Has-a relationship:

class Engine{ // dependent object

}

class car { // target object object

// to establish has-a relationship between these two classes

Engine engine = new Engine();

// to establish has-a relationship another class reference should be a part of this class reference.

// Engine object should be created and it should be injected to car object .

Engine engine ; // engine is a variable of reference type inside the class , so it can be said as instance variable.

}

Target means the one which can be given to user , that is target object .

The process of injecting depending object into target object is called as dependency injection.

Since engine is a instance variable it can be initialized in 2 ways ,

1. Through constructor
2. Through setter

Dependency injection is achieved in two ways .

1. Constructor dependency injection
2. Setter dependency injection

Constructor dependency injection : injecting dependant object to target object through constructor is called constructor dependency injection.

Setter dependency injection : injecting dependant object to target object through setter is called setter dependency injection .

Relationships in java :

=================

As a part of java application development we have to use entities as per entities ( classes ) as per application requirements .

In java application development , if we want to provide optimizations over memory utilization , code reusability , Execution time , sharing we have to define relationship between entities .

There are 3 types of relationships between entities

1. IS-A relationship
2. HAS-A relationship (extensively used in projects)
3. USE-A relationship (not popular)

Difference between IS-A relationship and HAS-A relationship

|  |  |
| --- | --- |
| IS-A | HAS-A |
| IS-A relation define inheritance between classes in java application , it will improve code reusability in applications . | HAS-A relation define association between classes in java application , it will improve communication between entities ( classes ) and data navigation between classes . |

// Communication between two entities , data navigation is explained below one-one association diagram.

Associations in java :

There are 4 types of association in java :

1. one to one Association (1:1)
2. one to many Association (1:M)
3. many to one Association ( M:1)
4. many to many Association (M:M)

Association : relating 2 classes in HAS-A style is called Association .

class Address{

---

}

class Account{

----

}

Class Employee{

----

Account account // if reference variable created like this 1:1 association ( one employee will have one salaried account )

Address[] address // 1:M ( one employee can have many address temporary home address , permenant home address , office address etc . to store many information of same type we can use array )

