**OOPS Interview Questions and Answers**

| Class | Package | Subclass | Subclass | World

| | |(same pkg)|(diff pkg)|

————————————+———————+—————————+——————————+——————————+————————

public | + | + | + | + | +

————————————+———————+—————————+——————————+——————————+————————

protected | + | + | + | + |

————————————+———————+—————————+——————————+——————————+————————

no modifier | + | + | + | |

————————————+———————+—————————+——————————+——————————+————————

private | + | | | |

+ : accessible

blank : not accessible

Welcome to OOPS interview questions and answers. There are many Object Oriented Programming languages such as Java, C++ and Python. Having a clear idea about OOPS concepts is very important if you are going to face any interview on these programming languages. That’s why I thought to share the top OOPS interview questions with you and provide detailed answers for them.

### What is OOPS?

Object Oriented Programming System is the programming technique to write programs based on the real world objects. The states and behaviors of an object are represented as the member variables and methods. In OOPS programming programs are organized around objects and data rather than actions and logic.

**State**: what the objects have, Student have a first name, last name, age, etc

**Behavior**: what the objects do, Student attend a course "Java for beginners"

**Identity**: what makes them unique, Student have Student-ID-number, or an email which is unique. (this is important when implementing the equals method, to determine if the objects are different or not)

### What are the advantages of OOPS concepts?

Major advantages of OOPS programming are;

* 1. **Simplicity**: OOPS programming objects model real world objects, so the complexity is reduced and the program structure is clear.
  2. **Modularity**: Each object forms a separate entity whose internal workings are decoupled from other parts of the system.
  3. **Modifiability**: It is easy to make minor changes in the data representation or the procedures in an OO program. Changes inside a class do not affect any other part of a program, since the only public interface that the external world has to a class is through the use of methods.
  4. **Extensibility**: Adding new features or responding to changing operating environments can be solved by introducing a few new objects and modifying some existing ones.
  5. **Maintainability**: Objects can be maintained separately, making locating and fixing problems easier.
  6. **Reusability**: Objects can be reused in different programs.

### What is the difference between Procedural programming and OOPS?

|  |  |
| --- | --- |
| **Procedural** | **OOPS** |
| Procedural language is based on functions | object oriented language is based on real world objects. |
| Procedural language gives importance on the sequence of function execution | object oriented language gives importance on states and behaviors of the objects. |
| Procedural language exposes the data to the entire program | object oriented language encapsulates the data |
| Procedural language is complex in nature so it is difficult to modify, extend and maintain | object oriented language is less complex in nature so it is easier to modify, extend and maintain. |
| Procedural language provides less scope of code reuse | object oriented language provides more scope of code reuse. |
|  |  |

### What are the core concepts of OOPS?

OOPS core concepts are;

* 1. Abstraction
  2. Encapsulation
  3. Polymorphism
  4. Inheritance
  5. Composition
  6. Association
  7. Aggregation

### What is Abstraction?

Abstraction is an OOPS concept to construct the structure of the real world objects. During this construction only the general states and behaviors are taken and more specific states and behaviors are left aside for the implementers.

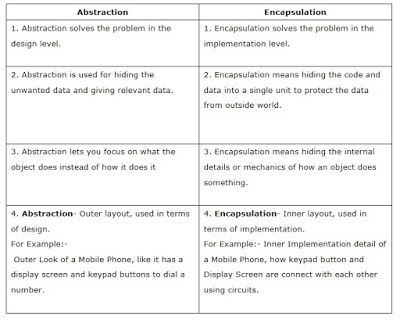
Abstraction in Java is achieved by using interface and abstract class in Java.  
One example of Abstraction is creating interface to denote common behavior without specifying any details about how that behavior works e.g. You create an [interface](http://javarevisited.blogspot.sg/2012/04/10-points-on-interface-in-java-with.html)called Server which has the start() and stop() method. This is called abstraction of Server because every server should have a way to start and stop and details may differ. 

What is Encapsulation?

Encapsulation is an OOPS concept to create and define the permissions and restrictions of an object and its member variables and methods. A very simple example to explain the concept is to make the member variables of a class private and providing public getter and setter methods. Java provides four types of access level modifiers: public, protected, no modifier and private.

### What is the difference between Abstraction and Encapsulation?

* 1. “Program to interfaces, not implementations” is the principle for Abstraction and “Encapsulate what varies” is the OO principle for Encapsulation.
  2. Abstraction provides a general structure of a class and leaves the details for the implementers. Encapsulation is to create and define the permissions and restrictions of an object and its member variables and methods.
  3. Abstraction is implemented in Java using interface and abstract class while Encapsulation is implemented using four types of access level modifiers: public, protected, no modifier and private.

