IACSD

Frequently Asked Questions
PG-DAC

Institute for Advanced Computing and Software Development
Dr. D.Y. Patil Educational Complex Sector 29,Near Akurdi Railway Station,
Nigdi, Pradhikaran,Akurdi,Pune-44
www.iacsd.com
Phone No: 9607690988

INDEX

CPP Interview Questions

1) What is the difference between C and C++?

Following are the differences between C and C++:

C	C++
C language was developed by Dennis Ritchie.	C++ language was developed by Bjarne Stroustrup.
C is a structured programming language.	C++ supports both structural and object-oriented programming language.
C is a subset of C++.	C++ is a superset of C.
In C language, data and functions are the free entities.	In the C++ language, both data and functions are encapsulated together in the form of a project.
C does not support the data hiding. Therefore, the data can be used by the outside world.	C++ supports data hiding. Therefore, the data cannot be accessed by the outside world.
C supports neither function nor operator overloading.	C++ supports both function and operator overloading.
In C, the function cannot be implemented inside the structures.	In the C++, the function can be implemented inside the structures.
Reference variables are not supported in C language.	C++ supports the reference variables.
C language does not support the virtual and friend functions.	C++ supports both virtual and friend functions.
In C, scanf() and printf() are mainly used for input/output.	C++ mainly uses stream cin and cout to perform input and output operations.

2) What is the difference between struct and class?

Structures	Class
A structure is a user-defined data type which contains variables of dissimilar data types.	The class is a user-defined data type which contains member variables and member functions.
The variables of a structure are stored in the stack memory.	The variables of a class are stored in the heap memory.
We cannot initialize the variables directly.	We can initialize the member variables directly.
If access specifier is not specified, then by default the access specifier of the variable is "public".	If access specifier is not specified, then by default the access specifier of a variable is "private".
The instance of a structure is a "structure variable".	

Declaration of a structure:
struct structure_name
{
// body of structure;
} ;
	Declaration of class:
	class class_name
	{
	// body of class;
	}

A structure is declared by using a struct keyword.	The class is declared by using a class keyword.
The structure does not support the inheritance.	The class supports the concept of inheritance.
The type of a structure is a value type.	The type of a class is a reference type.

3)What is the difference between reference and pointer?

Following are the differences between reference and pointer:

Reference	Pointer
Reference behaves like an alias for an existing variable, i.e., it is a temporary variable.	The pointer is a variable which stores the address of a variable.

Reference variable does not require any indirection operator to access the value. A reference variable can be used directly to access the value.	Pointer variable requires an indirection operator to access the value of a variable.
Once the reference variable is assigned, then it cannot be reassigned with different address values.	The pointer variable is an independent variable means that it can be reassigned to point to different objects.
A null value cannot be assigned to the reference variable.	A null value can be assigned to the reference variable.
It is necessary to initialize the variable at the time of declaration.	It is not necessary to initialize the variable at the time of declaration.

4) Explain oops concept with real time example.

- **Objects**
- **Classes**
- **Abstraction**
- **Encapsulation**
- **Polymorphism**

Objects: **(BY Steve Jobs):** Objects are like people. They're living, breathing things that have knowledge inside them about how to do things and have memory inside them so they can remember things. And can interact with them at high level of abstraction.

Example : : If I'm your laundry object, you can give me your dirty clothes and send me a message that says, "Can you get my clothes laundered, please." I happen to know where the best laundry place in San Francisco is. And I speak English, and I have dollars in my pockets. So I go out and hail a taxicab and tell the driver to take me to this place in San Francisco. I go get your clothes laundered, I jump back in the cab, I get back here. I give you your clean clothes and say, "Here are your clean clothes."

You have no idea how I did that. You have no knowledge of the laundry place. Maybe you speak French, and you can't even hail a taxi. You can't pay for one, you don't have dollars in your pocket. Yet, I knew how to do all of that. And you didn't have to know any of it. All that complexity was hidden inside of me, and we were able to interact at a very high level of

abstraction. That's what objects are. They encapsulate complexity, and the interfaces to that complexity are high level.

Classes: Classes are data types based on which objects are created .Objects with similar properties and methods are grouped together to form a Class. Thus a Class represents a set of individual objects. Characteristics of an object are represented in a class as Properties. The actions that can be performed by objects become functions of the class or Methods.

Example: An architect will have the blueprints for a house. Those blueprints will be plans that explain exactly what properties the house will have and how they are all laid out. However it is just the blueprint, you can't live in it. Builders will look at the blueprints and use those blueprints to make a physical house. They can use the same blueprint to make as many houses as they want....each house will have the same layout and properties. Each house can accommodate it's own families...so one house might have the Smiths live in it, one house might have the Jones live in it.(so here **The blueprint is the class...the house is the object. The people living in the house are data stored in the object's properties.)**

Abstraction: Abstraction means showing essential features and hiding non-essential features to the user.

Exmple:

1) Yahoo Mail... When you provide the user name and password and click on submit button..It will show Compose,Inbox,Outbox,Sent mails...so and so when you click on compose it will open...but user doesn't
know what are the actions performed internally....It just Opens....that is essential; User doesn't know internal actions ...that is non-essential things...

2) TV Remote… Remote is a interface between user and tv..right. which has buttons like 0 to 10 ,on /of etc but we don't know circuits inside remote...User does not need to know..Just he is using essential thing that is remote.

Encapsulation: Encapsulation means which binds the data and code (or) writing operations and methods in single unit (class).

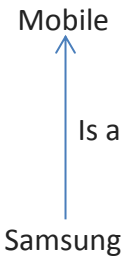
Example:
A car is having multiple parts..like steering,wheels,engine...etc..which binds together to form a

single object that is car. So, Here multiple parts of cars encapsulates itself together to form a single object that is Car.

In real time we are using Encapsulation for security purpose...

Inheritance:

The mechanism of forming hierarchy, creating a new class (sub class) from base/super class is called inheritance. When we inherit all the features from base/super class will be in in the sub class, subclass can also add new features for itself.



Example: Basic Mobile functionality is to Send Message, dial & receive call. So the brands of mobile is using this basic functionality by extending the mobile class functionality and adding their own new features to their respective brand.

Polymorphism:

Polymorphism means ability to take more than one form that an operation can exhibit different behavior at different instance depend upon the data passed in the operation.

Example:

1. We behave differently in front of elders, and friends. A single person is behaving differently at different time.

2. A software engineer can perform different task at different instance of time depending on the task assigned to him .He can done coding , testing , analysis and designing depending on the task assign and the requirement.

3. Consider the stadium of common wealth games. Single stadium but it perform multiple task like swimming, lawn tennis etc.

4. If a girl is married and is the mother of 2 children, and is teaching, then she is a women first ,a teacher in a school when she is in school, wife of someone at home, mother of her children, and obviously daughter of someone. This means a woman plays different roles at different times and this is polymorphism (many forms).

5)Storage Classes in C

Storage classes in C are used to determine the lifetime, visibility, memory location, and initial value of a variable. There are four types of storage classes in C

- Automatic
- External
- Static
- Register

Storage Classes	Storage Place	Default Value	Scope	Lifetime
auto	RAM	Garbage Value	Local	Within function
extern	RAM	Zero	Global	Till the end of the main program Maybe declared anywhere in the program
static	RAM	Zero	Local	Till the end of the main program, Retains value between multiple functions call
register	Register	Garbage Value	Local	Within the function

6)Call by value and call by reference in C++

There are two ways to pass value or data to function in C language: call by value and call by reference. Original value is not modified in call by value but it is modified in call by reference.

Call by value

In call by value, **original value is not modified.**

In call by value, value being passed to the function is locally stored by the function parameter in stack memory location. If you change the value of function parameter, it is changed for the current function only. It will not change the value of variable inside the caller method such as main().

Example:

```
#include <iostream>
using namespace std;
void change(int data);
int main()
```

```
{
int data = 3;
change(data);
cout << "Value of the data is: " << data<< endl;
return 0;
}
void change(int data)
{
data = 5;
}
```

Call by reference in C++

In call by reference, original value is modified because we pass reference (address).

Here, address of the value is passed in the function, so actual and formal arguments share the same address space. Hence, value changed inside the function, is reflected inside as well as outside the function.

Note: To understand the call by reference, you must have the basic knowledge of pointers.

Let's try to understand the concept of call by reference in C++ language by the example given below:

```
#include<iostream>

using namespace std;
void swap(int *x, int *y)
{
 int swap;
 swap=*x;
 *x=*y;
 *y=swap;
}
int main()
{
 int x=500, y=100;
 swap(&x, &y);  // passing value to function
 cout<<"Value of x is: "<<x<<endl;
 cout<<"Value of y is: "<<y<<endl;
 return 0;
}
```

Output:

```
Value of x is: 100
Value of y is: 500
```

7)Difference between call by value and call by reference in C++

No.	Call by value	Call by reference
1	A copy of value is passed to the function	An address of value is passed to the function
2	Changes made inside the function is not reflected on other functions	Changes made inside the function is reflected outside the function also
3	Actual and formal arguments will be created in different memory location	Actual and formal arguments will be created in same memory location

C++ static

In C++, static is a keyword or modifier that belongs to the type not instance. So instance is not required to access the static members. In C++, static can be field, method, constructor, class, properties, operator and event. **Advantage of C++ static keyword**

Memory efficient: Now we don't need to create instance for accessing the static members, so it saves memory. Moreover, it belongs to the type, so it will not get memory each time when instance is created.

8)Shallow Copy

- The default copy constructor can only produce the shallow copy.
- A Shallow copy is defined as the process of creating the copy of an object by copying data of all the member variables as it is.

Let's understand this through a simple example:

```
#include <iostream>

using namespace std;

class Demo
{
   int a;
   int b;
   int *p;
   public:
   Demo()
   {
      p=new int;
   }
```

```
  void setdata(int x,int y,int z)
  {
     a=x;
     b=y;
     *p=z;
  }
  void showdata()
  {
     std::cout << "value of a is : " <<a<< std::endl;
     std::cout << "value of b is : " <<b<< std::endl;
     std::cout << "value of *p is : " <<*p<< std::endl;
  }
};


int main()
{
  Demo d1;
  d1.setdata(4,5,7);
  Demo d2 = d1;
  d2.showdata();
   return 0;  }
```

Output:

```
value of a is : 4
value of b is : 5
value of *p is : 7
```

In the above case, a programmer has not defined any constructor, therefore, the statement **Demo d2 = d1;** calls the default constructor defined by the compiler. The default constructor creates the exact copy or shallow copy of the existing object. Thus, the pointer p of both the objects point to the same memory location. Therefore, when the memory of a field is freed, the memory of another field is also automatically freed as both the fields point to the same memory location. This problem is solved by the **user-defined constructor** that creates the **Deep copy**.

9)Deep copy

Deep copy dynamically allocates the memory for the copy and then copies the actual value, both the source and copy have distinct memory locations. In this way, both the source and copy

are distinct and will not share the same memory location. Deep copy requires us to write the user-defined constructor.

Let's understand this through a simple example.

```
#include <iostream>
using namespace std;
class Demo
{
   public:
   int a;
   int b;
   int *p;

   Demo()
   {
      p=new int;
   }
   Demo(Demo &d)
   {
      a = d.a;
      b = d.b;
      p = new int;
      *p = *(d.p);
   }
   void setdata(int x,int y,int z)
   {
      a=x;
      b=y;
      *p=z;
   }
   void showdata()
   {
      std::cout << "value of a is : " <<a<< std::endl;
      std::cout << "value of b is : " <<b<< std::endl;
      std::cout << "value of *p is : " <<*p<< std::endl;
   }
};
int main()
{
  Demo d1;
  d1.setdata(4,5,7);
```

```
    Demo d2 = d1;
    d2.showdata();
    return 0;
}
```

Output:

```
value of a is : 4
value of b is : 5
value of *p is : 7
```

In the above case, a programmer has defined its own constructor, therefore the statement **Demo d2 = d1;** calls the copy constructor defined by the user. It creates the exact copy of the value types data and the object pointed by the pointer p. Deep copy does not create the copy of a reference type variable.

10)Differences b/w Copy constructor and Assignment operator(=)

Copy Constructor	**Assignment Operator**
It is an overloaded constructor.	It is a bitwise operator.
It initializes the new object with the existing object.	It assigns the value of one object to another object.

Syntax of copy constructor:
Class_name(const class_name &object_name)
{
// body of the constructor.
}
	Syntax of Assignment operator:
	Class_name a,b;
	b = a;

○ The **copy constructor** is invoked when the new object is initialized with the existing object. ○ The object is passed as an argument to the function. ○ It returns the object.	The **assignment operator** is invoked when we assign the existing object to a new object.
Both the existing object and new object shares the different memory locations.	Both the existing object and new object shares the same memory location.

If a programmer does not define the copy constructor, the compiler will automatically generate the implicit default copy constructor.	If we do not overload the "=" operator, the bitwise copy will occur.

11)What is object slicing

"Slicing" is where you assign an object of a derived class to an instance of a base class, thereby losing part of the information - some of it is "sliced" away.

For example,

```
class A {
  int foo;
};
class B : public A {
  int bar;
};
```

So an object of type B has two data members, foo and bar.

Then if you were to write this:

```
B b;
A a = b;
```

Then the information in b about member bar is lost in a.

12)Operator and function overloading

Function Overloading: You can have multiple definitions for the same function name in the same scope. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list. You cannot overload function declarations that differ only by return type.

Ex:

```
#include <iostream>

using namespace std;
class printData {
  public:
    void print(int i)
    {
     cout << "Printing int: " << i << endl;
     }
     void print(double  f)
    {
      cout << "Printing float: " << f << endl;
```

```
    }
    void print(char* c)
    {
      cout << "Printing character: " << c << endl;
 }
};

int main(void)
 {
  printData pd;
   // Call print to print integer
  pd.print(5);
   // Call print to print float
  pd.print(500.263);
  // Call print to print character
  pd.print("Hello C++");
   return 0;}
```

Operator Overloading:

You can redefine or overload most of the built-in operators ,so that they can be use with user-defined types as well.

Overloaded operators are functions with special names: the keyword "operator" followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

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

class Box {
  public:
    double getVolume(void) {
      return length * breadth * height;
    }
    void setLength( double len ) {
      length = len;
    }
    void setBreadth( double bre ) {
      breadth = bre;
    }
    void setHeight( double hei ) {
      height = hei;
```

```
    }

    // Overload + operator to add two Box objects.
    Box operator+(const Box& b) {
      Box box;
      box.length = this->length + b.length;
      box.breadth = this->breadth + b.breadth;
      box.height = this->height + b.height;
      return box;
    }

  private:
    double length;     // Length of a box
    double breadth;    // Breadth of a box
    double height;     // Height of a box
};

// Main function for the program
int main() {
  Box Box1;            // Declare Box1 of type Box
  Box Box2;            // Declare Box2 of type Box
  Box Box3;            // Declare Box3 of type Box
  double volume = 0.0;     // Store the volume of a box here

  // box 1 specification
  Box1.setLength(6.0);
  Box1.setBreadth(7.0);
  Box1.setHeight(5.0);

  // box 2 specification
  Box2.setLength(12.0);
  Box2.setBreadth(13.0);
  Box2.setHeight(10.0);

  // volume of box 1
  volume = Box1.getVolume();
  cout << "Volume of Box1 : " << volume <<endl;

  // volume of box 2
  volume = Box2.getVolume();
  cout << "Volume of Box2 : " << volume <<endl;

  // Add two object as follows:
  Box3 = Box1 + Box2;
```

```
   // volume of box 3
   volume = Box3.getVolume();
   cout << "Volume of Box3 : " << volume <<endl;

   return 0;
}
```

13) Which operator can not be overloaded

Scope resolution::

Dereferencing *

Dot .

Conditional ?:

14) Difference between overloading and Overriding

Overloading	Overriding
Overloading is defining functions that have similar signatures, yet have different parameters.	When a function of base class is re-defined in the derived class called as Overriding
It doesn't needs inheritance.	It needs inheritance.
Overloading is the concept of compile time polymorphism	Overriding is the concept of runtime polymorphism
Having same method name with different Signatures.	Methods name and signatures must be same.
Return type can be same or different	Return type must be same

class Dog{
public void bark()
{
cout<<"woof";
}
//same method name different parameters
public void bark(int num)
{
	class Dog{
	public void bark()
	{
	cout<<"woof";
	}
	}
	//same method name same parameteres
	class Hound :public Dog

for(int i=0;i<num;i++)
cout<<"woof";
}
}
	{
	Public void sniff()
	{
	cout<<"Sinff";
	}
	public void bark()
	{
	cout<<"bowl";}}

15) Define namespace in C++.

- The namespace is a logical division of the code which is designed to stop the naming conflict.
- The namespace defines the scope where the identifiers such as variables, class, functions are declared.
- The main purpose of using namespace in C++ is to remove the ambiguity. Ambiquity occurs when the different task occurs with the same name.
- For example: if there are two functions exist with the same name such as add(). In order to prevent this ambiguity, the namespace is used. Functions are declared in different namespaces.
- C++ consists of a standard namespace, i.e., std which contains inbuilt classes and functions. So, by using the statement "using namespace std;" includes the namespace "std" in our program.
- Syntax of namespaces

```
namespace namespace_name
{
 //body of namespace;
}
```

Syntax of accessing the namespace variable:

```
namespace_name::member_name;
```

15)Define 'std'.

Std is the default namespace standard used in C++

16)How delete [] is different from delete?

Delete is used to release a unit of memory, delete[] is used to release an array

17) What is the full form of STL in C++?

STL stands for Standard Template Library.

18)C++ Vector

A vector is a sequence container class that implements dynamic array, means size automatically changes when appending elements. A vector stores the elements in contiguous memory locations and allocates the memory as needed at run time.

19)Difference between vector and array

An array follows static approach, means its size cannot be changed during run time while vector implements dynamic array means it automatically resizes itself when appending elements.

Syntax

Consider a vector 'v1'. Syntax would be:

1. vector<object_type> v1;

Example

Let's see a simple example.

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{
vector<string> v1;
v1.push_back("java ");
v1.push_back("tutorial");
for(vector<string>::iterator itr=v1.begin();itr!=v1.end();++itr)
cout<<*itr;
return 0;
}
```

In this example, vector class has been used to display the string.

19)C++ Vector Functions

Function	Description

at()	It provides a reference to an element.
back()	It gives a reference to the last element.
front()	It gives a reference to the first element.
swap()	It exchanges the elements between two vectors.
push_back()	It adds a new element at the end.
pop_back()	It removes a last element from the vector.
empty()	It determines whether the vector is empty or not.
insert()	It inserts new element at the specified position.
erase()	It deletes the specified element.
resize()	It modifies the size of the vector.
clear()	It removes all the elements from the vector.
size()	It determines a number of elements in the vector.
capacity()	It determines the current capacity of the vector.
assign()	It assigns new values to the vector.
operator=()	It assigns new values to the vector container.
operator[]()	It access a specified element.
end()	It refers to the past-lats-element in the vector.
emplace()	It inserts a new element just before the position pos.
emplace_back()	It inserts a new element at the end.
rend()	It points the element preceding the first element of the vector.
rbegin()	It points the last element of the vector.
begin()	It points the first element of the vector.
max_size()	It determines the maximum size that vector can hold.
cend()	It refers to the past-last-element in the vector.
cbegin()	It refers to the first element of the vector.
crbegin()	It refers to the last character of the vector.
crend()	It refers to the element preceding the first element of the vector.
data()	It writes the data of the vector into an array.
shrink_to_fit()	It reduces the capacity and makes it equal to the size of the vector.

20)What is the difference between an array and a list?

- An Array is a collection of homogeneous elements while a list is a collection of heterogeneous elements.

- Array memory allocation is static and continuous while List memory allocation is dynamic and random.
- In Array, users don't need to keep in track of next memory allocation while In the list, the user has to keep in track of next location where memory is allocated.

21)What is the difference between new() and malloc()?

- new() is a preprocessor while malloc() is a function.
- There is no need to allocate the memory while using "new" but in malloc() you have to use sizeof().
- "new" initializes the new memory to 0 while malloc() gives random value in the newly allotted memory location.
- The new() operator allocates the memory and calls the constructor for the object initialization and malloc() function allocates the memory but does not call the constructor for the object initialization.
- The new() operator is faster than the malloc() function as operator is faster than the function.

22)Define friend function.

Friend function acts as a friend of the class. It can access the private and protected members of the class. The friend function is not a member of the class, but it must be listed in the class definition. The non-member function cannot access the private data of the class. Sometimes, it is necessary for the non-member function to access the data. The friend function is a non-member function and has the ability to access the private data of the class.

To make an outside function friendly to the class, we need to declare the function as a friend of the class as shown below:

```
1. class sample
2. {
3.   // data members;
4.  public:
5. friend void abc(void);
6. };
```

Following are the characteristics of a friend function:

- The friend function is not in the scope of the class in which it has been declared.

- Since it is not in the scope of the class, so it cannot be called by using the object of the class. Therefore, friend function can be invoked like a normal function.
- A friend function cannot access the private members directly, it has to use an object name and dot operator with each member name.
- Friend function uses objects as arguments.

example:

```
#include <iostream>
using namespace std;
class Addition
{
 int a=5;
 int b=6;
 public:
 friend int add(Addition a1)
 {
    return(a1.a+a1.b);
 }
};
int main()
{
int result;
Addition a1;
 result=add(a1);
 cout<<result;
return 0;
}
```

Output: 11

23)What is a virtual function?

- A virtual function is used to replace the implementation provided by the base class. The replacement is always called whenever the object in question is actually of the derived class, even if the object is accessed by a base pointer rather than a derived pointer.
- A virtual function is a member function which is present in the base class and redefined by the derived class.
- When we use the same function name in both base and derived class, the function in base class is declared with a keyword virtual.
- When the function is made virtual, then C++ determines at run-time which function is to be called based on the type of the object pointed by the base class pointer. Thus, by

making the base class pointer to point different objects, we can execute different versions of the virtual functions.

Rules of a virtual function:

- The virtual functions should be a member of some class.
- The virtual function cannot be a static member.
- Virtual functions are called by using the object pointer.
- It can be a friend of another class.
- C++ does not contain virtual constructors but can have a virtual destructor.

24)What are the ways you can implement Operator overloading.

Operator overloading can be implemented in the following functions:

- Member function
- Non-member function
- Friend function

25)What is virtual inheritance?

Virtual inheritance facilitates you to create only one copy of each object even if the object appears more than one in the hierarchy.

26)Explain this pointer?

This pointer holds the address of the current object.

27)What is a virtual destructor?

A virtual destructor in C++ is used in the base class so that the derived class object can also be destroyed. A virtual destructor is declared by using the ~ tilde operator and then virtual keyword before the constructor.

28)What will be the output of following code

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

template <typename T>
void fun(const T&x)
```

```
{
   static int count = 0;
   cout << "x = " << x << " count = " << count << endl;
   ++count;
   return;
}

int main()
{
   fun<int> (1);
   cout << endl;
   fun<int>(1);
   cout << endl;
   fun<double>(1.1);
   cout << endl;
   return 0;
}
```

Ans:
x = 1 count = 0
x = 1 count = 1
x = 1.1 count = 0
Compiler creates a new instance of a template function for every data type. So compiler creates two functions in the above example, one for int and other for double. Every instance has its own copy of static variable. The int instance of function is called twice, so count is incremented for the second call.

29)What is inline function?

An inline function is a combination of macro & function. At the time of declaration or definition, function name is preceded by word inline. When inline functions are used, the overhead of function call is eliminated. Instead, the executable statements of the function are copied at the place of each function call. This is done by the compiler

30) What are Restrictions on static member functions?

1. They can directly refer to other static members of the class.
2. Static member functions do not have this pointer.
3. Static member function cannot be virtual.

31) Do inline functions improve performance? Explain.

- A function when defined as INLINE, the code from the function definition is directly copied into the code of the calling function.
- It avoids the overhead of calling the actual function. This is because the complier performs

and inline expansion which eliminates the time overhead when a function is called.
- Reduces space as no separate set of instructions in memory is written.

32) What are ways to avoid memory leaks?

- A memory leak is the effect of running out of memory.
- A memory leak is what happens when you forget to free a block of memory allocated with the new operator or when you make it impossible to so.
- The measures you can take are :
i. Delete before reallocating a memory
ii. Be sure you have a pointer to each dynamic variable so that you do not lose a location that you need to free.
iii. Avoid these combinations :
  - malloc() - delete and new - free()
  - new - delete [] and new [] - delete.

33) What are the syntax and semantics for a function template?

- Templates is one of the features of C++. Using templates, C++ provides a support for generic programming.
- We can define a template for a function that can help us create multiple versions for different data types.
- A function template is similar to a class template and it syntax is as follows :

```
template <class T>
Return-type functionName (arguments of type T)
{
  //Body of function with type T wherever appropriate
}
```

34) What is overloading template? Explain it with an example.

- A template function overloads itself as needed. But we can explicitly overload it too. Overloading a function template means having different sets of function templates which differ in their parameter list.
- Consider following example :

```
#include <iostream>
template <class X> void func(X a)
{
```

```
  // Function code;
  cout <<"Inside f(X a) \n";
}
template <class X, class Y> void func(X a, Y b) //overloading function template func()
{
  // Function code;
  cout <<"Inside f(X a, Y b) \n";
}
int main()
{
  func(10); // calls func(X a)
  func(10, 20); // calls func(X a, Y b)
  return 0;
}
```

35) Exception handling concept.

- Exceptions are certain disastrous error conditions that occur during the execution of a program. They could be errors that cause the programs to fail or certain conditions that lead to errors. If these run time errors are not handled by the program, OS handles them and program terminates abruptly, which is not good.
To avoid this, C++ provides Exception Handling mechanism.
- The three keywords for Exception Handling are: Try, Catch and Throw.
- The program tries to do something. If it encounters some problem, it throws an exception to another program block which catches the exception.

Example :

```
void main()
{
  int no1, no2;
  try
  {
     cout << "Enter two nos:";
     cin >> no1 >> no2;
     if (no2 == 0)
       throw "Divide by zero";
     else
       throw no1/no2;
  }
```

```
  catch (char *s)
  {
     cout << s;
  }
  catch (int ans)
  {
     cout << ans;
  }
}
```

- We know that divide by zero is an exception. If user enters second no as zero, the program throws an exception, which is caught and an error message is printed else the answer is printed.

36) What is the advantage of exception handling?

1) Remove error-handling code from the software's business code.

2) A method writer can chose to handle certain exceptions and delegate others to the caller.

3) An exception that occurs in a function can be handled anywhere in the function call stack.

37) What are user define Exceptions give example

You can define your own exceptions by inheriting and overriding **exception**class functionality. Following is the example, which shows how you can use std::exception class to implement your own exception in standard way –

```
#include <iostream>
#include <exception>
using namespace std;
struct MyException : public exception {
  const char * what () const throw () {
```

```
    return "C++ Exception";
  }
};
int main() {
  try {
    throw MyException();
  } catch(MyException& e) {
    std::cout << "MyException caught" << std::endl;
    std::cout << e.what() << std::endl;
  } catch(std::exception& e) {
    //Other errors
  }
}
```

This would produce the following result −

```
MyException caught
C++ Exception
```

Here, **what()** is a public method provided by exception class and it has been overridden by all the child exception classes. This returns the cause of an exception.

38) What is meant by exception specification?

C++ provides a mechanism to ensure that a given function is limited to throw only a specified list of **exceptions**. An **exception specification** at the beginning of any function acts as a guarantee to the function's caller that the function will throw only the **exceptions** contained in the **exception specification**.

39) What happens if an exception is thrown but not caught?

Yes, **that's** how **exceptions** work. **When an exception is thrown**, it is **caught** by the topmost function in the call stack that has a handler for that **exception** in the scope of execution. ... However, **If** any destructor throws an **exception** during stack unwinding, then it is bad and the std::terminate function will be called

40) C++ Files and Streams

In C++ programming we are using the **iostream** standard library, it provides **cin** and **cout** methods for reading from input and writing to output respectively.

To read and write from a file we are using the standard C++ library called **fstream**. Let us see

Data Type	Description
fstream	It is used to create files, write information to files, and read information from files.
ifstream	It is used to read information from files.
ofstream	It is used to create files and write information to the files.

the data types define in fstream library is:

41) C++ Read and Write Example

Let's see the simple example of writing the data to a text file **testout.txt** and then reading the data from the file using C++ FileStream programming.