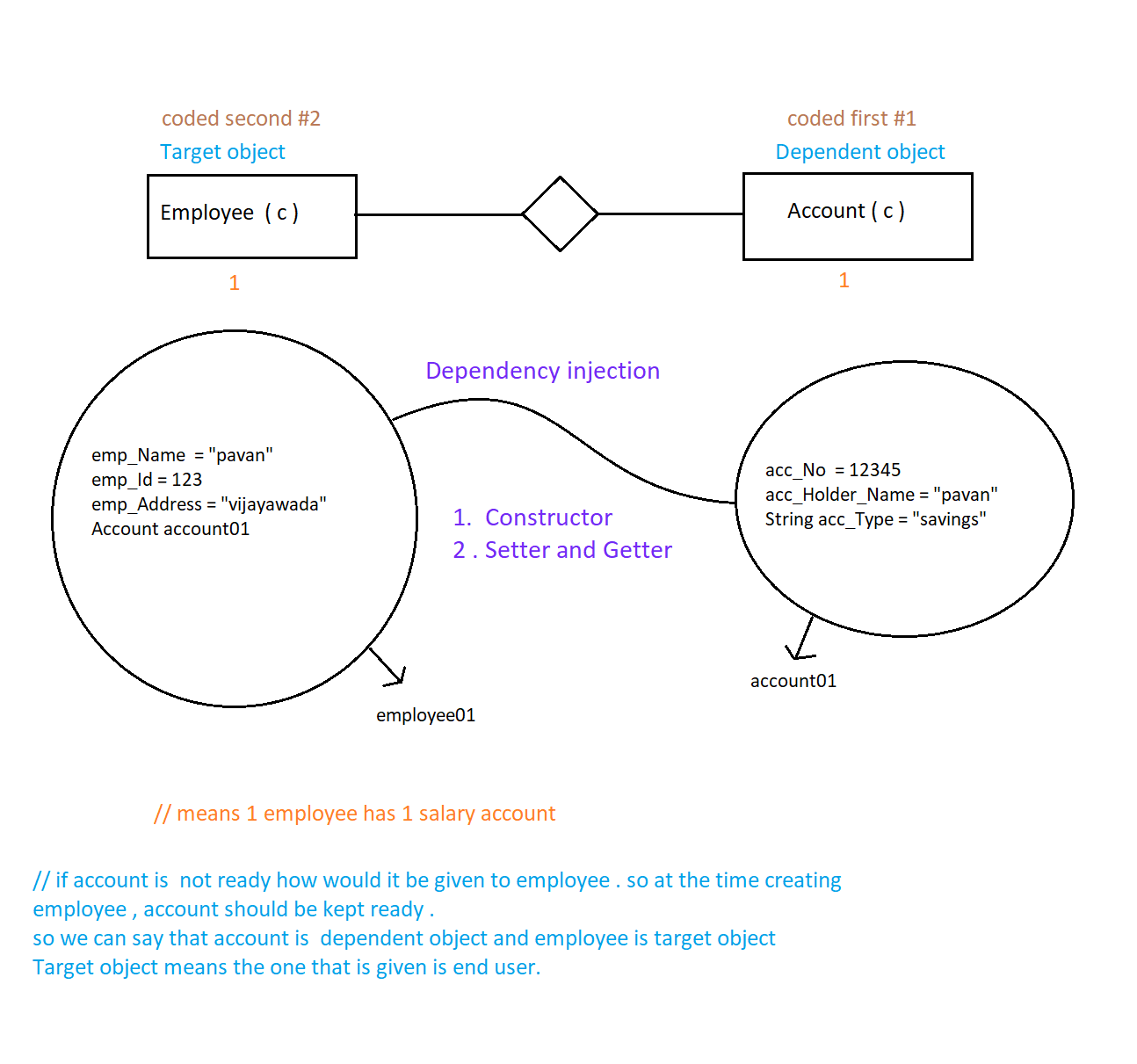
}

Note : predefined classes and methods we use in java mostly are based on inheritance . so without inheritance java cannot be made so popular .

Eg: Student01 , PrimitieValue-Injection

// the above example contains only primitive injection it does not contain any association .

Eg: Account01 , Employee01 , One\_One\_Association\_Using\_Constructor.



The has-a relationship only says there is a Communication between two entities.

Being in employee class you can get the data from account object is called data navigation (navigation means moment , from target object you are moving to dependent object for data)

Here account01 object is been injected into the employee01 object.

Here at dependent level one object is present and target level one object is present , so one -one association.

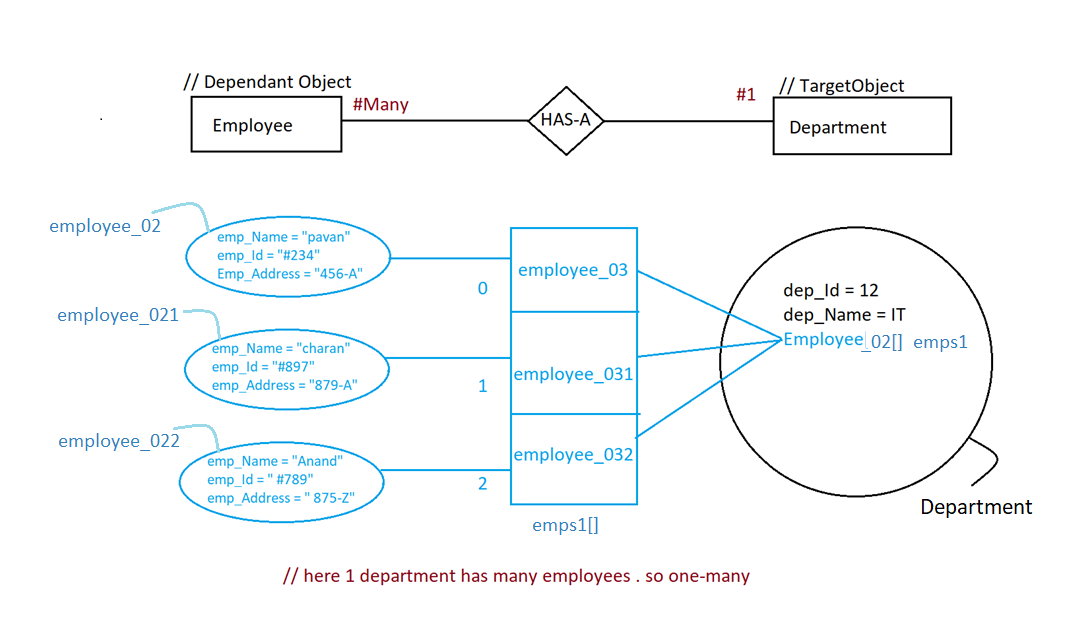
// memory map is same for above and below example .