1. **What is the difference between Abstraction and Encapsulation in Java?**  
   Even though both Abstraction and Encapsulation looks similar because both hide complexity and make the external interface simpler there is a subtle difference between them. Abstraction hides logical complexity while Encapsulation hides Physical Complexity.  
     
   [](https://1.bp.blogspot.com/-Y3M-YJIsyog/WTOU8aFYUjI/AAAAAAAAIwM/dYpkD6Ur46g2jBhxJVSC5y1Joorn1VTigCLcB/s1600/Difference+between+Abstraction+and+Encapsulation.jpg)

### What is Polymorphism?

Polymorphism is the occurrence of something in various forms. Java supports various forms of polymorphism like polymorphic reference variables, polymorphic method, polymorphic return types and polymorphic argument types.

### What is Inheritance?

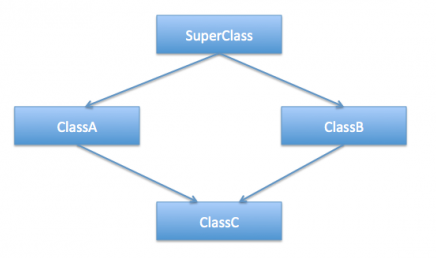
A subclass can inherit the states and behaviors of it’s super class is known as inheritance.

### What is multiple inheritance?

A child class inheriting states and behaviors from multiple parent classes is known as multiple inheritance.

### What is the diamond problem in inheritance?

In case of multiple inheritance, suppose class A has two subclasses B and C, and a class D has two super classes B and C.If a method present in A is overridden by both B and C but not by D then from which class D will inherit that method B or C? This problem is known as diamond problem.



To understand diamond problem easily, let’s assume that multiple inheritance was supported in java. In that case, we could have a class hierarchy like below image.

Let’s say SuperClass is an [**abstract class**](https://www.journaldev.com/1582/abstract-class-in-java) declaring some method and ClassA, ClassB are concrete classes.

public class ClassA extends SuperClass{

@Override

public void doSomething(){

System.out.println("doSomething implementation of A");

}

public class ClassB extends SuperClass{

@Override

public void doSomething(){

System.out.println("doSomething implementation of B");

}

Public class ClassC extends ClassA, ClassB{

public void test(){

//calling super class method

doSomething();

}

}

Diamond problem in java is the main reason java doesn’t support multiple inheritance (by extends keyword) in classes.

This issue resolved by multiple interface implementation in java

### Why Java does not support multiple inheritance?

Java was designed to be a simple language and multiple inheritance introduces complexities like diamond problem. Inheriting states or behaviors from two different type of classes is a case which in reality very rare and it can be achieved easily through an object association.

### What is Static Binding and Dynamic Binding?

Static or early binding is resolved at compile time. Method overloading is an example of static binding.

Dynamic or late or virtual binding is resolved at run time. Method overriding is an example of dynamic binding.

### What is the meaning of “IS-A” and “HAS-A” relationship?

“IS-A” relationship implies inheritance. A sub class object is said to have “IS-A” relationship with the super class or interface. If class A extends B then A “IS-A” B. It is transitive, that is, if class A extends B and class B extends C then A “IS-A” C. The “instanceof” operator in java determines the “IS-A” relationship.

When a class A has a member reference variable of type B then A “HAS-A” B. It is also known as Aggregation.

### What is Association?

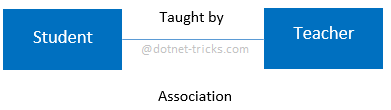
It represents a relationship between two or more objects where all objects have their own lifecycle and there is no owner.

Association is a relationship between two objects with multiplicity.

This is represented by a solid line.

http://www.dotnettricks.com/img/oops/association.png

Let’s take an example of relationship between Teacher and Student. Multiple students can associate with a single teacher and a single student can associate with multiple teachers. But there is no ownership between the objects and both have their own lifecycle. Both can be created and deleted independently.



### What is Aggregation?

### Aggregation is a weak association.

It is a specialized form of Association where all object have their own lifecycle but there is ownership. This represents “whole-part or a-part-of” relationship. This is represented by a hollow diamond followed by a line.

http://www.dotnettricks.com/img/oops/aggregation.png

Let’s take an example of relationship between Department and Teacher. A Teacher may belongs to multiple departments. Hence Teacher is a part of multiple departments. But if we delete a Department, Teacher Object will not destroy.

