C++ in Hindi

BccFalna.com
097994-55505

Kuldeep Chand

C with Class is C++. It means, without understanding Object Oriented Programming System (OOPS) Concepts properly, no one can understand C++ and any other Modern Programming Language like Java, C#, VB.Net, etc…

So, In this EBook, I have covered each and every concept related to OOPS and I have tried to Implement each OOPS Concept with easy to understand Demo C++ Program.

In this EBook, I have defined more then 350 Programs and hundreds of Code Fragement to clear each and every C++ and OOPS Concept.

So, This EBook would not only be easy to learn OOPS and C++ but also very useful if you are interested in learning Java or .NET Language like C# or VB.NET.

C++

In Hindi

□ □ □
Kuldeep Chand

BetaLab Computer Center
Falna

C++ Programming Language in Hindi

Lead Editors: **Kuldeep Chand**

Distributed to the book trade worldwide by Betalab Computer Center, Behind of Vidhya Jyoti School, Falna Station Dist. Pali (Raj.) Pin 306116

e-mail bccfalna@gmail.com,

or

visit http://www.bccfalna.com

For information on translations, please contact Betalab Computer Center, Behind of Vidhya Jyoti School, Falna Station Dist. Pali (Raj.) Pin 306116

Phone **097994-55505**

This book is dedicated to those

who really wants to be

a

PROFESSIONAL DEVELOPER

INDEX OF CONTENTS

Table of Contents

OOPS

&

C++

OOPS and C++

सबसे पहला सवाल तो यही है कि C++ क्यों सीखा जाए? ये आज की प्रभावशाली भाषा है। जब Programmers को बडे व जटिल प्रोग्राम बनाने होते हैं, तब Professional Programmers C++ को Choose करते हैं। कई और भी सरल व प्रभावी भाषाएं हैं, लेकिन उनकी कुछ कमियों की वजह से उनमें प्रोग्राम Development की एक सीमा है।

जैसे Visual Basic Microsoft Company की एक बहुत ही सरल भाषा है, लेकिन जब प्रोग्राम बहुत ही बडे व जटिल होते हैं, तब इस भाषा में Program Develop करना समझदारी की बात नहीं होती है। क्योंकि इस भाषा में Graphics का बहुत प्रयोग होता है, और भी कई कारण हैं, जिससे यदि इस भाषा में बडे प्रोग्राम Develop किए जाएं तो प्रोग्राम की Speed बहुत कम हो जाती है। Assembly भाषा भी काफी अच्छी है। इसमें लिखे गए प्रोग्रामों की गति काफी अच्छी होती है, लेकिन ये भाषा Hardware के Device Drivers के प्रोग्राम लिखने के लिये ज्यादा अच्छी है ना कि Application प्रोग्राम लिखने के।

इसी तरह Java Internet के लिये अच्छी है, हालांकि Java C++ से ही प्रेरित है। लेकिन एक बडे Standalone Application Development करने के लिये C++ सबसे लोकप्रिय भाषा है। ये एक बहुत ही Flexible व Best Performing Language है।

Procedural Languages

Pascal, C, Basic, Fortran जैसी पारम्परिक भाषाएं Procedural Languages के उदाहरण हैं। जिसमें प्रत्येक Statement Computer को कुछ काम करने का आदेश देता है। यानी Procedural Languages Instructions का एक समूह होता है। Procedural Languages में छोटे Programs के लिये किसी भी अन्य प्रकार के Pattern की आवश्यकता नही होती है। Programmer Instructions की List बनाता है और Computer उनके अनुसार काम करता है।

जब प्रोग्राम काफी बडे व जटिल हो जाते हैं, तब Instructions की यह List काफी परेशानी पैदा करती है। इसलिये एक बडे प्रोग्राम को छोटे–छोटे टुकडों में बांट दिया जाता है। इन छोटे–छोटे टुकडों को Functions कहा जाता है। Functions को दूसरी अन्य भाषाओं में Subroutine, Subprogram या Procedure कहा जाता है।

एक बडे प्रोग्राम को छोटे–छोटे Functions में विभाजित करने से पूरा Program Functions का एक समूह बन जाता है, जिसे **Module** कहा जाता है।

लेकिन ये Modules भी Procedural Programming के अन्तर्गत ही आते हैं क्योंकि सभी Functions में Statements की एक List होती है और सभी Functions मिल कर पूरा Program बनाते हैं, जिससे पूरा Program Instructions की एक बहुत बडी List बन जाता है।

Procedural Languages के शुरूआती दौर में इनमें ही Program Develop किए जाते थे। "C" भी एक Procedural Languages है और जब "C" भाषा का आविष्कार हुआ था, तब Programmers अन्य भाषाओं को छोड कर "C" में ही अपने Program Develop करने लगे थे।

लेकिन समय व आवश्यकता के अनुसार जब Program बडे व जटिल होने लगे, तब Programmers को इस भाषा में प्रोग्राम बनाने में दिक्कतें आने लगीं। उन्होने महसूस किया कि इस भाषा में कुछ सुधार की आवश्यकता है ताकि ये भाषा सरल व लोकप्रिय बन सके।

ये भाषा सरल बन सके इसके लिये इसका वास्तविक जीवन के अनुसार होना जरूरी था। यानी हम हमारे सामान्य जीवन में जिस प्रकार से व्यवहार करते हैं, इस भाषा का भी वैसा ही होना जरूरी था ताकि Programmers इसमें अधिक सरलता व सफलता से Program बना सकें।

भाषा वास्तविक जीवन के अनुसार हो, यही Concept Object Oriented Programming यानी OOPS का आधार बना। "C" भाषा की इन कमियों को पहचाना गया और इसमें सुधार किया गया। फलस्वरूप हमें "C" भाषा का एक नया संस्करण "C++" प्राप्त हुआ। आइयें, हम भी जानने की कोशिश करते हैं कि "C" भाषा में ऐसी कौनसी कमियां थीं, जिनमें सुधार की आवश्यकता महसूस की गई ?

Procedural Languages में काम होने का महत्व था Data का नहीं, यानी कि Keyboard से Data Input किया जाए, Data पर Processing की जाए, Errors को Check किया जाए आदि। Functions में भी इसी महत्व को जारी रखा गया। Functions कोई काम करते हैं, उसी प्रकार से जिस प्रकार से साधारण Statement करता है। Functions कोई जटिल काम भी कर सकते हैं, लेकिन इनमें भी काम के होने का ही महत्व था।

पूरे Program में Data पर कोई ध्यान नहीं दिया जाता था, जबकि पूरे प्रोग्राम का मूल आधार Data ही होता है। किसी Inventory के Program में इस बात का कोई ज्यादा महत्व नहीं होता है कि Data को किस प्रकार से Display किया जाता है या एक Function किस प्रकार से Corrupt Data को Check करता है, बल्कि इस बात का होता है कि Data क्या है और वह किस प्रकार से Program में काम आ रहा है। Procedural Program में Data को द्वितीय स्तर पर रखा गया था जबकि किसी भी Program का मूल आधार Data ही होता है।

उदाहरण के लिये, किसी Inventory के Program में किसी Data File को Memory में Load किया जाता है, तब ये File एक Global Variable की तरह होती है, जिसे कोई भी Function Use कर सकता है। ये Functions Data पर विभिन्न प्रकार के Operations करते हैं। यानी ये Data को Read करते है, Analyze करते हैं, Update करते हैं, Rearrange करते हैं, Display करते हैं और वापस Disk पर Write करते हैं। "C" में Local Variables भी होते हैं लेकिन Local Variables, महत्वपूर्ण Data के लिये इतने उपयोगी नहीं होते हैं, जो कि विभिन्न Functions द्वारा Access किए जाते हैं।

मान लो कि एक नए Programmer को Data को किसी खास तरीके से Analyze करने के लिये एक Function लिखने को कहा गया। प्रोग्राम की गूढता से अनभिज्ञ Programmer एक ऐसा Function बनाता है जो कि अचानक किसी महत्वपूर्ण Data को नष्ट कर देता है। ऐसा होना काफी आसान है क्योंकि कोई भी Function Data को Access कर सकता है, इसलिये क्योंकि Procedural Language में Data Global होता है। ये कुछ ऐसा ही है जैसे कि आप अपने Personal कागजात को Telephone Directory के पास रख दें जहां कभी भी कोई भी पहुंच सकता है, उससे छेडछाड कर सकता है और उसे नष्ट कर सकता है। इसी प्रकार से Procedural Languages में होता है जहां आपका Data Global होता है और कोई भी Function उसे Use करके खराब कर सकता है या नुकसान पहुंचा सकता है।

Procedural Languages की दूसरी कमी ये थी कि कई Functions एक साथ एक ही Data को Use कर रहे होते हैं, इसलिये Data को Store करने का तरीका काफी जटिल हो जाता है। समान Data को Use कर रहे सभी Functions को Modify किए बिना Data में किसी प्रकार का कोई परिवर्तन नहीं किया जा सकता है। उदाहरण के लिये यदि आप एक नया Data Add करते हैं तो उन सभी Functions को Modify करना होगा जो कि Data को use कर रहे हैं ताकि ये सभी Functions Add किए गए नए Data को Use कर सकें। ये पता करना कि कौन–कौन से Function Data को Use कर रहे हैं और सभी को बिल्कुल सही तरीके से Modify करना काफी कठिन होता है।

Procedural Programs को Design करना काफी मुश्किल होता है। समस्या ये होती है कि इनका Design वास्तविक जीवन से Related नहीं होता है। जैसे कि, माना आप एक Graphics User Interface में Menus Window के लिये Code लिखना चाहते हैं, तो आपको ये तय करना मुश्किल होगा कि कौनसा Function Use किया जाए? कौनसा Data Structure Use किया जाए? आदि। इनका कोई स्पष्ठ उत्तर नहीं है।

Procedural Programs के साथ कई और परेशानियां हैं। उनमें से एक समस्या नए Data Type की है। Computer Languages में कई प्रकार के Built-in Data Types होते हैं, जैसे कि Integer, Float, Character आदि। मानलो कि आप Complex Numbers के साथ प्रक्रिया करना चाहते हैं या Two-dimensional Coordinates के साथ काम करना चाहते हैं या Date के साथ प्रक्रिया करना चाहते हैं।

Built-in Data Type इनको आसानी से Handle नहीं कर सकते हैं। इसलिए हमें हमारी आवश्यकतानुसार स्वयं के Data Type बनाने की जरूरत होती है। यानी Real World Objects को Represent करने के लिए हमें एक ऐसे तरीके की जरूरत होती है, जिससे आसानी से Real World Objects को Computer में Represent किया जा सके और जिस तरह से Real World में विभिन्न Objects आपस में Interaction करके किसी समस्या का एक उचित समाधान प्राप्त करते हैं, उसी तरह से Computer में भी किसी समस्या का समाधान प्राप्त किया जा सके।

Procedural Language में स्वयं के Data Type बना कर हम उन्हे बिल्कुल Built-in Data Type की तरह Use नहीं कर सकते हैं। Procedural Language इतने उन्नत नहीं हैं। जटिल तरीकों को अपनाए बिना हम Procedural Languages में x व y दोनों Coordinates को एक ही Variable में Store करके उस पर Processing नहीं कर सकते हैं। Procedural Languages को लिखना व Maintain करना काफी मुश्किल काम होता है।

सारांश

- कामों को करने का महत्व था, Data का नहीं। यानी Primary Attention Problem को Functions में विभाजित करने का था Data की Security का नहीं। **(Algorithm)**
- बडे Program को छोटे Programs में विभाजित किया जाता था, जिन्हें Functions कहते हैं। ज्यादातर Functions Global Data को Share करते थे, जिससे Data Insecure हो जाता था।
- Application में Data का काफी आसानी से विभिन्न Functions के बीच Transaction होता था। विभिन्न Functions Data को विभिन्न Forms में Transform करते थे।
- Top-Down Approach को Support करते थे।

The Object-Oriented Approach

Object Oriented Language के पीछे का मूलभूत विचार ये है कि एक Program में Data और उस Data पर काम करने वाले Functions को Combine करके एक Unit के रूप में ले लिया जाए। इस Unit को **Object** कहा जाता है। एक Object के Function यानी Data व Data पर काम करने के लिये लिखे गए Function को "C++" में **Member Function** कहा जाता है।

