**The difference between abstract classes and interfaces**

When to use abstract classes and when to use interfaces?

This is a tricky question. In my opinion there is a basic rule that will work for you almost every time.

**When to use an abstract class?**

If there are some common properties and methods in a group of classes, we create a new class and put all those common properties and methods in it and make it a parent of all those classes in that group.

That parent class will have a parent child relationship with its child classes.

**Now what's the abstract method and abstract class?**

Abstract method is a common method in all child classes, but every child class need its own implementation of the method, we put that method in a parent class but we cannot give its implementation there. So, a method without implantation in a parent class qualifies to be an abstract method. You have to mark that method as abstract and when you mark a method as abstract, It becomes child classes responsibility to give its own implementation of that method.

If there are one or more abstract methods in a parent class you have to declare the whole class as abstract. So, abstract class is nothing but a parent class with abstract methods.

**Point to be noted:**

You don't have to explicitly make a class as abstract. When you work your class diagram, abstract classes are automatically identified by generalization.

**So, use abstract classes or inheritance where certain methods are required for a group of subclasses or Sibling classes (The child classes that are inheriting from one parent class).**

When to use interfaces ?

Interfaces lie outside the inheritance hierarchy. So, parent-child relationship doesn't exist between interface and the classes that implements it. Like abstract classes interfaces are also used for the common methods in the classes. but the main difference is, "those child classes may be from different inheritance trees or those child classes may be from same inheritance tree".

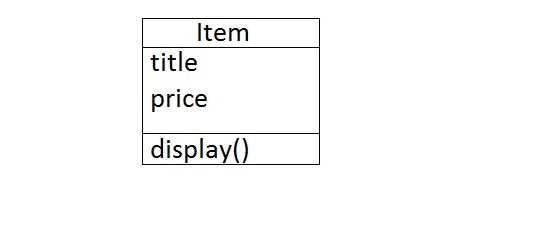
**So, Use Interfaces when a certain functionality is required by other classes regardless of where those classes are in the inheritance tree.**

Example of Abstract Classes and Interfaces

For example, you are working on an online shop that sells Text books, Books, Audio Books, Movies, Music, Games and Stationary. Here I am only going to discuss Books and Movies just to make this example very simple.

Book are available in two formats paper back and pdf and movies are also available in two formats DVDs and Watch Online.

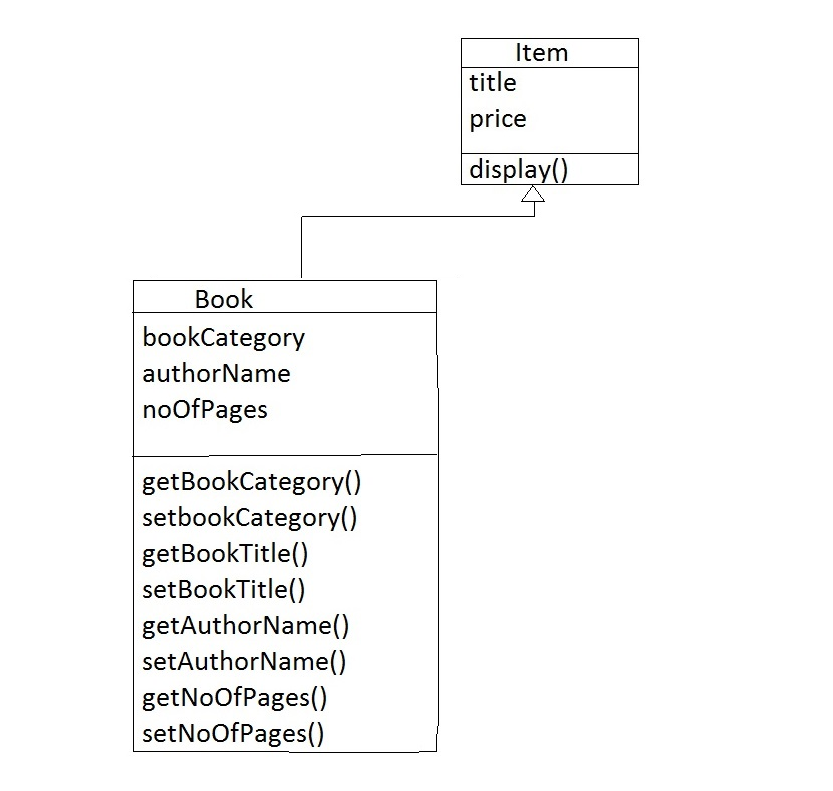
Now there may be four items in the shop "PaperBackBook", "EBook", "DVD" and  "OnlineMovie". Every item has a title and price and it needs a display() method to show its information to user. So, Item will be the parent class for all of them and all these common properties and methods will go in it.