Teacher has ownership for department.



### What is Composition?

Composition is a special form of Aggregation where the part cannot exist without the whole.

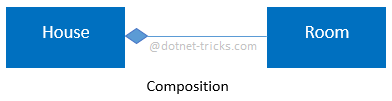
Composition is a strong Association.

Composition relationship is represented like aggregation with one difference that the diamond shape is filled.

It is a specialized form of Aggregation. It is a strong type of Aggregation. In this relationship child objects does not have their lifecycle without Parent object. If a parent object is deleted, all its child objects will also be deleted. This represents “death” relationship. This is represented by a solid diamond followed by a line.

http://www.dotnettricks.com/img/oops/composition.png

Let’s take an example of relationship between House and rooms. House can contain multiple rooms there is no independent life of room and any room cannot belongs to two different house if we delete the house room will automatically delete.



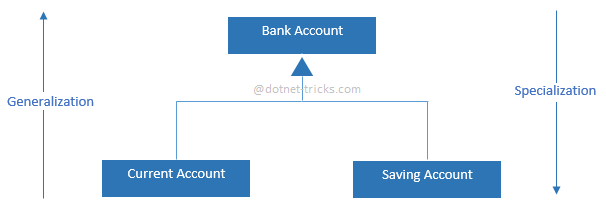
Let’s take another example of relationship between Questions and options. Single questions can have multiple options and option cannot belong to multiple questions. If we delete questions options will be automatically deleted.

Difference Aggregation & composition

|  |  |  |
| --- | --- | --- |
|  | **Aggregation** | **Composition** |
| 1 | It is a specialized form of Association where all object have their own lifecycle but there is ownership | It is a specialized form of Aggregation |
| 2 | all object have their own lifecycle but there is ownership | Composition is a special form of Aggregation where the part cannot exist without the whole. |
| 3 | Aggregation is a weak association. | Composition is a strong Association. |
| 4 | This is represented by a hollow diamond followed by a line. | Composition relationship is represented like aggregation with one difference that the diamond shape is filled. |
| 5 |  | This represents “death” relationship |
|  |  |  |

**Generalization Vs Specialization // asked in interview**

|  |  |
| --- | --- |
| **Generalization** | **Specialization** |
| The process of extracting common characteristics from two or more classes and combining them into a generalized superclass, is called Generalization. The common characteristics can be attributes or methods. Generalization is represented by a triangle followed by a line.  http://www.dotnettricks.com/img/oops/generalizationarrow.png | Specialization is the reverse process of Generalization means creating new sub classes from an existing class.  Let’s take an example of Bank Account; A Bank Account is of two types – Current Account and Saving Account. Current Account and Saving Account inherits the common/ generalized properties like Account Number, Account Balance etc. from a Bank Account and also have their own specialized properties like interest rate etc. |



### What is Dependency?

When one class depends on another because it uses that at some point in time then this relationship is known as Dependency. One class depends on another if the independent class is a parameter variable or local variable of a method of the dependent class. A Dependency is drawn as a dotted line from the dependent class to the independent class with an open arrowhead pointing to the independent class.

### What is the difference between Association and Dependency?

The main difference between Association and Dependency is in case of Association one class has an attribute or member variable of the other class type but in case of Dependency a method takes an argument of the other class type or a method has a local variable of the other class type.

### What is a Class?

A class is the specification or template of an object.

### What is an Object?

Object is instance of class.

### OOPS : Abstraction, Encapsulation, Inheritance, Polymorphism

**OOPS Features**

\* The object oriented programming (OOP) is a programming model   
  
where Programs are organized around object and data rather than   
  
action and logic.   
  
  
\* OOP allow decomposition of a problem into a number of entities called  
  
Object and then builds data and function around these objects.

* The Program is divided into number of small units called Object. The data and function are build around these objects.
* The data of the objects can be accessed only by the functions associated with that object.
* The functions of one object can access the functions of other object.

OOP has the following important features.

**Class:**

A class is the core of any modern Object Oriented Programming language such as C#.

In OOP languages it is must to create a class for representing data.

Class is a blueprint of an object that contains variables for storing data and functions to performing operations on these data.

Class will not occupy any memory space and hence it is only logical   
  
representation of data.

To create a class, you simply use the keyword "class" followed by the class name:

class Employee

{

}

**Object:**  
  
Objects are the basic run-time entities in an object oriented system.They may represent a person,a place or any item that the program has to handle.   
  
