ઑબ્જેક્ટ આધારિત ખ્યાલો 6

આપણે આજે ઇન્ટરનેટ, વેબસાઇટ અને વેબ આધારિત પ્રક્રિયાઓના યુગમાં છીએ કે જ્યાં વિનિયોગનો ઝડપી વિકાસ તેમજ સૉર્સ કોડ (source code)નો પુનઃ ઉપયોગ ઘણો અગત્યનો છે. સૉફ્ટવેર સિસ્ટમનાં વિશ્લેષણ, ડિઝાઇન અને અમલીકરણમાં ઑબ્જેક્ટ (object) આધારિત પદ્ધતિ કે ક્રિયા નોંધપાત્ર ભૂમિકા ભજવે છે. ઑબ્જેક્ટ આધારિત પદ્ધતિનો ઉપયોગ કરીને બનાવવામાં આવેલા સૉફ્ટવેર વધુ વિશ્વસનીય છે અને તેની જાળવણી, પુનઃઉપયોગ અને તેના વિકાસનું કાર્ય ખૂબ સરળ છે. આ પ્રકરણમાં ઑબ્જેક્ટ આધારિત ખ્યાલોની સામાન્ય સમજ આપેલી છે. જાવા પ્રોગ્રામિંગ ભાષાનો ઉપયોગ કરીને આ ખ્યાલોનું અમલીકરણ હવે પછીનાં પ્રકરણોમાં આવરી લેવામાં આવ્યું છે.

પરિચય (Introduction)

ઑબ્જેક્ટ (object) આધારિત પ્રોગ્રામિંગની શરૂઆત 1960ના સમયગાળામાં થઈ અને 1980ના દાયકાની મધ્યથી નવા સૉફ્ટવેર બનાવવામાં પ્રોગ્રામિંગની તે મુખ્ય પદ્ધતિ બની ગઈ. સૉફ્ટવેર સિસ્ટમના અતિ ઝડપથી વધતા કદ અને જટિલતાનું નિયંત્રણ કરવા માટે તેમજ મોટી અને જટિલ સિસ્ટમને સમય સાથે સુધારવાના કાર્યને સરળ બનાવવાના એક માર્ગ તરીકે આ પદ્ધતિને વિકસાવવામાં આવી હતી. કેટલીક પ્રચલિત પ્રોગ્રામિંગ ભાષાઓ જેવી કે C++, Java, C#, VB.net, ASP.net અને PHP ઑબ્જેક્ટ આધારિત પ્રોગ્રામિંગને સમર્થન આપે છે.

પ્રોગ્રામિંગની રીતને આપણે બે પ્રકારમાં વહેંચી શકીએ, જેમકે પ્રક્રિયાગત પ્રોગ્રામિંગ (Structure/Procedural Programming) અને ઑબ્જેક્ટ આધારિત પ્રોગ્રામિંગ. પ્રક્રિયાગત પ્રોગ્રામિંગમાં આપણું કેન્દ્રબિંદુ ડેટા ઉપરની કાર્યપ્રણાલી (functions) તેમજ પ્રક્રિયાઓ (procedures) લખવામાં કેન્દ્રિત રહે છે. ઉદાહરણ તરીકે, પુસ્તકાલય વિનિયોગ સૉફ્ટવેર માટે આપણે પુસ્તકાલયના વિનિયોગની બધી પ્રક્રિયા બાબત વિચારીશું અને આપણું ધ્યાન વિદ્યાર્થી–નોંધણી, પુસ્તક-વિતરણ, પુસ્તક પરત કરવું અને દંડની ગણતરી જેવા વિભાગો ઉપર કેન્દ્રિત રહેશે.

ઑબ્જેક્ટ આધારિત પદ્ધતિમાં આપણા કેન્દ્રબિંદુએ **ઑબ્જેક્ટ (object)** હોય છે કે જે ડેટા તેમજ ક્રિયાત્મકતા (functionality) એમ બંને એકસાથે ધરાવે છે. પુસ્તકાલય વિનિયોગ બનાવવામાં આપણું ધ્યાન વિનિયોગમાં સમાવેશ વિવિધ ઑબ્જેક્ટ ઉપર કેન્દ્રિત હોય છે. અહીં આપણે વિદ્યાર્થી, પુસ્તક અને ગ્રંથપાલ જેવા ઑબ્જેક્ટ વિચારી શકીએ. આવા ઑબ્જેક્ટ એકબીજા સાથે કઈ રીતે સંકળાયેલા છે (association) તે બાબત પણ વિચારવું જોઈએ. ઉદાહરણ તરીકે, વિદ્યાર્થી પુસ્તકાલયમાં પુસ્તક પરત કરે છે.

ઑબ્જેક્ટ આધારિત પ્રોગ્રામિંગ ભાષાઓની તાકાત (ક્ષમતા) પ્રોગ્રામરને વિભાગીય (modular), પુનઃઉપયોગમાં લઈ શકાય તેમજ તે પ્રોગ્રામનો વિકાસ કરી શકાય (extendable) તે પ્રકારનો પ્રોગ્રામનો કોડ લખવાનું સામર્થ્ય પૂરું પાડે છે. આ ક્ષમતાને કારણે પ્રોગ્રામર હયાત મૉડ્યુલમાં ફેરફાર કરીને નવા પ્રોગ્રામની રચના કરી શકે છે. સૉફ્ટવેરના અન્ય ભાગના કોડમાં ખલેલ પહોંચાડ્યા વગર મૉડ્યુલમાં ફેરફાર કરવા કે બદલવાની સમર્થતા વડે તે ભાષાઓને લવચિકતા બક્ષે છે. હયાત કોડનો ફરી ઉપયોગ તેમજ હયાત કોડને સુધારીને ઉપયોગ કરીને સૉફ્ટવેર બનાવવાની ઝડપ વધારી શકાય છે.

ઑબ્જેક્ટ આધારિત પ્રોગ્રામિંગ ઑબ્જેક્ટને (object) પાયાના એકમ તરીકે વાપરે છે. એકસરખા ઑબ્જેક્ટનું ક્લાસ (class)ના ખ્યાલ દ્વારા વર્ગીકરણ કરવામાં આવે છે. કમ્પ્યૂટરની જે ભાષાઓ ઑબ્જેક્ટના ચાર ચોક્કસ ગુણધર્મો (1) **ઍબ્સ્ટ્રૅક્શન** (abstraction) (2) **ઇનકૅપ્સ્યુલેશન** (encapsulation) (3) **પૉલિમૉર્ફિઝમ** (polymorphism) અને (4) **ઇનહેરીટન્સ** (inheritance) પૂરા પાડે છે, તે ભાષા ઑબ્જેક્ટ આધારિત ભાષા તરીકે ઓળખાય છે.

ઑબ્જેક્ટ (Object)

વાસ્તવિક વિશ્વમાં ઑબ્જેક્ટ એ આ દુનિયા જે વસ્તુઓ વડે બનેલી છે, તે વસ્તુ છે. આમાંની કેટલીક વસ્તુઓ વ્યક્તિ, કાર કે કૉફીના પ્યાલા જેવા કોઈ ભૌતિક સ્વરૂપમાં હયાતી ધરાવે છે. અન્ય વસ્તુઓ અમૂર્ત સ્વરૂપે (abstract) હોઈ શકે કે જેને સ્પર્શ ન કરી શકાય અથવા જોઈ ન શકાય, ઉદાહરણ તરીકે તારીખ અને સમય જેવા ખ્યાલો.

દરેક ઑબ્જેક્ટ (object)ની એક અનન્ય ઓળખ હોય છે અને દરેક ઑબ્જેક્ટને એકબીજાથી અલગ ઓળખી શકાય છે. ઉદાહરણ તરીકે, દરેક વ્યક્તિ નામ, શહેર, જાતિ, જન્મતારીખ અને વ્યવસાય જેવી લાક્ષણિકતાઓ ધરાવે છે. ઑબ્જેક્ટ આધારિત પરિભાષામાં આવાં લક્ષણો **પ્રોપર્ટી** (property) અથવા **ઍટ્રિબ્યૂટ** (attribute) તરીકે ઓળખાય છે. (જેને આપણે

ગુણધર્મ કે સંબંધિત ગુણધર્મ પણ કહી શકીએ.) એક વ્યક્તિને બીજી વ્યક્તિથી અલગ પાડવા માટે આપણે તેની લાક્ષણિકતાની કિંમતનો ઉપયોગ કરીએ છીએ. 'રામ' અને 'શ્યામ' નામ બે અલગ-અલગ વ્યક્તિઓની ઓળખ આપે છે, પણ જ્યારે બે વ્યક્તિઓનાં નામ એક જ હોય, ત્યારે જન્મતારીખ જેવી અન્ય લાક્ષણિકતાની મદદથી આપણે તેમને અલગ પાડી શકીએ. આ રીતે ઑબ્જેક્ટને ઓળખવા માટે આપણે આ લાક્ષણિકતા (attribute)ની કિંમત વાપરીએ છીએ. આ કિંમતો **સ્ટેટ** (state) તરીકે ઓળખાય છે. આ ઉપરાંત ઑબ્જેક્ટ સાથે **બિહેવ્યર** (behaviour) સંકળાયેલ હોય છે. ઉદાહરણ તરીકે, વ્યક્તિ જન્મ લે છે, નામ મેળવે છે, સ્થાન બદલે છે. બિહેવ્યર **મેથડ** (method) તરીકે પણ ઓળખાય છે. ઑબ્જેક્ટની કિંમત તેના બિહેવ્યરના કારણે બદલાઈ શકે છે. આ રીતે, વાસ્તવિક વિશ્વમાં કોઈ પણ વસ્તુ (ઑબ્જેક્ટ) તે કયા નામથી ઓળખાય છે (ઓળખ - identity), તે શું છે (તેની સ્થિતિ - state) અને તે શું કરે છે (behaviour) તેના દ્વારા વર્ણવી શકાય છે.

