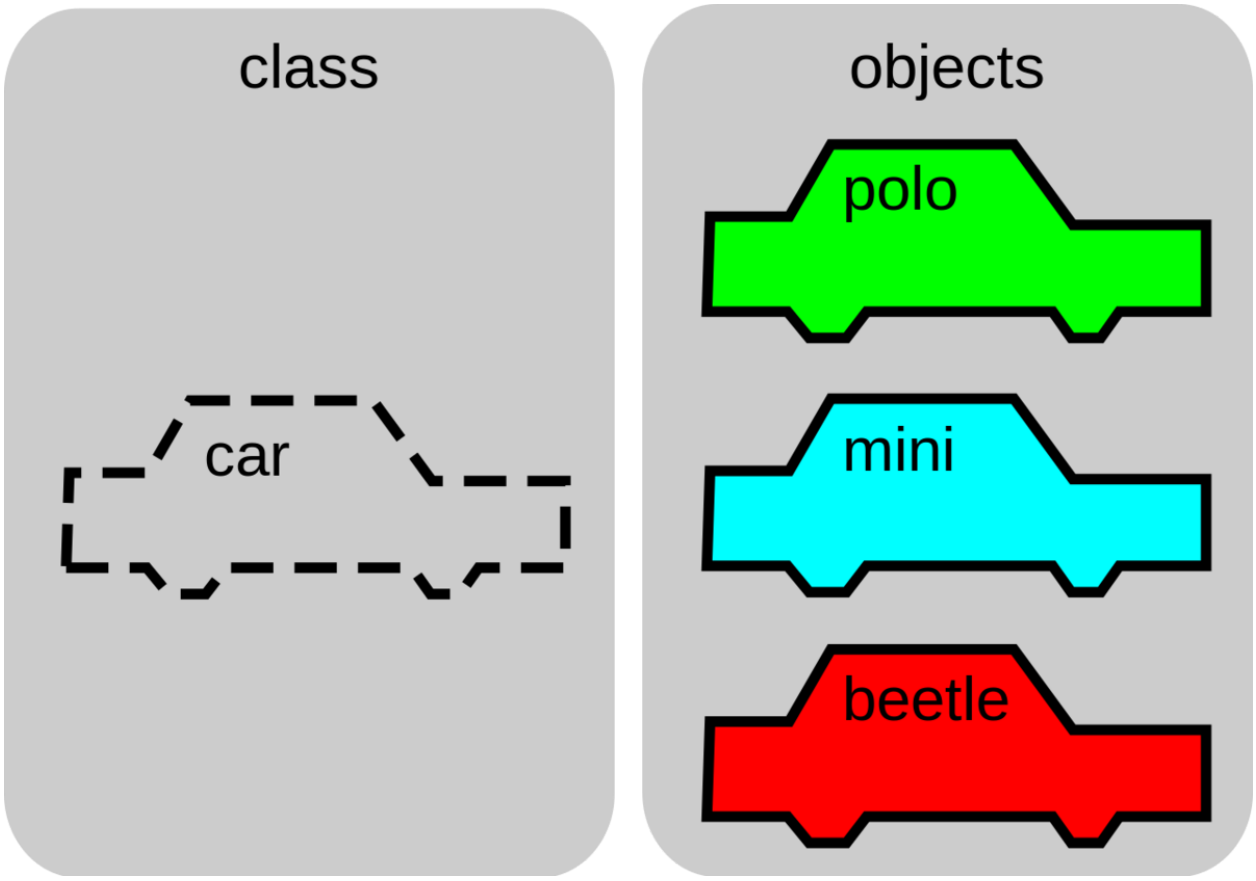
OOP

3/4/2020

Overview

This pill is designed to improve our understanding of object oriented programing. To clarify all the concepts of OOP the following questions are answered:

- What is Object oriented Programming ?
    - Object-oriented programming (OOP) refers to a type of computer programming (software design) in which programmers define the data type of a data structure, and also the types of operations (functions) that can be applied to the data structure. In this way, the data structure becomes an object that includes both data and functions. In addition, programmers can create relationships between one object and another. For example, objects can inherit characteristics from other objects.
- What is Class ?
    - In object-oriented programming, a class is a blueprint for creating objects (a particular data structure), providing initial values for state (member variables or attributes), and implementations of behavior (member functions or methods).
- What is an Object ?
    - The user-defined objects are created using the class keyword. The class is a blueprint that defines the nature of a future object. An instance is a specific object created from a particular class. Classes are used to create and manage new objects and support inheritance—a key ingredient in object-oriented programming and a mechanism of reusing code.