```
#include <fstream>
#include <iostream>
using namespace std;
int main () {
  char input[75];
  ofstream os;
  os.open("testout.txt");
  cout <<"Writing to a text file:" << endl;
  cout << "Please Enter your name: ";
  cin.getline(input, 100);
  os << input << endl;
  cout << "Please Enter your age: ";
  cin >> input;
  cin.ignore();
  os << input << endl;
  os.close();
  ifstream is;
  string line;
  is.open("testout.txt");
  cout << "Reading from a text file:" << endl;
  while (getline (is,line))
  {
  cout << line << endl;
```

```
  }
  is.close();
  return 0;
}
```

Output:

```
Writing to a text file:
Please Enter your name: Nakul Jain
Please Enter your age: 22
Reading from a text file:   Nakul Jain  22
```

CPP Questions Part-2

How do you define/declare constants in C++?
In C++, we can define our own constants using the **#define** preprocessor directive.
#define Identifier value

Comment on Assignment Operator in C++.
Answer: Assignment operator in C++ is used to assign a value to another variable.
a = 5;
This line of code assigns the integer value **5** to variable **a**.
The part at the left of the =operator is known as *lvalue* (left value) and the right as *rvalue* (right value). **L***value* must always be a variable whereas the right side can be a constant, a variable, the result of an operation or any combination of them.
The assignment operation always takes place from the right to left and never at the inverse.
One property which C++ has over the other programming languages is that the assignment operator can be used as the *rvalue* (or part of an *rvalue*) for another assignment.
Example:
a = 2 + (b = 5);
is equivalent to:
b = 5;
a = 2 + b;
Which means, first assign **5** to variable **b** and then assign to **a,** the value **2** plus the result of the previous expression of **b**(that is 5), leaves **a** with a final value of **7**.
Thus, the following expression is also valid in C++:
a = b = c = 5;
assign 5 to variables **a**, **b** and **c**.

What is the difference between equal to (==) and Assignment Operator (=)?
Answer: In C++, equal to (==) and assignment operator (=) are two completely different operators.

Equal to (==) is equality relational operator that evaluates two expressions to see if they are equal and returns true if they are equal and false if they are not.
The assignment operator (=) is used to assign a value to a variable. Hence, we can have a complex assignment operation inside the equality relational operator for evaluation.

What are the Extraction and Insertion operators in C++? Explain with examples.
Answer: In the iostream.h library of C++, cin, and cout are the two data streams that are used for input and output respectively. Cout is normally directed to the screen and cin is assigned to the keyboard.
"cin" (extraction operator): By using overloaded operator >> with cin stream, C++ handles the standard input.

```
1 int age;
2 cin>>age;
```

As shown in the above example, an integer variable 'age' is declared and then it waits for cin (keyboard) to enter the data. "cin" processes the input only when the RETURN key is pressed.
"cout" (insertion operator): This is used in conjunction with the overloaded << operator. It directs the data that followed it into the cout stream.
Example:

```
1 cout<<"Hello, World!";
2  cout<<123;
```

What do you mean by 'void' return type?
Answer: All functions should return a value as per the general syntax.
However, in case, if we don't want a function to return any value, we use "**void**" to indicate that. This means that we use "**void**" to indicate that the function has no return value or it returns "**void**".

Explain Pass by value and Pass by reference.
Answer: While passing parameters to the function using "Pass by Value", we pass a copy of the parameters to the function.
Hence, whatever modifications are made to the parameters in the called function are not passed back to the calling function. Thus the variables in the calling function remain unchanged.
Example:
By Value:

```
void printFunc(int a,int b,int c)
{
        a *=2;
        b *=2;
        c *=2;
}
int main()
{
```

```
        int x = 1,y=3,z=4;
        printFunc(x,y,z);
        cout<<"x = "<<x<<"\ny = "<<y<<"\nz = "<<z;
}
```

By Reference:

```
void printFunc(int& a,int& b,int& c)
{
                a *=2;
                b *=2;
                c *=2;
}
 int main()
{
         int x = 1,y=3,z=4;
         printFunc(x,y,z);
       cout<<"x = "<<x<<"\ny = "<<y<<"\nz = "<<z;
}
```

What are Default Parameters? How are they evaluated in C++ function?

Answer: Default parameter is a value that is assigned to each parameter while declaring a function.

This value is used if that parameter is left blank while calling to the function. To specify a default value for a particular parameter, we simply assign a value to the parameter in the function declaration.

If the value is not passed for this parameter during the function call, then the compiler uses the default value provided. If a value is specified, then this default value is stepped on and the passed value is used.

```
int multiply(int a, int b=2)
{
        int r;
        r = a * b;
        return r;
}

int main()
 {

       Cout<<multiply(6);
       Cout<<"\n";
       Cout<<multiply(2,3);
 }
```

Output:

```
12
6
```

State the difference between delete and delete[].
Answer: “delete[]” is used to release the memory allocated to an array which was allocated using new[]. “delete” is used to release one chunk of memory which was allocated using new.

What is wrong with this code?
T *p = new T[10];
delete p;
Answer: The above code is syntactically correct and will compile fine.
The only problem is that it will just delete the first element of the array. Though the entire array is deleted, only the destructor of the first element will be called and the memory for the first element is released.
10. What's the order in which the objects in an array are destructed?
Answer: Objects in an array are destructed in the reverse order of construction: First constructed, last destructed.
In the following Example, the order for destructors will be a[9], a[8], …, a[1], a[0]:

```
voiduserCode()
{
        Car a[10];
         ...
}
```

What is wrong with this code?
T *p = 0;
delete p;
Answer: In the above code, the pointer is a null pointer. Hence naturally, the program will crash in an attempt to delete the null pointer.

What is a Reference Variable in C++?
Answer: A reference variable is an alias name for the existing variable. This means that both the variable name and the reference variable point to the same memory location. Hence, whenever the variable is updated, the reference is updated too.

```
int a=10;
int& b = a;
```

Here, b is the reference of a.

What is a Storage Class? Mention the Storage Classes in C++.
Answer: Storage class determines the life or scope of symbols such as variable or functions.
C++ supports the following storage classes:
Auto
Static
Extern
Register
Mutable

Explain Mutable Storage class specifier.
Answer: The variable of a constant class object's member cannot be changed. However, by declaring the variables as "mutable", we can change the values of these variables.

What is a Static Variable?
Answer: A static variable is a local variable that retains its value across the function calls. Static variables are declared using the keyword "static". Numeric variables which are static have the default value as zero.
The following function will print 1 2 3 if called thrice.

```
void f()
{
static int i;
++i;
printf("%d ",i);
}
```

what is the purpose of Extern Storage Specifier?

Answer: "Extern" specifier is used to resolve the scope of a global symbol.

```
#include <iostream >
 using nam espace std;
 main()
 {
extern int i;
 cout<<i<<endl;
 }
 int i=20;
```

In the above code, "i" can be visible outside the file where it is defined.

Explain Register Storage Specifier.
Answer: "Register" variable should be used whenever the variable is used. When a variable is declared with a "register" specifier, then the compiler gives CPU register for its storage to speed up the lookup of the variable.

When to use "const" reference arguments in a function?
Answer: Using "const" reference arguments in a function is beneficial in several ways:
"const" protects from programming errors that could alter data.
As a result of using "const", the function is able to process both const and non-const actual arguments, which is not possible when "const" is not used.
Using a const reference, allows the function to generate and use a temporary variable in an appropriate manner.

What is a Class?
Answer: Class is a user-defined data type in C++. It can be created to solve a particular kind of problem. After creation, the user need not know the details of the working of a class.
In general, class acts as a blueprint of a project and can include in various parameters and functions or actions operating on these parameters. These are called the members of the class

Difference between Class and Structure.
Answer:
Structure: In C language, the structure is used to bundle different type of data types together. The variables inside a structure are called the members of the structure. These members are by default public and can be accessed by using the structure name followed by a dot operator and then the member name.
Class: Class is a successor of the Structure. C++ extends the structure definition to include the functions that operate on its members. By default all the members inside the class are private.

What is Namespace?
Answer: Namespaces allow us to group a set of global classes, objects and/or functions under a specific name.

The general form to use namespaces is:

namespace identifier { namespace-body }
Where identifier is any valid identifier and the namespace-body is the set of classes, objects, and functions that are included within the namespace. Namespaces are especially useful in the case where there is a possibility for more than one object to have the same name, resulting in name clashes.

What is the use of 'using' declaration?
Answer: Using declaration is used to refer a name from the namespace without the scope resolution operator.

What is Name Mangling?
Answer: C++ compiler encodes the parameter types with function/method into a unique name. This process is called name mangling. The inverse process is called demangling.
Example:

A::b(int, long) const is mangled as 'b__C3Ail'.
For a constructor, the method name is left out.
That is A:: A(int, long) const is mangled as 'C3Ail'.

What is the difference between an Object and a Class?
Answer: Class is a blueprint of a project or problem to be solved and consists of variables and methods. These are called the members of the class. We cannot access methods or variables of the class on its own unless they are declared static.

In order to access the class members and put them to use, we should create an instance of a class which is called an Object. The class has an unlimited lifetime whereas an object has a limited lifespan only.

What are the various Access Specifiers in C++?
Answer: C++ supports the following access specifiers:
Public: Data members and functions are accessible outside the class.
Private: Data members and functions are not accessible outside the class. The exception is the usage of a friend class.
Protected: Data members and functions are accessible only to the derived classes.
Example:

```
class A{
        int x; int y;
        public int a;
        protected bool flag;
        public A() : x(0) , y(0) {} //default (no argument) constructor
 };
main()
{
A MyObj;
MyObj.x = 5; // Compiler will issue a ERROR as x is private
int x = MyObj.x; // Compiler will issue a compile ERROR MyObj.x is private
MyObj.a = 10; // no problem; a is public member
int col = MyObj.a; // no problem
MyObj.flag = true; //Compiler will issue a ERROR; protected values are read only
bool isFlag = MyObj.flag; // no problem
}
```

What is a COPY CONSTRUCTOR and when is it called?
Answer: A copy constructor is a constructor that accepts an object of the same class as its parameter and copies its data members to the object on the left part of the assignment. It is useful when we need to construct a new object of the same class.

```
class A{
        int x; int y;
        public int color;
        public A() : x(0) , y(0) {} //default (no argument) constructor
        public A( const A& ) ;
};
A::A( const A & p )
{
         this->x = p.x;
         this->y = p.y;
         this->color = p.color;
}
```

```
main()
{
        A Myobj;
        Myobj.color = 345;
        A Anotherobj = A( Myobj ); // now Anotherobj has color = 345
}
```

What is a Default Constructor?

Answer: Default constructor is a constructor that either has no arguments or if there are any, then all of them are default arguments.

```
class B {
 public: B (int m = 0) : n (m) {} int n;
 };
 int main(int argc, char *argv[])
{
 B b; return 0;
 }
```

What is a Conversion Constructor?

A conversion constructor is a single-parameter constructor that is declared without the function specifier explicit. The compiler uses conversion constructors to convert objects from the type of the first parameter to the type of the conversion constructor's class.

The following example demonstrates this:

```
class Y
 {
  int a, b;
  char* name;
 public:
  Y(int i) { };
  Y(const char* n, int j = 0) { };
};

void add(Y) { };

int main() {

  // equivalent to
  // obj1 = Y(2)

  Y obj1 = 2;

  // equivalent to
```

```
 // obj2 = Y("somestring",0)

 Y obj2 = "somestring";

 // equivalent to
 // obj1 = Y(10)

 obj1 = 10;

 // equivalent to
 // add(Y(5))
   add(5);

}
```

The above example has the following two conversion constructors:
Y(int i)which is used to convert integers to objects of class Y.
Y(const char* n, int j = 0) which is used to convert pointers to strings to objects of class Y.
The compiler will not implicitly convert types as demonstrated above with constructors declared with the explicit keyword. The compiler will only use explicitly declared constructors in new expressions, the static_cast expressions and explicit casts, and the initialization of bases and members. The following example demonstrates this:

```
class A {
public:
  explicit A() { };
  explicit A(int) { };
};

int main() {
  A z;
//  A y = 1;
  A x = A(1);
  A w(1);
  A* v = new A(1);
  A u = (A)1;
  A t = static_cast<A>(1);
}
```

The compiler would not allow the statement A y = 1 because this is an implicit conversion; class A has no conversion constructors.

What is the role of Static keyword for a class member variable?
Answer: Static member variable shares a common memory across all the objects created for the respective class. We need not refer to the static member variable using an object. However, it can be accessed using the class name itself.

What's the order in which the local objects are destructed?

Answer: Consider following a piece of code:

```
Class A{
 ….
 };
 int main()
 {
 A a;
 A b;
 ...
 }
```

In the main function, we have two objects created one after the other. They are created in an order, first a then b. But when these objects are deleted or if they go out of the scope, the destructor for each will be called in the reverse order in which they were constructed.

Explain Function Overloading and Operator Overloading.

Answer: C++ supports OOPs concept Polymorphism which means “many forms”.

In C++ we have two types of polymorphism, i.e. Compile-time polymorphism, and Run-time polymorphism. Compile time polymorphism is achieved by using an Overloading technique. Overloading simply means giving additional meaning to an entity by keeping its base meaning intact.

C++ supports two types of overloading:

Function Overloading:

Function overloading is a technique which allows the programmer to have more than one function with the same name but different parameter list. In other words, we overload the function with different arguments i.e. be it the type of arguments, number of arguments or the order of arguments.

Function overloading is never achieved on its return type.

Operator Overloading:

This is yet another type of compile-time polymorphism that is supported by C++. In operator overloading, an operator is overloaded, so that it can operate on the user-defined types as well with the operands of the standard data type. But while doing this, the standard definition of that operator is kept intact.

For Example, Addition operator (+) that operates on numerical data types can be overloaded to operate on two objects just like an object of complex number class.

What is the difference between Method Overloading and Method Overriding in C++?

Answer: Method overloading is having functions with the same name but different argument list. This is a form of compile-time polymorphism.

Method overriding comes into picture when we rewrite the method that is derived from a base class. Method overriding is used while dealing with run-time polymorphism or virtual functions.

What is the difference between a Copy Constructor and an Overloaded Assignment Operator?
Answer: A copy constructor and an overloaded assignment operator basically serve the same purpose i.e. assigning the content of one object to another. But still, there is a difference between the two.

```
complex c1,c2;
c1=c2; //this is assignment
complex c3=c2; //copy constructor
```

In the above example, the second statement c1 = c2 is an overloaded assignment statement.Here, both c1 and c2 are already existing objects and the contents of c2 are assigned to the object c1. Hence, for overloaded assignment statement both the objects need to be created already.
Next statement, complex c3 = c2 is an example of the copy constructor. Here, the contents of c2 are assigned to a new object c3, which means the copy constructor creates a new object every time when it executes.

Name the Operators that cannot be Overloaded.
Answer:
sizeof – sizeof operator
. – Dot operator
.* – dereferencing operator
-> – member dereferencing operator
:: – scope resolution operator
?: – conditional operator

Function can be overloaded based on the parameter which is a value or a reference. Explain if the statement is true.
Answer: False. Both, Passing by value and Passing by reference look identical to the caller.

What are the benefits of Operator Overloading?
Answer: By overloading standard operators on a class, we can extend the meaning of these operators, so that they can also operate on the other user-defined objects.
Function overloading allows us to reduce the complexity of the code and make it more clear and readable as we can have the same function names with different argument lists.

What is Inheritance?
Answer: Inheritance is a process by which we can acquire the characteristics of an existing entity and form a new entity by adding more features to it.
In terms of C++, inheritance is creating a new class by deriving it from an existing class so that this new class has the properties of its parent class as well as its own.

What are the advantages of Inheritance?
Answer: Inheritance allows code re-usability, thereby saving time on code development.
By inheriting, we make use of a bug-free high-quality software that reduces future problems.

What are Multiple Inheritances (virtual inheritance)? What are its advantages and disadvantages?

Answer: In multiple inheritances, we have more than one base classes from which a derived class can inherit. Hence, a derived class takes the features and properties of more than one base class.

For Example, a class driver will have two base classes namely, employee and a person because a driver is an employee as well as a person. This is advantageous because the driver class can inherit the properties of the employee as well as the person class.

But in the case of an employee and a person, the class will have some properties in common. However, an ambiguous situation will arise as the driver class will not know the classes from which the common properties should be inherited. This is the major disadvantage of multiple inheritance.

Explain the ISA and HASA class relationships. How would you implement each?

Answer: "ISA" relationship usually exhibits inheritance as it implies that a class "ISA" specialized version of another class. Example, An employee ISA person. That means an Employee class is inherited from the Person class.

Contrary to "ISA", "HASA" relationship depicts that an entity may have another entity as its member or a class has another object embedded inside it.

So taking the same example of an Employee class, the way in which we associate the Salary class with the employee is not by inheriting it but by including or containing the Salary object inside the Employee class. "HASA" relationship is best exhibited by containment or aggregation.

Does a derived class inherit or doesn't inherit?

Answer: When a derived class is constructed from a particular base class, it basically inherits all the features and ordinary members of the base class. But there are some exceptions to this rule. For instance, a derived class does not inherit the base class's constructors and destructors. Each class has its own constructors and destructors. The derived class also does not inherit the assignment operator of the base class and friends of the class. The reason is that these entities are specific to a particular class and if another class is derived or if it is the friend of that class, then they cannot be passed onto them.

What is Polymorphism?

Answer: The basic idea behind polymorphism is in many forms. In C++, we have two types of Polymorphism:

(i) Compile time Polymorphism

In compile time polymorphism, we achieve many forms by overloading. Hence, we have Operator overloading and function overloading. (We have already covered this above)

(ii) Run-time Polymorphism

This is the polymorphism for classes and objects. General idea is that a base class can be inherited by several classes. A base class pointer can point to its child class and a base class array can store different child class objects.

This means, that an object reacts differently to the same function call. This type of polymorphism can use virtual function mechanism.

What are Virtual Functions?

Answer: A virtual function allows the derived classes to replace the implementation provided by the base class.

Whenever we have functions with the same name in the base as well as derived class, there arises an ambiguity when we try to access the child class object using a base class pointer. As we are using a base class pointer, the function that is called is the base class function with the same name.

To correct this ambiguity we use the keyword "virtual" before the function prototype in the base class. In other words, we make this polymorphic function Virtual. By using a Virtual function, we can remove the ambiguity and we can access all the child class functions correctly using a base class pointer.

Give an example of Run-time Polymorphism/Virtual Functions.

```
class SHAPE{
            public virtual Draw() = 0; //abstract class with a pure virtual method
 };
 class CIRCLE: public SHAPE{
             public int r;
             public Draw() { this->drawCircle(0,0,r); }
 };
class SQUARE: public SHAPE{
           public int a;
           public Draw() { this->drawSquare(0,0,a,a); }
 };

 int main()
 {
            SHAPE shape1*;
            SHAPE shape2*;

            CIRCLE c1;
            SQUARE s1;

           shape1 = &c1;
          shape2 = &s1;
         cout<<shape1->Draw(0,0,2);
         cout<<shape2->Draw(0,0,10,10);
 }
```

In the above code, SHAPE class has a pure virtual function and is an abstract class (cannot be instantiated). Each class is derived from SHAPE implementing Draw () function in its own way. Further, each Draw function is virtual so that when we use a base class (SHAPE) pointer each time with the object of the derived classes (Circle and SQUARE), then appropriate Draw functions are called.

What do you mean by Pure Virtual Functions?

Answer: A Pure Virtual Member Function is a member function in which the base class forces the derived classes to override. Normally this member function has no implementation. Pure virtual functions are equated to zero.

Example:

```
class Shape { public: virtual void draw() = 0; };
```

Base class that has a pure virtual function as its member can be termed as an "Abstract class". This class cannot be instantiated and it usually acts as a blueprint that has several sub-classes with further implementation.

What are Virtual Constructors/Destructors?

Virtual Destructors: When we use a base class pointer pointing to a derived class object and use it to destroy it, then instead of calling the derived class destructor, the base class destructor is called.

```
Class A{
        ….
        ~A();
};
Class B:publicA{
         …
         ~B();
};
B b;
A a = &b;
delete a;
```

As shown in the above example, when we say delete a, the destructor is called but it's actually the base class destructor. This gives rise to the ambiguity that all the memory held by b will not be cleared properly.

This problem can be solved by using the "Virtual Destructor" concept.

What we do is, we make the base class constructor "Virtual" so that all the child class destructors also become virtual and when we delete the object of the base class pointing to the object of the derived class, the appropriate destructor is called and all the objects are properly deleted.

This is shown as follows:

```
Class A{
        ….
        virtual ~A();
};
Class B:publicA{
         …
         ~B();
};
```

```
B b;
A a = &b;
delete a;
```

Virtual constructor: Constructors cannot be virtual. Declaring a constructor as a virtual function is a syntax error.

What is a friend function?

C++ class does not allow its private and protected members to be accessed outside the class. But this rule can be violated by making use of the “Friend” function.

As the name itself suggests, friend function is an external function which is a friend of the class. For friend function to access the private and protected methods of the class, we should have a prototype of the friend function with the keyword “friend” included inside the class.

What is a friend class?

Answer: Friend classes are used when we need to override the rule for private and protected access specifiers so that two classes can work closely with each other.

Hence, we can have a friend class to be the friend of another class. This way, friend classes can keep private, inaccessible things in the way they are.

When we have a requirement to access the internal implementation of a class (private member) without exposing the details by making the public, we go for friend functions.

What is a template?

Templates allow creating functions that are independent of data type (generic) and can take any data type as parameters and return value without having to overload the function with all the possible data types. Templates nearly fulfill the functionality of a macro.

Its prototype is any of the following ones:

```
template <class identifier> function_declaration;
template <typename identifier> function_declaration;
```

The only difference between both the prototypes is the use of keyword class or typename. Their basic functionality of being generic remains the same.

What is Exception Handling? Does C++ support Exception Handling?

C++ supports exception handling.

We cannot ensure that code will execute normally at all times. There can be certain situations which might force the code written by us to malfunction, even though it’s error-free. This malfunctioning of code is called Exception.

When an exception has occurred, the compiler has to throw it so that we know an exception has occurred. When an exception has been thrown, the compiler has to ensure that it is

handled properly, so that the program flow continues or terminates properly. This is called handling of an exception.
Thus in C++, we have three keywords i.e. try, throw and catch which are in exception handling.
The general syntax for exception block is:

```
try{
 ….
 # Code that is potentially about to throw exception goes here
 ….
 throw exception;
 }
 catch(exception type) {
 …
 #code to handle exception goes here
 }
```

As shown above, the code that might potentially malfunction is put under the try block. When code malfunctions, an exception is thrown. This exception is then caught under the catch block and is handled i.e. appropriate action is taken.

Comment on C++ standard exceptions?
Answer: C++ supports some standard exceptions that can be caught if we put the code inside the try block. These exceptions are a part of the base class “std:: exception”. This class is defined in the C++ header file <exception>.
Few Examples of Exceptions supported by this class include:
bad_alloc – thrown by ‘new’
runtime_error – thrown for runtime errors
bad_typeid – thrown by type id

What is a Standard Template Library (STL)? What are the various types of STL Containers?
Answer: A Standard Template Library (STL) is a library of container templates approved by the ANSI committee for inclusion in the standard C++ specification. We have various types of STL containers depending on how they store the elements.
Queue, Stack – These are the same as traditional queue and stack and are called adaptive containers.
Set, Map – These are basically containers that have key/value pairs and are associative in nature.
Vector, deque – These are sequential in nature and have similarity to arrays.

What is an Iterator class?
Answer: In C++ a container class is a collection of different objects.
If we need to traverse through this collection of objects, we cannot do it using simple index variables. Hence, we have a special class in STL called an Iterator class which can be used to step through the contents of the container class.
The various categories of iterators include input iterators, output iterators, forward iterators, bidirectional Iterators, random access etc.

What is the difference between an External Iterator and an Internal Iterator? Describe an advantage of the External Iterator.

Answer: An internal iterator is implemented with member functions of the class that has items to step through.

An external iterator is implemented as a separate class that can be bound to the object that has items to step through. The basic advantage of an External iterator is that it's easy to implement as it is implemented as a separate class.

Secondly, as it's a different class, many iterator objects can be active simultaneously.

1. What is data structure?

   Data Structure is an arrangement of data in computer's memory (or sometimes on a disk) so that we can retrieve it & manipulate it correctly and efficiently.

2. List out the areas in which data structures are applied extensively?

   Data structures are essential in almost every aspect where data is involved. In general, algorithms that involve efficient data structure is applied in the following areas:

   Numerical analysis,
   Operating system,
   Statistical analysis
   Compiler Design,
   Operating System,
   Database Management System,
   Statistical analysis package,
   Numerical Analysis,
   Graphics,
   Artificial Intelligence,
   Simulation

3. What are the major data structures used in the following areas :
   RDBMS, Network data model & Hierarchical data model.

   - RDBMS Array (i.e. Array of structures)
   - Network data model Graph
   - Hierarchical data model Trees

4. What are the limitations of array over linked list?
   - Array does not grow dynamically (i.e length has to be known)
   - Inefficient memory management.
   - In ordered array insertion is slow.
   - In both ordered and unordered array deletion is slow.
   - Large number of data movements for insertion & deletion
   - which is very expensive in case of array with large number of elements.

5. Sorting Algorithm:

Bubble sort:

```
for( int i = 0; i < passes; i++ ) {
        for( int j = 0; j < comps - i; j++ )
         {
          if( arr[j+1] < arr[j] )
         {
          int temp = arr[j+1];
          arr[j+1] = arr[j];
          arr[j] = temp;
          }
        }
```

- Number of iterations : (n-1)(n-1)
- $T \alpha n^2 - 2n + 1$
- If $n > 1, n^2 > n$ then all lower order terms are ignored
- $T \alpha n^2$
- Time complexity $= O(n^2)$

Selection Sort:

```
for( int i = 0; i < size-1; i++ ) {
    int min = i;
    for( int j = i+1; j < size; j++ ) {
      if( arr[j] < arr[min] ) {
        min = j;
      }
    }
    int temp = arr[i];
    arr[i] = arr[min];
    arr[min] = temp;
  }
```

- Number of iterartions: $1+2+3+\ldots.+n-1=n(n-1)/2$
- $T \alpha n(n-1)/2$
- $T \alpha n^2 - n$
- If $n > 1, n^2 > n$ then all lower order terms are ignored
- $T \alpha n^2$
- Time complexity $= O(n^2)$

Insertion Sort:

```
for( int  j = 0; j < size; j++ ) {
      for( int i = 0; i < j; i++ ) {
         if( arr[j] < arr[i] ) {
            int temp = arr[j];
            arr[j] = arr[i];
            arr[i] = temp;
         }
      }
   }
```

- Base case: O(n)
- Average/worst case: $O(n^2)$

Quick Sort :

```
void quickSort( int * arr, int left, int right ) {
      if( left >= right ) {
      return;
   }
    int leftHold = left, rightHold = right;
    int pivot = arr[left];
     while( left < right ) {
        while( arr[right] > pivot && left != right ) {
          right--;
       }
        if( left != right ) {
          arr[left] = arr[right];
          left++;
       }
        while( arr[left] < pivot && left != right ) {
          left++;
       }
        if( left != right ) {
          arr[right] = arr[left];
          right--;
       }
   }
   arr[left] = pivot;
   quickSort(arr, leftHold, left-1 );
   quickSort(arr, left+1, rightHold );    }
```

- Average/worst case: O(n log n)
- Worst case: $O(n^2)$

Merge Sort:

```
if( left < right ) {
      int mid = ( left + right ) / 2;
      mergeSort( arr, left, mid );
      mergeSort( arr, mid+1, right );
      merge( arr, left, mid, mid+1, right );
      }
```

Average/worst case: O(n log n)

```
void merge( int * arr, int leftStart, int leftEnd, int rightStart, int rightEnd ) {
  int temp[100];
  int tempPos = leftStart;
  int numEle = ( rightEnd - leftStart ) + 1;
   while( ( leftStart <= leftEnd ) && ( rightStart <= rightEnd ) ) {
     if( arr[leftStart] < arr[rightStart] ) {
       temp[tempPos] = arr[leftStart];
       tempPos++;       leftStart++;
     }
     else {
       temp[tempPos] = arr[rightStart];
       tempPos++;       rightStart++;
     }
}
    while( leftStart <= leftEnd ) {
     temp[tempPos] = arr[leftStart];
     tempPos++;
     leftStart++;
  }
    while( rightStart <= rightEnd ) {
     temp[tempPos] = arr[rightStart];
     tempPos++;       rightStart++;
  }
    for( int i = 1; i <= numEle; i++ ) {
     arr[rightEnd] = temp[rightEnd];
     rightEnd--;
  }  }
```

Binary Search:

```
while(left<=right)
{
        mid=(left + right)/2;
        if(key==arr[mid])
            return mid;
        if(key<arr[mid])
            right=mid-1;
        else
            left=mid+1;
}
```

6. Linked List?

   We can overcome the drawbacks of the sequential storage.

   If the items are explicitly ordered, that is each item contained within itself the address of the next Item.

   In Linked Lists elements are logically adjacent, need not be physically adjacent. Node is divide into two part.

   Operation can be performed on linked List:

   1. insert()
   2. insertAtFirst()
   3. insertAtLast()
   4. insertByPos()
   5. deleteAtFirst()
   6. deleteAtLast()
   7. deleteByPos()
   8. deleteByVal()
   9. deleteAllNode()
   10. Reserve()
   11. midElement()

1. Insert by value

```
Node * newNode = new Node(data);
        if (newNode == NULL) {
                return false;
        }
        if (head == NULL) {
                head = newNode;
                return true;
        }
        Node * last = head;
        while (last->getNext() != NULL) {
                last = last->getNext();
        }
        last->setNext(newNode);
        return true;
```

Algorithm for insert node:

1. Create node
2. Check for empty list
   a) If list is empty
      Set head is equal to newnode
   b) If list is not empty,
      - Create temp node
      - Locate temp to last node
      - Set last node next to newnode

2. To Insert node at first location: O(1)

```
bool insertAtFirst()
{
if(head==NULL)
{
   head=newnode;
   return true;
}
newnode->next=head;
head=newnode;
return true;
}
```

Algorithm for insert at first

1. check for empty list
   a. if list is empty
      set head=newnode
   b. if list is not empty
      set newnode->next=head;
      head=newnode;

3. To Insert node at last location: O(n)

