Uttara InfoSolutions

www.uttarainfo.com

Uttara Lab - final, super, this, abstract usage

Things to test today:
1) test final variables, methods, class
2) how to design immutable class
3) test redeclaration of instance variables in child and accessing them and overriding of methods and accessing parent implementation
4) abstract usage
5) super & this

Note: Go through working of Insect.java & other .java files given in this lab first.

1) Create a class A. Create a final int p variable in it. Compile it without assigning any value. Now use an instance initialiser to assign a value and compile. Then in a tester class, create object and try to modify obj.p and see if you can do so. Then remove the instance init and then create a parameterised constructor where you accept an int and assign it to p. Now in tester class, create object by passing in an int to constructor and verify if object has the assigned int value.
Create a public static final int R = 5. See if you can print the value directly in main(). Can you modify the value?

2) Create a class Person. A Person has a name and age. Design an immutable class for Person. Create an object in tester class and

verify if you can change the state of the person.
(Remember: Immutable class design = final class with final instance variable with parameterized constructors)

3) Create a class A with instance variable String name = "A" and an SOP in param constructor.
Create a class B which extends A and also has String name = "B" and an SOP in its constructor.

In tester class, create object of B and verify what constructors are getting fired. Using ref of B, point to the object and print the name value. Also using a ref of A, point to the same object and print name value.
Add a method in A named print(){ SOP("in A "+name)};
Invoke print() using both the ref in tester class and verify what happens.
Now override the print() method in B with print(){SOP("in B "+name)};
Recompile and run the tester class. Are you understanding what is happening? Type of ref dictates which redeclared/hidden variable is picked. Type of object dictates which instance method body is picked.
Now make both the method and variable as static in both the classes.
Recompile and run and verify how statics work when redeclared.
Ask queries if you have any doubts.

5) Create an Animal class. An Animal can eat and sleep.
When an Animal is asked to sleep, it closes its eyes and sleeps.
We do not know how an Animal eats. What kind of method will you code for eat?
Mark eat method abstract (no body => public void eat();). See if Animal.java will compile. Mark Animal class abstract and see if the compilation succeeds. Create an instance variable in Animal as String name. Create a no-arg constructor and parameterised constructor (which accepts a string and assigns it to name). Put SOPs in both the constructors. Is compilation succeeding? Can you have constructors in an abstract class? Create a TestAnimal class with main().
Try to create an object of Animal. Does it work? Now create a Croc extends Animal, add a new method swim() with SOP,
create one no-arg constructor with SOP and parameterised constructor which accepts string and passes the string to parent constructor by invoking super(s). In Tester class, create a croc object and invoke eat() and sleep(). What SOPs are being printed out? Are you able to

understand this? Override sleep() in Croc and see which implementation gets picked up by executing same tester class.

6) Create an abstract class Device. Create a Device no-arg constructor with SOP. Now in TestDevice, create main(). Try to test if you can create an object of Device. Can you do so? Now add a static method to Device with an SOP. Now see if you can call this from TestDevice.
Now add a doSomething() instance method which is abstract.
Now create a subclass of Device called Phone. A Phone can be used to call a given number (String). How a phone calls, we do not know. What will you mark call() as? Is it still mandatory to override doSomething() in Phone? Check. Now create a MobilePhone that IS-A Phone. How many methods you will have to override in this class for compilation to succeed. Now create a Person with a name and use() method that accepts a Device d as paramter. The Person will make the device to doSomething when asked to use (meaning in use(Device d), invoke d.doSomething()). Test it by passing a MobilePhone object and verify which implementations are getting picked up.

7) Add additional code to check the following:
a) Can an abstract class/method be marked final? Why not?
b) Can an abstract class/method be marked static? Why not?
c) Can an abstract class inherit from another abstract class? Should any method have to be mandatorily overridden?
d) Can a concrete class be inherited by an abstract class? Test it.
e) Can an abstract class have initializer/constructors? Test it
f) Can you have a private abstract method? Test it.
g) Can you have a abstract class with final method?
h) Can you have a final class with abstract method?

For all these above, you have to just test if compilation succeeds.

8) Design an Item class. Item has a price(double) and a name and a serialNo(int). Generate a random serialNo in the range 0-1000 and assign to every item when created. Price and name must be accepted by the class user. Create 2 param constructor - one that accepts only a String and the other that accepts both string and double. Reuse one constructor in another (using this()).Also in the first constructor, name

the name local variable "name". Test Item creation in a tester class. Put SOPs in constructor / init / methods and verify the working.

9) A person has a name, Car, Dog and a favourite Song (reuse classes Car, Dog and Song from earlier labs). When you ask a Person to commute and give him a destination (String parameter), then he will start the car, drive the car and stop the car and print that he has reached the destination. When you ask the person to sing, he will sing (print) his favourite song with lyrics. When you ask a person to take a walk, he will take his dog for a walk and the dog will bark. Person has the ability to eat Food. Food has name and price. Food must be given to Person when you invoke eat(). When a person is asked to eat, he will specify that he is eating food with name and say out his name as well. Person also has a generatePrime() behaviour. When you give him a number as input, then he will generate all prime numbers until that number and print to monitor.

10) A Man has a name and can eat. When you ask him to eat, he will simply say so (SOP). A Woman has a name and can work. When you ask her to work, she will say so (SOP). A man has a wife. A woman has a husband (bi-directional 1..1 multiplicity has-a relationship). A man will get a wife once he marries. At the same time, the woman will get the husband. Implement marry() correctly in man to set both the ends of the relationship (do validations such that if one is married, they cannot marry again). Make all instance variables private in both the classes with setter/getters. A man has printWifesName() which will print his wife's name if he has a wife. A woman has printHubbysName() which will print her husbands name if she has a husband. Create a divorce() in man which will disconnect both the ends of the relationship. Then test the code for the following:

a) Ask a man who is not married to printWifesName() and check what is printed
b) Ask a man named Arbaz to marry a woman named Malaika, ask them to print each other spouses names
c) Try to make Arbaz marry Rosie (another woman) which shouldn't work
d) Divorce them and ask them to marry new people and check if it works