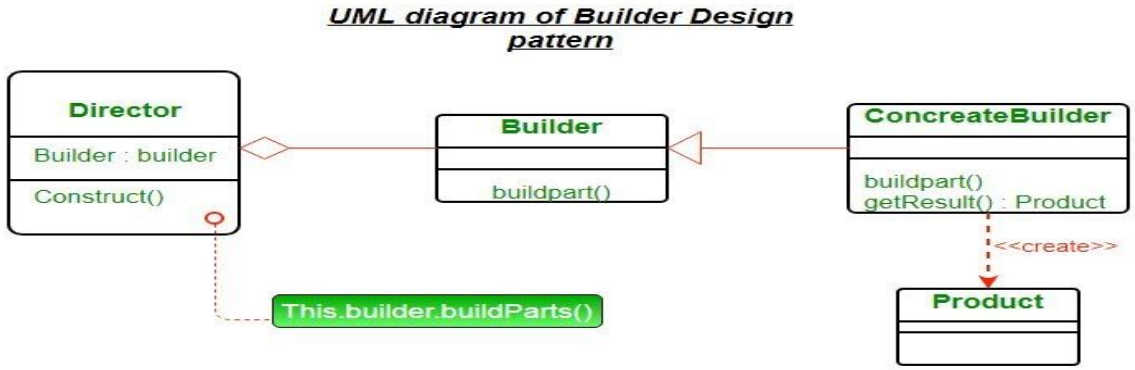


- What is an Instance ?
    - An instance is a unique copy of a Class that represents an Object. When a new instance of a class is created, the JVM will allocate a room of memory for that class instance.
    - In short, An object is a software bundle of related state and behavior. A class is a blueprint or prototype from which objects are created. An instance is a single and unique unit of a class.
- What is a Property ?
    - Properties are an object-oriented idiom. The term describes a one or two functions (depending on the desired program behavior) - a 'getter' that retrieves a value and a 'setter' that sets a value. By convention, properties usually don't have many side-effects. (And the side-effects they do have usually have limited scope: they may validate the item being set, notify listeners of a change, or convert an object's private data to or from a publicly-declared type.)
- What is a Method ?
    - Methods ("member functions") are similar to functions, they belong to classes or objects and usually express the verbs of the objects/class. For example, an object of type Window usually would have methods open and close which do corresponding operations to the object they belong to.

- A method in object-oriented programming is a procedure associated with a class. A method defines the behavior of the objects that are created from the class. Another way to say this is that a method is an action that an object is able to perform. The association between method and class is called binding. Consider the example of an object of the type 'person,' created using the person class. Methods associated with this class could consist of things like walking and driving. Methods are sometimes confused with functions, but they are distinct.

- What is the difference between a Function and Method ?
  - A function is a piece of code that is called by name. It can be passed data to operate on (i.e. the parameters) and can optionally return data (the return value). All data that is passed to a function is explicitly passed. A method is a piece of code that is called by a name that is associated with an object. In most respects it is identical to a function except for two key differences:

    - A method is implicitly passed the object on which it was called.
    - A method is able to operate on data that is contained within the class (remembering that an object is an instance of a class - the class is the definition, the object is an instance of that data).
- What is meant by Builder ?
  - Builder is a creational design pattern that lets you construct complex objects step by step. The pattern allows you to produce different types and representations of an object using the same construction code



  - Advantages of Builder Design Pattern

    - The parameters to the constructor are reduced and are provided in highly readable method calls.
    - Builder design pattern also helps in minimizing the number of parameters in constructor and thus there is no need to pass in null for optional parameters to the constructor.

        - Object is always instantiated in a complete state
        - Immutable objects can be build without much complex logic in object building process.
    - Disadvantages of Builder Design Pattern