क्योंकि ये किसी Object के किसी अमुक Class से सम्बंधित होते हैं, जो कि किसी Data को Access करने का एक मात्र माध्यम होते हैं। यदि आप किसी Object के अन्दर रखे किसी Data को Read करना चाहते हैं, तो आपको इसी Object के अन्दर लिखे उस Member Function को Call करना पडता है, जिसे उस Object के Data को Use करने के लिये ही लिखा गया है। यही एक Function होता है, जिसकी मदद से आप उस Object के Data को Read कर सकते हैं। आप सीधे ही Data के साथ किसी प्रकार की प्रक्रिया नहीं कर सकते हैं, क्योंकि Data Hidden रहता है। इसलिये किसी प्रकार से अचानक हुए परिवर्तन से Data सुरक्षित रहता है।

Data व Data को Use कर सकने वाले Function का एक साथ एक ही Unit के रूप में होना **Encapsulation** कहलाता है। Data का Hidden रहना यानी **Data Hiding** व **Encapsulation** Object Oriented Programming का मूल तथ्य या Key terms है। यदि आप किसी Data को Modify करना चाहते हैं, तो आपको पता होना चाहिए कि कौनसा Function उस Data पर काम करेगा यानी Object का वह Member Function जिसे Data के साथ लिखा गया है। कोई भी अन्य Function उस Data को Access नहीं कर सकता है। ये Processing Program को लिखना, Debug करना व Maintain करना आसान बनाती है। एक "C++" का

प्रोग्राम ढेर सारे विभिन्न प्रकार के Objects का बना होता है, जो कि अपने–अपने Member Functions के द्वारा आपस में Communication करते हैं। "C++" व कई अन्य OOP Languages में Member Functions को **Methods** कहा जाता है और Data Item को **Instance Variable** कहा जाता है। किसी Object के Member Function को Call करना उस Object को **Message Send** करना कहलाता है।

हम एक उदाहरण लेते हैं। माना एक बडा प्रीति–भोज (Party) का समारोह है, जिसमें सभी मेहमान किसी Dining Table के चारों ओर बैठे हैं। किसी Table के चारों ओर बैठे लोगों को हम Functions मान सकते हैं जो कि खाना खाने का काम करते हैं और जो भी खाना Table पर रखा है, उसे Data कह सकते हैं। जब भी किसी को Table पर रखे विभिन्न प्रकार के व्यंजनों में से कुछ लेना होता है, तो वह स्वयं ही उस व्यंजन तक पहुंचता है और उसे उपयोग में ले लेता है किसी पडौसी मेहमान से कोई भी व्यंजन Pass करने को नहीं कहता। Procedural Program का भी यही तरीका होता है।

ये तरीका तब तक बहुत ठीक है जब तक कि खाना खाने वाले मेहमानों की संख्या सीमित हो यानी छः से आठ लोग अलग–अलग समूह में अलग–अलग Table पर खाना खा रहे हैं। लेकिन यदि मेहमानों की संख्या बीस या इससे भी अधिक हो तो ये तरीका ठीक नहीं कहा जा सकता है।

क्योंकि जब मेहमान अधिक होंगे तो Table भी बडा होगा और खाने के विभिन्न सामान पूरे Table पर काफी दूर–दूर होंगे। ऐसे में यदि कोई मेहमान किसी दूर रखे व्यंजन तक पहुंचना चाहता है तो हो सकता है कि उसके Shirt की Sleeves किसी दूसरे मेहमान के खाने में चली जाए या कई मेहमान एक साथ किसी व्यंजन पर हाथ बढाएं और व्यंजन Table पर गिर कर खराब हो जाए। यानी यदि मेहमानों की संख्या काफी ज्यादा हो तो एक ही Table पर भोजन करना एक परेशानी वाला काम होगा। एक बडे Procedural Program में भी यही होता है।

इस समस्या के समाधान के रूप में यदि कई छोटे–छोटे Tables हों और उन पर एक सीमित मात्रा में मेहमान हों और सबके पास उनका अपना भोजन हो तो ये एक अच्छी व्यवस्था हो सकती है। इस छोटे Table पर सभी मेहमान किसी भी व्यंजन पर आसानी से पहुंच सकते हैं और किसी प्रकार की परेशानी Create नहीं होती है। यदि कोई मेहमान किसी अन्य Table पर रखे किसी व्यंजन को लेना चाहता है तो सम्भवतया वह किसी अन्य मेहमान से उस व्यंजन को लाने के लिये कह सकता है। ये तरीका Object Oriented Programming का है जिसमें हरेक छोटी Table को एक Object कहा जा सकता है।

हरेक Object में उसका स्वयं का Data और Function होता है उसी प्रकार से जिस प्रकार से हरेक Table पर अलग – अलग मेहमान होते हैं और हरेक Table पर अपना अलग खाना होता है। Data व Function के बीच होने वाले विभिन्न लेन–देन अधिकतर Object के अन्दर ही होते हैं लेकिन आवश्यकतानुसार ये भी सम्भव है कि किसी अन्य Object के Data को भी Use किया जा सके। इस तरह से किसी बडे Procedural Program को छोटे–छोटे Object के रूप में व्यवस्थित करके ज्यादा अच्छी तरह से Program को Maintain किया जा सकता है।

Features of Object-Oriented Languages

Objects

जब आप किसी समस्या को Object Oriented Language के रूप में बनाना चाहते हैं, तब ये तय नहीं करना होता है कि समस्या को Functions में किस प्रकार से विभाजित किया जाए बल्कि ये तय करना होता है कि समस्या को Objects में किस प्रकार से विभाजित किया जाए। साधारण सा सवाल दिमाग में आ सकता है कि ये Objects क्या होते हैं? इसका जवाब भी इतना ही साधारण है। हम जो कुछ भी सोंच सकते हैं, दुनिया की वह हर वस्तु Object है। फिर भी थोडा सा समझाने के उद्देश्य से हम यहां पर Objects के कुछ उदाहरण दे रहे हैं:

1 Physical Objects

- ❖ किसी Lift से सम्बंधित प्रोग्राम जिसमें Program का मूल बिन्दु Lift पर निर्भर करता है, इसमें Lift को एक Object कहा जा सकता है।
- ❖ किसी अर्थव्यवस्था से सम्बंधित प्रोग्राम में विश्व के सभी देशों को Object माना जा सकता है। क्योंकि इस प्रोग्राम का मूल बिन्दु विभिन्न देश होंगे, जिनकी अर्थव्यवस्था पर सारा प्रोग्राम निर्भर होगा।
- ❖ किसी Traffic Flow से सम्बंधित प्रोग्राम में विभिन्न प्रकार के वाहन Objects हो सकते हैं, क्योंकि पूरा प्रोग्राम वाहनों को केन्द्र में रख कर ही Develop किया जाएगा।
- ❖ किसी Air Traffic से सम्बंधित प्रोग्राम में विभिन्न देशों के Aircraft Object होंगे।

2 किसी Computer User Environment में कम्प्यूटर के विभिन्न अवयव जैसे कि Window, Menu, Graphics Objects (Line, Rectangle, Circle) Mouse, Keyboard, Toolbars, Command Buttons, Disks Drives, Printer आदि Objects होते हैं।

3 Human Entities में

- ❖ किसी Company के Employees
- ❖ किसी विद्यालय के विभिन्न विद्यार्थी
- ❖ विभिन्न ग्राहक
- ❖ Salesmen आदि Objects होते हैं, क्योंकि ये ही किसी प्रोग्राम की मूल इकाई होते हैं।

4 Data Storage Construct में

- ❖ विभिन्न `Customized Arrays`
- ❖ Arrays
- ❖ Stacks
- ❖ Linked Lists

- ❖ Binary Trees आदि Objects होते हैं।

5 Collection Of Data में

- ❖ Inventory
- ❖ Personal File
- ❖ Dictionary
- ❖ Longitude व Latitude की Table

6 User Defined Data Types में Time, Complex Numbers, Points Of Planes आदि Objects होंगे।

7 Computer Game में

- ❖ कोई चित्र
- ❖ Chess या Checkers के मोहरे आदि
- ❖ जानवरों के चिन्ह
- ❖ विभिन्न चिन्ह आदि Objects हो सकते हैं।

इन उदाहरणों से हम समझ सकते हैं कि किसी Program के किसी हिस्से में या पूरे प्रोग्राम में जो मूल वस्तु होती है, वह **Object** कही जा सकती है। किसी भी समस्या को Object के रूप में विभाजित करना काफी आसान है। क्योंकि दुनिया की हर वस्तु को जरूरत के अनुसार एक Object माना जा सकता है। हरेक Object की अपनी क्षमताएं होती हैं और हरेक Object कुछ ना कुछ काम कर सकता है। इसे समझने के लिये एक उदाहरण देखते हैं।

बडी इमारतों में विभिन्न मंजिलों पर आने–जाने के लिये Lifts होती हैं। इस Lift को एक Object माना जा सकता है। माना कि चौथी मंजिल पर किसी Lift में चार Passengers हैं और Passengers ने 8^{th}, 10^{th} व 15^{th} मंजिल पर जाने के लिये Button Press किया है, तो Lift में ये क्षमता होती है कि ये नीचे जा सकती है, ऊपर जा सकती है, ये इसके दरवाजों को खोल सकती है व बन्द कर सकती है, ये पता कर सकती है कि दूसरी Lifts कौनसी मंजिल पर हैं और उसे अगली किस मंजिल पर जाना है।

"C++" में एक Object के Data ये ध्यान रखते हैं कि Object में क्या–क्या क्षमताएं हैं और Object के Member Functions इस बात का ध्यान रखते हैं कि Object उन Data के साथ क्या–क्या कर सकता है। इस Lift Object में निम्न Data हो सकते हैं:

- Current_floor_number
- Number_of_passengers
- List_of_buttons_pussed

और Member Functions निम्न हो सकते हैं:

- GoDown()
- GoUp()
- OpenDoors()
- CloseDoors()
- GetInfo()
- CalculateWhereToGo()

Object Oriented Programming में किसी वस्तु की **विशेषता** व वस्तु की **क्षमता** एक साथ में होती है, ठीक उसी प्रकार से Object Oriented Program में Data व Functions एक साथ में एक Unit के रूप में होते हैं, जिसे **Object** कहा जाता है। Object के Data व उन Data की ***State*** में परिवर्तन करने वाले Functions को एक **Unit** या इकाई के रूप में Combine करने की प्रक्रिया को **Encapsulation** कहते हैं। Encapsulation की प्रक्रिया से प्राप्त होने वाला Template या Description **Class** कहलाता है।

Objects किसी Object Oriented System की Basic Run Time Entities होते हैं। Objects Memory में Space लेते हैं और हर Object का एक Memory Address होता है। जब हम किसी Object Oriented Program को Execute करते हैं, तब विभिन्न Objects एक दूसरे को Message Pass करके यानी उनके Member Functions को Call करके एक दूसरे से Communication करते हैं।

उदाहरण के लिए मान लो कि "Customer" व "Account" दो Objects हैं। अब यदि Customer को अपना Bank Balance जानना हो, तो वह Account Object को एक Message Pass करता है और अपने Bank Balance की जानकारी प्राप्त करने के लिए Request करता है। Account Object Customer Object की Request को पूरा करता है और उसे Bank Balance की जानकारी प्रदान कर देता है। इस तरह से Customer Object बिना ये जाने हुए कि Account Object में कौन–कौन से Data हैं, अपने Balance की जानकारी प्राप्त कर सकता है।

अपना Balance जानने के लिए Customer Object को केवल इतना ही ध्यान रखना होता है कि Account Object को कौनसा Message Pass करने पर यानी Account Object के किस Member Function को Call करने पर Account Object Account Balance की जानकारी प्रदान करेगा।

Classes

जब हमें एक ही Data Type के कई Data को किसी Variable में Store करना होता है, तब हम Array का प्रयोग करते हैं और जब विभिन्न Data Type के आपस में Logically Related कई