ઑબ્જેક્ટ આધારિત પ્રોગ્રામિંગમાં ઑબ્જેક્ટનું વર્ણન કરતી જુદી-જુદી લાક્ષણિકતાઓ **ડેટાફિલ્ડ** (data-field) તરીકે પણ ઓળખવામાં આવે છે. ઑબ્જેક્ટ સાથે સંકળાયેલા વિવિધ ડેટા એટ્રિબ્યૂટ અને બિહેવ્યર મેથડને સામૂહિક રીતે તેના **મેમ્બર** કે ફિચર (member અથવા feature) તરીકે ઓળખવામાં આવે છે.

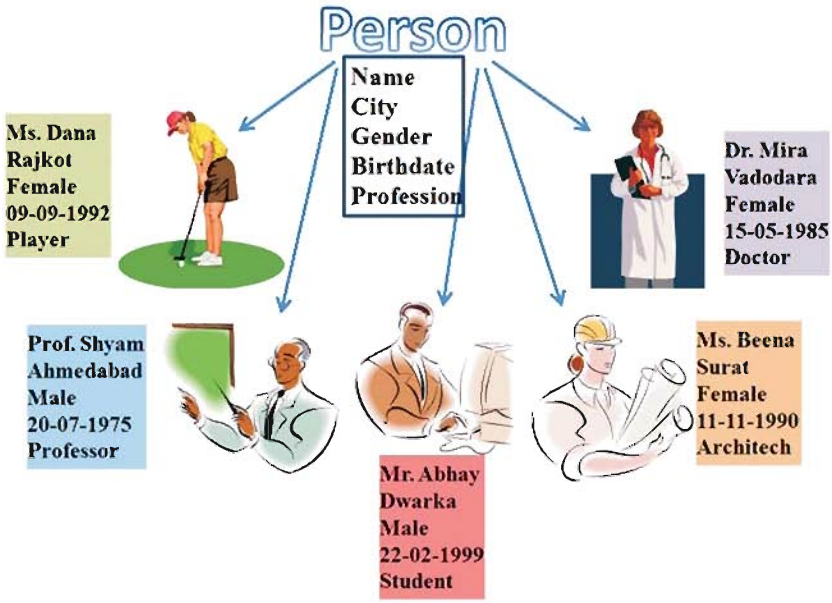
જ્યારે આપણે કોઈ સૉફ્ટવેર વિનિયોગની રચના કરીએ છીએ, ત્યારે આપણે જે ઑબ્જેક્ટ પસંદ કરીએ, તે વિનિયોગ માટે અર્થપૂર્ણ હોવા જોઈએ. ઉદાહરણ તરીકે, રેલ્વે-આરક્ષણ વિનિયોગ માટે રેલ્વે ટ્રેન, પ્રવાસી, ટિકિટ અને સ્ટેશન જેવા ઑબ્જેક્ટ યોગ્ય છે, પણ કાર, કમ્પ્યૂટર અને ઘડિયાળ જેવા ઑબ્જેક્ટ આ વિનિયોગ માટે અસંગત છે.

ક્લાસ (Class)

ઉપર જણાવેલાં person ઑબ્જેક્ટના ઉદાહરણમાં સહેલાઈથી જોઈ શકાય છે કે કેટલાક ઑબ્જેક્ટ એકસરખી લાક્ષણિકતા અને બિહેવ્યર ધરાવતા હોય છે પણ ફક્ત તેના સ્ટેટથી (state) (ડેટાની કિંમત) એકબીજાથી જુદા પડે છે. ઑબ્જેક્ટ આધારિત પદ્ધતિ ક્લાસ (class)ના ખ્યાલનો ઉપયોગ કરે છે, જે ફક્ત તેના એટ્રિબ્યૂટની કિંમતોથી જ અલગ હોય તેવા અમૂર્ત સમકક્ષ (abstractly equivalent) ઑબ્જેક્ટને અભિવ્યક્ત કરવા સમર્થ બનાવે છે. ક્લાસને અનેક જુદા-જુદા ઑબ્જેક્ટની એક બ્લ્યુપ્રિન્ટ (નકશો) તરીકે ગણી શકાય.

ક્લાસ એ સમાન લક્ષણો ધરાવતાં બહુવિધ ઑબ્જેક્ટનું એક ટેમ્પ્લેટ (template) છે. તે એકસમાન એટ્રિબ્યૂટ અને બિહેવ્યર ધરાવતાં વિવિધ ઑબ્જેક્ટનું એક જૂથ છે. એક જ ક્લાસના જુદા-જુદા ઑબ્જેક્ટનો સિમેન્ટિક હેતુ (semantic purpose) એકસરખો હોય છે. આ રીતે ક્લાસ એ કોઈ ચોક્કસ ઑબ્જેક્ટના જૂથની બધી જ સમાન લાક્ષણિકતાનો સમાવેશ કરવા માટેનો એક સર્વસાધારણ ખ્યાલ છે.

ઉદાહરણ તરીકે, આપણી પાસે બધી વ્યક્તિઓના સામાન્ય એટ્રિબ્યૂટ અને બિહેવ્યરનો સમાવેશ કરતો 'Person' નામનો ક્લાસ છે. અહીં દરેક વ્યક્તિ એ ઑબ્જેક્ટ છે અને તે તેના સ્ટેટ એટલે કે એટ્રિબ્યૂટના કિંમતથી તે ઓળખાય છે. આકૃતિ 6.1માં ક્લાસ અને તેના ઑબ્જેક્ટનો તફાવત દર્શાવેલો છે.



આકૃતિ 6.1 : 'Person' નામનો ક્લાસ અને તેના ઑબ્જેક્ટ

હવે આપણે અન્ય ઑબ્જેક્ટ આધારિત ખ્યાલો વિશે શીખતાં પહેલાં ક્લાસ-ડાયાગ્રામ (class-diagram) વિશે ટૂંકમાં પરિચય મેળવીએ.

ક્લાસ-ડાયાગ્રામનો પરિચય (Introduction to Class-Diagram)

ક્લાસ-ડાયાગ્રામ અનેક ક્લાસનો સમૂહ, અવરોધો (constraints) અને જુદા-જુદા ક્લાસ વચ્ચેના સંબંધની સ્થિતિ રજૂ કરે છે.

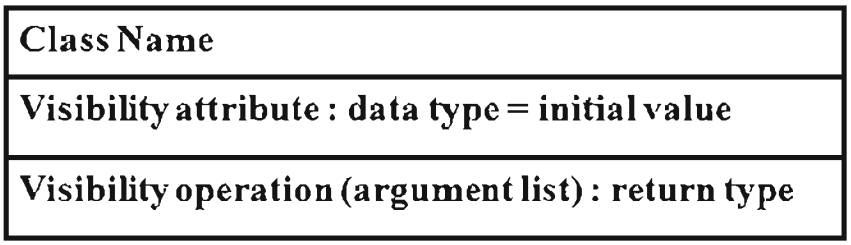
યુનિફાઇડ મૉડેલિંગ લૅંગ્વેજ (Unified Modelling Language - UML)નો ઉપયોગ ઑબ્જેક્ટ આધારિત સૉફ્ટવેરની પ્રતિકૃતિ (model) તૈયાર કરવામાં કરી શકાય કે જે વિનિયોગની રચના તૈયાર કરવામાં મદદરૂપ થાય. UML એ એક દૃશ્ય (visual) મૉડેલિંગ ભાષા છે અને તે ઑબ્જેક્ટ મૅનેજમેન્ટ ગ્રૂપ (Object Management Group - OMG) દ્વારા વ્યાખ્યાયિત કરેલી છે અને તેની જાળવણી કરે છે. UML સૉફ્ટવેર વિનિયોગનાં વિવિધ પાસાંઓ રજૂ કરવા માટે નામનિર્દેશવાળી અનેક આકૃતિઓ જણાવે છે.

ક્લાસ-ડાયાગ્રામનો હેતુ વિનિયોગના સ્થાયી દેખાવની પ્રતિકૃતિ બનાવવાનો છે. ક્લાસ-ડાયાગ્રામ જ ફક્ત રેખાકૃતિઓ (diagrams) છે કે જે ઑબ્જેક્ટ આધારિત ભાષાઓ સાથે સીધેસીધી જોડી શકાય છે અને આથી સૉફ્ટવેર બનાવનાર વ્યક્તિઓમાં તે વ્યાપકપણે વપરાય છે.

