

Figure Payable interface class

For this part, this was the first file I coded and this was meant to be the interface. This would later be adopted by the other classes and I coded it the way it was instructed, meaning I declared the method but I didn’t implement it.



Figure Invoice class: methods and instance variable declarations

I declared private instance variables so that it would not be easily accessible. Therefore, the codes you see here are getters and setters, and the purpose of those methods are to set the variables and get those variables without directly accessing them. There is also a constructor, which can be used to initialize those variables when the object is instantiated.

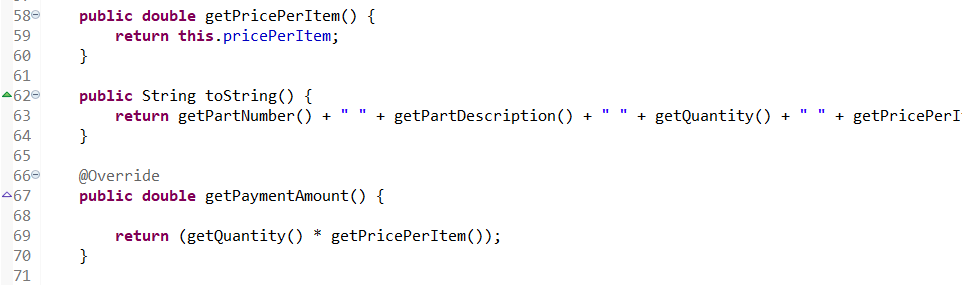


Figure Invoice class continuation

The last two methods are the toString and getPaymentAmount. The purpose of the two string is to utilize all the getter methods and return a string, that details the information of the object. The getPaymentAmount is another way to show polymorphism and is done by using @override keyword. Calling this method will always change depending on which object is calling it.

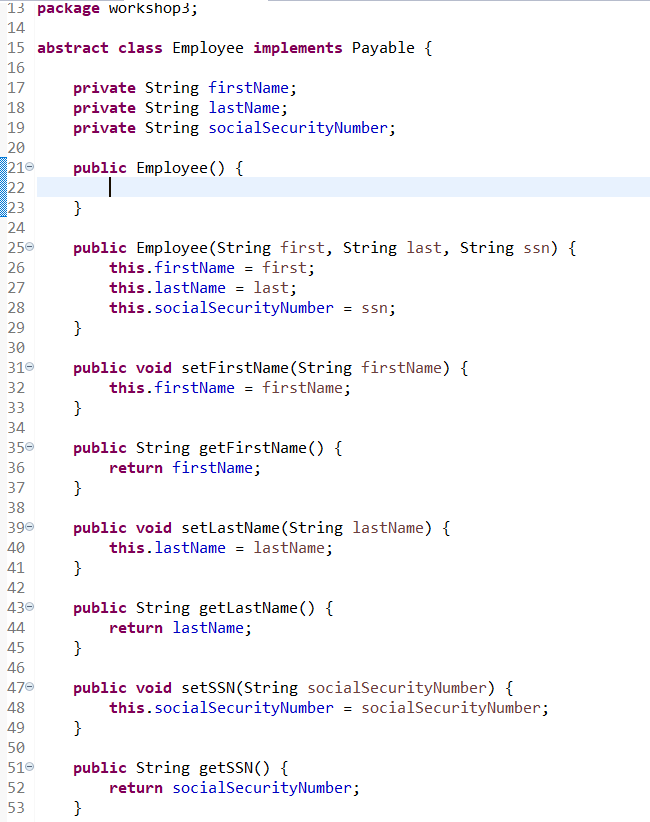


Figure Instance variables and methods for Employee class

This is an abstract class. The codes here includes all the basic information that different type of employees will need, such as first name, last name and SSN. The codes seen here includes the constructor and getter and setter methods. Reasons for them are the same as explained in the previous page.

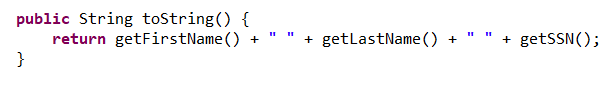


Figure Employee class continuation

This polymorphic method can be seen in most classes and will vary in information depending on the object called.