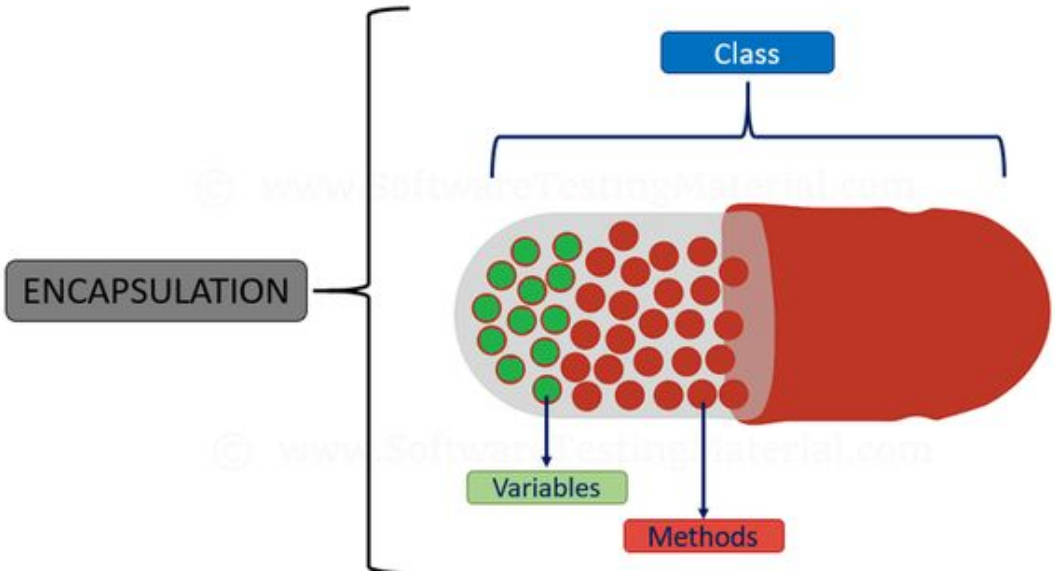
        - The number of lines of code increase at least to double in builder pattern, but the effort pays off in terms of design flexibility and much more readable code.
        - Requires creating a separate ConcreteBuilder for each different type of Product.
    - An example for a class for construction of a home. Home is the final end product (object) that is to be returned as the output of the construction process. It will have many steps like basement construction, wall construction and so on roof construction. Finally the whole home object is returned. Here using the same process you can build houses with different properties.
    - interface HousePlan
    - {
    - public void setBasement(String basement);
    -
    - public void setStructure(String structure);
    -
    - public void setRoof(String roof);
    -
    - public void setInterior(String interior);
    - }
    -
    - class House implements HousePlan
    - {
    -
    - private String basement;
    - private String structure;
    - private String roof;
    - private String interior;
    -
    - public void setBasement(String basement)
    - {
    - this.basement = basement;
    - }
    -
    - public void setStructure(String structure)
    - {
    - this.structure = structure;
    - }
    -
    - public void setRoof(String roof)
    - {

```
        this.roof = roof;
    }

    public void setInterior(String interior)
    {
        this.interior = interior;
    }

}


interface HouseBuilder
{

    public void buildBasement();

    public void buildStructure();

    public void bulidRoof();

    public void buildInterior();

    public House getHouse();
}

class IglooHouseBuilder implements HouseBuilder
{
    private House house;

    public IglooHouseBuilder()
    {
        this.house = new House();
    }

    public void buildBasement()
    {
        house.setBasement("Ice Bars");
    }

    public void buildStructure()
    {
        house.setStructure("Ice Blocks");
    }

    public void buildInterior()
    {
        house.setInterior("Ice Carvings");
    }
```

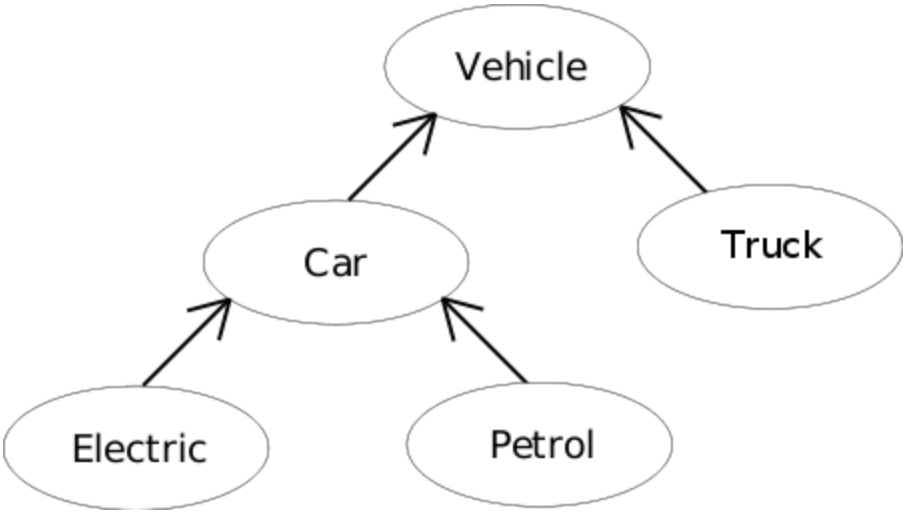
```
    public void bulidRoof()
    {
        house.setRoof("Ice Dome");
    }

    public House getHouse()
    {
        return this.house;
    }
}

class TipiHouseBuilder implements HouseBuilder
{
    private House house;

    public TipiHouseBuilder()
    {
        this.house = new House();
    }

    public void buildBasement()
    {
        house.setBasement("Wooden Poles");
    }

    public void buildStructure()
    {
        house.setStructure("Wood and Ice");
    }

    public void buildInterior()
    {
        house.setInterior("Fire Wood");
    }

    public void bulidRoof()
    {
        house.setRoof("Wood, caribou and seal skins");
    }

    public House getHouse()
    {
        return this.house;
    }

}

