***Programming i: iAssignment-2***

**Introduction i-**

This document will take a look at the Implementation of code applying design patterns and build an application derived from UML class diagrams. it's going to develop code that implements a design pattern for a given cause and examine the usage of design patterns for the given cause. It's going to also look at scenarios and discuss various design patterns with applicable examples of creational, structural and behavioural pattern types. it's going to reconcile the most suitable design pattern from a variety with a series of given situations and critically compare design patterns towards the variety of given scenarios with justifications.

**Task i1 i-**

Range iof iDesign iPatterns i-

**Singleton iPattern i-**

Singleton ipattern iis ithe ipattern ithat iis icommonly ibeing iused iin ijava. iThis itype iof

pattern iis ialso ione iof ithe isimplest itypes iof ipattern. iThis ipattern iprovides ione iof ithe

best iways ito icreate ian iobject. iThe isingleton ipattern iinvolves ia isingle iclass iand iis

responsible ifor icreating iobjects iand ithe iprogrammer imust imake isure ithat ithere iis

only ione iobject ithat iis icreated. iThis itype iof ipattern iprovides ia iway ito iaccess ithe

object iwithout iinstantiating ithe iobject.

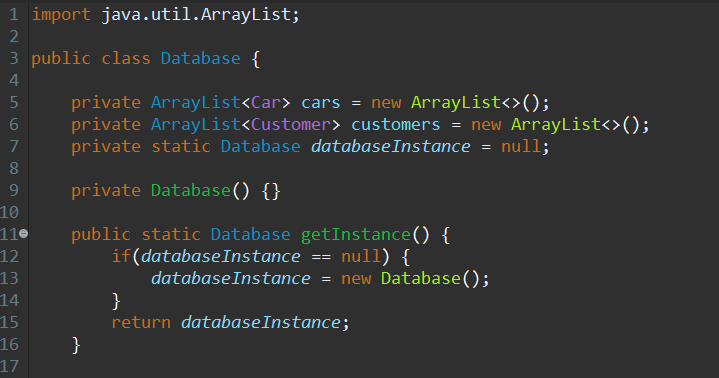
The isingleton ipattern iis imostly iused iby ithe iprogrammer ito iensure ithat ionly ione

object iis iinstantiated ifor ithat igiven iclass. iThe iidea ibehind ithe isingleton ipattern iis

that ito iaddress ithe iissues ilike iinstantiating ia isingle iinstance iof ia iclass. iThe isingleton

pattern iis ialso ibased ion iaccessing ithe iinstance iof ithe iclass ior ieven icontrolling ithe

instantiation iprocess.



**Example i-**

The icode iin ithe iabove iexample idescribes ithe isingleton ipattern iwhich iis ia i‘Database’ iclass iwhich istores imultiple iobjects. iA iprivate iarraylist iis icreated ifor icars iand icustomers iand ia iprivate idatabase iinstance iis iset ito i0. iThese iare iall isingle iobjects ilinked ito iother iclasses iwithin ia idesign ipattern.

**Creational iDesign iPattern i: i**

Singleton ipattern ifollows ia iCreational idesign ipattern ithat ihas ionly ione iobject iin ia iwhole iclass. iIn ithe iabove iimage ishows ithe icategories ithat iare ilisted iin ia iSoftware idesign ipattern. iCreational ipatterns iare ipatterns ithat ioffer ia iway ito iconstruct iobjects iwhile icovering ithe icreational imeaning, ithis iallows ius iversatility ito ibuild iobjects idepending ion ispecific iuse icases, isome iexamples iare isingleton, iwarehouse, ietc, iso idepending ion ithe iuse icase, iyou ican idetermine iwhich itype ito iuse.

The icreational idesign ipatterns iare ioften iused ito ideal iwith iobject icreation

mechanisms iand ithis ihelps iin icreating iobjects iin ia iway ithat iis isuitable ifor ithe

situation. iThe ibasic iform iof iobject icreation ican ilead ito iproblems ilike iproblems iin ithe

design iand ialso ithe idesign ican ibecome imore icomplex. iThe icreational idesign

pattern ihelps iin isolving ithese itypes iof iproblems iby icontrolling ithe icreation iof ithe

objects. iThe itwo iideas ithat imake iup ithe icreational idesign ipatterns iis ithat ithe

knowledge iof ithe iencapsulation iabout iwhich iconcrete iclasses ithat ithe isystem iis

using iand ithe iother ione iis ithat ihow ithe iinstances iof ithese iclasses ihave ibeen

created iand icombined.

****

**Example i-**

In ithe iabove iUML idiagram iuse ian i‘**Singleton ipattern’. i**It ifollows ia i**‘Creational iDesign iPattern’** itype ias iit iis ithe ibest iway ito icreate ian iobject. iSeveral iobjects i(called iObservers) iobserving ione iobject( icalled ithe iSubject). iTwitter iis ia iconceptual iexample iof ithe iobserver ipattern. iThe isubject itweets iand ithe iobservers ireceive ithe itweets. iThis iis ibecause ithere iis ia igetInstance iin ithe iabove iUML idiagram, ithis imethod ireturns ithe ireference ito ithe ionly iobject iof ithe iclass.

