1. A is-a relationship between classes involves inheritance, where one class(The child) directly extends and adds to the functionality of another(The parent) without modifying the parent class, example: A Disk is a Circle. The has-a relationship is where one class utilizes certain functions or methods of another class without being an extension of the class like with inheritance, this relationship was used in this chapter in the form of client codes, example: The Hockey class has the Puck class, but it doesn't inherit the Puck class.
2. Both methods will be available to be used by an object, this is part of how inheritance works, the object can use the parent methods as well as any new methods included with the child.
3. An abstract method comes from an abstract class, this method is not complete, it only contains the name, return type, and parameters, when a non-abstract subclass extends the abstract class, it must override the abstract methods that are inherited with complete ones with the same name, return type, and parameters. Method overriding is when the parent class already has a completed method, while it can be used by any subclass, depending on needs, a subclass may override its parents method to fulfill any needs not met by the parent method, but it doesn't have to.
4. An abstract class is a class made to contain abstract methods that are not fully complete, though it can also contain concrete methods with implementations. An interface is like a more flexible version of an abstract class, it contains abstract methods that must be implemented, but it can also have default methods and constant variables. Another key difference is that while a subclass can only inherit one parent class, one class can implement multiple interfaces at the same time.
5.
6. A: doThat is an abstract method
   B: Wo is an interface that contains a single method
   C: doThat is implemented in Roo because it implements the Wo interface, when an interface is implemented in a class, all of its abstract methods must be overridden.
   D: Based on the code of the Roo class, a Roo object could only use the doThis and doThat methods.
   E: The doThis in Roo overrides the doThis in Bo.
   F: As far as I can tell, nothing, it seems like it was supposed to be changing the value of the x in the Bo class, but since it is a private variable, it can only be changed by the parent class.
   G: Yes, by using super.method, it communicates that the code should use the parent method instead of the child method.
   H: Yes, quite simply actually, by implementing the call for the doThis method within the child method, specifying that it should be the parent method using super.doThis, it should be very easy to call upon the doThis method in Bo.