```
bool insertAtLast()
{
if(head==NULL)
{
   head=newnode;
   return true;
}
Node* last=head;
//locate temp to last node
while(last->next!=NULL)
{
last=last->next;
}
last->next=newnode;
return true;
```

Algorithm for insert at last

1. check for empty list
   a. if list is empty
      set head=newnode
   b. if list is not empty
      i. locate last pointer to last node
      ii. set last next to newnode
      last->next=newnode

4. Insert node at given location: O(n)

```
bool insertAtPos(int pos,int val)
{
  //check for invalid position
if(position <=0 || ( head == NULL && position > 1 ))
  {
            return false;
  }

  Node * newNode = new Node(data);
  //check for newnode get memory
  if(newNode == NULL){
            return false;
  }
  //check for position 1
  if(position == 1){
            if(head != NULL){
                      newNode->setNext(head);
            }
            head = newNode;
            return true;
  }

  Node * prev = head;
  // position other than 1
  for(int i = 1; i < position - 1; i++){
            prev = prev->getNext();

            if(prev == NULL){
                      delete newNode;
                      return false;
            }
  }
  newNode->setNext(prev->getNext());
  prev->setNext(newNode);
  return true;
}
```

Invalid position

5. Delete node at first location: O(1)

```
bool deleteAtFirst()
{
If (head==NULL)
{
        return false;
}
Node* del =head;
head=del->next;
delete[] del;
return true;
}
```

Algorithm for delete at first

1. check for empty list
   a. if list is empty
      no node will be deleted
      return false;
   b. if list is not empty
      i. locate del pointer to first node
      ii. set head to del next
      head=del->next
      iii. delete del pointer

6. Delete node at last location: O(n)

```
bool deleteAtLast()
{
if(head==NULL)
{
   return false;
}
Node* del=head,*prev=head;
//locate temp to last node
While(del->next!=NULL)
{
prev=del;
del=del->next;
}
prev->next=NULL;
return true;
}
```

Algorithm for delete at last

1. check for empty list
   a. if list is empty
      set head=newnode
   b. if list is not empty
      i. locate last pointer to last node
      ii. set last next to newnode
      last->next=newnode

7. Delete node at given location: O(n)

```
bool LinkedList::deleteByPos(int position) {

     //Check for invalid pos
      if (position <= 0) {
            return false;
      }
     //Check for position 1
      if (position == 1) {
            Node * del = head;
            head = head->getNext();
            delete del;
            return true;
      }
      // position other than 1
      Node * prev = head;
      for (int i = 1; i < position - 1; i++) {
            prev = prev->getNext();

            if (prev == NULL) {
                  return false;
            }
      }
      Node * del = prev->getNext();
      prev->setNext( del->getNext() );
      delete del;
      return true;
}
```

8. Delete node with given value: O(n)

```
bool LinkedList::deleteByVal(int data) {
      if (head == NULL) {
            return false;
      }
      //check for first node
      if (head->getData() == data) {
            Node * del = head;
            head = del->getNext();
            delete del;
            return true;
      }
      //check for given data
      Node * del = head, * prev = head;
      while (del->getData() != data) {
            prev = del;
            del = del->getNext();
            if (del == NULL) {
                  return false;
            }
      }
      prev->setNext( del->getNext() );
      delete del;
      return true ;
}
```

Invalid data

9. Delete all node : O(n)

```
while( head!=NULL)
        deleteAtFirst();
```

10. Reverse list

```
void LinkedList:: Reverse(){
        Node * stack[100];
        int top = -1;
        Node * temp = head;
        while (temp) {
                stack[++top] = temp;
                temp = temp->getNext();
        }
        while (top != -1) {
                cout << stack[top--]->getData()  << " ";
        }
        cout << endl;
}
```

11. Find out middle element in linked list:

Method 1(using two loop):

```
count=0;
while(temp!=NULL)
  {
        count++;
        temp=temp->getNext();
  }
  for(i=0;i<count/2;i++)
  {
        temp=temp->getNext();
  }
  return temp->getData();
```

Method 2(using one loop):

```
Node* temp=head, prev*=head;
while(temp=!NULL && temp->getNext()!=NULL)
{

        temp=temp->getNext()->getNext();
        prev=prev->getNext();
}
        return prev->getData();
```

7. Singly circular Linked List?
   Last node's next pointer keeps address of first (head) node.

1. Insert at circular list = O(n)

```
bool CircularLinkedList :: insert(int data) {
      Node * newNode = new Node( data );
//Check for memory allocation to newnode
      if( newNode == NULL ) {
      return false;
   }
//Check for empty list  (if yes insert node at first)

    if( head == NULL ) {
      head = newNode;
      newNode->setNext( newNode );
      return true;
   }
//create temporary pointer and locate it to last node
   Node * last = head;
   while( last->getNext() != head ) {
      last = last->getNext();
   }
   newNode->setNext( head ); //set last(newnode) node next to head
   last->setNext( newNode );

   return true;
}
```

2. Insert at first location= O(1)

```
bool insertAtFirst()
{
  if(head==NULL)
      {
          head=newnode;
          newnode->next=head;
          return true;
      }
      Node* temp=head;

      while(temp->next!=Null)
      {
                temp=temp->next;
      }
      Newnode->next=head;
      head=newnode;
      temp->next=head;
      return true;
}
```

3. **Insert at last location= O(1)**

```
bool insertAtLast()
{
   if(head==NULL)
     {
        head=newnode;
        newnode->next=head;
        return true;
     }
     Node* temp=head;

     while(temp->next!=Null)
     {
              temp=temp->next;
     }
     temp->next=newnode;
     newnode->next=head;

     return true;
}
```

4. **delete at first location= O(1)**

```
bool deleteAtFirst() circular
{
   if(head==NULL)
     {
        return false;
     }
     Node* del=head;
     head=del->next;
     delete[] del;
     return true;
}
```

5. **delete at last location= O(1)**

```
bool deleteAtLast()
{
   if(head==NULL)
     {
        return false;
     }
     Node* del=head,*prev=head;
     while(del->next!=NULL)
     {
              prev=del;
              del=del->next;
     }
     Node* del=head;
     head=del->next;
     delete[] del;
     return true;
}
```

8. Doubly Linked List?

1. Insert in doubly list:

```
bool DoublyLinkedList::insert(int data) {

      Node * newNode = new Node(data);

      if (newNode == NULL) {
            return false;
      }

      if (head == NULL) {
            head = newNode;
            return true;
      }

      Node * last = head;
      while (last->getNext() != NULL) {
            last = last->getNext();
      }

      last->setNext(newNode);
      newNode->setPrev(last);

      return true;

}
```

2. delete in doubly list:

```
bool DoublyLinkedList::deletebyPos(int position) {
      if (position <= 0 || head == NULL ) {
            return false;
      }
      if (position == 1) {
            Node * del = head;
            head = head->getNext();
            if (head != NULL) {
                  head->setPrev(NULL);
            }
            delete del;
            return true;
      }
      Node * del = head;
      for (int i = 1; i < position; i++) {
            del = del->getNext();
            if (del == NULL) {
                  return false;
            }
      }
      del->getPrev()->setNext(del->getNext());
      if (del->getNext() != NULL) {
            del->getNext()->setPrev(del->getPrev());
      }
      delete del;
      return true;

}
```

Stack:

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).

Following are operations are performed with stack.

1. Push()
2. Pop()
3. Peek()

We can implement stack by two ways,

- Using array
- Using linkedlist

Stack implementation using Array:

isFull() & isEmpty():

```
bool isEmpty() {
      return top == -1;
  }

 bool isFull() {
      return top == ( size - 1 );
  }
```

Push ():

```
bool push( int data ) {
      if( isFull() ) {
         cout<<"List is full"<<endl;
         return false;
      }

      arr[++top] = data;
      return true;
   }
```

Pop ():

```
int pop() {
      if( isEmpty() ) {
                cout<<"List is empty"<<endl;
         return -999;
      }
         return arr[--top];
   }
```

Peek ():

```
int peek() {
        if( isEmpty() ) {
        return -999;
      }
      return arr[top];
   }
```

Stack implementation using Linked List:

push() :

```
bool push(int data)
{
        node* newnode =new node(data);
        if (newnode == NULL)
        {
                return false;
        }
        newnode->setData( data);
        newnode->Setnext( head);
        head = newnode;
       return true;
}
```

pop():

```
bool pop()
{
        node* del = head;
        if (head == NULL)
        {
                return false;
        }
        head = del->getNext();
        delete [] del;
        return ture;

}
```

peek():

```
int peek()
{
        return head->getData();
}
```

Queue:

A Queue is an ordered collection of items into which new items may be inserted at rear end and items are deleted at one end called front.

Types of Queue:

1. **Linear Queue**
2. **Circular Queue**
3. **Priority Queue**
4. **Dequeue (Double Ended Queue)**

Queue implementation using Array:

isFull() & isEmpty():

```
bool isEmpty() {
    return ( front == -1 && rear == -1) || ( front > rear );
  }

  bool isFull()
      {
    return rear = ( size - 1 );
  }
```

Insert ():

```
bool insert( int data ){
      if( isFull() ) {
         return false;
      }
       arr[++rear] = data;
      if( front == -1 ) {
         front = 0;
      }
      return true;
   }
```

delete() :

```
int deleteData() {
      if( isEmpty() ) {
         return -999;
      }
      return arr[front++];
   }
```

Circular Queue implementation using Array:

isFull() & isEmpty():

```
bool isEmpty() {
      return front == rear;
   }
   bool isFull() {
      return  ( front == -1 && rear ==( size - 1 ) ) ||
      ( (rear+1) % size == front );
   }
```

Insert():

```
bool insert( int data ){
      if( isFull() ){
         return false;
      }

      rear = ( rear + 1 ) % size;
      arr[rear] = data;
      return true;
   }
```

delete() :

```
int deleteData() {
      if( isEmpty() ){
         return -999;
      }
     front = ( front + 1 ) % size;
      return arr[front];
  }
```

9. Difference between Stack and Queue?

Sr.No	Stack	Queue
1	A Stack Data Structure works on Last In First Out (LIFO) principle.	A Queue Data Structure works on First In First Out (FIFO) principle.
2.	Push and pop operations are done from same end i.e "top"	Push and pop operations are done from same end i.e "rear" and "front" respectively
3.	You can implement multi-stack approach	Threre are four types of Queue i.e linear, circular, priority and dequeue.
4.	Application: • Used in infix to postfix conversion, • scheduling algorithms • depth first search and evaluation of an expression	Application: • Printer mainatains queue of documents to be printed • OS uses queues for many functionalities : Ready Queue, Waiting Queue, Message Queue • To implements algo like Breadth first search.

10. Infix ,prefix, postfix

These are notations to represent math equations.

- Infix: A+B
- Prefix: +AB
- Postfix: AB+

To convert infix to prefix /postfix considers the priorities:

Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (*type*)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right
10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5	|	Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

Infix to postfix Evaluation rules:

1. Scan the infix exp. From left to right
2. If scan character is left parenthesis then push it into stack
3. If scan character is operand than that will display into postfix expression.
4. If scan character is operator than that will push into operator stack.
5. If scan character has higher precedence than stack operator then scanned operator will push into operator stack.
6. If scan character has less or equal precedence than stack operator then stack operator will pop out from stack to postfix expression and scanned operator will push into operator stack.

7. If scan character is right parenthesis then till left parenthesis of stack all operators will pop to postfix expression and left will remove from stack.
8. Repeat step 1 to step 6 until end of expression

Ex: A+ (B*C-(D/E^F)*G)*H

Symbol	Scanned	STACK	Postfix Expression	Description
1.		(		Start
2.	A	(	A	
3.	+	(+	A	
4.	(	(+(	A	
5.	B	(+(	AB	
6.	*	(+(*	AB	
7.	C	(+(*	ABC	
8.	-	(+(-	ABC*	'*' is at higher precedence than '-'
9.	(	(+(-(	ABC*	
10.	D	(+(-(	ABC*D	
11.	/	(+(-(/	ABC*D	
12.	E	(+(-(/	ABC*DE	
13.	^	(+(-(/^	ABC*DE	
14.	F	(+(-(/^	ABC*DEF	
15.	)	(+(-	ABC*DEF^/	Pop from top on Stack , that's why '^' Come first
16.	*	(+(-*	ABC*DEF^/	
17.	G	(+(-*	ABC*DEF^/G	
18.	)	(+	ABC*DEF^/G*-	Pop from top on Stack , that's why '^' Come first
19.	*	(+*	ABC*DEF^/G*-	
20.	H	(+*	ABC*DEF^/G*-H	
21.	)	Empty	ABC*DEF^/G*-H*+	END

Infix to prefix Evaluation rules:

1. Reserve the input string or start from right of the infix expression
2. Examine the next element in the input.
3. If it is operand , add it to output string .
4. If it is closing parenthesis ,push it on to stack.
5. If it is an operator ,then

- if stack is empty , push operator on stack.
- If the top of stack is closing parenthesis , push operator on stack
- If it has same or higher priority then the top of stack push operator on stack

- If it has lower priority the top, the pop stack and add it to post fix expression and push operator into stack
- Else pop the operator from the stack and add it to output string

6. if it is an opening parenthesis ,pop operators from stack and ass them to output string until a closing parenthesis is encountered.pop and discard the closing parenthesis.
7. If there is more input go to step 2.
8. If there is no more input , unstack the remaining operators and them to output string.
9. Reserve the output string.

((A+B)*(C+D)/(E-F)+G)

Input	Prefix_Stack	Stack
G	G	(empty)
+	G	+
)	G	+)
)	G	+))
F	GF	+))
-	GF	+)) -
E	GFE	+)) -
(	GFE-	+)
/	GFE-	+) /
)	GFE-	+) /)
D	GFE-D	+) /)
+	GFE-D	+) /) +
C	GFE-DC	+) /) +

11. Tree

A tree consists of nodes connected by edges, which do not form cycle.
For collection of nodes & edges to define as tree, there must be one & only one path from the root to any other node.
A tree is a connected graph of N vertices with N-1 Edges.

Tree Terminologies:

1. **Node:** A node stands for the item of information plus the branches of other items.
2. **Siblings:** Children of the same parent are siblings.
3. **Degree:** The number of sub trees of a node is called degree. The
4. degree of a tree is the maximum degree of the nodes in the tree.
5. **Leaf Nodes:** Nodes that have the degree as zero are called leaf nodes or terminal nodes. Other nodes are called non terminal nodes.

6. **Ancestor:** The ancestor of a node are all the nodes along the path from the root to that node.
7. **Level:** The level of a node is defined by first letting the root of the tree to be level = 1or level=0.
8. **Height/Depth:** The height or depth of the tree is defined as the maximum level of any node in the tree.

Binary Search Tree

A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties –

The left sub-tree of a node has a key less than or equal to its parent node's key.

The right sub-tree of a node has a key greater than to its parent node's key.

Insert () in to Binary search tree:

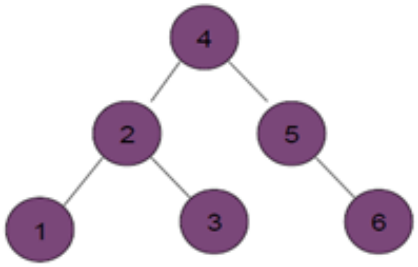
```
bool BST::insert(int data) {
   //create a node
       Node * newNode = new Node( data );
 //check for newnode and for duplicate data point
      if( newNode == NULL ) {
        return false;
   }
//check for tree is empty
      if( root == NULL ) {
        //if empty
      root = newNode;
      return true;
   }
//if tree is not empty  create temporary pointer
      Node * temp = root;
//compare data
      while( temp->getData() != data ) {
         if( data < temp->getData() ) {
         //insert to left
         //check if temp has left child
         if( temp->getLeft() == NULL ) {
            temp->setLeft( newNode );
            return true;
         }
         temp = temp->getLeft();
      }
      else {
         //insert to right
         //check if temp has right child
         if( temp->getRight() == NULL ) {
            temp->setRight( newNode );
            return true;
         }
         temp = temp->getRight();
    } }
   delete newNode;
  return false;
```

Delete () in to Binary search tree:

```
bool BinarySearchTree :: deleteData(int data) {
	if (root == NULL) {
		return false;
	}
	Node * del = root, *parent = root;
	//step 1 locate the node to be deleted along with the parent
	while (data != del->getData()) {
		if (data < del->getData()) {
			//to left
			parent = del;
			del = del->getLeft();
		}
		else {
			//to right
			parent = del;
			del = del->getRight();
		}
		if (del == NULL) {
			return false;
		}
	}
	while (true) {
		//check if del is terminal
		if (del->getLeft() == NULL && del->getRight() == NULL) {

			//check if the del is root node
			if (del == root) {
				delete del;

				root = NULL;

				return true;

			}

			if (del == parent->getLeft()) {

				parent->setLeft(NULL);

			}
			else {
				parent->setRight(NULL);
			}
			delete del;
			return true;
		}
```

```
//if del is non terminal node
                if (del->getLeft()) {
                        //select max from left
                        Node * max = del->getLeft();
                        parent = del;


                        while (max->getRight()) {

                                parent = max;

                                max = max->getRight();
                        }
                        int temp = max->getData();
                        max->setData(del->getData());
                        del->setData(temp);
                        del = max;
                }

                else {
                        //min from right
                        Node * min = del->getRight();
                        parent = del;

                        while (min->getLeft()) {
                                parent = min;
                                min = min->getLeft();
                        }

                        int temp = del->getData();
                        del->setData(min->getData());
                        min->setData(temp);
                        del = min;
                }
        }
        return false;
}
```

Tree traversals:



- **Preorder(Root-Left-Right)**
  421356
- **Inorder(Left-Root-Right)**
  123456
- **Postorder(Left-Right-Root)**
  132654

PreOrder:

```
void BST::preOrder() {
    Node * temp = root;
  Node * stack[100];
  int top = -1;
   cout<<"Pre : ";
  while( temp != NULL || top != -1 ) {
        while( temp ) {
        cout<<temp->getData()<<" ";
        stack[++top] = temp;
        temp = temp->getLeft();
      }
       temp = stack[top--];
      temp = temp->getRight();
    }
```

Algorithm for preorder Traversal:

1. Visit node
2. Backup its address (use stack)
3. goto left child
4. If temp is null then pop stack
5. go back to popped parent and goto it's right
6. Repeat till stack is empty or temp is not NULL

Inorder:

```
void BST::inOrder() {
   Node * temp = root;
   Node * stack[100];
   int top = -1;
   cout<<"In : ";
   while( temp != NULL || top != -1 ) {
       while( temp ) {
         stack[++top] = temp;
         temp = temp->getLeft();
       }
       temp = stack[top--];
       cout<<temp->getData()<<" ";
       temp = temp->getRight();
     }
}
```

Algorithm for Inorder Traversal:

1. Push temp to stack
2. Goto left of temp
3. Repeat till temp is NULL
4. now pop stack & get back to popped node
5. visit that popped node and then shift to its right child
6. Repeat till stack is empty or temp is not NULL

Postorder:

```
void BST::postOrder() {
      typedef struct {
      Node * address;
      char flag;
   } Pair;
   Node * temp = root;
   Pair stack[100];
   int top = -1;
   while( temp != NULL || top != -1) {
       while( temp ) {
         Pair p;
         p.address = temp; p.flag = 'L';
         stack[++top] = p;
         temp = temp->getLeft();
       }
       Pair p = stack[top--];
         if( p.flag == 'L') {
         temp = p.address->getRight();
         p.flag = 'R';
          stack[++top] = p; }
       else {
          cout<<p.address->getData()<<" ";
       } }}
```

Graph:

A Graph is a collection of nodes, which are called vertices 'V', connected in pairs by line segments, called Edges E.
Sets of vertices are represented as V(G) and sets of edges are represented as E(G).

So a graph is represented as G = (V, E).
There are two types of Graphs
- Undirected Graph
- Directed Graph

Directed Graph are usually referred as Digraph for Simplicity. A directed Graph is one, where each edge is represented by a specific direction or by a directed pair <v1, v2>.Hence <v1, v2> & <v2, v1> represents two different edges.

Terminology related to Graph:

- Out – degree: The number of Arc exiting from the node is called the out degree of the node.
- In- Degree: The number of Arcs entering the node is the In degree of the node.
- Sink node: A node whose out-degree is zero is called Sink node.
- Path: path is sequence of edges directly or indirectly connected between two nodes.
- Cycle: A directed path of length at least L which originates and terminates at the same node in the graph is a cycle

The graphs can be implemented in two ways
1. Array Method
2. Linked List

Graph traversal
1. **Breadth first search**

```
void Graph::bfs(int v) {
        int * visited = new int[noOfVertices] ;
        int q[100];
        int front = -1, rear = -1;
        q[0] = v;
        front = rear = 0;
        cout << "BFS : ";
        while (front <= rear) {
        v = q[front++];
        if (visited[v] == 0) {
        cout << vertices[v] << " ";
        visited[v] = 1;
        for (int i = 0; i < noOfVertices; i++) {
                if (adjMat[v][i] == 1 && visited[i] == 0) {
                q[++rear] = i;
                } }}

        delete[] visited;

}
```

2. Depth first search

```
void Graph::dfs(int v) {
        int * visited = new int[noOfVertices] ;
        int stack[100];
        int top = -1;
        cout << "DFS : ";
        cout << vertices[v] << " ";
        visited[v] = 1;
        stack[++top] = v;
        while (top != -1) {
                for (int i = 0; i < noOfVertices; i++) {
                        if (adjMat[v][i] == 1 && visited[i] == 0) {
                                cout << vertices[i] << " ";
                                visited[i] = 1;
                                stack[++top] = i;
                                v = i;
                                i = -1;
                        }
                }
                v = stack[top--];}