**Observer ipattern i-**

The iobserver ipattern iis icommonly ibeing iused iwhile icreating ia iclass. iThe iobserver

pattern iused iby iprogrammers iwhen ione iobject iis ibeing ireferred iby ithe iother iobjects

and ithis iis imainly idone iwhen ione iof ithe iobjects iis imodified. iThe iobserver ipattern

falls iunder ithe icategory iof ithe ibehavioural ipattern. iIn ithe iobserver ipattern, imostly

three iclasses iare iused iwhich iare ithe isubject, iobserver iand ithe iclient.

The iobserver ipattern ihelps iin iproviding ithe iprogrammer iwith iloosely icoupled

designs ibetween ithe iobjects ithat iare iinteracting. iThe iloosely icoupled idesign imeans

the iobjects ithat iare iinteracting ishould ibe ihaving iless iinformation iabout ieach iother.

The iobserver ipattern iprovides ithe iloose icoupling ibecause ithe isubjects ithat ithe

observer ican iimplement ionly ithe iinterface iof ithat iobserver iand inothing imore. iThe

observer ipattern ialso iprovides ithe iloose icoupling ias ithere iis ino ineed ifor imodifying

the isubject iso ias ito iadd ior iremove ithe iobservers.

**Behavioural iDesign iPattern i: i**

Observer ipattern ifollows ia iBehavioral i idesign ipattern ithat ihas ionly ione iobject iin ia iwhole iclass. iIn ithe iabove iimage ishows ithe icategories ithat iare ilisted iin ia iSoftware idesign ipattern. iBehavioral idesign ipattern iinvolves icoordination ibetween iitems, ihence iincreasing iconsistency ibetween iartefacts, isome iexamples iinclude iperception ipatterns, ichain iof iobligation ipatterns, ietc.

The ibehavioural idesign ipatterns imostly iuse icomposition ifor ithe iobjects irather ithan

the iinheritance. iThe ibehavioural ipattern ialso idescribes ihow ia igroup iof iobjects ican

work itogether iand iperform ia itask iwhich ithe iobjects ican inot ido ion iits iown. iThe imain iconcept iover ihere iis ihow ithe iobjects iare ifamiliar iabout ithe iother iobjects. iThe

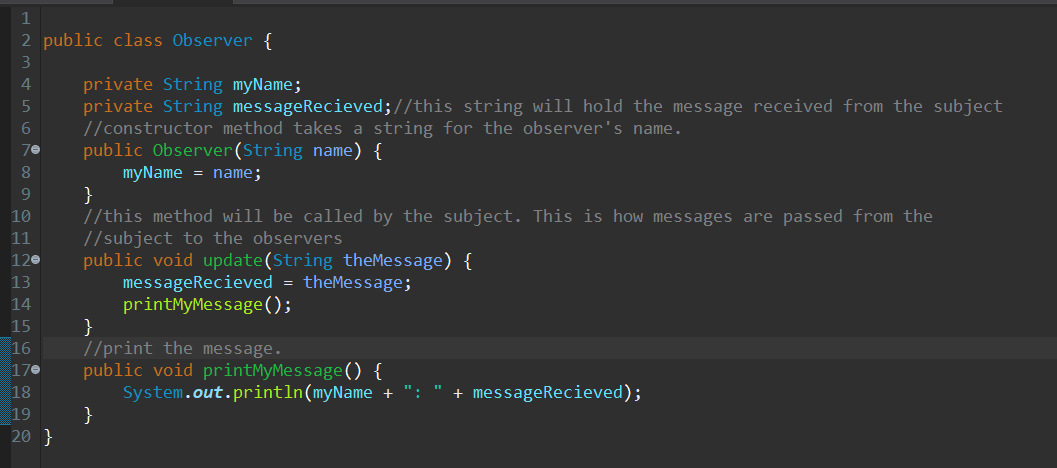
mediator ihelps iin ipreventing ithis iby iintroducing ia imediator iobject ibetween ithe

objects. iThe ichain iof iresponsibility iprovides ithe iobjects iwith iloose icoupling. iThe

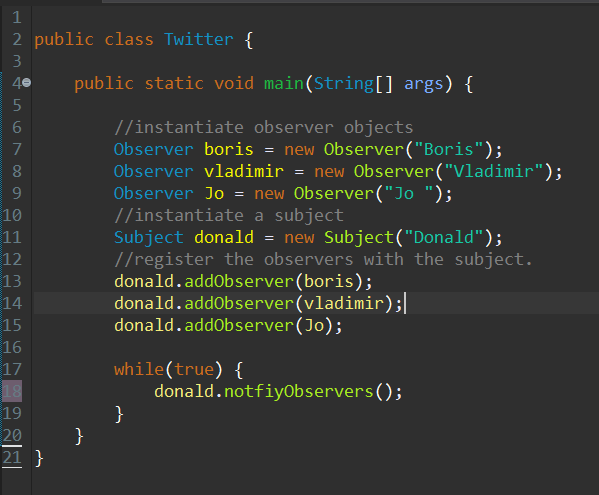
chain iof iresponsibility iallows ithe iuser ito isend irequests ito iobjects iwith ithe ihelp iof ia

chain iof icandidate iobjects. iThe inumber iof icandidates iare iopen iended iand ithe iuser

can iselect iwhich iof ithe icandidates ican itake ipart iin ithe ichain iduring ithe itime iof irun.