Eg: Account011 , Employee011 , One\_One\_Association\_Using\_Setters\_Getters

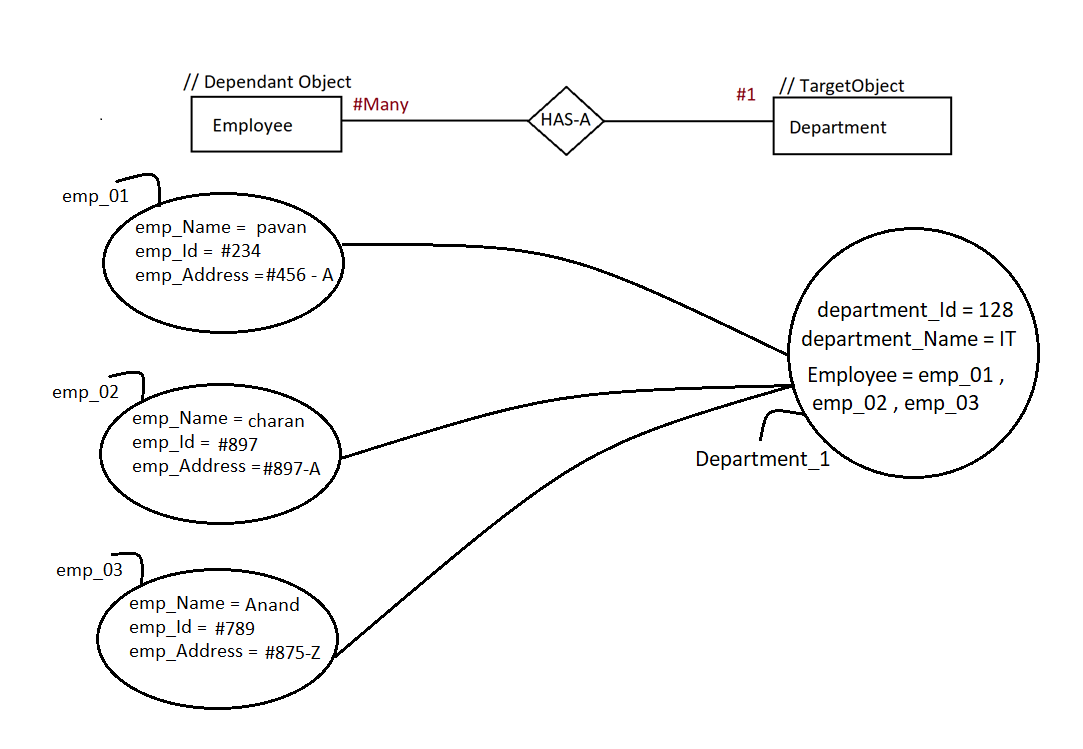
( the memory map is same as on One\_One\_Association\_Using\_Constructor)

One-One Association : It is a relation between two entities ( classes ) ,where one instance of an entity should be mapped with exactly one instance of another entity .

Eg: Employee\_02 , Department , One\_Many\_Association\_Using\_Constructor



Eg: One-Many\_Association\_Using\_Setter\_Getter , Employee\_03 , Department\_1



One-Many-Association : It is a relationship between entities , where one instance of any entity should be mapped with multiple instances of another entity.