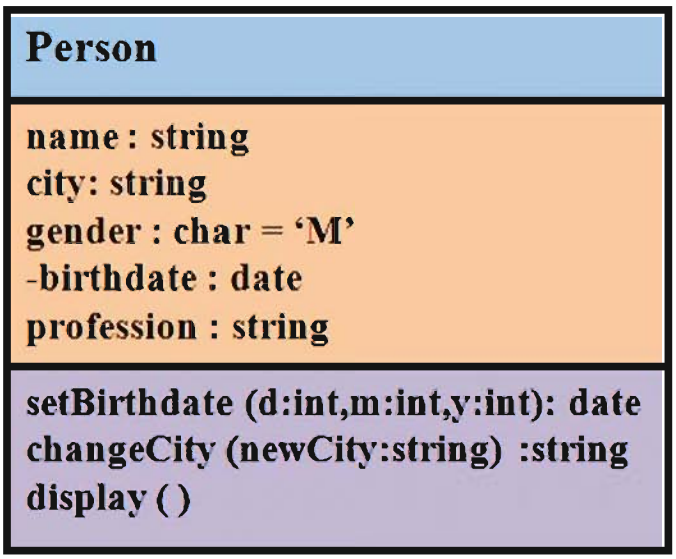
ક્લાસ-ડાયાગ્રામમાં કોઈ પણ ક્લાસને રજૂ કરવા માટેનો આઇકોન નીચે જણાવ્યા પ્રમાણે name, attribute અને behaviourનો સમાવેશ કરવા માટે એક લંબચોરસ ત્રણ વિભાગમાં વિભાજિત કરેલો હોય છે :

1. સૌથી ઉપરના વિભાગમાં ક્લાસનું નામ (class name) હોય છે.
2. વચ્ચેના વિભાગમાં ક્લાસનાં ઍટ્રિબ્યૂટ કે પ્રોપર્ટી હોય છે.
3. સૌથી નીચેના વિભાગમાં ક્લાસનું બિહેવ્યર અથવા ઑપરેશન (operation) અથવા મેથડ (method) હોય છે.

આકૃતિ 6.2માં UML પ્રણાલિકા પ્રમાણે ક્લાસની સચિત્ર રજૂઆત દર્શાવી છે, જ્યારે આકૃતિ 6.3માં 'Person'નો ક્લાસ-ડાયાગ્રામ દર્શાવ્યો છે.



આકૃતિ 6.2 : UML પ્રણાલિકાગત ક્લાસની રજૂઆત



આકૃતિ 6.3 : 'Person' ક્લાસની રેખાકૃતિ

'Person' ક્લાસનો હેતુ ક્લાસ-ડાયાગ્રામના સંદર્ભમાં ક્લાસની કામગીરી સમજવામાં મદદરૂપ માહિતી પૂરી પાડવાનો છે. તેણે ક્લાસનાં દરેક ઍટ્રિબ્યૂટ અને કાર્યોનો સમાવેશ કરવાની જરૂર નથી.

આકૃતિ 6.2માં દર્શાવ્યા પ્રમાણે, UMLની સંકેતલિપિમાં ઍટ્રિબ્યૂટની વાક્યરચના નીચે દર્શાવ્યા પ્રમાણે કરી શકાય :

[<visibility>] <attribute name> [: <attribute data type> [= <initial value>]]

અહીં, ચોરસ કૌંસ [ ]ની જોડીમાં લખવામાં આવેલી બાબત વૈકલ્પિક છે અને કોણીયકૌંસ < > ની જોડીમાં લખેલી બાબતની કિંમત વપરાશકર્તાએ જણાવવી પડે છે. અહીં દ્રશ્યતા (visibility) અંગત, સુરક્ષિત, જાહેર અથવા પૅકેજ (package) હોઈ શકે અને તે જણાવવા માટે અનુક્રમે –, #, + અને ~ સંકેતો વાપરવામાં આવે છે. આપણે દ્રશ્યતા વિશે હવે પછીના પ્રકરણમાં અભ્યાસ કરીશું. ઍટ્રિબ્યૂટ સામાન્ય રીતે ચલ (variable)નો નિર્દેશ કરે છે. ડેટાટાઇપ અને તેની પ્રારંભિક કિંમત પ્રોગ્રામની શરૂઆતમાં ક્યા પ્રકારનો ડેટા સંગ્રહ કર્યો છે અને તેની કિંમત શું છે તેનો નિર્દેશ કરે છે. અહીં જોઈ શકાય છે કે માત્ર ઍટ્રિબ્યૂટ નેઇમ (attribute-name) જ ફરજિયાત છે અને અન્ય તમામ બાબત વૈકલ્પિક છે.

ઍટ્રિબ્યૂટ ઘોષિત કરવા માટે નીચે કેટલાંક ઉદાહરણ આપેલાં છે :

name : string

- balance : float = 0.0

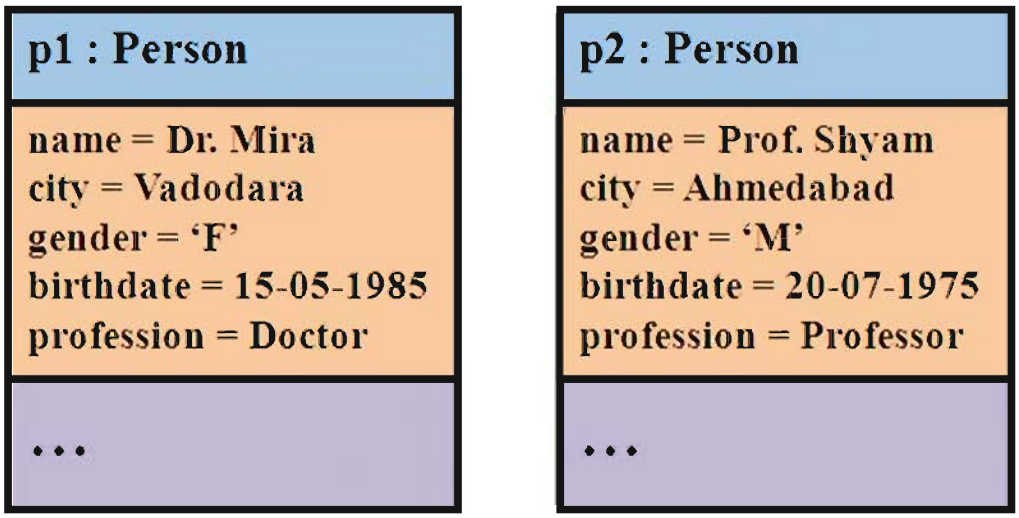
આકૃતિ 6.2માં દર્શાવ્યા પ્રમાણે, UMLની સંકેતલિપિમાં કોઈ કાર્યને નીચે જણાવેલી વાક્યરચના વાપરીને જણાવી શકાય છે :

[<visibility>] <method name> (parameter list separated by comma) : <return data type>

અગાઉના ઉદાહરણમાં મેથડ (method)નું ઉદાહરણ setBirthdate(d:int, m:int, y:int) : date છે. અહીં પ્રાચલ ડેટાટાઇપ અને વૈકલ્પિક પ્રારંભિક કિંમત સાથે જણાવી શકાય છે; ઉદાહરણ તરીકે gender : char = 'M'. પ્રાચલ ફક્ત વાંચી જ શકાય (read only) તેવા છે કે નહીં, તેના આધારે તે નિવેશ અથવા નિર્ગમ તરીકે જણાવી શકાય છે.

UML રેખાકૃતિ વિનિયોગ કઈ પ્રોગ્રામિંગ ભાષામાં કોડ કરવાનો છે, તેના ઉપર આધારિત નથી. કેટલીક સૉફ્ટવેર બનાવનાર વ્યક્તિઓ ઍટ્રિબ્યૂટ અને કાર્યોને પ્રમાણભૂત UML સંકેતલિપિમાં જણાવવાને બદલે પ્રોગ્રામિંગ ભાષાના વધુ પરિચિત માળખામાં જણાવવાનું પસંદ કરે છે. આ પદ્ધતિ જ્યાં સુધી દરેક સંબંધિત વ્યક્તિને માટે અર્થપૂર્ણ રહે છે, ત્યાં સુધી તે બરાબર છે.

વિનિયોગના અમલ દરમિયાન ઑબ્જેક્ટને તેના સ્ટેટ (state) વડે રજૂ કરવામાં આવે છે. આ રીતે ઑબ્જેક્ટ ચલિત (dynamic) હોય છે. આકૃતિ 6.1ને અનુરૂપ 'person' ક્લાસના ઑબ્જેક્ટ (ઇન્સ્ટન્સ - instance તરીકે પણ ઓળખવામાં આવે છે)ને આકૃતિ 6.4માં દર્શાવ્યા પ્રમાણે રજૂ થાય છે.



આકૃતિ 6.4 : 'Person' ક્લાસના p1 અને p2 ઑબ્જેક્ટની રેખાકૃતિ

ઈનકૅપ્સ્યુલેશન (Encapsulation)

કોઈ પણ કમ્પ્યૂટર પ્રોગ્રામનાં બે મૂળ અંગ ડેટા અને કાર્ય છે. સંગઠિત પ્રોગ્રામિંગ (Structured Programming) આ બે ઘટકોને અલગ-અલગ ઘટક તરીકે ગણે છે જ્યારે ઑબ્જેક્ટ આધારિત પ્રોગ્રામિંગ તેને એક ઘટક તરીકે જુએ છે.