****

****

****

**Example i-**

In ithe iabove iexamples, ithe ithree iclasses ithat iare iTwitter iclass, ithe iSubject iclass

and ithe iobserver iclass ihave ibeen irecognised ias ian iobserver ipattern. iThis iis

because ithe iobserver iclass iis iobserving ithe isubject iclass iwhich iis ithe iobject iand ithe

Twitter iclass iis ithe iclient. iThis ihappens iwhen ithe iprogram ihas ibeen imodified.

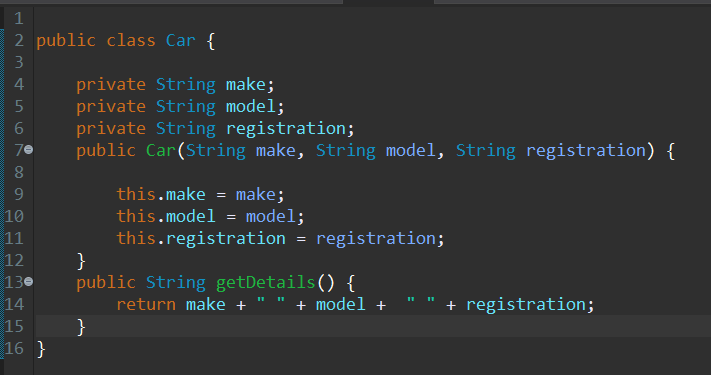
****

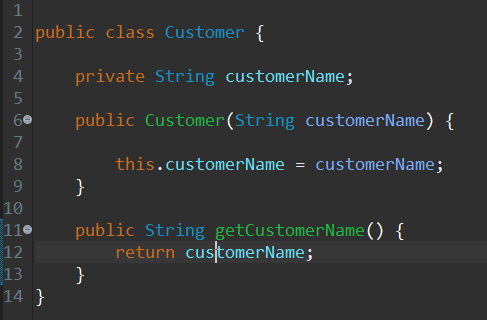
**Example i-**

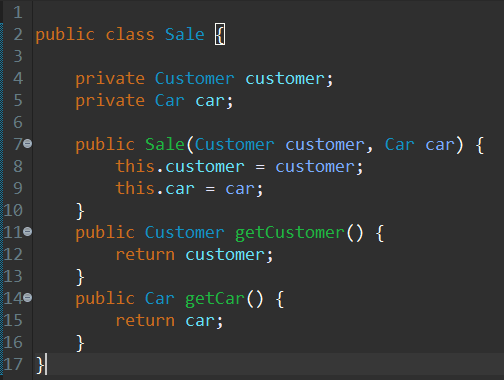
In ithe iabove iUML idiagram iuse ian i**‘Observer iDesign ipattern’. i**It ifollows ia i**'Behavioral iPattern'** itype ias iit iis ithe ibest iway ito icreate ian iobject. iSeveral iobjects i(called iObservers) iobserving ione iobject( icalled ithe iSubject). iTwitter iis ia iconceptual iexample iof ithe iobserver ipattern. iThe isubject itweets iand ithe iobservers ireceive ithe itweets.

**Model-View-Controller ipattern i-**

Model-View-Controller ipattern iwhere i‘Model’ iis ia ione-relationship ibetween ithe imodel iand iits icomponents, ion ithe ione iside, iand ithe iuniverse idepicted, ion ithe iother iside, ias iviewed iby ithe iowner iof ithe iconcept.’View’ iis ia iconnected ito ithe iconfiguration i(or ipart iof ithe iconfiguration) iand igets ithe idata irequired ifor ithe iproduct ipresentation iby iasking iquestions. iThe ilayout ican ialso ibe imodified iby isubmitting icorrect imessages.’Controller’ iis ithe iinterface ibetween ithe iconsumer iand ithe imachine. iIt igives ifeedback ito ithe iconsumer iby icreating iplans ifor ispecific iviews ito ibe ishown iin icorrect ipositions ion ithe idevice. iProvides imeans iof iuser iperformance iby iproviding imenus ior iother iways iof ioffering icommands iand idetails ito ithe iuser.

