Uttara InfoSolutions

www.uttarainfo.com

Interfaces,CC,Initialisers Practicals

Basics:

1) Build an interface Pet

```
public interface Pet
{
        public void play();
}
```

and save it in Pet.java.
Compile it. Now try to add a constructor and see if compilation succeeds. Try to add a method with body and see if it compiles. Try to add a final method and see if it compiles. Add a variable without a value and see if compiles.

Change to:

```
public interface Pet
{
        int X = 10;
        String NAME = “Pingu";
        public void play();
}
```

Compile it.
Then create TestPet.java with:

```
…
main(..)
{
        System.out.println(Pet.NAME); // with it print the name or not, check.
        Pet p = new Pet(); // will this compile? check.
}
```

Then create Doggy.java:

```
public class Doggy implements Pet
{
        public void barky()
        {
                SOP(“woof woof”);
        }
}
```

Check without overriding any method, with the class compile. Then override the method play() with SOP. Check if it compiles.

In main(),

Pet p = new Doggy();

what methods you can invoke using p and why? check and run.

2) Create a class A. Create a final int p variable in it. Compile it without assigning any value. Now use a inst initalizer 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 inst init and then create a parameterized constructor where you accept an int and assign it to p. Now in tester class, create object by passing in an int to constr 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?

3) 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 inst variable with parameterized constructors)

4) Create a class X. Create an int field a and assign 10 to it.
Create an instance initializer and put in SOP("in inst init 1 a = "+a);
and change a value to 20.
Create another instance initializer and put in SOP("in inst init 2 a = "+a);
and change a value to 30.
Create a no-arg constructor, print value of a and then assign 40 to it.
In a Tester class, in main()-> create object of X and then print obj.a to the monitor.
Do you understand how the initialisation is happening?
Now add a static variable in X named b = 15. Create 2 static init where you print the value of b and change them like earlier. Add printing of X.b in Tester class and then run it. Do you understand now how the initialisation is occuring. Create one more object of X and see if the static initializers are getting fired.
Now create a subclass of X named Y with same class defn (copy paste) except change the name of var to c and d and SOPs to indicate Y class.
Now in Tester class, create object of Y. Think what should be the order of execution of init and constructors. Verify if that is correct.
Ask doubts if you have any.
X.java->
public class X
{

```
	int a = 10;
	static int b = 15;
	static
	{
		System.out.println("in static init 1 b = "+b);
		b = 25;
	}
	{
		System.out.println("in inst init 1 a = "+a);
		a = 20;
	}
	{
		System.out.println("in inst init 2 a = "+a);
		a = 30;
	}
	static // does order matter for init execution?
	{
		System.out.println("in static init 2 b = "+b);
		b = 35;
	}
	public X()
	{
		System.out.println("in constr of X a = "+a);
		a = 40;
	}
}
```

Y.java->

```
public class Y extends X
{
	int c = 10;
	static int d = 25;
	static
	{
		System.out.println("in static init 1 d = "+d);
		d = 35;
	}
	static
	{
		System.out.println("in static init 1 d = "+d);
		d = 45;
	}
	{
		System.out.println("in inst init 1 c = "+c);
		c = 20;
	}
	{
		System.out.println("in inst init 2 c = "+c);
		c = 30;
```

```
        }
        public X()
        {
                System.out.println("in constr of X c = "+c);
                c = 40;
        }
}
```

5) Create a class A with inst 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 constr.
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 inst method body is picked.
Now make both the method and variable as static in both the classe.
Recompile and run and verify how statics work when redeclared.
Ask queries if you have any doubts.

6) 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.

7) Try these permutations combinations by creating simple interfaces/classes.
a) Can an interface extend multiple interfaces?
b) Can a class implement multiple interfaces?
c) What happens when a class implements multiple interfaces with same method declaration?

d) Interface I1 has m1() and Interface I2 has m1() and m2(). m1() in I1 returns void and m1() in I2 returns int. Now create a class C1 that implements both I1 and I2. What methods will have to override? Does compilation succeed? Why?
e) Abstract class A1 implements I2 and overrides m1(). Now class C2 extends A1. Is it necessary to override any methods in C2? Put SOPs in all methods.

Problems:

Problem 1:

3) Create an interface Thing.
4) Add a method doSomething() to Thing interface. What kind of method is this?
5) Create a Tool class and implement Thing interface. Override the doSomething() and add a SOP("Thing doing something").
6) Create a Spanner that extends Tool, override doSomething() and add a SOP.
7) A TubeLight is a Thing as well. It renders light when you ask it to do something (simple SOP).
8) In a tester class, create a Spanner object, point it to by a Thing ref, Tool ref and Spanner ref and invoke doSomething(). See which implementation gets picked up.
9) Create a Vehicle class that implements Thing. A Vehicle has a name and when asked to doSomething, it will drive (sop).
10) Create a Person class. Create a testThing() method which takes in any thing as a parameter. What kind of method is this which should work for all Things?
11) In the tester class, create a Person object,a spanner, a vehicle object and ask the person to test the spanner and vehicle. Examine the SOP output and verify which implementations are being picked up and why?

Go through the example code given to you to verify.

Problem 2:

There are ducks in a pond. Ducks swim. They swim according to their sizes.
Ducks come in sizes chota, bada and biggest which are int values (5, 10, 15) and are constants (where can you hold constants?).
Provide a constructor in Duck which takes in the int size to store the state of the duck. When swim
is invoked, they swim fastest, slow, slowest correspondingly.
Keep a count to keep track of the number of duck objects created as well (single copy variable incremented each time object is created?).
Write a tester class. Create 3 Duck objects with each of the mentioned sizes. Ask them to swim. Verify if they are swimming as per the sizes. Also try to create a Duck object by passing in a different int value, there should be an error displayed.
Use the constant name to pass the value into the constructor and not a int value directly. Print the number of duck objects created as well.

Problem 3:

Create a Stack as an interface with following code

```
public interface Stack
{
	public void push(int element);
	public int pop();
	public int peek();
	public void printElements()
}
```

Create a class called ArrayStack that implements Stack interface like this:

ArrayStack has ints -> 1..n multiplicity which means you can store them in an array instance variable.

```
public class ArrayStack implements Stack
{
	int[] arr = new int[10]; // instance variable where elements will be stored

	int count; // to keep a track of how many elements are filled into the stack

	public void push(int element)
	{
		1. check if count is < length of array
		2. store element into arr[count].
		3. increment count.
	}
	public int pop()
	{
		1. check if count >0
		2. return arr[count].
		3. decrement count

	}
	public int peek()
	{
		1. check if count > 0
		2. return arr[count]  without decrementing count.
	}
	public void printElements()
	{
		1. Loop over arr count num of times
		2. print each element via SOP
```

```
        }
}
```

Build a tester class to test ArrayStack something like this:

```
main(..)
{
        ArrayStack myStack = new ArrayStack();
        myStack.push(5);
        myStack.push(10);
        myStack.push(15);
        myStack.printElements(); -> should print 15 10 5
        SOP(myStack.peek()); -> should print 15
        SOP(myStack.pop());  -> should print 15
        SOP(myStack.peek()); -> should print 10
        myStack.printElements(); -> should print 10 5
        myStack.push(20);
        myStack.printElements(); -> should print 20 10 5

        // test all boundary conditions!
}
```

Problem 4:

Build a StringReverser interface. A StringReverser is one which can reverse a string. Create two different imply of reversing a string. Test out both the implementations.