SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

21CSC101T OBJECT ORIENTED DESIGN AND PROGRAMMING

(PROFESSIONAL CORE)

L-2, T-1, C-3

Course Learning Objectives (CLO)

- *Programs using object-oriented approach and design methodologies for real-time application development*
- *Method overloading and operator overloading for real-time application development programs*
- *Inline, friend and virtual functions and create application development programs*
- *Exceptional handling and collections for real-time object-oriented programming applications*
- *Model the System using Unified Modelling approach using different diagrams*

Course Outcomes (CO)

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- *Create programs using object-oriented approach and design methodologies*
- *Construct programs using method overloading and operator overloading*
- *Create programs using inline, friend and virtual functions, construct programs using standard templates*
- *Construct programs using exceptional handling and collections*
- *Create Models of the system using UML Diagrams*

TOPICS

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Object-Oriented Programming - Features of C++ - I/O Operations, Data Types, Variables-Static, Constants-Pointers-Type Conversions – Conditional and looping statements – Arrays - C++ 11 features - Class and Objects, Abstraction and Encapsulation, Access Specifier, Methods- UML Diagrams Introduction – Use Case Diagram - Class Diagram. (9 hours)

Object-Oriented Programming

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- A computer programming model that organizes software design around data, or objects, rather than functions and logic.
  - An object has unique attributes and behaviour.
- OOP focuses on the objects that developers want to manipulate rather than the logic required to manipulate them.
  - well-suited for programs that are large, complex and actively updated or maintained.
  - Develop programs for manufacturing and design, as well as mobile applications.
    - for example, OOP can be used for manufacturing system simulation software.
- **Aim of OOP:**
  - bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

OOP

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- Beneficial to collaborative development, where projects are divided into groups.
  - Additional benefits of OOP:
    - code reusability, scalability and efficiency.
- First step in OOP:
  - collect all of the objects a programmer wants to manipulate and identify how they relate to each other.
- Examples of an object:
  - physical entities, e.g. a human being who is described by properties like name and address
  - small computer programs, e.g. widgets.
- Once an object is known, it is labelled with a class of objects that defines the kind of data it contains and any logic sequences that can manipulate it. Each distinct logic sequence is known as a method.
- Objects can communicate with well-defined interfaces called messages.

Procedural Programming vs. OOP

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- **Procedural programming**
  - Writing procedures or functions that perform operations on the data.
- **Object-oriented programming**
  - Creating objects that contain both data and functions.
- **Advantages of OOP over procedural programming:**
  - faster and easier to execute
  - provides a clear structure for the programs
  - Makes the code easier to maintain, modify and debug
  - create full reusable applications with less code and shorter development time.

Building blocks of OOP

- **Classes** are user-defined data types that act as the blueprint for individual objects, attributes and methods.
- **Objects** are instances of a class created with specifically defined data.
  - Objects can correspond to real-world objects or an abstract entity.
  - When class is defined initially, the description is the only object that is defined.
- **Methods** are functions that are defined inside a class that describe the behaviours of an object.
  - Each method contained in class definitions starts with a reference to an instance object.
  - Additionally, the subroutines contained in an object are called instance methods. Programmers use methods for reusability or keeping functionality encapsulated inside one object at a time.
- **Attributes** are defined in the class template and represent the state of an object. Objects will have data stored in the attributes field. Class attributes belong to the class itself.

Example

- Class: Fruit
- Objects: Apple, Banana, Mango
- Methods: getData(), displayData(), delete(), add(), etc.
- Attributes: name, color, price, size, etc.

Principles of OOP/ *Features of C++*

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- **Encapsulation.**
  - This principle states that all important information is contained inside an object and only select information is exposed.
  - The implementation and state of each object are privately held inside a defined class.
  - Other objects do not have access to this class or the authority to make changes. They are only able to call a list of public functions or methods.
  - This characteristic of data hiding provides greater program security and avoids unintended data corruption.
- **Abstraction.**
  - Objects only reveal internal mechanisms that are relevant for the use of other objects, hiding any unnecessary implementation code.
  - The derived class can have its functionality extended.
  - This concept can help developers more easily make additional changes or additions over time.

Features of OOP

- **Inheritance.** Classes can reuse code from other classes.
  - Relationships and subclasses between objects can be assigned, enabling developers to reuse common logic while still maintaining a unique hierarchy.
  - This property of OOP forces a more thorough data analysis, reduces development time and ensures a higher level of accuracy.
- **Polymorphism.** Objects are designed to share behaviours and they can take on more than one form.
  - The program will determine which meaning or usage is necessary for each execution of that object from a parent class, reducing the need to duplicate code.
  - A child class is then created, which extends the functionality of the parent class.
  - Polymorphism allows different types of objects to pass through the same interface.

Examples of OOP Languages

- Simula is the first object-oriented programming language
- popular pure OOP languages include:
  - Ruby
  - Scala
  - JADE
  - Emerald
- Programming languages designed primarily for OOP include:
  - Java
  - Python
  - C++
- Other programming languages that pair with OOP include:
  - Visual Basic .NET
  - PHP
  - JavaScript

Benefits of OOP

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- **Modularity**
  - Encapsulation enables objects to be self-contained, making troubleshooting and collaborative development easier.
- **Reusability**
  - Code can be reused through inheritance, meaning a team does not have to write the same code multiple times.
- **Productivity**
  - Programmers can construct new programs quicker through the use of multiple libraries and reusable code.
- **Easily upgradable and scalable**
  - Programmers can implement system functionalities independently.

Benefits of OOP

- **Interface descriptions.**
  - Descriptions of external systems are simple, due to message passing techniques that are used for objects communication.
- **Security.**
  - Using encapsulation and abstraction, complex code is hidden, software maintenance is easier and internet protocols are protected.
- **Flexibility.**
  - Polymorphism enables a single function to adapt to the class it is placed in.
  - Different objects can also pass through the same interface.