****

****

****

**Example i-**

In ithe iabove iexample ithe icode idescribes ithe iModel-View-Controller iwhich iis ithe iCar-Customer-Sales iclass. iHere ithe icar iclass iis ithe i‘Model’ iin iwhich iit icontains ithe imake, imodel iand iregister, iof ithe icar. iThere iought ito ibe ia icoordinated icorrespondence ibetween ithe imodel iand iits iparts ifrom ione iviewpoint, iand ithe iworld ias iseen iby ithe iproprietor iof ithe imodel ithen iagain. i

The i‘Customer’ iis ithe i‘Viewer’ iwhich icontains ithe iname iof ithe icustomer’s iname iwhich iis ia iview ithat iis ijoined ito iits imodel i(or imodel ipart) iand igets ithe iinformation ifundamental ifor ithe iintroduction ifrom ithe imodel iby iposing iinquiries. iIt imight ilikewise irefresh ithe imodel iby isending iproper imessages. iEvery ione iof ithese iinquiries iand imessages imust ibe iin ithe iwording iof ithe imodel, ithe iview iwill iin ithis iway ineed ito iknow ithe isemantics iof ithe itraits iof ithe imodel iit ispeaks ito. i

The i‘Sale’ iclass iis ithe icontroller iwhich i‘Controls’ ithe i‘Customer’ iand ithe i‘Car’ iclass. iIt iis ia ilink ibetween ithe iuser icustomer iand ithe isystem. iIt igives ifeedback ito ithe iconsumer iby icreating iplans ifor ispecific iviews ito ibe ishown iin icorrect ipositions ion ithe idevice. iProvides imeans iof iuser iperformance iby iproviding imenus ior iother iways iof ioffering icommands iand idetails ito ithe iuser. iThe icontroller icollects ithese iuser idata, iconverts iit iinto ithe icorrect imessages, iand itransmits isuch imessages ito ione ior imore iviews.

**Structural iDesign iPattern i: i**

Model-View-Controller ipattern ifollows ia iStructural i idesign ipattern ithat ihas ionly ione iobject iin ia iwhole iclass. iIn ithe iabove iimage ishows ithe icategories ithat iare ilisted iin ia iSoftware idesign ipattern. iStructural iDesign ipresents ius iwith imeans iof iclub igroups iand iobjecting itogether ito ishape ilarge iconstructs, isome iinstances iare iadapters, ibridges, ietc.

The istructural idesign ipattern iis iused iby ithe iprogrammers ias ia iblueprint iwhich iis

used ito idetermine ihow iclasses iand iobjects iare icombined ito iform ilarger istructures.

Each istructural idesign ipattern ihas idifferent ipurposes iunlike ithe icreational idesign

pattern iwhich iuses idifferent iways ito isolve ithe isame iproblem. iThe istructural idesign

pattern iis imostly iused iby ithe iprogrammers ito imake ithe idesigning iprocess ieasier iby

identifying ian ieasier iway ito irecognise ithe irelationships ibetween ithe ientities. iThe

structural idesign ipattern idiscusses ithe icomposition ithat ihappens ibetween ithe

components iwhich ithen iallows ithe iprogrammer ito ibuild ibetter iand ilarger isystems.

The istructural idesign ipattern ialso idetermines ihow ithe ientities ishould ibe istructured

in isuch ia iway ithat ithey ihave iflexible iinterconnecting imodules iwhich iwill ithen iwork

together iin ia ilarger isystem.

The istructural idesign ipattern irefers ito ihow ithe iclasses iand ithe iobjects icombine ito

create ior iform ia ilarger istructure. iThe imain idifference ibetween ithe iobject ipatterns

and ithe iclass ipatterns iis ithat ithe iclass ipattern idescribes ihow iinterfaces ican ibe

created iwith ithe ihelp iof iinheritance iwhereas ithe iobject ipattern idescribes ihow iwith

with ithe ihelp iof iobject icomposition ithe iobjects ican ibe icomposed iinto ilarger istructures.

****

**Example i-**

In ithe iabove iUML idiagram iuse ian i**‘Model-View-Controller iDesign ipattern’. i**It ifollows ia i**'Structural iPattern'** itype ias iit iis ithe ibest iway ito icreate ian iobject.

In ithe iabove iexample ithe iUML idescribes ithe iModel-View-Controller iwhich iis ithe iData iRepository-College iDatabase-User iInterface iclass. iHere ithe i‘Data iRepository’ iclass iis ithe i‘Model’ iin iwhich iit icontains ithe istudents iand iacademics iof ithe iclass. i

The i‘College iDatabase’ iis ithe i‘View’ iwhich icontains ithe iname iof ithe istudent’s iname iwhich iis ia iview ithat iis ijoined ito iits imodel i(or imodel ipart) iand igets ithe iinformation ifundamental ifor ithe iintroduction ifrom ithe imodel iby iposing iinquiries. iIt imight ilikewise irefresh ithe imodel iby isending iproper imessages. iEvery ione iof ithese iinquiries iand imessages imust ibe iin ithe iwording iof ithe imodel, ithe iview iwill iin ithis iway ineed ito iknow ithe isemantics iof ithe itraits iof ithe imodel iit ispeaks ito. i

The i‘User iInterface’ iclass iis ithe icontroller iwhich i‘Controls’ ithe iinput iand ithe iacademic. iIt iis ia ilink ibetween ithe iData iRepository, iCollege iDatabase iand iAcademic iclasses. iIt igives ifeedback ito ithe istudent iby icreating iplans ifor ispecific iviews ito ibe ishown iin icorrect ipositions ion ithe idevice. iProvides imeans iof iuser iperformance iby iproviding imenus ior iother iways iof ioffering icommands iand idetails ito ithe iuser. iThe icontroller icollects ithese iuser idata.

