Exercises Module 18 - Polymorphism and Abstract Classes

Task 1

We are using the Product class as a superclass. This is done for two purposes, both as a place to have reusable functionality, and to be able to handle books and movies in a more general way as products. But should we be able to instantiate new objects of type Product? What does a Product that is only a Product look like?

If the Product class is only meant to be used as a general way of using "real" objects like books and movies, and as a place to put reusable functionality, then it shouldn't be possible to instantiate a Product object.

Prevent this by declaring the Product class as abstract.

Did anything else need to change in the program? What happens now if you try to instantiate a new Product object with the new keyword?

What about the Book class, being used as a superclass to ChildrensBook? Should it be abstract? Why not?

Task 2

It is an advantage to be able to handle objects in a more general way by refering to them by a superclass reference. But the disadvantage with this is that you don't have access to the functionality in the subclasses. You can cast the object from superclass reference to subclass reference to be able to access this functionality but do we really have to go through all this hassle with instanceof and casting? Isn't there a more efficient way?

If the subclass overrides a method in a superclass, a method call to the superclass method will actually run the most specific implementation of the method that is available in the object. If the object is really a subclass, the method in the subclass will execute even though the method reference is the type of the superclass.

Can we use this functionality to print the details of books and movies without having to test the type with the instanceof operator, and without casting the objects to their "real" types?

Hints:

Add a method printDetails in the superclass. It could print out something from the Product or just do nothing at all.

Change the printDetails methods in Book and Movie to override this printDetails method (change their names).

Iterate over the Product list but remove the instanceof tests and the casting. Instead directly run the printDetails method on the Product reference.

What happens? Is it the printDetails method in the Product class that gets executed, or is it "by magic" the more specific methods in the Book or Movie class that gets executed?

By using inheritance and overriding methods in this way, we get both the advantages at the same time of handling objects in a more general way, and automatically executing the more specific implementation in the subclass!

Task 3

In the Movie class, we sometimes have a director and sometimes not. If there is a director, the printDetails method will print the director, otherwise it will not. We have so far solved this with an if statement, checking if director is null och not and if it is not null we print the director.

Could we handle this in another way using inheritance and polymorphism so that we could get rid of the if statement?

Some hints:

Create a new class MovieWithDirector that extends the Movie class. Move the director variable and the setters and getters to director from the Movie class to the new MovieWithDirector class.

Only have the constructor taking productId, title, genre and price in the Movie class. Add a constructor to the MovieWithDirector that takes the same arguments and also a String director. Call the superclass constructor with all the arguments except the director and assign the director argument directly to the instance variable.

In the Movie class, only print out the variables that are left, deleting the if statement with the director.

In the MovieWithDirector class, print out all variables including director. You could reuse the functionality in the method in the superclass by first calling the printDetails method in the superclass and then only print out the director.

If there is a problem accessing the title and genre from MovieWithDirector, you will need to change the access modifier in Movie from private to protected to allow the subclass to access these variables. But you should be able to use the methods in the superclass when accessing the variables in the superclass, so you could also keep the variables private if you don't need to access them in the subclass.

In the main method, use the new class MovieWithDirector for the movie with a director.

Run the program. Is everything working? Does the call to product.printDetails automatically print out the correct variables, the variables that exist in Movie when a movie is handled, and all the variables

including the director when a MovieWithDirector is handled?

Did you print out the recommendedAgeInfo from ChildrensBook? If not, add this functionality to the ChildrensBook class. Override the printDetails method, use the method in the superclass, and print out the recommendedAgeInfo.

Now all the different products should automatically print out all of their variables automatically without using if statements!