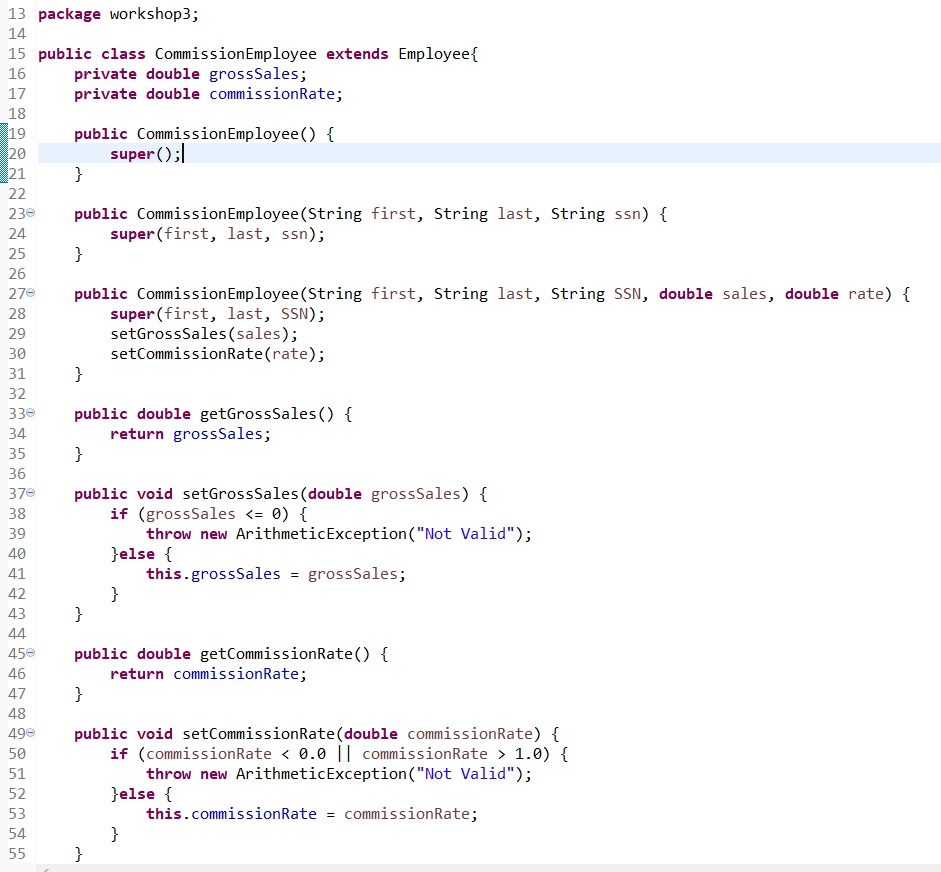


Figure CommisionEmployee's instance variables and methods

Since CommissionEmployee class has Employee as its Superclass, the keyword extends needs to be present to tell the program that this is a subclass. This class will have its own instance variables and the concepts of the methods and constructors are the same as the previous classes. One unique thing about this class is that it includes exception handling. Based on the data received in the parameters, it will evaluate if it is valid. For example, for the setGrossSales method, it takes the received value and determines if it fits the rules. Gross Sales cannot be less than 0 because that’s not possible. If it does receive a negative value, instead of the program terminating, it will throw a new exception but the program will continue to execute. SetCommissionRate method has the same concept.

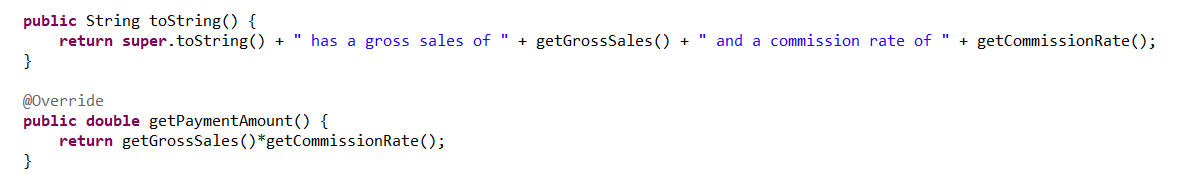


Figure CommissionEmployee continuation

We can finally see getPaymentAmount finally being implemented here. This method will get the amount this employee is earned by multiplying the returned values from the two getter methods.