Data को एक ही Variable में रखना होता है तब हम Structure का प्रयोग करते हैं। "C++" में **Class** भी यही काम करता है। यानी Class में भी हम विभिन्न Data Types के, आपस में Logically Related Variables Declare करते हैं। Structure व Class में मूलभूत अन्तर यही है कि एक Structure के Members Default रूप से **Global** या **Public** होते हैं जबकि एक Class के Members **Local** या **Private** होते हैं।

Class की दूसरी विशेषता ये है कि Class के Data Members को Access करने के Functions भी Class के अन्दर ही लिखे जाते हैं और हम इन्हीं Member Functions की सहायता से किसी Class के Object के Data Members को Access कर सकते हैं।

एक Class में Data छुपा हुआ रहता है क्योंकि केवल उस Class के Objects ही उस Class के Data को Access करने में सक्षम होते हैं। किसी अन्य Class के Objects या किसी External Functions के लिए ये Data Accessible नहीं होते हैं। इस प्रक्रिया को **Data Hiding** कहते हैं।

Data व Data को Access करने के लिए Authorized Function दोनों को एक Unit के रूप में Combined Form में Describe किया जाता है। किसी Object के Data व Data पर Perform होने वाले Operations के Functions को एक Unit के रूप में रखने की प्रक्रिया को **Encapsulation** कहा जाता है। किसी Class के अन्दर Declare किए गए वे Variables जिनमें Object के Data Stored रहते हैं, **Data Members** कहलाते हैं और उन Data के साथ प्रक्रिया करने वाले Functions **Member Functions** कहलाते हैं।

किसी Object के Data व Data पर Perform होने वाले Operations को Encapsulate करके एक Unit में रखने पर Create होने वाले Template या Description को Class कहते हैं। Class एक नए प्रकार का User Defined Data Type होता है। इस Create होने वाली Class का हम जितने चाहें उतने Objects Create कर सकते हैं। यानी एक Class समान प्रकार के Objects के समूह की Description या Template को Represent करता है। Classes User Defined Data Type होती हैं और Built-In Data Type की तरह Behave करती हैं।

जिस प्रकार से किसी Structure को Define करने के बाद उस Structure के प्रकार के Variables Declare किए जाते हैं, उसी प्रकार से किसी Class के भी Variables Declare किए जाते हैं ताकि उस Class के Data को Use किया जा सके। "C++" में Class प्रकार से Declare किए गए Variables को **Object** कहा जाता है। जब कोई Class Define की जाती है, तब वह Memory में कोई Space Reserve नहीं करता है। एक Class प्रकार का Variable जिसे Object कहते हैं, Declare करने के बाद ही वह Memory में अपने Class की क्षमता के अनुसार Space Reserve करता है, ठीक उसी प्रकार से जिस प्रकार से किसी Structure को Define करने पर वह Memory में कोई Space Reserve नहीं करता है। जब हम Structure प्रकार के Variable Declare करते हैं तभी Memory में Space Reserve होती है।

Data Hiding, Data Abstraction and Encapsulation

किसी System की Inner Working को Hide करके उसे उपयोग करने के लिए एक Well Defined Interface तैयार करने की प्रक्रिया को हम ***Encapsulation*** कहते हैं। Create होने वाले Object को Real World में जब Use करना होता है, तब हम उसी Well Defined Interface द्वारा उस Object के Data को Access करते हैं।

Encapsulation का सबसे अच्छा उदाहरण एक Watch का लिया जा सकता है। एक Wristwatch किस प्रकार से काम करता है, Internally उस Wristwatch में क्या प्रक्रिया होती है, Wristwatch चाहे Analog हो या Digital, वह किस प्रकार से Hours, Minutes व Seconds को Manage करता है, यानी Wristwatch की Internal Working से हमें तब तक कोई मतलब नहीं होता है, जब तक कि Wristwatch सही तरीके से काम करती है। हम केवल समय जानने के लिए किसी Wristwatch को Use करते हैं।

Data व Data पर Perform होने वाले Operations के Functions को एक **Unit** के रूप में Bind करके Object की Class बनाने की प्रक्रिया को ***Encapsulation*** कहते हैं। इस Encapsulation के Concept के आधार पर बनने वाली Class के Data को केवल उसी Class में Define किए गए Member Functions ही Access कर सकते हैं।

इन Member Functions के अलावा कोई भी External Function उस Specific Class के Data को Access नहीं कर सकता है। किसी Class के अन्दर Define किए गए Member Function ही Program व Object के Data के बीच ***Interface*** प्रदान करते हैं। बिना इन Member Function के Main Function, किसी अन्य Class में Define किया गया Member Function या कोई भी अन्य User Defined External Function उस Specific Object के Data को Access नहीं कर सकता है। यानी Main Program से Object का Data पूरी तरह से ***Insulated*** या अलग होता है।

इस Insulation के कारण किसी Specific Object का Data किसी अन्य User Defined External Function या Main Program से छुपा हुआ रहता है। इसलिए इस Insulation की प्रक्रिया को ***Data Hiding*** कहते हैं।

Data Abstraction एक ऐसी प्रक्रिया होती है, जिसमें किसी समस्या से सम्बंधित जरूरी बातों को बिना जरूरी बातों से अलग किया जाता है। फिर उन जरूरी बातों को समस्या के किसी Object की Properties के रूप में Describe किया जाता है। उदाहरण के लिए मानलो कि किसी Company के Administrator को अपनी Company के सभी Employees की Educational Information की जानकारी रखनी है। इस स्थिति में किसी Employee की विभिन्न Properties निम्नानुसार हो सकती हैं:

- **Name Of the Employee**
- Father's Name of the Employee

- Address of the Employee
- City of the Employee
- State of the Employee
- Country of the Employee
- Date Of Birth of the Employee
- **Education Of the Employee**
- Hobbies of the Employee
- Experience in the Company of the Employee
- **Extra Degree Of the Employee**
- Cast of the Employee
- Religion of the Employee

किसी Employee की और भी बहुत सी विशेषताएं हो सकती हैं, लेकिन चूंकि समस्या में Company के Administrator को अपने Employees की केवल Educational Information से ही मतलब है, शेष Information का Company के Administrator के लिए कोई उपयोग नहीं है, इसलिए Administrator की समस्या से सम्बंधित जरूरी बातें केवल निम्नानुसार ही हैं:

- **Name Of the Employee**
- **Education Of the Employee**
- **Extra Degree Of the Employee**

इनके अलावा एक Employee की जो भी Information हैं, वे इस समस्या के लिए जरूरी नहीं हैं, हालांकि किसी अन्य समस्या के लिए ये जरूरी हो सकती हैं। इस तरह से हमने एक Employee की विभिन्न विशेषताओं में से उन विशेषताओं को अलग किया जो Company के Administrator के लिए जरूरी हैं या हमारी Current समस्या से सम्बंधित हैं। इस प्रक्रिया को ही ***Abstraction*** कहते हैं, जिसमें हमने Employee की उन जरूरी बातों को उन बिना जरूरी बातों से अलग किया है, जिनकी Company के Administrator को जरूरत है।

किसी अन्य समस्या जैसे कि Employees के Bio Data को Manage करने वाली Class को Create करते समय ये सभी Information जरूरी हो सकती हैं। उस स्थिति में कुछ अन्य बातें होंगी जो Employee के Bio Data से सम्बंधित नहीं होंगी। तब भी हमें Employee के विभिन्न Data पर Abstraction की प्रक्रिया को लागू करके जरूरी Data को बिना जरूरी Data से अलग करना होगा।

जब हम किसी समस्या के समाधान के लिए किसी Object की जरूरी बातों को बिना जरूरी बातों से अलग कर लेते हैं, यानी Object की विभिन्न Properties पर Data Abstraction की Process को लागू करने के बाद जो जरूरी Properties प्राप्त होती हैं, उन Properties को Class के **Attributes** के रूप में Define कर लिया जाता है और इन Attributes या Properties की **State** या स्थिति में परिवर्तन करने वाले Operations के Functions को Attributes के साथ एक **Unit** के रूप में **Encapsulate** कर लिया जाता है, जिससे एक Description बन जाता है। फिर इस

Description को एक नाम दे दिया जाता है, जो कि किसी **Class** को Define करता है, यानी हम एक नई Class बना लेते हैं।

चूंकि हम जानते हैं कि Class OOPS में एक नए User Defined Data Type को Represent करता है और हम जो Class बनाते हैं, उस Class में **Abstraction** प्रक्रिया के बाद प्राप्त हुए Attributes होते हैं, इसलिए इस नए Create होने वाले Data Type को ***Abstract Data Type*** कहते हैं।

Inheritance

जिस प्रकार से हम वास्तविक जीवन में विभिन्न Classes को Subclasses में विभाजित करते हैं, ठीक उसी प्रकार से हम "C++" में भी Classes को विभिन्न Subclasses में विभाजित कर सकते हैं। जैसे जानवरों के समूह को हम पानी में रहने वाले जानवरों, वायु में उडने वाले जानवरों व धरती पर चलने वाले जानवरों में बांट सकते हैं। फिर हरेक जानवर को उसकी विशेषताओं के आधार पर और भागों में विभाजित कर सकते हैं। ठीक इसी प्रकार से हम "C++" में भी Classes का विभाजन कर सकते हैं, जो कि पिछली Classes की विशेषताओं के साथ अपनी भी कुछ अन्य विशेषताओं को उपयोग में लेता हैं। ये प्रक्रिया **OOPS** में ***Inheritance*** कहलाती है।

जो मुख्य Class होती है, उसे ***Base Class*** कहते हैं और इसकी अपनी विशेषताएं होती हैं। इस Class के अन्दर और विभाजन किया जाए तो वह विभाजित Class ***Derived Class*** कहलाती है, जिसमें उसकी Base Class के गुण तो होते ही हैं, साथ ही इसके अपने भी कुछ अलग गुण होते हैं जो कि Base Class में नहीं होते।

Inheritance एक ऐसा तरीका है जिससे एक Class के Objects किसी दूसरी Class के गुणों को प्राप्त कर लेते हैं। ये Hierarchical Classification को Support करता है। OOPS में Inheritance का ये Concept Reusability का Idea Provide करता है। इस Concept के आधार पर एक पहले से बनी हुई Class के गुणों को Derive करके नई Class बनाई जाती है।

Reusability

एक बार एक Class लिख कर तैयार कर लेने के बाद हम उसी Class को कई अन्य प्रोग्रामों में आवश्यकतानुसार उसी प्रकार से Use कर सकते हैं जिस प्रकार से Procedural Languages में Library Functions को **Use** करते हैं। Inheritance की विशेषता को Use करते हुए हम ऐसा भी कर सकते हैं कि किसी पहले से बनाई गई Class में परिवर्तन किये बिना ही हम उसमें एक Derived Class बनाकर Base Class के गुण भी Use कर लें और अपनी Derived Class के गुण भी उसमें जोड सकते हैं।

जैसे कि माना आपने एक Class बनाया जो कि Menu बनाने का काम करता है। आप इस Class में परिवर्तन करना नहीं चाहते हैं, लेकिन आप चाहते हैं कि उसमें Animation की Capability भी आ जाए। ऐसे में आप एक नई Class बना सकते हैं, जिसमें Base Class के सारे गुण तो होंगे ही साथ में आप Animation का गुण भी उसमें जोड सकते हैं। इस तरह से OOPS की वजह से आप किसी Class को बार–बार लिखने से बच जाते हैं और एक ही Class को कई जगह Use कर सकते हैं। ये भी OOPS की एक खास विशेषता है।

Creating New Data Types

Object की एक विशेषता ये भी है कि ये Programmers को आवश्यकतानुसार नए Data Types बनाने की भी सुविधा प्रदान करता है। एक Programmer जो भी Class Create करता है, वह Class Computer में एक नए प्रकार के Data Type को Represent करता है और Class किसी Real World Object की विशेषताओं और क्षमताओं का Description या Specification होता है कि उस Class का Object किस तरह का है और क्या–क्या कर सकता है।

Polymorphism and Overloading

जैसाकि हमने पहले भी बताया कि एक Class "C" के Structure का ही Modified रूप है। यानी हम Structure प्रकार के Variable तो Declare कर सकते हैं, लेकिन जिस प्रकार से Built - In प्रकार के Data Type के दो Variables को हम आपस में जोड सकते हैं, ठीक उसी प्रकार से किसी Structure के दो Variables को हम नहीं जोड सकते। इसे समझने के लिये हम एक उदाहरण देखते हैं। माना एक Structure निम्नानुसार है:

```
struct Add
    {
        int num1;
        int num2;
    };

struct Add A, B, C ;
```