Criticism of OOP

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

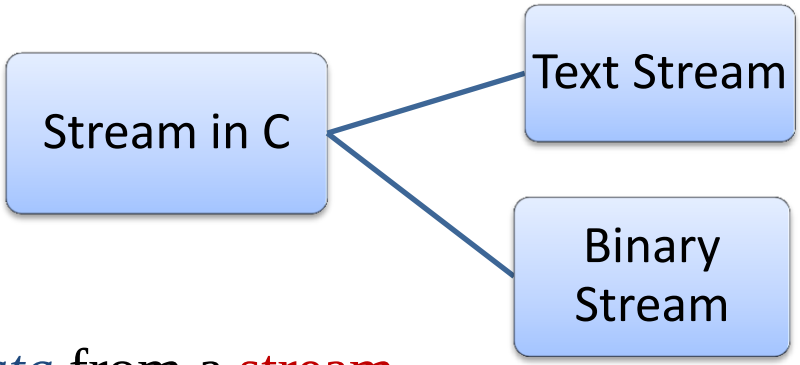
- OOP overemphasizes the data component of software development and does not focus enough on computation or algorithms.
- OOP code may be more complicated to write and take longer to compile.

Alternative methods to OOP

- **Functional programming.**
  - This includes languages such as Erlang and Scala, which are used for telecommunications and fault tolerant systems.
- **Structured or modular programming.**
  - This includes languages such as PHP and C#.
- **Imperative programming.**
  - This alternative to OOP focuses on function rather than models and includes C++ and Java.
- **Declarative programming.**
  - This programming method involves statements on what the task or desired outcome is but not how to achieve it. Languages include Prolog and Lisp.
- **Logical programming.**
  - This method, which is based mostly in formal logic and uses languages such as Prolog, contains a set of sentences that express facts or rules about a problem domain. It focuses on tasks that can benefit from rule-based logical queries.
- **Note:** Most advanced programming languages enable developers to combine models, because they can be used for different programming methods. For example, JavaScript can be used for OOP and functional programming.

I/O Operations

- In C++, I/O operation occurs in streams, which involve transfer of information into byte
- It's a sequences of bytes
- Stream involved in two ways
  - Source
  - Destination of data



- C++ programs *input data* and *output data* from a stream.
- Streams are related with a physical device such as the monitor or with a file stored on the secondary disk.
- In a text stream, the sequence of characters is divided into lines, with each line being terminated with a new-line character (\n) . On the other hand, a binary stream contains data values using their memory representation.

Cascading of Input or Output Operators

- << **operator** – It can use multiple times in the same line.
- Its called **Cascading**
- **cout , cin** can be cascaded
  - For example
    - cout<<“\n Enter the Marks”;
    - cin>> Computer Networks>>OODP;

Reading and Writing Characters and Strings

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

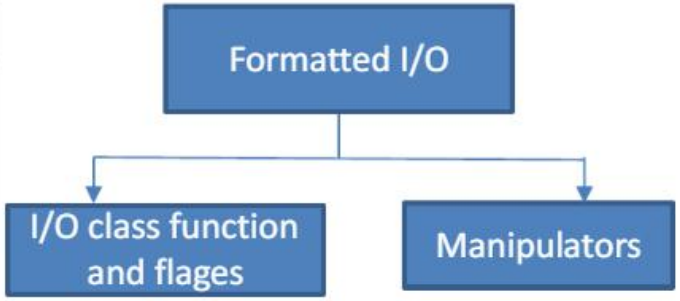
- int marks;
- cin.get(marks);//The value for marks is read

  Or
- marks=cin.get();//A character is read and assigned to marks
- string name;
- cin>>name;
- string empname;
- cin.getline(empname,20);
- cout<<"\n Welcome ,"<<empname

Formatted Input and Output Operations

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Function	Purpose	Syntax	Usage	Result	Comment
width()	Specifies field size (no. of columns) to display the value	cout. width(w)	cout.width(6);cout<<1239;	1239	It can specify field width for only one value
precision()	Specifies no. of digits to be displayed after the decimal point of a float value	cout. precision(d)	cout. precision(3);cout<<1.234567;	1.234	It retains the setting in effect until it is reset
fill()	Specify a character to fill the unused portion of a field	cout.fill(ch)	Cout.fill('#');cout. width(6);cout<<1239;	##1239	It retains the setting in effect until it is reset




```
#include<iostream.h>
#define PI 3.14159
main()
{
cout.precision(3);
cout.width(10);
cout.fill('*');
cout<<PI;
Output
*****3.142
```

Formatting with flags

- The setf() is a member function of the class that is used to set flags for formatting output.

syntax -cout.setf(flag, bit-field)

- Here, flag defined in the ios class specifies how the output should be formatted bit-field is a constant (defined in ios) that identifies the group to which the formatting flag belongs to.
- There are two types of setf()—one that takes both flag and bit-fields and the other that takes only the flag

Flag	Bit-field	Usage	Purpose	Comment
ios :: left	ios :: adjustifield	cout.setf(ios :: left, ios :: adjustifield)	For left justified output	If no flag is set, then by default, the output will be right justified.
ios :: right	ios:: adjustifield	cout.setf(ios :: right, ios :: adjustifield)	For right justified output	
Ios :: internal	ios:: adjustifield	cout.setf(ios :: internal, ios :: adjustifield)	Left-justify sign or base indicator, and right-justify rest of number	
ios :: scientific	ios:: floatifield	cout.setf(ios :: scientific, ios :: floatifield)	For scientific notation	If no flag is set then any of the two notations can be used for floating point numbers depending on the decimal point
ios :: fixed	ios :: floatifield	cout.setf(ios :: fixed, ios :: floatifield)	For fixed point notation	
ios :: dec	ios :: basefield	cout.setf(ios :: dec, ios :: basefield)	Displays integer in decimal	If no flag is set, then the output will be in decimal.
ios :: oct	ios :: basefield	cout.setf(ios :: oct, ios :: basefield)	Displays integer in octal	
ios :: hex	ios :: basefield	cout.setf(ios :: hex, ios :: basefield)	Displays integer in hexadecimal	
ios :: showbase	Not applicable	cout.setf(ios :: showbase)	Show base when displaying integers	
ios :: showpoint	Not applicable	cout.setf(ios :: showpoint)	Show decimal point when displaying floating point numbers	
ios :: showpos	Not applicable	cout.setf(ios :: showpos)	Show + sign when displaying positive integers	
ios :: uppercase	Not applicable	cout.setf(ios :: uppercase)	Use uppercase letter when displaying hexadecimal (OX) or exponential numbers (E)	