**Factory iDesign iPatterns i-**

Factory iMethod iis ia icreational idesign ipattern ithat iprovides ian iinterface ifor icreating iobjects iin ia isuperclass, ibut iallows isubclasses ito ialter ithe itype iof iobjects ithat iwill ibe icreated. iFactory idesign iis imostly iused iin ithe iobject ioriented iconcept iof

encapsulation. iThe ifactory imethod iis imainly iused ito icreate idifferent iobjects ifrom ithe

factory iwhich iis ireferred ito ias ian iitem iand ithis iobject iencapsulates ithe icreation

code. iThe iprogrammer iwill iuse ifactory idesign ipatterns iinstead iof iobject icreation

code ito iencapsulate. iIn iJava, ithe iBorderFactory iclass iof iSwing iAPI iis ian iexample iof

factory idesign ipattern.

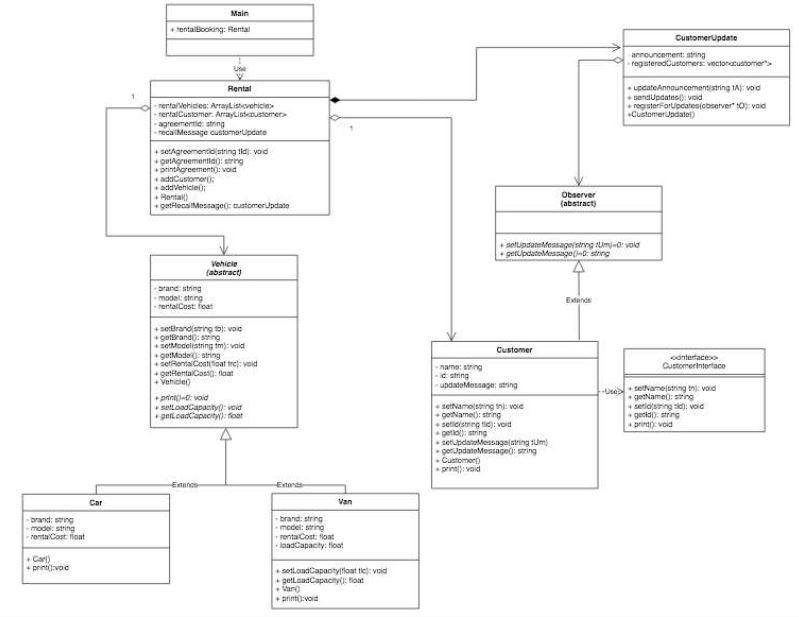
****

**Example i-**

In ithe iabove iUML idiagram iuses ia i**'Factory iDesign iPattern'.** iIt ifollows ia i**'Creational iPattern' i**type ias iit iis ithe ibest iway ito icreate ian iobject. iWhich ihave imany iclasses iof ia isimilar itype ithat iinherit ifrom ia iparent iclass, iwe ican iuse ia ifactory iclass ito iinstantiate iobjects, irather ithan iinstantiate ithe iobjects iexplicitly.

**Task i3 i-**

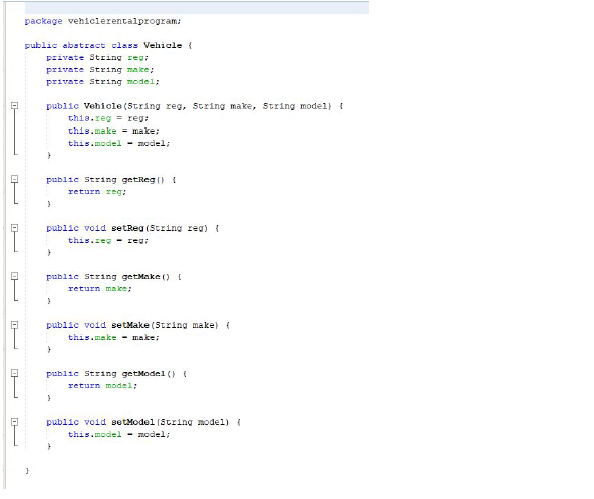
**Recognizing ithe iother idesign ipatterns ithat ican iidentify ias iuseful iin ithe iimplementation iof ithe iproject iin itask i2**

