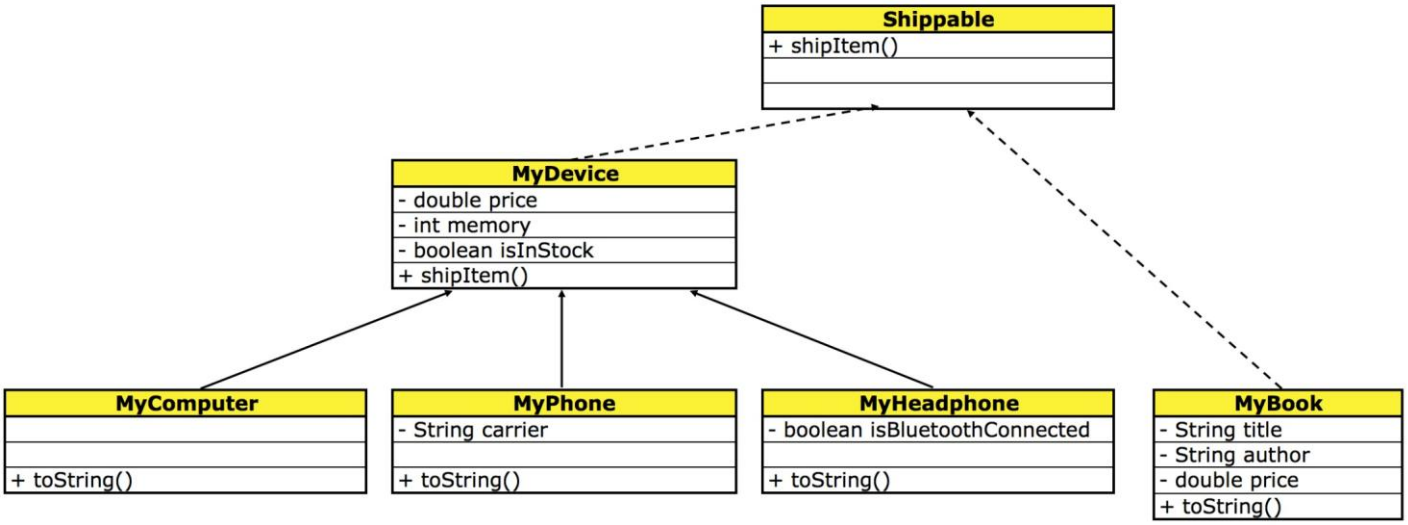
LAB 63B
AP Computer Science
Inheritance & Interfaces

The `MyDevice` superclass in your inheritance hierarchy can accommodate electronic devices since it includes an instance variable for the amount of memory a device has (i.e., 100 GB, 1TB, etc.) However, what if to add non-electronic products such as paperback and hardcover books to our inventory? These books don't have a "`memory`" instance variable associated with them. How will we be able to group them together with the other electronic items in our shopping cart?

We cannot make a `MyBook` class a subclass of `MyDevice` since paperback and hardcover books don't have an instance variable for its "memory" that can be factored out into the `MyDevice` superclass. Besides artificially adding a "memory" instance variable into classes that represent books (which is not a good idea), we again have a problem if we wish to group electronic and non-electronic items together in an array or `ArrayList` and call the `shipItem()` method as we have done previously.

Luckily, this is a common problem and Java's solution to it is an "Interface". An interface is a mechanism used to group classes together by common methods (functionality) instead of instance variables (state). In our example, we can group the classes together by their "`shipItem()`" method since all items, electronic or not, will have this method. This makes sense since this is the method that we would probably want to call when traversing a `shoppingCart`, `speedOfLightShipping`, or `standardShipping` list. We will factor out the `shipItem()` method into an "Interface" called `Shippable` as shown below:



So now we are ready to include non-electronic devices in our online store. Here are the design specs for adding this new item to your inventory:

a. Write a new class called "My`Book`" that includes instance variables for the title of the book, the author of the book, price of the book, and whether or not the book in stock. Include a four-argument constructor and a `toString()` method for the class.
b. Create an instance of your favorite book in a driver class using its three-argument constructor.
c. Factor the "`shipItem()`" method in all classes out into an interface called "Shippable".
d. Redefine all lists to be of type "`Shippable`".
e. Add your newly created book to the "`shoppingCart`".
f. Traverse all three lists (`shoppingCart`, `speedOfLightShipping`, or `standardShipping`) and call the `shipItem()` method for each item in each list.