પ્રક્રિયાગત પ્રોગ્રામિંગમાં (Procedural Programming) પ્રોગ્રામના કોઈ પણ ભાગમાં ડેટાની કિંમત બદલી શકાય છે. તે ફેરફારથી સુરક્ષિત નથી. આ સમસ્યાનો ઉકેલ ઑબ્જેક્ટ આધારિત પ્રોગ્રામિંગમાં ઈનકૅપ્સ્યુલેશન દ્વારા લાવી શકાય છે. અહીં ડેટા અને મેથડ વડે માહિતીની ગણતરી થાય તે સમયે પ્રોગ્રામના અન્ય ઘટકો વડે ડેટામાં ફેરફાર અથવા દુરુપયોગ સામે સુરક્ષિતતા બક્ષવામાં આવે છે. આ પ્રકારની રચના કે જેના વડે ડેટા અને મેથડ સામે રક્ષણ પૂરું પાડવામાં આવે છે તેને **ઈનકૅપ્સ્યુલેશન (Encapsulation)** કહેવામાં આવે છે. ડેટા અને મેથડને લપેટીને (વીંટીને) બનાવેલા એક એકમ જે ક્લાસના નામથી જાણીતા છે તેને અંગત (private) ઘોષિત કરવાથી શક્ય બને છે, અને ક્લાસના અંગત સભ્ય (private member) બહારના વિશ્વને સીધા ઉપલબ્ધ થવા દેતા નથી. જો તેની જરૂર હોય, તો સાર્વજનિક મેથડ (public method) વડે ડેટાને ઉપલબ્ધ બનાવી શકાય છે. આ રીતે, ઈનકૅપ્સ્યુલેશન ડેટાને છુપાવવાની ક્ષમતા પૂરી પાડે છે. આપણે આ પછીનાં પ્રકરણોમાં ક્લાસ અને અંગત/સાર્વજનિક ડેટા અને મેથડ વિશે અભ્યાસ કરીશું.

ઈનકૅપ્સ્યુલેશન બિનઈરાદાપૂર્વકની ક્રિયાઓ અને બહારના અન્ય ઑબ્જેક્ટ વડે જરૂર વગરના ડેટાને મેળવવાથી સલામત રાખે છે. પ્રક્રિયાગત પ્રોગ્રામિંગમાં સાર્વજનિક ડેટાવિસ્તાર માહિતીની વહેંચણી માટે સામાન્ય રીતે વપરાય છે, જ્યારે ઑબ્જેક્ટ આધારિત પ્રોગ્રામિંગ તેનાથી વિરુદ્ધ અન્ય પ્રોગ્રામ્સ વડે સાર્વજનિક ડેટાને (વૈશ્વિક ચલ સિવાયના ડેટાના ઉપયોગને) સીધો મેળવવાની ક્રિયાને અટકાવે છે. ઑબ્જેક્ટ પોતાના ડેટાની કિંમતમાં જ ફેરફાર કરી શકે છે. અન્ય ઑબ્જેક્ટ તેને જોઈ શકે છે અથવા તે ઑબ્જેક્ટના માલિક (owner)ને સંદેશો મોકલીને આ ડેટાને બદલી શકે છે.

ડેટા-ઍબ્સ્ટ્રૅક્શન (Data-Abstraction)

ડેટા-ઍબ્સ્ટ્રૅક્શન એ ઑબ્જેક્ટની જરૂરી લાક્ષણિકતાઓને તેના અમલીકરણની વિગત સિવાયની માહિતીને રજૂ કરવાની પ્રક્રિયા છે. ઍબ્સ્ટ્રૅક્શન એ એક ખ્યાલ છે જે આંટીઘૂંટી કે ગૂંચને છુપાવે છે અને તે જે કાર્ય કરે છે, તે જણાવે છે પણ કઈ રીતે કરે છે, તે જણાવતો નથી.

હવે આપણે આપણી જિંદગીનું એક વાસ્તવિક ઉદાહરણ ટેલિવિઝનનું લઈએ. આપણે ટેલિવિઝનને ચાલુ અને બંધ કરી શકીએ છીએ, ચેનલ બદલી શકીએ છીએ તેમજ આપણને અનુકૂળ અવાજનું માપ રાખી શકીએ છીએ. પણ આપણે તેની આંતરિક રચના જાણતા નથી કે તે હવા કે તાર દ્વારા કઈ રીતે સંકેતો મેળવે છે, તે સંકેતોનું કઈ રીતે રૂપાંતર કરે છે અને અંતમાં તેને પડદા ઉપર પ્રદર્શિત કરે છે. આ રીતે ટેલિવિઝનનું આંતરિક કાર્ય તેના બાહ્ય સેતુથી તદ્દન અલગ છે. આપણે તેની આંતરિક રચનાની સમજ કે જ્ઞાન વિના તેનો ઉપયોગ પાવર-બટન, ચેનલ બદલવાનાં બટન અને અવાજના વૉલ્યુમમાં ફેરફાર તેના બાહ્ય સેતુ વડે કરી શકીએ છીએ.

આ રીતે ડેટા-ઍબ્સ્ટ્રૅક્શન એ એક એવી કાર્યરીતિ (ટેક્નિક) છે જે તે વાપરવાના સેતુ અને અમલીકરણને અલગ કરવાના ખ્યાલ પર આધારિત છે. તે પ્રોગ્રામિંગમાં નવો ખ્યાલ નથી. અનેક વ્યક્તિઓ કદાચ આ બાબતની જાણ વગર અમુક અંશે તેનો ઉપયોગ અત્યાર સુધી કરે જ છે. ઉદાહરણ તરીકે, આપણે જ્યારે C પ્રોગ્રામિંગમાં sqrt(25) અને printf("Hello world") જેવા વિધેયનો ઉપયોગ કરીએ છીએ ત્યારે આપણે કદી પણ એ નથી વિચાર્યું કે તે કઈ રીતે કાર્ય કરતા હશે.

જરૂરી નિવેશ ડેટા પ્રાચલો સાથે ઉપયોગકર્તા નિર્મિત (user-defined) વિધેય પણ ડેટા-ઍબ્સ્ટ્રૅક્શન પૂરું પાડે છે. હવે આપણે સ્ટ્રક્ચર (structure)નો ઉપયોગ કરીને એક C પ્રોગ્રામ વડે 'date' પ્રકારના ડેટા જણાવીએ. અહીં ડેટા ત્રણ ઘટકોનો બનેલો છે : દિવસ, માસ અને વર્ષ. આપણે આ ઘટકો માટે ડેટાનો મૂળભૂત પૂર્ણાંક સંખ્યા પ્રકાર (primitive integer datatype)નો ઉપયોગ કરીએ. હવે 'dateDiff' નામનું વિધેય લો, જેમાં બે તારીખ પ્રાચલ તરીકે લેવામાં આવે છે અને આ બે તારીખ વચ્ચે કેટલા દિવસ છે તે કિંમત પરત આપે છે. આ વિધેયના ઉપયોગકર્તાએ જાણવાની જરૂર નથી કે બે તારીખના તફાવતની ગણતરી કઈ રીતે થાય છે. પણ ઉપયોગકર્તા તે વિધેયની ફક્ત સિગ્નેચર(signature), એટલે કે વિધેયનું નામ, પ્રાચલની સંખ્યા અને તેનો પ્રકાર તેમજ પરત મળતી કિંમતના પ્રાચલનો પ્રકાર જાણે તે જરૂરી છે. જો કોઈ કારણથી તફાવત શોધવાની પ્રક્રિયામાં ફેરફાર થાય, તો આ વિધેય વાપરનાર પ્રોગ્રામના કોઈ પણ ભાગમાં તેની અસર થતી નથી.

આ રીતે ડેટા-ઍબ્સ્ટ્રૅક્શન આપણને વાપરવા માટેની એક રૂપરેખા કે ઢાંચો (templates) પૂરો પાડે છે. સિસ્ટમ ડેટાનો સંગ્રહ, નિર્માણ અને જાળવણી કઈ રીતે થાય છે, તેની કેટલીક માહિતી ગુપ્ત રાખે છે. બહારની દુનિયાને જે કંઈ દૃશ્યમાન

છે, તે ડેટા પ્રકારની અમૂર્ત (abstract) રીતભાત (behaviour) છે; પણ તેનું અમલીકરણ કઈ રીતે થયેલું છે, તે ગોપનીય રહે છે અને આથી તેનો ઉપયોગ કરનાર વ્યક્તિને અસર કર્યા વગર જરૂરિયાત પ્રમાણે તેમાં ફેરફાર કરી શકાય છે.

C અને C++ ના એબ્સ્ટ્રૅક્ટ ડેટાટાઇપ્સ (Abstract Data Types - ADT) અથવા સ્ટ્રક્ચર (structures - struct) અને C++/જાવાના ક્લાસ એ ડેટા-એબ્સ્ટ્રૅક્શનનાં ઉદાહરણ છે. ADTમાં આપણે ડેટાનો પ્રકાર વ્યાખ્યાયિત કરીને તેના ઉપરની પ્રક્રિયાઓ ફક્ત જણાવીએ છીએ. આપણે તે પ્રક્રિયાઓનું અમલીકરણ કઈ રીતે થાય છે, તે બતાવતા નથી.

ડેટા ઇનકૅપ્સ્યુલેશન અને ડેટા-એબ્સ્ટ્રૅક્શનનો મૂળભૂત તફાવત : ઇનકૅપ્સ્યુલેશન પ્રોગ્રામની બહારથી ડેટા મેળવવાનું અટકાવીને (inaccessible) ડેટાને સુરક્ષિત કરે છે, જ્યારે એબ્સ્ટ્રૅક્શન પ્રક્રિયાના અમલીકરણની માહિતી ગુપ્ત રાખીને ડેટાની રજૂઆત વડે ડેટાને સુરક્ષિત બનાવે છે.