```

```
class CivilEngineer
{

    private HouseBuilder houseBuilder;

    public CivilEngineer(HouseBuilder houseBuilder)
    {
        this.houseBuilder = houseBuilder;
    }

    public House getHouse()
    {
        return this.houseBuilder.getHouse();
    }

    public void constructHouse()
    {
        this.houseBuilder.buildBasement();
        this.houseBuilder.buildStructure();
        this.houseBuilder.bulidRoof();
        this.houseBuilder.buildInterior();
    }
}

class Builder
{
    public static void main(String[] args)
    {
        HouseBuilder iglooBuilder = new IglooHouseBuilder();
        CivilEngineer engineer = new CivilEngineer(iglooBuilder);

        engineer.constructHouse();

        House house = engineer.getHouse();

        System.out.println("Builder constructed: "+ house);
    }
}
Output :

Builder constructed: House@6d06d69c
```

- What is the difference between a Class, an Object and an instance ?
    - A class is a blueprint which you use to create objects. An object is an instance of a class. Instance describes the relationship between objects and classes
- What is the concept of encapsulation ?



    - In object-oriented programming (OOP), encapsulation refers to the bundling of data with the methods that operate on that data, or the restricting of direct access to some of an object's components. Encapsulation is used to hide the values or state of a structured data object inside a class, preventing unauthorized parties' direct access to them. Publicly accessible methods are generally provided in the class (so-called "getters" and "setters") to access the values, and other client classes call these methods to retrieve and modify the values within the object.
- What do we understand by abstraction ?
    - In object-oriented programming theory, abstraction involves the facility to define objects that represent abstract "actors" that can perform work, report on and change their state, and "communicate" with other objects in the system.
    - Abstraction allows us to expose limited data and functionality of objects publicly and hide the actual implementation. It is the most important pillar in OOPS.
- What is understood by inheritance?
    - Inheritance is an eminent concept in Object Orient Programming (OOPS) Paradigm. It provides a mechanism for establishing relationships and building hierarchies of class in object composition. Inheritance means the use of code that is pre-written or created previously. And one thing to keep in mind is that we are just using the code and not updating or changing it. The functions and methods defined in one class may be used in manipulating other data members of the class.
    - Types of Inheritance:
        - Single Inheritance: One derived class inherits from one base class.
        - Multiple Inheritance: One derived class inherits from many base classes.

        - Multilevel Inheritance: One derived class inherits from other derived classes.
        - Hierarchical Inheritance: More than one derived classes inherit from one base class.
        - Hybrid Inheritance: A combination of more than one type of inheritance.
    - 

- What is understood by Polymorphism ?
    - In object-oriented programming, polymorphism refers to a programming language's ability to process objects differently depending on their data type or class. More specifically, it is the ability to redefine methods for derived classes. For example, given a base class shape, polymorphism enables the programmer to define different area methods for any number of derived classes, such as circles, rectangles and triangles. No matter what shape an object is, applying the area method to it will return the correct results. Polymorphism is considered to be a requirement of any true object-oriented programming language (OOPL).
- What is understood by overload ?
    - Method or function overloading allows a method with the same name to be declared more than once given that they have different input parameters. For example, there is a method to calculate the area of shapes. There are different shapes such as circles and rectangles. In these two cases, if someone calls the function calculateArea, the result should be correct for the given shape, regardless of whether the shape is a circle or a rectangle. The declaration of this in a programming language in shown below:
        - Rectangle: calculateArea(double length, double width)
        - Circle: calculateArea(double radius), A great advantage of method overloading is that it allows a programmer to use the function appropriately without having to know the inner-workings of that method.
- What is meant by overriding ?