"Object is a Software bundle of related variable and methods. "

“Object is an instance of a class”

Class will not occupy any memory space. Hence to work with thedata represented by the class you must create a variable for the class, which is called as an object.

When an object is created by using the keyword **new**, then memory will be allocated for the class in heap memory area, which is called as an instance and its starting address will be stored in the object in stack memory area.

 When an object is created without the keyword new, then memory will not be allocated in heap I.e. instance will not be created and object in the stack contains the value **null.**

When an object contains null, then it is not possible to access the members of the class using that object.

class Employee

{

}

Syntax to create an object of class Employee:-

Employee objEmp = new Employee();

All the programming languages supporting object oriented Programming will be supporting these three main concepts:

1. Encapsulation
2. Inheritance
3. Polymorphism

**Abstraction:**

Abstraction is "To represent the essential feature without representing the back ground details."

Abstraction lets you focus on what the object does instead of how it does it.

Abstraction provides you a generalized view of your classes or object by providing relevant information.

Abstraction is the process of hiding the working style of an object, and showing the information of an object in understandable manner.

**Real world Example of Abstraction:** -

Suppose you have an object Mobile Phone.

Suppose you have 3 mobile phones as following:-

Nokia 1400 (Features:- Calling, SMS)

Nokia 2700 (Features:- Calling, SMS, FM Radio, MP3, Camera)

Black Berry (Features:-Calling, SMS, FM Radio, MP3, Camera, Video Recording, Reading E-mails)

Abstract information (Necessary and Common Information) for the object "Mobile Phone" is make a call to any number and can send SMS."

so that, for mobile phone object you will have abstract class like following:-

    abstract class MobilePhone

    {

        public void Calling();

        public void SendSMS();

    }

    public class Nokia1400 : MobilePhone

    {

    }

    public class Nokia2700 : MobilePhone

    {

        public void FMRadio();

        public void MP3();

        public void Camera();

    }

    public class BlackBerry : MobilePhone

    {

        public void FMRadio();

        public void MP3();

        public void Camera();

        public void Recording();

        public void ReadAndSendEmails();

    }

Abstraction means putting all the variables and methods in a class which are necessary.

For example: - Abstract class and abstract method.

Abstraction is the common thing.

example:

If somebody in your collage tell you to fill application form, you will fill your details like name, address, data of birth, which semester, percentage you have got etc.

If some doctor gives you an application to fill the details, you will fill the details like name, address, date of birth, blood group, height and weight.

See in the above example what is the common thing?

Age, name, address so you can create the class which consist of common thing that is called abstract class.

That class is not complete and it can inherit by other class.

**Encapsulation:**

Wrapping up data member and method together into a single unit (i.e. Class) is called Encapsulation.

Encapsulation is like enclosing in a capsule. That is enclosing the related operations and data related to an object into that object.

Encapsulation is like your bag in which you can keep your pen, book etc. It means this is the property of encapsulating members and functions.

    class Bag

    {

        book;

        pen;

        ReadBook();

    }

Encapsulation means hiding the internal details of an object, i.e. how an object does something.

Encapsulation prevents clients from seeing its inside view, where the behaviour of the abstraction is implemented.

Encapsulation is a technique used to protect the information in an object from the other object.

Hide the data for security such as making the variables as private, and expose the property to access the private data which would be public.

So, when you access the property you can validate the data and set it.

Example:

class Demo

{

   private int \_mark;

   public int Mark

   {

     get { return \_mark; }

     set { if (\_mark > 0) \_mark = value; else \_mark = 0; }

   }

 }

**Real world Example of Encapsulation:-**

Let's take example of Mobile Phone and Mobile Phone Manufacturer

Suppose you are a Mobile Phone Manufacturer and you designed and developed a Mobile Phone design(class), now by using machinery you are manufacturing a Mobile Phone(object) for selling, when you sell your Mobile Phone the user only learn how to use the Mobile Phone but not that how this Mobile Phone works.

This means that you are creating the class with function and by making object (capsule) of it you are making availability of the functionality of you class by that object and without the interference in the original class.

**Example-2:**

TV operation

It is encapsulated with cover and we can operate with remote and no need to open TV and change the channel.

Here everything is in private except remote so that anyone can access not to operate and change the things in TV.

**Inheritance:**

When a class acquire the property of another class is known as inheritance.

Inheritance is process of object reusability.