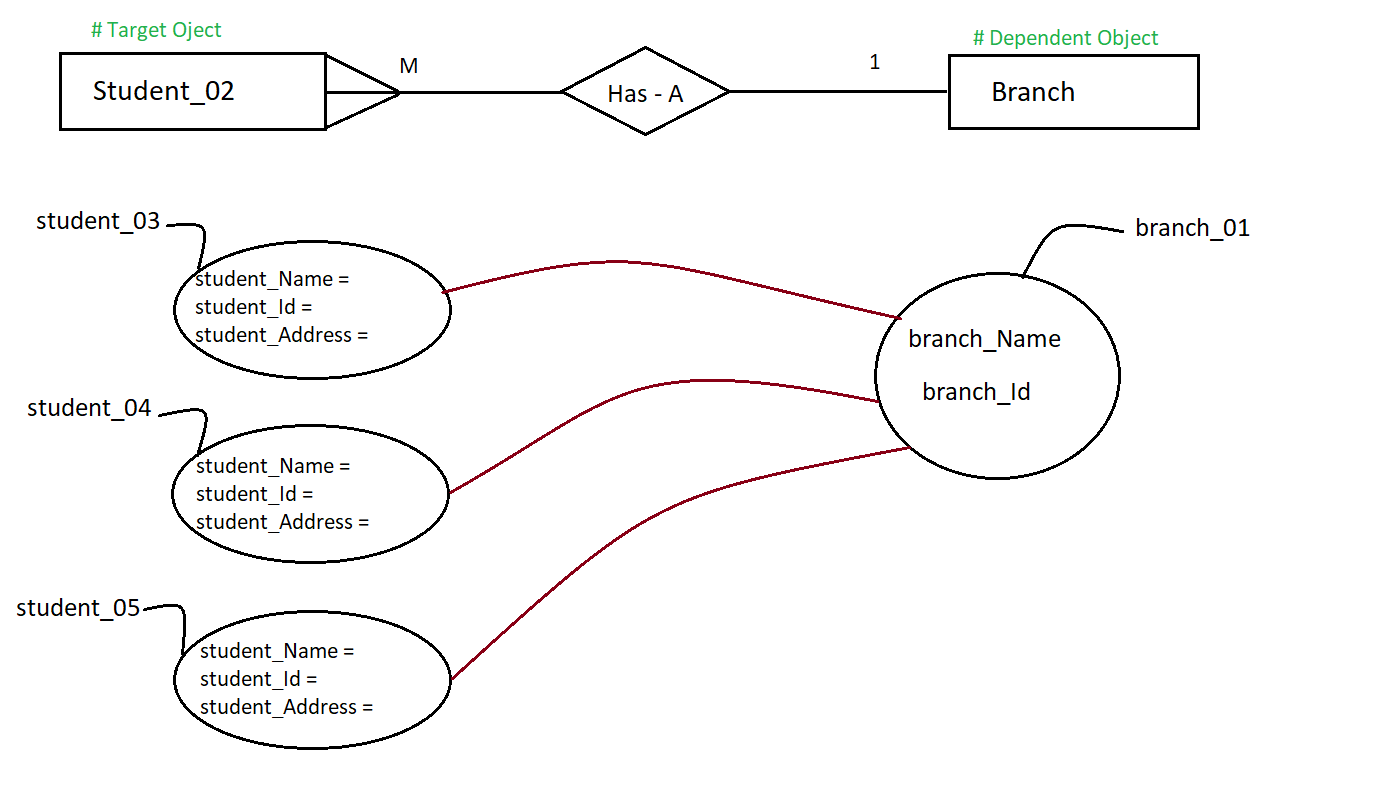
Many-One-Association

It is a relationship between entities , where multiple instances of an entity should be mapped with exactly with one instance another entity

Example : Multiple students joined in single branch.