}
```

Q. What is normalization?

Database normalization is a data design & organizational process applied to data structures based on rules that help to build relational databases.

In relational database design, the process of organizing data to minimize redundancy is called normalization.

Normalization usually involves dividing database data into different tables and defining relationships between the tables.

The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then retrieved through the rest of the database via the defined relationships.

The key traits for Normalization are eliminating redundant data and ensuring data dependencies.

Q. What is de-normalization?

De-normalization is the process of attempting to optimize the performance of a database by adding redundant data.

It is sometimes necessary because current DBMSs implement the relational model poorly.

A true relational DBMS would allow for a fully normalized database at the logical level, while providing physical storage of data that is tuned for high performance.

De-normalization is a technique to move from higher to lower normal forms of database modeling in order to **speed up database access**.

Q. How is the ACID property related to databases?

ACID (an acronym for Atomicity Consistency Isolation Durability)

For a reliable database, all four of these attributes should be achieved:

Atomicity is an all-or-none rule for database modifications.

Consistency guarantees that a transaction never leaves your database in a half-finished state.

Isolation keeps transactions separated from each other until they are finished.

Durability guarantees that the database will keep track of pending changes in such a way that the server can recover from an abnormal termination and committed transactions will not be lost.

Q. What is SQL?

SQL stands for Structured Query Language, and it is used to communicate with the Database. This is a standard language used to perform tasks such as retrieval, updation, insertion and deletion of data from a database.

Q. What is a stored procedure?

A stored procedure is a named group of SQL statements that have been previously created and stored in the server database.

Stored procedures are objects that do the work they are designed to do when you call upon them. You need to make sure they have what they need (the right values and parameters), so they can perform their important tasks. Stored procedures can also make updates, create objects, or even be set up to backup a database or perform other maintenance tasks.

Stored procedures accept input parameters so that a single procedure can be used over the network by several clients using different input data. When the procedure is modified, all clients automatically get the new version. Stored procedures **reduce network traffic and improve performance**.

Q. What is a Trigger?

A trigger is a SQL procedure initiates an action when an event (like INSERT, DELETE or UPDATE) occurs on an object. Based on events which take place in your database & actions to run automatically.

Triggers are stored in and managed by the DBMS.

Triggers can be used to maintain the referential integrity of data by changing the data in a systematic fashion.

A trigger cannot be called or executed directly; DBMS automatically fires the trigger as a result of a data modification to the associated table or in the case of DDL triggers to a DDL event in the database.

Triggers are similar to stored procedures like, both consist of procedural logic that is stored at the database level. Stored procedures, however, are not event-driven and are not attached to a specific table as most triggers are. Stored procedures are explicitly executed by invoking a call

to the procedure while triggers are implicitly executed by events. In addition, triggers can also execute stored procedures

Q. What are the different types of triggers?

Basically we have 13 types of triggers.

1. Row level triggers (6 types): **Row-level triggers** execute once for each **row** in a transaction.

2. Statement level triggers (6 types): A statement level trigger is also called as **table level trigger.** A **table level trigger** is a **trigger** that doesn't fire for each row to be changed.

3. Instead of trigger: Instead of trigger is use to write trigger on Views.

Row Level Triggers:

- before insert on each row
- before update on each row
- before delete on each row
- after insert on each row
- after update on each row
- after delete on each row

Statement Level Triggers:

- before insert
- before update
- before delete
- after insert
- after update
- after delete

Instead of Trigger: mainly using for complex views.

Q. What is a Cursor & type of Cursor?

A cursor is a temporary work area created in the system memory when a SQL statement is executed & we can give a name to it and manipulate multiple records. A cursor contains information on a select statement and the rows of data accessed by it. This temporary work area is used to store the data retrieved from the database, and manipulate this data. A cursor can hold more than one row, but can process only one row at a time. By default record pointer set to first record and when record processed then pointer automatically goes to second record. The set of rows the cursor holds is called the active set.

Types of cursors:-

1) Implicit cursors:-These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed. They are also created when a SELECT statement that returns just one row is executed.

2) Explicit cursors:-They must be created when you are executing a SELECT statement that returns more than one row. Even though the cursor stores multiple records, only one record can be processed at a time, which is called as current row. When you fetch a row the current row position moves to next row.

Q. which are the steps for using an Explicit Cursor?

There are four steps in using an Explicit Cursor.

1) DECLARE the cursor in the declaration section.

2) OPEN the cursor in the Execution Section.

3) FETCH the data from cursor into PL/SQL variables or records in the Execution Section.

4) CLOSE the cursor in the Execution Section before you end the PL/SQL Block.

Q. Which are Different Parameters in Procedure and Functions?

1) IN type parameter: These types of parameters are used to send values to stored procedures.

2) OUT type parameter: These types of parameters are used to get values from stored procedures. This is similar to a return type in functions.

3) INOUT parameter: These types of parameters are used to send values and get values from stored procedures.

Q. What is a View?

A view is a virtual table based on the result-set of an SQL statement.

It can be used for retrieving data as well as updating or deleting rows.

It cannot hold physical data or physical memory.

Two main purposes of creating a view are

1) Provide a security mechanism which restricts users to a certain subset of data.

2) Provide a mechanism for developers to customize how users can logically view the data.

Q. What is an index?

An index is a physical structure containing pointers to the data. Indices are created in an existing table to **locate rows more quickly and efficiently.** It is possible to create an index on one or more columns of a table, and each index is given a name. The users can see the index name but **cannot see the indices themselves**; **they are just used to speed up queries**. Effective indices are one of the best ways to improve performance of a database application.

An Index can give you improved query performance because a seek action occurs for retrieving records from your table in a query. A seek means you were able to locate record(s) without having to examine every row to locate those record(s).

A table scan occurs when there is no index available or when a poorly created index exists on the table for a query running against that table. In a table scan, Server examines every row in the table to satisfy the query results.

Q. What are Different Types of Joins?

Join allow us to **extract meaningful data** from more than one table **based on some column which is common** to them.

INNER JOIN / Equijoin: Returns records that have matching values in both tables

Cross join / Cartesian Join - The absence of join condition it takes **Cartesian** product of table. That is no of rows in table A & no of rows in table B.

Natural join: Returns records that have matching values in both tables.

Non-Equi join: Returns records base on non-matching values in both tables.

Self-join: Returns records that have matching values in same (self) table.

LEFT OUTER JOIN: Return all records from the left table, and the matched records from the right table

RIGHT OUTER JOIN: Return all records from the right table, and the matched records from the left table

FULL OUTER JOIN: Return all records when there is a match in either left or right table.

Q. What is a PRIMARY KEY?

A PRIMARY KEY constraint is a unique identifier for a row within a database table. A primary key prevents, duplicates, and ensures that all records have their own distinct values. Primary keys don't allow nulls, so you are guaranteed that each record has its own unique populated value.

Every table should have a primary key constraint to uniquely identify each row, and **only one primary key constraint can be created for each table**. The primary key constraints are used to enforce entity integrity.

Q. What is a UNIQUE KEY Constraint?

A UNIQUE constraint enforces the uniqueness of the values in a set of columns; so no duplicate values are entered. The unique key constraints are used to enforce entity integrity as the primary key constraints.

Primary key is also a unique key internally, but cannot allow NULLs. Unique keys on the other hand allow a single NULL but not multiple NULLs over the columns.

Q. What is a CHECK constraint?

A CHECK constraint is used to limit the values that can be placed in a column. CHECK constraints are most often used to enforce domain integrity.

Q. What is a NOT NULL constraint?

A not null constraint enforces that the column will not accept null values. Not null constraints are used to enforce domain integrity.

Q. What is a DEFAULT definition?

A DEFAULT definition is used to add values into a column when values were omitted. The default value must be compatible with the data type of the column to which the DEFAULT definition applies.

Q. What is the difference between the DELETE and TRUNCATE commands & Drop command?

The delete command removes the rows from a table on the basis of the condition that we provide in WHERE clause. After delete if we use rollback then we will get original records again.

Truncate will actually remove all of the rows from a table, and there will be no data in the table after we run the truncate command. After Truncate if we use rollback then we will not get original records again.

Drop will actually remove all records in table as well it will remove table structure from the database. After drop rollback command will not work.

Q. What is the difference between a HAVING clause and a WHERE clause?

The HAVING clause specifies a search condition for a GROUP BY or an aggregate. The difference is that **HAVING can be used only with the SELECT statement whereas the WHERE can be used during update and delete operations**. HAVING is typically used with a GROUP BY clause. The HAVING clause is used in an aggregate function or a GROUP BY clause in a query, whereas a **WHERE Clause is applied to each row before** they are part of the GROUP BY clause or aggregate function in a query.

Q. What are the requirements of sub-queries?

•A sub-query must be enclosed in the parenthesis.

•A sub-query must be put on the right hand of the comparison operator.

•A sub-query cannot contain an ORDER BY clause.

•A query can contain more than one sub-query.

Q. What is Data integrity?

Data integrity is the consistency and accuracy of the data which is stored in a database. A constraint performs data validation to help maintain database integrity by preventing invalid data from being entered.

Q. What is Co-related Sub Query?

A Co-related Sub Query is nested sub query which is executed once for each 'row' considered by the main query and which on execution uses a value from a column in the outer query.

In Co-related Sub Query, the column value used in inner sub query refers to the column value present in the outer query forming a Co-related Sub Query.

The Sub Query is executed repeatedly, once for each row of the main (outer) query table.

Q. Difference between Stored Procedure and function?

Basic Differences:

A Function must return a value but in Stored Procedures it is optional: a procedure can return 0 or n values.

Functions can have only input parameters for it, whereas procedures can have input/output parameters.

For a Function it is mandatory to take one input parameter, but a Stored Procedure may take 0 to n input parameters.

Functions can be called from a Procedure whereas Procedures cannot be called from a Function.

Advanced Differences:

Exceptions can be handled by try-catch blocks in a Procedure, whereas a try-catch block cannot be used in a Function.

We can go for Transaction Management in a Procedure, whereas in a Function we can't.

In SQL:

A Procedure allows SELECT as well as DML (INSERT, UPDATE, and DELETE) statements in it, whereas Function allows only SELECT statement in it.

Procedures cannot be utilized in a SELECT statement, whereas Functions can be embedded in a SELECT statement.

Stored Procedures cannot be used in SQL statements anywhere in a WHERE (or a HAVING or a SELECT) block, whereas Functions can.

Q. Define The Integrity Rules?

There are two Integrity rules.

Entity Integrity: States that Primary key cannot have NULL value

Referential Integrity: States that Foreign Key can be either a NULL value or should be Primary Key value of other relation.

Q. What is a difference between Commit, Rollback and Savepoint?

COMMIT: Ends the current transaction by making all pending data changes permanent.

ROLLBACK: Ends the current transaction by discarding all pending data changes.

SAVEPOINT: Divides a transaction into smaller parts. You can rollback the transaction till a particular named savepoint.

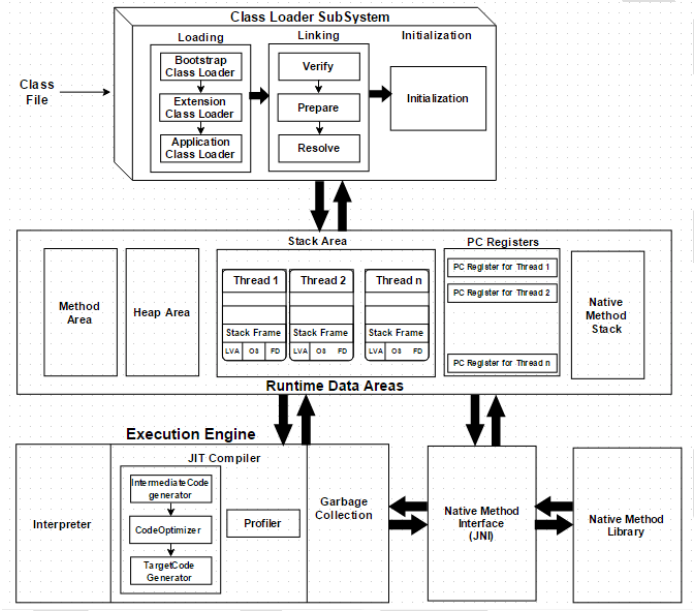
1. Explain JVM,JRE,JDK?

Java Development Kit [JDK] is the core component of Java Environment and provides all the tools, executable and binaries required to compile, debug and execute a Java Program. JDK is a platform specific software and that's why we have separate installers for Windows, Mac and Unix systems.

Java Virtual Machine[JVM] is the heart of java programming language. When we run a program, JVM is responsible to converting Byte code to the machine specific code. JVM is also platform dependent and provides core java functions like memory management, garbage collection, security etc.

Java Runtime Environment [JRE] is the implementation of JVM, it provides platform to execute java programs. JRE consists of JVM and java binaries and other class libraries to execute any program successfully. To execute any java program, JRE is required.

2. Explain journey of java program from source code to execution? Or Explain JVM architecture?



As shown in the above architecture diagram, JVM subsystems are :

- Class Loader Subsystem
- Runtime Data Area
- Execution Engine

1. Class Loader Subsystem

- *Loading :* Class loader dynamically loads java classes. It loads, links and initializes the class file when it refers to a class for the first time at runtime, not compile time. Class Loaders follow Delegation Hierarchy Algorithm while loading the class files. 3 types are,
  - **Boot Strap** Class Loader – Responsible for loading classes from the bootstrap classpath, nothing but **rt.jar.** Highest priority will be given to this loader.
  - **Extension Class Loader** – Responsible for loading classes which are inside the ext folder **(jre\lib).**
  - **Application Class Loader** –Responsible for loading Application Level Class path, path mentioned Environment Variable etc.
- ***Linking***:
  - **Verify** – Byte code verifier will verify byte codeusing checksum.

- **Prepare** – For all static variables memory will be allocated and assigned with default values.
- **Resolve** – All symbolic memory references are replaced with the original references from Method Area.

➢ Initialization: Here all static variables will be assigned with the original values, and the static block will be executed.

2. Runtime Data Area

The Runtime Data Area is divided into 5 major components:

➢ **Method Area** – All the class level data will be stored here, including static variables. There is only one method area per JVM, and it is a shared resource.

➢ **Heap Area** – All the Objects and their corresponding instance variables and arrays will be stored here. There is also one Heap Area per JVM. Since the Method and Heap areas share memory for multiple threads, the data stored is not thread-safe.

➢ **Stack Area** – For every thread, a separate runtime stack will be created. For every method call, one entry will be made in the stack memory which is called as Stack Frame. All local variables will be created in the stack memory. The stack area is thread-safe since it is not a shared resource. The Stack Frame is divided into three sub entities:

- **Local Variable Array** – Related to the method how many local variables are involved and the corresponding values will be stored here.
- **Operand stack** – If any intermediate operation is required to perform, operand stack acts as runtime workspace to perform the operation.
- **Frame data** – All symbols corresponding to the method is stored here. In the case of any **exception**, the catch block information will be maintained in the frame data.

➢ **PC Registers** – Each thread will have separate PC Registers, to hold the address of current executing instruction once the instruction is executed the PC register will be updated with the next instruction.

➢ **Native Method stacks** – Native Method Stack holds native method information. For every thread, a separate native method stack will be created.

3. Execution Engine

The byte code which is assigned to the **Runtime Data Area** will be executed by the Execution Engine. The Execution Engine reads the byte code and executes it piece by piece.

➢ **Interpreter** – The interpreter interprets the byte code faster, but executes slowly. The disadvantage of the interpreter is that when one method is called multiple times, every time a new interpretation is required.

➢ **JIT Compiler** – The JIT Compiler neutralizes the disadvantage of the interpreter. The Execution Engine will be using the help of the interpreter in converting byte code, but when it finds repeated code it uses the JIT compiler, which compiles the entire bytecode and changes it to native code. This native code will be used directly for repeated method calls, which improve the performance of the system.

- **Intermediate Code generator** – Produces intermediate code
- **Code Optimizer** – Responsible for optimizing the intermediate code generated above
- **Target Code Generator** – Responsible for Generating Machine Code or Native Code
- **Profiler** – A special component, responsible for finding hotspots, i.e. whether the method is called multiple times or not.

4. **Garbage Collector:**Collects and removes unreferenced objects
5. **Java Native Interface (JNI): JNI will be** interacting with the Native Method Libraries and provides the Native Libraries required for the Execution Engine.

6. **Native Method Libraries**: This is a collection of the Native Libraries which is required for the Execution Engine.

3. What is difference between Heap and Stack Memory?

Heap memory is used by all the parts of the application whereas stack memory is used only by one thread of execution. Whenever an object is created, it's always stored in the Heap space and stack memory contains the reference to it. Stack memory only contains local primitive variables and reference variables to objects in heap space. Memory management in the stack is done in a LIFO manner whereas it's more complex in Heap memory because it's used globally.

4. What is Bytecode?

Bytecode is in a compiled Java programming language [by javac command] format and has the .class extension executed by Java Virtual Machine (JVM).

The Java bytecode gets processed by the Java virtual machine (JVM) instead of the processor. The JVM transforms program code into readable machine language for the CPU because platforms utilize different code interpretation techniques. A JVM converts bytecode for platform interoperability, but bytecode is not platform-specific.JVM is responsible for processing & running the bytecode.

5. What is JIT?

The magic of java "Write once, run everywhere" is bytecode. JIT improves the performance of Java applications by compiling bytecode to native machine code at run time. JIT is activated when a Java method is called. The JIT compiler compiles the bytecode of that method into native machine code, compiling it "just in time" to run. When a method has been compiled, the JVM calls the compiled code of that method directly instead of interpreting it.

Typical compilers take source code and completely convert it into machine code, JITs take the same source code and convert it into an intermediary "assembly language," which can then be pulled from when it's needed. And that's the key. Assembly code is interpreted into machine code on call—resulting in a faster translation of only the code that you need. JIT have access to dynamic runtime information and are able to optimize code. JITs monitor and optimize while they run by finding code more often called to make them run better in the future.

JITs reduce the CPU's workload by not compiling everything all at once, but also because the resulting compiled code is optimized for that particular CPU. It's why languages with JIT compilers are able to be so "portable" and run on any platform or OS.

6. Why java is platform independent?

Java is a platform independent programming language, because your source code can be executed on any platform [e.g. Windows, Mac or Linux etc..]. When you install JDK software on your system , JVM is automatically installed on your system. When we compile Java code then .class file or bytecode is generated by javac compiler. For every operating system separate JVM is available which is capable to read the **.class** file or **byte code** and execute it by converting to native code for that specific machine. We compile code once and run everywhere.

7. What are the main differences between the Java platform and other platforms?

There are the following differences between the Java platform and other platforms. Java is the software-based platform whereas other platforms may be the hardware platforms or software-based platforms. Java is executed on the top of other hardware platforms whereas other platforms can only have the hardware components.

8. What gives Java its 'write once and run anywhere' nature?

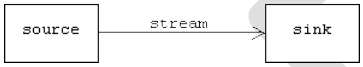
The bytecode. Java compiler converts the Java programs into the class file (Byte Code) which is the intermediate language between source code and machine code. This bytecode is not platform specific and can be executed on any computer.

9. Why char data type in java have 2 byte size?

In old languages like C and C++ we can use only ASCII characters and to represent all ASCII characters 8-bits are enough. Hence char size is 1 byte. But in Java we can use Unicode characters which cover worldwide all alphabets sets. The number of Unicode characters is greater than 256 and hence 1 byte is not enough to represent all characters. Hence we should go for 2 bytes.

10. What is a Stream? Types of streams in java?

Basic:A stream is a sequence of bytes or characters that connects a data source to a destination/ sink.



Technically, both source and sink are programs. For example, the source might be an application and the sink might be an application ,file on a disk, printer, or monitor. We refer to the stream as a pipe.The source places data (bytes or characters) into the stream using a special non-blocking write operation & sink extracts data using a special read operation.

Streams in java have types as byte streams &char streams.

Java provides abstract **InputStream** and **OutputStream** classes for representing byte streams. This hierarchy specifies an abstract method for reading/writting a single byte to/from a source/destination.

Also provides abstract **Reader** and **Writer** classes for representing char streams,having abstract method for reading/writing a single char to/from a source/destination.

Advanced:In Java 8, new Stream API is added. It's used to process collections of objects. It can be pipelined to produce the desired result.

- takes input from the Collections, Arrays or I/O channels.
- keeps original data structure, they only provide the result as per the pipelined methods.
- operations are lazy and returns a stream as a result so pipelined.

11. What is Serialization and Deserialization [Marshaling/Unmarshaling]?

Serialization is the process of converting an object into a stream of bytes to store the object or transmit it to memory, a database, or a file. Its main purpose is to save the state of an object in order to be able to recreate it when needed. The reverse process is called deserialization.

Through serialization, a developer can perform actions like sending the object to a remote application by means of a Web Service, passing an object to a file or over network, maintaining security or user-specific information across applications.

Implement the Serializable interface on class whose instance is to be serialized. An exception is thrown if you attempt to serialize but the type doesn't have implemented the Serializable interface [java.io.NotSerializableException].

12. What is the main purpose of serialization in java?

The main uses of serialization are :

- Persistence: We can write data to a file or database and can be used later by deserializing it.
- Communication : To pass an object over network by making remote procedure call.
- Copying : We can create duplicates of original object by using byte array. 4) To distribute objects across different JVMs.

13. What are alternatives to java serialization?

- XML based data transfer
- JSON based data transfer.

XML based data transfer : We can use JIBX or JAXB where we can marshall our object's data to xml and transfer data and then unmarshall and convert to object.

JSON based transfer : We can use json to transfer data.

14. Explain about serializable interface in java?

To implement serialization in java there is an interface defined in java.io package called serializable interface. Java.io.Serializable interface is an marker interface which doesnot contain any any methods. A class implements Serializable lets the JVM know that the instances of the class can be serialized.

Syntax: public interface Serializable { }

15. How to make object serializable in java?

- Our class must implement serializable interface.If our object contains other objects those class must also implement serializable interface.
- We use ObjectOutputStream which extends OutputStream used to write objects to a stream.
- We use ObjectInputStream which extends InputStream used to read objects from stream

16. What is serial version UID and its importance in java? Serial version unique identifier is a 64 bit long value ?

This 64 bit long value is a hash code of the class name,super interfaces and member. Suid is a unique id no two classes will have same suid. Whenever an object is serialized suid value will also serialize with it. When an object is read using ObjectInputStream, the suid is also read. If the loaded class suid does not match with suid read from object stream, readObject throws an InvalidClassException.

17. Explain use of static keyword in java?

The **static keyword** is used in java mainly with variables, methods, blocks and nested class.

Static Variables gets memory allocated only once in the class area at the time of class loading.Sharable among all objects of class.e.g. course for all students is common

Static methodsscenarios

- If you are writing utility classes and they are not supposed to be changed.
- If the method is not using any instance variable.
- If any operation is not dependent on instance creation.
- If there is some code that can easily be shared by all the instance methods, extract that code into a static method.
- If you are sure that the definition of the method will never be changed or overridden. As static methods cannot be overridden.

Static initializer/Block

Java provides a feature called a static initializer that's designed specifically to let you initialize static fields.

Static class

A class can be made static only if it is a nested class.

- Nested static class doesn't need reference of Outer class
- A static class cannot access non-static members of the Outer class

18. What is the default value of the local variables?

The local variables are not initialized to any default value, neither primitives nor object references.

19. What are the advantages of Packages in Java?

There are various advantages of defining packages in Java. Packages avoid the name clashes. The Package provides easier access control. We can also have the hidden classes that are not visible outside and used by the package. It is easier to locate the related classes.

20. What is the difference between import java.util.Date and java.util.* ?

The star form (java.util.*) includes all the classes of that package and that may increase the compilation time – especially if you import several packages. However it doesn't have any effect run-time performance.

21. **What is static import?**

Static import allows you to access the static member of a class directly without using the fully qualified name.

If you are going to use static variables and methods a lot then it's fine to use static imports. for example if you want to write a code with lot of mathematical calculations then you may want to use static import.

e.g. import static java.lang.Math.*;

22. What is the difference between an object-oriented programming language and object-based programming language?

There are the following basic differences between the object-oriented language and object-based language. Object-oriented languages follow all the concepts of OOPs whereas, the object-based language doesn't follow all the concepts of OOPs like inheritance and polymorphism. Object-oriented languages do not have the inbuilt objects whereas Object-based languages have the inbuilt objects, for example, JavaScript has window object. Examples of object-oriented programming are Java, C#, Smalltalk, etc. whereas the examples of object-based languages are JavaScript, VBScript, etc.

23. **What is Object class? Explain methods inside?**

Class **Object** is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.

- clone() - Creates and returns a copy of this object.
- equals() - Indicates whether some other object is "equal to" this one.
- finalize() - Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
- getClass() - Returns the runtime class of an object.
- hashCode() - Returns a hash code value for the object.
- notify() - Wakes up a single thread that is waiting on this object's monitor.
- notifyAll() - Wakes up all threads that are waiting on this object's monitor.
- toString() - Returns a string representation of the object.
- wait() - Causes current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.

24. What is meant by Local variable and Instance variable?

Local variables are defined in the method and scope of the variables that have existed inside the method itself. An instance variable is defined inside the class and outside the method and scope of the variables exist throughout the class.

25. **Difference between this() and super() in java ?**

this() is used to access one constructor from another with in the same class while super() is used to access superclass constructor. Either this() or super() exists it must be the first statement in the constructor.

26. What is method overloading?

Method overloading is the polymorphism technique which allows us to create multiple methods with the same name but different signature. We can achieve method overloading in two ways. Changing the number of arguments or Changing the sequence of arguments. Method overloading increases the readability of the program.

27. What is method overriding?

If a subclass provides a specific implementation of a method that is already provided by its parent class, it is known as Method Overriding. It is used for runtime polymorphism and to implement the interface methods. Rules for Method overriding The method must have the same name as in the parent class. The method must have the same signature as in the parent class. Two classes must have an IS-A relationship between them.

28. What is association?

Association is a relationship where all object have their own lifecycle and there is no owner. Let's take an example of Teacher and Student. Multiple students can associate with a single teacher and a single student can associate with multiple teachers but there is no ownership between the objects and both have their own lifecycle. This relationship can be one to one, one to many, many to one and many to many.

29. What do you mean by aggregation?

Aggregation is a specialized form of Association where all object have their own lifecycle but there is ownership and child object cannot belongs to another parent object. Let's take an example of Department and teacher. A single teacher cannot belongs to multiple departments, but if we delete the department teacher object will not destroy.

30. What is wrapper class? Give Examples? What is boxing / unboxing in java?

Many times we need to use objects in place of primitives [e.g. Collection framework works only with objects] .So java platform provides wrapper classes for each of the primitive data types. These classes "wrap" the primitive in an object.

e.g. Integer,Boolean,Double etc… are wrapper classes which wrap primitive types int,boolean,double

Autoboxing is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes. For example, converting an int to an Integer, a double to a Double, and so on. If the conversion goes the other way, it is called ***unboxing***.

e.g. int i=100;

Integer Iobj=I; ///boxing

Int j=Iobj;//unboxing

31. Explain String in java ? What is SCP [string constant pool]? Why strings are immutable?

In java String is final class. If we create object of String it gets stored in SCP. String objects are most used data objects in Java. Hence, java has a special arrangement to store the string objects. String Constant Pool is memory space in heap memory specially allocated to store the string objects created using string literals. Whenever you create a string object using string literal, JVM first checks the content of the object to be created. If there exist an object in the string constant pool with the same content, then it returns the reference of that object. It doesn't create a new object. If the content is different from the existing objects then only it creates new object.

Immutability:String is immutable in Java. An immutable class is simply a class whose instances cannot be modified. All information in an instance is initialized when the instance is created and the information cannot be modified.

- Due to immutability you do not need to calculate hash code frequently, more efficient.
- Immutable objects are thread safe.
- String is widely used for e.g. network connection, opening files, etc. If not immutable, a connection or file would be changed and this can lead to a serious security threat.

32. Explain Mutable String Classes in java or Explain StringBuffer/StringBuilder?

In java String is final class & Immutable, but if want modifiable strings you have to use StringBuffer & StringBuilder classes. Both *StringBuilder* and *StringBuffer* create objects that hold a mutable sequence of characters. StringBuffer is synchronized and therefore thread-safe. StringBuilder is compatible with StringBuffer but with no guarantee of synchronization. Because StringBuilder not a thread-safe implementation, it is faster and it is recommended to use it in places where there's no need for thread safety.

33. What is the purpose of a default constructor?

The purpose of the default constructor is to assign the default value to the objects. The java compiler creates a default constructor implicitly if there is no constructor in the class.

34. What are the main uses of 'this' keyword?

Keyword **this** can be used to refer to the current class instance variable. this can be used to invoke current class method (implicitly) this() can be used to invoke the current class constructor. this can be passed as an argument in the method call. this can be passed as an argument in the constructor call. this can be used to return the current class instance from the method

35. What is hash code? Explain significance of hashCode() & equals method?

As name suggest Object class's equals() method used for comparing two objects together. The equals() method if it return true it mean the two objects are equal or return false otherwise. But equals() method compare the two objects unlike using == operator in java. equals() method is define to compare two objects semantically i.e it compares value of member variable of objects, whereas the == operator compares two objects by comparing their references. One thing to be note is the default implementation of equals() method in the Object class compares references of two objects. That means we have to override it in our classes for semantic comparison. All wrapper classes like Integer, Double, Long, String etc. implement the equals () method accordingly.

This method returns a hash code value for specific object. It is used by HashTable based collections like HashTable, HashSet and HashMap to store objects in small containers called "bucket". The default implementation of hashCode () in the Object class returns an integer number which is the logical memory address of the object. We can override it in our own classes.

36. What is Interface? Explain Concept of Marker Interface?

Interfaces are used for Role Based Inheritance & Programming by Contract. When we want a common role/ behavior to be implemented by multiple classes in different hierarchies we use interfaces. Also for creating a contract as the class that implements an interface must implement all the methods declared in the interface. In the Java programming language, an interface is a reference type, similar to a class, which can contain only constants, method signatures, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods. Interfaces cannot be instantiated—they can only be implemented by classes or extended by other interfaces.

Java8 have a new feature called functional Interface i.e. interface with only one method. Functional Interfaces are used to implement Lambda Expressions in java. Also Java8 allows to define default implementation of a method in interface using default keyword.

Marker Interfaces are interfaces with no methods inside i.e. empty. They are used to mark or supply extra information about a class facilities to JVM. e.g. Serializable

37. What is abstract class?

An abstract class is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be inherited..

An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon). If a class includes abstract methods, then the class itself must be declared abstract.

When an abstract class is inherited, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract.

38. Differentiate Abstract class & Interface?

Abstract classes are similar to interfaces. You cannot instantiate them, and they may contain a mix of methods declared with or without an implementation. However, with abstract classes, you can declare fields that are not static and final, and define public, protected, and private concrete methods. With interfaces, all fields are automatically public, static, and final, and all methods that you declare or define (as default methods) are public. In addition, you can extend only one class, whether or not it is abstract, whereas you can implement any number of interfaces.

Usage of Abstract class

You want to share code in same hierarchy. Have many common methods or fields, or require access modifiers other than public (such as protected and private).Declare non-static or non-final fields.

Usage of Interface

Common behavior to inject to many unrelated classes. Not concerned about who implements behavior.

39. Why does Java not support pointers?

The pointer is a variable that refers to the memory address. They are not used in Java because they are unsafe(unsecured) and complex to understand.

40. Why is multiple inheritance not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java. Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class. Since the compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have the same method or different, there will be a compile time error.

Example:

```
class A{
        void msg(){
                System.out.println("Hello");
        }
}
class B{
        void msg(){
```

```
                    System.out.println("Welcome");
                }
    }
    class C extends A,B{
        public static void main(String args[]){
        C obj=new C();
        obj.msg();//Now which msg() method would be invoked?
        }
    }
```

Output: Compile Time Error

41. What is Exception in java? How to handle Exceptions in java?

An *exception* is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions. When an error occurs within a method, the method creates an object and hands it off to the runtime system. The object, called an *exception object*, contains information about the error, including its type and the state of the program when the error occurred. Creating an exception object and handing it to the runtime system is called *throwing an exception*.

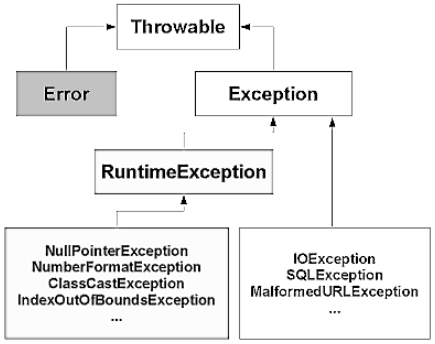
When an appropriate handler is found, the runtime system passes the exception to the handler , which typically catches the exception and have some alternative logic of code to handle the exception. Java provides 5 keywords for Exception handling mechanism: try, catch, throw, throws & finally.

First step in constructing an exception handler is to enclose the code that might throw an exception within a try block. If an exception occurs within the try block, that exception is handled by an exception handler associated with it. Catch is parameterized block with argument type, *Exception Type*. Single try can have multiple catch blocks. Java7 added a new feature that we can catch multiple exceptions in a single catch block with pipe(|) operator.

In Java exception handling, **throw** keyword is used to explicitly throw an exception from a method or constructor. And **throws** keyword is used declare the list of exceptions which may be thrown by that method or constructor.

42. What are Checked & Unchecked Exceptions?

Checked Exceptions are exceptions compiler forces to handle. These are classes extended from Exception but not under Runtime Exception. Java wants you to handle them because they somehow are dependent on external factors outside your program. A checked exception indicates an expected problem that can occur during normal system operation. Mostly these exceptions happen when you try to use external systems over network or in file system. An unchecked exception simply doesn't force you to handle them as they are mostly generated at runtime due to programmatic errors. They extend **RuntimeException**.



43. What is purpose of the finally Block?

The finally block always executes when the try block exits. A finally block makes sure that however you exit that block, it will get executed. That's important for deterministic cleanup of resources. Putting cleanup code in a finally block is always a good practice.Even though our application is closed forcefully there will be some tasks, which we must execute (like memory release, closing database, release lock, etc), if you write these lines of code in the finallyblock it will execute whether an exception is thrown or not...

44. What is the difference between error and an exception?

An error is an irrecoverable condition occurring at runtime such as Out Of Memory error. These JVM errors and you can not repair them at runtime. While exceptions are conditions that occur because of bad input etc. e.g. FileNotFoundException will be thrown if the specified file does not exist. Or a NullPointerException will take place if you try using a null reference. In most of the cases it is possible to recover from an exception (probably by giving user a feedback for entering proper values etc.

45. What is OutOfMemoryError in Java? How do you deal with that?

The Java virtual machine throws java.lang.OutOfMemoryError when there is not enough memory to run the application e.g. no more memory to create new objects, no more memory to create new threads etc. The most common OutOfMemoryError is the java.lang.OutOfMemoryError: java heap space, which comes when there is no more memory left to create a new object.

46. What is the use of the finally block? Is finally block in Java guaranteed to be called? When finally block is NOT called?

The finally block always executes when the try block exits. This ensures that the finally block is executed even if an unexpected exception occurs. But finally is useful for more than just exception handling — it allows having cleanup code accidentally bypassed by a return, continue, or break. Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.

If the JVM exits while the try or catch code is being executed, then the finally block may not execute. Likewise, if the thread executing the try or catch code is interrupted or killed, the finally block may not execute even though the application as a whole continues.

47. Can we catch more than one exception in single catch block?

From Java 7, we can catch more than one exception with single catch block. This type of handling reduces the code duplication. Note : When we catch more than one exception in single catch block , catch parameter is implicity final. We cannot assign any value to catch parameter.

Ex :