Formatting Output Using Manipulators

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

certain manipulators to format the output

Manipulator	Purpose	Usage	Result	Alternative
setw(w)	Specifies field size (number of columns) to display the value	cout<<setw(6) <<1239;	␣ ␣ 1 2 3 9	width()
setprecision(d)	Specifies number of digits to be displayed after the decimal point of a float value	cout<<setprecision(3) <<1.234567;	1.235	precision()
setfill(c)	Specify a character to fill the unused portion of a field	cout<<setfill('#') <<setwidth(6) <<1239;	# # 1 2 3 9	fill()

Data Types in C++

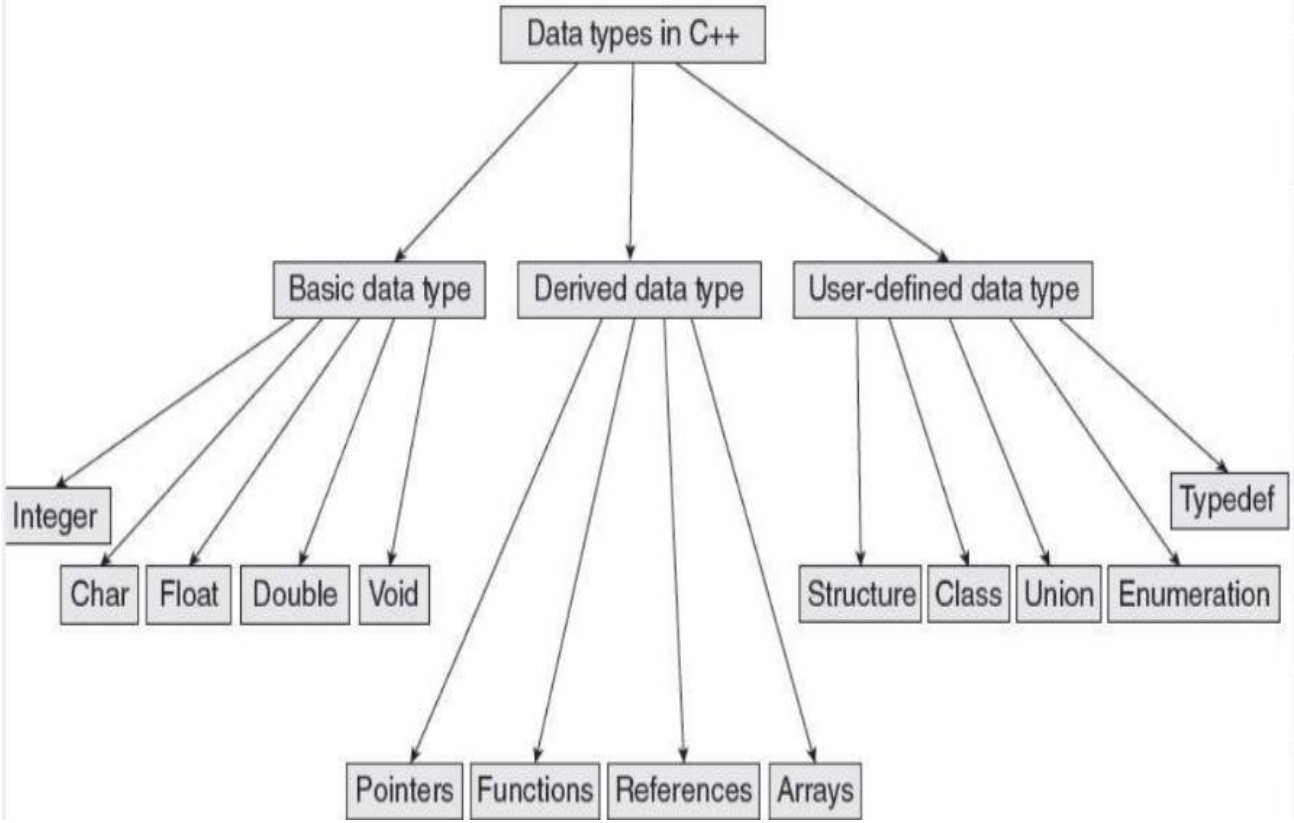
SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- A *type* defines a set of values and a set of operations that can be applied on those values. The set of values for each type is known as the domainfor the type.
- Data types in C++ is mainly divided into two types:

1. Primitive Data Type (directly by the user to declare variables)
    - Integer
    - Character
    - Boolean
    - Floating Point
    - Double Floating Point
    - Valueless or Void
    - Wide Character
    - Also : String
2. Abstract or User Defined Data type (data types are defined by user itself)

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Data Types in C++



Data type	Size in bytes	Range
Char	1	-128 to 127
unsigned char	1	0 to 255
signed char	1	-128 to 127
Int	2	-32768 to 32767
unsigned int	2	0 to 65535
signed short int	2	-32768 to 32767
signed int	2	-32768 to 32767
short int	2	-32768 to 32767
unsigned short int	2	0 to 65535
long int	4	-2147483648 to 214748364
unsigned long int	4	0 to 4294967295
signed long int	4	-2147483648 to 214748364
Float	4	3.4E-38 to 3.4E+38
Double	8	1.7E-308 to 1.7E+308

Variables

- A variable is the content of a memory location that stores a certain value. A variable is identified or denoted by a variable name.
- The variable name contains a sequence of one or more letters, digits or underscore, for example: empname

Rules for defining variable name:

- A variable name can have one or more letters or digits or underscore for example character.
- White space, punctuation symbols or other characters are not permitted to denote variable name.
- A variable name must begin with a letter.
- Variable names cannot be keywords or any reserved words of the C++ programming language.
- Data C++ is a case-sensitive language. Variable names written in capital letters differ from variable names with the same name but written in small letters

Types of Variables

SRM INSTITUTE OF SCIENCE & TECHNOLOGY (Deemed to be University u/s 3 of UGC Act, 1956)

Variables

Local Variables

Local variable: These are the variables which are declared within the method of a class.

Example:

```
public class Car {
public:
 void display(int m){  //
Method
        int model=m;
// Created a local variable
model
        cout<<model;
   }
```

Instance Variables

Instance variable: These are the variables which are declared in a class but outside a method, constructor or any block.

Example:

```
public class Car {
     private: String color;
 //  Created an instance
variable color
 Car(String c)
  {
   color=c;
  }}
```

Static Variables

Static variables: Static variables are also called as class variables. These variables have only one copy that is shared by all the different objects in a class.