મૅસેજિંગ (Messaging)

ઑબ્જેક્ટ આધારિત પરિભાષામાં મૅથડ (method) બોલાવવાની ક્રિયાને **મૅસેજ (message)** કહેવામાં આવે છે. ઇનકૅપ્સ્યુલેશનને કારણે બધી મૅથડકૉલનું નિયંત્રણ ઑબ્જેક્ટ વડે થાય છે, જે (ઑબ્જેક્ટ) મૅથડને ઓળખે છે.

જુદા-જુદા ક્લાસમાં સમાન નામ ધરાવતી એકસરખી મૅથડ હોઈ શકે. ઉદાહરણ તરીકે, 'date' ક્લાસમાં 'display' નામની મૅથડ છે જે 'date' ઑબ્જેક્ટને અમુક ચોક્કસ સ્વરૂપમાં પ્રદર્શિત કરે છે. ધારો કે 'time' અને 'person' નામના અન્ય ક્લાસ છે, અને સમય ચોક્કસ સ્વરૂપમાં પ્રદર્શિત કરવા અને વ્યક્તિની માહિતી પ્રદર્શિત કરવા માટે બંનેની એક જ નામની મૅથડ 'display' છે. જ્યારે પ્રોગ્રામમાં 'display' મૅથડ વાપરવામાં આવે છે, ત્યારે કઈ મૅથડનો અમલ કરવાનો છે તે કઈ રીતે જાણવું ? જે ઑબ્જેક્ટ વડે મૅથડ વાપરવામાં આવે તેની મદદથી તે નક્કી થાય છે. ઉદાહરણ તરીકે, જો 'person' ક્લાસના ઑબ્જેક્ટ વડે 'display' વપરાય, તો તે 'person' ક્લાસની 'display' મૅથડનો અમલ કરશે.

પૉલિમૉર્ફિઝમ (Polymorphism)

પૉલિમૉર્ફિઝમ એટલે 'અનેક સ્વરૂપ' કે 'બહુરૂપતા'. એક મૅથડ કે પ્રક્રિયાનાં જુદાં-જુદાં સ્વરૂપ (form) હોઈ શકે.

ધારો કે આપણે બે સંખ્યામાંથી મોટી સંખ્યા શોધવાનું વિધેય 'max' લખેલું છે. આ વિધેય પ્રાચલ તરીકે બે પૂર્ણાંક સંખ્યા લે છે અને મોટી પૂર્ણાંક કિંમત પરત આપે છે. હવે ધારો કે 'max' નામનું એક વધારે વિધેય આપણે ઉમેરવા ઇચ્છીએ છીએ, જે કોઈ ઍરે (array)માં સંગ્રહ કરેલી પૂર્ણાંક સંખ્યાઓમાંથી મહત્તમ સંખ્યા શોધે. આ વિધેય તે ઍરે અને તેના કદને પ્રાચલ તરીકે લેશે અને મહત્તમ પૂર્ણાંક સંખ્યા પરત આપશે. શું એક જ નામનાં એક કરતાં વધુ વિધેયો વ્યાખ્યાયિત કરવા શક્ય છે ? કેટલીક પ્રોગ્રામિંગ ભાષામાં આ પ્રશ્નનો જવાબ 'ના' છે, પણ ઑબ્જેક્ટ આધારિત પ્રોગ્રામિંગમાં જ્યાં સુધી મૅથડ તેની સિગ્નેચર (પ્રાચલની સંખ્યા અને પ્રકાર)થી અલગ છે, ત્યાં સુધી તે પ્રશ્નનો જવાબ 'હા' છે.

ઑબ્જેક્ટ આધારિત પ્રોગ્રામિંગ એક જ ક્લાસમાં એક કરતાં વધારે મૅથડ કે જેનાં નામ સરખાં હોય પણ સિગ્નેચરથી અલગ હોય તેને વ્યાખ્યાયિત કરવાની મંજૂરી આપે છે. આ લાક્ષણિકતાને **ફંક્શન કે મૅથડ ઓવરલોડિંગ (function or method overloading)** કહેવામાં આવે છે.

ઑબ્જેક્ટ આધારિત પ્રોગ્રામિંગ ઑબ્જેક્ટ ઉપર પ્રક્રિયકો સાથેની પદાવલિ લખવાની પણ છૂટ આપે છે. ઉદાહરણ તરીકે, આપણે 'date1-date2' જેવી પદાવલિ પણ વાપરી શકીએ, જ્યાં બંને સંકાર્ય (operands) એ 'date' ક્લાસના ઑબ્જેક્ટ છે. અહીં '-' પ્રક્રિયા એ બે સંખ્યાની બાદબાકી કરવાની પ્રક્રિયા જેમકે n1-n2, કરતાં અલગ રીતે કરવામાં આવે છે. અહીં એ જ પ્રક્રિયાનો અર્થ સંકાર્ય (operands)ના ડેટા પ્રકારના આધારે અલગ કરવામાં આવે છે. આ પ્રકારની બહુરૂપતા (પૉલિમૉર્ફિઝમ) **ઑપરેટર ઓવરલોડિંગ (operator overloading)**ને કારણે શક્ય બને છે.

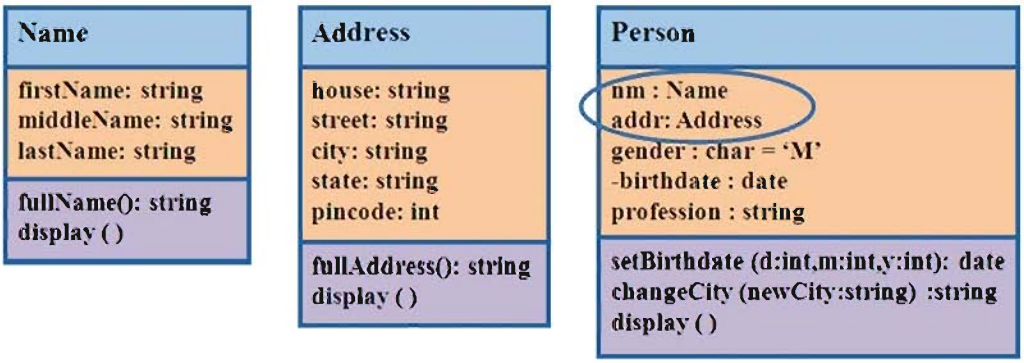
આ રીતે બે પ્રકારનાં ઓવરલોડિંગ વડે પૉલિમૉર્ફિઝમ શક્ય બને છે : (1) ફંકશન ઓવરલોડિંગ (function overloading) અને (2) ઑપરેટર ઓવરલોડિંગ (operator overloading). સામાન્ય રીતે, અલગ-અલગ સંદર્ભમાં એક જ નામના જુદા-જુદા અર્થની ક્ષમતા કે સામર્થ્યને ઓવરલોડિંગ (overloading) કહેવામાં આવે છે.

એગ્રિગેશન અને કૉમ્પોઝિશન (Aggregation and Composition)

જ્યારે એક ક્લાસના ઑબ્જેક્ટ બીજા ક્લાસના ઑબ્જેક્ટથી બનેલા હોય, ત્યારે તેને એગ્રિગેશન (aggregation) અથવા કૉમ્પોઝિશન (composition) કહેવામાં આવે છે. તે અલગ-અલગ ક્લાસ વચ્ચેનો 'પૂર્ણ' અથવા 'તેમાંનો અંશ' સંબંધ રજૂ કરે છે. ઉદાહરણ તરીકે, મધરબોર્ડ કમ્પ્યૂટરનો એક ભાગ છે.

આપણે જાણીએ છીએ કે કમ્પ્યૂટર મધરબોર્ડ, સ્ક્રીન, કી-બોર્ડ અને માઉસ જેવા અનેક ઘટકોનું બનેલું હોય છે. સ્ક્રીન સ્વયં જ લંબાઈ, પહોળાઈ અને મોડલ જેવા ગુણધર્મો (ઍટ્રિબ્યૂટ) સાથેનો એક ક્લાસ હોઈ શકે. એ જ રીતે મધરબોર્ડને પણ મોડલ અને કંપની જેવા ગુણધર્મો સાથેનો ક્લાસ બનાવી શકાય. જ્યારે આપણે 'computer' નામનો ક્લાસ વ્યાખ્યાયિત કરીએ, ત્યારે તેમાં 'motherboard' અને 'screen' ક્લાસના ઑબ્જેક્ટ જેવા ઍટ્રિબ્યૂટનો સમાવેશ કરેલો હશે. આ રીતે તેમાં સમાવેશનો નિર્દેશ સૂચવે છે.

