HOMEWORK WEEK 3
DANIELA TRIPON

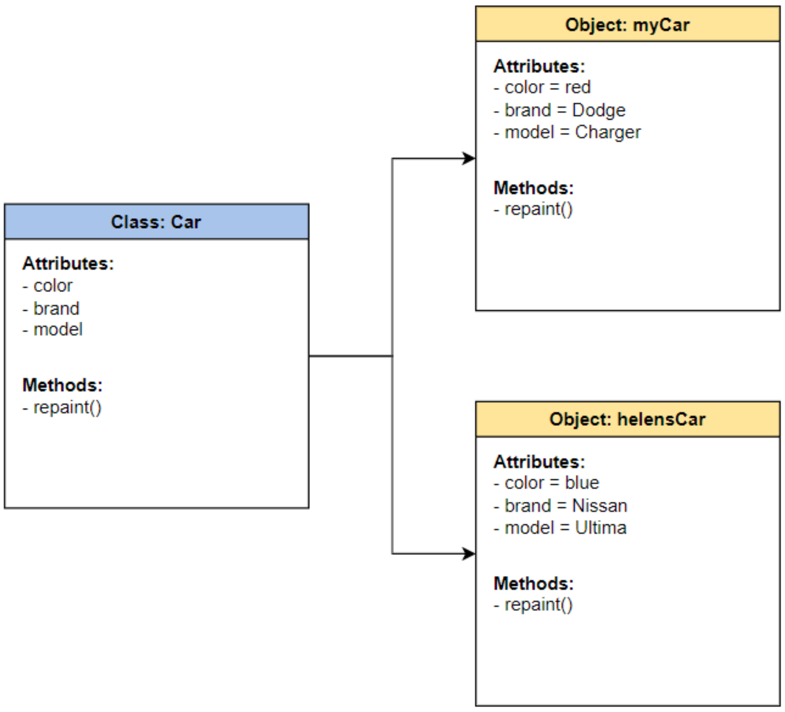
This week's homework is a research based one. You'll need to conduct independent learning, in combination with existing material (where available), to answer the questions below. The reason for this homework is to ensure you are aware of critical topics in CS. These topics were difficult to cover within the existing lesson schedules, but due to their importance are placed within the homework instead. Make sure to research, learn and then answer the following:

1. What is OOP? How may you have already made use of it (e.g. class components)?
    a. *Feel free to give a fairly light answer here - as you'll need to do the deep-part / actual meat in the following questions when you cover each of OOP's pillars*

Answer

OOP stands for Object Oriented Programming and is a very important programming paradigm that should be known by any developers and is based on objects and classes. The idea behind it is to facilitate the code reuse based on individual instances of objects. A few examples of programming languages that use the concept are Python, Java, C++, JavaScript, and many others.

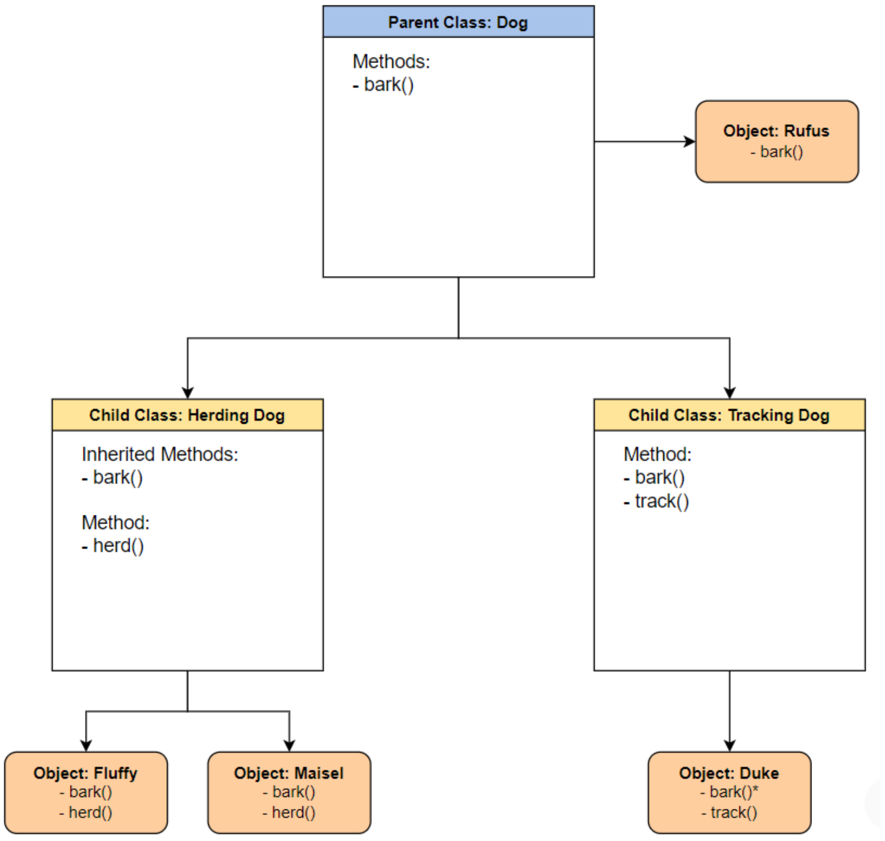
Classes are the blueprint or the skeleton on which we create individual and more specific **objects**.
Example of a class: Animal;
Example of objects based on class Animal: Dog, Koala, Kangaroo.
Please see example Class Car diagram.



