Problems Relating to Unit 2: Basics of Inheritance

1. Terminology of inheritance: Consider the following inheritance hierarchy diagram:

Animal

Cat

Bear

1. From this diagram identify,
   * 1. The base class(es) Animal is the base class
     2. The derived class(es) Cat and Bear

(b) In terms of inheritance, give another name for

i the base class Superclass

ii the derived class Subclass

(c) With respect to inheritance, say how the superclass differs from its subclass. What type of relationship is implied by inheritance? Parent child relationship the super class sets up the characteristics of the object and the subclass manipulates the object to shape it into the required state

(d) Imagine that you intend to create a class called **PetCat** that must inherit from **Cat**. Give the line of code for the **PetCat** class definition header. public class PetCat extends Cat

(e) What is wrong with the following java inheritance diagram, assuming all three entities are classes? The super class is always on top

Pet

Cat

PetCat

(f) Explain the difference between *single* and *multiple* *class* *inheritance*. Which type does Java support? Single inheritance is inheriting a single class and multiple inheritance in inheriting multiple classes java supports single inheritance

1. Final methods and classes:

(a) When should you make a method **final**? In the Person class, make the toString() method final and recompile. Now attempt to compile the Student class. Is this as you expect? Restore the Person class to its previous state. If final is used an entity cannot be changed later

(b) What about final classes? In the file Person.java, make the Person class final. Now attempt to compile the Student class. Is this as you expect? when a class has final in the declaration a subclass cannot be created from that class

1. Writing stub classes for a class hierarchy: The lecture notes include a hierarchy chart for animal-related classes. Write stub classes for each class in the hierarchy, making sure to get the relationships right. Use abstract classes where appropriate.
2. A subclass of Player: The Player class has attributes name and score. Write a subclass PacmanPlayer which also has an attribute ‘lives’ representing the number of lives left for the player. This should initially be 3 for all players. Include methods loseALife() and isDead(). Write a tester for the PacmanPlayer class. Advanced version: your tester should be frame-based, with buttons for ‘lose life’ and ‘query status’. Once the player loses its last life, the frame-based tester should display a ‘totally dead’ dialog then shut down.
3. Writing a hierarchy of classes: Consider the following **inheritance hierarchy diagram**

Vehicle

Car

Tank

Write all three classes, assuming the following:

**Vehicle** is a class with the attributes*price*, *length*, w*eight*, and *model,* mutator and accessor methods for each of these attributes, a multi-argument constructor, a no-argument constructorwhich initialises the Vehicle attributes with default values and a toString() method for displaying the state of a generic Vehicle object (use the accessors here to access the attributes indirectly, and include a call to getClass()).

**Car** is a class with extra attributes *maxPassengers* (int) and *registrationNumber* (a String e.g. “09 KY 1234”), a no-argument constructor that initializes a Car object with a set of default values, a 6-argument constructorthat sets up indirectly (via mutators) the values of its own attributes as well as making an appropriate call to the Vehicle constructor to initialize the attributes defined there. Also needed are accessor methods, mutators and a toString() method that can be used to display the state of all 6 attributes associated with a Car, as well as the class name. The toString() method should override, yet reuse, the toString() method from the Vehicle class.

**Tank** is a class with an extra attribute *armoured* (a boolean), a 5-argument constructor that sets the value of its own attribute as well as calling the Vehicle constructor to initialize the attributes defined there, a no-argument constructor (tanks should be not armoured by default), an accessor and mutator method for its own attribute as well as a toString() method that can be used to display the values of all the 5 features associated with a Tank. Again, the toString() method should override, yet reuse, the toString() method from Vehicle.

**Driver Program**: Once you have written Vehicle, Car and Tank, write a driver program that fully tests the functionality of all the classes concerned. It should create an array of 3 vehicles called fleet, ask the user whether the next vehicle is to be a tank or a car, get the details from the user, create the object and add it to the array. When all have been created, their details should be output. Use getClass() or *instanceof* to determine whether each object from the array is a car or a tank. Write pseudocode for the driver before you begin to code it.