Chapter 7 Important Notes :

- we will now go to extra level of abstraction and that by using the inheritance and the polymorphism
- now when the classes inherit from another class now we will have parent class which is the superclass and we will have children classes which they are called subclasses now the subclasses will inherit the superclass functionalities automatically but what about there is one of the subclasses has different behavior or extra implementation details? Here comes one of the polymorphism concepts which is called override and here we will redefine the same parent functionality but with different behavior be suitable to this subclass one and in the runtime the JVM will decide which one this object will implement the parent method or the child method
- so when the subclass inherits from the superclass the subclass can add new methods and variables of its own and also can override the methods it inherits from the superclass
- the override can happened into just the superclass methods we can not make it to the super class variables and that cause of the variables do not define any special behavior so we will not need to redefine their behaviors using the override concept
- now we will design a program from the start and from now we know that we need to make our program flexible cause we want other programmers to be able to add new kinds of animals to the program at any time so that means we need to make the initial design very good and flexible
- okay now in the beginning of design a program the first thing to think about it is to think about the objects that have common attributes and behaviors then the second step is to think about designing a class that represents the common state and behavior and then the third step is to decide if a subclass needs behavior that are specific to its own class think about if there is subclasses will need to redefine some methods behaviors now the fourth step is to look for more opportunities to use abstraction and that by finding 2 or more subclasses that might need common behavior and this step is very important cause it avoids you from having multiple duplicated code in your classes
- here is another thinking point we will speak about it when I define an object which method the JVM choose to make this object implement it so now the JVM start from the object specific type which is the lowest type in the inheritance hierarchy and if there is a method having the same name so it runs it and if there is no match the JVM go step up and gets its superclass and make the same check and so on so remember that the lowest one wins okay but what about if the JVM continue going

up in the tree and that cause it does not find any matching method ? here come the role of the compiler the compiler first check if this method existed or no if no it doesnot run anyway so when the JVM searching about the matching method it will sure find it but its goal here is to find the right one the most specific version for that particular object .

- there is another relation ship between the objects and it is called HAS A one and the inheritance relationship called a IS A relation ship and to know the difference between them ask yourself firs the question is A is HAS A b or A IS A b object? If it is the first option so that means in your design the class A will have an attribute from the class B but if it is the second option that means the class B extensa from the class A so first always ask your self this question

- keep in mind that the inheritance IS A relationship works in only one direction like for example if we say that triangle is a shape it is correct but we can not say the opposite one cause the shape is not a triangle it is not must thing

- we see before that if subclass has different parent behavior it is can make override to replace parent logic with its own logic but what about I want to use the both logics which I mean use the superclass version and also use the subclass version too that's can happen and that's by in the subclass method the first thing you must to do is to calling the same method but of the parent one and then after you calling it you can write your own special behavior and it will be also implemented after finishing implemented the parent logic first like this one :

  ```
  public void roam() {
   super. roam() ;
   // my own roam stuff
  }
  ```
  Super keyword here is using to calling the parent method
- now we will start playing with the access modifiers which is controls who sees what and we have 4 ones but now we will focus on 2 the public one which means public members are inherited automatically and the second one is the private level control which means private members are not inherited is is just own for the class itself only okay lets take a look too for the protected and the default one too :
  - the default one : we do not write any keyword for it but when you do not mention any access modifier it is putted as default one and that means that this variable will be available for any class but in this current package only

- the protected one : we use it be using the protected keyword and it give the benefit of that it will like default but with extra level that It is will be available for any inherited classes but in another packages which the default can not do that

- now we will think about another think about how you can use the inheritance good using to make your design well designed this is the steps you must be careful about it :
  - do inheritance when one class is more specific type of a superclass
  - consider the inheritance when you have behavior that should be shared among multiple classes of the same general type like circle and square, triangle all having the same shared things like the rotate and play sound so the inheritance here will be useful in the reusability and making the maintenance more easier
  - do not use inheritance just for reuse code from another class if the relationship between the superclass and the subclass violates the 2 previous codes you can not use the inheritance here and just ask yourself the question of is there is IS A relationship or no?
  - the question of the IS A relationship if its answer is false so do not use the inheritance relationship here
- so the inheritance benefits one of them is that you avoid the duplication code and that cause you put the common code in just one place and just once and if there Is a change happened you can just change one place in the super class and that without needing to changing the another subclasses you will need to change just one place which make the life easy
- in the normal situations and what we learned in previous chapters is that when you define Dog x = new Dog() both here is the same but now we will go with the polymorphism and will see that with the polymorphism the reference type and the object type can be different like that Animal x=new Dog();
- with the polymorphism the reference type can be a superclass of the actual object type but what the benefit of this thing? It is really very very important thing and lets think about it with the polymorphism here you can define an array that carry different classes types but still having the same common class lets get an example :
  Animal[] animals = new Animal[5];
  animals[0] = new Dog
  animals[1]=new Cat();
  animals[2] = new Wolf
  animals[3] = new Hippo() ;
  animals[4] = new Lion()

so here we can put any subclass of animal in the animal array here comes the benefit of the inheritance and the polymorphism together and after declaring the array you can traverse over this array and in every iteration you will can call its own special method and every object will do its right thing but the life is not happy in all cases what about if there is different arguments or different logic in the return types ?

```
for (Animal a : animals) {
  a.makeSound();
}
```

The polymorphism here will be good in just one case if the make sound method is existed in the superclass and the same time all the subclasses having it and override it or no it will be good and this method having the same signature in all the subclasses (return type and the parameters type and order and number)

- so with polymorphism you can write code that doesnot have to change when you introduce new subclass into your program
- one of the benefits of using the OO is that if we have a class and we can not access its source code to rewrite specific method logic the inheritance here can come and solve your problem and that be extend the class and override it
- there is some things can prevent a class from being subclassed which means prevent class from extend any other classes :
    - first one is If the class Is not public that means It will be valid for only its package classes it can not extend any other classes in another packages
    - the FINAL keyword a final class means that is the end of the inheritance branch nobody ever can extend the final class
    - the third thing is if the class having its constructors private ones so the class will can not be subclassed
- if you want to prevent specific method from being overridden you can define it as a final method but if you want to prevent all the class methods from being overridden you can define the class itself as a final class and that is for security reasons like the string class make it is declared as a final class and that to prevent any one to change the way the strings behave
- when you make override for method the arguments must be the same as it is existed in the superclass and the same for the return type too it must be compatible and that means return the same thing like the superclass or return one of the subclasses of the superclass return type for example if in the superclass method it return animal object in the subclass method it can return the same return type which is animal object or can return any of animal subclasses like dog object both will be okay and the second thing is that the method can not be less accessible that means

the access level must be the same or more bigger so if the method in the parent defines as public you can not change it in the child class as a private one it will give you error

- here the last thing we will talk about it is the overloading a method and it is means an overloaded method is just a method that happens to have the same method name but it differ in the parameters type , order , number and the it has nothing to do with the inheritance and polymorphism cause the overloaded method is not the same as the override methos , the return type can be different in the overloaded method you free to change the return type as long as the arguments lists are different but you can not change only the return type and you can vary the access levels in any direction