Eg: Many\_One\_Using\_Setters\_Getters , Branch , Student\_01

Eg: Many\_One\_Using\_Constructor , Branch\_01 , Student\_02



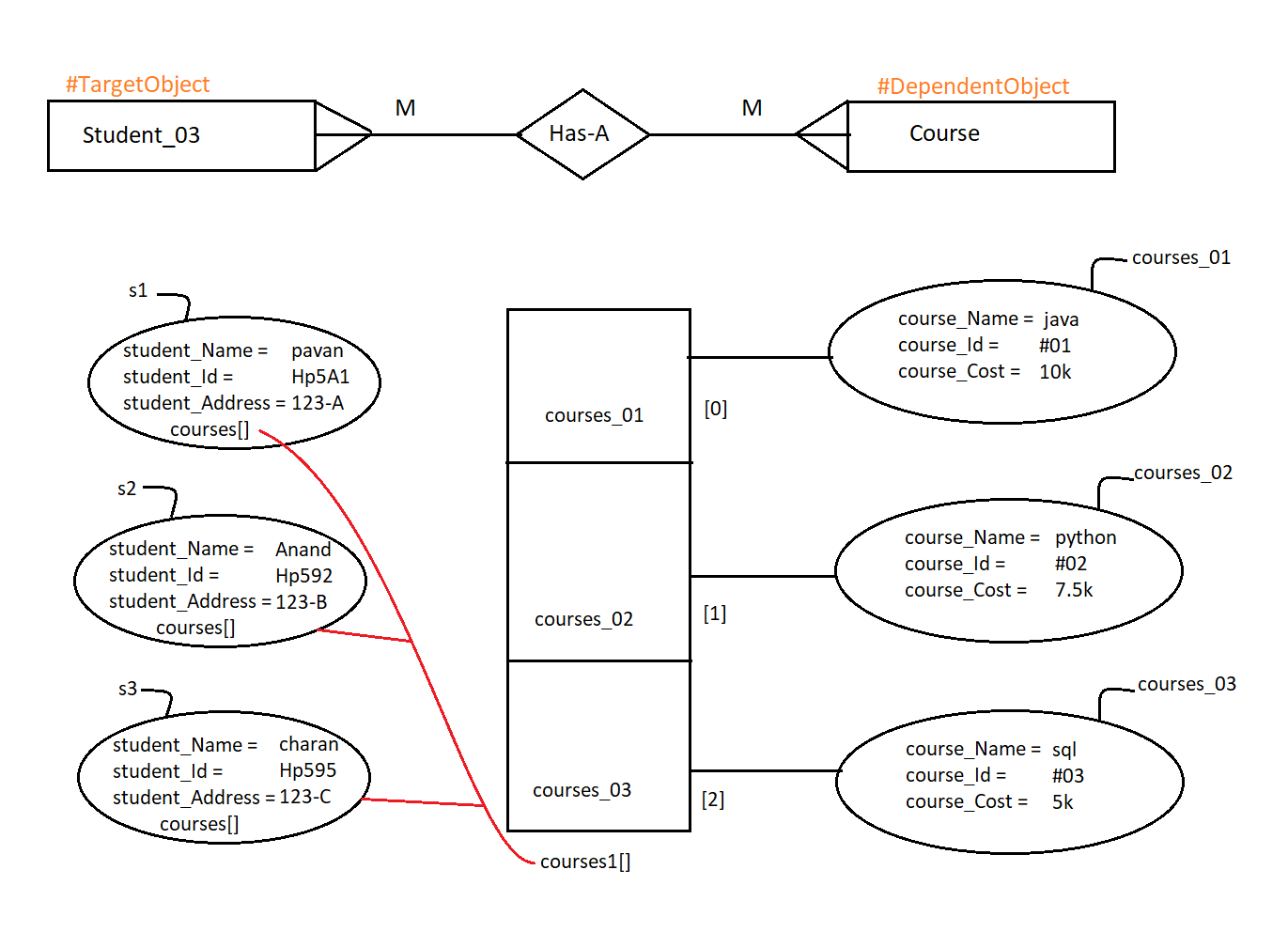
Many to Many Association :

It is a relationship between entities , where multiple instances of one entity should be matched with multiple instances of another entity.