Then there are some properties and methods that are common in both book formats Like common properties are (bookCategory, bookTitle, authorName, noOfPages).

Common methods are (getBookCategory(), setbookCategory(), getBookTitle(), setBookTitle(), getAuthorName(), setAuthorName(), getNoOfPages(), setNoOfPages()).

Here is our Book class.



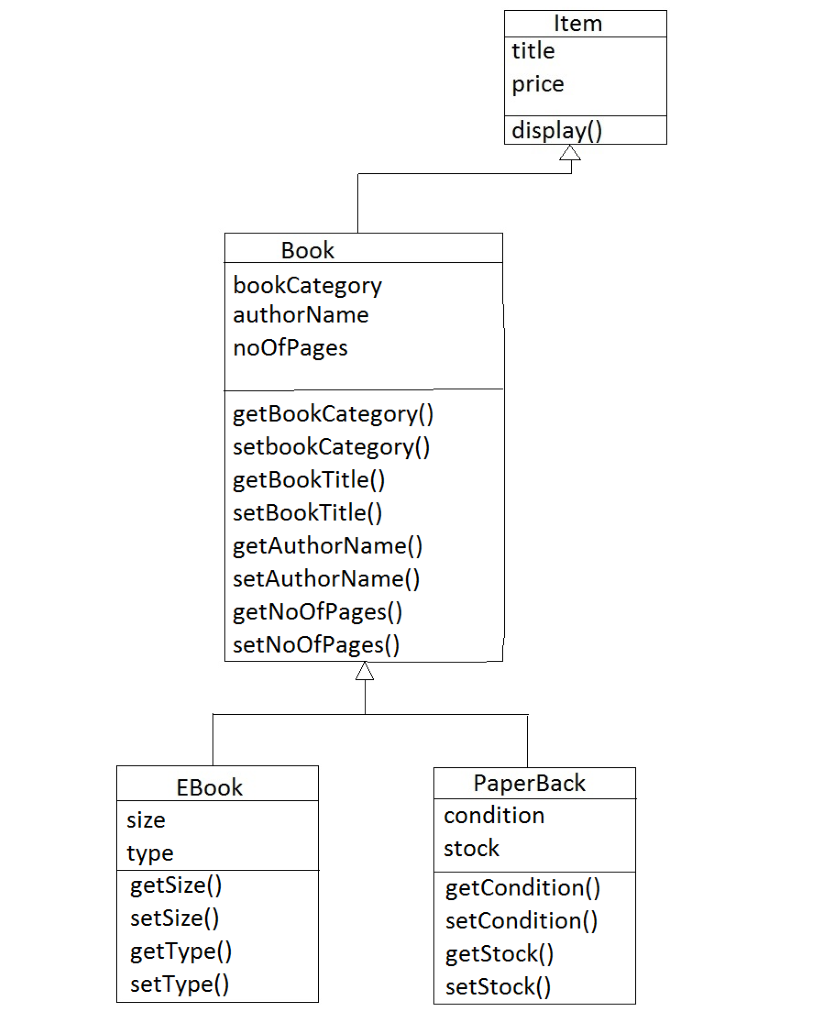
Now  "PaperBackBook" and "EBook" will be the child classes of this "Book" class.

