

State the benefits of encapsulation in object oriented design and write code that implements tightly encapsulated classes and the relationships "is a" and "has a".

Encapsulation means the internal wirings (variables/fields/properties) of the class are not visible outside. Instead, the class provides accessor (getter and setter) methods for the properties it supports. In other words, have private fields and have public setters and getters.

Consider the following declarations:

```
public class GearBox
{
    public void shiftGear(int gearNo){ ... }
}
public interface Movable{ }
```

```
public class Car implements Movable
{
    private GearBox gb = new GearBox();
}
```

```
public class SportsCar extends Car
{
}
```

In the above situation:

- 1.SportsCar is-a Car ( because SportsCar extends Car )
- 2.Car has-a GearBox ( because Car has a variable of class GearBox )
- 3.SportsCar has-a GearBox ( because SportsCar is a Car and Car has a GearBox )
- 4.Car and SportsCar is-a Movable (because Car implements Movable and SportsCar extends Car)

Although, this is strictly an is-like-a relationship but still can be clubbed together with is-arelationship.

Write code to invoke overridden or overloaded methods and parental or overloaded constructors; and describe the effect of invoking these methods.

Points to remember: . Overloaded methods are entirely independent methods. Two overloaded methods behave as if they were two methods with different names. Method name is NOT important, it's the signature (ie. method name + parameterlist) that governs the behavior and overloaded methods have different signatures.

Write code to construct instances of any concrete class including normal top level classes, inner classes, static inner classes, and anonymous inner classes.

---

Overriding - same method names with same arguments and same return types associated in a class and its subclass.

Overloading - same method name with different arguments, may or may not be same return type written in the same class itself.

---

Don't get overriding and overloading confused.

You can think of overloading as expanding the versions of a method along the horizontal axis. For example, here we show class A with several overloaded versions of the do() method:

```
public class A {  
    ...  
    void do ()  
    {...}  
    void do (int i)  
    {...}  
    void do (float x)  
    {...}  
    void do (float x, float y)  
    {...}  
}  
A: do () do (int i) do (float x) do (float x, float y)
```

We display the various versions of do() in columns on the horizontal axis.

Below we show two subclasses of class A. Inheritance allows subclasses to override some or all of these versions of do() and also overload new ones. We show the methods of the base class and subclasses vertically below the base class.

Here class B extends class A and overrides three of the do() methods. Class C extends class B and overrides the do(int i) method in class B:

```

public class B extends A
{
    ...
    void do (int i)
    {..}
    void do (float x, float y)
    {..}
    void do (float x, int k)
    {..}
}

```

```

public class C extends B
{
    ...
    void do (int i)
    {..}
}

```

A: do () do (int i) do (float x) do (float x,float y)

B: " do (int i) " do (float x,float y) do (float x,int k)

C: " do (int i) " " "

Here the quotation mark " indicates that the code of the inherited method above it is used. Where a class method signature is written, the method overrides the inherited version.

The base class A overloads the do() function with four different variations.

Class B is a subclass of A and inherits its four methods but overrides two of them and overloads with a new one of its own.

Class C is a subclass of B and inherits two do() methods from class A and two from class B. It overrides one method from class B.