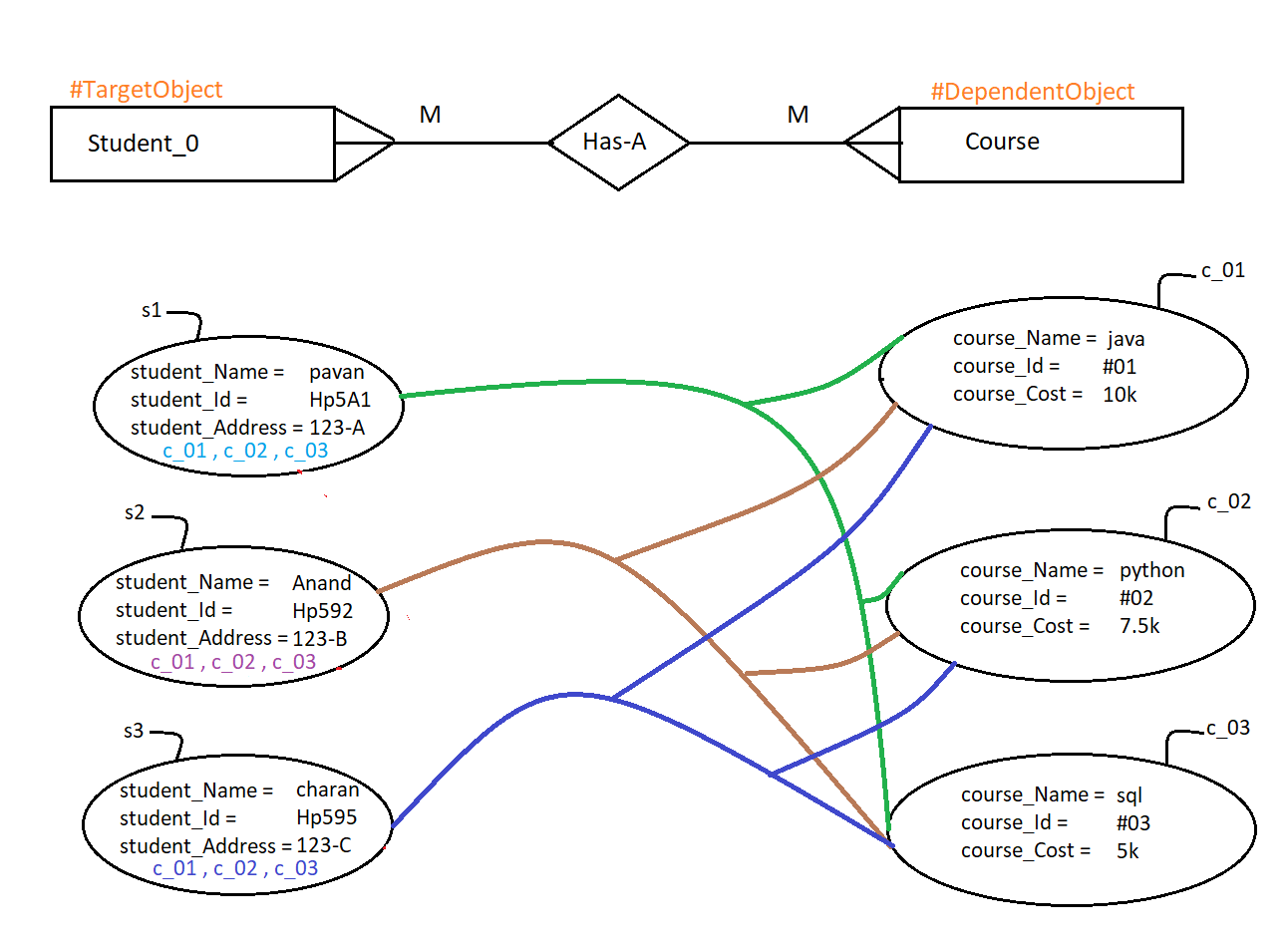
Example : multiple students have joined with multiple courses

Note: which ever given to someone is dependent , and which ever given to end user is target