**Properties and methods related to "PaperBackBook" class.**

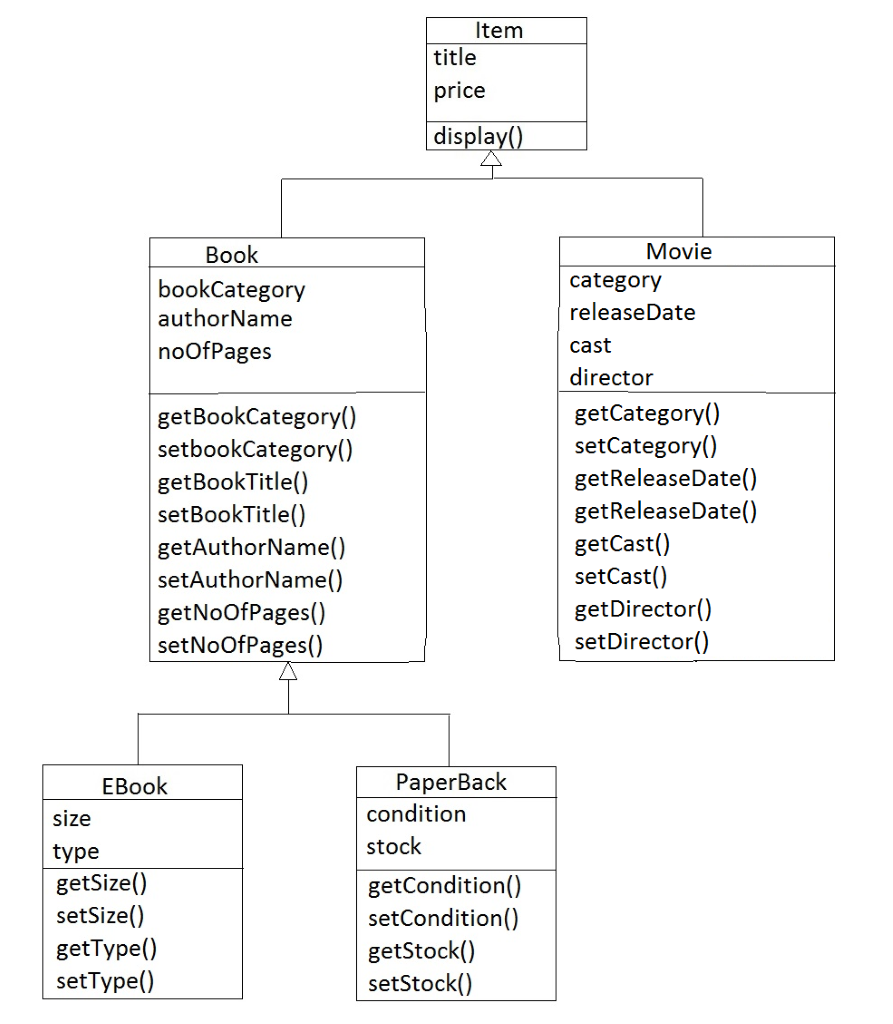
Its properties will be "condition" (A "condition" of a book may be used or new) and "stock". Its methods will be (getCondition(), setCondition(), getStock(), setStock()).

**Properties and methods related to "EBook" class.**

Its properties will be "size"(like 1823 kb), "type"(EBook is available in different types like pdf, epub). Its method will be (getSize(), setSize(), getType(), setType()). 

