Comparison of three different designs of class inheritance and interface

This document compares three versions of class interface design, namely, version1, version2, and version3. Their source code files are included in “***exampleOfClassInterfaceDesign.zip***”. The purpose is to illustrate how we can implement dynamic-binding, under the context of using parent reference refers to child objects in an array containing child objects of different child classes.

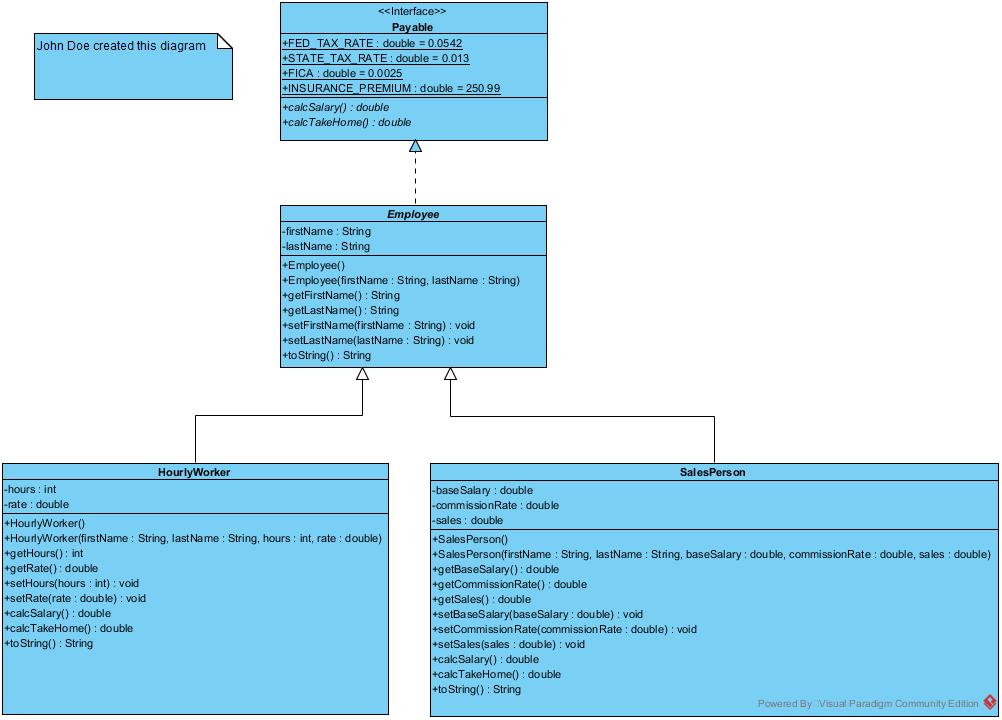
We need to know that, in order for dynamic binding to happen, **the parent class must have a prototype** **of the method that is handled with dynamic binding**. Thus the designs in version1, 2, and 3 differs in a way how the method prototype is implemented in the superclass (parent class). Having this in mind, we can see from the table below how the three versions of designs differ from each other.

|  |  |  |
| --- | --- | --- |
|  | How does superclass have the prototype of dynamic binding methods ? | Is dynamic binding enable? |
| version1 | superclass (Employee) inherits from interface (Payable), thus superclass has the prototypes of the abstract methods defined in interface, and these abstract methods defined in the interface are to be used as dynamic binding methods. The superclass itself becomes an abstract class because it inherits from an interface without implementing all abstract methods in that interface. | Yes |
| version2 | superclass (Employee) does NOT inherits from interface (Payable), instead, superclass has dummy implementation of the dynamic binding methods. The superclass has no need to become abstract. | Yes |
| version3 | superclass (Employee) does NOT inherits from interface (Payable), nor does it have dummy implementation of the dynamic binding methods, thus dynamic binding is not possible. | No.  Therefore, the **instanceof** operator is used to determine which child object is referred to by the parent reference variable |