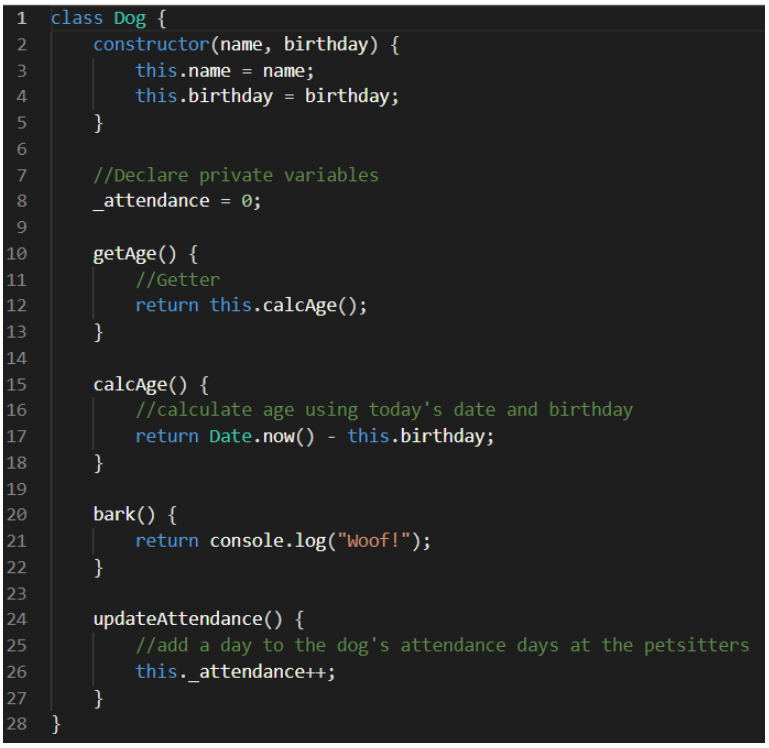
Class blueprint being used to create two Car type objects, myCar and helensCar

Class Car example

Classes hold common **attributes**, but generic values, for example colour. They also have **methods** which are functions inside the class that are helpful only for objects of the same type. For example, we can have a method fur() that changes the type of fur only for objects (animals) that have fur. Animal class can have children classes that inherit the attributes and methods, but also have specific methods that are passed to objects which inherit the methods from the Animal class and the child class they derive from. Please see the diagram bellow based on a Class Dog.



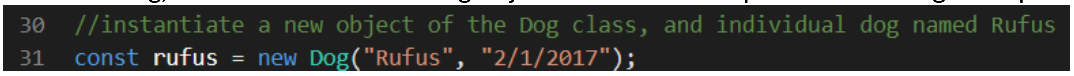
Class Dog example



```javascript
class Dog {
    constructor(name, birthday) {
        this.name = name;
        this.birthday = birthday;
    }

    //Declare private variables
    _attendance = 0;

    getAge() {
        //Getter
        return this.calcAge();
    }

    calcAge() {
        //calculate age using today's date and birthday
        return Date.now() - this.birthday;
    }

    bark() {
        return console.log("Woof!");
    }

    updateAttendance() {
        //add a day to the dog's attendance days at the petsitters
        this._attendance++;
    }
}
```

JavaScript example of class Dog

Based on class Dog, we can create a new dog object called Rufus as per the following example.



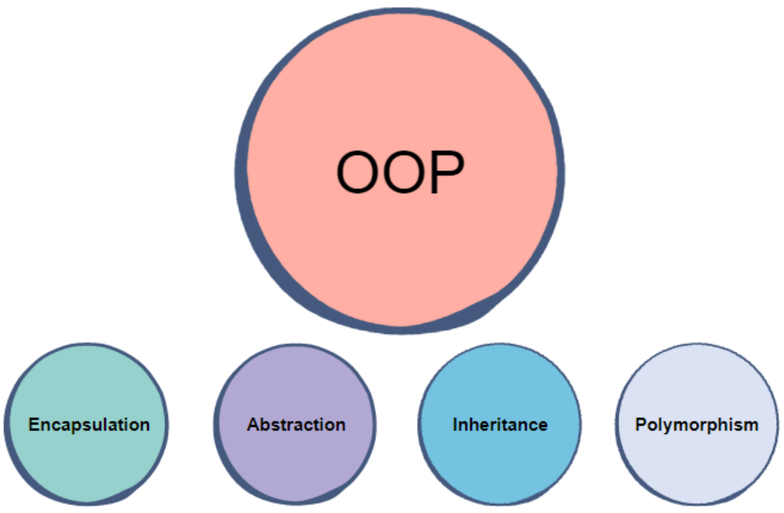
```javascript
//instantiate a new object of the Dog class, and individual dog named Rufus
const rufus = new Dog("Rufus", "2/1/2017");
```

Besides attributes and methods, we see the constructor method that is used to build the objects. In this example, the constructor has two arguments name, and birthday, and based on them, we create the new object and pass the values “Rufus”, and “2/1/2017”. The new object inherits all methods from class Dog.