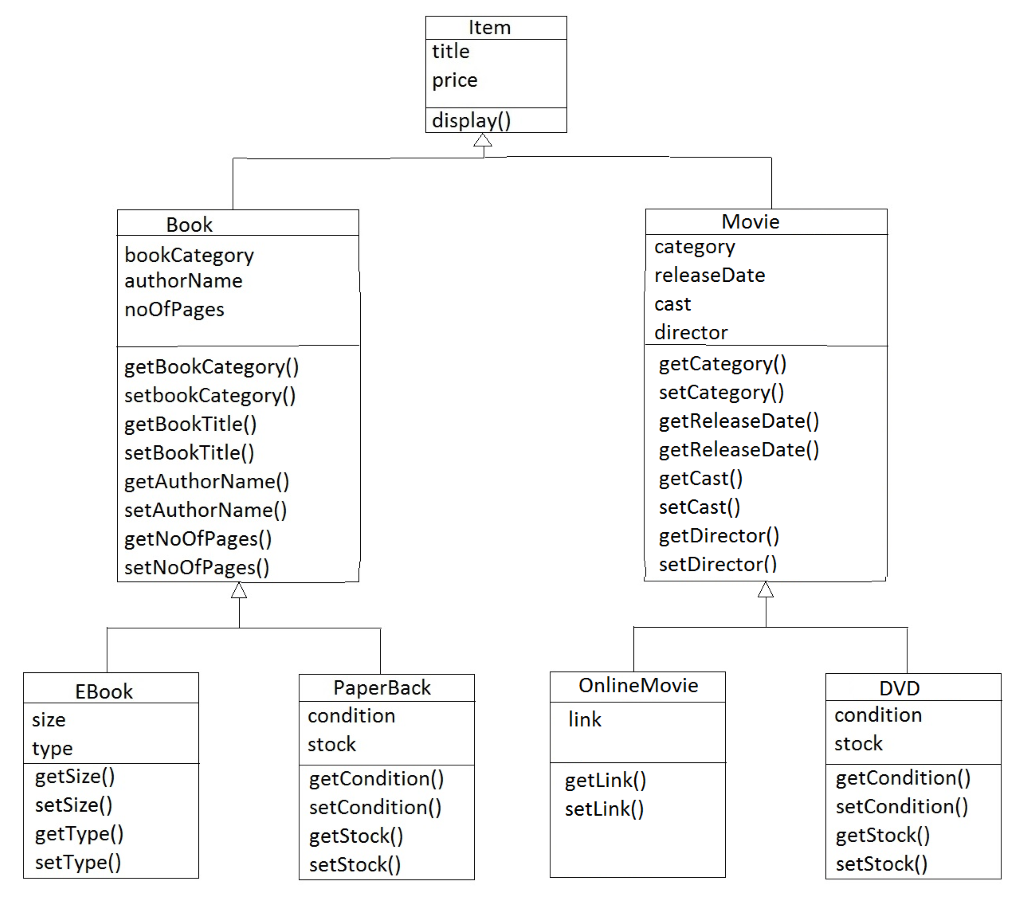


Properties ("category", "releaseDate", "cast", "director") and their getter and setter methods that are common in "DVD" and  "OnlineMovie" will go in movie class.



Properties related to "OnlineMovie" class will be "link" (Watch online and download link for the movie) and its getter and setter methods.

Properties related to "DVD" class will be "condition" and "stock" and their getter and setter methods.



This is our class hierarchy. Top class is "Item" class which has only one method display. So, all our four child classes "EBook", "PaperBack", "OnlineMovie" and "DVD" will inherit this method.

As you can see all these four class are different so, they need their own version of display method.

For example, if you call display() method of an "EBook", It should display this book information( bookCategory, authorName, noOfPages) and the information that is specifically related to this "EBook" like its type is pdf and its size is 1198 kb.

display() method of "PaperBook" should first check that the book is in stock or not and if the book is in stock then it should show the bookCategory, authorName and noOfPages and then the condition of the book(It is used or new).

Display() method of "OnlineMovie" and "DVD" will also be different.

So you can't give implementation of display() method in "Item" class, we have to mark it as abstract. By marking a method as abstract in parent class, we force child classes to provide its implementation.

In future, If we add any more child classes, like we introduce audio books in our shop or we start selling music cds, all those new classes has to provide implementation of this display() method.

**If a class has an abstract method, we have to declare the whole class as abstract.**

So "Item" class will be abstract. Its child classes "Book" and "Movie" are inheriting this display() method, but again we are not giving implementation of this method because its implementation is related to their child classes. So, these two classes will also be abstract. So, here you will have three abstract classes.

**Abstract classes make more sense when you can make "IS-A" statement. "DVD" IS-A "Movie" and "Movie" IS-AN "Item". An "EBook" IS-A "Book" and a "Book" IS-AN "Item".**

**OR**

**Abstract classes make more sense when you are defining an object itself. Like when we define "PaperBack", all the properties and method in its parent classes "Book" an "Item" are part of "PaperBack".**

When and where the interfaces come?

