Object Oriented Programming Principles

This report seeks to explain the following OOP principles; encapsulation, inheritance, polymorphism and abstraction, using a payroll application of a fictitious marketing agency, Vortex Telemarketing.

Vortex's payroll application uses employees' basic information such as name, date of birth, age, telephone numbers, house address, etc. which are represented as attributes in the application. These are personal information which should not be manipulated by another programme.

Encapsulation: is a mechanism of wrapping (hiding) attributes (variables) and operations (methods) into objects. This is usually done to prevent objects of a class from manipulating the attributes of another class. Using the below code as example:

```
public class Employee{

        public String name = "James Smith";
        public Date doB = new Date (12 Mar 1978);
}
```

Attributes of the above class can be accessed and modified by objects of another class, because class *Employee* attributes have a *public* access modifier. To prevent objects of other classes from manipulating the object attributes of class *Employee*, the attributes access modifier should be declared as *private* and a *public* method should be provided for other class to use to access *Employee* attribute data. This mechanism will only allow objects of other class to access *Employee* objects attributes through the *public* method without modifying the data, this can ensure data consistency. Only objects of class *Employee* can manipulate the data (variables).

```
public class Employee{

        private String name = "James Smith";
        private Date dob = new Date (12 Mar 1978);
        // public methods to set and get the above attributes
        public void setName(String name){
                this.name = name;
        }
        public String getName(){
                return name;
        }
}// end class Employee
```

Vortex Telemarketing has two types of employees; commission employees and salaried employees. Commission employees are paid a percentage of their sales while salaried employees receive a base salary. Both types of employees have similar attributes and operations, but differ in the way their wages are calculated.

Inheritance: is a form of software reuse in which a new class is created by absorbing an existing class's members and adding new or modified capabilities. The new class is called the subclass, while the class from which the subclass is derived is called the superclass. In our example, class *Employee* is the super while *CommissionEmployee* and *SalariedEmployee* are subclasses. Subclass has *is-a* relationship with the super class i.e. *CommissionEmployee is-an Employee.* A subclass absorbs all superclass members using the *extends* keyword:

```
public class CommissionEmployee extends Employee{

        private double commissionRate = 0.09
        private double totalSales = 5000;


        public double earnings(){
                return commissionRate * totalSales;
        }
}// end class CommissionEmployee


public class SalariedEmployee extends Employee{

        private double weeklySalary = 100;

        public void setWeeklySalary(double salary){
                weeklySalary = salary < 0.0 ? 0.0 : salary;
        }
        public double getWeeklySalary(){
                return weeklySalary;
        }
        public double earnings(){
                return  getWeeklySalary();
        }
}// end class SalariedEmployee
```

The payroll application can process the objects of *CommissionEmployee* and *SalariedEmployee* as if they are all objects of class *Employee.*

Polymorphism: Is the ability of an object to take on many forms. This enables one to "programme in the general" rather than "programme in the specific". This enables the processing of objects that share the same superclass (either directly or indirectly) as if they are all objects of the super class. This is commonly used when a parent class reference is used to refer to a child class object.

```
public class PayrollSystemTest{
        public static void main (String[] args){

        Employee commissionEmployee = new CommissionEmployee();
        Employee salariedEmployee = new SalariedEmployee();
        }
}
```

This superclass reference variable can be used to invoke only the methods declared in the superclass. To invoke a subclass-only methods on a subclass objects referenced by a superclass variable, the programme must first downcast the superclass reference variable to a subclass reference. Using the *salariedEmployee* reference variable as example:

SalariedEmployee salaried = (SalariedEmployee) salariedEmployee; *//downcast superclass salariedEmployee reference variable type to subclass salariedEmployee reference type.*

The *Employee* class may not necessarily be used to create objects, since it may just be used only as super classes in inheritance hierarchy. Such superclass can be declared as abstract and are referred to as abstract superclass.

Abstraction: is the concept of exposing only the required essential characteristics and behaviours with respect to a context. In Java, the purpose of an abstract class is to provide an appropriate superclass from which other classes can inherit and thus share a common design. Using our code example, class *Employee* can be declared abstract with an abstract method *earnings* which has no implementation. The *earnings* method must be implemented by any subclasses that inherit the *Employee* class for it to be a concrete class.

```
//abstract super class Employee

public abstract class Employee{

        private String name = "James Smith";
        private Date dob = new Date (12 Mar 1978);
        // public methods to set and get the above attributes
        public void setName(String name){
                this.name = name;
        }
        public String getName(){
                return name;
        }
        public abstract double earnings(); //this method must be implemented
by the subclasses
}// end class Employee


public class SalariedEmployee extends Employee{

        private double weeklySalary = 100;

        public void setWeeklySalary(double salary){
                weeklySalary = salary < 0.0 ? 0.0 : salary;
        }
        public double getWeeklySalary(){
                return weeklySalary;
        }
        @Override
        public double earnings(){
                return  getWeeklySalary();
        }
}// end class SalariedEmployee
```