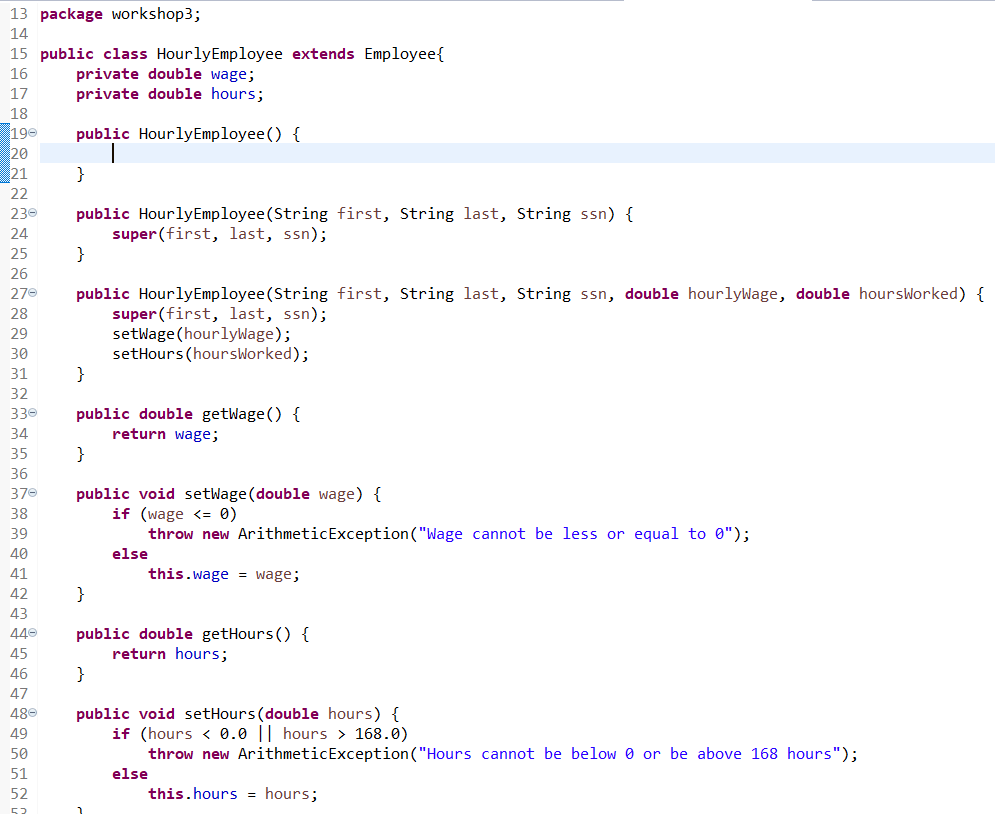


Figure HourlyEmployee class and its instance variables and methods

Concept of this class is the same as the previous class. It has its own instance variables and methods.

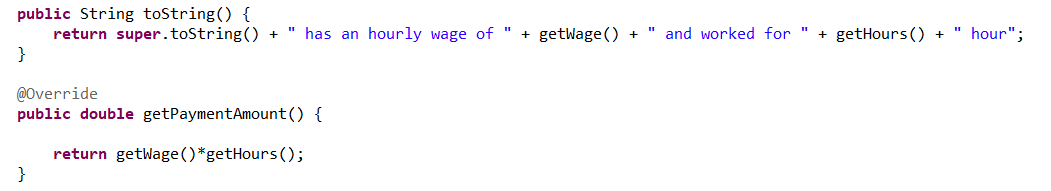


Figure HourlyEmployee continuation

Shows the two polymorphic methods. These methods will display data unique for HourlyEmployee objects.