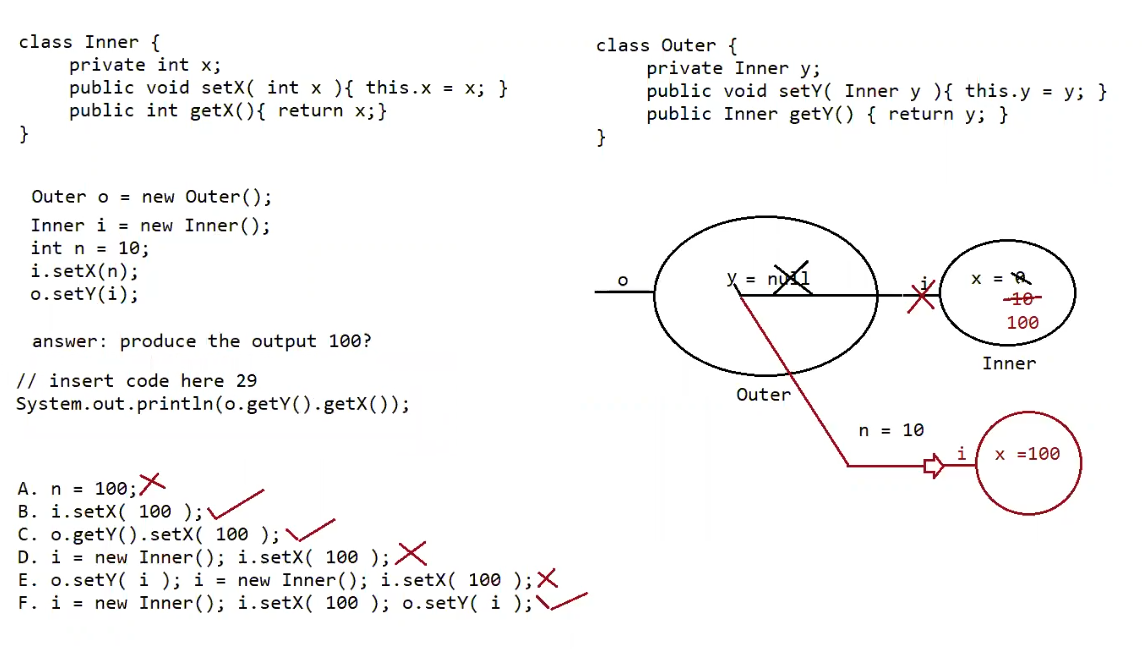
Eg: Many\_Many\_Using\_Constructor , Course , Student\_03



Eg: Many\_Many\_Using\_Setter\_Getter , Course\_01 , Student\_04



Eg: Has-A-Relation\_Eg1



Program flow :

First outer object is created , and it consists of Inner reference variable y , since y is not assigned default value for reference null is given to it. Outer object address is given to reference variable o.

Inner object is created memory for instance variable x is given and default value 0 is assigned to it. Inner object address is stored in the reference variable i.

After that local variable n with value 10 is declared . and n is passed as an argument setX() of i . so now x value is asasigned to 10.

And address of inner object is given to inner type reference variable y .

Option checking :

b) i.setX(100)

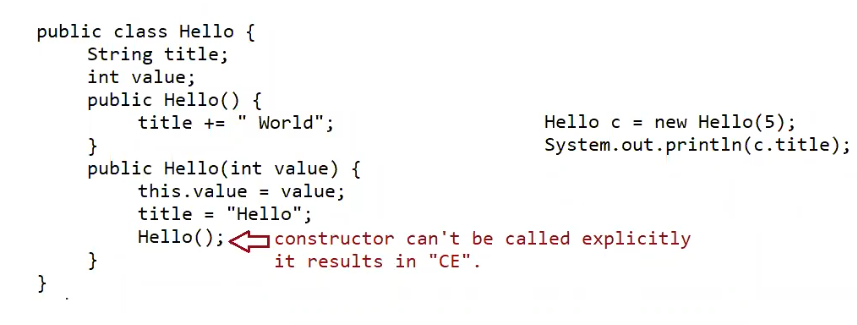
//if 100 is assigned to x . then x value is updated to 100 . and reference i is previously assigned to y.

now in this line o.get(Y).get(X) , we are calling method get(Y) which returns the address of inner object i . and with that reference we are calling method get(X) which gives 100.

// work similarly for c and f options.

Option d is not possible because since a new object is created and address of inner object is changed , which is not updated in y.

In option f even though a new object is created , x value is set to 100 , and address of new object is updated to y.



Types of Coupling

There are two types of coupling:

1. Tight coupling:

In general, Tight coupling means the two classes often change together. In other words, if A knows more than it should about the way in which B was implemented, then A and B are tightly coupled.

Example: If you want to change the skin, you would also have to change the design of your body as well because the two are joined together – they are tightly coupled. The best example of tight coupling is RMI(Remote Method Invocation).

Eg: Subject

// go through the code

Eg: Volume

// go through the code

### 2. Loose coupling

In simple words, loose coupling means they are mostly independent. If the only knowledge that class A has about class B, is what class B has exposed through its interface, then class A and class B are said to be loosely coupled. In order to over come from the problems of 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. Example : If you change your shirt, then you are not forced to change your body – when you can do that, then you have loose coupling. When you can’t do that, then you have tight coupling. The examples of Loose coupling are Interface, JMS.

Eg: Subject\_1

// go through the code

Tightly coupled , aggregation

Loosely coupled composition

Inner classes tightly coupled