Objects have behaviours and states. Behaviours is what the object can do and is represented by the methods, whilst state is the data stored in the constructor, for example in this case name and birthday.

Classes can be abstract classes that allow children/ subclasses to use methods to override or implement them and create a functionality. An abstract class can be extended by exactly one class only. There are also classes defined as interfaces and which allow only the definition of its functionality and not its implementation. A class can have many interfaces.

There are four principles in Object Oriented Programming, and these are covered in the next answers. Also, classes can have private, public, and protected fields and allows or restricts access to properties and methods inside a class.

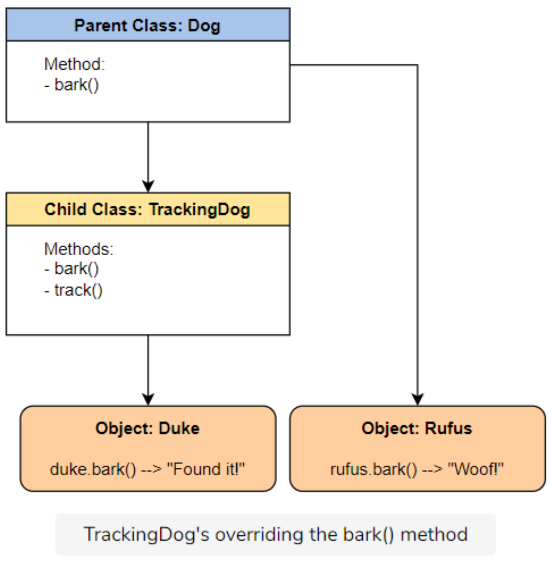


OOP needs lots of planning before coding, so we have a clear understanding of the program' structure. We need to have a clear idea about the classes we define and how we can reuse them to create specific objects. They are great for code reusability, a clear data structure, and saving time in due course.

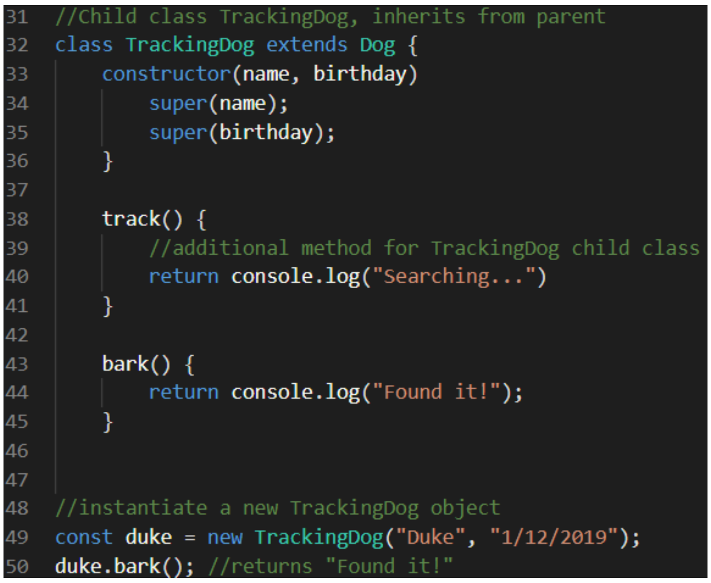
2. What is Polymorphism?

Answer
The objects are designed to share behaviours, where a method can override or overload the behaviour from the parent class. As a short description, we can say that polymorphism allows multiple methods to do the same thing/ task.

In **method overriding**, a method can use a different implementation than the parent's class. We can look at the following example for child class "TrackingDog" of the class Dog saw in the answer 1. We want to override the method bark() so it runs a different code.



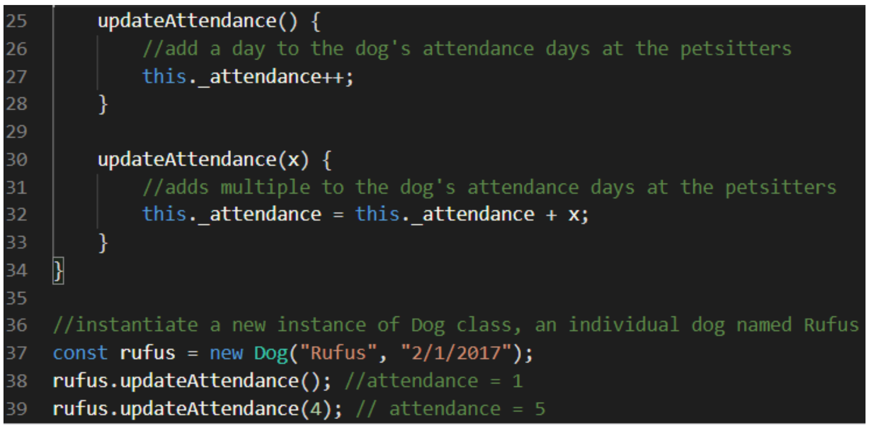
Override method diagram



```
//Child class TrackingDog, inherits from parent
class TrackingDog extends Dog {
    constructor(name, birthday)
        super(name);
        super(birthday);
    }

    track() {
        //additional method for TrackingDog child class
        return console.log("Searching...")
    }

    bark() {
        return console.log("Found it!");
    }


//instantiate a new TrackingDog object
const duke = new TrackingDog("Duke", "1/12/2019");
duke.bark(); //returns "Found it!"
```

