Controlling access to class members - Encapsulation

Terminology

- **Access Modifier**
  - determines access rights for the class and its members
  - defines where the class and its members can be used

Why use these

It is important in many applications to hide data from the programmer

E.g., a password program must be able to read in a password and compare it to the current one or allow it to be changed

But the password should **never be accessed directly!**

- **public class Password {**
- **public String my_password;**
- **:**
- **}**
- **Password ProtectMe;**
- **:**
- **ProtectMe.my_password = "backdoor"; // this is bad**

Access Modifiers

- Member modifiers change the way class members can be used
- *Access modifiers* describe how a member can be accessed

Modifier	Description
(no modifier)	member is accessible within its package only
`public`	member is accessible from any class of any package
`Protected`	member is accessible in its class package and by its subclasses
`Private`	member is accessible only from its class

public vs. *private*

- Classes are usually declared to be *public*
- Instance variables are usually declared to be *private*
- Methods that will be called by the client of the class are usually declared to be *public*
- Methods that will be called only by other methods of the class are usually declared to be *private*

Public etc

- public means that **any class can access the data/methods**
- private means that **only the class can access the data/methods**
- protected means that only **the class and its subclasses can** access the data/methods

	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World
public	+	+	+	+	+
protected	+	+	+	+	
no modifier	+	+	+		
private	+				

+ : accessible blank : not accessible

Information Hiding

- To drive a car, do you need to know how the engine works? Why?
- `println` method
  - need to know ***what*** the method does
  - but not ***how*** `println` does it
- Provide a more abstract view and hide the details

Defining Encapsulation

- *Encapsulation* is the process of hiding an object's implementation from another object, while presenting only the interfaces that should be visible.
- Encapsulation is the technique for packaging the information in such a way as to hide what should be hidden, and make visible what is intended to be visible.

Encapsulating a Class

- Members of a class must always be declared with the minimum level of visibility.
- Provide *setters and getters* (also known as accessors/mutators) to allow *controlled* access to private data.
- Provide other public methods (known as *interfaces*) that other objects must adhere to in order to interact with the object.

Accessors and Mutators

- Accessor methods—public methods that allow attributes (instance variables) to be read
  - Get methods are also commonly called accessor methods or query methods.
  - Check to make sure that changes are appropriate.
  - Much better than making instance variables public

- Mutator methods—public methods that allow attributes (instance variables) to be modified
  - Set methods are also commonly called mutator methods, because they typically change an object's state—i.e., modify the values of instance variables.
  - private attributes (instance variables) with public accessor and mutator methods.

Setters and Getters

- Setters and Getters allow controlled access to class data
- *Setters* are methods that (only) alter the state of an object
  - Use setters to validate data before changing the object state
- *Getters* are methods that (only) return information about the state of an object
  - Use getters to format data before returning the object's state

Encapsulation ensures that structural changes remain local

- Changes in the code create software maintenance problems
- Usually, the structure of a class (as defined by its fields) changes more often than the class's constructors and methods
- Encapsulation ensures that when fields change, no changes are needed in other classes (a principle known as "locality")

END