```
catch(ArrayIndexOutOfBoundsException || ArithmeticException e)
 {……. }
```

In the above example e is final we cannot assign any value or modify e in catch statement

48. What are user defined exceptions?

To create customized error messages we use userdefined exceptions. We can create user defined exceptions as checked or unchecked exceptions. We can create user defined exceptions that extend Exception class or subclasses of checked exceptions so that userdefined exception becomes checked. Userdefined exceptions can extend RuntimeException to create userdefined unchecked exceptions.

Note : It is recommended to keep our customized exception class as unchecked,i.e we need to extend Runtime Exception class but not Excpetion class.

49. Explain the importance of throwable class and its methods?

Throwable class is the root class for Exceptions. All exceptions are derived from this throwable class. The two main subclasses of Throwable are Exception and Error.

The three methods defined in throwable class are :

1) void printStackTrace() : This prints the exception information in the following format : Name of the exception, description followed by stack trace.

2) getMessage() This method prints only the description of Exception.

3) toString(): It prints the name and description of Exception.

50. Explain when ClassNotFoundException will be raised ?

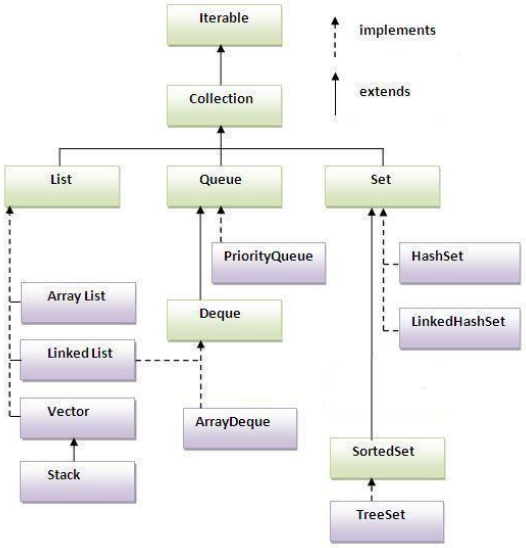
When JVM tries to load a class by its string name, and couldn't able to find the class classNotFoundException will be thrown. An example for this exception is when class name is misspelled and when we try to load the class by string name hence class cannot be found which raises ClassNotFoundException.

51. Explain when NoClassDefFoundError will be raised ?

This error is thrown when JVM tries to load the class but no definition for that class is found NoClassDefFoundError will occur. The class may exist at compile time but unable to find at runtime. This might be due to misspelled classname at command line, or classpath is not specified properly , or the class file with byte code is no longer available.

52. What is Collections Framework?

Java Collections framework is consist of the interfaces and classes which helps in working with different types of collections such as **lists, sets, maps, stacks and queues** etc.



- ***List*** : A List is an ordered Collection Lists may contain duplicate elements.
  ArrayList :It's Resizable-array implementation of the List interface which internally uses dynamic array. This class is roughly equivalent to Vector, except that it is unsynchronized i.e. not thread safe.
  LinkedList:Doubly-linked list implementation of the List and Deque interfaces.This is also unsynchronized.
- ***Set***: A Set is a Collection that cannot contain duplicate elements.
  HashSet: Uses hashtable (actually a HashMap instance) to store elements.It does not guarantee that the order of elements will remain constant.

TreeSet : Its NavigableSet implementation .The elements are ordered using their natural ordering, or by a Comparator provided at set creation time.Results into Sorted set of elements.

- ***Map:*** It is a key-value pair type.A Map cannot contain duplicate keys; each key can map to at most one value.

  HashMap :The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls. It does not guarantee that the order of items will remain constant over time.

  TreeMap:A Red-Black tree based NavigableMap implementation. The map is sorted according to the natural ordering of its keys, or by a Comparator provided at map creation time, depending on which constructor is used. It is unsynchronized.

53. What is an Iterator ?

The Iterator interface is used to step through the elements of a Collection. Iterators let you process each element of a Collection.

Iterators are a generic way to go through all the elements of a Collection no matter how it is organized. Iterator is an Interface implemented a different way for every Collection.

54. Difference between Comparable & Comparator? How you sort ArrayList with Comparable & Comparator?

We generally use Collections.sort() method to sort a simple array list. However if the Array List is of **custom object** type then in such case you have two options for sorting- **comparable and comparator** interfaces.

Since Comparable is implemented by the same class whose objects are sorted so it binds code with that sorting logic which is ok in most of the cases but in case you want to have more than way of sorting your class objects you should use comparators.

55. Difference between Vector& ArrayList?

	ArrayList	Vector
Performance	ArrayList is fast as it is non synchronized	Vector is slow as it is thread safe
Automatic Increase in Capacity	A Vector defaults to doubling size of its array .	it increases its Array size by 50%
Set Increment Size	ArrayList does not define the increment size .	Vector defines the increment size
Traversals	ArrayList can only use Iterator	Vector is the only other class which uses both Enumeration and Iterator
Legacy	ArrayList is not a legacy class. It is introduced in JDK 1.2.	Vector is a Legacy class.

56. Difference between ArrayList& LinkedList?

	ArrayList	LinkedList
Implementation	Resizable Dynamic Array	Doubly-LinkedList
Reverse Iterator	No	Yes , descendingIterator()
Initial Capacity	10	Constructs empty list
Get operation	Fast	Slow in comparison
Add operation	Slow in comparison	Fast

Memory Overhead	No	Yes
Thread Safe	No Not Synchronized	No Not Synchronized
Best For	for storing and accessing data.	Manipulating data.

57. Difference between HashMap & TreeMap?

	HashMap	TreeMap
Ordering	No	Yes
Implementation	Hashing	Red-Black Tree
Null Keys and Null values	One null key ,Any null values	Do not permit null keys ,but any null values
Performance	O(1)	log(n)
Speed	Fast	Slow in comparison
Functionality	Provide Basic functions	Provide Rich functionality
Comparison	uses equals() method	uses compareTo() method
Interface implemented	Map	Navigable Map

58. What is load factor?

The load factor represents at what level the capacity of any storage object like List,Map,Tree etc should be increased.

Every object had initial capacity and when that capacity got filled till certain factor then it's capacity should increased or doubled.

Default initial capacity of the HashMap takes is 16 and load factor is 0.75f (i.e 75% of current map size). The load factor represents at what level the HashMap capacity should be doubled.

59. What is difference between fail-safe and fail-fast iterators?

Iterators in java are used to iterate over the Collection objects. Fail-Fast iterators immediately throw *ConcurrentModificationException* if there is **structural modification** of the collection. Structural modification means adding, removing or updating any element from collection while a thread is iterating over that collection. Iterator on ArrayList, HashMap classes are some examples of fail-fast Iterator.

Fail-Safe iterators don't throw any exceptions if a collection is structurally modified while iterating over it. This is because, they operate on the clone of the collection, not on the original collection and that's why they are called fail-safe iterators. Iterator on CopyOnWriteArrayList, ConcurrentHashMap classes are examples of fail-safe Iterator

60. What is a linkedHashSet? How is different from a HashSet?

A HashSet is unordered and unsorted Set. LinkedHashSet is the ordered version of HashSet.The only difference between HashSet and LinkedHashSet is thatLinkedHashSet maintains the insertion order. When we iterate through a HashSet, the order is unpredictable while it is predictable in case of LinkedHashSet.

61. What is a TreeSet? How is different from a HashSet?

Major difference between HashSet and TreeSet is performance. HashSet is faster than TreeSet and should be preferred choice if sorting of element is not required. HashSet doesn't guaranteed any order while TreeSet maintains objects in Sorted order defined by either Comparable or Comparator method in Java.

62. What is a TreeMap? How is different from a HashMap?

HashMap is implemented as a hash table, and there is no ordering on keys or values. TreeMap is implemented based on red-black tree structure, and it is ordered by the key defined by either Comparable or Comparator method in Java.

63. What is BlockingQueue ? Can you give example implementations of the BlockingQueue interface?

BlockingQueue in Java is added in Java 1.5 along with various other concurrent Utility classes like ConcurrentHashMap, Counting Semaphore, CopyOnWriteArrrayList etc. BlockingQueue is a unique collection type which not only store elements but also supports flow control by introducing blocking if either BlockingQueue is full or empty. take() method of BlockingQueue will block if Queue is empty and put() method ofBlockingQueue will block if Queue is full. This property makes BlockingQueue an ideal choice for implementing Producer consumer design pattern where one thread insert element into BlockingQueue and other thread consumes it.

ArrayBlockingQueue and LinkedBlockingQueue are common implementation of BlockingQueue <E> interface. ArrayBlockingQueue is backed by array and Queue impose orders as FIFO. head of the queue is the oldest element in terms of time and tail of the queue is youngest element. ArrayBlockingQueue is also fixed size bounded buffer on the other hand LinkedBlockingQueue is an optionally bounded queue built on top of Linked nodes. In terms of throughput LinkedBlockingQueue provides higher throughput than ArrayBlockingQueue in Java.

64. What are atomic operations in Java?

Atomic operations are performed in a single unit of task without interference from other operations. Atomic operations are necessity in multi-threaded environment to avoid data inconsistency.
"The Java language specification guarantees that reading or writing a variable is an atomic operation(unless the variable is of type long or double). Operations variables of type long or double are only atomic if they declared with the volatile keyword."

65. What are Generics?

Generics enable types (classes and interfaces) to be parameters when defining classes, interfacesand methods. Much like the more familiar formal parameters used in method declarations, type parameters provide a way for you to re-use the same code with different inputs. The difference is that the inputs to formal parameters are values, while the inputs to type parameters are types.

Generics was added in Java 5 to provide **compile-time type checking** and removing risk of ClassCastException that was common while working with collection classes.
We can define our own classes with generics type. We use angle brackets (<>) to specify the type parameter.

66. What are the restrictions in using generic type that is declared in a class declaration?

To use Java generics effectively, you must consider the following restrictions:

- Cannot Instantiate Generic Types with Primitive Types
- Cannot Create Instances of Type Parameters
- Cannot Declare Static Fields Whose Types are Type Parameters
- Cannot Use Casts or instance Of With Parameterized Types
- Cannot Create Arrays of Parameterized Types
- Cannot Create, Catch, or Throw Objects of Parameterized Types

67. How do you declare a generic class?

```
public class GenericsType<T>{
 private T t;
 public T get()
{ return this.t; }
 public void set(T t1)
{ this.t=t1; }

public static <T> boolean isEqual(GenericsType<T> g1, GenericsType<T> g2){ return
g1.get().equals(g2.get()); }

public static void main(String args[]){
      GenericsType<String> g1 = new GenericsType<>();
      g1.set("Pankaj");
      GenericsType<String> g2 = new GenericsType<>();
      g2.set("Pankaj");
      boolean isEqual = GenericsMethods.<String>isEqual(g1, g2);
      //above call can  be written as
      isEqual = GenericsMethods.isEqual(g1, g2} }
}}
```

68. Differentiate between process and thread?

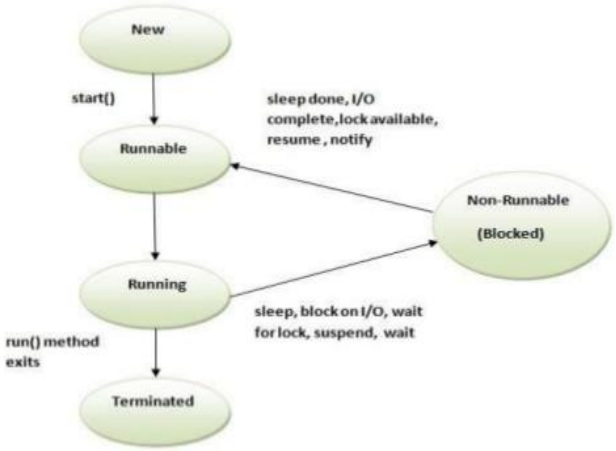
A Program in the execution is called the process whereas; A thread is a subset of the process Processes are independent whereas threads are the subset of process. Process have different address space in memory, while threads contain a shared address space. Context switching can be faster between the threads as compared to context switching between the threads. Inter-process communication is slower and expensive than inter-thread communication. Any change in Parent process doesn't affect the child process whereas changes in parent thread can affect the child thread.

69. What do you mean by Thread? Explain Thread Life Cycle?

When you want application to provide the most responsive interaction with the user, even if the application is currently doing other work we use **multithreading**. Using multiple threads of execution is one of the most powerful ways to keep your application responsive to the user and at the same time make use of the processor in between or even during user events.

LifeCycle of thread

- *New*: The thread is in new state if you create an instance of Thread class but before the invocation of start() method.
- *Runnable*: The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.
- *Running:* The thread is in running state if the thread scheduler has selected it.
- *Non-Runnable (Blocked) :* This is the state when the thread is still alive, but is currently not eligible to run.
- *Terminated:* A thread is in terminated or



dead state when its run() method exits.

70. What is need of Thread Synchronization?

Synchronization enables you to control program flow and access to shared data for concurrently executing threads. Synchronization can be achieved by mutex locks, read/write locks.

Any code written by using synchronized block or enclosed inside synchronized method will be mutually exclusive, and can only be executed by one thread at a time.

Synchronized keyword in Java is used to provide mutually exclusive access to a shared resource with multiple threads in Java. Important Point to be noted for synchronization.

- You can use java synchronized keyword only on synchronized method or synchronized block.
- Whenever a thread enters into java synchronized method or blocks it acquires a lock and whenever it leaves java synchronized method or block it releases the lock. The lock is released even if thread leaves synchronized method after completion or due to any Error or Exception.
- Java Thread acquires an object level lock when it enters into an instance synchronized java method and acquires a class level lock when it enters into static synchronized java method.
- Java Synchronization will throw NullPointerException if object used in java synchronized block is null
- One Major disadvantage of Java synchronized keyword is that it doesn't allow concurrent read, which can potentially limit scalability.
- One more limitation of java synchronized keyword is that it can only be used to control access to a shared object within the same JVM. If you have more than one JVM and need to synchronize access to a shared file system or database, the Java synchronized keyword is not at all sufficient. You need to implement a kind of global lock for that.
- Java synchronized keyword incurs a performance cost. A synchronized method in Java is very slow and can degrade performance. So use synchronization in java when it absolutely requires and consider using java synchronized block for synchronizing critical section only.
- Java synchronized block is better than java synchronized method in Java because by using synchronized block you can only lock critical section of code and avoid locking the whole method which can possibly degrade performance.

71. What is the difference between sleep() and wait()?

- sleep() – It causes the current thread to suspend execution for a specified period. When a thread goes into sleep state it doesn’t release the lock.
- wait() – It causes current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

72. What is a daemon thread?

A daemon thread is a thread, that does not prevent the JVM from exiting when the program finishes but the thread is still running. An example for a daemon thread is the garbage collection.

73. What is race-condition?

A Race condition is a problem which occurs in the multithreaded programming when various threads execute simultaneously accessing a shared resource at the same time. The proper use of synchronization can avoid the Race condition.

74. What is the volatile keyword in java?

Volatile keyword is used in multithreaded programming to achieve the thread safety, as a change in one volatile variable is visible to all other threads so one variable can be used by one thread at a time.

75. What is the use of join method in threads?

java.lang.Thread class provides the join() method which allows one thread to wait until another thread completes its execution. If t is a Thread object whose thread is currently executing, then t.join() will make sure that t is terminated before the next instruction is executed by the program.join() method puts the current thread on wait until the thread on which it is called is dead. If thread is interrupted then it will throw InterruptedException.

76. Can we call run() method of a Thread class?

Yes, we can call run() method of a Thread class but then it will behave like a normal method. To actually execute it in a Thread, you should call Thread.start() method to start it.

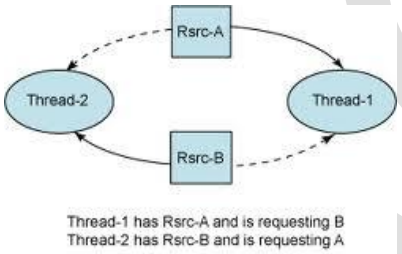
77. What is Starvation?

Starvation describes a situation where a thread is unable to gain regular access to shared resources and is unable to make progress. This happens when shared resources are made unavailable for long periods by "greedy" threads. For example, suppose an object provides a synchronized method that often takes a long time to return. If one thread invokes this method frequently, other threads that also need frequent synchronized access to the same object will often be blocked.

78. What is a deadlock?

In Java, a deadlock is a situation where minimum two threads are holding the lock on some different resource, and both are waiting for other's resource to complete its task. And, none is able to leave the lock on the resource it is holding.

In given figure Thread-1 has A but need B to complete processing and similarly Thread-2 has resource Bbut need A first.



79. Explain about wait(), notify() and notifyAll() methods?

The Object class in Java has three final methods that allow threads to communicate about the locked status of a resource.

- wait() : It tells the calling thread to give up the lock and go to sleep until some other thread enters the same monitor and calls notify(). The wait() method releases the lock prior to waiting and reacquires the lock prior to returning from the wait() method
- notify(): It wakes up one single thread that called wait() on the same object. It should be noted that calling notify() does not actually give up a lock on a resource. It tells a waiting thread that that thread can wake up. However, the lock is not actually given up until the notifier's synchronized block has completed.
- notifyAll(): It wakes up all the threads that called wait() on the same object. The highest priority thread will run first in most of the situation, though not guaranteed. Other things are same as notify()method above.

80. Why wait(), notify() and notifyAll() methods Are in Object Class And Not in Thread Class?

wait() and notify() methods work at the monitor level, thread which is currently holding the monitor is asked to give up that monitor through wait() method and through notify() method (or notifyAll)

threads which are waiting on the object's monitor are notified that threads can wake up. Monitor is assigned to an object not to a particular thread. That's one reason why these methods are in Object class.

wait(), notify() and notifyAll() are used for inter-thread communication. But threads themselves have no knowledge of each others status. It is the shared object among the threads that acts as a communicator among the threads.Threads lock an object, wait on an object and notify an object. When a wait method is called it checks which thread has the lock on the object and that is the thread which has to give up the lock. Same way notify() method when called looks for all the thread that are waiting to get hold of the Object's monitor and wakes one of the thread, notifyAll() wakes up all the thread that are waiting on an Object's monitor.

So it is the shared object among the thread which allows them to communicate with each other and wait(), notify() and notifyAll() are the methods used for inter-thread communication.

81. What is Reflection in Java and what are applications of Reflection?

Reflection is a process that provides the ability to inspect and modify the runtime behavior of applications. Using reflection we can inspect a class, interface, enums, get their structure, methods and fields information at runtime even though class is not accessible at compile time. We can also use reflection to instantiate an object, invoke it's methods, change field values.

The java.lang.Class class provides many methods that can be used to get metadata, examine and change the run time behavior of a class.

The java.lang and java.lang.reflect packages provide classes for java reflection.

The java.lang.Class class performs mainly two tasks:

- provides methods to get the metadata of a class at run time.
- provides methods to examine and change the run time behavior of a class.

Uses of Reflection:The Reflection API is mainly used in

- IDE (Integrated Development Environment) e.g. Eclipse, MyEclipse, NetBeans etc.
- Debugger
- Test Tools etc.

82. What is the basic Functionality of Reflection API in Java?

- Determination of the class of an object.
- Obtain the information about modifiers, fields, methods, constructors of a class
- Identifying the constants and method declarations of a specific interface.
- Creation of an instance of a class without knowing its name until runtime.
- Setting and getting the value of a field of an object, without knowing the field name until runtime.

83. Difference between Introspection and Reflection?

Introspection is the ability of a program to examine the type or properties of an object at runtime. Whereas, Reflection is the ability of a program to examine and modify the structure and behavior of an object at runtime.

Introspection is a subset of reflection. Some of the languages support introspection, but do not support reflection, e.g., C++.

Introspection Example: The instanceOf operator determines which class an object belongs to.

```
if(obj instanceOf Account){
  Account acc = (Account)obj;
  acc.getBalance();
}
```

Reflection Example:
The Class.forName() method returns the Class object associated with the class/interface with the given name(a string and full qualified name). The forName method causes the class with the name to be initialized.

```
Class c = Class.forName("classpath.classname");
Object acc = c.newInstance();
Method m = c.getDeclaredMethod("getBalance", new Class[0]);
m.invoke(acc);
```

84. What is Dynamic Class Loading using java Reflection Api ?

```
class Test {            //class to load dynamically
        --------
        --------
        }

public class ClassLoading {
        public static void main(String[] args) {
        ClassLoader cl = ClassLoading.class.getClassLoader();
        try {
        Class c = cl.loadClass("Test");
        System.out.println("c.getName() = " + c.getName());
        }
        catch (Exception e) {
        System.out.println("Exception: " + e.toString());
        }
}
```

85. What are the new features in Java 7?

Features of java 7

- Binary Literals
- Strings in switch Statement
- Try with Resources or ARM (Automatic Resource Management)
- Multiple Exception Handling
- Suppressed Exceptions
- underscore in literals
- Type Inference for Generic Instance Creation using Diamond Syntax

86. What are the new features in Java 8?

- *Lambda Expressions& Functional Interface*
  An interface with exactly one abstract method becomes Functional Interface. Lambda expressions gives the ability to pass a functionality as a method argument. Lambda expression help us reduce the code.

```
interface Printer{
   abstract void print();
}
public class DemoFunctionalInterface {
   public static void main(String[] args) {
      Printer printer = ()-> System.out.println("IACSD -CDAC!");
      printer.print();
   }
```

- *Pipelines and Streams*

Pipelines and streams enrich the Java collections framework. Sequence of aggregate operations is a pipeline. Stream is used to propagate elements from a source through a pipeline. It is a sequence of elements. Pipeline and streams will make our task easier in accessing the elements from collections and applying operations on it.

- *Date and Time API*
- *Default Methods*

Default methods gives the ability to add default implementation for methods in an interface.

- *Type Annotations*

Before Java 8 Java annotations can be applied to type declarations. From this Java 8 release onwards, annotations can be applied to type use. Annotations can be applied wherever a type is used like in new instance creates, exception throws clause etc.

- *Nashorn JavaScript Engine*

Nashorn is a brand new JavaScript engine provided along with the Java 8 release. Using this we can develop standalone JavaScript applications in Java. Pre Java 8, we got JDK with a JavaScript engine based on Rhino. It is a developed from scratch.

- *Concurrent Accumulators*

java.util.concurrent.atomic package is getting additional classes as part of Java 8 release. These will help to sum values from across multiple threads.

- *Parallel operations*

Iterating over a collection can be done in parallel using the aggregate operations easily. Pre Java 8 Iterators are used to parse the elements of a collection one by on explicitly. Now that can be done in parallel internally with the use of streams and aggregate operations.

- *PermGen Space Removed*

The PermGen space is removed from Java 8 and instead we have MetaSpace introduced. One of the most dreaded error, "java.lang.OutOfMemoryError: PermGen error" will no longer be seen from Java 8. The MetaSpace default is unlimited and that the system memory itself becomes the memory.

87. What is JDBC API? What are different types of JDBC Drivers?

Java Database Connectivity API allows us to work with relational databases. JDBC API interfaces and classes are part of java.sql and javax.sql package. We can use JDBC API to get the database connection, run SQL queries and stored procedures in the database server and process the results.

There are four types of JDBC drivers.

- *ODBC-ODBC Bridge plus ODBC Driver (Type 1):* It uses ODBC driver to connect to database. We should have ODBC drivers installed to connect to database, that's why this driver is almost obsolete.
- *Native API partly Java technology-enabled driver (Type 2):* This driver converts JDBC class to the client API for the database servers. We should have database client API installed. Because of extra dependency on database client API drivers, this is also not preferred driver.
- *Pure Java Driver for Database Middleware (Type 3):* This driver sends the JDBC calls to a middleware server that can connect to different type of databases. We should have a middleware server installed to work with this driver. This adds to extra network calls and slow performance and thats why not widely used JDBC driver.
- *Direct-to-Database Pure Java Driver (Type 4):* This driver converts the JDBC calls to the network protocol understood by the database server. This solution is simple and suitable for database

connectivity over the network. However for this solution, we should use database specific drivers, for example OJDBC jars by Oracle for Oracle DB and MySQL Connector/J for MySQL databases.

88. Explain Steps to connect to database using jdbc?

- Register the Driver class
- Create connection
- Create statement
- Execute queries
- Close connection

89. What is the use of JDBC DriverManager class?

JDBC DriverManager is the factory class through which we get the Database Connection object. When we load the JDBC Driver class, it registers itself to the DriverManage.Then when we call DriverManager.getConnection() method by passing the database configuration details, DriverManager uses the registered drivers to get the Connection and return it to the caller program.

90. What is JDBC Statement?

JDBC API Statement is used to execute SQL queries in the database. We can create the Statement object by calling Connection.createStatement() method. We can use Statement to execute SQL queries by passing query through different execute methods such as execute(), executeQuery(), executeUpdate() etc. Since the query is generated in the java program, if the user input is not properly validated it can lead to SQL injection issue. By default, only one ResultSet object per Statement object can be open at the same time.

91. What is the difference between execute, executeQuery, executeUpdate?

- execute( ) is used to execute any SQL query and it returns TRUE if the result is an ResultSet such as running Select queries. The output is FALSE when there is no ResultSet object such as running Insert or Update queries. We can use getResultSet() to get the ResultSet and getUpdateCount() method to retrieve the update count.
- executeQuery() is used to execute Select queries and returns the ResultSet. ResultSet returned is never null even if there are no records matching the query. When executing select queries we should use executeQuery method so that if someone tries to execute insert/update statement it will throw java.sql.SQLException with message “executeQuery method can not be used for update”.
- executeUpdate() is used to execute Insert/Update/Delete (DML) statements or DDL statements that returns nothing. The output is int and equals to the row count for SQL Data Manipulation Language (DML) statements. For DDL statements, the output is 0.

You should use execute() method only when you are not sure about the type of statement else use executeQuery or executeUpdate method.

92. What is JDBC PreparedStatement?

JDBC PreparedStatement object represents a precompiled SQL statement. We can use it’s setter method to set the variables for the query.Since PreparedStatement is precompiled, it can then be used to efficiently execute this statement multiple times improving performance.

Benefits of using PreparedStatement

- PreparedStatement helps us in preventing SQL injection attacks because it automatically escapes the special characters.
- PreparedStatement allows us to execute dynamic queries with parameter inputs.

- PreparedStatement is faster than Statement. It becomes more visible when we reuse the PreparedStatement or use it's batch processing methods for executing multiple queries.

93. What is JDBC ResultSet?

JDBC ResultSet is like a table of data representing a database result set, which is usually generated by executing a statement that queries the database.

ResultSet object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The next() method moves the cursor to the next row. If there are no more rows, next() method returns false and it can be used in a while loop to iterate through the result set.

A default ResultSet object is not updatable and has a cursor that moves forward only. Thus, you can iterate through it only once and only from the first row to the last row.

94. What are different types of ResultSet?

There are different types of ResultSet objects that we can get based on the user input while creating the Statement.

There are three types of ResultSet object.

- **ResultSet.TYPE_FORWARD_ONLY**: This is the default type and cursor can only move forward in the result set.
- **ResultSet.TYPE_SCROLL_INSENSITIVE**: The cursor can move forward and backward, and the result set is not sensitive to changes made by others to the database after the result set was created.
- **ResultSet.TYPE_SCROLL_SENSITIVE**: The cursor can move forward and backward, and the result set is sensitive to changes made by others to the database after the result set was created.

Based on the concurrency there are two types of ResultSet object.

- **ResultSet.CONCUR_READ_ONLY**: The result set is read only, this is the default concurrency type.
- **ResultSet.CONCUR_UPDATABLE**: We can use ResultSet update method to update the rows data.

95. How to use JDBC API to call Stored Procedures?

Stored Procedures are group of SQL queries that are compiled in the database and can be executed from JDBC API. JDBC CallableStatement can be used to execute stored procedures in the database. The syntax to initialize CallableStatement is,

```
CallableStatement stmt = con.prepareCall("{call insertEmployee(?,?,?)}");
stmt.setInt(1, id);
stmt.setString(2, name);
stmt.setString(3, city);

//register the OUT parameter before calling the stored procedure
stmt.registerOutParameter(6, java.sql.Types.VARCHAR);
stmt.executeUpdate();
```

We need to register the OUT parameters before executing the CallableStatement.

96. Can we execute a program without main() method?

Yes, one of the way is static block but in previous version of JDK not in JDK 1.7. class A{ static{ System.out.println("static block"); }} Output: static block (if not JDK7) In JDK7 and above, output will be: Output:Error: Main method

97. Can we override java main() method?

No, because main is a static method. A static method is bound with class whereas instance method is bound with object. Static belongs to class area and instance belongs to heap area. If a subclass defines a static method with the same signature as a static method in the superclass, then the method in the subclass hides the one in the superclass. The version of the overridden instance method that gets invoked is the one in the subclass. The version of the hidden static method that gets invoked depends on whether it is invoked from the superclass or the subclass.

98. Why use Java vs. C ++?

Diffrences between Java and C++:

- Platform dependency – C++ is platform dependent while java is platform independent
- No goto support – Java doesn't support goto statement while C++ does.
- Multiple inheritance – C++ supports multiple inheritance while java does not.
- Multithreading – C++ does not have in-build thread support, on the other hand java supports multithreading
- Virtual keyword – C++ has virtual keyword, it determines if a member function of a class can be overridden in its child class. In java there is no concept of virtual keyword.

99. How to create a immutable Class-object in Java?

- An immutable class is one whose state cannot be changed once created. Here, state of object essentially means the values stored in instance variable in class whether they are primitive types or reference types.
- To make a class immutable, below steps needs to be followed:
- Don't provide "setter" methods or methods that modify fields or objects referred to by fields. Setter methods are meant to change the state of object and this is what we want to prevent here.
- Make all fields final and private. Fields declared private will not be accessible outside the class and making them final will ensure the even accidentally you cannot change them.
- Don't allow subclasses to override methods. The simplest way to do this is to declare the class as final. Final classes in java cannot be overridden.
- Always remember that your instance variables will be either mutable or immutable. Identify them and return new objects with copied content for all mutable objects (object references). Immutable variables (primitive types) can be returned safely without extra effort.

100. What are features of Java7?

- **Allow using String in Switch case**
- **Multiple Exception in one catch block**
- **Automatic resource management or ARM blocks**

  Automatic resource management (ARM blocks) also known as try with resource block
- **Diamond operator <> for type inference**

  Diamond operator <> is a new Java 7 feature which provides type inference while creating object of Generic classes.

101. What are features of Java8?

- **Lambda Expression**

In Java programming language, a Lambda expression (or function) is just an *anonymous function*, i.e., a function with no name and without being bounded to an identifier. They are written typically *as a parameter to some other function*.

The basic *syntax of a lambda expression* is:

either

(parameters) -> expression
or
(parameters) -> { statements; }
or
() -> expression

A typical lambda expression example will be like this:
(x, y) -> x + y //This function takes two parameters and return their sum.

- **Functional Interface**

Functional interfaces are also called *Single Abstract Method interfaces (SAM Interfaces)*. As name suggest, they **permit exactly one abstract method** inside them. Java 8 introduces an annotation i.e. @FunctionalInterface which can be used for compiler level errors when the interface you have annotated violates the contracts of Functional Interface.
A typical functional interface example:

```
@FunctionalInterface
public interface MyFunctionalInterface {
  public void myWork();
}
```

Please note that a functional interface is valid even if the @FunctionalInterface annotation would be omitted. It is only for informing the compiler to enforce single abstract method inside interface.
Also, since default methods are not abstract you're *free to add default methods* to your functional interface as many as you like.

- **Default Methods**

Java 8 allows you to add non-abstract methods in interfaces. These methods must be declared default methods. Default methods were introduces in java 8 to enable the functionality of lambda expression.
Default methods enable you to add new functionality to the interfaces of your libraries

```
public interface Drivable {
  default void drive(){
    System.out.println("drive called");
  }
}
```

Moveable interface defines a method drive () and provided a default implementation as well. If any class implements this interface then it need not to implement it's own version of drive () method. It can directly call instance. drive ().
e.g.