Method Overriding JavaScript code example

Method overloading is used by compile time polymorphism. In this case, we keep the same method name, however we pass a different number of arguments to the function call. The results will be different, based on the number of arguments given. Please see bellow the following example based on the Dog class and updateAttendance() methods.



```
    updateAttendance() {
        //add a day to the dog's attendance days at the petsitters
        this._attendance++;
    }

    updateAttendance(x) {
        //adds multiple to the dog's attendance days at the petsitters
        this._attendance = this._attendance + x;
    }
}

//instantiate a new instance of Dog class, an individual dog named Rufus
const rufus = new Dog("Rufus", "2/1/2017");
rufus.updateAttendance(); //attendance = 1
rufus.updateAttendance(4); // attendance = 5
```

Method overloading JavaScript code example

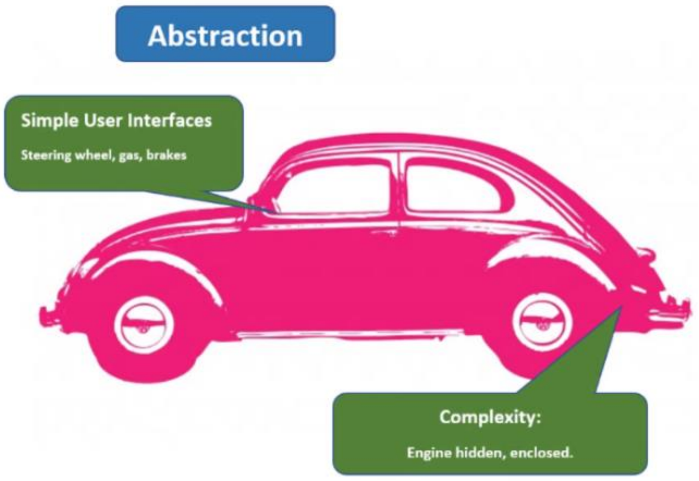
We can see that if there are no arguments passed to the function when we call it, then updateAttendance() runs and increments the attendance value by 1. Otherwise, if we pass a value as an argument, then the updateAttendance(x) runs and adds that value to the attendance.

Among the benefits of using polymorphism, we see method overriding, method overloading, and same interface can pass different objects.

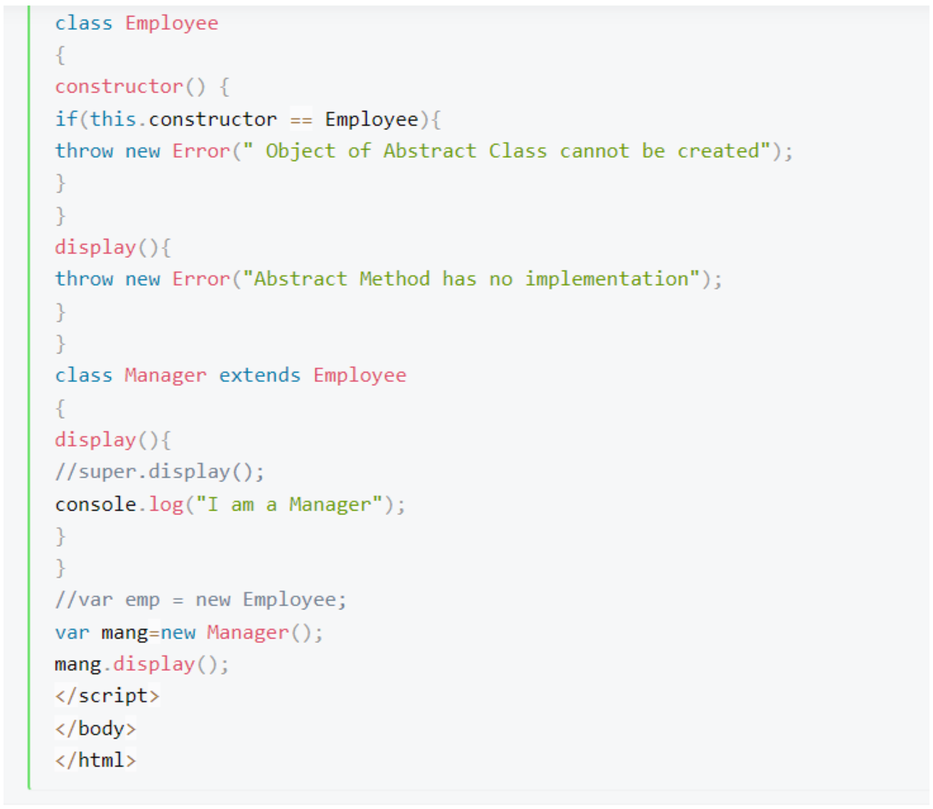
3. What is Abstraction?