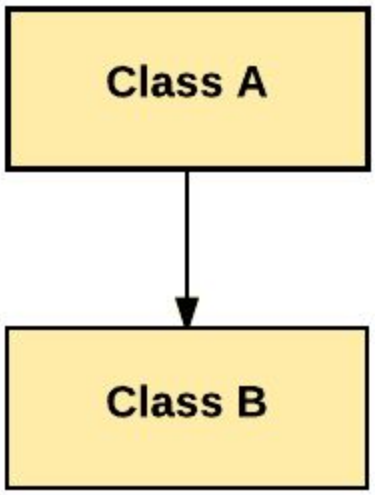
    - Overriding is an object-oriented programming feature that enables a child class to provide different implementation for a method that is already defined and/or implemented in its parent class or one of its parent classes. The overridden method in the child class should have the same name, signature, and parameters as the one in its parent class. Overriding enables handling different data types through a uniform interface. Hence, a generic method could be defined in the parent class, while each child class provides its specific implementation for this method.
- What difference exists between overload and override ?
    - When two or more methods (functions) in the same Class have the same name but different parameters is called method overloading.

    - Class myClass
    - {
    - int Func(int x)
    - {
    - }
    - int Func(string s)
    - {
    - }
    - int Func(int x,string s)
    - {
    - }
    - }

    - When two or more methods (functions) have the exact same method name, return type, number of parameters, and types of parameters as the method in the parent class is called method Overriding.
    - Method overloading happens in the same class shares the same method name but each method should have different number of parameters or parameters having different types and order. But in method overriding derived class have the same method with same name and exactly the same number and type of parameters and same return type as a parent class.
    - Method Overloading happens at compile time while Overriding happens at runtime. In method overloading, method call to its definition has happens at compile time while in method overriding, method call to its definition happens at runtime.
    - In method Overloading, two or more methods share the same name in the same class but having different signatures while in method overriding, method of parent class is re-defined in the inherited class having the same signature.
    - In the case of performance, method overloading gives better performance when compared to overriding because the binding of overridden methods is being done at runtime.

    - Static binding happens when method overloaded while dynamic binding happens when method overriding.
    - Method overloading add or extend more to the method functionality while method overriding is to change the existing functionality of the method
    - Static methods can be overloaded, that means a class can have more than one static method of the same name. But static methods cannot be overridden, even if you declare the same static method in derived class it has nothing to do with the same method of base class.
- What is a static class ?
    - can use a class you must instantiate it. And like blueprints and houses you can instantiate as many instances of a class as you like.
    - But a static class is different in a few ways.
        - It doesn't need to (and cannot) be instantiated.
        - You cannot have multiple copies of a static class
        - A static class cannot inherit or be inherited
        - Static classes are loaded into memory before other classes are instantiated.
    - Statics are like little gatekeepers or the mediators between the internal communication of source code and its data propagating throughout a program at runtime. Therefore, statics, especially - play an important role in Object Oriented Programming (OOP). They are very influential when dealing with large code bases and complex software development in particular.
- Look for 3 advantages of OOP ?
    - It provides a clear modular structure for programs which makes it good for defining abstract data types in which implementation details are hidden
    - Objects can also be reused within an across applications. The reuse of software also lowers the cost of development. More effort is put into the object-oriented analysis and design, which lowers the overall cost of development.
    - It makes software 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
    - Reuse also enables faster development. Object-oriented programming languages come with rich libraries of objects, and code developed during projects is also reusable in future projects.
- What is the disadvantage of this paradigm ?
    - It requires more data protection.
    - Inability to work with existing systems.
    - Larger program size.
    - Not suitable for all types of problems-for smaller problems it is in general not suitable.
- Explain in which cases OOP will be used as opposed to other paradigms like functional programing?