```
public class Car implements Drivable {
  public static void main(String[] args){
    Car hondaobj = new Car();
    hondaobj.drive();
  }
}
```

Output: drive called

If class willingly wants to customize the behavior of drive () method then it can provide it's own custom implementation and override the method.

- **Java 8 Streams**

Java 8 Streams API, which provides a mechanism for processing a set of data in various ways that can include filtering, transformation, or any other way that may be useful to an application.

Streams API in Java 8 supports a different type of iteration where you simply define the set of items to be processed, the operation(s) to be performed on each item, and where the output of those operations is to be stored.

- **Java 8 Date/Time API Changes**

Date class has even become obsolete. The new classes intended to replace Date class are LocalDate, LocalTime and LocalDateTime.
The LocalDate class represents a date. There is no representation of a time or time-zone.
The LocalTime class represents a time. There is no representation of a date or time-zone.
The LocalDateTime class represents a date-time. There is no representation of a time-zone.

1. What are Servlets?

   A servlet is a Java technology-based Web component, managed by a container called servlet container or servlet engine that generates dynamic content and interacts with web clients via a request - response paradigm.

2. Why is Servlet so popular?

   Because servlets are platform-independent Java classes that are compiled to platform-neutral byte code that can be loaded dynamically into and run by a Java technology-enabled Web server.

3. What is servlet container?

   The servlet container is a part of a Web server or application server that provides the network services over which requests and responses are sent, decodes MIME-based requests, and formats MIME-based responses. A servlet container also contains and manages servlets through their lifecycle.

4. What are the functions of Servlet container?

   The main functions of Servlet container are:

   - Lifecycle management : Managing the lifecycle events of a servlet like class loading, instantiation, initialization, service, and making servlet instances eligible for garbage collection.
   - Communication support : Handling the communication between servlet and Web server.
   - Multithreading support : Automatically creating a new thread for every servlet request and finishing it when the Servlet service() method is over.
   - Declarative security : Managing the security inside the XML deployment descriptor file.
   - JSP support : Converting JSPs to servlets and maintaining them.

5. What is a deployment descriptor?

   A deployment descriptor is an XML document with an .xml extension. It defines a component's deployment settings. It declares transaction attributes and security authorization for an enterprise bean. The information provided by a deployment descriptor is declarative and therefore it can be modified without changing the source code of a bean.The JavaEE server reads the deployment descriptor at run time and acts upon the component accordingly.

6. What are the differences between the ServletConfig interface and the ServletContext interface?

ServletConfig	ServletContext
The ServletConfig interface is implemented by the servlet container in order to pass configuration information to a servlet. The server passes an object that implements the ServletConfig interface to the servlet's init() method.	A ServletContext defines a set of methods that a servlet uses to communicate with its servlet container.
There is one ServletConfig parameter per servlet.	There is one ServletContext for the entire webapp and all the servlets in a webapp share it.
The param-value pairs for ServletConfig object are specified in the <init-param> within the <servlet> tags in the web.xml file	The param-value pairs for ServletContext object are specified in the <context-param> tags in the web.xml file.

7. What's the difference between forward() and sendRedirect() methods?

forward()	sendRedirect()
A forward is performed internally by the servlet.	A redirect is a two step process, where the web application instructs the browser to fetch a second URL, which differs from the original.
The browser is completely unaware that it has taken place, so its original URL remains intact.	The browser, in this case, is doing the work and knows that it's making a new request.
Any browser reload of the resulting page will simple repeat the original request, with the original URL	A browser reloads of the second URL ,will not repeat the original request, but will rather fetch the second URL.
Both resources must be part of the same context (Some containers make provisions for cross-context communication but this tends not to be very portable)	This method can be used to redirect users to resources that are not part of the current context, or even in the same domain.
Since both resources are part of same context, the original request context is retained	Because this involves a new request, the previous request scope objects, with all of its parameters and attributes are no longer available after a redirect. (Variables will need to be passed by via the session object).
Forward is marginally faster than redirect.	redirect is marginally slower than a forward, since it requires two browser requests, not one.

8. What is the difference between the include() and forward() methods?

include()	forward()
The RequestDispatcherinclude() method inserts the the contents of the specified resource directly in the flow of the servlet response, as if it were part of the calling servlet.	The RequestDispatcherforward() method is used to show a different resource in place of the servlet that was originally called.
If you include a servlet or JSP document, the included resource must not attempt to change the response status code or HTTP headers, any such request will be ignored.	The forwarded resource may be another servlet, JSP or static HTML document, but the response is issued under the same URL that was originally requested. In other words, it is not the same as a redirection.
The include() method is often used to include common "boilerplate" text or template markup that may be included by many servlets.	The forward() method is often used where a servlet is taking a controller role; processing some input and deciding the outcome by returning a particular response page.

9. What is the <load-on-startup> element?

The <load-on-startup> element of a deployment descriptor is used to load a servlet file when the server starts instead of waiting for the first request. It is also used to specify the order in which the files are to be loaded.The <load-on-startup> element is written in the deployment descriptor as follows:

```
<servlet>
<servlet-name>ServletName</servlet-name>
<servlet-class>ClassName</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
```

Note: The container loads the servlets in the order specified in the <load-on-startup> element.

10. What is JSTL ?

JSTL stands for JSP Standard Tag Library and JSTL represents a set predefined tags.The main purpose ofJSTL is to simplify the JSP development. JSTL is not language like JSP it's a tag library by which we canwrite java program in JSP, like for loop , if else statement etc.

Advantages of JSTL

1. Fast development.
2. Easy understandable for Humans.
3. No need to use scriptlet tag in JSP page.

11. How to use JSTL in JSP?

To use JSTL in our JSP pages, we need to download the JSTL jars and we need to add jar into WEBINF/lib directory.JSTL has some JSTL Core Tags and JSTL Functions. Before writing any JSTL Core tag or JSTL function into JSP page we need to add tag library into JSP page Header section.

JSTL Core Tags library

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

JSTL Functions Tag library

```
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
```

12. What is ORM?

Object-Relational Mapping (ORM) is a technique which provides us facility to query and manipulate data from a database using an object-oriented programming pradigm. ORM binds our tables or stored procedures in java classes, so that instead of writing SQL statements to interact with your database, you use methods and properties of objects.

13. Explain Advantages of Hibernate Framework

1.Hibernate is non-invasive framework means it does not force use to extend or implement API classes or interfaces.
2. Simple to learn and work means no need to learn complicated SQL queries.
3. Hibernate is a bit faster than JDBC queries.
4. By default transaction support is given.
5. Database independent query
6. Light-weight framework.

14. Limitations of Hibernate?

1. Hibernate is slower than pure JDBC driver API.
2. Hibernate is not recommended for small project.
3. Hibernate is not suitable for Batch processing
4. So Many configurations needed for simple query also.

15. Explain Hibernate Architecture ?

Elements of Hibernate Architecture

1. Configuration
2. SessionFactory
3. Session
4. Transaction
5. Query

16. What is Configuration?

Configuration is a class given by hibernate people to load the hibernate XML configuration file. Configuration cfg = new Configuration();

17. What is SessionFactory?

SessionFactory is an interface which is used to create the session object and SessionFactory contains the connection information, hibernate configuration information and mapping files,location path.Instances of SessionFactory are thread-safe and typically shared throughout an application.SessionFactory holds second level cache also. SessionFactorysessionFactory = configurationObject.buildSessionFactory();

18. What is Session?

Session is an interface which is used to execute SQL queries like insert , update , delete and it is holds first level cache.

19. What is Transaction?

Transaction is an interface using which we maintain transaction in hibernate application.

20. What is Query?

Query is an interface which is used to execute DML statement in hibernate application like select , delete etc.

21. What is Hibernate Mapping file?

Hibernate mapping file is used to configure java persistent class details like class name, fields name etc. In hibernate application our java class name will treated as table name for any database and fields name will be treated as column name of table. We can construct one or more hibernate mapping files in hibernate application. The naming convention of hibernate mapping file is javaClassName.hbm.xml.

Syntax

```
<hibernate-mapping>
        <class name="java _persistent_class" table="table_name">
        <!– Here id should be primary key of database –>
        <id name="field_name" column="database_column_name" type="Java_data_type" />
        <property name="field_name" column="database_column_name"type="Java_data_type"/>
        </class>
</hibernate-mapping>
```

22. What is Hibernate Configuration file?

Hibernate Configuration file is used to pass java JDBC driver name, driver class name, username and password and configuration file will be loaded file Configuration hibernate class in hibernate application. Note that we have to pass hibernate mapping file into Hibernate Configuration file.

Syntax

```
<hibernate-configuration>
```

```
<session-factory>
        <!– JDBC Connection info –>
        <property name="connection_driver_class">Driver Class Name</property>
        <property name="connection_url">URL </property>
        <property name="connection_user">user </property>
        <property name="connection_password">password</property>
        <!– Hibernate properties –>
        <property name="show_sql">true/false</property>
        <property name="Hibernate_Dialect">Hibernate Gievn dialect</property>
        <property name="hbm2ddl">create/update/auto</property>
        <!– Hibernate mapping file –>
        <mapping resource="mappingfilename.xml" / >
</session-factory>
</hibernate-configuration>
```

23. Explain Hibernate mapping?

To reduce data redundancy in our database table we will divide the properties of one class into two classes and then we will apply relationship between objects of two classes is called Hibernate mapping. In hibernate we can apply 4 types of relationship between two POJO classes.

1. One-to-Many
2. Many-to-One
3. Many-to-Many
4. One-to-One

In Hibernate if we want to apply relationship mapping between two classes then first we need to add Child class object to collection and then we need to set that collection object to Parent class.We can use List, Set or Map collection for mapping.

24. What is Hibernate Caching ?

Hibernate supports three level of cache to optimize the performance of hibernate application.

1. First level cache
2. Second level cache
3. Query level cache

25. Why Hibernate support Cache ?

There are so many problems we have to face if we will not use any cache concept in our application like.

1. Performance issue we will get widely.
2. Network bandwidth issue we will.

26. When to use Hibernate cache

If same data is available around the application and if data will change very rarely then we can implement
the cache concept in our application. But if data is changing frequently then we should not use cache.

27. What is Hibernate Caching – First Level Cache?

Hibernate supports First Level Cache in the form of Session object and it is configured by default we can not disable first level cache but we can remove some of the object from Session object using some predefined method.
The scope of Hibernate First level cache is of per session object. Once session is closed cached objects are

gone forever.

28. What is Hibernate second level cache?

Hibernate Second Level cache will be created and associated with Hibernate SessionFactory and It will be available to in all sessions.
We can use Second level cache in all sessions within the application.Once Hibernate SessionFactoryobjectwillbe closed then all cached values will be gone.
Working of Hibernate second level cache

1. Hibernate application will try to load Entity from Session cache first if found that entity then it won't go in Second level cache.
2. If Hibernate does not find any entity in session object cache then it will search in Second level cache , if that entity found in second level cache then it will process.
3. If that entity does not find then Hibernate query will hit database.

29. How to implement Second level cache in hibernate?

Hibernate does not support Second level cache by default so we need to use some vendor provided second level cache like EHCache is one of the most popular second level cache vendor using this we can implement.

30. What is Spring framework?

Spring is complete application development framework which is use to develop all type of application like standalone, distributed, EJB, Enterprise application.

31. What are Advantages of spring framework ?

1. Spring is light-weight.
2. Spring is versatile.
3. It is non-invasive framework.
4. It provides rapid application development.
5. It has given boiler plate code to reduce development effort of programmer.
6. It has strong dependency management.
7. It provides loosely coupling.

32. What is Non-Invasive Framework?

A framework that does not forces us to implement or extend any interface or class is called noninvasive framework.
Example Spring is non-invasive because it passes all that need to our application development at run time.

33. What is Invasive Framework?

A framework that forces us to implement or extend any interface or class is called invasive framework. e.g. Struts is invasive framework because it forces us to extend their own ActionServlet class to create normal application.

34. Why Spring framework came?

When we design our application on the basis of J2EE then It will take more time to develop project and it complex to develop our application and API enforces us to write boiler plate code.

35. What are drawbacks of J2EE?

J2EE API has lots of problems that is why people switched to spring framework.

1. It is not giving boiler plate code,It enforces us to write boiler plate code.
2. It is taking more time develop application.
3. It is taking more cost to develop the application.
4. It provides tightly coupling.

36. What is boilerplate code?

The code which is repeatedly used in our application development is called Boiler Plate Code.
For example in JDBC we have to write 4 or 5 line to save a single record in our database.
J2EE API enforces us to write boiler plate code.

37. Why Spring is Light-Weight?

Spring framework is Light-Weight because it has provided all the modules separately.
When we want to use only Core Module of Spring then we can use only core module, no need to integrate other module like J2EE or JDBC ..etc

38. Explain Different Modules of spring framework?

There are six modules in spring framework, But this spring framework will vary with version to version.
Spring Modules

1. Spring Core(IOC Container)
2. Spring MVC
3. Spring JDBC
4. AOP
5. ORM
6. JEE

39. What is dependency injection in spring?

Dependency Injection is a process which is use to inject dependent object into target object by container(IOC Container) itself.

40. What is Advantage of Dependency Injection?

1. Dependency injection makes the code loosely coupled so easy to maintain
2. It makes configuration and code separation.
3. makes the code easy to test

41. Who does manage dependency injection?

IOC Container is responsible to manage our object in spring framework.

Ways to inject dependency in spring:

In three ways we can inject dependency using application-context.xml configuration file in spring.

1. Setter injection
2. Constructor injection
3. Interface injection

42. What is IOC container in spring?

IOC Container is a principle which is use to manage and collaborate life cycle of object and it is responsible to instantiate and manage the objects.IOC Container is use to manage dependency between two class in spring framework.IOC Container is logical memory which is located on top of java JVM.

Types of IOC Container

1. BeanFactory
2. ApplicationContext

43. What is Dependency Injection?

The Dependency injection is when all dependent objects are automaticallybinded when an instance is initialized.

44. What is Dependency Lookup?

Dependency lookup is concept which is trying to find a dependency .For example when we create ApplicationContext object in spring application thenthis ApplicationContext object is trying to get all dependency object from Spring XML Configuration file.
Note :Dependency Lookup is slow and Dependency injection is fast.

45. What is bean in spring?

A bean is an object which is instantiated, assembled, and managed by a Spring IOC container. These beans are created with the spring XMLconfiguration file that you given to the IOC container at the time of creating BeanFactory or ApplicationContext.
Beans are use to configure database connection parameters, security etc in spring XML Configuration file. Beans are use to avoid hard coding. Beans are use to manage the dependency of related class objects in context of IOC Container.

46. How to create bean in spring?

Here id you have to pass in main class at the time of creating BeanFactory or ApplicationContext. And You have to pass fully qualified class path and name in main class.

47. What is bean inheritance in spring?

Inheritance is one of the most important features of OOPS of any language by which we can reuse the existing functionality.
Like any language spring also has given parent attribute to reuse existing bean feature.

48. What is bean scope in spring?

In spring framework all bean has a scope , means every bean has its own visibility.
When we declare a class as a bean then by default the bean will be created under singleton scope.
How many types of bean scope

1. Singleton
2. Prototype
3. Request
4. Session
5. Global Session

49. What is singleton scope?

When will try to create multiple object of same bean then spring IOC container will return same object. Singleton scope is default scope in bean. Singleton scope functionality is same as singleton class in java.

50. What is prototype scope?

When we will declare bean as prototype scope then every time new object will be return by Spring IOC container.

51. What is request scope?

When we will declare a bean scope as request then for every HTTPRequest a new bean instance will be injected.

52. What is session scope?

When we will declare a bean scope as session then for every new HttpSession new bean instance will be injected.

53. What is auto-wiring in spring?

Auto-wiring is feature given by spring framework which is use to manage the dependency automatically.

Spring IOC container is able to find dependency and manage dependency implicitly. By default AutoWiring is disable in spring framework. If we want to enable Auto-Wiring to manage dependency automatically then we have to use autowire attribute in spring XML configuration file.

54. How to enable auto-wire?

We can enable auto-wire spring IOC container in four ways.

1. byName
2. byType
3. Constructor
4. Autodetect

What is the Microsoft.NET?

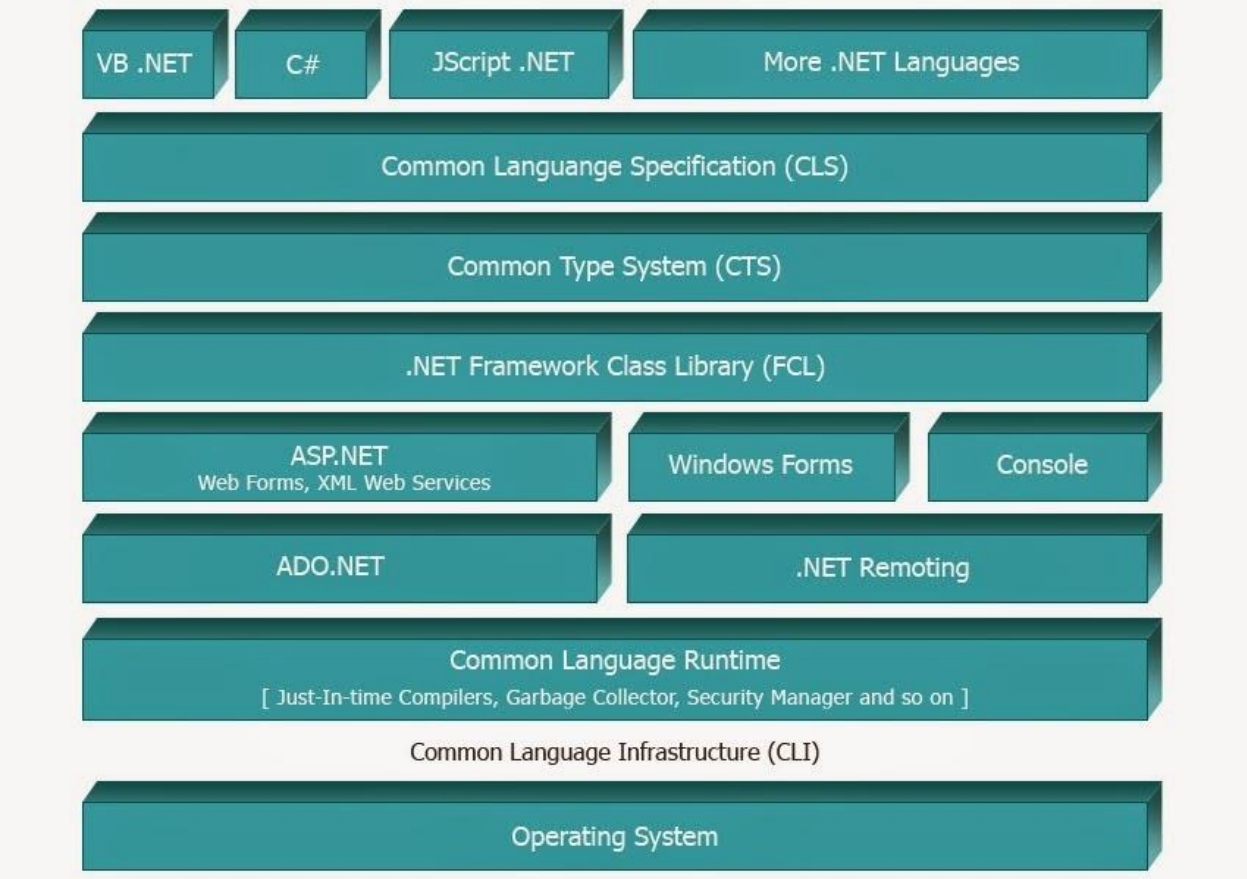
.NET is a set of technologies designed to transform the internet into a full scale distributed platform. It provides new ways of connecting systems, information and devices through a collection of web services. It also provides a language independent, consistent programming model across all tiers of an application.

The goal of the .NET platform is to simplify web development by providing all of the tools and technologies that one needs to build distributed web applications.

What is the .NET Framework?

The .NET Framework is set of technologies that form an integral part of the .NET Platform. It is Microsoft's managed code programming model for building applications that have visually stunning user experiences, seamless and secure communication, and the ability to model a range of business processes.

The .NET Framework has two main components: the common language runtime (CLR) and .NET Framework class library. The CLR is the foundation of the .NET framework and provides a common set of services for projects that act as building blocks to build up applications across all tiers. It simplifies development and provides a robust and simplified environment which provides common services to build and execute an application. The .NET framework class library is a collection of reusable types and exposes features of the runtime. It contains of a set of classes that is used to access common functionality.



What is CLR?

The .NET Framework provides a runtime environment called the Common Language Runtime or CLR. The CLR can be compared to the Java Virtual Machine or JVM in Java. CLR handles the execution of code and provides useful services for the implementation of the program. In addition to executing code, CLR provides services such as memory management, thread management, security management, code verification, compilation, and other system services. It enforces rules that in turn provide a robust and secure execution environment for .NET applications.

What is CTS?

Common Type System (CTS) describes the datatypes that can be used by managed code. CTS defines how these types are declared, used and managed in the runtime. It facilitates cross-language integration, type safety, and high performance code execution. The rules defined in CTS can be used to define your own classes and values.

What is CLS?

Common Language Specification (CLS) defines the rules and standards to which languages must adhere to in order to be compatible with other .NET languages. This enables C# developers to inherit from classes defined in VB.NET or other .NET compatible languages.

CLR and CTS advantages

Language interoperability
Code reusability
Faster development.

What is managed code?

The .NET Framework provides a run-time environment called the Common Language Runtime, which manages the execution of code and provides services that make the development process easier. Compilers and tools expose the runtime's functionality and enable you to write code that benefits from this managed execution environment. The code that runs within the common language runtime is called managed code.

What is MSIL?

When the code is compiled, the compiler translates your code into Microsoft intermediate language (MSIL). MSIL is understood by any environment where .Net framework is installed i.e. write once run anywhere. It is platform independent. MSIL is a part of assembly. The common language runtime includes a JIT compiler for converting this MSIL then to native code.

MSIL contains metadata that is the key to cross language interoperability. Since this metadata is standardized across all .NET languages, a program written in one language can understand the metadata and execute code, written in a different language. MSIL includes instructions for loading, storing, initializing, and calling methods on objects, as well as instructions for arithmetic and logical operations, control flow, direct memory access, exception handling, and other operations. It is also known as CIL(common Intermediate Language).

What is JIT?

JIT(Just in time) is a compiler that converts MSIL to native code JIT is a part of CLR. The native code consists of hardware specific instructions that can be executed by the CPU.

Rather than converting the entire MSIL (in a portable executable[PE]file) to native code, the JIT converts the MSIL as it is needed during execution. This converted native code is stored so that it is accessible for subsequent calls. This quality makes it faster compiler.

What is portable executable (PE)?

PE is the file format defining the structure that all executable files (EXE) and Dynamic Link Libraries (DLL) must use to allow them to be loaded and executed by Windows. PE is derived from the Microsoft Common Object File Format (COFF). The EXE and DLL files created using the .NET Framework obey the PE/COFF formats and also add additional header and data sections to the files that are only used by the CLR.

What is an application domain?

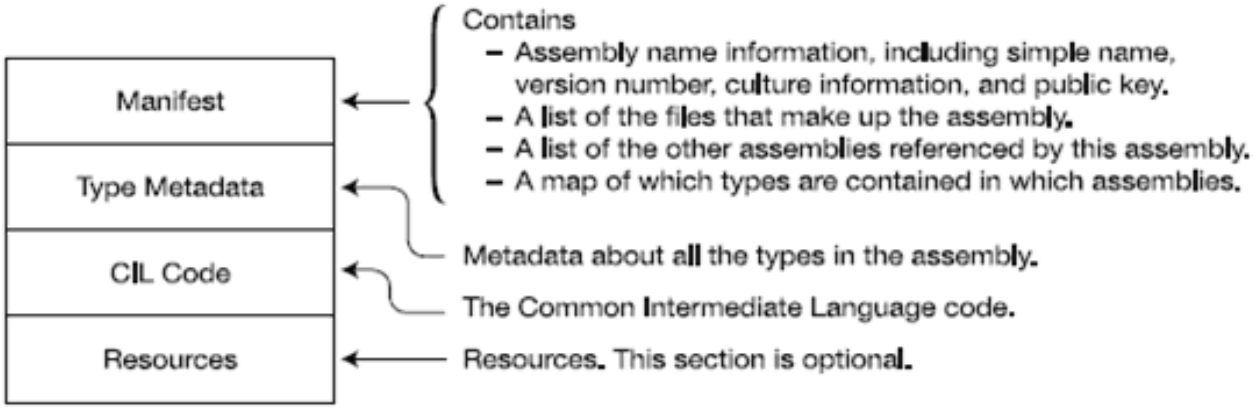
Application domain is the boundary within which an application runs. A process can contain multiple application domains. Application domains provide an isolated environment to applications that is similar to the isolation provided by processes. An application running inside one application domain cannot directly access the code running inside another application domain. To access the code running in another application domain, an application needs to use a proxy.

How does an AppDomain get created?

AppDomains are usually created by *hosts*. Examples of hosts are the Windows Shell, ASP.NET and IE. When you run a .NET application from the command-line, the host is the Shell. The Shell creates a new AppDomain for every application. AppDomains can also be explicitly created by .NET applications.

What is an assembly?

An assembly is a collection of one or more .exe or dll's. An assembly is the fundamental unit for application development and deployment in the .NET Framework. An assembly contains a collection of types and resources that are built to work together and form a logical unit of functionality. An assembly provides the CLR with the information it needs to be aware of type implementations.



What are the contents of assembly?

A static assembly can consist of four elements:

- Assembly manifest - Contains the assembly metadata. An assembly manifest contains the information about the identity and version of the assembly. It also contains the information required to resolve references to types and resources.
- Type metadata - Binary information that describes a program.
- Microsoft intermediate language (MSIL) code.
- A set of resources.

What are the different types of assembly?

Assemblies can also be private or shared. A private assembly is installed in the installation directory of an application and is accessible to that application only. On the other hand, a shared assembly is shared by multiple applications. A shared assembly has a strong name and is installed in the GAC.

We also have satellite assemblies that are often used to deploy language-specific resources for an application.

What is a dynamic assembly?

A dynamic assembly is created dynamically at run time when an application requires the types within these assemblies.

What is a strong name?

You need to assign a strong name to an assembly to place it in the GAC and make it globally accessible. A strong name consists of a name that consists of an assembly's identity (text name, version number, and culture information), a public key and a digital signature generated over the assembly. The .NET Framework provides a tool called the Strong Name Tool (Sn.exe), which allows verification and key pair and signature generation.

What is GAC? What are the steps to create an assembly and add it to the GAC?

The global assembly cache (GAC) is a machine-wide code cache that stores assemblies specifically designated to be shared by several applications on the computer. You should share assemblies by installing them into the global assembly cache only when you need to.

Steps

- Create a strong name using sn.exe tool eg: sn -k mykey.snk

- in AssemblyInfo.cs, add the strong name eg: [assembly: AssemblyKeyFile("mykey.snk")]

- recompile project, and then install it to GAC in two ways :

- drag & drop it to assembly folder (C:\WINDOWS\assembly OR C:\WINNT\assembly) (shfusion.dll tool)
- gacutil -i abc.dll

What is a garbage collector?

One of the very important services provided by CLR during application execution is automatic memory management(allocation and deallocation of memory). In this service, garbage collector plays an important role, it is a back ground thread that performs periodic checks on the managed heap to identify objects that are no longer required by the program and removes them from memory.When a program starts, the system allocates some memory for the program to get executed.

When a C# program instantiates a class, it creates an object.

The program manipulates the object, and at some point the object may no longer be needed. When the object is no longer accessible to the program and becomes a candidate for garbage collection.

There are two places in memory where the CLR stores items while your code executes i.e stack and Heap.The stack keeps track of what's executing in your code (like your local variables), and the heap keeps track of your objects.Value types can be stored on both the stack and the heap.

For an object on the heap, there is always a reference on the stack that points to it.

The garbage collector starts cleaning up only when there is not enough room on the heap to construct a new object.

The stack is automatically cleared at the end of a method. The CLR takes care of this and you don't have to worry about it.

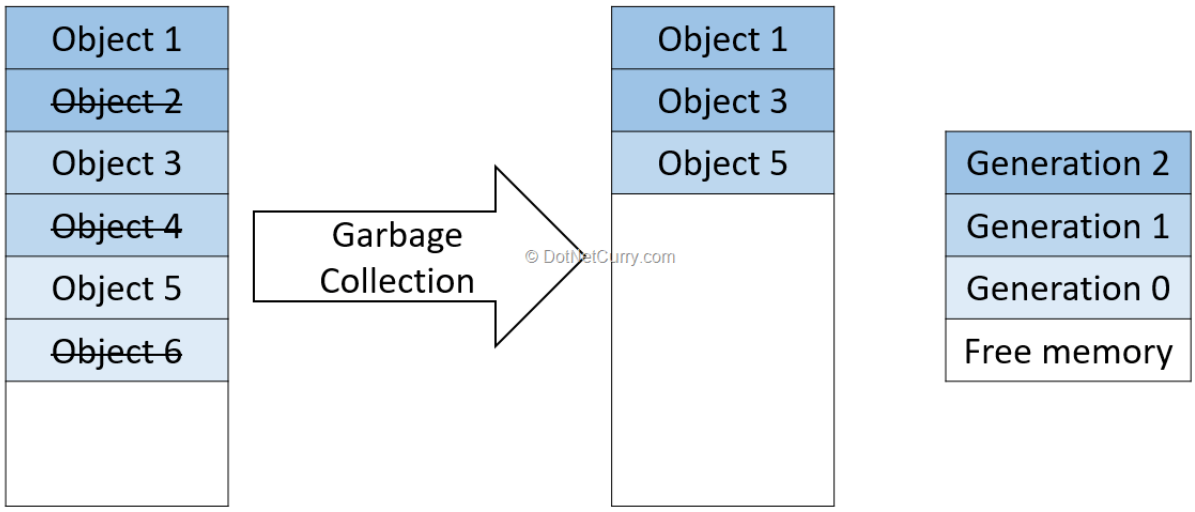
The heap is managed by the garbage collector.

In unmanaged environments without a garbage collector, you have to keep track of which objects were allocated on the heap and you need to free them explicitly. In the .NET Framework, this is done by the garbage collector.

All managed objects in the .NET framework are allocated on the managed heap. They are placed contiguously as they are created.

The garbage collection consists of three phases:

- In the **marking phase,** a list of all objects in use is created by following the references from all the root objects, i.e. variables on stack and static objects. Any allocated objects not on the list become candidates to be deleted from the heap. The objects on the list will be relocated in the compacting phase if necessary, to compact the heap and remove any unused memory between them.
- In the **relocation phase,** all references to remaining objects are updated to point to the new location of objects to which they will be relocated in the next phase.
- In the **compacting phase,** the heap finally gets compacted as all the objects that are not in use any more are released and the remaining objects are moved accordingly. The order of remaining objects in the heap stays intact.



The heap is organized into generations so it can handle long-lived and short-lived objects. Garbage collection primarily occurs with the reclamation of short-lived objects that typically occupy only a small part of the heap. There are three generations of objects on the heap:
Generation 0. This is the youngest generation and contains short-lived objects. An example of a short-lived object is a temporary variable. Garbage collection occurs most frequently in this generation.
Newly allocated objects form a new generation of objects and are implicitly generation 0 collections, unless they are large objects, in which case they go on the large object heap in a generation 2 collection.
Most objects are reclaimed for garbage collection in generation 0 and do not survive to the next generation.
Generation 1. This generation contains short-lived objects and serves as a buffer between short-lived objects and long-lived objects.
Generation 2. This generation contains long-lived objects. An example of a long-lived object is an object in a server application that contains static data that is live for the duration of the process.
Garbage collections occur on specific generations as conditions warrant. Collecting a generation means collecting objects in that generation and all its younger generations. A generation 2 garbage collection is also known as a full garbage collection, because it reclaims all objects in all generations (that is, all objects in the managed heap).
Generation 0 – Short-lived Objects
Generation 1- As a buffer between short lived and long lived objects
Generation 2 – Long lived objects

How value types get collected v/s reference types?

Value types are gets stored on the stack and therefore it gets removed from the stack by its pop method when application is done with its use. Reference types get stored on heap so gets collected by garbage collector.

Dispose v/s Finalize

Dispose	*Finalize*
It is used to free unmanaged resources at any time.	It can be used to free unmanaged resources held by an object before that object is destroyed.
It is called by user code and the class which is implementing dispose method, must has to implement IDisposable interface.	It is called by Garbage Collector and cannot be called by user code.
It is implemented by implementing IDisposable interface Dispose() method.	It is implemented with the help of Destructors
There is no performance costs associated with Dispose method.	There is performance costs associated with Finalize method since it doesn't clean the memory immediately and called by GC automatically.

How to Force Garbage Collection?

You can force this by adding a call to GC.Collect.

What is Finalizer?

inalizers (which are also called **destructors**) are used to perform any necessary final clean-up when a class instance is being collected by the garbage collector.

Remarks

- Finalizers cannot be defined in structs. They are only used with classes.
- A class can only have one finalizer.
- Finalizers cannot be inherited or overloaded.
- Finalizers cannot be called. They are invoked automatically.
- A finalizer does not take modifiers or have parameters.

For example, the following is a declaration of a finalizer for the Car class.