Answer
Abstraction allows user interaction with specific attributes and methods only from an object. It uses simplicity to represent complexity by hiding the complexity of classes and objects derived from them from the user. Abstract class can’t be instantiated, and abstract methods cannot be implemented, only declared. To use an abstract class, it needs to be inherited and it requires subclasses to implement it’s methods.

This works like an extension from the encapsulation, which will be discussed at number 5.
For example, we can drive a car by using simple components (objects) such as pedal, steering wheel, brake, fuel, but we don't know all the components of the engine. These are hidden from the user and known by the mechanical engineer only, otherwise will confuse and distract the user.

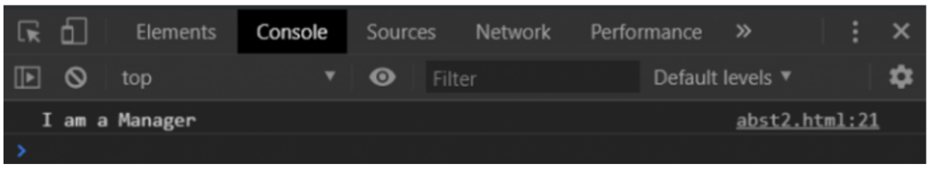


Abstract class example overview



```
class Employee
{
constructor() {
if(this.constructor == Employee){
throw new Error(" Object of Abstract Class cannot be created");
}
}
display(){
throw new Error("Abstract Method has no implementation");
}
}
class Manager extends Employee
{
display(){
//super.display();
console.log("I am a Manager");
}
}
//var emp = new Employee;
var mang=new Manager();
mang.display();
</script>
</body>
</html>
```

Output 1 – Correct Output



```
I am a Manager                                        abst2.html:21
```

Example Abstract class in JavaScript

Among the benefits for using abstraction, we can include:

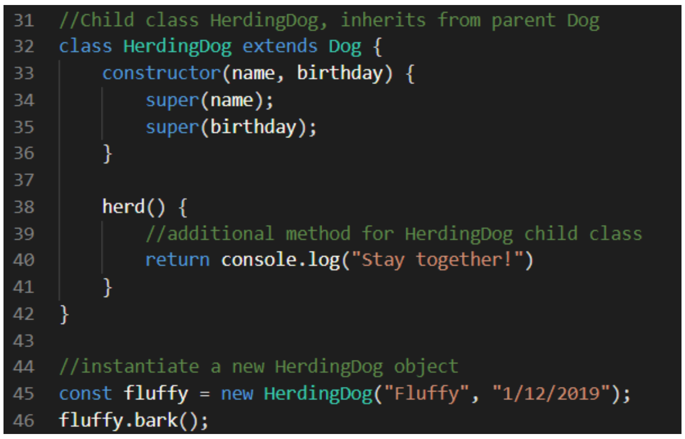
- Security – only selected data is displayed, modified, and accessed through classes so it reduces data exposure.
- High level, simple user interfaces used.
- Abstraction is rarely changed by code updates.
- Maintenance is facile.

4. What is Inheritance?

Answer

Through inheritance, parent classes can pass down to children classes their behaviours and attributes. Children classes inherit these attributes and methods by extending parent's class functionality and add additional methods and attributes. Thanks to this, inheritance is great for reusability. A parent class is also called a base or a super class, and a child class is called an extended or derived class.

Going back to the class Dog example, we can consider a subclass for dogs that can herd animals. We can see that even though this subclass is created for another type of dog, not all dogs have this ability, which makes them unique. Even though this subclass inherits from the parent class, we can add this new method that represents the ability to herd. Based on a common parent class, we can create children classes specific for their functionalities. This way, we avoid code duplication and take advantage of the parent class's functionality. Please see the example below.



```
31  //Child class HerdingDog, inherits from parent Dog
32  class HerdingDog extends Dog {
33      constructor(name, birthday) {
34          super(name);
35          super(birthday);
36      }
37
38      herd() {
39          //additional method for HerdingDog child class
40          return console.log("Stay together!")
41      }
42  }
43
44  //instantiate a new HerdingDog object
45  const fluffy = new HerdingDog("Fluffy", "1/12/2019");
46  fluffy.bark();
```

Example Inheritance class in JavaScript

On this example, we can see that child class inherits the methods from the parent class and doesn't copy them (see method bark() example on line 46). When the method is called with a new object, the program looks for it in the child class first, and if it cannot find it, it looks in the parent class and runs the code once it is found.