यदि हम `C = A + B ;` करें, तो ये एक गलत Statement होगा। किसी Class के Objects को भी हम ठीक इसी प्रकार से नहीं जोड सकते हैं, क्योंकि Class Structure का ही Modified रूप है।

इसका कारण ये है कि Compiler ये नहीं जानता है कि User द्वारा Define किए गए Variables के साथ किस प्रकार से जोड किया जाए। जबकि Built-in Data Types में जोड करने के इस तरीके

को Compiler में पहले से ही निश्चित कर दिया गया है और + Operator ये जानता है कि इन Variables को कैसे जोडा जाए।

इसलिये User Defined Data Type के Variables या Object को जोडने का तरीका **Operators** को बताना पडता है। यानी + Operators को ये बताना पडता है कि किस प्रकार से User Defined Data Type के Variables या Objects को ये Operator जोडेगा।

इसके लिये हमें नए प्रकार के Operations Define करने होते हैं। इस प्रकार से Operators व Functions को अलग–अलग तरीके से इस बात पर निर्भर करते हुए कि वे किस प्रकार के Data पर Operation कर रहे हैं, Use किया जाता है और इस प्रक्रिया को **Polymorphism** कहा जाता है।

किसी Existing Operator जैसे कि +, = आदि को इस लायक बनाया जाता है कि वे User Defined Data Type के विभिन्न प्रकार के Variables या Objects पर क्रिया कर सकें। विभिन्न Operators को इस लायक बनाने की क्रिया को Operators की **Overloading** करना कहा जाता है।

जब कई Functions के नाम समान होते हैं लेकिन उनके Arguments या Parameters की संख्या या Return Data Type या फिर Formal Arguments के Data Type में आपस में अन्तर होता है, तो इसे Functions की **Overloading** होना कहा जाता है। Overloading से एक Programmer के लिए Program लिखना व समझना आसान हो जाता है। यह भी Polymorphism का एक तरीका है।

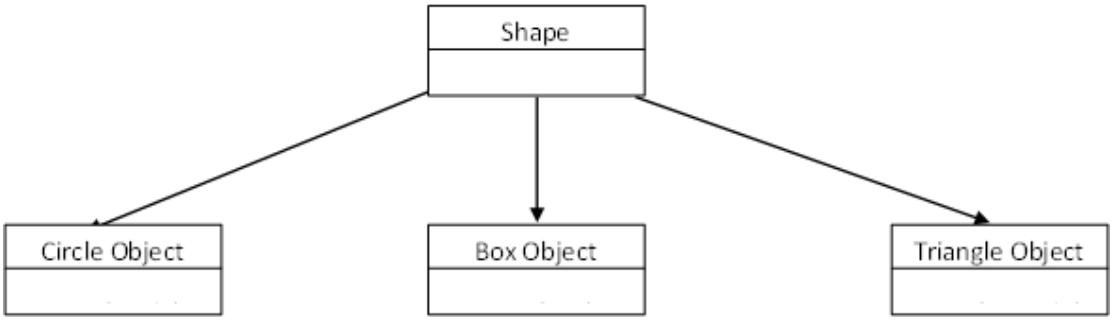
Dynamic Binding

किसी Object के Reference में कौनसा Function Call होना चाहिए, जब ये बात Program के Compile Time में तय होती है, तो इसे **Early Binding** कहते हैं। जबकि किसी Object के Reference में किसी काम के लिए कौनसा Procedure Execute होगा, ये बात जब Program के Runtime में तय होती है, तब इसे **Late Binding** या **Dynamic Binding** कहते हैं। Polymorphism के अन्तर्गत Dynamic Binding का काम होता है। इसे समझने के लिए निम्न चित्र देखिए:

इस चित्र में हम देख सकते हैं कि Shape Class एक Base Class है जिसे Inherit करके तीन नई Classes Circle, Box व Triangle को Create किया गया है। चूंकि ये तीनों ही Classes Shape Class से Inherited हैं, इसलिए Base Class Shape का Draw() Method तीनों ही Classes में Inherited है।

अब मानलो कि हमने तीनों Derived Classes का एक–एक Object Create किया और उस Object को Draw करने के लिए Draw() Method को Call किया। ऐसे में जब हम Circle

Class के Object के लिए Draw Method Call करते हैं, तब Compiler Circle Class के Draw Method को Call करके Circle Draw करता है।



जब हम Box Object Draw करने के लिए Box Class के Object के Reference में Draw() Method को Call करते हैं, तब Compiler Box Class के Draw Method को Execute करता है। इसी तरह से जब हम Triangle Class का Object Create करना चाहते हैं, तब Compiler Triangle Class के Draw Method को Execute करके Triangle Draw कर देता है। यानी एक ही नाम का Draw Method Create हो रहे Object की Class के आधार पर उसी Class के Draw Method को Execute करता है, जिस Class का Object Create किया जा रहा है।

इस प्रक्रिया को Binding Object के साथ Method की Binding होना कहते हैं। चूंकि किस Object के Reference में कौनसा Draw Method Call होगा, इसका निर्णय Compiler Program के Runtime में करता है, इसलिए इस प्रक्रिया को Late Binding या Dynamic Binding कहते हैं।

Message Passing

Object Oriented Program में Objects का पूरा एक समूह होता है। ये Objects आपस में Communication करते हैं। किसी Object Oriented Programming में निम्न तीन Concepts होते हैं:

- Abstract Data के आधार पर एक ऐसी Class Definition Create करना जो Required Object की Properties व उसके Behaviors को Describe करे।
- Create किए गए Abstract Data Type या Class के Objects Create करना।
- Create होने वाले विभिन्न Objects के बीच Communication को Establish करना।

दो Objects आपस में उसी प्रकार से Information भेज कर व प्राप्त करके Communication करते हैं, जिस तरह से Real World में आम लोग आपस में Message Pass करके Communication करते हैं।

Message Passing के Concept से हम Computer Application के रूप में किसी समस्या को उसी प्रकार से Directly Modal कर सकते हैं, जिस तरह से उस समस्या को Real World में Describe करते हैं। Message Passing Concept में जब एक Object A किसी दूसरे Object B से Communication करना चाहता है, तब वह Object A उस दूसरे Object B को एक Message Pass करता है यानी उस दूसरे Object B के किसी Member Function को Call करता है। इस Concept में तीन बातें होती हैं:

- **Object**
- **Message** (Member Function of the object)
- **Information** (Arguments in the Member Function of the object)

उदाहरण के लिए मानलो कि *Customer* Object *Account* Object से अपने Bank Balance की जानकारी प्राप्त करना चाहता है। ऐसे में Customer Object Account Object को एक **Message** भेजता है (यानी Account Object के Reference में किसी **Member Function** को Call करता है।) और Message में ***Bank Balance*** की जानकारी प्राप्त करने के लिए Information (***Arguments***) भेजता है। यानी Customer Object Account Object से निम्नानुसार Communication करता है:

```
Account.Information(Bank_Balance);
```

जब हम इस प्रकार से Account Object से Bank Balance Return करवाते हैं, तो उस Bank Balance को Customer Object के लिए Use कर सकते हैं। इस प्रकार से एक Object (Customer) दूसरे Object (Account) से Communication करने के लिए दूसरे Object के किसी Appropriate Member Function को Call करते हुए उसे **Message** भेजता है, तथा किसी जानकारी के लिए **Arguments** के रूप में Request करता है। दूसरा Object उस Request को Message द्वारा प्राप्त करता है और Appropriate Member Function के Execution द्वारा पहले Object को उसकी Required Information प्रदान कर देता है।

OOPS एक ऐसा Concept है, जिसके आधार पर हम किसी समस्या को Design करते समय उसे विभिन्न प्रकार के Physical Objects के रूप में परिभाषित करते हैं और इन विभिन्न प्रकार के Physical Objects को Computer में किसी भी Programming Language में Logically Represent कर सकते हैं। विभिन्न प्रकार के Objects को जिस Programming Language में अच्छी तरह से Represent किया जा सकता है, उस Programming Language को हम **Object Oriented Programming Language** कह सकते हैं।

"C++" एक ऐसी ही Programming Language है, जो OOPS के सभी Concept को Computer में Implement करने में सक्षम है। यदि हम Object को सरल रूप में परिभाषित करें, तो हम ये कह

सकते हैं कि Object एक ऐसा Variable होता है, जिसमें एक से ज्यादा प्रकार के "**Basic Data Type"** के मानों को Store व Manipulate किया जाता है।

ये तय करने के लिए कि Object किस प्रकार के और कितने मानों को Manipulate करने का काम करेगा या Object किस Physical Real World Object को Logically Computer में Represent करेगा, हमें एक Specification (Modal) बनाना होता है। ये Specification या Modal जो कि Object की विशेषताओं या **Attributes** (Data Members) और उन विशेषताओं (Data Members की States) में परिवर्तन करने वाले Object के **Behaviors** (Member Functions) को Represent करता है।

इस Specification को ही **Class** कहते हैं, जो कि एक तरफ तो किसी Real World Object को Computer में Logically Represent कर रहा होता है और दूसरी तरफ वही Class किसी Object का एक ऐसा Specification होता है, जो Programming Language में उस Real World Object को Logically एक नए Data Type के रूप में Represent कर रहा होता है।

सारांश

- समस्या में Procedures के बजाए Data का महत्व है। यानी OOPS में Data को Primary Level पर रखा गया है और Data पर Perform होने वाले Operations के Functions को Secondary Level पर रखा गया है।
- Problem को Functions में विभाजित करने के बजाय Objects में विभाजित किया जाता है।
- Data Structure को इस प्रकार से Design किया गया है, जो कि Object को Characterize करते हैं।
- Data पर Perform होने वाले Operations वाले Functions को Object के Data Structure के साथ ही Combined कर दिया गया है, जिसे Encapsulation कहते हैं।
- Data को केवल Data के साथ Associate किए गए Functions ही Access कर सकते हैं, जिससे Data External Functions के लिए Hidden रहता है। इस प्रक्रिया को OOPS में Data Hiding कहते हैं।
- Objects आपस में Functions द्वारा Communication करते हैं। इस प्रक्रिया को Message Passing करना कहते हैं।
- आवश्यकता होने पर Object में नए Data व Data पर Perform होने वाले Operations को Add किया जा सकता है। इस प्रक्रिया को OOPS में Inheritance कहते हैं।
- Program Design में OOPS के Approach को Bottom-Up Approach कहते हैं।

Benefits Of OOPS

OOPS Program Designer व Program User दोनों के लिए कई सुविधाएं प्रदान करता है। OOPS द्वारा Develop किए जाने वाले Programs में निम्न विशेषताएं होती हैं:

- Inheritance का प्रयोग करके एक Programmer बार–बार एक जैसी Coding लिखने से बच जाता है। वह एक बार लिखी गई Coding को बार–बार Reuse कर पाने में सक्षम हो जाता है। Coding को Reuse करने के कारण Programmer को Program Develop करने में कम समय लगता है और Program को Maintain करना सरल हो जाता है।
- चूंकि, OOPS में Program का Data पूरी तरह से Outer World से Hide रहता है। Data को Access करने के लिए अधिकृत Member Functions ही उस Data को Access करने में सक्षम होते हैं। चूंकि, Program के Data को कोई भी Unauthorized External Function Access करने में सक्षम नहीं होता है, इसलिए Accidental Modifications से Data सुरक्षित रहता है।
- एक ही Object के कई Instances बिना किसी Interference के एक ही Program में एक साथ Exist हो सकते हैं।
- Program को Objects में विभाजित करने के कारण Program Real World Concepts को Logically ज्यादा बेहतर तरीके से Computer में Represent करने में सक्षम होता है।
- OOPS पर आधारित Application को Upgrade, Modify व Change करना काफी सरल व सुविधाजनक होता है।
- बहुत ही जटिल समस्याओं के समाधान को भी OOPS Concept के आधार पर काफी आसान तरीके से Develop व Manage किया जा सकता है।