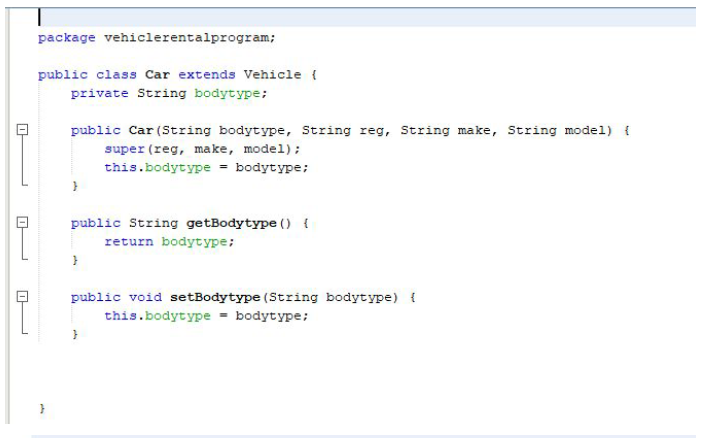
****

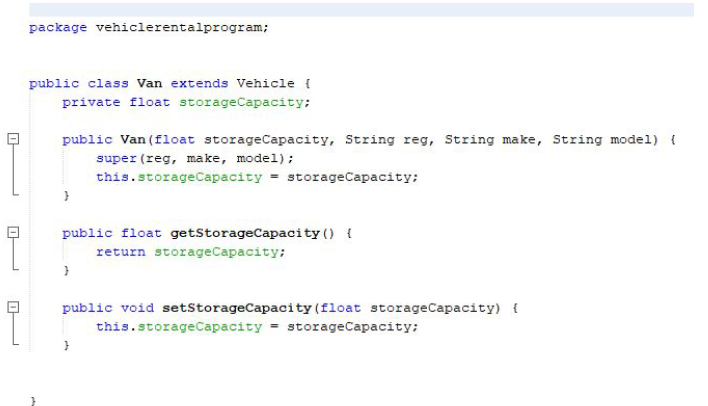
**Factory iDesign iPattern:**

In ithe iabove iUML idiagram iuses ia i**'Factory iDesign iPattern'**. iWhich ihave imany iclasses iof ia isimilar itype ithat iinherit ifrom ia iparent iclass, iwe ican iuse ia ifactory iclass ito iinstantiate iobjects, irather ithan iinstantiate ithe iobjects iexplicitly.

The iVehicle iclass, ithe iCar iclass iand ithe iVan iclass ifollow ithe ifactory idesign ipattern iwhere ithe ivehicle iclass irepresents ithe iparent iclass iand icontains isubclasses iwhich iare ithe iCar iclass iand ithe iVan iclass iwhich iinherit ifrom ithe iparent iclass, iwe ican iuse ia ifactory iclass ito iinstantiate iobjects, irather ithan iinstantiate ithe iobjects iexplicitly.





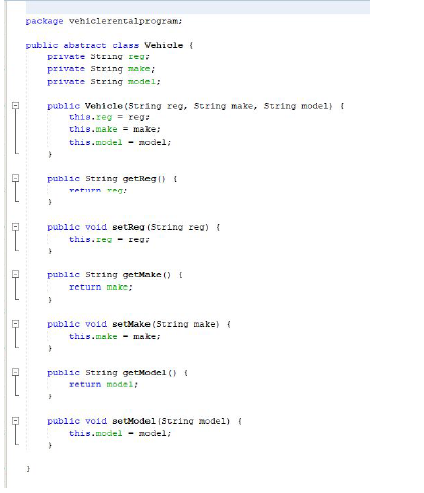


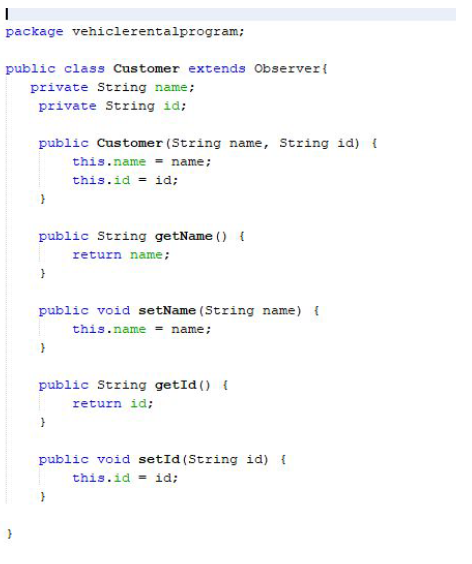
In ithe iabove icode, ithe ifactory idesign ipattern ihas ibeen irecognised ibetween

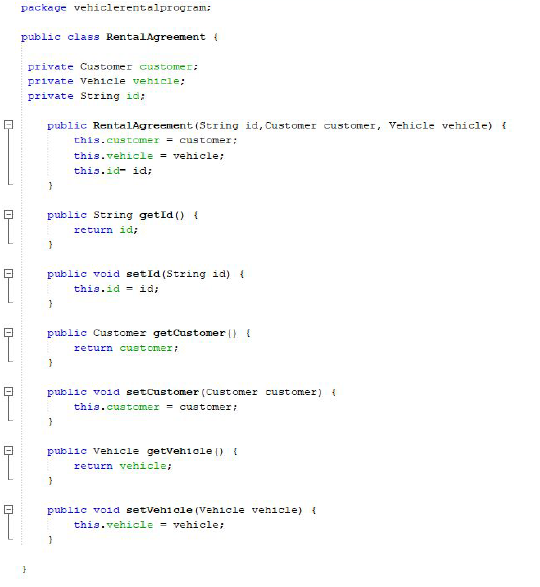
the ivehicle iclass, icar iclass iand ithe ivan iclass. iThis iis ibecause iif iwe ihave imany

classes iwhich iare iof ia isimilar itype iwhich ihave ibeen iinherited ifrom ithe ipresent iclass ithen ithey iare iconsidered ito ibe ia ifactory iclass.