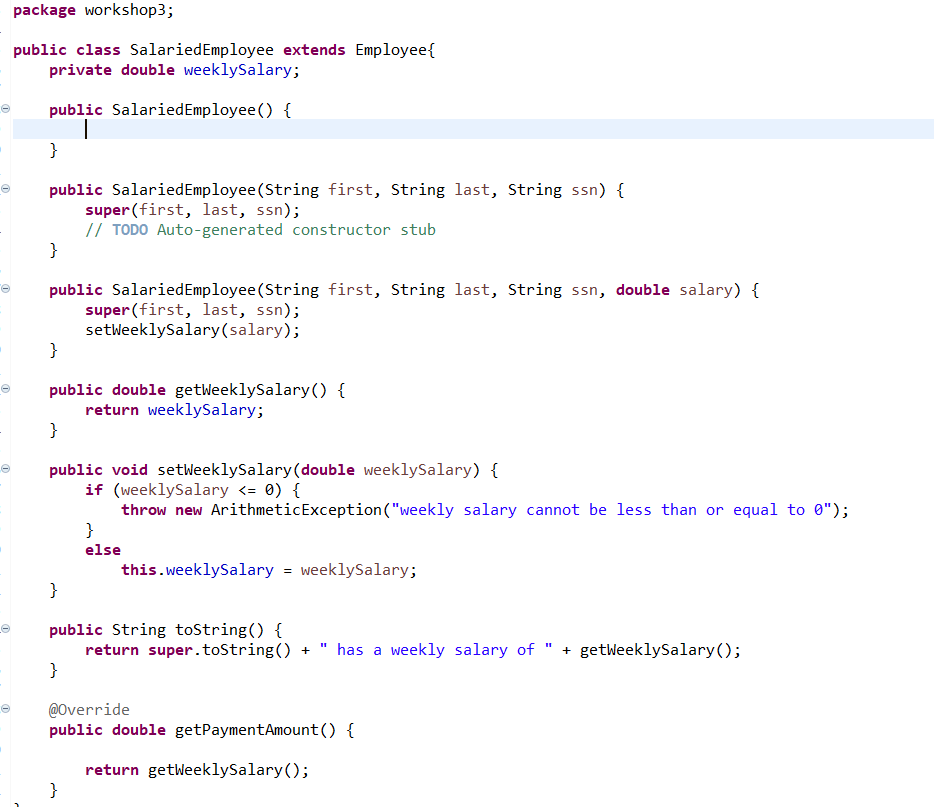


Figure SalariedEmployee class and its instance variables and methods

The idea of this class is the same as the previous 2 classes. Again, it has their own instance variables and methods unique to this. The last two methods will display data that is unique for this specific class object.