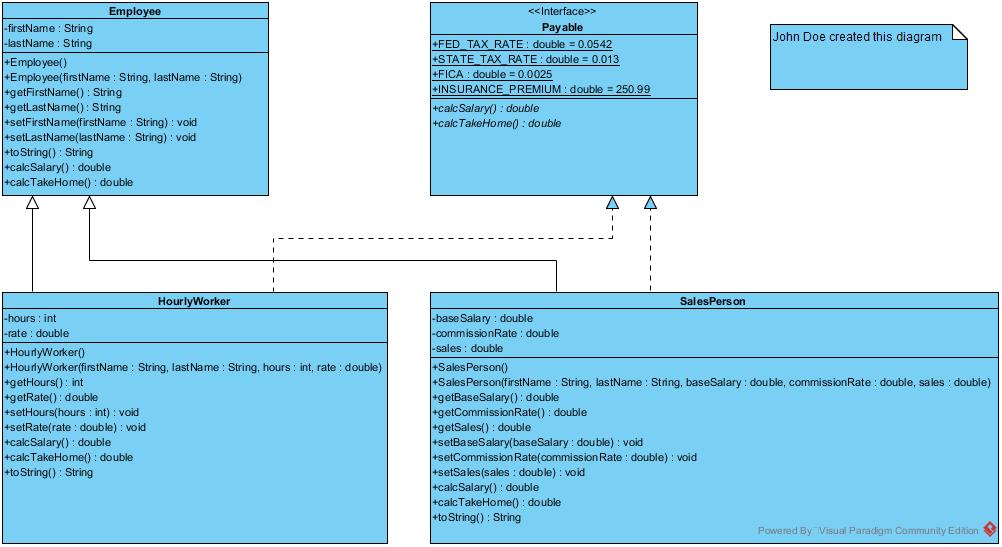
The rest of the document provides the UML class diagrams for the three different design versions of class and interface design. In the following UML class diagrams, you need to pay attention to which classes/interface have/has these two methods: calcSalary() and calcTakeHome(), and whether these two methods are abstract, or have dummy implementation, or concrete implementation. A jpg file containing the UML class diagram is also provided for each version.

Version 1’s UML class diagram is as below. We can see that superclass Employee is an abstract class (its name is *italic*), and it inherits from the interface Payable thus obtains the prototypes of the two abstract methods, so that the dynamic binding for these two methods: *calcSalary()* and *calcTakeHome()* are enabled in the application file.

There are three generations of class/interface in version 1’s design. The grandchild class HourlyWorker and SalesPerson inherit from their grandparent, which is the Payable interface, two abstract methods named ***calcSalary()*** and ***calcTakeHome()***, and the two grandchild classes have the obligation to implement these two methods, in order for these two classes to become a normal non-abstract class. They do fulfill the obligation passed from their grandparent, and they implement these two methods in their class, respectively, and we can see from the UML class diagram that, these two methods are not italic anymore in class HourlyWorker and class SalesPerson, and the class name HourlyWorker and SalesPerson are not italic either. In comparison, the parent class of HourlyWorker and SalesPerson, which is the *Employee* class, has to be declared as abstract (*italic)* in the UML class diagram, because *Employee* class does NOT fulfill the obligation of implementing the abstract method defined in its parent, the Payable interface in this case.



Version 2’s UML class diagram is as below. We can see that superclass Employee is not an abstract class anymore (its class name **Employee** is no longer ***italic***), and this superclass has no inheritance relationship with the interface Payable. To obtain the prototypes of the two dynamic binding methods, the superclass Employee have these two methods implemented as dummy methods: **calcSalary**() and **calcTakeHome**(), and they are not abstract methods (not ***italic***), either.



Version 3’s UML class diagram is as below. We can see that the difference between version 3 and version 2 is that, the Employee class in version 3 does not have any dummy implementation of the two methods that are overridden by its subclasses: calcSalary() and calcTakeHome(). In other words, there is no prototype of these two methods in superclass Employee, therefore the dynamic binding of these two methods are not possible. That is why in version 3, the application source code “*PayrollArrayDemo.java*”, line 19 and 25, program has to use operator ***instanceof*** to determine which child object the parent reference variable is referring to.