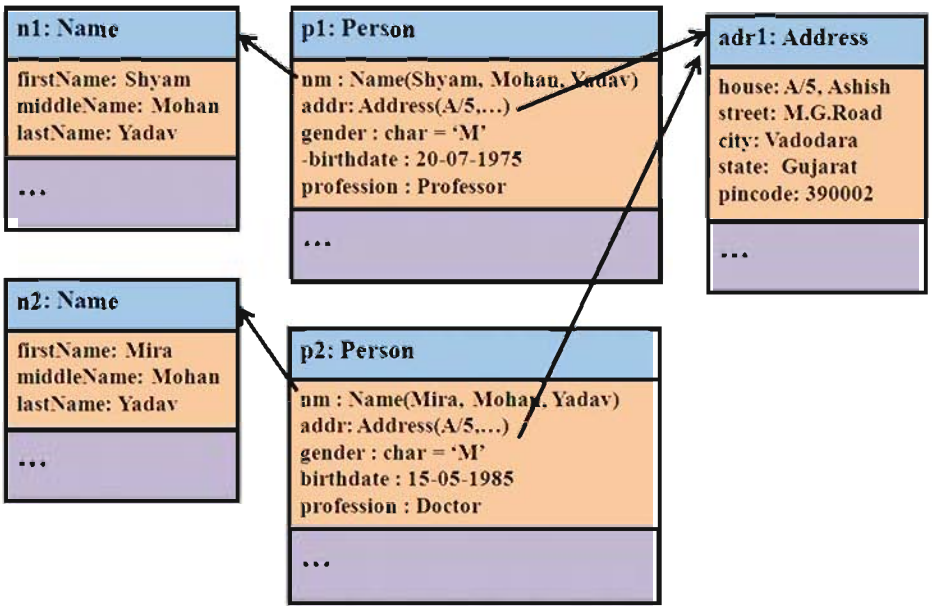
હવે આપણે અગાઉ ચર્ચા કરેલા 'Person' ક્લાસમાં ફેરફાર કરીએ. સૌપ્રથમ આકૃતિ 6.5માં દર્શાવ્યા પ્રમાણે આપણે બે નવા ક્લાસ 'Name' અને 'Address' બનાવીએ. આ 'Name' ક્લાસમાં firstname, middlename અને lastname ઍટ્રિબ્યૂટ છે અને 'Address' ક્લાસમાં house (જે ઘરની વિગતોનો નિર્દેશ કરે છે), street, city, state અને pin code ઍટ્રિબ્યૂટ છે. હવે 'Person' ક્લાસને બદલીએ અને ઍટ્રિબ્યૂટનાં નામ Name અને Addressને બદલી અનુક્રમે nm અને addr રાખીએ. nm અને addr ઍટ્રિબ્યૂટનો ડેટાપ્રકાર અનુક્રમે 'Name' અને 'Address' ક્લાસ છે. આ રીતે 'Person' ક્લાસમાં 'Name' અને 'Address' ક્લાસના ઑબ્જેક્ટ છે.



આકૃતિ 6.5 : 'Name' અને 'Address' ક્લાસના ઍટ્રિબ્યૂટ સાથેનો 'Person' ક્લાસ

એગ્રિગેશન અને કૉમ્પોઝિશનની તુલના (Aggregation Vs Composition)

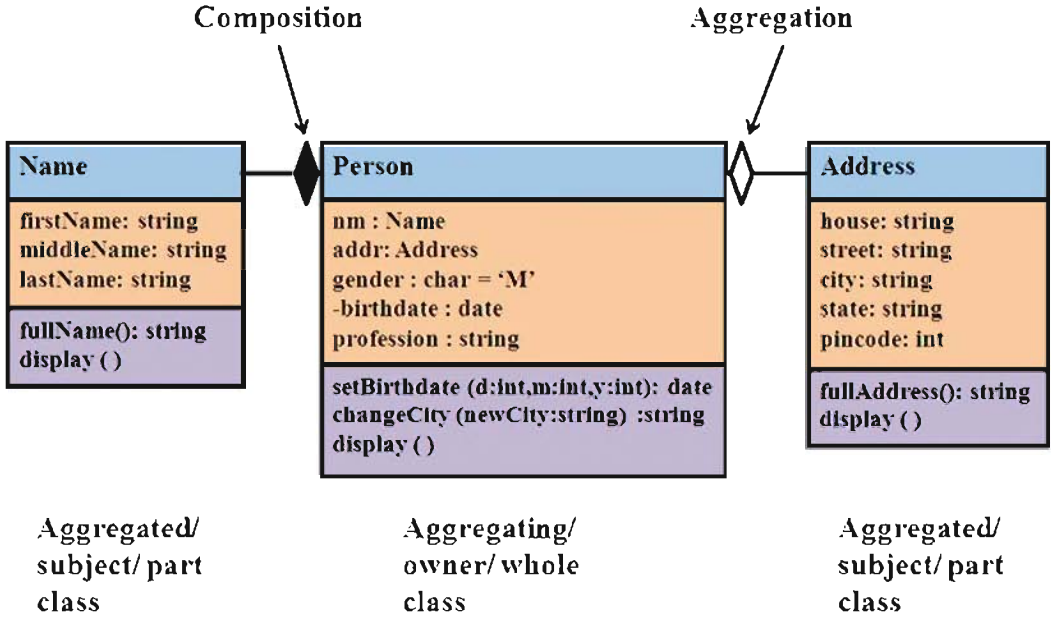
એગ્રિગેશન બે ક્લાસ વચ્ચેના ભિન્ન (non-exclusive) સંબંધને રજૂ કરે છે. એગ્રિગેશનમાં કોઈ ક્લાસ કે જે ઓનર ક્લાસનો એક ભાગ છે, તે સ્વતંત્ર રીતે અસ્તિત્વ ધરાવે છે. આ આંશિક ક્લાસ ઑબ્જેક્ટનો કાર્યકાળ ઓનર ક્લાસ (ownerclass)થી નક્કી થતો નથી. ઉદાહરણ તરીકે, મધરબોર્ડ જોકે કમ્પ્યૂટરનો એક ભાગ છે, પણ તે કમ્પ્યૂટરથી સ્વતંત્ર એક અલગ ઘટક તરીકે અસ્તિત્વ ધરાવે છે. આ રીતે મધરબોર્ડ કમ્પ્યૂટર સાથે અભિન્ન રીતે (exclusively) સંકળાયેલ નથી. આ જ રીતે આકૃતિ 6.6માં દર્શાવ્યા પ્રમાણે બે અથવા વધારે વ્યક્તિઓ address ઑબ્જેક્ટનો ઉપયોગ કરે છે. આથી, address એ કોઈ એક જ વ્યક્તિ માટે હોય તે જરૂરી નથી.



આકૃતિ 6.6 : અલગ-અલગ name ધરાવતા પણ એક જ address ધરાવતા બે 'Person' ઑબ્જેક્ટ

આકૃતિ 6.7માં દર્શાવ્યા પ્રમાણે મૂળભૂત એગ્રિગેશનને પૂર્ણ ક્લાસને અડીને એક ખાલી હીરાના ચિહ્નનો ઉપયોગ કરીને બનાવવામાં આવે છે.

કોમ્પોઝિશન બે ક્લાસ વચ્ચેનો અજોડ (exclusive) સંબંધ જણાવે છે. કોમ્પોઝિશન એક પ્રબળ કે મજબૂત પ્રકારનું એગ્રિગેશન છે, જેમાં આંશિક ક્લાસનો કાર્યકાળ ઓનર ક્લાસના અસ્તિત્વ આધારિત હોય છે. જો એકત્ર ક્લાસ (aggregating class)નો ઑબ્જેક્ટ દૂર કરવામાં આવે, તો તેના આંશિક ક્લાસ ઑબ્જેક્ટ પણ નષ્ટ થશે. ઉદાહરણ તરીકે, 'Person' ક્લાસનો કોઈ ઑબ્જેક્ટ રિલીટ કરવામાં આવે, તો 'Name' ક્લાસનો ઑબ્જેક્ટ પણ નષ્ટ થશે. આકૃતિ 6.6માં દર્શાવ્યા પ્રમાણે Name અજોડ રીતે માત્ર એક જ વ્યક્તિ સાથે જોડાયેલું છે. કોમ્પોઝિશનનો સંબંધ આકૃતિ 6.7માં દર્શાવ્યા પ્રમાણે એક ભરાયેલા હીરાના ચિહ્નને પૂર્ણ ક્લાસને અડીને બતાવવામાં આવે છે.



આકૃતિ 6.7 : કોમ્પોઝિશન અને એગ્રિગેશન

ઉપર આપેલા ઉદાહરણમાં, 'Person' ક્લાસ અને 'Name' ક્લાસ વચ્ચેનો સંબંધ કોમ્પોઝિશન રિલેશનશિપ છે, જ્યારે 'Person' ક્લાસ અને 'Address' ક્લાસ વચ્ચેનો સંબંધ એગ્રિગેશન રિલેશનશિપ છે. એક જ એડ્રેસ એક કરતાં વધારે વ્યક્તિઓનું હોઈ શકે. આથી, જ્યારે કોઈ વ્યક્તિ નષ્ટ કરવામાં આવે છે, ત્યારે તેને સુસંગત 'Name' ઑબ્જેક્ટને નષ્ટ કરવામાં આવે છે, પણ 'Address' નષ્ટ કરવામાં આવતું નથી.

નોંધ : કોઈ ક્લાસ જ્યારે અન્ય ક્લાસના ઑબ્જેક્ટ ધરાવતા હોય ત્યારે તે ઓનર ક્લાસ (ownerclass) અથવા પૂર્ણ ક્લાસ (whole class) અથવા એકત્ર ક્લાસ (aggregating class) તરીકે ઓળખાય છે. ઉદાહરણ તરીકે, આકૃતિ 6.7માં દર્શાવેલ 'Person' ક્લાસ એગ્રિગેટિંગ ક્લાસનું ઉદાહરણ છે.