For example, A Child acquire property of Parents.

public class ParentClass

    {

        public ParentClass()

        {

            Console.WriteLine("Parent Constructor.");

        }

        public void print()

        {

            Console.WriteLine("I'm a Parent Class.");

        }

    }

    public class ChildClass : ParentClass

    {

        public ChildClass()

        {

            Console.WriteLine("Child Constructor.");

        }

        public static void Main()

        {

            ChildClass child = new ChildClass();

            child.print();

        }

    }

**Output:**

    Parent Constructor.  
    Child Constructor.  
    I'm a Parent Class.

**Polymorphism:**

Polymorphism means **one name many forms**.

One function behaves different forms.

In other words, "Many forms of a single object is called Polymorphism."

**Real World Example of Polymorphism:**

**Example-1:**

A Teacher behaves to student.

A Teacher behaves to his/her seniors.

Here teacher is an object but attitude is different in different situation.

**Example-2:**

Person behaves SON in house at the same time that person behaves EMPLOYEE in office.

**Example-3:**

Your mobile phone, one name but many forms

* As phone
* As camera
* As mp3 player
* As radio

**Extracting real world relationships from requirement**

The whole point of OOP is that your code replicates the real world object, thus making your code readable and maintainable. The time we say real world, real world have relationships. Let�s consider the simple requirement listed below:-

1. Manager is anemployee of XYZ limited corporation.
2. Manager uses a swipe card to enter XYZ premises.
3. Manager has workers who work under him.
4. Manager has the responsibility of ensuring that the project is successful.
5. Manager's salary will be judged based on project success.

If you flesh out the above 5 point requirement we can easily visualize 4 relationships:-

* Inheritance
* Aggregation
* Association
* Composition

Let�s understand them one by one.

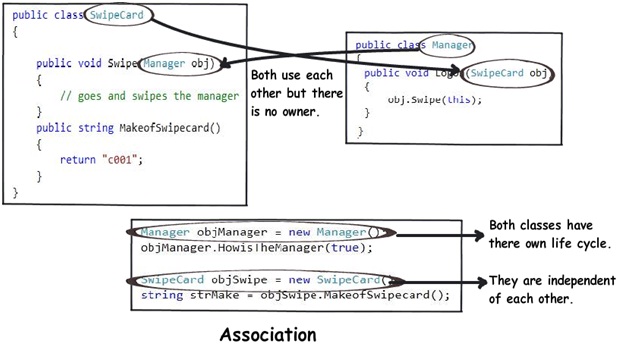
**Requirement 1 (The IS A relationship)**

If you see the first requirement (Manager is an employee of XYZ limited corporation) it�s a parent child relationship or inheritance relationship. The sentence above specifies that Manager is a type of employee, in other words we will have two classes one the parent class �Employee� and the other a child class �Manager� which will inherit from �Employee� class.

*Note: -The scope of this article is only limited to aggregation, association and composition. So we will not discuss inheritancein this article as its pretty straight forward and I am sure you can get1000 of articles on the net which will help you in understanding the same.*

**Requirement 2 (The Using relationship: - Association)**

The requirement 2 is an interesting requirement (Manager uses a swipe card to enter XYZ premises). In this requirement the manager object and swipe card object use each other but they have their own object life time. In other words they can exist without each other. The most important point in this relationship is that there is no single owner.



The above diagram shows howthe �SwipeCard� class uses the �Manager� class and the �Manager� class uses the �SwipeCard� class. You can also see how we can create the object of the �Manager� class and �SwipeCard� independently and they can have their own object life time.

This relationship is called as the �Association� relationship.

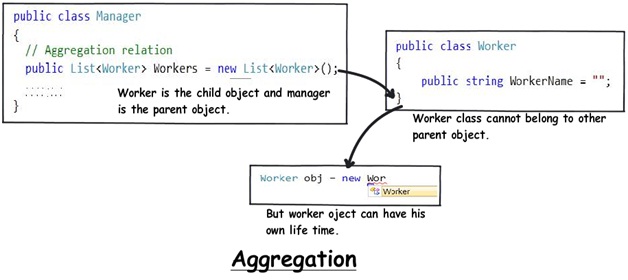
**Requirement 3 (The Using relationship with Parent: - Aggregation)**

The third requirement from our list (Manager has workers who work under him) denotes the same type of relationship like association but with a difference that one of them is an owner. So as per the requirement the �Manager� object will own �Workers� object.

The child �Worker� objects can not belong to any other objects. For instance the �Worker� object cannot belong to the �SwipeCard� object.

But But�.the �Worker� object can have his own life time which is completely disconnected from the �Manager� object. Looking from a different perspective it means that if the �Manager� object is deleted the �Worker� object does not die.