Object Oriented Languages

OOPS के Concepts को “C” या “Pascal” जैसी Procedural Languages में भी पूरी तरह से Implement किया जा सकता है। लेकिन जब हम “C” जैसी Procedural Language में OOPS के Concepts को Implement करने की कोशिश करते हैं, तब हमें कई अन्य Compiler सम्बंधित सीमाओं का सामना करना पडता है।

जबकि “C++” जैसी OOPS को ध्यान में रख कर Design किए गए Compiler को Use करने पर हमें इस प्रकार की समस्याओं का सामना नहीं करना पडता। OOPS को Implement करने के सम्बंध में भी हम Programming Languages को दो भागों में बांट सकते हैं:

Object-Based Programming Language

Microsoft Company का Visual – Basic एक Object Based Programming Language है। इस प्रकार की Programming Languages Encapsulations व Object Identity को Support करता है। Object Based Programming Languages के मुख्य Features Encapsulation, Data Hiding व Access Mechanism, Objects का Automatically Initialize व Clear होना तथा Operator Overloading होते हैं। Object-Based Programming Languages Inheritance व Dynamic Binding को Support नहीं करते हैं। वे Languages जो Objects को Use करते हुए Programming सम्भव बनाती हैं, Object – Based Programming Languages कहलाती हैं।

Object-Oriented Programming Language

Object Oriented Programming में Object Based Programming के सभी Features Available होने के साथ ही इनमें Inheritance व Dynamic Binding की भी सुविधा होती है। "C++" एक Hybrid Language है, क्योंकि ये Procedural Programming के साथ–साथ Object Oriented Programming को भी Support करती है।

OOPS

WITH

C++

OOPS with C++

Object Oriented Programming Concept को समझे बिना हम "C++" की Capabilities का पूरा फायदा नहीं उठा सकते हैं। Class "C++" का सबसे जरूरी Concept है। Object Oriented Programming का Concept समझे बिना हम कोई Serious Program नहीं बना सकते हैं। इसलिए Class को समझना "C++" को समझने के लिए सबसे जरूरी है। Object Oriented Concept को "C++" के साथ Use करते हुए OOPS को समझाना काफी जटिल काम है। O

bject Oriented Technology का सार यही है कि Programmers अपने Program उसी प्रकार से Design कर सकें जिस तरह से Real World में किसी काम या समस्या को Design किया जाता है। हम यहां पर पहले यही समझेंगे कि Object Oriented Design क्या होता है और किस प्रकार से किसी समस्या को Object Oriented Form में Design करके Computer पर उस समस्या का समाधान प्राप्त किया जाता है। Object Oriented Programming Concept का मुख्य आधार **Object** व **Class** हैं।

Class and Objects

OOPS का Modal इस तथ्य पर आधारित है कि दुनिया की हर वह वस्तु जिसे हम Physically देख सकते हैं, छू सकते हैं या Logically अनुभव कर सकते हैं, एक Real World **Object** है और हर Object कई अन्य छोटे Objects से बना होता है। साथ ही हर Object किसी ना किसी Class का एक उदाहरण (Instance) या सदस्य होता है।

यदि एक उदाहरण देखें तो Computer एक Object है। लेकिन ये Computer स्वयं कई अन्य छोटे Objects जैसे कि Motherboard, Processor, RAM, Hard Disk, Floppy Drive, Cabinet, SMPS, Monitor, Keyboard, Mouse आदि से मिलकर बना होता है।

हम Real World में भी देखते हैं कि कई Objects के Features समान होते हैं। जैसे कि जितने भी Computers होते हैं, उन सभी में Motherboard होता है, Processor होता है, Hard Disk होती है, RAM होती है, CD ROM होता है, आदि। यानी हर Computer के Basic Features या आधारभूत अवयव समान होते हैं। इन सभी Objects के चारित्रिक गुणों या Characteristics में समरूपता है। इसलिए एक Computer किसी Computer Class का एक Instance या Object होता है।

Class एक ऐसा **Modal, Design** या **Description** होता है, जिसके आधार पर समान Characteristics वाले कई Object Create किए जा सकते हैं। यानी यदि दूसरे शब्दों में कहें तो Class एक ऐसा Modal, Design या Description होता है, जो किसी एक अमुक (**Particular**) समूह के Objects (**Entity Set**) की विशेषताओं (**Characteristics**) या **Features** को Represent करता है।

जब हम किसी Object Oriented Programming Language में Program लिखते हैं, तो Programming के समय हम किसी Object को Define नहीं करते हैं, बल्कि हम उस Object की विशेषताओं का एक Modal बना लेते हैं, जिसे Class कहते हैं और उस Modal **(Class)** के आधार पर आवश्यकतानुसार कई Objects Create कर लेते हैं। इसे एक उदाहरण द्वारा समझने की कोशिश करते हैं।

मानलो कि यदि आपसे कहा जाए कि एक Car की परिभाषा दीजिए। परिभाषा के रूप में आप क्या कहेंगे? आप Car की ऐसी विशेषताओं (**Characteristics**) का वर्णन (**Description**) करेंगे जो उसे दुनिया की दूसरी चीजों (**Objects**) से अलग बनाती है। आप कह सकते हैं कि Car एक ऐसी चीज होती है जिसके चार Wheels होते हैं। उसमें बैठने के लिए कुछ सीटें होती हैं। Car में एक Engine होता है। ये Engine Diesel से चलता है। इसमें गति को प्रभावित करने के लिए एक Accelerator और एक Gear Box होता है। इसी तरह से कई और विशेषताएं बताई जा सकती हैं, जिनसे एक Car की पहचान हो सकती है।

आपने एक Car की जो परिभाषा बताई है, क्या वह परिभाषा कार है। नहीं! ये परिभाषा कार नहीं है बल्कि ये परिभाषा तो Car का एक विवरण या **Description** मात्र है।

मानलो कि आपको किसी Automobile Company से ये Offer आता है कि आप जिस तरह की Car चाहते हैं, Company उस तरह की Car बनाने के लिए तैयार है। आप जिस तरह की चाहेंगे, उस तरह की Car बन जाएगी। अब आपको Company को बताना है कि आपको किस तरह की Car पसन्द है। सीधी सी बात है कि Company के Engineers आपकी कल्पनाओं वाली Logical Car को तभी साकार रूप की Physical Car में बदल सकते हैं, जब आप किसी तरीके से उन Engineers को अपनी उस सपनों वाली Logical कार का विवरण दे पाऐं। मानलो कि आप अपनी Car में निम्न विशेषताएं (Characteristics) या Features चाहते हैं:

1. आपकी कार पर Yellow, Blue व White ये तीन Color होने चाहिएं।
2. Car हल्की होनी चाहिए।
3. Car में केवल दो सीटें ही होनी चाहिए।
4. Car में सामान्य कारों की तुलना में एक Extra Gear होना चाहिए ताकि जरूरत पडने पर उस Gear के प्रयोग से Car जमीन से ऊपर भी उड सके।
5. Car का Engine Latest Quality का व सबसे Highest Power का होना चाहिए।
6. कार ऐसी होनी चाहिए की वह अपने चालक को Identify कर सके ताकि आप ये तय कर सकें कि उस Car को कौन–कौन चला सकेंगे। इस विशेषता के कारण आपकी कार कोई चोरी नहीं कर सकता।
7. जब भी आप Car के पास जाएं Car आपके लिए दरवाजा खोल दे।
8. जब भी आप अपने गन्तव्य स्थान पर पहुंचे, Car के रूकते ही दरवाजा खुल जाना चाहिए।

वगेरह, वगेरह। कल्पनाओं की कमी नहीं होती है, इसलिए ऐसी हजारों खूबियों वाली Car की आप कल्पना कर सकते हैं।

ये तो आपने अपनी सपनों वाली Logical Car का **Description** दे दिया। लेकिन कई स्थानों पर ऐसी जरूरत भी पड सकती है जब आप अपनी बात को केवल Description से नहीं समझा सकते। उस स्थिति में आप क्या करेंगे। आप मानें या ना माने, आप पेन और कागज उठाएंगे और स्वयं अपनी Car का एक पूर्ण सन्तुष्टिदायक **Modal** या **Design** बनाएंगे। क्या ये Modal आपकी Car है? क्या आप इस Modal को चला सकते हैं?

नहीं! ये वास्तविक कार नहीं है बल्कि एक Logical Car या Car का Modal मात्र है। लेकिन ये Modal फालतू नहीं है बल्कि ये Modal ही सबसे महत्वपूर्ण चीज है। एक बार इस Modal के बन जाने के बाद Company के Engineers आपकी Logical Car को एक Physical Car में बदल सकते हैं और आपके सपनों वाली Car को एक Physical Modal का रूप दे सकते हैं।

मानलो कि Company के Engineers ने आपकी Car बना दी और वह Car बिल्कुल वैसी ही है जैसी आप चाहते थे। अब यदि Company बिल्कुल उसी तरह की एक और Car बनाना चाहे, तो क्या Company को शुरू से वापस सारा काम करना पडेगा। क्या उसे दुबारा आपसे Modal बनवाना पडेगा।

नहीं! Modal केवल एक ही बार बनता है, लेकिन उस Modal के आधार पर हजारों कारें बन सकती हैं और जिस Modal के आधार पर हजारों Cars बन रही हैं, हम समझ सकते हैं कि जब तक Modal Change नहीं किया जाएगा, तब तक बनने वाली सभी कारों की सभी खूबियां एक समान होंगी। किसी भी Car के किसी भी Feature या Characteristic में किसी प्रकार का कोई अन्तर नहीं होगा।

जो Logical Modal आपने बनाया है, वह Object Oriented Programming Language में एक **Class** को Represent करता है और उस Modal के आधार पर Company के Engineers ने जो Physical Car बनाई है, वह Car एक Physical **Object** को Represent करता है। ये Physical Car आपके द्वारा बनाए गए Modal (**Class**) का एक उदाहरण (Instance) मात्र है, क्योंकि यदि Company चाहे तो उसी Modal के आधार पर कई अन्य Cars बना सकती है।

जिस तरह से एक Modal के आधार पर यदि Company चाहे तो हजारों कारें बना सकती है, उसी तरह से एक Object Oriented Programming Language में एक बार एक Class Design कर लेने के बाद एक Programmer उस Class के जितने चाहे उतने Instance Create कर सकता है।

"C++" में Class Create करने के लिए **class** Keyword का प्रयोग करना पडता है। वास्तव में Class एक नए प्रकार का **User Defined Data Type** होता है। ये एक ऐसा Data Type होता

है, जिसे Programmer Real World के Physical Objects को Computer Software या Program के Logical World के Object के रूप में Represent करता है।

हम एक और उदाहरण लेते हैं। पिछले उदाहरण में हमने एक Modal के आधार पर Object बनाया था। इस उदाहरण में हम एक Real World Object के आधार पर अपने Computer Program के लिए एक Logical Class बना रहे हैं। एक Real World Object के आधार पर उस Object की Logical Class बनाने के बाद, उस Class के आधार पर हम जितने चाहें उतने Objects बना सकते हैं, क्योंकि एक Object के Description के विश्लेषण से हमें Car का Modal (**Class**) मिल जाएगा।

मानलो कि हमें एक Physical Car Object को Computer में एक Logical Car के रूप में Represent करना है। अब Physical कार की ऐसी क्या विशेषताएं हैं, जो उसे किसी दूसरे Real World Object से अलग बनाती हैं, उन Characteristics को निम्नानुसार लिखा जा सकता है। यानी

1. एक Physical Car का कोई एक नाम होता है।
2. एक Physical Car का कोई ना कोई Modal Number होता है।
3. एक Physical Car के Engine का एक Unique Serial Number होता है।
4. एक Physical Car के Chassis का एक Unique Serial Number होता है।
5. एक Physical Car किसी ना किसी ईंधन से चलती है।
6. एक Physical Car का कोई Color होता है।
7. एक Physical Car की कोई Size होती है।
8. एक Physical Car में कुछ Seats होती हैं।
9. एक Physical Car में कुछ Gear होते हैं।
10. एक Physical Car का Pickup Rate होता है।
11. एक Physical Car किसी ना किसी अधिकतम Speed पर दौड सकती है।
12. एक Physical Car में चार Wheels होते हैं।