```
classCar
{
  ~Car()  // finalizer
  {
// cleanup statements...
```

```
    }
}
```

The finalizer implicitly calls Finalize on the base class of the object. Therefore, a call to a finalizer is implicitly translated to the following code:

```
protectedoverridevoidFinalize()
{
try
    {
// Cleanup statements...
    }
finally
    {
base.Finalize();
    }
}
```

This means that the Finalize method is called recursively for all instances in the inheritance chain, from the most-derived to the least-derived.

The programmer has no control over when the finalizer is called because this is determined by the garbage collector. The garbage collector checks for objects that are no longer being used by the application. If it considers an object eligible for finalization, it calls the finalizer (if any) and reclaims the memory used to store the object. In .NET Framework applications (but not in .NET Core applications), finalizers are also called when the program exits.

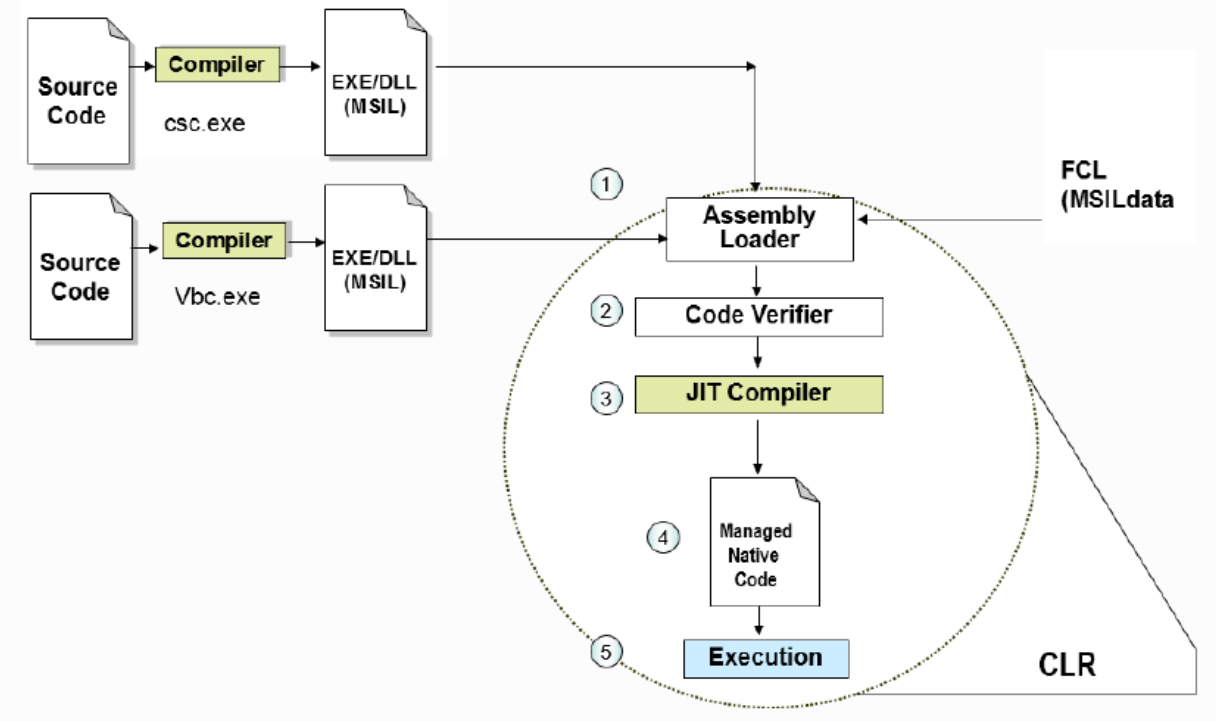
.NET Framework garbage collector implicitly manages the allocation and release of memory for your objects. However, when your application encapsulates unmanaged resources such as windows, files, and network connections, you should use finalizers to free those resources. When the object is eligible for finalization, the garbage collector runs the `Finalize` method of the object.

Explicit release of resources

If your application is using an expensive external resource, we also recommend that you provide a way to explicitly release the resource before the garbage collector frees the object. You do this by implementing a Dispose method from the IDisposable interface that performs the necessary cleanup for the object. This can considerably improve the performance of the application. Even with this explicit control over resources, the finalizer becomes a safeguard to clean up resources if the call to the Dispose method failed.

Explain the execution cycle of DotNet Application.

Execution Process in .NET Environment



1. Sorce code is compiled by language specific compilers.
2. After first compilation, assembly(exe/dll) is created, which is a logical unit of deployment.
3. Assembly is deployed on CLR for actual execution
4. Loader is responsible for loading and initializing assemblies, modules, resources, and types. When you run a .NET application on one of these systems that have an updated OS loader, the OS loader recognizes the .NET application and thus passes control to the CLR. The CLR then finds the entry point, which is typically Main( ), and executes it to jump-start the application. But before Main( ) can execute, the class loader must find the class that exposes Main( ) and load the class. In addition, when Main( ) instantiates an object of a specific class, the class loader also kicks in. In short, the class loader performs its magic the first time a type is referenced. The class loader loads .NET classes into memory and prepares them for execution. Before it can successfully do this, it must

locate the target class. the class loader uses the appropriate metadata to initialize the static variables and instantiate an object of the loaded class for you.

5. Code Verifier: The key here is type safety, and it is a fundamental concept for code verification in .NET. Within the VES, the verifier is the component that executes at runtime to verify that the code is type safe. Note that this type verification is done at runtime and that this is a fundamental difference between .NET and other environments. By verifying type safety at runtime, the CLR can prevent the execution of code that is not type safe and ensure that the code is used as intended. In short, type safety means more reliability.Let's talk about where the verifier fits within the CLR. After the class loader has loaded a class and before a piece of IL code can execute, the verifier kicks in for code that must be verified. The verifier is responsible for verifying that:

- The metadata is well formed, meaning the metadata must be valid.
- The IL code is type safe, meaning type signatures are used correctly.

  Both of these criteria must be met before the code can be executed because JIT compilation will take place only when code and metadata have been successfully verified. In addition to checking for type safety, the verifier also performs rudimentary control-flow analysis of the code to ensure that the code is using types correctly. You should note that since the verifier is a part of the JIT compilers, it kicks in only when a method is being invoked, not when a class or assembly is loaded. You should also note that verification is an optional step because trusted code will never be verified but will be immediately directed to the JIT compiler for compilation.

6. JIT Compiler: JIT compilers play a major role in the .NET platform because all .NET PE files contain IL and metadata, not native code. The JIT compilers convert IL to native code so that it can execute on the target operating system. For each method that has been successfully verified for type safety, a JIT compiler in the CLR will compile the method and convert it into native code.

WHAT IS DELEGATE?

A delegate is reference type that basically encapsulates the reference of methods. It is a similar to class that store the reference of methods which have same signature as delegate has.

It is very similar to a function pointer in C++, but delegate is type safe, object oriented and secure as compare to C++ function pointer.

WHY DO WE USE DELEGATE?

There are following reason because of we use delegate.

1. It is used for type safety.
2. For executing multiple methods through one execution.
3. It allows us to pass methods as parameter.
4. For asynchronous programming.
5. For Call back method implementation.
6. It is also used when working with event based programming.
7. When creating anonymous method.
8. When working with lambda expression.

HOW MANY TYPES OF DELEGATE IN C#?

There are three types of delegates available in C#.

1. Simple/Single Delegate
2. Multicast Delegate
3. Generic Delegate

WHAT ARE THE WAYS TO CREATE AND USE DELEGATE?

There are only five steps to create and use a delegate and these are as following.

1. Declare a delegate type.
2. Create or find a method which has same type of signature.
3. Create object/instance of delegate
4. Pass the reference of method with delegate instance.
5. At last Invoke the delegate object.

WHAT IS SINGLE DELEGATE?

When Delegate takes reference with single method only then it is called Single Delegate. It is used for invocation of only one reference method.

WHAT IS MULTICAST DELEGATE?

When Delegate takes reference with multiple methods then it is called multicast delegate. It is used for invocation of multiple reference methods. We can add or remove references of multiple methods with same instance of delegate using (+) or (-) sign respectively. It need to consider here that all methods will be invoked in one process and that will be in sequence.

WHAT IS MULTICAST DELEGATE

When Delegate takes reference with multiple methods then it is called multicast delegate. It is used for invocation of multiple reference methods. We can add or remove references of multiple methods with same instance of delegate using (+) or (-) sign respectively. It need to consider here that all methods will be invoked in one process and that will be in sequence.

Following example has a delegate name as "TestMultiCastDelegate" with one parameter as int.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;


namespaceEventAndDelegateExamples
{
delegatevoidTestMultiCastDelegate(int a);
classMultiCastDelegateExample
  {
internalvoidGetSalary(int AnnualSalary)
{
```

```
        Console.WriteLine("Monthly Salary is "+ AnnualSalary / 12);
    }

internalvoidGetSalaryAfterDeduction(int AnnualSalary)
{
if (AnnualSalary >12000)
        {
            Console.WriteLine("Monthly Salry after deduction is "+ ((AnnualSalary / 12) - 500));
        }
else
        {
            Console.WriteLine("Monthly Salry after deduction is " +AnnualSalary / 12);
        }
    }
  }
}
```

Below code snippet shows that first we are creating a instance of TestMultiCastDelegate and passing the reference of methods [GetSalary] and [GetSalaryAfterDeduction] with "objDelegate2".

```
MultiCastDelegateExample objMultiCastDelegateExample = new MultiCastDelegateExample();

//Creating the instance of the delegate
TestMultiCastDelegate objDelegate2 = null;
```

```
//Referencing the multiple methods using + sign
objDelegate2 += objMultiCastDelegateExample.GetSalary;
objDelegate2 += objMultiCastDelegateExample.GetSalaryAfterDeduction;
objDelegate2.Invoke(60000);
```

HOW TO ACHIEVE CALLBACK IN DELEGATE?

Callback is term where a process is going on and in between it targets some achievement then it return to main method. For callback, we just need to encapsulate the method with delegate.

```
objCallBackMethodExample.CheckEvenEvent += new OnEvenNumberHandler(objCallBackMethodExample.CallBackMethod);
```

Let see an example, CallBackMethodExample is a class which have a method CallBackMethod. It will be executed when some criteria will be fulfill.

```
publicdelegatevoidOnEvenNumberHandler(object sender, EventArgs e);

publicclassCallBackMethodExample
{
publicvoidCallBackMethod(object sender, EventArgs e)
{
        Console.WriteLine("Even Number has found !");
    }
}
```

When we are going to call this method using Delegate for callback, only need to pass this method name as a reference.

```
CallBackMethodExample objCallBackMethodExample = new CallBackMethodExample();

objCallBackMethodExample.CheckEvenEvent += new OnEvenNumberHandler(objCallBackMethodExample.CallBackMethod);
```

```
Random random = new Random();
for (int i = 0; i <6; i++)
{
var randomNumber = random.Next(1, 10);
  Console.WriteLine(randomNumber);
if (randomNumber % 2 == 0)
  {
     objCallBackMethodExample.OnCheckEvenNumber();
  }
}
```

HOW TO ACHIEVE ASYNC CALLBACK IN DELEGATE?

Async callback means, process will be going on in background and return back when it will be completed. In C#, Async callback can be achieved using AsyncCallback predefined delegate as following.

```
publicdelegatevoidAsyncCallback(IAsyncResult ar);
```

Kindly follow the following example to understand the Async callback. There is a class name as CallBackMethodExample which have two method, one is TestMethod and other one is AsyncCallBackFunction. But as we can see AsyncCallBackFunction is taking IAsyncResult as a parameter.

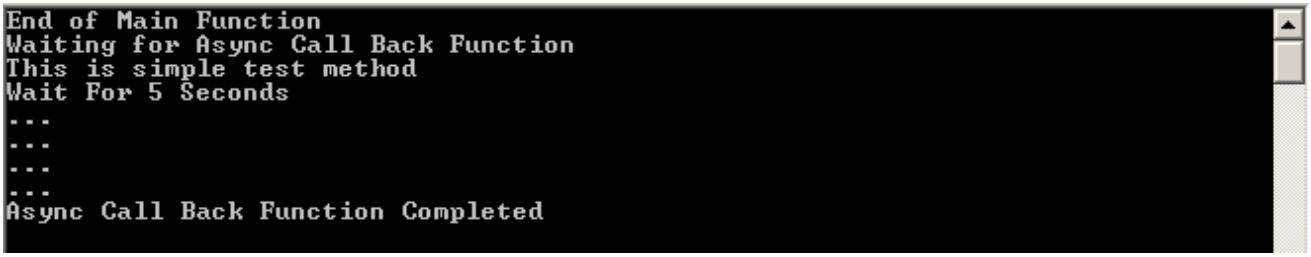
```
publicdelegatevoidOnEvenNumberHandler(object sender, EventArgs e);
```

```
publicclassCallBackMethodExample
  {
publicvoidTestMethod(object sender, EventArgs e)
{
        Console.WriteLine("This is simple test method");
    }

publicvoidAsynCallBackFunction(IAsyncResult asyncResult)
{
        OnEvenNumberHandler del = (OnEvenNumberHandler)asyncResult.AsyncState;
        del.EndInvoke(asyncResult);
        Console.WriteLine("Wait For 5 Seconds");
        Console.WriteLine("...");
        Console.WriteLine("...");
        Console.WriteLine("...");
        Console.WriteLine("...");
        System.Threading.Thread.Sleep(5000);
        Console.WriteLine("Async Call Back Function Completed");
    }
  }
```

When we are going to invoke the delegate, only just need to pass AsycnCallBackFunction with delegate instance.

```
CallBackMethodExample objCallBackMethodExample = new CallBackMethodExample();
OnEvenNumberHandler objDelegate3 = new OnEvenNumberHandler(objCallBackMethodExamp
le.TestMethod);
```

```
objDelegate3.BeginInvoke(null, EventArgs.Empty, new AsyncCallback(objCallBackMethodExample.AsynCallBackFunction), objDelegate3);

Console.WriteLine("End of Main Function");

Console.WriteLine("Waiting for Async Call Back Function");
```



What are events?

Events are higher level of encapsulation over delegates. Events use delegates internally. Delegates are naked and when passed to any other code, the client code can invoke the delegate. Event provides a publisher / subscriber mechanism model.

So subscribers subscribe to the event and publisher then push messages to all the subscribers. Below is a simple code snippet for the same:-

Create a delegate and declare the event for the same.

```
public delegate void CallEveryone();
public event CallEveryone MyEvent;
```

Raise the event.

```
MyEvent();
```

Attached client methods to the event are fired / notified.

```
obj.MyEvent += Function1;
```

Difference between delegate and events: -

They cannot be compared because one derives from the other.

- Actually, events use delegates in bottom. But they add an extra layer of security on the delegates, thus forming the publisher and subscriber model.
- As delegates are function to pointers, they can move across any clients. So any of the clients can add or remove events, which can be confusing. But events give the extra protection / encapsulation by adding the layer and making it a publisher and subscriber model.

What are attributes?

Attributes provide a powerful method of associating metadata, or declarative information, with code (assemblies, types, methods, properties, and so forth). After an attribute is associated with a program entity, the attribute can be queried at run time by using a technique called *reflection.*

Attributes have the following properties:

- Attributes add metadata to your program. *Metadata* is information about the types defined in a program. All .NET assemblies contain a specified set of metadata that describes the types and type members defined in the assembly. You can add custom attributes to specify any additional information that is required.
- You can apply one or more attributes to entire assemblies, modules, or smaller program elements such as classes and properties.
- Attributes can accept arguments in the same way as methods and properties.
- Your program can examine its own metadata or the metadata in other programs by using reflection.

  There are two types of attributes:

  1. Predefined attributes: these are provided by FCL. Ex. Obsolete, Serializable, Conditional etc.
  2. Custom attributes: these are developed by developers as per the requirements.

Note: All the attributes are derived directly or indirectly from System.Attribute class.

How to create custom attributes?

The primary steps to properly design custom attribute classes are as follows:

- Applying the AttributeUsageAttribute
- Declaring the attribute class
- Declaring constructors
- Declaring properties

Applying the AttributeUsageAttribute

A custom attribute declaration begins with the System.AttributeUsageAttribute, which defines some of the key characteristics of your attribute class. For example, you can specify whether your attribute can be inherited by other classes or specify which elements the attribute can be applied to. The following code fragment demonstrates how to use the AttributeUsageAttribute.

```
[AttributeUsage(AttributeTargets.All, Inherited = false, AllowMultiple = true)]
```

The AttributeUsageAttribute has three members that are important for the creation of custom attributes: AttributeTargets, Inherited, and AllowMultiple.

AttributeTargets Member

In the previous example, AttributeTargets.All is specified, indicating that this attribute can be applied to all program elements. Alternatively, you can specify AttributeTargets.Class, indicating that your attribute can be applied only to a class, or AttributeTargets.Method, indicating that your attribute can be applied only to a method. All program elements can be marked for description by a custom attribute in this manner.

You can also pass multiple AttributeTargets values. The following code fragment specifies that a custom attribute can be applied to any class or method.

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method)]
```

Inherited Property

The AttributeUsageAttribute.Inherited property indicates whether your attribute can be inherited by classes that are derived from the classes to which your attribute is applied. This property takes either a true (the default) or false flag. In the following example, MyAttribute has a default Inheritedvalue of true, while YourAttribute has an Inherited value of false.

```
// This defaults to Inherited = true.
publicclassMyAttribute : Attribute
{
//...
}
[AttributeUsage(AttributeTargets.Method, Inherited = false)]
publicclassYourAttribute : Attribute
{
//...
}
```

The two attributes are then applied to a method in the base class MyClass.

```
publicclassMyClass
```

```
{
   [MyAttribute]
   [YourAttribute]
publicvirtualvoidMyMethod()
   {
//...
   }}
```

Finally, the class YourClass is inherited from the base class MyClass. The method MyMethod shows MyAttribute, but not YourAttribute.

```
publicclassYourClass : MyClass
{
// MyMethod will have MyAttribute but not YourAttribute.
publicoverridevoidMyMethod()
   {
//...
   }

}
```

AllowMultiple Property

The AttributeUsageAttribute.AllowMultiple property indicates whether multiple instances of your attribute can exist on an element. If set to true, multiple instances are allowed; if set to false(the default), only one instance is allowed.

In the following example, MyAttribute has a default AllowMultiple value of false, while YourAttribute has a value of true.

```
publicclassMyAttribute : Attribute
{
}

[AttributeUsage(AttributeTargets.Method, AllowMultiple = true)]
publicclassYourAttribute : Attribute
{
}
```

When multiple instances of these attributes are applied, MyAttribute produces a compiler error. The following code example shows the valid use of YourAttribute and the invalid use of MyAttribute.

```
publicclassMyClass
```

```
{
// This produces an error.
// Duplicates are not allowed.
   [MyAttribute]
   [MyAttribute]
publicvoidMyMethod()
   {
//...
   }

// This is valid.
   [YourAttribute]
   [YourAttribute]
publicvoidYourMethod()
   {
//...
   }
}
```

If both the AllowMultiple property and the Inherited property are set to true, a class that is inherited from another class can inherit an attribute and have another instance of the same attribute applied in the same child class. If AllowMultiple is set to false, the values of any attributes in the parent class will be overwritten by new instances of the same attribute in the child class.

Declaring the Attribute Class

After you apply the AttributeUsageAttribute, you can begin to define the specifics of your attribute. The declaration of an attribute class looks similar to the declaration of a traditional class, as demonstrated by the following code.

```
[AttributeUsage(AttributeTargets.Method)]
publicclassMyAttribute : Attribute
{
// . . .
}
```

This attribute definition demonstrates the following points:

- Attribute classes must be declared as public classes.
- By convention, the name of the attribute class ends with the word **Attribute**. While not required, this convention is recommended for readability. When the attribute is applied, the inclusion of the word Attribute is optional.
- All attribute classes must inherit directly or indirectly from System.Attribute.

- In Microsoft Visual Basic, all custom attribute classes must have the System.AttributeUsageAttribute attribute.

Declaring Constructors

Attributes are initialized with constructors in the same way as traditional classes. The following code fragment illustrates a typical attribute constructor. This public constructor takes a parameter and sets a member variable equal to its value.

```
publicMyAttribute(bool myvalue)
{
this.myvalue = myvalue;
}
```

You can overload the constructor to accommodate different combinations of values. If you also define a property for your custom attribute class, you can use a combination of named and positional parameters when initializing the attribute. Typically, you define all required parameters as positional and all optional parameters as named. In this case, the attribute cannot be initialized without the required parameter. All other parameters are optional. Note that in Visual Basic, constructors for an attribute class should not use a ParamArray argument.

The following code example shows how an attribute that uses the previous constructor can be applied using optional and required parameters. It assumes that the attribute has one required Boolean value and one optional string property.

```
// One required (positional) and one optional (named) parameter are applied.
[MyAttribute(false, OptionalParameter = "optional data")]
publicclassSomeClass
{
//...
}
 [MyAttribute(false)]
publicclassSomeOtherClass
{
//...
}
```

Declaring Properties

If you want to define a named parameter or provide an easy way to return the values stored by your attribute, declare a property. Attribute properties should be declared as public entities with a description of the data type that will be returned. Define the variable that will hold the value of your property and associate it with the **get** and **set** methods. The following code example demonstrates how to implement a simple property in your attribute.

```
publicbool MyProperty
{
get {returnthis.myvalue;}
set {this.myvalue = value;}
}
```

Custom Attribute Example

This section incorporates the previous information and shows how to design a simple attribute that documents information about the author of a section of code. The attribute in this example stores the name and level of the programmer, and whether the code has been reviewed. It uses three private variables to store the actual values to save. Each variable is represented by a public property that gets and sets the values. Finally, the constructor is defined with two required parameters.

```
[AttributeUsage(AttributeTargets.All)]
publicclassDeveloperAttribute : Attribute
{
// Private fields.
privatestring name;
privatestring level;
privatebool reviewed;

// This constructor defines two required parameters: name and level.

publicDeveloperAttribute(string name, string level)
   {
this.name = name;
this.level = level;
this.reviewed = false;
   }

// Define Name property.
// This is a read-only attribute.


publicvirtualstring Name
   {
get {return name;}
   }
```

```
// Define Level property.
// This is a read-only attribute.

publicvirtualstring Level
   {
get {return level;}
   }

// Define Reviewed property.
// This is a read/write attribute.

publicvirtualbool Reviewed
   {
get {return reviewed;}
set {reviewed = value;}
   }
}
```

You can apply this attribute using the full name, DeveloperAttribute, or using the abbreviated name, Developer, in one of the following ways.

```
[Developer("Joan Smith", "1")]
```

-or-

```
[Developer("Joan Smith", "1", Reviewed = true)]
```

The first example shows the attribute applied with only the required named parameters, while the second example shows the attribute applied with both the required and optional parameters.

What is reflection?

Reflection provides objects (of type Type) that describe assemblies, modules and types. You can use reflection to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties. If you are using attributes in your code, reflection enables you to access them.

Uses for Reflection C#

There are several uses including:

1. Use Module to get all global and non-global methods defined in the module.

2. Use MethodInfo to look at information such as parameters, name, return type, access modifiers and implementation details.
3. Use EventInfo to find out the event-handler data type, the name, declaring type and custom attributes.
4. Use ConstructorInfo to get data on the parameters, access modifiers, and implementation details of a constructor.
5. Use Assembly to load modules listed in the assembly manifest.
6. Use PropertyInfo to get the declaring type, reflected type, data type, name and writable status of a property or to get and set property values.
7. Use CustomAttributeData to find out information on custom attributes or to review attributes without having to create more instances.

Other uses for Reflection include constructing symbol tables, to determine which fields to persist and through serialization.

Reflection Examples

This example shows how to dynamically load assembly, how to create object instance, how to invoke method or how to get and set property value.

Create instance from assembly that is in your project References

The following examples create instances of DateTime class from the System assembly:

```
// create instance of class DateTime
DateTime dateTime = (DateTime)Activator.CreateInstance(typeof(DateTime));

// create instance of DateTime, use constructor with parameters (year, month, day)
DateTime dateTime = (DateTime)Activator.CreateInstance(typeof(DateTime),
new object[] { 2008, 7, 4 });
```

Create instance from dynamically loaded assembly

All the following examples try to access to **sample class Calculator** from Test.dll assembly. The calculator class can be defined like this.

