Abstraction

Abstraction is a concept of having an entity that is specific for solving a task or a problem without letting users know the detail implementation of how problem is solved. For example, we can call a API and the API will give the result we need but we would not know how API generate the result.

In Java, Interface class is a way to achieve Abstraction. An interface class looks like below, which has the names of a interface methods that does not have a body. Let's say we want to have a object that can update the student info to the database.

```
interface StudentOps {
   void insert(Student);
   void delete(Student);
   void update(Student);
}
```

Then, the interface class would be implemented by a concrete class, which will implement all the interface methods and has a connect method that handle the implementation of db connection(This method does not and need not to be view by the user of StudentOps object).

```
class JdbcStudentOpsImpl implements StudentOps {
   public void insert(Student s) {
     // code
        connect ()
   }
   public void delete(Student s) {
     // code
        connect ()
   }
   public void update(Student s) {
     // code
        connect ()
   }

   public void connect(Student s) {
     // connect the jdbc
   }
}
```

When we use the object to update student info, we create an instance of interface by executing the constructor of a concrete class that implements the interface instead of use the *JdbcStudentOpsImpl* directly.

```
StudentOps studentOps = new JdbcStudentOpsImpl

studentOps.insert(std)
studentOps.delete(std)
studentOps.update(std)
```

In this way, the user who want to update the student info would not see how the db is connect and can still update the data. This prevent user from viewing the detail or wrongly use the

connect function. Besides, when we want to change to a different db connection method, we can create a new concrete class that implement *studentOps* to substitute old implementation without change the code of *studentOps.*

StudentOps studentOps = new odbcStudentOpsImpl

studentOps.insert(std)
studentOps.delete(std)
studentOps.update(std)

---

https://stackoverflow.com/questions/62112528/how-does-interface-helps-in-hiding-details -The example java code

Encapsulation
"Encapsulation is defined as the wrapping up of data under a single unit." The data/function should be only used or seen within a class if it is unnecessary for other part of code.

In Java, it could be achieved by "private" declaration. By declaring private before a variable or a function, the variable and function can only be access within the class or function and cannot be called outside of the class or function.

```
Class Birthday{
        Private String date;

        Public void setDate(int month, int day){
                date=Integer.toString(month)+"*"+Integer.toString(day)
        }
}
```

In the case above, the date can only be changed by called by the setDate function and cannot be directly changed through called the variable from the object. That's say the date variable in Birthday object should strictly follow the format "month*day", by using private variable, it prevent the user change the variable and not followed the format.