

We are the developer we need to write the business logic for SavingAcount and CurrentAccount.

The task of Dynamic Polymorphism is invoking specific function or method and understanding the request what the client is making for then creating the object of that type then passing it to client

That all handled by the IOC Container.

But how?

Spring framework read the configuration about the classes that how we are going to instantiated using the configuration file.

Spring framework to all the rest of the task we don’t have to think about.

**Inversion of Control(IoC)is an object-oriented programming practice whereby**

**the object coupling is bound at run time by an "assembler"**

**object and are typical not knowable at compile time using the static analysis**

-----------------------------------------------------------------------------

**package** com.lara;

**public** **class** **Account**

{

**public** **void** withdrow()

{

System.**out**.println("----Account class withdrow()------");

}

}

**public** **class** **CurrentAccount extends** **Account**

{

@Override

**public** **void** withdrow()

{

System.**out**.println("----CurrentAccount class withdrow()------");

}

**public** **void** overDraft()

{

System.**out**.println(“overDraft CurrentAccount section---- ----");

}

}

**public** **class** **SavingAccount extends** **Account**

{

@Override

**public** **void** withdrow()

{

System.**out**.println("----SavingAccount class withdrow()------");

}

}

**public** **class** **Decide**

{

**public** **void** withdrowMyAccount(Account acc)

{

**//case : 1**

**acc.withdrow();**

**//acc.overDraft();** not present

**/\*the method which is present in the Top class that method is only accessible here**

**i.e. withdrow()**

**and if the method present in the Top class(Account) and these method are overriden in the child class and we are passing the instance of child class then the overridden method of that child class will be executed\*/**

**//case : 2**

**/\*CurrentAccount c1 = (CurrentAccount)acc;**

**c1.withdrow();**

**c1.overDraft();\*/**

**//case : 3**

overcome the problem of ClassCastException

**if**(acc **instanceof** CurrentAccount)

{

CurrentAccount c = (CurrentAccount)acc;

c.withdrow();

c.overDraft();

}

**else**

{

SavingAccount s = (SavingAccount)acc;

s.withdrow();

}

}

}

**/\***

**Here acc is Account Type but we do not know which account it may be SA and CA**

**based on Account Type it will call the corresponding method.**

**\*/**

**public** **class** **ImpAccount01**

{

**public** **static** **void** main(String[] args)

{

Account acc = **new** CurrentAccount();

acc.withdrow();

**//case : 1**

/\*

**Here instance of CA is existing in acc so that it will withdrow() of CA.**

**Here we are creating the generic variable of Account type and storing the reference of any child class through that we are accessing the child class method.**

**Account is the top level class**

**why we go for this type we can also create the instance of CA and called the withdrow()**

**Reasons:-Here we are achieving the polymorphism (having many form)**

**acc is the variable of Account or types of representation of Account.**

**and Account is representing many types of Account like SavingAccount, CurrentAccount, FixedDeposit and many other accounts.**

**Here we are achieving static polymorphism or compile time polymorphism.**

**At compile time we decide the program flow.**

**Here at CT we decide the acc is containing the reference of CA and it will call the withdrow() of CA.**

**THIS IS THE STATIC POLYMORPHISM (VVI)**

**How to achieve the dynamic polymorphism?**

**---------------------------------------**

**We want to at Runtime what the acc of Account offer.**

**Based on the user choice it will invoke or choose the CA, SA, withdrow()**

**\*/**

}

}

**public** **class** **ImpAccount02**

{

**public** **static** **void** main(String[] args)

{

Decide d1 = **new** Decide();

d1.withdrowMyAccount(**new** CurrentAccount());

d1.withdrowMyAccount(**new** SavingAccount());

**/\***

**Here we are free to pass the Account type it may be CA or SA based on type of Account**

**it will call the withdrow() method.**

**In Decide class we are calling withdrowMyAccount(---) method and inside it, it decide which Account type withdrow() should be called it may be CA or SA**

**\*/**

}

}

**public** **class** **ImpAccount03**

{

**public** **static** **void** main(String[] args)

{

SavingAccount s = **new** SavingAccount();

s.withdrow();

CurrentAccount c = **new** CurrentAccount();

c.withdrow();

**/\***

**we can call the SA and CA withdrow() by creating the instance of corresponding class**

**If we go for above mention code then it is highly coupled of very tight coupled application.**

**CA and SA is highly coupled with this implementation class i.e. ImpAccount03**

**\*/**

}

}

**public** **class** **ImpAccount04**

{

**public** **static** **void** main(String[] args)

{

Decide d = **new** Decide();

**//d.withdrowMyAccount(new CurrentAccount());**

**//good here because we are passing the reference of CA to the withdrowMyAccount(--)**

**// so that Account acc is properly downcasted in CA and acc call the overDraft() method**

**//d.withdrowMyAccount(new SavingAccount());**

**d.withdrowMyAccount(new CurrentAccount());**

**//but if we pass here SA reference then we will got ClassCastExeption**

**//in the Decide class we are forcefully downcasted SA reference to the CA**

**//no complie time error because the compiler does not know type of Account**

**//we can overcome this problem by checking the instance type**

}

}

**/\***

**Inversion of Control(IoC)is an object-oriented programming practice whereby**

**the object coupling is bound at run time by an "assembler"**

**object and are typical not knowable at compile time using the static analysis**

**\*/**