इसी तरह से हम कुछ और विशेषताएं भी बता सकते हैं, जो किसी Car को परिभाषित करती हैं। किसी एक Car के आधार पर हम Car की इतनी विशेषताएं प्राप्त कर सकते हैं। इन सभी Descriptions को निम्नानुसार भी लिखा जा सकता हैः

Description of the Car

(**Class**)

nameOfCar
modalNumberOfCar
serialNumberOfEngine
serialNumberOfChassis
fuelOfCar

colorOfCar
sizeOfCar
noOfSeatsInCar
noOfGearsInCar
pickupRateOfCar
maximumSpeedOfCar
noOfWheelsInCar

यदि हम चाहें तो इन Features में निम्नानुसार विभिन्न प्रकार के मान भी Fill कर सकते हैं:

Description of the Maruti800 Car

(**Class**)

nameOfCar	=	*Maruti800*
modalNumberOfCar	=	*March1999*
serialNumberOfEngine	=	*12325465*
serialNumberOfChassis	=	*65457898*
fuelOfCar	=	*Diesel*
colorOfCar	=	*Yellow*
sizeOfCar	=	*4.5ft*
noOfSeatsInCar	=	*6*
noOfGearsInCar	=	*5*
pickupRateOfCar	=	*60KMPH After 4 Seconds*
maximumSpeedOfCar	=	*140KMPH*
noOfWheelsInCar	=	*4*

यदि हम थोडा ध्यान दें तो समझ सकते हैं, कि ये कुछ ऐसी Common विशेषताएं या कुछ ऐसे Common Features (Characteristics) हैं, जो लगभग सभी प्रकार की कारों के Features को Represent कर सकते हैं। उदाहरण के लिए हमने ऊपर Maruti800 के विभिन्न Features के मानों को देखा। अब निम्नानुसार Features के मानों में परिवर्तन करने पर हम इन्ही Features के आधार पर Tata Indica या Tata Sumo को भी Computer में Logically Represent कर सकते हैं। यानी

Description of the Tata Indica Car

(**Class**)

nameOfCar	=	*TataIndica*
modalNumberOfCar	=	*Feb2000*
serialNumberOfEngine	=	*25465454*
serialNumberOfChassis	=	*98787898*
fuelOfCar	=	*Diesel*

colorOfCar	=	*Silver*
sizeOfCar	=	*4.25ft*
noOfSeatsInCar	=	*5*
noOfGearsInCar	=	*5*
pickupRateOfCar	=	*80KMPH After 4 Seconds*
maximumSpeedOfCar	=	*160KMPH*
noOfWheelsInCar	=	*4*

हम देख सकते हैं कि Car के केवल एक Physical Modal के आधार पर हमने Computer में एक Logical Modal बनाया और उस Logical Modal की Description के आधार पर हम जितनी चाहें उतनी कारों को Logically Computer में Represent कर सकते हैं। इस Description को ही **Class** कहते हैं।

इस Car Class के आधार पर हम जितनी भी कारों को Computer में Logically Represent करेंगे, वे सभी कारें इसी Car Class के उदाहरण (Instance) या Objects होंगे, क्योंकि उन सभी कारों के Basic Features एक समान होंगे केवल उन Features के मानों में ही अन्तर होगा।

हमने ये Description किसी एक Car के आधार पर Develop किया था, लेकिन ये Description अन्य सभी कारों को भी समान प्रकार से Represent कर सकता है। Computer में भी यही किया जाता है। किसी समस्या के सबसे मूल Object का पता लगाया जाता है। फिर उस मूल Object के आधार पर समस्या से सम्बंधित जरूरी Features को लेकर एक Description तैयार किया जाता है।

इस Description के आधार पर एक Modal बनाया जाता है और उस Modal के आधार पर हम उस Class के जितने चाहें उतने Logical Objects Create कर सकते हैं और उसी तरह से उन्हें Represent कर सकते हैं जिस तरह से Real World Objects को Represent करते हैं।

यदि आपसे कहा जाए कि पांच कारों के उदाहरण दो तो आप Tata Sumo, Tata Indica, Maruti Zen, Maruti 1000, व Maruti 800 के उदाहरण दे सकते हैं। क्या इन सभी प्रकार की कारों को उपरोक्त Description के आधार पर Represent नहीं किया जा सकता है? निश्चित रूप से किया जा सकता है, क्योंकि ये सभी Objects समान Class के उदाहरण हैं। ये सभी एक Car Class के Instances या Objects हैं और इनके Basic Features समान हैं।

यदि यहां हम वापस से Class की परिभाषा को दोहराएं तो कह सकते हैं कि एक Class समान Featurs वाले Objects के एक समूह (Entity Set with Similar Features) का Logical या **Abstract Representation** होता है, जबकि उस Class का कोई Instance (उदाहरण) उस Class का **Actual Representation** या Physical Representation होता है जिसे **Object** कहते हैं।

सामान्यतया "C++" की हर Class दो मुख्य Components से बनी होती है :

1 Attributes

उदाहरण के लिए किसी Objects के एक समूह (**Entity Set**) को लीजिए। आप देखेंगे कि उन सभी में कुछ ऐसी बातें हैं, जो सभी Objects को समान रूप से प्रभावित करती हैं। ये वही विशेषताएं होती हैं जो किसी Object को किसी दूसरे Object से अलग बनाती हैं। इन विशेषताओं को **Attributes** कहा जाता है।

हर Object किसी ना किसी Class का एक Instance या सदस्य होता है। किसी Entity Set के ये Attributes ही उनकी Class के Features होते हैं। जैसे यदि हम एक Book को ही लें तो Book की अपनी एक Class हो सकती है, जिससे हम दुनिया की किसी भी Book को Describe कर सकते हैं।

अब इस Book की Class के Features वही होंगे जो किसी एक Book के Features हैं। Book की ऐसी कौनसी Properties हो सकती हैं, जो एक Book को दूसरे किसी Object से अलग बनाती हैं? इसके सवाल के जवाब में हम किसी Book को निम्न Properties द्वारा परिभाषित कर सकते हैं:

1. हर Book का एक नाम होता है।
2. हर Book का एक Author होता है।
3. हर Book किसी ना किसी Publication से प्रकाशित होती है।
4. हर Book का एक ISBN Number होता है, जिससे उस Book की Unique पहचान होती है।
5. हर Book में कुछ Pages होते हैं।
6. हर Book की कुछ कीमत होती है।

इसी तरह से Book की कई और Characteristics हो सकती हैं। ये सभी Characteristics उस Book Object की Class के Features होते हैं। इन्हीं Features को Book Class के **Attributes** भी कह सकते हैं। यदि हम इस Book की Description को Class के Modal के रूप में Describe करें तो इस Book Object की Class को निम्नानुसार Represent कर सकते हैं:

Book's Attributes Description

(Book Class)
bookName
bookAuthor
bookPublication
bookISBN
bookPages
bookPrice

किसी Class के **Attributes** किसी **Object** की **Appearance, State** व **Condition** के आधार पर तय होते हैं। यानी किसी Class की Description के Attributes इस तथ्य पर आधारित होते हैं कि उस Class के Object किस प्रकार के दिखाई देते हैं और उनके Objects की स्थिति (Position or Situation) क्या हो सकती है। किसी Object के इन Appearance व State की विभिन्न Description ही उस Object को दुनिया के सभी Object से अलग पहचान प्रदान करती है।

उदाहरण के लिए इस Book Class को ही लेते हैं। हम इस Book Class का एक Object Create करके उसके विभिन्न Attributes को निम्नानुसार मान प्रदान कर सकते हैं:

Book's Attributes Description

(Book Class)		
bookName	=	B with OOPS through C++ in Hindi
bookAuthor	=	Kuldeep Mishra
bookPublication	=	Publication
bookISBN	=	010101010101
bookPages	=	600
bookPrice	=	300

ये सभी Fields Book की State यानी Situation बता रहे हैं। यानी Book का नाम "B with OOPS through C++ In Hindi" है, जो कि Book की एक स्थिति है। इसी तरह से Book की Price 300 है, ये भी Book की स्थिति को Represent कर रहा है।

यदि हम Car का उदाहरण लें, तो Car का Color व Car की Size Car के Appearance को Represent करते हैं। यानी ये बताते हैं कि Car दिखाई कैसी देती है। जबकि Car का नाम Car की State या Situation को Represent करता है, जो ये बताता है कि Car की स्थिति क्या है यानी Car बनाने वाली Company की Market में अच्छी साख है या कोई साधारण Company है।

Class की Common Descriptions उसकी Class के सभी Objects के लिए समान होती हैं और Object की ये Common Descriptions एक Object को दुनिया के किसी भी अन्य Object से अलग बनाती हैं। जैसे एक Car और एक Book दोनों अलग Object हैं। Car को Car Class के Attributes से पहचाना जाता है जबकि Book को Book Class के Attributes से। Class किसी भी एक Object से दूसरे Object को अलग Represent करने का एक Description होता है जबकि एक Object की उसी Class के दूसरे Objects से अलग पहचान Object के Features या Attributes को प्रदान किए जाने वाले मान पर निर्भर होती है।

यानी यदि एक Book का नाम **"C In Hindi"** है और दूसरी Book का नाम **"OOPS with C++ In Hindi"** है, तो ये दोनों ही Books **Book** Class के उदाहरण (Instance) या सदस्य हैं, लेकिन

दोनों ही Books अलग Objects को Represent कर रहे हैं, क्योंकि दोनों के **bookName** Attribute के मान या Data में अन्तर है।

Class एक Derived Data Type होता है, जिसे एक Programmer दुनिया के Actual Objects को Computer में Logically Represent करने के लिए Create करता है। किसी Real World Problem को Computer द्वारा Solve करने के लिए हमें हमेंशा एक ऐसे Data Type की जरूरत होती है, जो Real World Object को Reflect कर सके। ये सुविधा "C++" में हमें Class द्वारा प्राप्त होती है।

चलिए एक उदाहरण द्वारा समझने की कोशिश करते हैं कि एक Real World Problem द्वारा किस प्रकार से एक नए Data Type को प्राप्त किया जा सकता है जिसके आधार पर Computer में Logically किसी Object को Define व Declare किया जा सकता है।

Real World में भी Data (मान या मानों का समूह [**Value** or a **Set of Values**]) कई प्रकार के होते हैं। उदाहरण के लिए मानलो कि एक School का Principal अपनी School के विभिन्न Students की जानकारी को Computer पर Maintain करना चाहता है। वह चाहता है कि उसे जब भी जरूरत हो, वह Computer का प्रयोग करके अपनी School के किसी भी Student की Basic जानकारी को प्राप्त कर ले। यदि हम किसी Student की Basic Information को देखें तो किसी भी Student की निम्न Basic Information हो सकती हैं, जिन्हें School का Principal जानना पसन्द कर सकता हैः

- Student का नाम
- Student के पिता का नाम
- Student का Address
- Student की City
- Student की तहसील
- Student का जिला
- Student का State
- Student की Class
- Student के Contact Number
- Student की Date of Birth
- Student की School में Join करने की Date of Admission
- Student की Age
- Student का Serial Number
- Student की जाति

ये कुछ ऐसी जानकारियां हैं, जिन्हें School का Principal जानना पसन्द कर सकता है। किसी Student की पूरी Information को इस प्रकार से टुकडों में विभाजित किया जा सकता है और

जानकारी के इन सभी हिस्सों में कोई मान (Data) Fill किया जा सकता है। किसी जानकारी के इन छोटे–छोटे हिस्सों या टुकडों को **Field** कहा जाता है। जब विभिन्न Fields मिलते हैं तब किसी एक Student के बारे में पर्याप्त जानकारी या Information प्रदान करते हैं। यानी किसी समस्या से सम्बंधित विभिन्न Fields मिलकर उस समस्या के बारे में पूरी जानकारी प्रदान करते हैं। इन Fields के समूह को **Record** कहा जाता है। यानी हर Student का एक पूरा Record होता है जिसमें उसकी जानकारी के विभिन्न हिस्से Fields के रूप में होते हैं। इसी तरह के कई Records को जब एक साथ रखा जाता है, तो उस एक साथ रखे गए विभिन्न Records के समूह को **File** कहा जाता है।