    - FP is most suited for solving problems where there is a lot of data transformations or maths involved. Such use cases could be machine learning. We train models, filter them, transform them, do a lot of different calculations. Solution implementation looks close to a bunch of equations combined together, so doing it in FP feels more natural.
    - OOP is best suited for implementing rich domains, such us game logic, also enterprise applications, focused on services and relations and model processing from the outside. SOLID makes complex business or game rules simple putting it under a well dozed layer of abstraction. Working with concepts while seeking for simplicity is what OOP is for.
- Use cases for encapsulation ?
    - Encapsulation binds together the data and functions that manipulate the data, keeping it safe from interference and misuse.
    - Real World Example: Every time you log into your email account(GMail, Yahoo, Hotmail or official mail), you have a whole lot of processes taking place in the backend, that you have no control over. So your password, would probably be retrieved in an encrypted form, verified and only then you are given access. You do not have any control over how the password is verified, and this keeps it safe from misuse.
- Use cases for abstraction ?
    - Imagine there is a graphics library called "nicepic" that contains pre-defined functions for all abstractions discussed above: rectangles, squares, triangles, house, village.
    - Say you want to create a program based on the above abstractions that paints a nice picture of a house, all you have to write is this:
        - import nicepic
        - Draw_house()
    - So that's just two lines of code to get something much more elaborate.
- Use case for inheritance ?



    -
        - We use it for when we want to inherit behavior from a parent class. Of course since people love jargon so much, the parent class could also be referred to as the base class or super class. The child class goes by things such as the derived class, the sub class, or the heir class.
- Use cases of static class ?
    - Two common uses of static fields are to keep a count of the number of objects that have been instantiated, or to store a value that must be shared among all instances. Static methods can be overloaded but not overridden, because they belong to the class, and not to any instance of the class.
- An “object” is an “instance” of a “class”

Objectives of project

- Improve your knowledge in object-oriented programming.
- Understand what are the fundamental principles of the OOP

Table of Contents

2. Objectives
3. Project requirements
4. Risk management plan
5. Tasks for the project
6. Chronogram
7. Calendar
8. Git Workflow
9. Technologies used
10. Incidents

Project requirements

- You must make use of object-oriented programming using Javascript in its ES6 version or higher.
- Create a clear and orderly directory structure.
- Both the code and the comments must be written in English
- Use the camelCase code style for defining variables and functions
- Remember that it is important to divide the tasks into several sub-tasks so that in this way you can associate each particular step of the construction with a specific commit
- You should try as much as possible that the commits and the planned tasks are the same
- Delete files that are not used or are not necessary to evaluate the project

Risk management

Every project has risks. These risks must be taken into account to improve the workflow of the project. Managing these risks is important for due completion of the project. Unchecked risks could not only lead to hamper of project deliverance but also a badly executed project. The risks hence associated with the project are documented as follows:

Risk	Risk level
Unfamiliarity with OOP	Medium to Low
Health and computer issues	High
Delivering in time	Medium
Completing all the project requirements	Medium

Task Management

The following are the tasks which needed to be accomplished for the completion of the project. The tasks have been divided according to the project specifics. Each task has been clearly defined and their priority as well as their difficulty and time needed. Difficulty level is explained on a scale of 1 to 5 (5 being the most difficult). Priority is explained on the level of 1 to 5 (again 5 the highest parameter being the most prioritized work).

Task	Priority	Description	Difficulty	Time
Read the description of the project	5	This task involves complete understanding of requirements for the project	1	30 mins
Create git repo	5	Creating of git repository for project execution and delivery	1	2 mins
Create the directory structure	4	Creating the necessary files, folders and subfolders for this project.	1	2 mins
OOP theoretical	4	Understand completely the overview of OOP	2	1-2 hrs
OOP practical	4	Use the understanding gained in the previous section to apply in practical purposes	1	30mins-2 hrs
Presentation	3	Create a presentation for the project	1	1 hr
Documentation	1	Write the documentation for this project	1	3-4 hr

Defining this part is crucial to the development of the project. It is important to make a good analysis of the situation to organize the project in a good way.

Chronogram

This shows the time-line it took for the project to finish and also the tasks that were achieved during those timeline.

The tasks have been mentioned in the left axis

Tasks	Fri 1000 hrs	Fri 1200 hrs	Fri 1400 hrs	Fri 1600 hrs	Fri 1800 hrs	Mon 1000 hrs	Mon 1200 hrs	Mon 1400 hrs
Read project descriptio -n	X							
Git Repo	X							
Create files	X							
OOP theory	X	X	X					
OOP practical				X			X	X
Presentati on							X	
Documen tation					X	X	X	

Git Workflow

For this project, all commits we'll be pushed directly to the Master branch. All commits will use a descriptive message, so that the admin and other internet user can follow through the processes.
For more information go to :
https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Incidents

- None

Technologies used

For this project, we will use the following technologies:

- PhP
- OOP