Inheritance is called prototyping in JavaScript and is an object known as a template from which other objects inherit its behaviours and properties. A prototype chain is made of many object templates.

Benefits for using it:

- Reusability
- Programs are easier to read
- Short and concise programs that are easy to understand

- Code is easy to debug
- Great flexibility to change

Disadvantages:

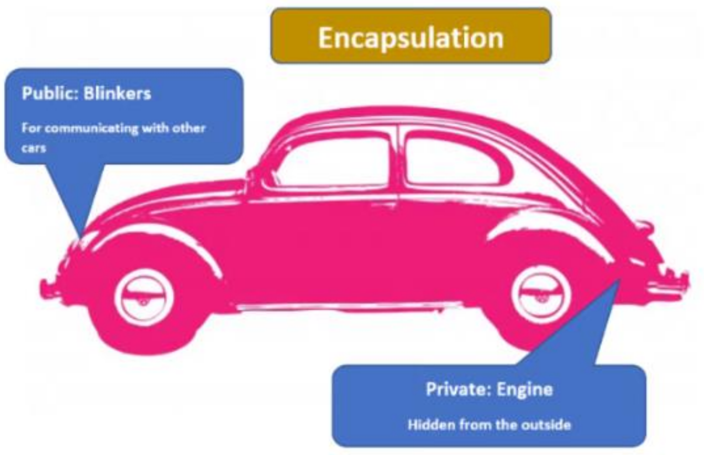
- Derived class and super class are tight together and dependent to one another.
- In case of changes in the super class, the derived class needs to be updated as well.
- If any methods are deleted from the super class, the base's class functionality is affected.

5. What is encapsulation?

Answer
By using encapsulation, we ensure all behaviours and attributes from a class template are stored inside an object and we reveal only certain information we select. Once we instantiate a new object, all attributes and methods are encapsulated in it.

Encapsulation uses two types of fields public and private. Any properties and methods written in the public field grant external access, whilst the properties and methods in the private fields are accessible only inside the class, to other methods. As mentioned in the answer 1, classes can have another section besides the public and private, which is protected, which only child cases can access it. In JavaScript, protected fields can be inherited and are written with "_" in front of the field. On the other side, private fields can't be inherited, and are written with the "#" symbol as a prefix.

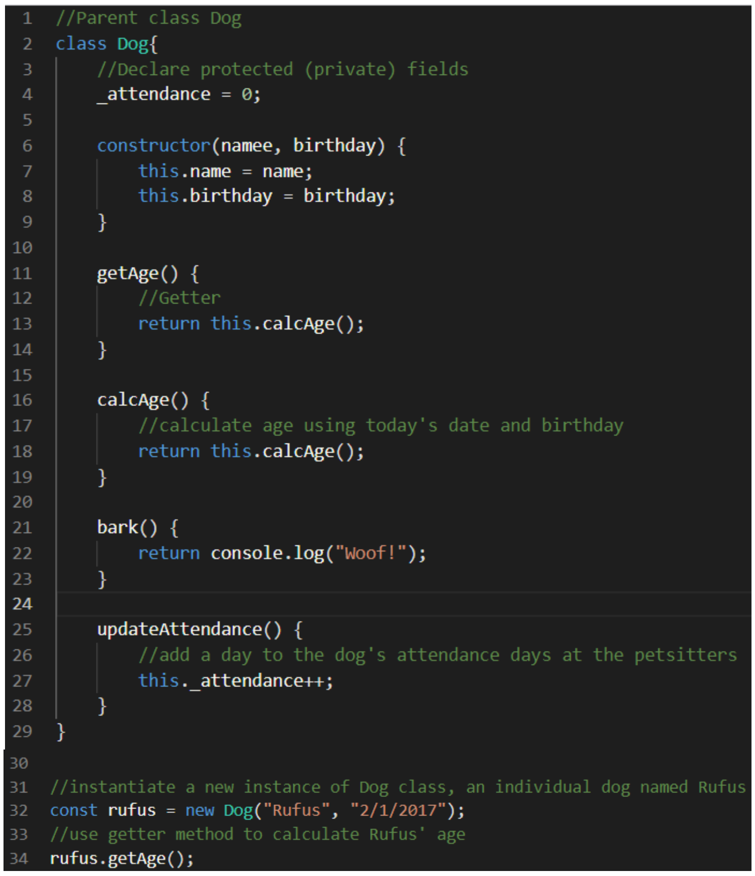
Going back to the Car class example, we have public methods such as blinkers, that are used to communicate with other drivers and are known as public interfaces. The engine, which is hidden, doesn't need to pass any information to other drivers, so is hidden using the private field.



Encapsulation class example overview

Encapsulation classes are also great for security since a developer can decide the properties and methods that can be updated in an object by assigning them to the desired type of field.