Student की कुछ और भी जानकारियां हो सकती हैं, जो एक Student को अन्य Student से अलग बनाती हैं। जैसे Student क्या खाना पसन्द करता है, किस प्रकार से रहना पसन्द करता है, उसे कौनसा TV Channel अच्छा लगता है, आदि। लेकिन School का Principal Student की इन जानकारियों को जानने में Interested नहीं है। ये जानकारियां उसके लिए फालतू हैं। यानी केवल उपरोक्त जानकारियां School के Principal के लिए जरूरी हैं। किसी समस्या की प्रकृति के अनुसार समस्या की केवल जरूरी बातों को लेना और बिना जरूरी बातों को छोड देना, इस प्रक्रिया को Object Orientation में **Abstraction** कहते हैं।

किसी Student की इन सभी जरूरी जानकारियों को यदि मान प्रदान किया जाए तो हम किसी एक Student के इन Fields को निम्नानुसार विभिन्न मान प्रदान कर सकते हैं:

1	Student का नाम	=	Bal Gopal
2	Student के पिता का नाम	=	Nandlal
3	Student का Address	=	Bedal Road
4	Student की City	=	Falna
5	Student की तहसील	=	Bali
6	Student का जिला	=	Pali
7	Student का State	=	Rajasthan
8	Student की Class	=	10^{th}
9	Student के Contact Number	=	9352768938
10	Student की Date of Birth	=	06/03/1982
11	Student की School में Join करने की Date of Admission	=	03/07/1996
12	Student की Age	=	15
13	Student का Serial Number	=	1234545
14	Student की जाति	=	Brahmin

हम देख सकते हैं कि ये सभी Fields एक दूसरे से Logically Related हैं। यदि हम चाहें तो इसे एक Structure के रूप में Define कर सकते हैं और Structure एक नए प्रकार का User

Defined Data Type ही तो होता है। "C++" में इस Structure को ही Modify करके Class के रूप में परिभाषित किया गया है। इसलिए हम इसी Description के आधार पर एक Class का निर्माण कर सकते हैं।

जब किसी Object को Computer में Logically Represent करने के लिए Class की Description या Class के Modal को कागज पर तैयार कर लिया जाता है, तब उस Class को Computer में Programming Language के अनुसार Define किया जाता है। Class को Logically Computer में Define करने के बाद उस Class के Objects Create किए जा सकते हैं और उनके साथ उसी प्रकार से काम किया जाता है, जिस प्रकार से किसी Real World Object के साथ काम किया जाता है। "C++" में एक Class को Define करने का Format निम्नानुसार होता है:

```
class class_name
{
private:
    Data_Members;
    Member_Functions;

public:
    Data_Members;
    Member_Functions;
};
```

class "C++" का एक Keyword है। किसी Class को Computer में Create करने के लिए इस Keyword का प्रयोग किया जाता है। इस Keyword के बाद एक Programmer को अपनी Class का नाम देना होता है। ये वही नाम होता है, जिसके आधार पर Programmer अपने Program में उस Class के Instances या Objects Create करता है।

यदि हम हमारी Book व Car Class के सन्दर्भ में देखें तो यहां हमें Book या Car लिखना होता है। ये एक Identifier होता है, इसलिए Identifier को नाम देने के लिए जिन नियमों का "C" में पालन करना होता है, उन नियमों का पालन यहां भी करना होता है।

यानी हम केवल Small व Capital Letters का प्रयोग या Underscore चिन्ह का प्रयोग अपनी Class के नाम में कर सकते हैं। यदि हमें अंकों का प्रयोग करना हो तो वह अंक Class के नाम की शुरूआत में नहीं आ सकता।

एक Student के Record को हम Class के रूप में Describe कर सकते हैं। उपरोक्त Description किसी भी Student के उन Fields को Represent करते हैं, जिनकी School के Principal को जरूरत है। यदि हम इन Description को सामान्य Fields में Convert करें तो हमें निम्नानुसार विभिन्न Fields प्राप्त होते हैं, जिनको Combined रूप में देखने पर Principal को किसी Student के बारे में वे सभी जानकारियां प्राप्त हो सकती हैं, जिन्हें उस School का Principal जानना चाहता हैः

1. studentName
2. studentFName
3. studentAddress
4. studentCity
5. studentTehsil
6. studentDistrict
7. studentState
8. studentClass
9. studentContactNumber
10. studentDateOfBirth
11. studentDateOfAdmission
12. studentAge
13. studentSerialNumber
14. studentCast

चूंकि Student एक Real World Object है और इसे Computer में Logically Represent करना है, इसलिए हमें इसे Class द्वारा Represent करना होगा। चूंकि Student की सभी Basic जानकारियों के Fields जिनकी School के Principal को जरूरत है, हमारे पास Student Class की Description के रूप में उपलब्ध है, इसलिए हम इन्हीं Attributes का प्रयोग Student Class के लिए कर सकते हैं। लेकिन इससे पहले हम किसी Student के इन सभी Fields को मान प्रदान करते हैं। ये मान निम्नानुसार प्रदान किए जा सकते हैंः

1. studentName = Bal Gopal
2. studentFName = Nandlal
3. studentAddress = Bedal Road
4. studentCity = Falna
5. studentTehsil = Bali
6. studentDistrict = Pali
7. studentState = Rajasthan
8. studentClass = 10^{th}

9	studentContactNumber	=	9352768938
10	studentDateOfBirth	=	06/03/1982
11	studentDateOfAdmission	=	03/07/1996
12	studentAge	=	15
13	studentSerialNumber	=	1234545
14	studentCast	=	Brahmin

हम देख सकते हैं कि एक Student के विभिन्न Attributes को मान यानी Data प्रदान किया जा सकता है। चूंकि हम जानते हैं कि Computer में मानों "**Values / Set of Values**" को Store करने के लिए Variables का प्रयोग किया जाता है, इसलिए इस Record के आधार पर हम समझ सकते हैं कि Student के हर Attribute को Computer में Store करने के लिए हमें एक Variable की जरूरत होगी। अब Variable किस Data Type का होगा, ये हम उपरोक्त Record में Fill किए गए मानों के आधार पर तय कर सकते हैं। चलिए इस Record के विभिन्न Attributes को Class में Describe करने के लिए हर Attribute का Data Type तय करते हैं।

हमेंशा Data Type तय करते समय हमें सबसे पहले ये तय करना होता है कि किसी Field का या किसी Attribute का मान Numerical होगा या Non-Numerical, यदि मान Numerical है, तो हमें ये देखना होता है कि उस Numerical Field के साथ किसी प्रकार की प्रक्रिया हो सकती है या नहीं। यदि प्रक्रिया हो सकती है, तब तो उस Attribute को उसकी Size के अनुसार किसी Numeric Data Type का Declare करना चाहिए अन्यथा उस Numerical दिखाई देने वाले Field या Attribute को भी String प्रकार का ही Declare करना चाहिए। चलिए, इसी आधार पर हम Student के विभिन्न Attributers का Data Type तय करते हैं।

चूंकि Student का नाम, उसके पिता का नाम, Address, जाति, City, District, तहसील, State, Class ये सभी ऐसे Fields हैं, जिनमें Non-Numerical Data Store होगा, इसलिए इन सभी को Character प्रकार के Data Type के एक One – Dimensional Array के रूप में Declare करना होगा।

Admission की Date, Birth of Date व Age ये तीनों ऐसे Fields हैं, जिनके साथ ये जानने के लिए Processing की जा सकती है, कि कोई Student कितने समय से School में है और कितने दिन से School में है। ये जानकारी प्राप्त करने के लिए Current Date को Admission Date में से घटाया जा सकता है। साथ ही ये तीनों ही पूर्णांक संख्याएं हैं इसलिए इस प्रकार के Fields को Integer प्रकार का Declare करना होगा।

Serial Number, Class व Contact Number ये तीन Attributes ऐसे हैं कि इनमें Store तो संख्या ही होती है, लेकिन इन संख्याओं के साथ किसी प्रकार की Calculation नहीं की जा सकती है, इसलिए यदि हम चाहें तो इन्हें String प्रकार के Variable में Store कर सकते हैं।

लेकिन String प्रकार का Variable एक Character को Store करने के लिए 1 Byte लेता है। इसलिए यदि 10 अंकों के Contact Number को Store करना हो, तो हमें दस Byte की Memory इसके लिए Reserve करनी होगी, जबकि यदि हम **long** प्रकार का Variable Declare करते हैं, तो हम केवल 8 Bytes में ही दस अंकों की संख्या Store कर सकते हैं। इसलिए यदि हम चाहें तो इन्हें Long प्रकार का Declare कर सकते हैं।

इस प्रकार से हम Students के सभी Attributes के Data Type तय कर सकते हैं। Data Type तय करने के बाद इन Data Type के साथ हमें इन Attributes को केवल Class के Attributes Section में लिखना होता है। यानी हम एक Student की Class को "C++" Language द्वारा निम्नानुसार Computer में Logically Describe कर सकते हैं:

Student Class Defining

```
class Student
{
    private:
        String studentName[20];
        String studentFName[20];
        String studentAddress[40];
        String studentCity[15];
        String studentTehsil[15];
        String studentDistrict[15];
        String studentState[15];
        char studentClass[1];
        long studentContactNumber;
        long studentDateOfBirth;
        long studentDateOfAdmission;
        char studentAge[1];
        String studentSerialNumber[10];
        String studentCast;

    //Behaviors of the Objects
};
```

चूंकि किसी Class के Create होने वाले सभी Instances एक अलग Object होते हैं और उनके Attributes के मान भी अलग–अलग होते हैं। जैसे **Govind** नाम के किसी Student के Record व **Shyam** नाम के किसी Student के Record, इन दोनों Objects के Record के हर Field या Attribute का मान अलग होगा। ये अलग मान Memory में तभी Store हो सकते हैं जब हर

Object के इन Attributes के लिए Memory में अलग Space हो। वास्तव में होता भी ऐसा ही है।

जैसाकि हमने पहले भी कहा कि Class तो मात्र किसी Object के Attributes की एक Description होती है। उस Description के अनुसार जितने भी Objects बनते हैं, उन सभी Objects के हर Attribute को Store होने के लिए एक अलग Variable मिलता है।

चूंकि हर Object किसी Class का एक Instance होता है इसलिए हर Object के इन Attributes को Represent करने वाले Variables को भी **Instance Variable** कहते हैं। इसी उदाहरण के आधार पर हम Car व Book की Class को भी Create कर सकते हैं।

हम देख सकते हैं कि हमने Student के विभिन्न Attributes को **private:** Section में Declare किया है। हमने ऐसा इसलिए किया है, ताकि किसी Object के Data को केवल वही Object Access कर सके। Outer World के लिए किसी Object के लिए ये Data Hide रहते हैं।

ये प्रक्रिया ठीक उसी प्रकार की है जिस प्रकार से आपके घर के विभिन्न सामानों को केवल आप ही उपयोग में ले सकते हैं। कोई अनजान व्यक्ति आपके घर में घुस कर आपके सामान को तब तक उपयोग में नहीं ले सकता जब तक कि आप उसे इस बात की अनुमति प्रदान ना करें।

हमने Class तो बना दी लेकिन ये Class अभी पूरी नहीं है। जिस तरह से हर Object की कुछ States या Situations होती हैं, जिनसे वह Object अन्य Objects से अलग पहचान प्राप्त करता है। इसके अलावा हर Object किसी ना किसी प्रकार का व्यवहार भी करता है। यानी हर Object में व्यवहार करने की भी कुछ (**Abilities**) क्षमताएं होती हैं।

जैसे एक पक्षी Class के Instances को देखें तो हर पक्षी के पास पंख होते हैं, हर पक्षी के चोंच होती है। ये दोनों Features तो ये बताते हैं कि कोई पक्षी किस तरह का दिखता (Appearance) है। इसके अलावा एक पक्षी कुछ काम भी कर सकता है। यानी वह दाना खा सकता है और उड भी सकता है।

यानी हर Object की अपनी कुछ स्थितियां (**States**) होती हैं और हर Object की अपनी कुछ क्षमताएं (**Abilities**) होती हैं। Object की उन क्षमताओं से Object की States में परिवर्तन होता है। किसी भी Object की स्थिति (**State and Appearance**) के मान में परिवर्तन हो सकता है।