The basic purpose of the shop is to sell these four items "EBook", "PaperBack", "OnlineMovie" and "DVD". So, we need to add a method sellable in all four classes. You must be thinking that this method should go in item class but here is a little twist, we make an interface "Sellable" and put the "sell()" method in it.

**Why?**

When you can make a statement that an object "Can-Also-do" that or an object "Is-Also" that, use an interface.

For example,

"DVD" Is-An "Item" which Is-Also "Sellable" and "Deliverable".

"OnlineMovie" Is-An "Item" which Is-Also "Sellable" and "Watchable".

"EBook" Is-An "Item" which Is-Also "Sellable" and "Downloadable"

"PaperBack" Is-An "Item" that Is-Also "Sellable" and "Deliverable".

Here "Is-Also" statement shows that, we need interface.

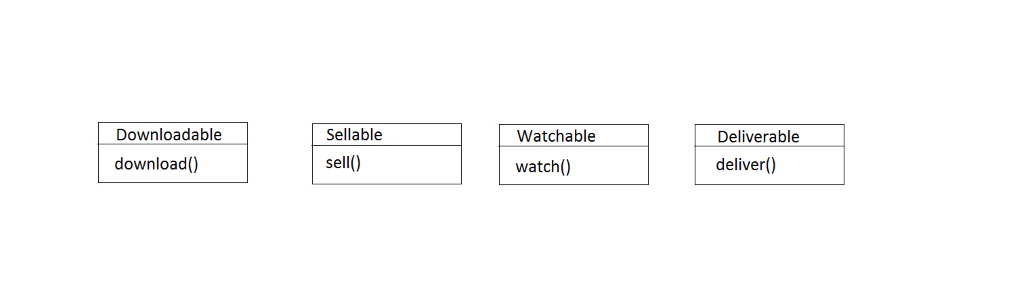
**So, Use Interfaces when a certain additional functionality is required by an object.**

**Or**

**A certain additional functionality is required by other classes regardless of where those classes are in the inheritance tree.**

Here selling, delivering, downloading, watching online are additional functionalities. For example, when you are going to define "PaperBack" class, properties and methods that are specifically related to "PaperBack" will go in this class or its parent classes. All other additional functionalities will go in interfaces like "PaperBack" needs "Sellable" and "Deliverable" interfaces.

So, our interfaces will be "Sellable", "Deliverable", "Downloadable" and "Watchable".