Example:

```
public class Car {
     public static int tyres;
// Created a class variable
void init(){
        tyres=4;
} }
```

Constant Variables

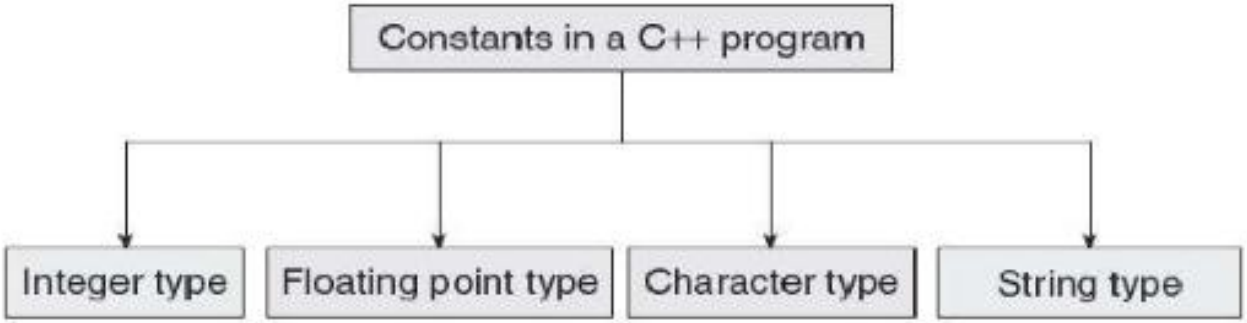
Constant is something that doesn't change. In C language and C++ we use the keyword const to make program elements constant.

Example:

```
const int i = 10;
void f(const int i)
class Test
{
   const int i;
};
```

Constants

- Constants are identifiers whose value does not change. While variables can change their value at any time, constants can never change their value.
- Constants are used to define fixed values such as Pi or the charge on an electron so that their value does not get changed in the program even by mistake.
- A constant is an explicit data value specified by the programmer.
- The value of the constant is known to the compiler at the compile time



Declaring Constants

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- Rule 1 : Constant names are usually written in capital letters to visually distinguish them from other variable names which are normally written in lower case characters.
- Rule 2: No blank spaces are permitted in between the # symbol and define keyword. E.g. #define PI 3.14
- Rule 3: Blank space must be used between #define and constant name and between constant name and constant value.
- Rule 4: #define is a preprocessor compiler directive and not a statement. Therefore, it does not end with a semi-colon.

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Data Operators

Arithmetic Operators

1 Arithmetic Operators

2 Unary operators

3 Relational operators:

4 Logical Operators

+	Add two operands or unary plus	>> 2 + 3 5 >> +2
-	Subtract two operands or unary subtract	>> 3 – 1 2 >> -2
*	Multiply two operands	>> 2 * 3 6
/	Divide left operand with the right and result is in float	>> 6 / 3 2.0
%	Remainder of the division of left operand by the right	>> 5 % 3 2

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Unary operators

1. Arithmetic Operators
2. Unary operators
3. Relational operators:
4. Logical Operators

++

X++ - Post Increment
++X – Pre Increment

```
>> x = 5
>> ++x
>> print(x)
6
```

--

X-- - Post Decrement
--X – Pre Increment

```
>> x--
>> print(x)
4
```

Relational operators

1 Arithmetic Operators

2 Unary operators

3 Relational operators:

4 Logical Operators

>	True if left operand is greater than the right	>> 2 > 3 False
<	True if left operand is less than the right	>> 2 < 3 True
==	True if left operand is equal to right	>>2 == 2 True
!=	True if left operand is not equal to the right	>> x >>= 2 >>print(x) 1

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Logical operators

1 Arithmetic Operators

2 Unary operators

3 Relational operators:

4 Logical Operators

and

Returns True if both x and y are True, False otherwise

```
>> True
&& False
False
```

or

Returns True if at least x or y are True, False otherwise

```
>> True ||
False
True
```

not

Returns True if x is False, True otherwise

```
>>! 1
False
```

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Special Operators

1

Scope resolution operator

In C, the global version of a variable cannot be accessed from within the inner block. C++ resolves this problem by using scope resolution operator (::), because this operator allows access to the global version of a variable.

2

New Operator

The new operator denotes a request for memory allocation on the Heap. If sufficient memory is available, new operator initializes the memory and returns the address of the newly allocated and initialized memory to the pointer variable.

3

Delete Operator

Since it is programmer's responsibility to deallocate dynamically allocated memory, programmers are provided delete operator by C++ language.

4

Member Operator

C++ permits us to define a class containing various types of data & functions as members. To access a member using a pointer in the object & a pointer to the member.

Pointers

- Pointers are symbolic representation of addresses.
- They enable programs to simulate call-by-reference as well as to create and manipulate dynamic data structures.

Syntax data_type *pointer_variable;

- Example

int *p,sum

Assignment

- integer type pointer can hold the address of another int variable
- To assign the address of variable to pointer-ampersand symbol (&)

p=∑

this pointer holds the address of current object

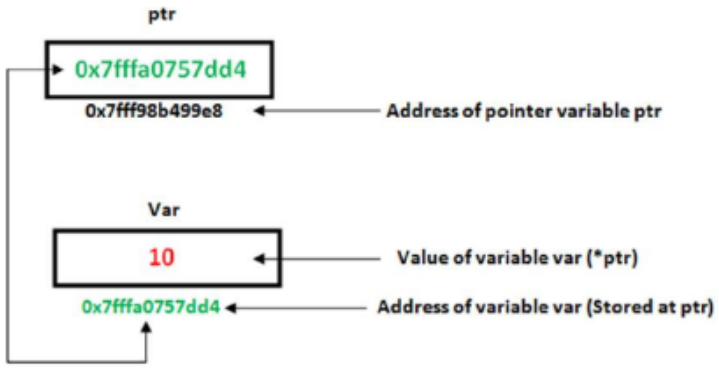
- int num;
- this->num=num;

Void?