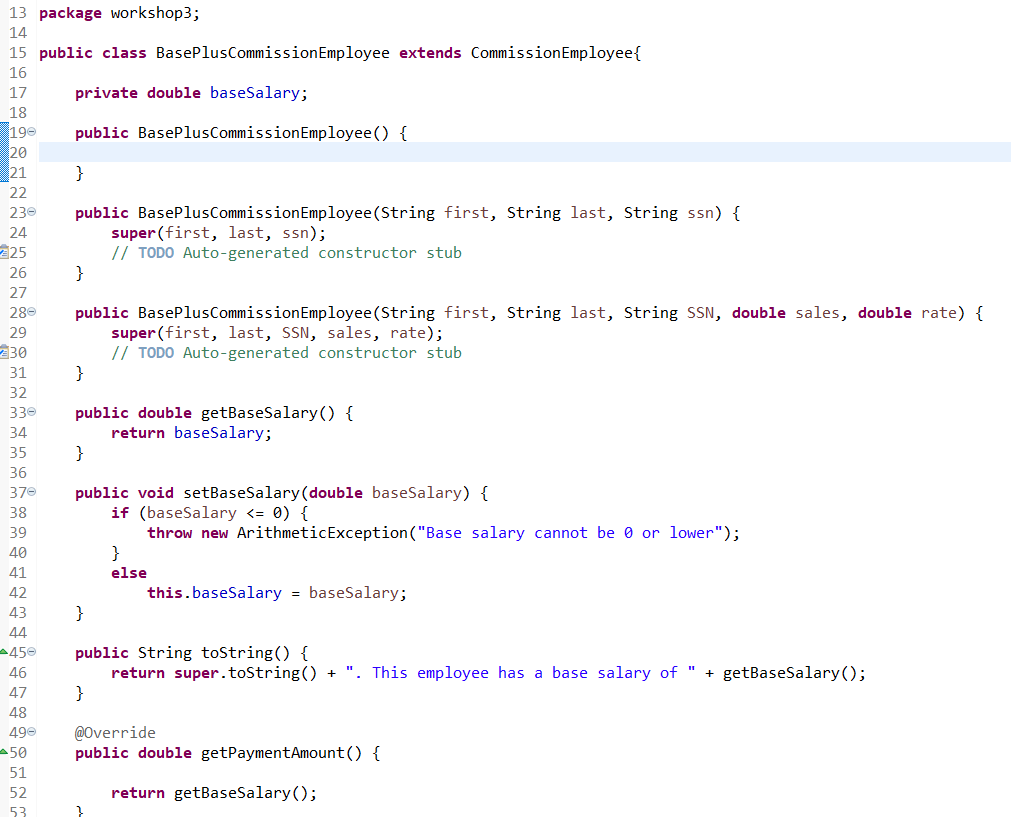


Figure BasePlusCommissionEmployee and its method and instance variables

Instance variables and methods that are unique for this class. Concepts is explained in previous pages.