This relationship is termed as the �Aggregation� relationship.



**Requirement 4 and 5 (The Deathrelationship: - Composition)**

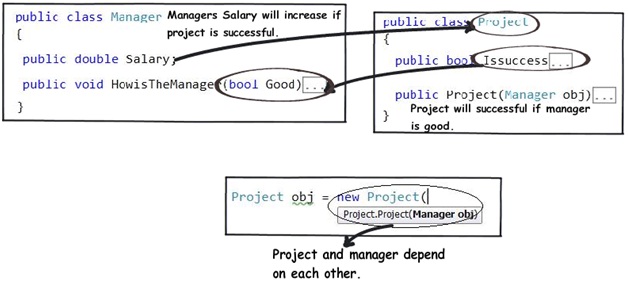
The last two requirements are actually logically one. If you read closely both the requirements which are as follows:-

1. Manager has the responsibility of ensuring that the project is successful.
2. Manager's salary will be judged based on project success.

Below is the conclusion from analyzing the above requirements:-

1. Manager and the project objects are dependent on each other.
2. The lifetimes of both the objects are same. In other words the project will not be successful if the manager is not good and manager will not get good increments if project has issues.

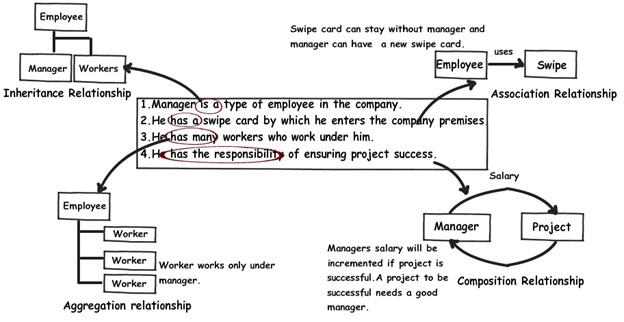
Below is how the class formation will look like. You can also see when I go to create the project object it needs the manager object.



This relationship is termed as the composition relationship. In this relationship both objects are heavily dependent on each other. In other words if goes for garbage collection the other also has to garbage collected , or putting from a different perspective the life time of the objects are same. That�s why I have put in the heading �Death� relationship.

**Putting things together**

Below is a visual representation of how the relationships have emerged from the requirements.



**The source code**

You can also download source code for this article

**Summarizing**

To avoid confusion hence forth in these 3 terms I have put forward a table below which will help us compare them from 3 angles owner , life time and child object.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Association** | **Aggregation** | **Composition** |
| **Owner** | No Owner | Single owner | Single Owner |
| **Life time** | Have their own life time | Have their own life time. | Owners life time |
| **Child object** | No Child objects all are independent | Child objects belong to single parent. | Child objects belong to single parent. |

**Video on Association, Aggregation and Composition**

## Loose Coupling Between Java Objects

Let us see loose coupling between java objects, take an example..

In order to over come tight coupling between objects, spring framework uses dependency injection mechanism with the help of POJO/POJI model and through dependency injection its possible to achieve loose coupling

In the above example Traveler , Car are tightly coupled.  If we want to achieve loose coupling between the objects Traveler and Car, we need to re-write the application like….



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | class Traveler  {      Vehicle v;      public void setV(Vehicle v)      {        this.v = v;      }        void startJourney()      {         v.move();      }  } |



|  |  |
| --- | --- |
| 1  2  3  4 | Interface Vehicle  {     void move();  } |



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | class Car implements Vehicle  {      public void move()      {           // logic      }  } |



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | class Bike implements Vehicle  {      public void move()      {           // logic      }  } |

In above example, spring container will inject either Car object or Bike object into the Traveler by calling setter method, So if Car object is replaced with Bike then no changes are required in Traveler class, this means there is loose coupling between Traveler and Vehicle object.  Actually setter method will be activated through **xml** file, will see this later for now we are good to go ;)

## Tight Coupling Between Java Objects

Let us see tight coupling between java objects first, take an example..



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | class Traveler  {      Car c=new Car();      void startJourney()      {         c.move();      }  }    class Car  {    void move()    {       // logic...    }  } |

* In the above example, Traveler object is depends on car object.  So traveler class creating an object of Car class inside it [ line number 3 ]
* If the other class object is created in the dependent class [ like Car class object in Traveler class ], there exist tight coupling, i mean if method in car object is changed then we need to do the changes in the Traveler class too so its the tight coupling between Traveler and Car class objects