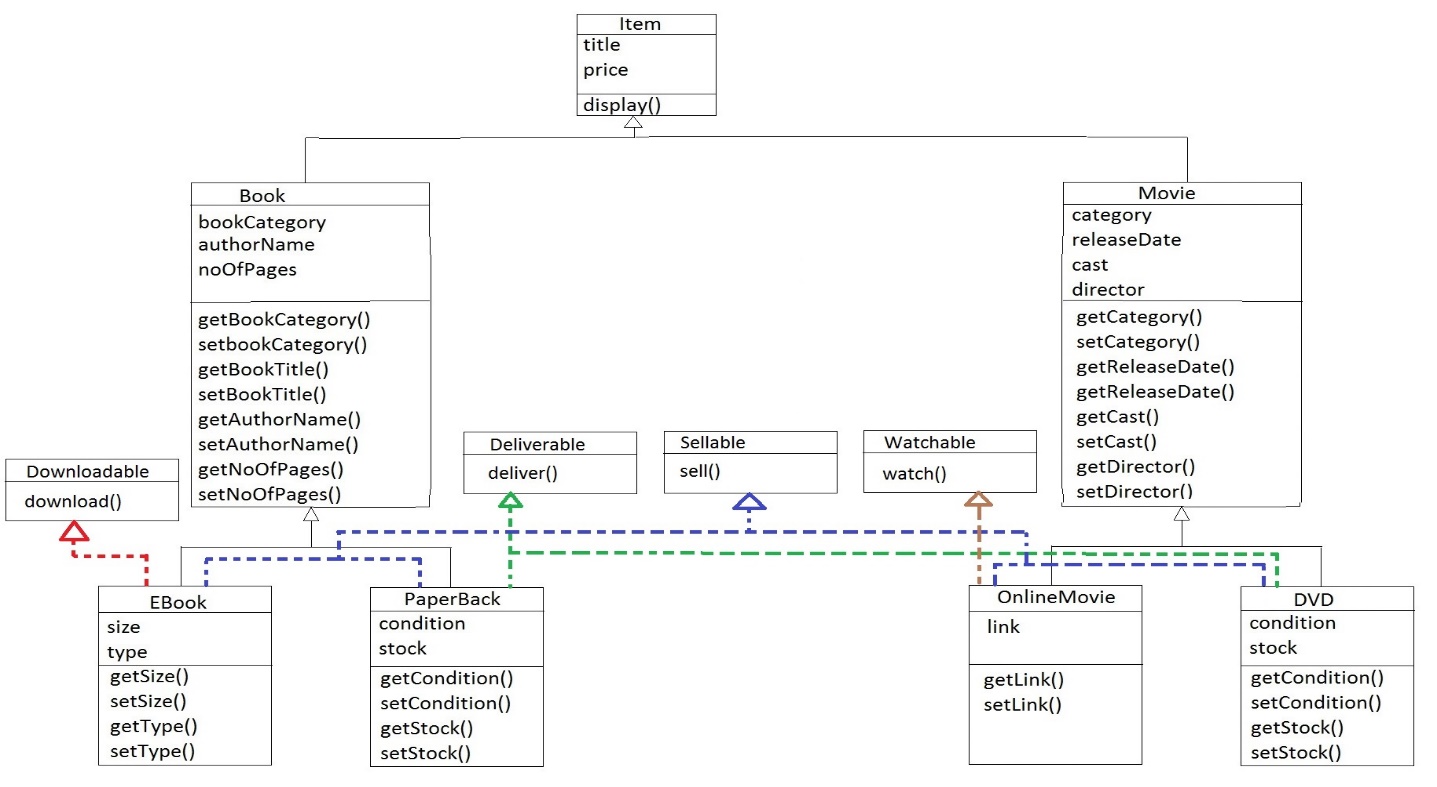


All the items are sellable so they should implement "Sellable" interface,

Point to be noted "All the items are Sellable", then item class should implement this interface, but there is one problem, in future there may be a possibility that we introduce free items like free pdf books, or free movies.

So, parent classes "Item", "Book" and "Movie" cannot implement "Sellable" interface, our child classes will implement this Sellable interface.

Our class diagram will look like this.



Book" is Sellable and Downloadable, "PaperBack" is Sellable and Deliverable, "OnlineMovie" is Sellable and Watchable, "DVD" is Sellable and Deliverable.

Form this class diagram, you can see that **A class can implement more than one interface.**

Extendable Interfaces.

