Unit 2

JVM - Java Virtual Machine: Platform independent and make Java bytecode. Implementation of JVM is JRE.

JRE - Java Runtime Environment

Encapsulation:

- The wrapping up of data and functions into a single unit is known as encapsulation.
- The data is not accessible to the outside world, only those function which are wrapped in can access it.
- These functions provide the interface between the object's data and the program. This can be like getters and setters or display functions.

Data Abstraction:

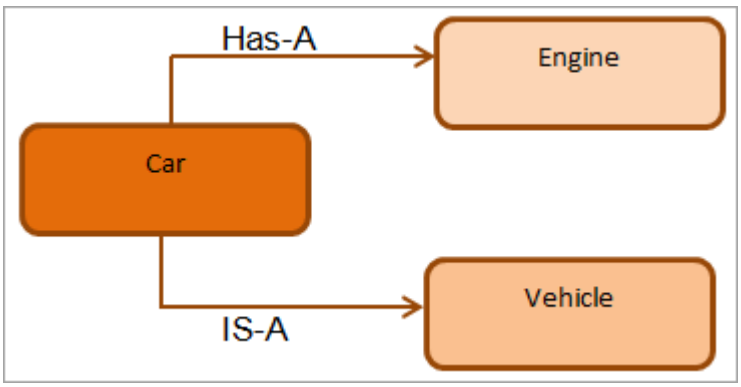
- Abstraction refers to the act of representing essential features without including the background details or explanations.

Composition:

- This represents the "has-a" relationship.

Inheritance:

- This represents the "is-a" relationship.



Class:

This is a way to define a new data type or structure. A class is a template and the object is an instance of a class.

Access modifiers:

- **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
- **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
- **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
- **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Access	Inside	Package	Child	Outside
Private	☑	✗	✗	✗
Default	☑	☑	☑	✗
Protected	☑	☑	☑	☑ (Only through a child)
Public	☑	☑	☑	☑

Abstract Class:

- Same like interface but has data types.
- These cannot be instantiated!!!

```
public abstract class ClassOne {

    public void printSomething()
    {
        System.out.println("Hello in abstract class");
    }
}


class InheritClassOne {

    public static void main(String[] args)
    {
        ClassOne obj = new ClassOne() {};

        obj.printSomething();
    }
}
```

> Hello in abstract class

Here it is instantiated by using a anonymous class. The {} after `ClassOne obj = new ClassOne() {};` is the anonymous class. Below is another way.

```
public abstract class ClassOne {

    public void printSomething()
    {
        System.out.println("Hello in abstract class");
    }
}


class InheritClassOne {

    public static void main(String[] args)
    {
        ClassOne obj = new ClassOne() {
              public void printSomething()
            {
                System.out.println("Hello in the wierdo thing");
            }
        };

        obj.printSomething();
    }
}
```

> Hello in the wierdo thing