Example that demonstrates advantages of encapsulation

Since Attributes can only be accessed via Services such representation creates a strong information hiding mechanism – cannot change the object’s state arbitrarily, can only “ask” the Object itself to change it. This increases robustness of software.

Another advantage is that the implementation of each object is hidden. Consider the following example that demonstrates and motivates how OO-type encapsulation is hiding the implementation of the object from the outside world.

Consider the following scenario taken from the DRS domain: A student’s printed transcript has student’s name on it. Suppose two different people are working on implementation of Student-related and Transcript-related functionality.

Let’s start with

**Non-OOP implementation:**
The function that prints the transcript (call it `PrintTranscript`) must get and print the name of the student. A typical implementation in procedural languages would represent student-related information with a record structure with fields for each attribute and would look similar to the following:

```c
structure Student {
    id
    year-enrolled
    name
    address
    ...
}
```

Suppose we’re dealing with a particular instance of a Student structure (variable) `astudent` representing John Smith:

```c
astudent isa Student

astudent.name = “John Smith”
astudent.id = “ABC005”
astudent.year-enrolled = 1998
.....
```
To print the name of a student, the function PrintTranscript must include the following line

```
print (astudent.name)
```

Suppose now that the person working on the Student-related part of the system chooses to modify the way the name is represented. Instead of a single field called `name` which is eliminated, the name will be stored in three different fields: `firstName`, `middleName`, `lastName`, yielding the following (modified) structure

```
structure Student
{
    id
    year-enrolled
    firstName
    middleName
    lastName
    address
}
```

The statement `print(astudent.name)` used in the PrintTranscript is now invalid, because there is no field called `name` in structure Student anymore. The instance `astudent` now has the following fields instead:

```
astudent.firstName = “John”
astudent.middleName = “”
astudent.lastName = “Smith”
```

The function PrintTranscript will not work properly until the line that prints the name is replaced with the following three different lines:

```
print (astudent.firstName)
print (astudent.middleName)
print (astudent.lastName)
```

Thus, each time implementation of the Student changes, the rest of the system may have to be modified too! These changes create an overhead in having to

- communicate the change from one programmer to another, and
- re-implement parts of the system

**In the OO approach** the access to the student’s name is implemented via a service called, say, `getName` of the Student class. The Print service of the Transcript class can get access to a student’s name only by using the `getName` service, and thus would print it using the following statement
print(student.getName)

When implementation of the Student class changes eliminating the name attribute and replacing it with firstName, middleName and lastName, the only other part of the system that must be modified is the getName method of the Student - instead of returning the value of the attribute name it must now combine the three attribute values into a full name and return it.

The transcript printing function does not have to change at all!

So information hiding results in lower volatility of software to changes, greater robustness as well as less communication between the developers (thus less time spent on implementation).