यदि कार का ही उदाहरण लें तो कोई Car Yellow Color की भी हो सकती है और किसी दूसरी Car का Color Blue भी हो सकता है। हम किसी Car के Color को Red Color से भी Paint कर सकते हैं और White Color से भी Paint कर सकते हैं। Real World में किसी भी Object की स्थिति में परिवर्तन किया जा सकता है। उसी प्रकार से Computer में भी किसी Object के Color State या Attribute का मान Change किया जा सकता है।

हम ये भी कह सकते हैं कि Real World में एक Object में उसके Attributes के मान को Change करने की क्षमता होती है। एक Real World Object जिस Operation को Perform करके अपनी स्थिति या Appearance में परिवर्तन करता है, उस Operation को Object का **Behavior** या Object की **Ability** कहते हैं। चलिए, एक और उदाहरण देखते हैं।

हम सभी ने देखा है कि बडे शहरों में कई–कई मंजिलों की इमारतें होती हैं इसलिए अक्सर उन मंजिलों पर पहुंचने के लिए Lift का प्रयोग किया जाता है। Lift को सामान्यतया Elevator कहते हैं। Elevator भी एक Real World Physical Object है, क्योंकि हम इसे देख और छू सकते हैं। यदि हम इसकी States व Appearance यानी Characteristics को Describe करें तो इस Elevator का निम्नानुसार Description दे सकते हैं:

1. एक Elevator किसी ना किसी मंजिल (Floor) पर रूका हुआ या स्थित हो सकता है।
2. एक Elevator अधिकतम चार लोगों को वहन कर सकता है।
3. एक Elevator में कुछ Buttons होते हैं, जिनको Press करके विभिन्न Passengers किसी ना किसी Floor पर पहुंच सकते हैं।
4. एक Elevator में एक दरवाजा हो सकता है जो Open व Close होता है।

किसी Elevator में ये चारों गुण हो सकते हैं जो उस Elevator की स्थिति व Appearance को Describe कर रहे हैं। लेकिन ये Elevator Passengers की इच्छानुसार कुछ Operations भी Perform कर सकता है, क्योंकि इन Operations को Perform करने की Ability भी एक Elevator Object में होती है। Elevator की इन Abilities को निम्नानुसार Describe किया जा सकता है:

1. एक Elevator किसी भी समय किसी ना किसी Floor पर स्थित होगा, इसलिए वह अपनी Current State में परिवर्तन करने के लिए अपनी मंजिल (Floor) से ऊपर की मंजिल (Floor) पर जा सकता है।
2. एक Elevator किसी भी समय किसी ना किसी Floor पर स्थित होगा, इसलिए वह अपनी Current State में परिवर्तन करने के लिए अपनी मंजिल (Floor) से नीचे की मंजिल (Floor) पर आ सकता है।
3. Passengers Elevator में प्रवेश कर सकें और Elevator से बाहर निकल सकें, इसके लिए एक Elevator का दरवाजा (Door) Open या Close हो सकता है।
4. जब कोई Passenger किसी मंजिल पर जाने के लिए उस मंजिल का कोई Number Press करता है, तब Elevator ये Calculate कर सकता है कि उसे किस जगह जाना है।
5. एक Elevator ये भी पता लगाता है कि उस Building के अन्य Elevators किस स्थिति (Floor) पर हैं।

इस Elevator Object की States या Characteristics व Object द्वारा उन Characteristics पर Perform किए जा सकने वाले Operations को हम निम्नानुसार Describe कर सकते हैं:

//Characteristics **(Attributes** or **States)**

1 noOfCurrentFloor
2 noOfPassengers
3 listOfButtonsPushed

//Performable Operations (**Behaviors** or **Abilities**)

1 GoDown
2 GoUp
3 OpenDoors
4 CloseDoors
5 GetOtherElevatorsInfo
6 CalculateWhereToGoNext

इस उदाहरण से हम समझ सकते हैं कि एक Real World Physical Object में भी वास्तव में दो Components होते हैं। पहला Component Object की स्थिति बताता है और दूसरा Component Object की क्षमता बताता है। जब एक Real World Physical Object के भी दो Component होते हैं तो उस Object को Represent करने वाली Logical Class में भी इन दोनों Components का Description होना चाहिए, ताकि Computer का Logical Object Real World के Physical Object को पूरी तरह से Represent करे।

एक और उदाहरण लेते हैं, Real World के Class व Objects की परिभाषा के अनुसार दुनिया की हर चीज को उसकी Class बना कर, उस Class के Instance यानी Object के रूप में परिभाषित किया जा सकता हैं। इसी आधार पर आप भी एक Real World Physical **Object** हैं और किसी एक समय पर आप भी किसी ना किसी एक Class के Instance होते हैं।

उदाहरण के लिए मान लीजिए कि आप एक Student हैं और 10th Class में पढते हैं इसलिए आप भी एक Student का उदाहरण (Instance) हैं क्योंकि आपको किसी कक्षा (Class) का एक विद्यार्थी (Object) कहा जा सकता है। आप जैसे ही कई और Students होंगे जो आपके साथ पढते होंगे।

मानलो कि 10th Class में पढने वाले सभी Students को मुख्य रूप से 6 Subjects पढने होते हैं। सभी Students को सभी प्रकार के Exams देने होते हैं। सभी Students समान समय पर Class में जाते हैं और एक समय में सभी Students समान Subject पढते हैं। जब एक कक्षा के सभी Students उस कक्षा में जो भी काम करते हैं, वे सभी काम सभी को एक साथ करने होते है तो इस आधार पर हम ये कह सकते हैं कि आप सभी Student एक ही कक्षा (Class) के विद्यार्थी(Objects) हैं, क्योंकि आप सभी एक ही कक्षा की सभी स्थितियों (States) को एक साथ वहन करते हैं।

सभी Students को एक समूह के रूप में देखा जाए तो सभी Students Objects का एक ऐसा समूह (**Entity Set**) है, जो कई समान Features को Share करते हैं। चूंकि आप 10th कक्षा के

सभी नियमों (**Descriptions**) का पालन करते हैं, इसलिए आप 10th कक्षा (**Class**) के Students (**Objects**) हैं।

इस उदाहरण में हमने देखा कि एक Student उसकी कक्षा का एक विद्यार्थी होता है। ये Description वास्तव में कक्षा का Description है और कक्षा के Description के आधार पर बनी Class का Instance विद्यार्थी होता है। यदि कक्षा के Description को प्रदर्शित करने वाले Attributes (Instance Variable) के मान को Change किया जाए, तो Student का पूरा समूह Change हो जाता है, ना कि केवल एक Student Change होता है।

यानी यदि 10th कक्षा की Description Change करके 11th कक्षा की Description ले लिया जाए, तो उस कक्षा के सभी विद्यार्थी बदल जाते हैं, क्योंकि वास्तव में पूरी कक्षा ही बदल जाती है। लेकिन यदि हम चाहते हैं कि हम Particular किसी विद्यार्थी की Description बनाएं तो हमें कक्षा की Class बनाने के बजाय Student की Class बनानी होगी।

जब किसी Student के Class की Description तय करनी होती है या एक Student को Represent करने वाला Modal (Class) Create करना होता है, तब हमारा मुख्य Object Student होता है। इसलिए हमें केवल Student के Attributes व States का ही पता लगाना होता है।

अब यदि देखा जाए किस एक Student के समूह (**Entity Set**) की ऐसी कौनसी Characteristics होती हैं, जो उसे दुनियां के सभी Objects से अलग बनाते हैं, तो हम एक Student की निम्न Attributes या States प्राप्त कर सकते हैं:

1. Student Entity Set के हर Entity का एक Serial Number होता है।
2. Student Entity Set के हर Entity का एक नाम होता है।
3. Student Entity Set के हर Entity के पिता का नाम होता है।
4. Student Entity Set के हर Entity का एक Address होता है।
5. Student Entity Set के हर Entity की एक City होती है।
6. Student Entity Set के हर Entity की एक District होती है।
7. Student Entity Set के हर Entity का एक State होता है।
8. Student Entity Set के हर Entity के City का एक Pin Code Number होता है।
9. Student Entity Set के हर Entity की कोई जाति होती है।
10. Student Entity Set के हर Entity का कोई रंग होता है।
11. Student Entity Set के हर Entity की कोई कक्षा होती है।
12. Student Entity Set के हर Entity के कद की एक लम्बाई होती है। आदि

इसी तरह से हम एक Student की कई और Attributes व States का पता लगा सकते हैं। चलिए, Real World में हम देखते हैं कि हर Object अपना काम खुद करता है।

मानलो कि आप एक Student हैं। क्या आपको भूख लगने पर आपका खाना कोई और खा सकता है। क्या आपको प्यास लगने पर आपके लिए कोई और पानी पी सकता है। क्या ऐसा होता है कि School जाते समय आपकी Uniform कोई और पहने और School आप चले जाएं। क्या ऐसा होता है कि आपके बदले कोई और आपका Exam Fight करे और मिलने वाला Certificate आपके नाम का हो और आप को प्राप्त हो जाए।

नहीं! Real World में ऐसा नहीं होता। इसीलिए Computer में भी यदि ऐसा होता है, तो उस Programming Language को Object Oriented नहीं कहा जा सकता। "C++" में भी ऐसा नहीं होता है। किसी Class के Description के आधार पर जो Object Create किया जाता है, उस हर Object का अपना स्वयं का Attribute या States को Represent करने वाला स्वतंत्र Variable होता है, जिसमें केवल उसी Object के Attributes या States के मान होते हैं।

चलिए, अब Object के Behavior या Ability पर थोडा और विचार करते हैं। हम सभी इन्सान हैं। हमारी एक Physical आकृति, रंग, रूप, लम्बाई आदि है, इसलिए हम सभी Human Being Class के Instances या Objects हैं। यदि हम एक Human Being Class का Description देना चाहें, तो निम्न Description दे सकते हैं:

1. हर Human Being का एक नाम होता है।
2. हर Human Being Object के पास उसका स्वयं का दिमाग होता है।
3. हर Human Being के दो हाथ होते हैं।
4. हर Human Being के दो पैर होते हैं।
5. हर Human Being के एक सिर होता है।
6. हर Human Being के सिर पर बाल होते हैं।
7. हर Human Being कान होते हैं
8. हर Human Being के दो आंखें होती हैं।
9. हर Human Being के चेहरे पर एक मुंह होता है।
10. हर Human Being के दोनों हाथों में पांच–पांच अंगुलियां होती हैं। आदि

इन सभी States व Attributes के अलावा एक Human Being में कुछ Abilities भी होती हैं। कुछ Abilities को निम्नानुसार Describe किया जा सकता है:

1. हर Human Being अपना नाम Change कर सकता है।
2. हर Human Being Object अपने दिमाग से सोंचता है और निर्णय लेता है।
3. हर Human Being अपने हाथों का प्रयोग करके किसी प्रकार का इशारा कर सकता है।
4. हर Human Being अपने पैरों से चल सकता है।
5. हर Human Being भाग सकता है।
6. हर Human Being अपने सिर पर टोपी रख सकता है।
7. हर Human Being अपने सिर पर पगडी बांध सकता है।
8. हर Human Being अपने सिर के बिखरे हुए बालों को कंघे से संवार सकता है।

How to Get Complete PDF EBook

आप **Online Order** करके **Online** या **Offline** Payment करते हुए इस Complete EBook को तुरन्त Download कर सकते हैं।

Order करने और पुस्तक को Online/Offline Payment करते हुए खरीदने की पूरी प्रक्रिया की विस्तृत जानकारी प्राप्त करने के लिए आप **BccFalna.com** के निम्न Menu Options को Check Visit कर सकते हैं।

How to Make Order

How to Order?

How to Buy Online

How to Pay Online using PayUMoney

How to Pay Online using Instamojo

How to Pay Online using CCAvenue

How to Buy Offline

How to Pay Offline

Bank A/c Details

जबिक हमारे Old Buyers के **Reviews** भी देख सकते हैं ताकि आप इस बात का निर्णय ले सकें कि हमारे Buyers हमारे PDF EBooks से कितने Satisfied हैं और यदि आप एक से अधिक EBooks खरीदते हैं, तो **Extra Discount** की Details भी Menubar से प्राप्त कर सकते हैं।