:** OO programs are slower than other programs, partially because of their size. Other aspects of OO Programs also demand more system resources, thus slowing the program down.

|  |  |  |
| --- | --- | --- |
|  | Procedure Oriented Programming | Object Oriented Programming |
| Divided Into | In POP, program is divided into small parts called **functions**. | In OOP, program is divided into parts called **objects**. |
| Importance | In POP, Importance is not given to **data** but to functions as well as **sequence** of actions to be done. | In OOP, Importance is given to the data rather than procedures or functions because it works as a **real world**. |
| Approach | POP follows**Top Down approach**. | OOP follows **Bottom Up approach**. |
| Access Specifiers | POP does not have any access specifier. | OOP has access specifiers named Public, Private, Protected, etc. |
| Data Moving | In POP, Data can move freely from function to function in the system. | In OOP, objects can move and communicate with each other through member functions. |
| Expansion | To add new data and function in POP is not so easy. | OOP provides an easy way to add new data and function. |
| Data Access | In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system. | In OOP, data cannot move easily from function to function, it can be kept public or private so we can control the access of data. |
| Data Hiding | POP does not have any proper way for hiding data so it is **less secure**. | OOP provides Data Hiding so provides **more security**. |
| Overloading | In POP, Overloading is not possible. | In OOP, overloading is possible in the form of Function Overloading and Operator Overloading. |
| Examples | Example of POP are: C, VB, FORTRAN, Pascal. | Example of OOP are: C++, JAVA, VB.NET, C#.NET. |

|  |  |  |
| --- | --- | --- |
|  | Structured Programming | Object Oriented Programming |
|  | Structured Programming is designed which focuses on **process**/ logical structure and then data required for that process. | Object Oriented Programming is designed which focuses on **data**. |
|  | Structured programming follows**top-down approach**. | Object oriented programming follows **bottom-up approach**. |
|  | Structured Programming is also known as **Modular Programming** and a subset of **procedural programming language**. | Object Oriented Programming supports **inheritance, encapsulation, abstraction**, **polymorphism**, etc. |
|  | In Structured Programming, Programs are divided into small self-contained **functions**. | In Object Oriented Programming, Programs are divided into small entities called **objects**. |
|  | Structured Programming is **less** secure as there is no way of **data hiding**. | Object Oriented Programming is more secure as having data hiding feature. |
|  | Structured Programming can solve **moderately** complex programs. | Object Oriented Programming can solve any **complex** programs. |
|  | Structured Programming provides **less** **reusability**, more function dependency. | Object Oriented Programming provides more reusability, less function **dependency**. |
|  | Less abstraction and less flexibility. | More abstraction and more **flexibility**. |

**EXPLANATION #1**

Structured Programming can save time and energy when writing simple programs that implement classes or complex functions as would be the case in object-oriented programming. In most cases straightforward pieces of code would get the job done. Structured programming is ideal for development of small programs as it would beat the purpose spending a lot of time and energy designing classes when an entire working program could be developed within the same time period.

This will also hold true in that small programs are easy to maintain and most fit within a single page or so, making more sense for the Web developer to visualize the program code. It therefore make structured programming well suited for small to medium website which would probably not require much maintenance.

Structured programs are easy to read and understand as you would simply need to follow the source code as it is written on the file as you would be guaranteed that there are no jumps or deviations to other pieces of code in other files. This however does not mean such a program would necessarily be easier to maintain; that brings me to the reason why object-oriented programming would be suited in other situations.

Object-oriented programming takes pride in its suitability for sustaining huge software and web development projects. This is a far better option than using structured programming when you have massive code bases. The sheer nature of object-oriented programs allows the developer to save a lot of time and energy when developing programs as the components of the programs are in the form of objects which can be plugged into the program wherever they are needed. So having an application with several hundred buttons implemented as objects would be pretty easy to maintain, for example, if the developer wanted to change the style or behavior of all the buttons, it would simply entail changing a single object which defines everything about the button and this would change every instance of the button object.

Even though object-oriented programming is suitable for large project and long term maintenance of software of web projects, this method of programming poses its own disadvantages as object-oriented programming tends to be more complex than structured programming as there is a lot as far as design the decisions that need to be made and consequently the entire task of managing the project may be a bit difficult to the inexperienced web developer.

As for performance, from a user stand point, the difference between structured and object-oriented programming may be minimal. However in some cases the fact that object-oriented programs are slower seeing as there is extra work the interpreter has to go through to compiler the classes as opposed to the structure method of running in a top down sequence. This could make one consider what method to use for their next web development project.

**EXPLANATION #2**

* OOP is closer to the phenomena in real world due to the use of objects whereas structured programming is a bit away from the natural way of thinking of human beings.
* Structured programming is a subset of object-oriented programming. Therefore, OOP can help in developing much larger and complex programs than structured programming.
* Under structured programming, the focus of a program is on procedures or functions (behavior) and the data is considered separately (data structures are not well organized within the program) whereas in OOP, data (structure) and methods (behavior) both are in collective focus.
* In OOP, the basic units of manipulation are the objects whereas in case of structured programming, functions or procedures are the basic units of manipulation.
* The focus of a program is on manipulation of data in structured programming whereas the focus of OOP is on both data (structure and states of objects) as well as on its manipulation (behavior of objects).
* In OOP, data is hidden within the objects and it manipulation can be strictly controlled whereas in structure programming the data in the form of variables is exposed to unintended manipulation too.
* The OOP promotes the reuse of classes and their parts of the code too. In addition, it also supports the inheritance of state and behavior. This feature is missing in structured programming.
* The OOP supports polymorphism of operations.
* The concepts of OOP have got integrated with most of the prominent object oriented methodologies of software development in a better way and help in reducing the development effort and time as compared with that of structured methodologies based on structured programming.
* The structured programming normally emphasized on single exit point in their constituent function or procedures. Since each function/procedure allocates some memory to itself for storing the values of its variables and code, before the exit, there must be a provision for deallocating this memory. Otherwise, more and more of the memory gets occupied by each function/procedure and in large programs, there may occur a shortage of memory for use by other function/procedures and the processing can get slower or even completely halted. When in any function/procedure, memory is allocated, but not deallocated, a memory leak is said to have occurred (the memory has leaked out of the computer) in it.