- When used in the declaration of a pointer, void specifies that the pointer is "universal." If a pointer's type is void* , the pointer can point to any variable that's not declared with the const or volatile keyword

- P=∑//assign address of another variable
- cout<<∑ //to print the address of variable
- cout<<p;//print the value of variable
- Example of pointer

```
#include<iostream.h>
using namespace std;
int main()
{ int *p,sum=10;
p=&sum;
cout<<"Address of sum:"<<&sum<<endl;
cout<<"Address of sum:"<<p<<endl;
cou<<"Address of p:"<<&p<<endl;
cout<<"Value of sum"<<*p;
}
```



Output:
Address of sum : 0X77712
Address of sum: 0x77712
Address of p: 0x77717
Value of sum: 10

Type Conversions

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- A type cast is basically a conversion from one type to another. There are two types of type conversion:
- **Implicit Type Conversion** Also known as 'automatic type conversion'.
  - Done by the compiler on its own, without any external trigger from the user.
  - Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.
  - All the data types of the variables are upgraded to the data type of the variable with largest data type.bool -> char -> short int -> int -> unsigned int -> long -> unsigned -> long long -> float -> double -> long double
  - It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

Type Conversions

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- // An example of implicit conversion

```
1.  #include <iostream>
2.  using namespace std;
3.  int main()
4.  {
5.      int x = 10; // integer x
6.      char y = 'a'; // character c
7.
8.      // y implicitly converted to int. ASCII
9.      // value of 'a' is 97
10.     x = x + y;
11.
12.     // x is implicitly converted to float
13.     float z = x + 1.0;
14.
15.     cout << "x = " << x << endl
16.          << "y = " << y << endl
17.          << "z = " << z << endl;
18.
19.     return 0;
20. }
```

Output: x = 107 y = a z = 108

Type Conversions

- **Explicit Type Conversion**: This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type. In C++, it can be done by two ways:
- **Converting by assignment:** This is done by explicitly defining the required type in front of the expression in parenthesis. This can be also considered as forceful casting.
- **Syntax:**

(type) expression

- where *type* indicates the data type to which the final result is converted.

```
// C++ program to demonstrate
// explicit type casting
 #include <iostream>
using namespace std;
 int main()
{
    double x = 1.2;
     // Explicit conversion from double to int
    int sum = (int)x + 1;
     cout << "Sum = " << sum;
     return 0;
}
```

Output: Sum = 2

Type Conversions

- **Conversion using Cast operator:** A Cast operator is an **unary operator** which forces one data type to be converted into another data type.
  C++ supports four types of casting:
  - Static Cast
  - Dynamic Cast
  - Const Cast
  - Reinterpret Cast
- **Example:**

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5.     float f = 3.5;
6.      // using cast operator
7.     int b = static_cast<int>(f);
8.       cout << b;
9. }
```

Output: 3

Conditional and looping statements

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- QUIZ ACTIVITY

Arrays

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- QUIZ ACTIVITY

Class and Objects

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- Class :
  - A user-defined data type that we can use in our program, and it works as an object constructor, or a "blueprint" for creating objects.
- For example:
  - car : an **object**.
  - **Attributes:** weight and color
  - **Methods:** drive, apply_brake
- Attributes: variables, methods: functions
  - Attributes and methods are often referred to as "class members".

Create a Class

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- use the **class** keyword:
- **Example**
  - Create a class called "MyClass":
- class MyClass { // The class
  public: // Access specifier
  int room_num; // Attribute (int variable)
  string dept; // Attribute (string variable)
  };

Explanation

- **public**: specifies that members (attributes and methods) of the class are accessible from outside the class.
- The class consists of an integer variable **room_num** and a string-type variable **dept**.
- When variables are declared within a class, they are called **attributes**.
- The class definition ends with a semicolon (;)

Create an Object

- To create an object of MyClass, specify the class name, followed by the object name.
- To access the class attributes, use the object name, dot operator (.) and variable/attribute name.

- **Example**
  - Create an object called "myObj" and access the attributes:

```
class MyClass {       // The class
  public:             // Access specifier
    int room_num;       // Attribute (int variable)
    string dept;  // Attribute (string variable)
};
int main() {
  MyClass myObj;  // Create an object of MyClass
  // Access attributes and set values
  myObj.room_num = 602;
  myObj.dept = ”dsbs";

  // Print attribute values
  cout << myObj.room_num << "\n";
  cout << myObj.dept;
  return 0;
}
```

Access Specifiers

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- three access specifiers:
  - **public** - members are accessible from outside the class
  - **private** - members cannot be accessed (or viewed) from outside the class
  - **protected** - members cannot be accessed from outside the class, however, they can be accessed in inherited classes.

Example

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- class MyClass {
  public: // Public access specifier
  int x; // Public attribute
  private: // Private access specifier
  int y; // Private attribute
  };

  int main() {
  MyClass myObj;
  myObj.x = 25; // Allowed (public)
  myObj.y = 50; // Not allowed (private)
  return 0;
  }

- If you try to access a private member, an error occurs:

  error: y is private

Methods

- Methods are **functions** that belongs to the class.
- There are two ways to define functions that belongs to a class:
  - Inside class definition
  - Outside class definition
- Example
  - define a function inside the class, and name it "myMethod".
- **Note:** access methods by creating an object of the class, using the dot operator (.), and the method name.

Inside the class

```
class MyClass {        // The class
  public:              // Access specifier
    void myMethod() {
// Method inside the class
      cout << "Hello World!";
    }
};

int main() {
  MyClass myObj;     // Create an object of MyClass
  myObj.myMethod();  // Call the method
  return 0;
}
```

Outside the class

- To define a function outside the class definition, declare the method inside the class and then define it outside of the class.
- Define outside by specifying the name of the class, followed the scope resolution :: operator, followed by the name of the function.

Example

```
class MyClass {        // The class
  public:              // Access specifier
    void myMethod();   // Method/function declaration
};

// Method/function definition outside the class
void MyClass::myMethod() {
  cout << "Hello World!";
}

int main() {
  MyClass myObj;     // Create an object of MyClass
  myObj.myMethod();  // Call the method
  return 0;
}
```

Abstraction and Encapsulation

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- **Abstraction:**
  - Displaying only essential information and hiding the details.
  - Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.
  - In C++, an abstraction is implemented by using **a class, a template, or a function**.

  Example:
    - when we are driving a car, we are only concerned about driving the car like start/stop the car, accelerate/ break, etc.
    - We do not know how on pressing the accelerator the speed is actually increasing, do not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car.

Types of Abstraction

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- **Data abstraction**
  - This type only shows the required information about the data and hides the unnecessary data.
- **Control Abstraction**
  - This type only shows the required information about the implementation and hides unnecessary information.

How do we implement abstraction?

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- We can implement Abstraction in C++ using classes.
- The class helps us to group data members and member functions using available access specifiers.
- A class can decide which data member will be visible to the outside world and which is not.
  - Use the access specifiers : public and private.

- **Example:**
  - the members that define the internal implementation can be marked as **private** in a class.
  - the important information needed to be given to the outside world can be marked as **public**.
  - the public members can access the private members as they are inside the class.

Abstraction in Header files

- One more type of abstraction in C++ can be header files.
- For example,
  - consider the pow() method present in math.h header file.
  - Whenever we need to calculate the power of a number, we simply call the function pow() present in the math.h header file and pass the numbers as arguments without knowing the underlying algorithm according to which the function is actually calculating the power of numbers.

Example 1

