0:15

Now, it's time for you to learn about encapsulation. Encapsulation is a fundamental design principle in object oriented modeling and programming. There are many things that you can represent as objects. For example, you could represent a university course as an object. The course object can have many attribute values, like the specific number of students enrolled, credit value, and prerequisites, as well as specific behaviors dealing with these values. And the course class defines the essential attributes and behaviors of all the course objects. Encapsulation involves three ideas. As the name suggests, it's about making a sort of capsule. The capsule contains something inside, some of which you can access from the outside, and some of which you cannot. First, you bundle attribute values or data, and behaviors or functions, that manipulate those values together into a self-contained object. Second, you can expose certain data and functions of that object, which can be accessed from other objects. Third, you can restrict access to certain data and functions to only within that object. In short, encapsulation forms a self-contained object by bundling the data and functions it requires to work, exposes an interface whereby other objects can access and use it, and restricts access to certain inside details. You naturally bundle when you define a class for a type of object. Abstraction helps to determine what attributes and behaviors are relevant about a concept in some context. Encapsulation ensures that these characteristics are bundled together in the same class. The distinct objects thus made from a particular class, will have their own data values for the attributes and exhibit resulting behaviors. You'll find that programming is easier when the data, and the code that manipulates that data, are located in the same place. An object's data should only contain what is relevant for that object. A student would only contain relevant data for themselves, like the degree program. This is, as if the student object knows its degree program like a real student would. A course object would know a list of students taking it. The professor object would know a list of courses the professor teaches. And none of these types of objects would contain a list of other courses offered, because that is not relevant data for them. Besides attributes, a class also defines behaviors through methods. For an object of the class, the methods manipulate the attribute values or data in the object to achieve the actual behaviors. You can expose certain methods to be accessible to objects of other classes, thus, providing an interface to use the class. For example, a course can provide a method to allow a student to enroll in the course, or a course that a professor is teaching, can provide a method that allows the professor to see the list of students in that course. Encapsulation helps with data integrity. You can define certain attributes and methods of a class to be restricted from outside to access. In practice, you often present outside access to all the attributes, except through specific methods. That way, the attribute values of an object cannot be changed directly through variable assignments. Otherwise, such changes could break some assumption, or dependency for the data within an object. As well, Encapsulation can secure sensitive information. For example, you may allow a student class to store a degree program and grade point average, GPA. The student class itself could support queries involving the GPA, without necessarily revealing the actual value of the GPA. For example, the student class could provide a method that tells whether the student is in good standing for the degree program, which uses the GPA and the calculation, but never reveals its actual value. Encapsulation helps with software changes. The accessible interface of a class can remain the same, while the implementation of the attributes and methods can change. Outsiders using the class, do not need to care how the implementation actually works behind the interface. To explain this, I'm going to use the university example again. Let's say, a professor wants a student to calculate and declare their GPA before getting admitted into a course. This is an action that the student may perform. However, there could be many different ways that this action could be done. A student might fill out a paper form and hand that to the administrator, who would check the paper files, write it a list of courses and grades for that student, from which to calculate the GPA. Or, this action may have been automated, so that the student would fill out the paper form and hand it into an administrator, who would check the computer database for the GPA. Or the student can go online, and get the GPA themselves using a student information system. All these different ways will still achieve the end goal, reporting the GPA to the professor. As you can see, the professor does not have to care how the student gets their GPA, a student is the only one that really needs to know how to do it. You can apply this notion when you program software. For example, suppose a student class has a method to return its major when called. The actual steps to retrieve the major does not need to be known by any other class. In programming, this sort of thinking is commonly referred to as, Black Box Thinking. Think of a class like a black box that you cannot see inside for details about, how attributes are represented, or how methods compute the result, but you provide inputs and obtain outputs by calling methods. It doesn't matter what happens in the box to achieve the expected behaviors. This distinction between what the outside world sees of a class, and how it works internally is important. Encapsulation achieves what is called, the Abstraction Barrier. Since the internal workings are not relevant to the outside world, this achieves an abstraction that effectively reduces complexity for the users of a class. This increases re-usability, because another class only needs to know the right method to call to get the desired behavior, what arguments to supply as inputs, and what appear as outputs or effects. In the real world, if I ask you to buy me a soda, you can get the soda in many ways. You can go to the vending machine and buy one, or you can drive to another city and buy one there. The input and output is the same. I ask you to buy me a soda, and you give me a soda. I don't need to know the details of how you got it. Encapsulation is a key design principle in achieving a well written program. It keeps your software modular and easier to work with. It also keeps your classes easy to manage, whose behaviors are accessed like black boxes.