Problems relating to Unit 3: abstract and concrete classes, and polymorphism

Polymorphism:

1. Explain what you understand by the OO term **polymorphism**. Is The ability to manipulate objects to suit your requirements like an Animal a dog and a cat have similar attributes
2. Consider the following inheritance hierarchy diagram:

Vehicle

Car

Tank

Assuming you have already written the Vehicle, Car and Tank classes (if you haven’t, write stub classes for them with headers that correspond to the hierarchy), write an application which declares and creates an ArrayList called fleet of Vehicles and adds one car and one tank to it. Explain how polymorphism can be used to display the details of each vehicle in the fleet. If we need to write a Train subclass of Vehicle, and add a Train to the fleet, will we need to change the generic part of the application that displays the subclass instances in a polymorphic manner? Explain your answer. All the attributes of vehicle still apply and another attribute number of passengers can be added in the train class

Public class Vehicle {

Public class Car extends Vehicle {

Public class Tank extends Vehicle {

1. Explain very briefly what is meant by the term **dynamic method binding** – illustrate your answer with references to the code from the previous exercise.

This is where Vehicle v = new Car();

So when a method is called it still refers to the car object and not the Vehicle e.g. new Car()

Abstract and Concrete classes

1. What is the fundamental difference between an **abstract** and a **concrete** class? Give *any* suitable example you like of each type. You can’t create an instance of an abstract class e.g. Animal a = new Animal(); == Concrete class instance and a.getWeight() how to use the extended abstract class
2. Where do **abstract** classes normally appear in an inheritance hierarchy? Where do **concrete** classes normally appear in an inheritance hierarchy? On top of the hierarchy chart concrete class is in the same place

The sample classes Employee, Manager, HourlyWorker and WorkerTest:

1. Copy the files Employee.java, Manager.java, HourlyWorker.java and WorkerTest.java to your H: drive. First examine the **Employee.java** file and answer the following questions based on it.
   1. The class is defined to be abstract, which, among other things, means that the class should not be instantiable. In the driver program WorkerTest.java, add code that will attempt to instantiate an Employee object. What is the outcome? Why does it make sense that the Employee class should be abstract?

Employee class is abstract; cannot be instantiated, abstract classes are incomplete and can be manipulated to suit each employee type e.g. fill in the missing parts of the methods to suit the requirements.

* 1. Try to make the Employee class a regular concrete class, by removing the word abstract from its class definition header. Recompile. What happens? What does this tell you about abstract methods?

Abstract methods can only be contained within abstract classes

* 1. Notice that the Employee class does not contain any no-argument constructor and yet the application compiles fine. How is this possible?

There is no requirement for a no argument constructor in any class this is used to setup an initial state of and object.

* 1. The Employee class contains just one abstract method called earnings(). Would it be possible to create an abstract class without any abstract methods in it? Try it out and recompile. What is the outcome? Its compiles successfully there is no requirement for an abstract method in an abstract class if a class has an abstract method the extended class has to implement any abstract methods.
  2. Notice that the abstract method is written without any opening and closing curly braces after the (). Would you expect the program to compile if braces {}were put in before the semi-colon? Try it out and recompile. What is the outcome? What if you omit the semicolon? Must the class still be declared abstract? Abstract methods cannot have a body if you omit the semicolon the method would not be declared abstract

1. Examine the Manager.java file and answer the following questions based on it:
   1. How can you tell that Manager is a concrete class? Its not abstract
   2. The Manager class is defined to be final. What does this mean? You would declare a class final if you didn’t want a derived class overriding the methods of the superclass
   3. Does Manager actually inherit the attributes defined in the Employee class? Prove this. No local variables of the same type are declared
   4. Explain very briefly what the purpose of the following line of code is in the Manager constructor: super(fn,ln);

This is a creation of the constructor from the superclass this is possible trough inheritance

* 1. Looking at the code for setWeeklySalary(), what would you expect to happen if a value such as –1000 was input for the Manager’s weekly salary?

The managers salary would be 0

* 1. The Manager class contains an earnings() method. Does this method absolutely have to be in the class or could it be omitted or called something else e.g. managerEarnings()? Try out these possibilities and report your results. What does this tell you about the inheritance of abstract methods? They need to be called exactly the with the same modifiers and parameters
  2. The toString() method in the Manager class makes a call to the Employee toString() method in order to do its job, via the ‘super’ reference. Is it essential that the method toString() be called on the super reference or could it be omitted? It can be overridden because Employee is extended it has access to methods of Employee e.g. getFirstName()
  3. Give any example within the Manager class where overriding occurs.

Public double earnings(){return earnings;}

1. Examine the driver program WorkerTest.java and answer the following questions based on it.

(a) The driver program here contains polymorphic behaviour. Explain where the **polymorphism** occurs. The object of employee is used emp == m1 and emp == hw1 two different types of employees with the same attributes

(b) Indicate exactly where in the driver program **dynamic method binding** occurs.

(c) Notice in the driver program the lines of code such as

eRef=b1; and

eRef=hw1;

emp.toString (), emp.earnings())); would still refer to b1 or the hw1 object

where the superclass object reference is set to refer to appropriate subclass objects. Do you think it should be possible to do the opposite i.e. have the subclass objects set to refer to the superclass reference? Try it out to see the result. Does this make sense?

No

it won’t work the other way around