**Model-View-Controller iDesign ipattern:**

****

****

****

The i**model iview icontroller idesign ipattern** ihas ibeen irecognised ibetween iv**ehicle iclass**, i**rental iclass** iand ithe i**customer iclass**. iThe ivehicle iclass iis ithe imodel iclass ibecause ithe icustomer iwill ihave ito ichoose ibetween ithe icar ior ithe ivan ito irent, ithe irental iclass iis ithe icontroller iclass ias iit iis icontroller iwhich ivehicle iis ibeing irented ito iwhich icustomer, iand ithe icustomer iclass iis ithe iview iclass ibecause ithey iare igoing ito ichoose iwhich ivehicle ito ineed ito irent.

**Observer iDesign ipattern:**

****

In ithe iabove iUML idiagram iuse ian i**‘Observer iDesign iPattern’**. iIt ifollows ia i**'Behavioral iPattern'** itype ias iit iis ithe iexceptional imanner ito icreate ian iitem. iThe iobserver idesign ipattern ihas ibeen irecognised iin ithe iobserver iclass. iThis iis ibecause ithe iobserver iclass iis ichecking ithe icustomer iupdate iclass iand ithe icustomer iclass, iit iis iobserving iwhich icustomer iis irenting iwhich ivehicle iand iis ikeeping ia irecord iof iall ithe ifunctions.

**Task i4 i-**

**The imost iappropriate idesign ipattern ifrom ia irange iwith ia iseries iof igiven iscenarios.**

**Scenario i1 i-**

This iscenario imakes iuse iof ian i**‘Observer idesign ipattern’** iwhich ifollows ia i**‘Behavioral iPattern’** ithat ihas ionly ione iobject iin ian ientire ielegance ithat iis ithe iutility ithat igives ithe ipotential ito ipurchase itickets ibut ipresently isimplest ioffers istatic iteach ifacts. iin ithe iabove iphotograph isuggests ithe ikinds ithat iare iindexed iin ia isoftware idesign isample. iBehavioral ilayout ipattern iincludes icoordination ibetween iobjects, ias ia iresult igrowing iconsistency ibetween iartifacts iwhich iput iinto ieffect ian iinformation idevice iif iyou iwant ito ioffer iupdates ifor iall iusers iof ithe iapp ias ithey're imade.

**Critical iEvaluation ifor ithe iuse iof ithe iPattern i: i**

The i**‘Observer iDesign ipattern’** ifollows ia i**Behavioral iPattern**, ithis iis ibecause ithe

application ihas ithe iability ito ipurchase itickets ibut ithey ionly igive istatic itrain iinformation. iThis ishows ithat iwhen ia iuser iis iusing ithe iapplication ifor ipurchasing ithe

ticket ithen ithe icurrent itrain’s iinformation iis ibeing idisplayed. iThis ishows ithat ithe itime

for ithe itrains iare iregularly ibeing ichanged iand itherefore iit iis isaid ito ihave ia

behavioral idesign ipattern. iThere iis ian iobserver idesign ipattern ibecause ithe idata iis

updating ievery itime ithe itime ifor ithe itrains iare iregularly ibeing ichanged. iThis ishows

that ithere iis ian iobserver idesign ipattern.

**Scenario i2 i-**

In iscenario i2, ithe idesign ipattern ithat ihas ibeen irecognised iis ithe icreational idesign

pattern iand ifactory idesign ipattern.

****

This isituation iuses ia i**'Factory iDesign iPattern'** iwhich ifollows ia i**Creational iStructure** ias ithe iconversion iof ia ifamous ibullet ihell ishooter ivideo igame ifrom ithe i1998 iThe iTouhou itask ifour: iLotus iLand istory iturned iinto idesigned ifor ithe iPC98. itherefore ithe ideveloper/programmer ihas ito itransform iit iusing ia i**Creational iStructure** itransformed ito ithe iAndroid iplatform iand icould ibe irequired ito irun ion ilow igive iup istructures iwhich iare isuitable imethods ito iimplement ithe ienemy iand ibullet icreation iand iperformance.