Now, going back to the class Dog example, we can add _attendance as a protected field.



```
//Parent class Dog
class Dog{
    //Declare protected (private) fields
    _attendance = 0;

    constructor(namee, birthday) {
        this.name = name;
        this.birthday = birthday;
    }

    getAge() {
        //Getter
        return this.calcAge();
    }

    calcAge() {
        //calculate age using today's date and birthday
        return this.calcAge();
    }

    bark() {
        return console.log("Woof!");
    }

    updateAttendance() {
        //add a day to the dog's attendance days at the petsitters
        this._attendance++;
    }
}

//instantiate a new instance of Dog class, an individual dog named Rufus
const rufus = new Dog("Rufus", "2/1/2017");
//use getter method to calculate Rufus' age
rufus.getAge();
```

Protected field example

We can see that the method used to calculate age is hidden in the class, so it is encapsulated. By hiding important fields, the developer can prevent misuse or mistaken updates by other developers, or from phishing. This is done by deciding which fields can be updated and which cannot and make those fields public or private. This adds security to the class templates and protects the data they hold. We can grant external developers access to public methods by allowing them to call methods on an object and prepare specific documentation, so they know how to do it.

Benefits for using encapsulation:

- Complexity is hidden and only certain data can be accessed.
- Manageable in terms of code improvement and updates.
- IP is protected as the more important code is hidden in a class.
- Security as private methods cannot be accessed from outside.
- Protects against phishing and mistakes.

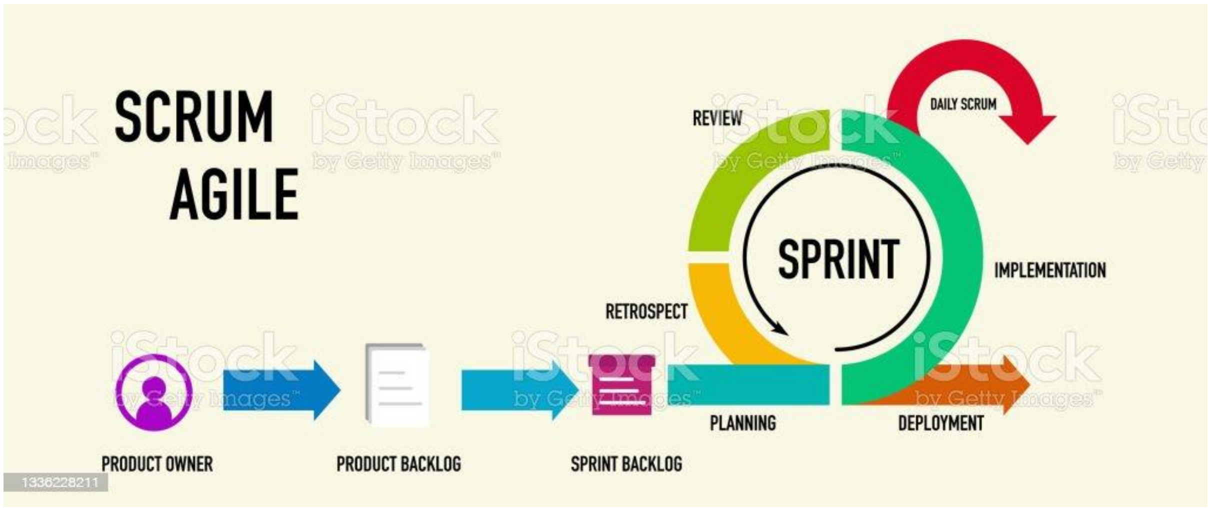
6. What is:
    a. Agile development?
    b. Waterfall development?
    c. How do they differ? Which is suited for which situation?

Once complete, please return to your instructor your answers! Remember:

- Justify and be critical of everything! This distinguishes a great answer from a good answer
- Analyse - why does x even exist? Who needs it or uses it? Who is it important to, what's the point of it at all?

Answer
Both Waterfall and Agile are two methodologies very popular in project management and describe the way teams collaborate on projects. Waterfall follows a linear approach where each phase of the SDLC (Software Development Lifecycle) must be completed before we can move to the next one. Agile is an iterative method, where the tasks are divided in small, achievable iterations, and runs in sprints. It allows different team members to work simultaneously on various parts of the project.

Agile methodology:



Agile model

Agile model approach relies on frequent interaction within teams and with stakeholders, has a high flexibility, is based on short-term deadlines, and depends on team initiative. Project development can be very long (sometimes years) and there can be many changes in the development during this time, including technology advancement or stakeholders' feedback which is greatly valuable and frequently requested. In this development environment, teams have a great input and can change the direction of the project. Thanks to this approach, people are self-directed, motivated, and productive.