```
// C++ Program to Demonstrate the
// working of Abstraction
#include <iostream>
using namespace std;

Class AbstractionDemo {
private:
    int a, b;

public:
    // method to set values of
    // private members
    void set(int x, int y)
    {
        a = x;
        b = y;
    }
    void display()
    {
        cout << "a = " << a << endl;
        cout << "b = " << b << endl;
    }
};

int main()
{
    AbstractionDemo obj;
    obj.set(10, 20);
    obj.display();
    return 0;
}
```

Output: a = 10 b = 20

we are not allowed to access the variables a and b directly. However, one can call the function set() to set the values in a and b and the function display() to display the values of a and b.

Advantages of Data Abstraction

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- Helps the user to avoid writing the low-level code
- Avoids code duplication and increases reusability.
- Can change the internal implementation of the class independently without affecting the user.
- Helps to increase the security of an application or program as only important details are provided to the user.
- It reduces the complexity as well as the redundancy of the code, therefore increasing the readability.

Encapsulation

- Hide the sensitive data from users.
- To achieve this, we declare class variables/attributes as **private** so that it cannot be accessed from outside the class.
  - To allow others to read or modify the value of a private member, we provide *public* **get** and **set** methods.
- Encapsulation ensures better control of your data, because you (or others) can change one part of the code without affecting other parts.
- It provides an increased security of data.

Example

```
#include <iostream>
    using namespace std;

    class Employee {
      private:
        // Private attribute
        int salary;

      public:
        // Setter
        void setSalary(int s) {
          salary = s;
        }
        // Getter
        int getSalary() {
          return salary;
        }
    };
```

