Abstract method program (set0):

```
package com.ravi.abstract_method;

abstract class Vehicle
{
    protected String name;

    public Vehicle(String name)
    {
        super();
        this.name = name;
    }

    public abstract void run();
}

class Bike extends Vehicle
{
    public Bike(String sirname)
    {
        super(sirname);
    }

    @Override
    public void run()
    {
        IO.println(name+" is running");
    }
}

class Car extends Vehicle
{
    public Car(String carname)
    {
        super(carname);
    }

    @Override
    public void run()
    {
        IO.println(name+" is running");
    }
}

public class DistractOmeet
{
    public static void main(String[] args)
    {
        Vehicle vehicle = null;
        vehicle = new Car("Nexon", 2000); vehicle.run();
        vehicle = new Bike("Bullet"); vehicle.run();


    }
}
```

```
package com.ravi.abstract_method;

abstract class Animal
{
    protected String name;

    public Animal(String name)
    {

        super();
        this.name = name;
    }

    public abstract void healthCheckup();
}

class Lion extends Animal
{
    public Lion(String name)
    {

        super(name);
    }

    @Override
    public void healthCheckup()
    {
        IO.println(name+" lion is going for regular health check up");
    }
}

class Elephant extends Animal
{
    public Elephant(String name)
    {
        super(name);
    }

    @Override
    public void healthCheckup()
    {
        IO.println(name+" is going for regular health check up");
    }
}

public class DistractOmeet
{
    public static void main(String[] args)
    {
        Lion lion[] = {new Lion("Simba"), new Lion("Mufasa"), new Lion("Kaja") };
        HealthCheckup(lion);

        IO.println("......................");

        Elephant elephant[] = {new Elephant("Dumbo"), new Elephant("Tusk"),};
        HealthCheckup(elephant);


    }

    public static void HealthCheckup(Animal ...animal)
    {
        for(Animal animal : animal)
        {
            animal.healthCheckup();
        }
    }
}
```

*** Downcasting = instanceof**

```
package com.ravi.abstract_method;

abstract class Animal
{
    protected String name;

    public Animal(String name)
    {
        super();
        this.name = name;
    }

    public abstract void healthCheckup();
}

class Lion extends Animal
{

    public Lion(String name)
    {

        super(name);
    }

    @Override
    public void healthCheckup()
    {
        IO.println(name+" lion is going for regular health check up");
    }

    public void roar()
    {
        IO.println(name+" lion is roaring");
    }

}
class Elephant extends Animal
{
    public Elephant(String name)
    {
        super(name);
    }

    @Override
    public void healthCheckup()
    {
        IO.println(name+" is going for regular health check up");
    }

    public void trumpet()
    {
        IO.println(name+" is trumpeting...");
    }

}

public class DistractOmeet
{
    public static void main(String[] args)
    {
        Lion lion[] = {new Lion("Simba"), new Lion("Mufasa"), new Lion("Kaja") };
        HealthCheckup(lion);

        IO.println("......................");

        Elephant elephant[] = {new Elephant("Dumbo"), new Elephant("Kaja"),};
        HealthCheckupComp(elephant);


    }

    public static void HealthCheckup(Animal ...animal)
    {
        for(Animal animal : animal)
        {
            if(animal instanceof lion lion)
            {
                lion.healthCheckup();
                lion.roar();
            }
            else if(animal instanceof Elephant elephant)
            {
                elephant.healthCheckup();
                elephant.trumpet();
            }

        }
    }

}
```

HAS-A Relation :

* HAS-A relation can achieve by using Association concept.
* Association describes how much a class known about another class.
* Association is further classified into two categories :

*** If we use a class as a property to another class then it is called HAS-A relation.**

```
Example :
-----------
public class Customer                          public class Order
{                                              {
    private int cusId;                             private int orderId;
    private String cusName;                        private String itemName;
    private int cusPhoneNumber;                    private int orderDate;
    private String address;                        private int quantity;
    private Order order; //HAS-A Relation      }
}
```


Composition (Strong Reference)

Composition in Java is a way to design classes such that one class contains an object of another class. It is a way of establishing a "HAS-A" relationship between classes.

Composition represents a strong relationship between the containing class and the contained class.If the containing object (Car object) is destroyed, all the contained objects (Engine object) are also destroyed.

A car has an engine. Composition makes strong relationship between the objects. It means that if we destroy the car object, the engine object will be also destroyed with it. For example, if the Car is destroyed the engine will also be destroyed as well.

Program Guidelines :
----
1) One object cannot exist without another object
2) Car object can't exist without engine object.In OOJ data, during the creation of Car object, Engine object should be automatically created.
3) Car object has one black first field so, engine initialization is compulsory as well as we cannot change the Engine object.

```
package com.ravi.has_a_relo

public class Engine
{
  private String type;
  private int horsePower;

  public Engine(String type, int horsePower)
  {

    super();
    this.type = type;
    this.horsePower = horsePower;
  }

  @Override
  public String toString() {
    return "Engine [type= " + type + ", horsePower= " + horsePower + "]";
  }
}


package com.ravi.has_a_relo;

public class Car
{
  private String carName;
  private int carModel;
  private Engine engine; //HAS-A Relation

  public Car(String carName, int carModel)
  {
    super();
    this.carName = carName;
    this.carModel = carModel;
    this.engine = new Engine("Battery", 1000);  //Composition
  }

  @Override
  public String toString()
  {
    return "Car [carName= " + carName + ", carModel= " + carModel + ", engine= " + engine + "]";
  }


}


  package com.ravi.has_a_relo;

  public class CompositionDemo {

      public static void main(String[] args)
      {
        Car car = new Car("Nexon", 2000);
        IO.println(car);
      }

}
```