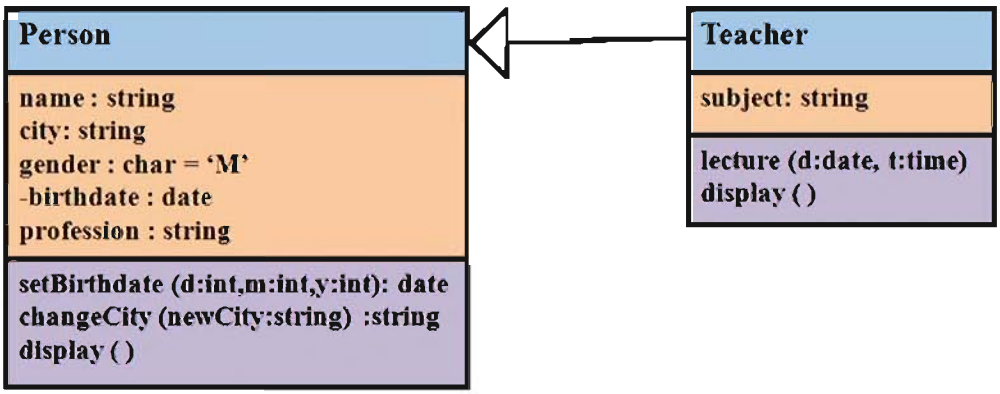
જે ક્લાસ ઓનરક્લાસમાં સમાયેલ છે, તેને સબ્જેક્ટ ક્લાસ (subject class) અથવા આંશિક ક્લાસ (part class) અથવા એકત્રિત ક્લાસ (aggregated class) તરીકે ઓળખાય છે. ઉદાહરણ તરીકે, આકૃતિ 6.7માં દર્શાવેલ 'Name' અને 'Address' ક્લાસ એગ્રિગેટેડ ક્લાસનાં ઉદાહરણ છે.

ઈનહેરિટન્સ (Inheritance)

ઈનહેરિટન્સ સામાન્ય રીતે બે ક્લાસ વચ્ચે 'એક પ્રકારનો' ('is-a-kind-of') સંબંધ જણાવે છે. જ્યારે એક ક્લાસ અન્ય ક્લાસનો પ્રકાર હોય ત્યારે આ યોગ્ય છે. ઉદાહરણ તરીકે, શિક્ષક એક પ્રકારની વ્યક્તિ છે. આથી, 'Person' ક્લાસના બધા એટ્રિબ્યૂટ અને મેથડ 'Teacher' ક્લાસને પણ લાગુ પડે છે. બીજા શબ્દોમાં કહીએ તો 'Person' ક્લાસના બધા એટ્રિબ્યૂટ અને બિહેવ્યર (ગુણધર્મ) 'Teacher' ક્લાસમાં વારસા (inherit)માં મળે છે. આ ઉપરાંત 'Teacher' ક્લાસના વધારાના એટ્રિબ્યૂટ જેવાં કે subject અને મેથડ જેમકે વિષયનાં lecture હોઈ શકે. આવા સંજોગોમાં 'Person' ક્લાસનો ઉપયોગ કરીને 'Teacher' ક્લાસ વ્યાખ્યાયિત કરી શકાય.

ઇનહેરિટન્સ અન્ય હયાત ક્લાસના ગુણધર્મોને વારસામાં મેળવીને ઑબ્જેક્ટના નવા ક્લાસને વ્યાખ્યાયિત કરવાના સામર્થ્યનો નિર્દેશ કરે છે. ઑબ્જેક્ટ આધારિત પરિભાષામાં નવા ક્લાસને સબક્લાસ (subclass) અથવા ચાઇલ્ડક્લાસ (childclass) અથવા ડિરાઇવ્ડ ક્લાસ (derived class) કહેવામાં આવે છે, જ્યારે હયાત ક્લાસને સુપર ક્લાસ (super class) અથવા પેરેન્ટક્લાસ (parent class) અથવા બેઇઝ ક્લાસ (base class) કહેવામાં આવે છે. સુપર ક્લાસનાં ડેટા ઍટ્રિબ્યૂટ અને મેથડ ઑબ્જેક્ટનો સબક્લાસમાં તેનાં declarations ફરી લખ્યા વિના ઉપલબ્ધ હોય છે. જ્યાં હયાત મેથડ ફરી વ્યાખ્યાયિત કર્યા સિવાય વાપરવાની હોય ત્યાં આ ગુણધર્મ પુનઃઉપયોગી સુવિધા પૂરી પાડે છે. વધારામાં સબક્લાસમાં નવો ડેટા અને મેથડ સત્યનો ઉમેરો વિસ્તૃત કરવા માટે કરી શકાય છે. સબક્લાસમાં જરૂર પ્રમાણે મેથડને ફરી વ્યાખાયિત કરવાની મંજૂરી આપે છે.

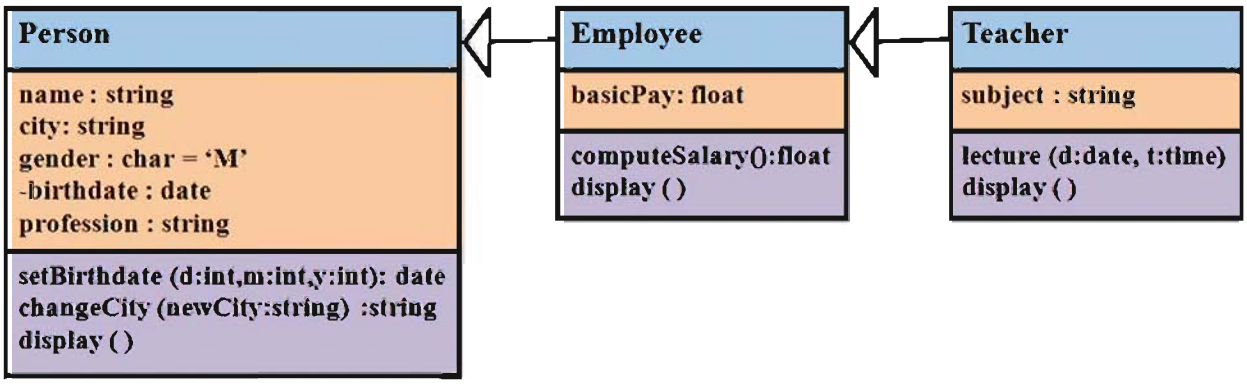
ક્લાસ ડાયાગ્રામમાં ઇનહેરિટન્સ બતાવવા માટે આકૃતિ 6.8માં દર્શાવ્યા પ્રમાણે સુપરક્લાસ તરફ નિર્દેશિત કરતા તીરનો ઉપયોગ કરવામાં આવે છે. ઉપરના ઉદાહરણમાં 'Person' એ સુપરક્લાસ છે અને 'Teacher' સબક્લાસ છે.



આકૃતિ 6.8 : ઇનહેરિટન્સનું ઉદાહરણ

સર્વસામાન્ય નિયમ એ ઇનહેરિટન્સનું બીજું નામ છે અથવા એક સંબંધ છે. બે ક્લાસમાંથી જ્યાં એક ક્લાસ બીજા ક્લાસની વિશિષ્ટ આવૃત્તિ છે, તેની વચ્ચેના સંબંધનો નિર્દેશ કરે છે. સુપરક્લાસમાં સામાન્ય ઍટ્રિબ્યૂટ અને મેથડને વ્યાખ્યાયિત કરવામાં આવે છે. સબક્લાસ એ વધારાના ઍટ્રિબ્યૂટ અને મેથડ સાથેની વિશિષ્ટ આવૃત્તિ છે.

અમુક સમયે વિવિધ ક્લાસ વચ્ચે ઇનહેરિટન્સનો આદર્શ પદાનુક્રમ હોઈ શકે. ઉદાહરણ તરીકે, 'Person' ક્લાસમાંથી 'Employee' ક્લાસ મેળવી શકાય અને પછી 'Employee' ક્લાસમાંથી 'Teacher' ક્લાસ મેળવી શકાય. અહીં કર્મચારી એ એક વ્યક્તિ છે અને શિક્ષક એ એક કર્મચારી છે. આ પ્રકારના ઇનહેરિટન્સ **મલ્ટિલેવલ ઇનહેરિટન્સ (multilevel inheritance)** કહેવાય છે. આકૃતિ 6.9માં મલ્ટિલેવલ ઇનહેરિટન્સનું ઉદાહરણ આપેલું છે.



આકૃતિ 6.9 : પદાનુક્રમિત ઉત્તરાધિકાર (Multilevel inheritance)

ક્લાસને એક કરતા વધુ પેરન્ટ ક્લાસનો ઉપયોગ કરીને પણ તારવી શકાય છે. ઉદાહરણ તરીકે, બાળક વારસામાં તેના માતા અને પિતા બંનેની લાક્ષણિકતાઓ ધરાવે છે; એરપ્લેન એક પ્રકારનું વાહન છે તથા ઊડી શકે તેવો ઑબ્જેક્ટ પણ છે. જ્યારે કોઈ ક્લાસ બે કે વધુ ક્લાસ પરથી તારવવામાં આવે, તો તેને **મલ્ટિપલ ઇન્હેરિટન્સ (multipal inheritance)** કહેવામાં આવે છે.