```
int main() {
      Employee myObj;
      myObj.setSalary(50000);
      cout << myObj.getSalary();
      return 0;
    }
```

Explanation

- The salary attribute is **private**, it has restricted access.
- The **public setSalary()** method takes a parameter (s) and assigns it to the salary attribute (salary = s).
- The **public getSalary()** method returns the value of the private salary attribute.
- Inside main(), we create an object of the Employee class.
  - Now we can use the setSalary() method to set the value of the private attribute to 50000.
  - Then we call the getSalary() method on the object to return the value.

UML Diagrams Introduction

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- UML is **not a programming language**, it is rather a **visual language**.
  - We use UML diagrams to portray the **behaviour and structure** of a system.
- UML helps software engineers, businessmen and system architects with modelling, design and analysis.
- The Object Management Group (OMG) adopted Unified Modelling Language as a standard in 1997.
- International Organization for Standardization (ISO) published UML as an approved standard in 2005.

Need for UML Diagrams

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- Complex applications need collaboration and planning from multiple teams
  - hence it requires a clear and concise way to communicate amongst them.
- UML becomes essential to communicate the following aspects with non programmers
  - essential requirements, functionalities and processes of the system.
- A lot of time is saved down the line when teams are able to visualize processes, user interactions and static structure of the system.

Classification of UML Diagrams

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- **Structural Diagrams**
  - Capture static aspects or structure of a system.
  - Component Diagrams, Object Diagrams, Class Diagrams and Deployment Diagrams.
- **Behaviour Diagrams**
  - Capture dynamic aspects or behaviour of the system.
  - Use Case Diagrams, State Diagrams, Activity Diagrams and Interaction Diagrams.

Use Case Diagram

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

- Use-case diagrams model the high-level functions and scope of a system.
  - Helps to identify the interactions between the system and its actors.
- The use cases and actors in use-case diagrams describe what the system does and how the actors use it.
  - but not how the system operates internally.

Elements of use case diagram

- **Use cases**

  A use case describes a function that a system performs to achieve the user's goal. A use case must yield an observable result that is of value to the user of the system.
- **Actors**
  An actor represents a role of a user that interacts with the system that you are modeling.
  - The user can be a human user, an organization, a machine, or another external system.
- **Subsystems**
  In UML models, subsystems are a type of stereotyped component that represent independent, behavioural units in a system.
  - Subsystems are used in class, component, and use-case diagrams to represent large-scale components in the system that you are modeling.
- **Relationships in use-case diagrams**
  In UML, a relationship is a connection between model elements. A UML relationship is a type of model element that adds semantics to a model by defining the structure and behaviour between the model elements.

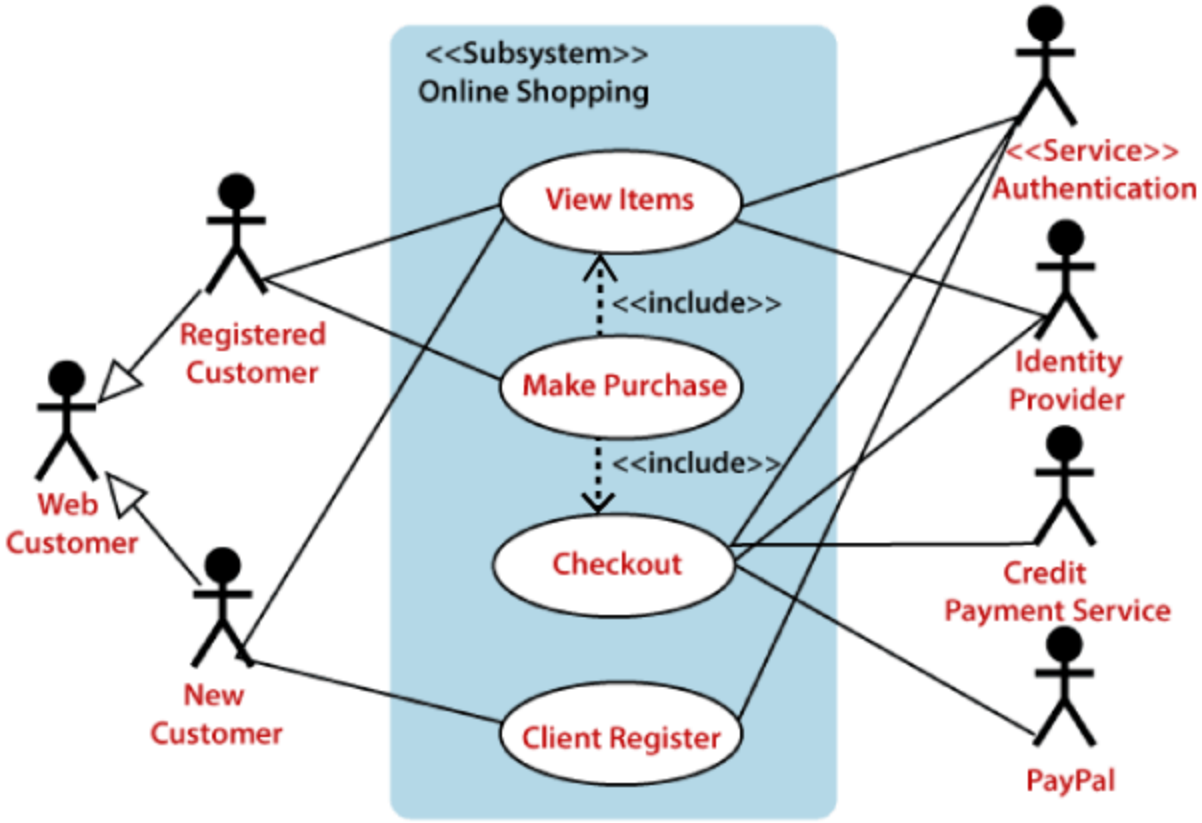
Steps

- Analyze the whole system before starting with drawing a use case diagram
- Find the system's functionalities
- List the actors that interact with the system
- Identify the relationship between the actor and use case/ system.
- Basically, an actor can interact multiple times with a use case or system at a particular instance of time.

Example

- Create a use case diagram to depict the Online Shopping website.
