Transcript: Encapsulation

>> Now let's talk about the concept of encapsulation. A client program, think of the main program, can access and modify an object's state through the public methods. For example, if we wanted to set the face value on an object, dieObj, for example. You could just call that method, with the parameter 6, and that would set the face value.

What we don't want to happen, is to make it, to allow the, program to actually set face value directly. The following statement here, dieObj.faceValue, references the field itself, and here we're assigning it to 6. Normally we want this to, cause a compile time error, otherwise the class violates encapsulation.

And, and the way we do that, we just make sure that faceValue is not accessible outside the class. So in general, we want to think of an encapsulated object, as a black box. That is, its inner workings are hidden from the client, the client invokes the interface methods, To actually get at the data, for example, here, this black box denotes the object.

It's got methods and it's got the data, the data being the instance field. And the client, to get at the data, actually needs to go through the methods. So it could call the getters and setters and so on if the, class wants to allow the client to actually get the data at all, and usually there's, there's some of that.

So how do we make sure that we don't violate encapsulation? Well, we have access modifiers, and they define where an instance variable, or a method can be accessed. Java has three access modifiers, public, protected, and private. Protected, involves inheritance, and we'll talk about that later when we get to that topic.

But all, otherwise, all the fields and methods of a class, collectively called members, as well as constructors. They can each have an access modifier, so they can be declared public, protected, or private, or it can be left off altogether. So members of a class that are declared public can be referenced anywhere, just like the, the name public implies.

Now, public instance variables, that would violate encapsulation, so that, that's a no-no. So we, we normally declare all of our instance variables private. Now, so a class of anything declared private, and that could be a method or an instance variable can only be referenced within the class. So that's what we definitely want for instance variables.

Transcript: Encapsulation

Now, members that are declared with no modifier, as in we didn't declare them private, public or protected, they have default access and that means they can be referenced within the class and also by classes in the same package. Now if you don't have a formal package, Java has a default package and that is the same folder that the classes are declared in.

So keep that in mind, all classes in a given folder, if they are not in a specific package which ours won't be they can actually access any field in a class that's not declared to be private. Now, so looking at visibility, here for methods, the idea of service methods or those that are public, they offer useful behavior our example of the die class included role, and also the getter and setter for the face value and it's to string method.

All those were declared public. Support methods, if there are any, and, we'll have some at some point, but probably not for a while. These can be declared private and only be used by other methods in the class. This would occur when, methods get too large or you've got multiple methods that use to same code, maybe some intricate calculation.

And so you could put that in a method by itself and all the methods inside the class could call that method, but you wouldn't want the public using it. And so, again, you would make that support method a private, that would be the idea there. Now, the Java API, when you look a the API for particular class like string, all the methods and constructors, everything you see there, these are the public ones.

There could be lots of private things there, but the API only shows the public items. So, here is sort of an overview. If we've got instance variables, they should definitely not be public. Because public violates encapsulation, they should be private to enforce encapsulation. And methods, if it's a service method, like roll, for our die, it, it should be public.

But if you've got some kind of support method in there, it should probably be, be private. That's the idea there. Now, because instance data is private, a class may provide methods to access and modify the data. And these are what we call our getters and setters. An example there is getFaceValue in the Die class.

A mutator method is one that changes a value. It's sometimes called a setter. Again, the accessor method is a getter. It just gets a value, doesn't change it. A mutator method on the other hand changes things. It mutates it, therefore we call it a setter. An example there is setFaceValue in Die.

Transcript: Encapsulation

Now the names accessor and mutator methods usually take the form of getX and setX, where X is the name of the field. Usually you would actually capitalize the name of the field to to show it, it being a separate word like, like up here in setFaceValue. Face value with a lower case F was the name of the field but we capitalize the face value when we put it together in a setter.

Now you could restrict mutators. So we've got the setters and and we, we may not you know wanna allow another class to do this. So not, not all classes will have getters and setters but but many will. For example, in the string and scanner class n-neither of those classes have, have setters.

And then finally, here, if you've got a setter you could use an if-else statement that we covered in course notes 2 and, and and probably the 5 module, so at module 2 and 5 there, to provide mutator restrictions on the field. That is, it can take on only specific values.

For example, for a setFaceValue, when we call that method, we can have it checked to see if the parameter coming in is greater than or equal to one, and less or equal to six. If that's true, we would go ahead and assign the field and we would return, true.

If it's not within that range, we would not assign the field and just return, false. That would let the calling program know if the set actually set a value, if it comes back false, you'd know that it, it didn't set a value.