Uttara Inheritance Practicals

I want you to go through the below problem and then follow the steps I have mentioned in detail below. In the unzipped contents, the Animal and Vehicle discussed example have been coded. You can compile them all in one shot and go through the code of TestAnimal.java and TestVehicle.java.

a) A device has a name and can do something. TV, Printer and Microwave are devices. Using TV, you can watch movie; using Printer, you can print; using Microwave, you can cook (all these are doSomething() implementations - so override them in the respective classes).When a device is made to dosomething, it shouts out its name as well. An Electrician is one who can test any device. When he is asked to test a device, he will make it do something and test it. A TV can switchChannel as well (delta functionality). So if a tv is given to be tested to an electrician, he will make the tv dosomething and switch channel as well. Write a tester class to test how devices, electricians work.

If Device has a private String name, do Printer, TV, etc subclasses have a name? If yes, how to access it in overridden doSomething() method? Try it out.

Steps to implement program:

1) Frist build the Device.java class in source folder with the Device class definition.

```
Device
        String name;
        public void doSomething()
        {
                SOP(name+" doing something");
        }
```

Compile the .java file from source to classes.

2) Build the TestDevice tester class with main(), Test device working:

```
Device d = new Device();
d.name = "Davy";
d.doSomething();
```

Compile, execute TestDevice and verify working.

3) Add 2 constructors to Device - one a no-arg constr and another which takes a String as paratmer (in this set the passed param to name state).
Put SOPs in the constructors. Recompile Device.java

4) Re-execute the TestDevice program without recompilation. Do you see the SOP from the no-arg constructor? How come even without re-compiling TestDevice, the latest Device.class was used? (class loa...happens at ru..)

5) Build the TV class and make it extend Device. Put a no-arg constructor in TV and put an SOP. In tester class, plug in Device d = new TV(); and compile, execute. Do you 2 constructors getting executed? (constructor chaining). Verify in the tester class, if d.doSomething() is called, parent implementation is getting picked up.

6) Override doSomething in TV.java and plug in a SOP. Now execute the tester. This time the overridden implementation of TV`s doSomething should be executed (as method call to method body linking happens at runtime depending on the object on which the method is called).

7) Add another method called switchChannel() in the TV with a SOP. Verify in TestDevice, whether you can invoke d.switchChannel()? Why not?

```
TV t = (TV) d;
t.switchChannel();
```

Is this working?

8) Now create Printer.java and Microwave.java and make them become IS-A Device. Compile them.

10) Create an Electrician.java (not IS-A Device!).Now create a testDevice(Device d) -> coding to the parent ref method. You should call d.doSomething() in its body. Compile.

11) In TestDevice->main()-> remove all earliler code, create an Electrician object, create a TV object, Printer object, Microwave object and ask the electrician to test each of the devices by passing their reference. Verify each time by recompiling and executing is he testing the correct device and is the polymorphic code of testDevice() is working correctly.

12) Now add instanceof check in testDevice() with respect to TV and then downcast and invoke t.switchChannel() in the method. Verify if the Electrician is switching channel only if you pass him a TV.

13) Now make your name state variable private and setter/getters in Device. Now change code accordingly in the subclasses and verify if all subclasses have a name.

14) Any doubts, call Manju / Suma / Ajay / Sneha / me.

b) Animals can eat,sleep and dance. Hippo,tiger,croc etc are animals. Every animal has name. An animal snores when it sleeps. A vet can treat animals. When an animal is treated, it is made to dance, eat and sleep. A Croc can swim, a hippo can smoke. When a Vet is given a croc, he will make it swim as well. When he is given a hippo to treat, he will make it smoke as well. Test the working of Vets polymorphic code as per the steps followed in the earlier example. Put all classes in relevant packages.

c) Person has a name and age. A person has a number of pet names(20max) which he obtains over a period of time. He can dance; if his age is less than 25 he can do chacha. If his age is greater than 25, he does the waltz. He can sing too and when he is asked to sing, he uses his petnames to form the song (randomly). Write a tester program to test persons.

Person

```
	private String name;
	private int age;
	private String[] petNames = new String[20];
	int count=0;

	public void sing()
	{
		String song = "";
		for(int i = 0; i < petNames.length; i++)
		{
			int n = (int)(20 * Math.random());
			song = song + petNames[n];

		}
		SOP(song);
	}
	public void addPetName(String n)
	{
		if(count < petNames.length)
			petNames[count++] = n;
		else
			SOP(..);
	}
	public boolean searchPetName(String n)
	{
		// search in the petNames array whether a name equal to n exists...and if yes,
return true, else return false;

		for(String s : petNames)
		{
			if(s.equals(n))
				return true;
		}
		return false;

	}
```

Important Polymorphism usage problem:
d) There are Bags. You can use the bag to store items (for which the user of the bag will give max number of items at construction time). You can then retrieve items from the bag. An item has a name and a price. Caps, notebooks, pens, lipstick are all items. A bag can be used to

search for a given item. You can request the bag to give you the total of prices of all the items in the bag. Write a tester class to test creation of bags, items, then add items into the bags and invoke the various methods of bag to test how to search, retrieve, get total, get individual price of items.

Item

```
String name;
double price;
public Item()
{

}
public Item(String n, double p)
{
        name = n;
        price = p;
}
```

Pen, NoteBook are subclasses of Item with a similar param constr where we explicitly invoke super.

Pen extends Item

Bag

```
String name;
Item[] items; // create Item.java and compile it first
// bag has-a item(s) -> 1..n multiplicity

int count = 0; // to keep track of how many items added

public Bag(String n, int size)
{
        name = n;
        items = new Item[size];
}
public void addItem(Item it)
{

}
public boolean searchItem(String n)
```

```
{
        //loop over the items array and compare each items name
        // with n and if equal, return true, else after comparing
        // the entire array, return false
}
public double getItemPrice(String n)
{

}
public double getTotal()
{
        // add all the items prices to a sum var and return it
}
public Item getItem(int pos)
{
        // return item ref from the position
}
```