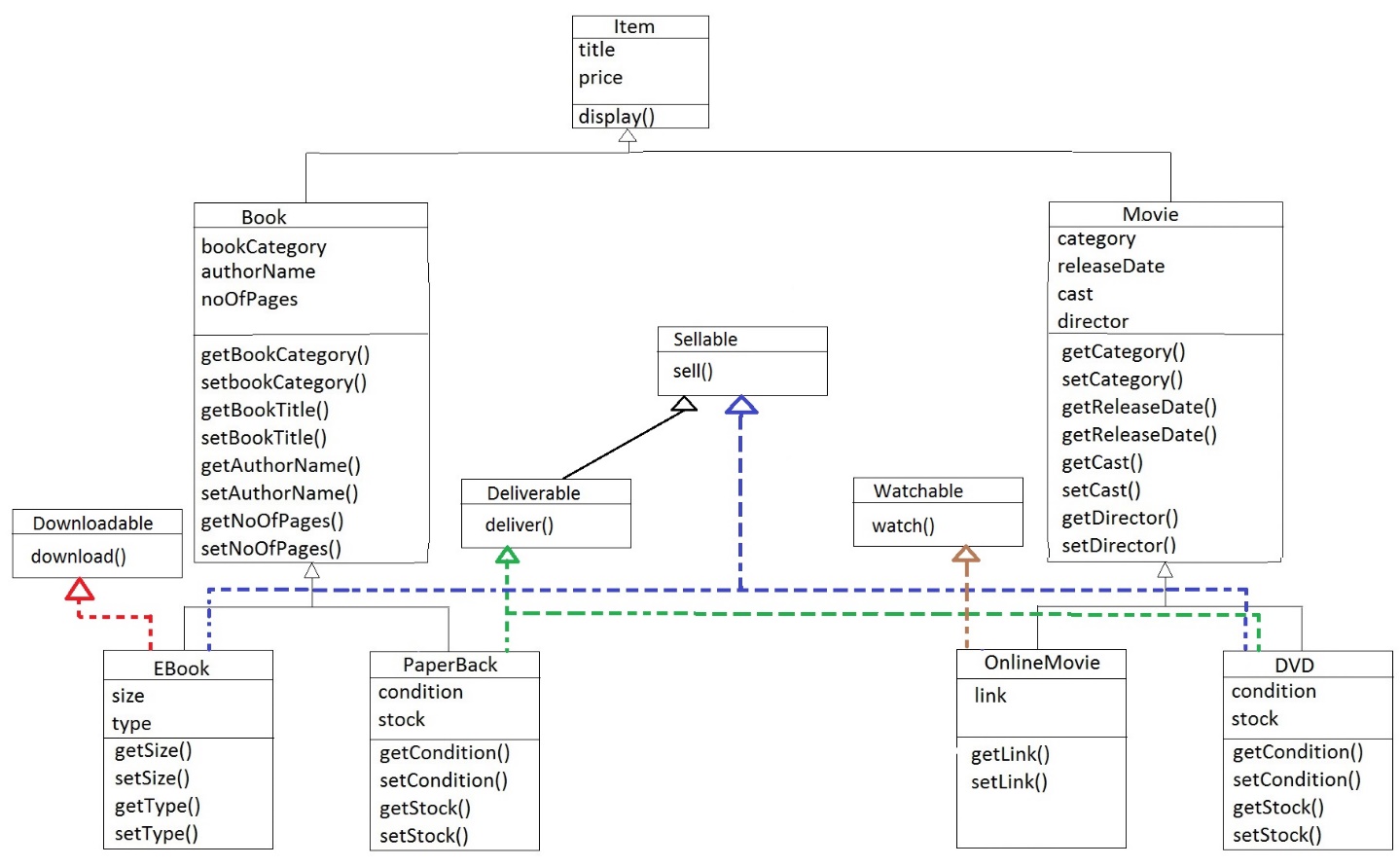
An item can only be delivered to the customer when it is sold. It is not possible to deliver an item without selling it. The task of delivering an item is depending on the task of selling it.

So, we have made a little change here, now our Deliverable interface extends the Sellable interface.

If some class will implement deliverable interface, It has to provide implementation of both methods sell() and deliver().

The same is not true for Downloadable and Watchable interfaces because in future there may be some items which will be free to download or watch. So, Downloadable and Watchable are not extending sellable interface.

**Interfaces can be extended like classes using the**[**extends**](http://php.net/manual/en/language.oop5.inheritance.php)**operator.**



That was a little example about difference between abstract classes and interfaces and you have also learnt that a class can implement multiple interfaces and an interface can extend another interface.

Now the difference between abstract classes and interfaces in code.

**The differences between abstract classes and interfaces in PHP**

|  |  |
| --- | --- |
| Interfaces | Abstract Classes |
| An interface can only contain abstract method and constants. | Abstract classes are classes that contain one or more abstract methods. It may also contain concrete methods, constants and properties. |
| All Interface methods must be public. | An abstract class may contain public, private and protected methods. |
| A class can implement more than one interfaces. | A class can only extend from one base class (That base class may be abstract or concrete.) |