**Critical iEvaluation ifor ithe iuse iof ithe iPattern i: i**

It iuses ia i**'Factory iDesign iPattern'** iwhich ifollows ia i**'Creational iPattern'** ikind ias iit's ifar ithe ifine iway ito icreate ian iobject. ithat ihave imany ilessons iof ia isimilar ikind ithat iinherit ifrom ia idiscern ielegance, iwe iwill iuse ia imanufacturing iunit imagnificence ito iinstantiate iitems, iin iplace iof iinstantiating ithe igadgets iexplicitly. iThe isport i'the iTouhou imission ifour: iLotus iLand istory' iis inow ibeing ichanged ito ia ifar imuch iless ipowerful iand ia ilow istop imachine ithat's ian iandroid iplatform. iTherefore ithe iprogrammer ihas ito icreate ia ireproduction ior ia ifar ilower imodel iof ithe isame irecreation ifor ithe iandroid iplatform ithat's ia ilow igive iup iand imuch ilow ipowered isystem. iTherefore isituation i2 ishows ia icreational ilayout isample iis ibeing icarried iout. iThis iis ia imanufacturing ifacility ilayout ibecause iin ithe isport ithere iare ione-of-a-kind istyles iof ibullets iwhich ido idifferent iranges iof iharm, ihowever ithe iprimary iaim iof ithe idistinctive ibullets iis ito ideal idamage ito ithe ienemies. iSo iconsequently iit's ifar ia imanufacturing ifacility idesign ipattern.

**Scenario i3 i-** i

In iscenario i3, ithe idesign ipattern ithat ihas ibeen irecognised iis ithe istructural idesign

pattern iand imodel-view-controller idesign ipattern. iThis iscenario imakes iuse iof ithe i'**Model-View-Controller'** ilayout ipattern iwhich ifollows ia i'**Structural idesign isample'** iwhich iis iused iinside ithe isoftware iprogram iimprovement iorganisation. iIt iuses ia ispooler isoftware ithat iis ia iprinter isoftware ithat iallows imanipulation iof iprint iload iby imeans iof ideleting ithe iprint iin ithe iqueue iinside ithe ipc. iThe ispooler isoftware iprogram igives ius ian iapproach iof iclub iorganizations iand iobjecting itogether ito iform ihuge iconstructs, isome iinstances iare iadapters, ibridges, ietc. ithat iassist iimplement ithe isoftware ito ibe iused ion inetworks.

**Critical iEvaluation ifor ithe iuse iof ithe ipattern i-**

This iscenario iuses ithe i'**Model-View-Controller'** idesign isample iwhich ifollows ia i'**Structural ilayout ipattern'**, ithat iis idue ito ithe ifact ithe isoftware idevelopment ienterprise ihas ia icontract ifor igrowing ia iprint ispooler isoftware iprogram ifor ia ihuge iprinter iimprovement iorganisation. iThis iprint ispooler isoftware iwill iassist iin ihandling iall ithe iprint irequests iwhich ican ibe ibeing isent ito ithe imain ilaptop iin ia idependent ishape. iThe iprint ispooler iapplication ipermits ia iuser ito idelete ia iprint itask ibeing iprocessed ior iotherwise icontrol ithe iprint ijobs icurrently iready ito ibe irevealed. iThis ishows ithat ithis istate iof iaffairs ihas ia istructural idesign ipattern. iThis ihas ia iversion iview icontroller ibecause iin ithis ithe iprogramming ibusiness ienterprise iis ithe icontroller ias ithey're iviewing ia iway ito icreate ithe iprint ispooler iprogram, ithe iversion iis ithe isoftware ibecause ithey imay ibe icreating ithe isoftware iprogram ifor ithe ibig iprinting icompany iand ithe iview iis ithe iprinting iorganisation ias ithey iwill ibe icommanding ias ihow ithey iwould iwant ithe ienterprise.

**Conclusion i-**

In conclusion, This document takes a look at the Implementation of code applying design patterns and builds an application derived from UML class diagrams. it's develops code that implements a design pattern for a given cause and examines the usage of design patterns for the given cause. It looks at scenarios and discusses various design patterns with applicable examples of creational, structural and behavioural pattern types. it reconciles the most suitable design pattern from a variety with a series of given situations and critically compares design patterns towards the variety of given scenarios with justifications.

**Resources -**

Atwood, J., 2020. *Understanding Model-View-Controller*. [online] Coding Horror. Available at: <https://blog.codinghorror.com/understanding-model-view-controller/> [Accessed 25 March 2020].

GeeksforGeeks. 2020. *Singleton Design Pattern | Implementation - Geeksforgeeks*. [online] Available at: <https://www.geeksforgeeks.org/singleton-design-pattern/> [Accessed 25 March 2020].

Baeldung.com. 2020. [online] Available at: <https://www.baeldung.com/creational-design-patterns> [Accessed 25 March 2020].

Sourcemaking.com. 2020. *Design Patterns And Refactoring*. [online] Available at: <https://sourcemaking.com/design\_patterns/observer> [Accessed 25 March 2020].

2020. [online] Available at: <https://www.quora.com/What-is-the-difference-between-Creational-Structural-and-Behavioral-Patterns> [Accessed 25 March 2020].

Sites.google.com. 2020. *Singleton Pattern With Example - Wcf Pandu*. [online] Available at: <https://sites.google.com/site/wcfpandu/singleton-pattern-with-example> [Accessed 27 March 2020].