કૉમ્પોઝિશન અને ઇનહેરિટન્સની તુલના (Composition Vs Inheritance)

ઇનહેરિટન્સમાં ક્લાસની ક્રિયાત્મકતા (functionality) ભેગી વાપરવા, પુનઃઉપયોગ કરવા અથવા વધારવા માટે અન્ય ક્લાસમાંથી વારસામાં મેળવે છે. અહીં સુપર ક્લાસ અને સબક્લાસ વચ્ચે 'એક પ્રકારનો' ('a kind of') સંબંધ હોય છે.

કૉમ્પોઝિશનમાં ક્લાસ અન્ય ક્લાસમાંથી વારસામાં બનતો નથી, પણ બીજા ક્લાસ વડે 'બનેલો' ('composed of') હોય છે. ક્લાસ અમુક ઍટ્રિબ્યૂટ ધરાવે છે, જેમાંના કેટલાક ઍટ્રિબ્યુટ બીજા પ્રકારના ક્લાસના હોય છે.

અહીં નોંધ કરો કે ક્લાસ-ડાયાગ્રામમાં બીજી અન્ય પ્રકારની રિલેશનશિપ (relationship) અને કન્સ્ટ્રેઇન્ટ (constraints) પણ દર્શાવવામાં આવે છે. આ બધા ખ્યાલોનો અભ્યાસ આ પુસ્તકના કાર્યક્ષેત્રની મર્યાદા બહાર છે.

સારાંશ

સૉફ્ટવેર સિસ્ટમનાં વિશ્લેષણ, ડિઝાઇન અને અમલીકરણમાં ઑબ્જેક્ટ આધારિત પદ્ધતિ ઘણી નોંધપાત્ર ભૂમિકા ભજવે છે. આ ફેરફારમાં **ઑબ્જેક્ટ (object)** કેન્દ્રબિંદુ છે કે જેમાં ડેટા અને ક્રિયાત્મકતા બંનેનો સમાવેશ થયેલો હોય છે. ક્લાસ ઍટ્રિબ્યૂટ (ડેટા) અને મેથડ (બિહેવ્યર અથવા ક્રિયાત્મકતા)ને ભેગી કરીને (encapsulates) માળખા તરીકે (template) બધા ઑબ્જેક્ટ વચ્ચે સહિયારો ઉપયોગ કરે છે. ઑબ્જેક્ટ એકબીજાથી તેના સ્ટેટ (state)થી જુદા પડે છે, એટલે કે તેના ઍટ્રિબ્યૂટની કિંમતોથી એક કરતાં વધારે ક્લાસ એકબીજા સાથે જોડાયેલો હોય છે. જ્યારે બે ક્લાસ વચ્ચે 'પૂર્ણ' અથવા 'આંશિક' સંબંધની સ્થિતિ હોય છે, ત્યારે તેને એગ્રિગેશન અથવા કૉમ્પોઝિશન કહેવાય છે. જ્યારે એક ક્લાસ અન્ય ક્લાસના ઑબ્જેક્ટ ધરાવે, ત્યારે તે ધરાવનાર ક્લાસ ઓનર ક્લાસ (owner class) અથવા હોલક્લાસ (whole class) અથવા એગ્રિગેટિંગ ક્લાસ (aggregating class) કહેવાય છે. ઓનર ક્લાસમાં સમાયેલ ક્લાસ સબ્જેક્ટ ક્લાસ (subject class) અથવા પાર્ટક્લાસ (part class) અથવા એગ્રિગેટેડ ક્લાસ (aggregated class) કહેવાય છે. એગ્રિગેશન બે ક્લાસ વચ્ચે ભિન્ન (non-exclusive) સંબંધની સ્થિતિ રજૂ કરે છે. કૉમ્પોઝિશન બે ક્લાસ વચ્ચે અભિન્ન સંબંધની સ્થિતિ જણાવે છે. જ્યારે બે ક્લાસ વચ્ચે 'એક પ્રકારનો' સંબંધ હોય, ત્યારે તેને ઇનહેરિટન્સની સ્થિતિ કહેવામાં આવે છે. સામાન્ય લાક્ષણિકતાઓને સુપરક્લાસમાં રાખવામાં આવે છે અને વિશિષ્ટ લાક્ષણિકતાઓને સબક્લાસમાં રાખવામાં આવે છે.

સ્વાધ્યાય

1. ઑબ્જેક્ટ આધારિત પ્રોગ્રામિંગ ભાષામાં ઉપલબ્ધ લાક્ષણિકતાઓની યાદી બનાવો.
2. ક્લાસ અને ઑબ્જેક્ટનો તફાવત જણાવો.
3. 'ઇનકેપ્સ્યુલેશન' અને 'ડેટા-ઍબ્સ્ટ્રૅક્શન' વચ્ચેની ભિન્નતા જણાવો.
4. પોલિમૉર્ફિઝમનો અર્થ શો છે ? પોલિમૉર્ફિઝમ મેળવવા માટે વપરાતા બે પ્રકારનાં ઓવરલોડિંગનાં નામ જણાવો.
5. એગ્રિગેશન અને કૉમ્પોઝિશનનો ઉપયોગ સમજાવો.
6. ઇનહેરિટન્સ ક્યારે વાપરવું જોઈએ ? ઉદાહરણ આપો.
7. વિવિધ પ્રકારના ઇનહેરિટન્સ સમજાવો.
8. **નીચેના પ્રશ્નોનો સાચો જવાબ પસંદ કરો :**

   (1) ઑબ્જેક્ટ આધારિત પદ્ધતિમાં કઈ વસ્તુ કેન્દ્રબિંદુ ઉપર હોય છે ?

   (a) ડેટા (b) વિધેય

   (c) ઑબ્જેક્ટ (d) ઉપરના બધા વિકલ્પ

(2) જાવા માટે નીચેનામાંથી શું વધારે યોગ્ય છે ?

(a) પ્રક્રિયાગત પ્રોગ્રામિંગ ભાષા

(b) ઑબ્જેક્ટ આધારિત પ્રોગ્રામિંગ ભાષા

(c) ક્વેરી-લૅંગ્વેજ

(d) ઉપરના બધા વિકલ્પ

(3) નીચેનામાંથી શું ઑબ્જેક્ટને એકબીજાથી જુદા પાડે છે ?

(a) એટ્રિબ્યૂટ (b) સ્ટેટ

(c) બિહેવ્યર (d) ઉપરના બધા વિકલ્પ

(4) એકસમાન ઑબ્જેક્ટની સામાન્ય લાક્ષણિકતાઓને વ્યાખ્યાયિત કરવા માટે નીચેનામાંથી શું વપરાય છે ?

(a) ક્લાસ (b) ઑબ્જેક્ટ

(c) મેથડ (d) ઉપરના બધા વિકલ્પ

(5) નીચેનામાંથી ક્યું દૃશ્યતાનું ચિહ્ન નથી ?

(a) ~ (b) * (c) # (d) -

(6) ઇનકૅપ્સ્યુલેશન વડે નીચેનામાંથી શું પૂરું પાડવામાં આવે છે ?

(a) ડેટાની સુરક્ષિતતા (b) ડેટાનો સહિયારો ઉપયોગ

(c) ડેટા અને મેથડ છૂટા પાડવા (d) આપેલ બધા વિકલ્પ

(7) ડેટા-ઍબ્સ્ટ્રૅક્શન વડે શું શક્ય બનાવી શકાય છે ?

(a) ડેટા-સુરક્ષિતતા (b) ડેટા છુપાવવો

(c) ડેટાની ગણતરી કરવા બાબતની માહિતીનું અમલીકરણ છુપાવવું

(d) ઉપરના બધા વિકલ્પ

(8) નીચેનામાંથી ક્યો વિકલ્પ પોલિમૉર્ફિઝમ વડે પ્રાપ્ત થતો નથી ?

(a) મેથડ ઓવરલોડિંગ (b) ઑપરેટર ઓવરલોડિંગ

(c) ડેટા-હાઇડિંગ (d) આપેલ બધા વિકલ્પ

(9) એગ્રિગેશન ક્યા પ્રકારના સંબંધની સ્થિતિ જણાવે છે ?

(a) 'પૂર્ણ' સંબંધ ('is-a' relationship) (b) સમાન સંબંધ ('is-like' relationship)

(c) અંશતઃ સંબંધ (a-part-of' relationship) (d) ઉપરના બધા વિકલ્પ

(10) ઇનહેરિટન્સ ક્યા પ્રકારના સંબંધની સ્થિતિ જણાવે છે ?

(a) 'પૂર્ણ' સંબંધ ('is-a' relationship) (b) 'એક છે' સંબંધ ('has-a' relationship)

(c) અંશતઃ સંબંધ ('a-part-of' relationship) (d) ઉપરના બધા વિકલ્પ

(11) ક્લાસ-ડાયાગ્રામમાં કૉમ્પોઝિશનને ક્યા ચિહ્ન વડે દર્શાવવામાં આવે છે ?

(a) ખાલી હીરાના ચિહ્ન વડે (b) ભરેલ હીરાના ચિહ્ન વડે

(c) ખાલી ત્રિકોણ ચિહ્ન વડે (d) ઉપરના બધા વિકલ્પ