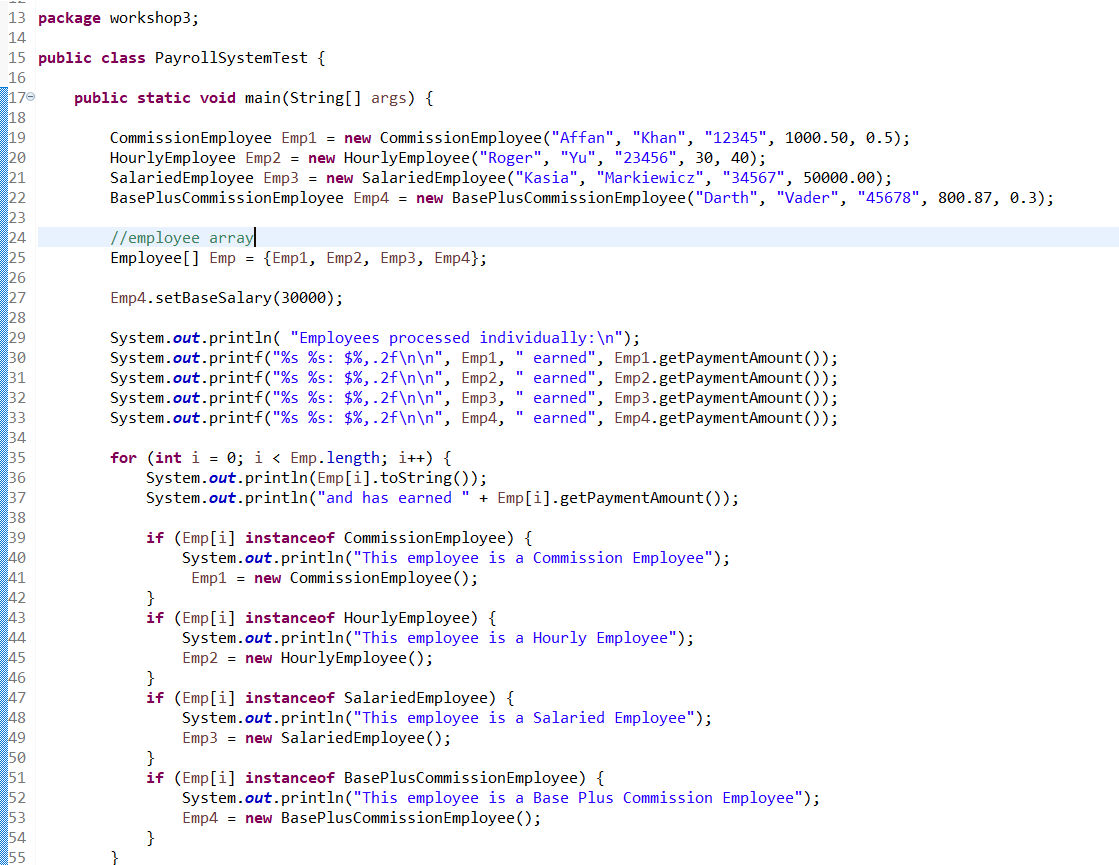


Figure Tester codes

This class makes sure the program runs as expected by and creating objects for each class. Here, we can see 4 objects are created, one for each class. I then put them into an employee array and set the base salary for BasePlusCommissionEmployee object. I then printed out the data for each object based on the sample format provided in the workshop instructions. Afterwards, I made a loop to print out the toString and getPaymentAmount to see if the polymorphic methods work. This is done thanks to the keyword instanceof. If worked as planned, each object within the array will have different data to display.

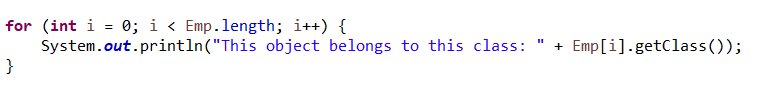


Figure getting class data

This last for loop will display which class each object belongs to.

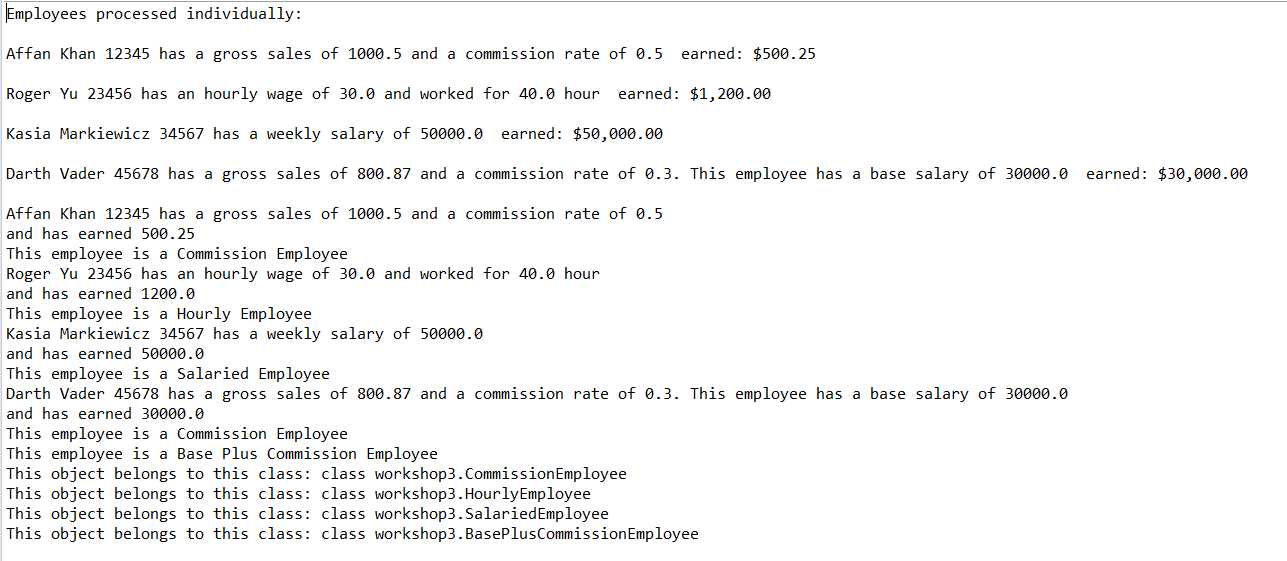


Figure results

These are the results of the code. I would say it ran successfully as each object clearly displayed their own unique and specific data. This goes to show that polymorphism worked as intended.