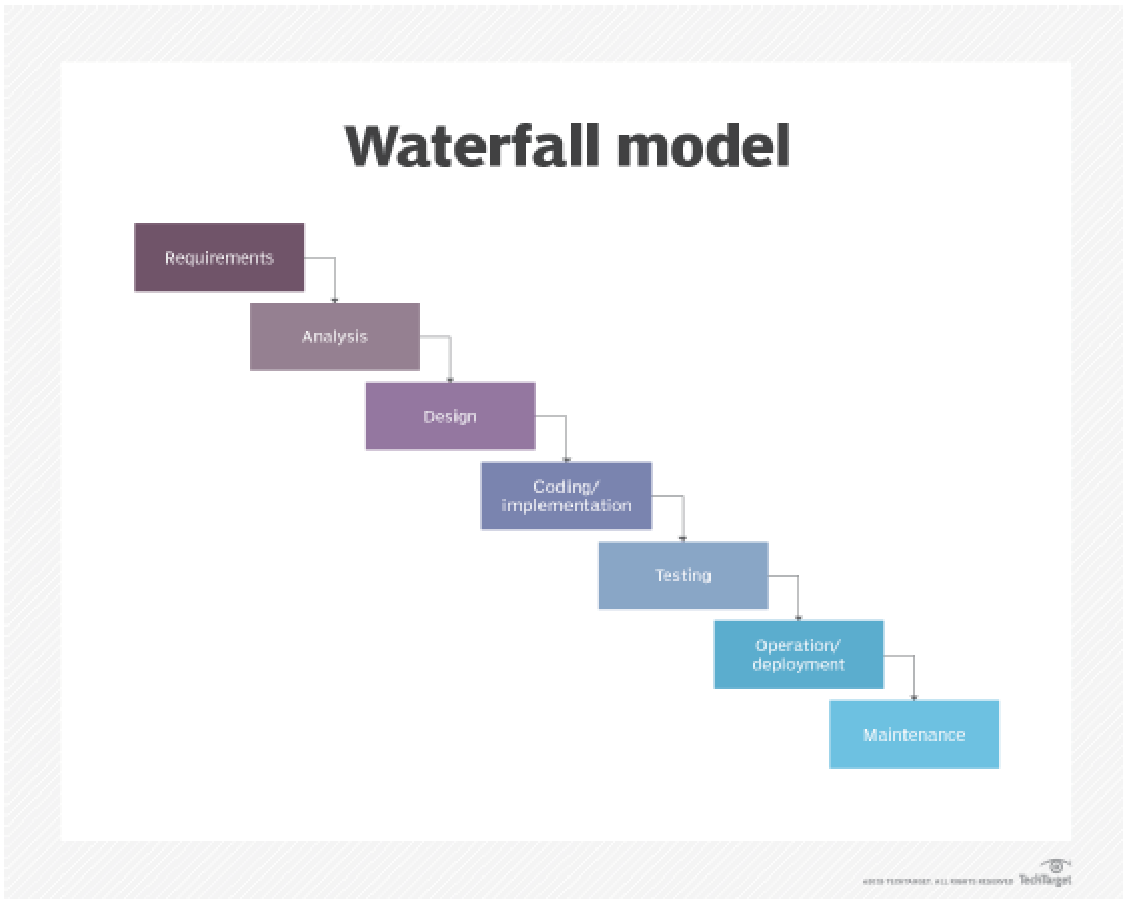
Advantages:

- Increased efficacy and productivity thanks to the short deadlines.
- Great flexibility in terms of product's direction.
- Lots of interaction with stakeholders and which helps the team to receive and implement feedback.

Disadvantages:

- Since team members work simultaneously on different parts of the project, if there are any changes in the requirements or during the process, one or more people can be affected.
- To progress to the following phase is not mandatory to have all deliverables done.
- In large teams, it can be difficult to monitor the progress especially if there are other departments involved or team members change jobs.
- Project deadline is hard to estimate, and it is dependent on changes.

Waterfall methodology:



Waterfall model

This model's approach is based on vigorous analysis from the beginning and sets a clear goal, with an established deadline. It has almost no flexibility and requires all deliverables to be completed to be able to move to the next stage/ phase.

Advantages:

- A clear plan of the project, from beginning until the end.
- During analysis and requirements phase, team spends lots of time to plan the project deliverables and timeline accordingly which will save time later.
- To proceed to next phase, we need to follow a structured flow which helps to always understand the exact position, for the team members.

Disadvantages:

- Sometimes it requires a longer time to progress to the next phase if not all team members completed the tasks as expected.
- Is hard to find errors and the process can be very lengthy in case we need to go through all the phases to discover from where the error comes.
- Since the project is determined at the beginning, it doesn't allow change or flexibility.
- Stakeholders are not directly involved and in case there is a disagreement at a later stage, it is very hard to go back.

Waterfall model is best suited for projects with a clear description and a specific deadline, where the product owner is confident that there will be no changes during the development process. Agile on the other side, is best suited for projects where the product owner expects changes along the way and testing or researching during development are important. In terms of budget, Waterfall projects tend to have one less flexible as the requirements are analysed during the planning and analysis phase. Agile’s budget is more flexible and adaptive to project’s needs since the project can change direction during development. Waterfall model is adopted in projects where the requirements play a key role, the project has a clear structure, and stakeholders have a minimum impact. In Agile model stakeholders’ involvement is very important and their feedback can change the project’s direction during development.