  - the Web Customer actor makes use of any online shopping website to purchase online.
  - The top-level uses are as follows:
    - View Items, Make Purchase, Checkout, Client Register.
  - The **View Items** use case is utilized by the customer who searches and view products.
  - The **Client Register** use case allows the customer to register itself with the website for availing gift vouchers, coupons, or getting a private sale invitation.
  - the **Checkout** is an included use case, which is part of **Making Purchase,** and it is not available by itself.

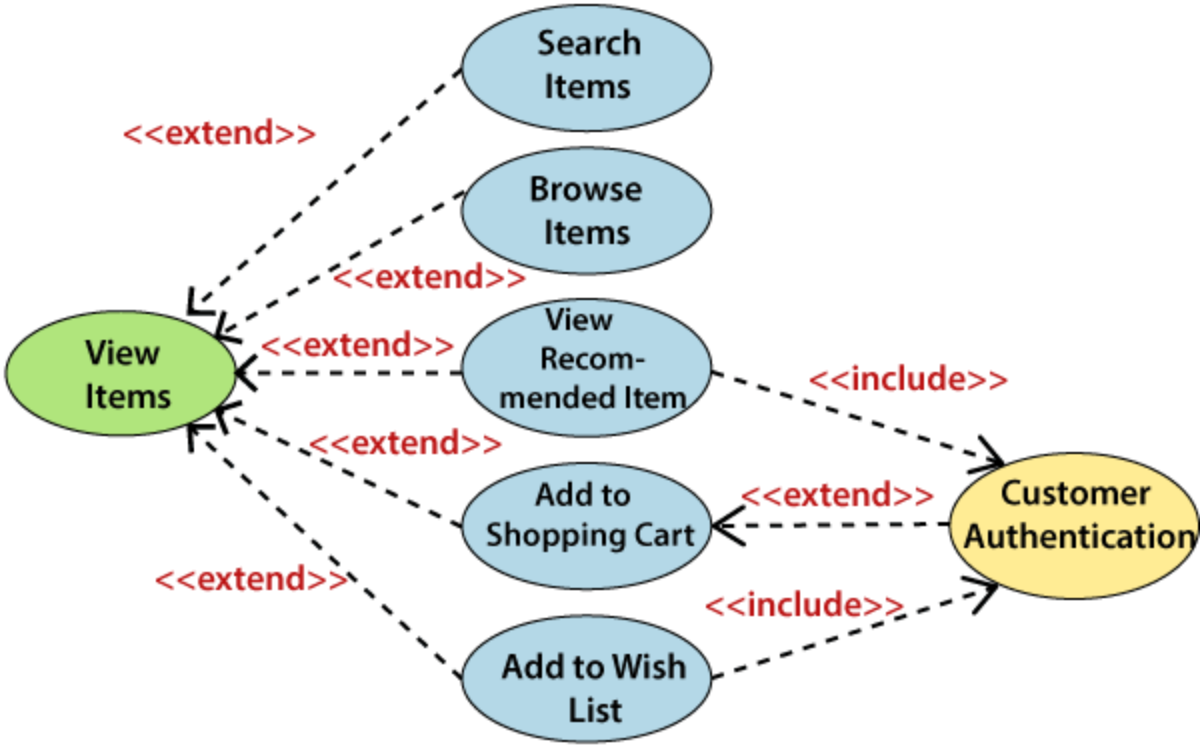
Example


<<Subsystem>>
Online Shopping
View Items
<<include>>
Make Purchase
<<include>>
Checkout
Client Register
Registered Customer
Web Customer
New Customer
<<Service>>
Authentication
Identity Provider
Credit Payment Service
PayPal

Use case: View Items

- The **View Items** is further extended by several use cases such as.,
  - Search Items, Browse Items, View Recommended Items, Add to Shopping Cart, Add to Wish list.
  - All of these extended use cases provide some functions to customers, which allows them to search for an item.
- Both **View Recommended Item** and **Add to Wish List** include the Customer Authentication use case, as they necessitate authenticated customers.
  - simultaneously item can be added to the shopping cart without any user authentication.

Use case: View Items

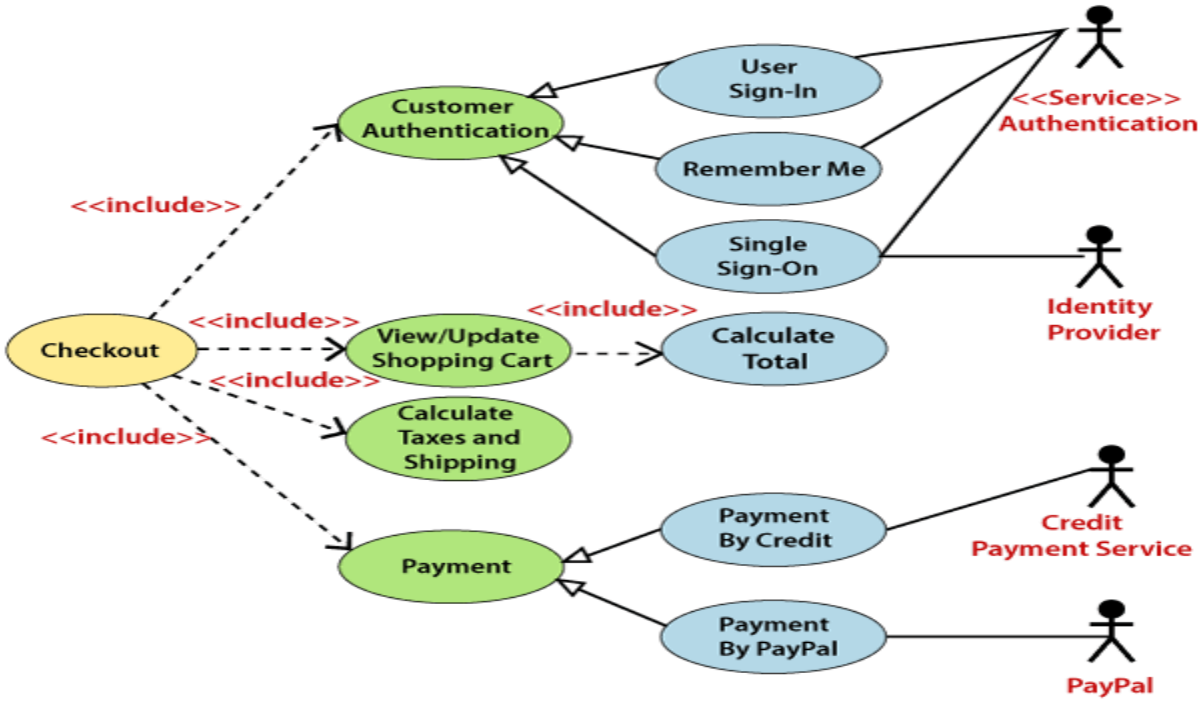


Use case: Checkout

- the **Checkout** use case also includes the following use cases:
  - It requires an authenticated Web Customer, which can be done by login page, user authentication cookie ("Remember me"), or Single Sign-On (SSO).
  - SSO needs an external identity provider's participation.
  - Web site authentication service is utilized in all these use cases.

Use case: Checkout

- The Checkout use case involves Payment use case that can be done either by the credit card and external credit payment services or with PayPal.



Class diagram

- It depicts the static structure of a system by showing system's classes, interfaces, methods, attributes, associations, collaboration, and constraints imposed in the relationships.
- Helps to identify relationship between different classes or objects.

Purpose of the class diagram

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

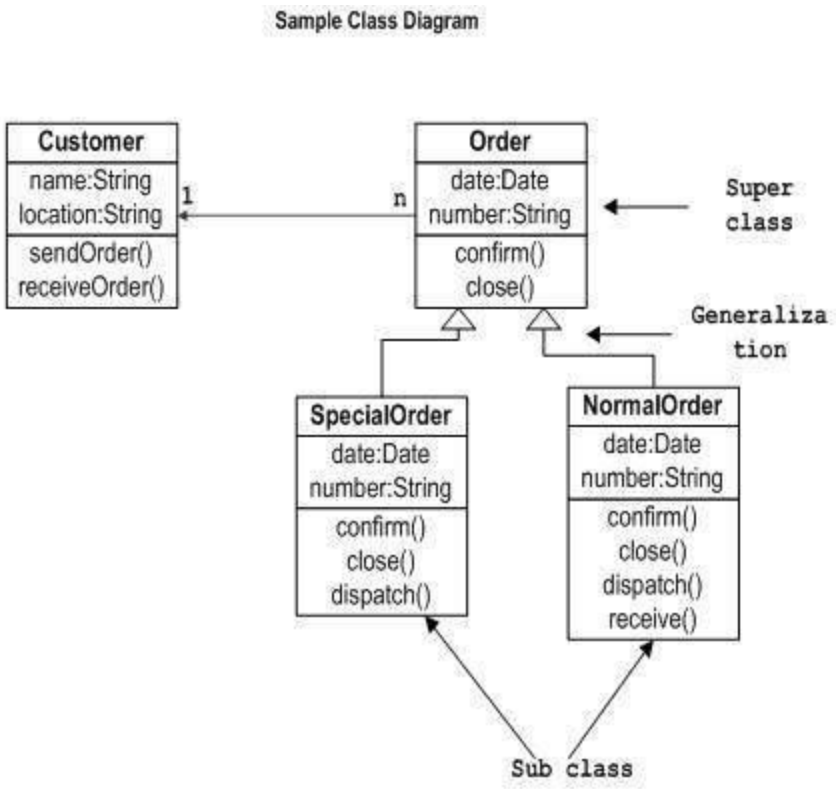
Steps

- Identify the meaningful name of the class diagram.
- Each element and their relationships should be identified in advance.
- Responsibility (attributes and methods) of each class should be clearly identified.
- For each class, minimum number of properties should be specified, as unnecessary properties will make the diagram complicated.
- Use notes whenever required to describe some aspect of the diagram. At the end of the drawing it should be understandable to the developer/coder.
- Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.

Example

- Let us consider an example of an Order System of an application. It describes a particular aspect of the entire application.
  - First of all, Order and Customer are identified as the two elements of the system.
    - They have a one-to-many relationship because a customer can have multiple orders.
  - Order class is an abstract class and it has two concrete classes (inheritance relationship) SpecialOrder and NormalOrder.
  - The two inherited classes have all the properties as the Order class.
    - In addition, they have additional functions like dispatch () and receive ().

Example


Sample Class Diagram
Customer
name:String
location:String
sendOrder()
receiveOrder()
1
n
Order
date:Date
number:String
confirm()
close()
Super class
Generalization
SpecialOrder
date:Date
number:String
confirm()
close()
dispatch()
NormalOrder
date:Date
number:String
confirm()
close()
dispatch()
receive()
Sub class