```
namespace Test
{
public classCalculator
   {
```

```
public Calculator() { ... }
private double _number;
public double Number { get { ... } set { ... } }
public void Clear() { ... }
private void DoClear() { ... }
public double Add(double number) { ... }
public static double Pi { ... }
public static double GetPi() { ... }
   }
}
```

Examples of using reflection to load the Test.dll assembly, to create instance of the Calculator class and to access its members (public/private, instance/static).

```
// dynamically load assembly from file Test.dll
Assembly testAssembly = Assembly.LoadFile(@"c:\Test.dll");

// get type of class Calculator from just loaded assembly
Type calcType = testAssembly.GetType("Test.Calculator");

// create instance of class Calculator
object calcInstance = Activator.CreateInstance(calcType);

// get info about property: public double Number
PropertyInfo numberPropertyInfo = calcType.GetProperty("Number");

// get value of property: public double Number
double value = (double)numberPropertyInfo.GetValue(calcInstance, null);

// set value of property: public double Number
numberPropertyInfo.SetValue(calcInstance, 10.0, null);

// get info about static property: public static double Pi
PropertyInfo piPropertyInfo = calcType.GetProperty("Pi");

// get value of static property: public static double Pi
double piValue = (double)piPropertyInfo.GetValue(null, null);

// invoke public instance method: public void Clear()
calcType.InvokeMember("Clear",
BindingFlags.InvokeMethod | BindingFlags.Instance | BindingFlags.Public,
null, calcInstance, null);
```

```
// invoke private instance method: private void DoClear()
calcType.InvokeMember("DoClear",
BindingFlags.InvokeMethod | BindingFlags.Instance | BindingFlags.NonPublic,
null, calcInstance, null);

// invoke public instance method: public double Add(double number)
double value = (double)calcType.InvokeMember("Add",
BindingFlags.InvokeMethod | BindingFlags.Instance | BindingFlags.Public,
null, calcInstance, new object[] { 20.0 });

// invoke public static method: public static double GetPi()
double piValue = (double)calcType.InvokeMember("GetPi",
BindingFlags.InvokeMethod | BindingFlags.Static | BindingFlags.Public,
null, null, null);
// get value of private field: private double _number
double value = (double)calcType.InvokeMember("_number",
BindingFlags.GetField | BindingFlags.Instance | BindingFlags.NonPublic,
null, calcInstance, null);
```

File Handling in C#

All the objects created at runtime reside on heap section primary memory. Once they are out of scope the objects are released by garbage collector. So the data stored in an object cannot be reused once the program is terminated. Therefore we store the data on secondary storage devices in the form of file. File handling is an unmanaged resource in your application system. It is outside your application domain (unmanaged resource). It is not managed by CLR.

What is a stream?

A stream is a sequence of bytes. In the file system, streams contain the data that is written to a file, and that gives more information about a file than attributes and properties. When you open a file for reading or writing, it becomes stream. Stream is a sequence of bytes traveling from a source to a destination over a communication path. The two basic streams are input and output streams. Input stream is used to read and output stream is used to write. The System.IO namespace includes various classes for file handling.

Stream is the abstract base class of all streams. A stream is an abstraction of a sequence of bytes, such as a file, an input/output device, an inter-process communication pipe, or a TCP/IP socket. The Stream class and its derived classes provide a generic view of these different types of input and output, and isolate the programmer from the specific details of the operating system and the underlying devices.

Streams involve three fundamental operations:

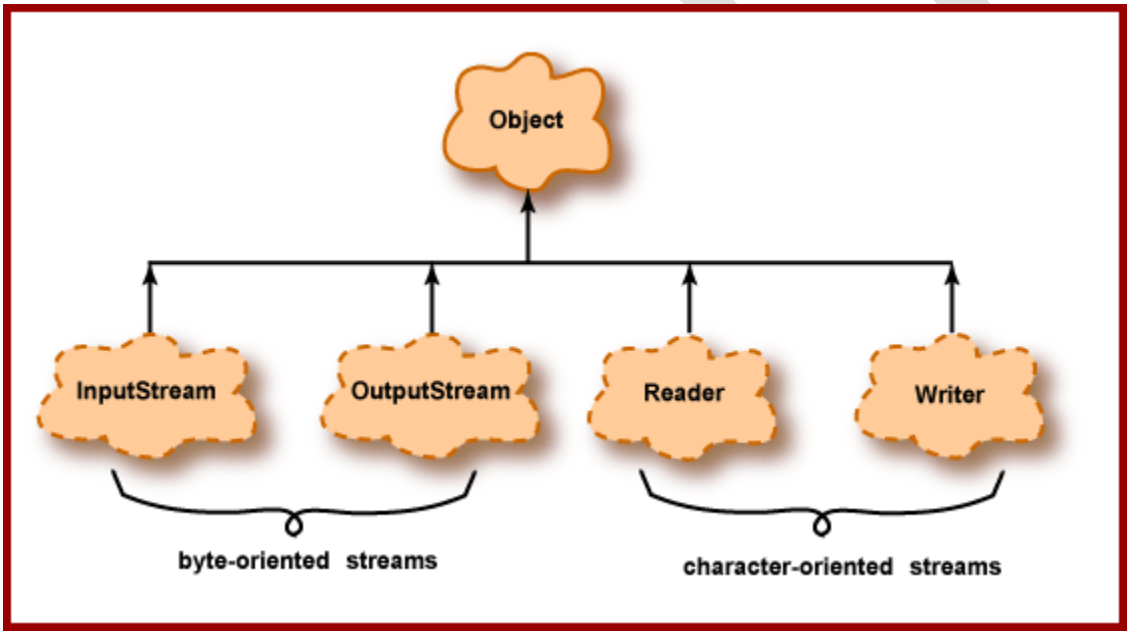
- You can read from streams. Reading is the transfer of data from a stream into a data structure, such as an array of bytes.
- You can write to streams. Writing is the transfer of data from a data structure into a stream.
- Streams can support seeking. Seeking refers to querying and modifying the current position within a stream. Seek capability depends on the kind of backing store a stream has. For example, network streams have no unified concept of a current position, and therefore typically do not support seeking.

Some of the more commonly used streams that inherit from Stream are FileStream, and MemoryStream.

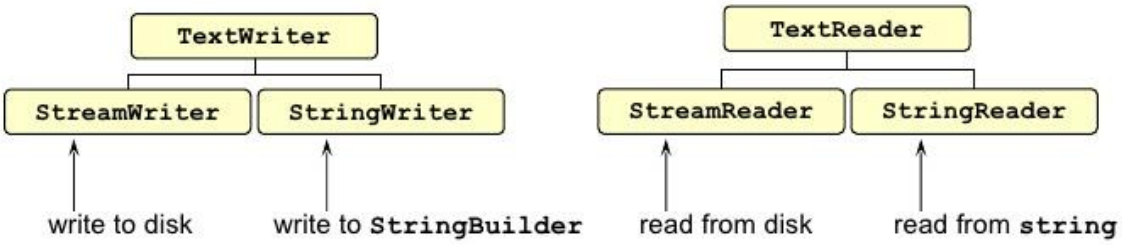
What is FileStream class?

Use the FileStream class to read from, write to, open, and close files on a file system, and to manipulate other file-related operating system handles, including pipes, standard input, and standard output. You can use the Read, Write, CopyTo, and Flush methods to perform synchronous operations, or the ReadAsync, WriteAsync, CopyToAsync, and FlushAsync methods to perform asynchronous operations.

Why there is a need of reader and writer classes?



As shown in the picture above, FileStream is byte oriented. But if we want to read and write characters then we need character reader and writer streams .



TextReader is the abstract base class of StreamReader and StringReader, which read characters from streams and strings, respectively. Use these derived classes to open a text file for reading a specified range of characters, or to create a reader based on an existing stream.

TextWriter is the abstract base class of StreamWriter and StringWriter, which write characters to streams and strings, respectively. Create an instance of TextWriter to write an object to a string, write strings to a file, or to serialize XML.

The BinaryReader class is used to read binary data from a file. A BinaryReader object is created by passing a FileStream object to its constructor.

The BinaryWriter class is used to write binary data to a stream. A BinaryWriter object is created by passing a FileStream object to its constructor.

Program of reading a file using StreamReader class:

```
using System;
using System.Collections.Generic;
using System.IO;


class Program
{
   static void Main()
   {
      List<string> list = new List<string>();
      using (StreamReader reader = new StreamReader("file.txt"))
      {
         string line;
         while ((line = reader.ReadLine()) != null)
         {
            list.Add(line); // Add to list.
            Console.WriteLine(line); // Write to console.
```

```
        }

    }

  }

}
```

Program for reading and writing binary data using BinaryReader and BinaryWriter:

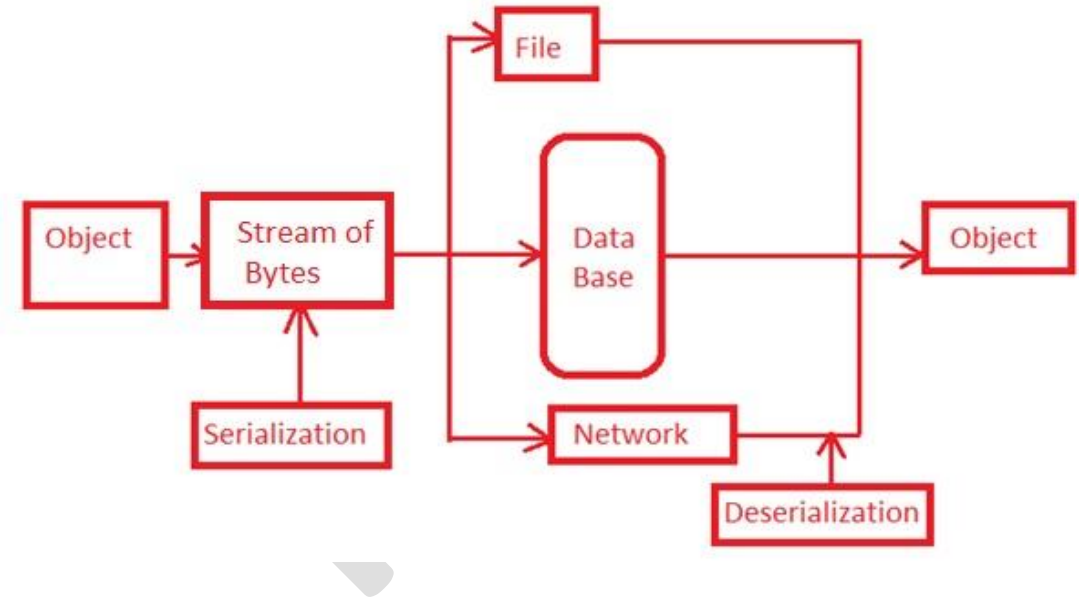
```
class Program
  {
    static void Main(string[] args)
    {

      // Create the new, empty data file.
      string fileName = @"C:\Temp.data";
      if (File.Exists(fileName))
      {
        Console.WriteLine(fileName + " already exists!");
        return;
      }
      FileStream fs = new FileStream(fileName, FileMode.CreateNew);
      // Create the writer for data.
      BinaryWriter w = new BinaryWriter(fs);
      // Write data to Test.data.
      for (int i = 0; i < 11; i++)
      {
        w.Write((int)i);
      }
      w.Close();
      fs.Close();
      // Create the reader for data.
      fs = new FileStream(fileName, FileMode.Open, FileAccess.Read);
      BinaryReader r = new BinaryReader(fs);
      // Read data from Test.data.
      for (int i = 0; i < 11; i++)
      {
        Console.WriteLine(r.ReadInt32());
      }
      r.Close();
      fs.Close();
    }
  }
```

What is Serialization?

Serialization is converting object to stream of bytes. It is a process of bringing an object into a form that it can be written on stream. It's the process of converting the object into a form so that it can be stored on a file, database or memory. It is transferred across the network. Main purpose is to save the state of the object.

What is deserialization?

Deserialization is converting stream of byte to object. Deserialization is the reverse process of serialization. It is the process of getting back the serialization object (so that) it can be loaded into memory. It lives the state of the object by setting properties, fields etc.



What are different types of Serialization techniques?

Different Types of Serialization

The Microsoft .NET Framework provides an almost bewildering variety of ways to serialize an object. This chapter focuses on XML serialization, but before we get into the details, I'd like to briefly examine and compare the various serialization methods offered.

XML Serialization

XML serialization allows the public properties and fields of an object to be reduced to an XML document that describes the publicly visible state of the object. This method serializes only public properties and fields—private data will not be persisted, so XML serialization does not provide full fidelity with the original object in all cases. However, because the persistence format is XML, the data being saved can be read and manipulated in a variety of ways and on multiple platforms.

The benefits of XML serialization include the following:

Allows for complete and flexible control over the format and schema of the XML produced by serialization.

Serialized format is both human-readable and machine-readable.

Easy to implement. Does not require any custom serialization-related code in the object to be serialized.

The XML Schema Definition tool (xsd.exe) can generate an XSD Schema from a set of serializable classes, and generate a set of serializable classes from an XSD Schema, making it easy to programmatically consume and manipulate nearly any XML data in an object-oriented (rather than XML-oriented) fashion.

Objects to be serialized do not need to be explicitly configured for serialization, either by the SerializableAttribute or by implementing the ISerializable interface.

The restrictions of XML serialization include the following:

The class to be serialized must have a default (parameterless) public constructor.

Read-only properties are not persisted.

Only public properties and fields can be serialized.

SOAP Serialization

SOAP serialization is similar to XML serialization in that the objects being serialized are persisted as XML. The similarity, however, ends there. The classes used for SOAP serialization reside in theSystem.Runtime.Serialization namespace rather than

the System.Xml.Serializationnamespace used by XML serialization. The run-time serialization classes (which include both the SoapFormatter and the BinaryFormatter classes) use a completely different mechanism for serialization than the XmlSerializer class.

The benefits of SOAP serialization include the following:

Produces a fully SOAP-compliant envelope that can be processed by any system or service that understands SOAP.Supports either objects that implement the ISerializable interface to control their own serialization, or objects that are marked with the SerializableAttribute attribute.

Can deserialize a SOAP envelope into a compatible set of objects.

Can serialize and restore non-public and public members of an object.

The restrictions of SOAP serialization include the following:

The class to be serialized must either be marked with the SerializableAttributeattribute, or must implement the ISerializable interface and control its own serialization and deserialization.

Only understands SOAP. It cannot work with arbitrary XML schemas.

Binary Serialization

Binary serialization allows the serialization of an object into a binary stream, and restoration from a binary stream into an object. This method can be faster than XML serialization, and the binary representation is usually much more compact than an XML representation. However, this performance comes at the cost of cross-platform compatibility and human readability.

The benefits of binary serialization include the following:

It's the fastest serialization method because it does not have the overhead of generating an XML document during the serialization process.

The resulting binary data is more compact than an XML string, so it takes up less storage space and can be transmitted quickly.

Supports either objects that implement the ISerializable interface to control its own serialization, or objects that are marked with the SerializableAttribute attribute.

Can serialize and restore non-public and public members of an object.

The restrictions of binary serialization include the following:

The class to be serialized must either be marked with the SerializableAttributeattribute, or must implement the ISerializable interface and control its own serialization and deserialization.

The binary format produced is specific to the .NET Framework and it cannot be easily used from other systems or platforms.

The binary format is not human-readable, which makes it more difficult to work with if the original program that produced the data is not available.

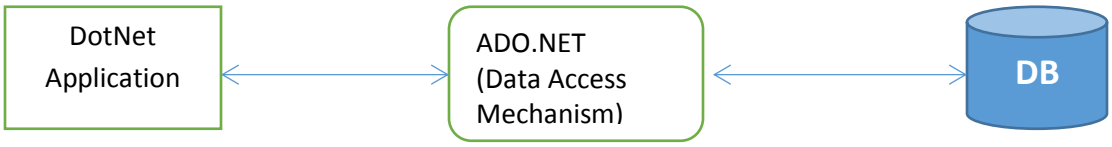
Program for binary serialization:

```
using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

namespace ConsoleApplication1
{
   class Program
   {
      public static void SerializeData()
      {
         string str = "hello world.";
         // Create file to save the data.
         FileStream fs = new FileStream(@"D:\MyDataFile.txt", FileMode.Create);
         // BinaryFormatter object will perform the serialization
         BinaryFormatter bf = new BinaryFormatter();
         // Serialize() method serializes the data to the file
         bf.Serialize(fs, str);
         // Close the file
         fs.Close();
      }
      public static void DeSerializeData()
      {
         // Open file to read the data
         FileStream fs = new FileStream(@"D:\MyDataFile.txt", FileMode.Open);
         // BinaryFormatter object performs the deserialization
         BinaryFormatter bf = new BinaryFormatter();
         // Create the object to store the deserialized data
         string data = "";
         data = (string)bf.Deserialize(fs);
         //// Close the file
         fs.Close();
```

```
        // Display the deserialized strings
        Console.WriteLine("Your deserialize data is ");
        Console.WriteLine(data);
      }
      static void Main(string[] args)
      {
        SerializeData();
        DeSerializeData();
        Console.ReadLine();
      }
   }
}
```

What does ActiveX Data Object.NET (ADO.NET) mean?

ActiveX Data Object.NET (ADO.NET) is a software library in the .NET framework consisting of software components providing data access services.



ADO.NET is designed to enable developers to write managed code for access to data sources, which can be relational or non-relational (such as XML or application data). This feature of ADO.NET helps to create data-sharing, distributed applications. ADO.NET provides connected access to a database connection using the .NET-managed providers and disconnected access using datasets, which are applications using the database connection only during retrieval of data or for data update. Dataset is the component helping to store the persistent data in memory to provide disconnected access for using the database resource efficiently and with better scalability.

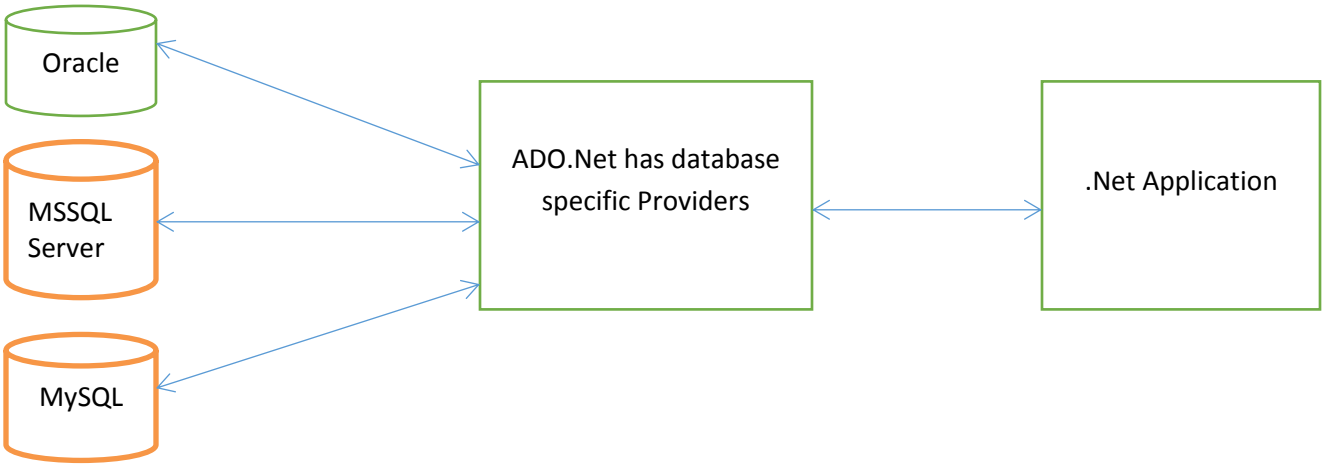
The architecture of ADO.NET is based on two primary elements:
.NET framework data provider and DataSet

A .NET data provider is a software library consisting of classes that provide data access services such as connecting to a data source, executing commands at a data source and fetching data from a data source with support to execute commands within transactions. It resides as a lightweight layer between data source and code, providing data access services with increased performance.

The .NET data provider is a component of ADO.NET, a subset of the .NET framework class library. The ADO.NET data access mode is designed such that the data set object can be used to represent an in-memory, relational structure with built-in XML support that can exist in a standalone, disconnected manner with its data, which can be passed through various layers of a multitier application. ADO.NET provides a set of interfaces to implement a custom .NET provider for specific data access needs, such as easier maintenance and better performance.

A .NET data provider makes it possible to process data directly in the data source or data stored in data sets, allowing for manipulation by the user. Data from various sources can also be combined, or passed between tiers of the application.

A .NET data provider serves as a channel to retrieve and update data existing in the data store irrespective of data sources. ADO.Net already has given providers that are database specific.



A .NET data provider consists of the following core objects:

- The Connection object is used to connect to a specific data source
- The Command object executes a command against a data source
- DataReader reads data from the data source in read-only, forward-only mode
- DataAdapter populates a data set and resolves an update with the data source

A .NET data provider abstracts the database's interaction with the application and therefore simplifies application development. However, to achieve the best performance of an application together with capability and integrity, the right .NET data provider has to be selected based on factors like design, the data source of the application, application type (middle or single tier), etc.

DataSet

The huge mainstream of applications built today involves data manipulation in some way -- whether it be retrieval, storage, change, translation, verification, or transportation. For an application to be scalable and allow other apps to interact with it, the app will need a common mechanism to pass the data around. Ideally, the vehicle that transports the data should contain the base data, any related data, and metadata, and should be able to track changes to the data. Here's where the ADO.NET DataSet steps in. The ADO.NET DataSet is a data construct that can contain several relational rowsets, the relations that link those rowsets, and the metadata for each rowset. The DataSet also tracks which fields have changed, their new values and their original values, and can store custom information in its Extended Properties collection. The DataSet can be exported to XML or created from an XML document, thus enabling increased interoperability between applications.

When we look at the DataSet object model, we see that it is made up of three collections; Tables, Relations, and ExtendedProperties. These collections make up the relational data structure of the DataSet. The DataSet.Tables property is a DataTableCollection object, which contains zero or more DataTable objects. Each DataTable represents a table of data from the

data source. Each DataTable is made of a Columns collection and a Rows collection, which are zero or more DataColumns or DataRows.

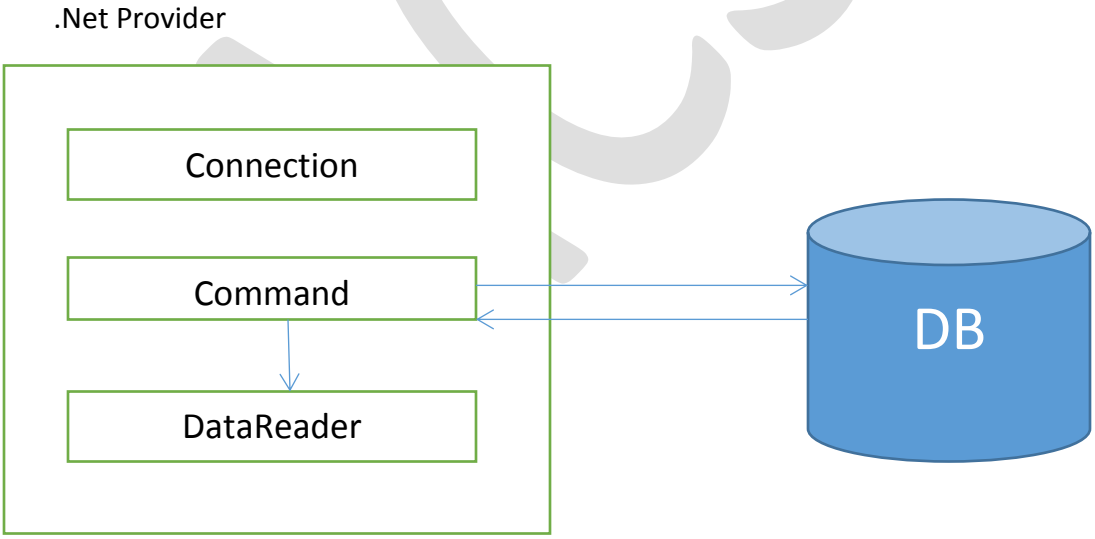
The Relations property as a DataRelationCollection object, which contains zero or more DataRelation objects. The DataRelation objects define a parent-child relationship between two tables based on foreign key values. On the other hand, The ExtendedProperties property is a PropertyCollection object, which contains zero or more user-defined properties. The ExtendedProperties collection can be used contains zero or more user-defined properties. This property collection can be used to store custom data related to the DataSet, such as the time when the DataSet was constructed.

One of the key points to remember about the DataSet is that it doesn't care where it originated. Unlike the ADO 2.x Recordset, the DataSet doesn't track which database or XML document its data came from. In this way, the DataSet is a standalone data store that can be passed from tier to tier in an n-tiered architecture.

There are two ways of Data access mechanisms.

1. Connected Architecture
2. Disconnected Architecture

Connected Architecture



Connection Oriented architecture is achieved by the use of Connection, Command and DataReader object.

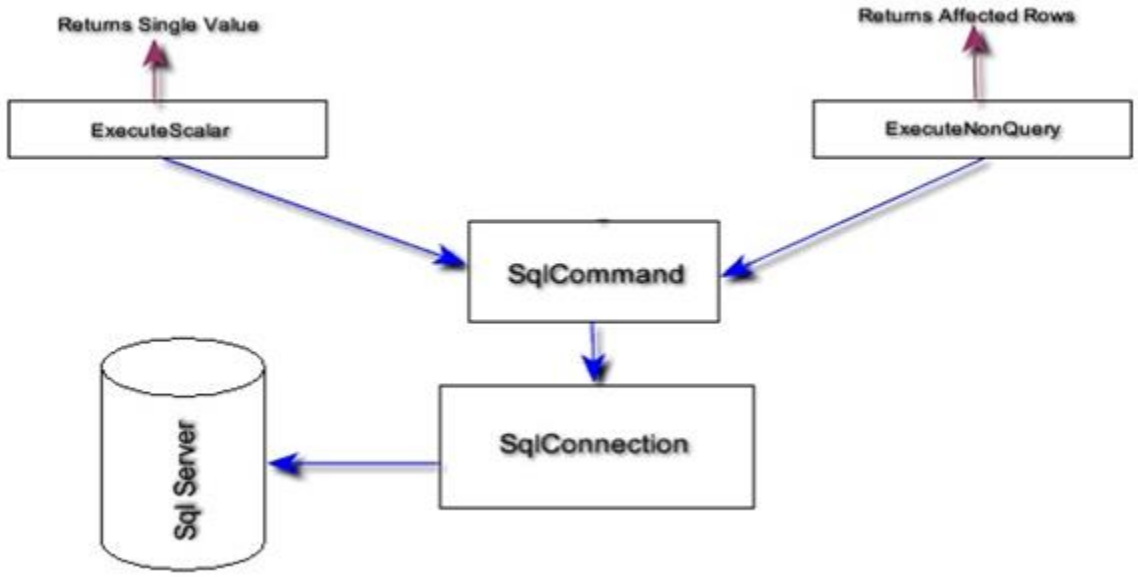
The Connection objects define the data provider, database manager instance, database, security credentials, and other connection-related properties.

Important methods of connection object
Open()-Opens a database connection with the property settings specified by the ConnectionString.
Close() - Closes the database connection.

The Command object works with the Connection object and used to execute SQL queries or Stored Procedures against the data source. You can perform insert, update, delete, and select operations with this object. It requires an instance of a Connection Object for executing the SQL statements as example.

Steps for executing SQL query with command object

- Create a Connection Object and initialize with connection string.
- Create the command object and initialize with SQL query and connection object.
- Open the connection.
- Execute query with the help of any one method of command object.
- Store the result in DataReader object (In case of select SQL query)
- Close the connection.



Important method of command object

ExecuteReader: This method works on select SQL query. It returns the DataReader object. Use DataReader read () method to retrieve the rows.

ExecuteScalar: This method returns single value. Its return type is Object. When you want single value (First column of first row), then use ExecuteScalar () method. Extra columns or rows are ignored. ExecuteScalar method gives better performance if SQL statements have aggregate function.

ExecuteNonQuery: If you are using Insert, Update or Delete SQL statement then use this method. Its return type is Integer (The number of affected records).

ExecuteXMLReader: It returns an instance of XmlReader class. This method is used to return the result set in the form of an XML document.

Important properties of Command class

- Connection
- CommandText
- CommandType
- CommandTimeout

CommandType is an important property of Command class. This property is used to determine which type of command being executed. It supports the following enumerators.

CommandType.StoredProcedure: It informs the Command object that the stored procedure will be used in place of simple SQL statements.

CommandType.Text: It informs the Command object that the CommandText value contains a SQL query.

CommandType.TableDirect: It indicates that the CommandText property contains the name of a table.

Using DataReader object
DataReader object works in connected mode. It is read only and forward only object. It is fast compare to DataSet. DataReader provides the easy way to access the data from database. It can increase the performance of application because it reads records one by one. Use read() method of DataReader to access the record.

For initializing the DataReader object, call the ExecuteReader method of the Command object. It returns an instance of the DataReader.

```
SqlCommand cmd = new SqlCommand("Your SQL query", conn);
SqlDataReader readerObj = cmd.ExecuteReader();
```

Once your task completed with the data reader, call Close() method to close the dataReader.

```
readerObj.Close();
```

Important properties of DataReader object

FieldCount: It provides the number of columns in the current row.

HasRows: It provides information that, whether data reader contains row or not.

IsClosed: It indicates that whether data reader is closed or not.

RecordsAffected: Returns the number of affected records.

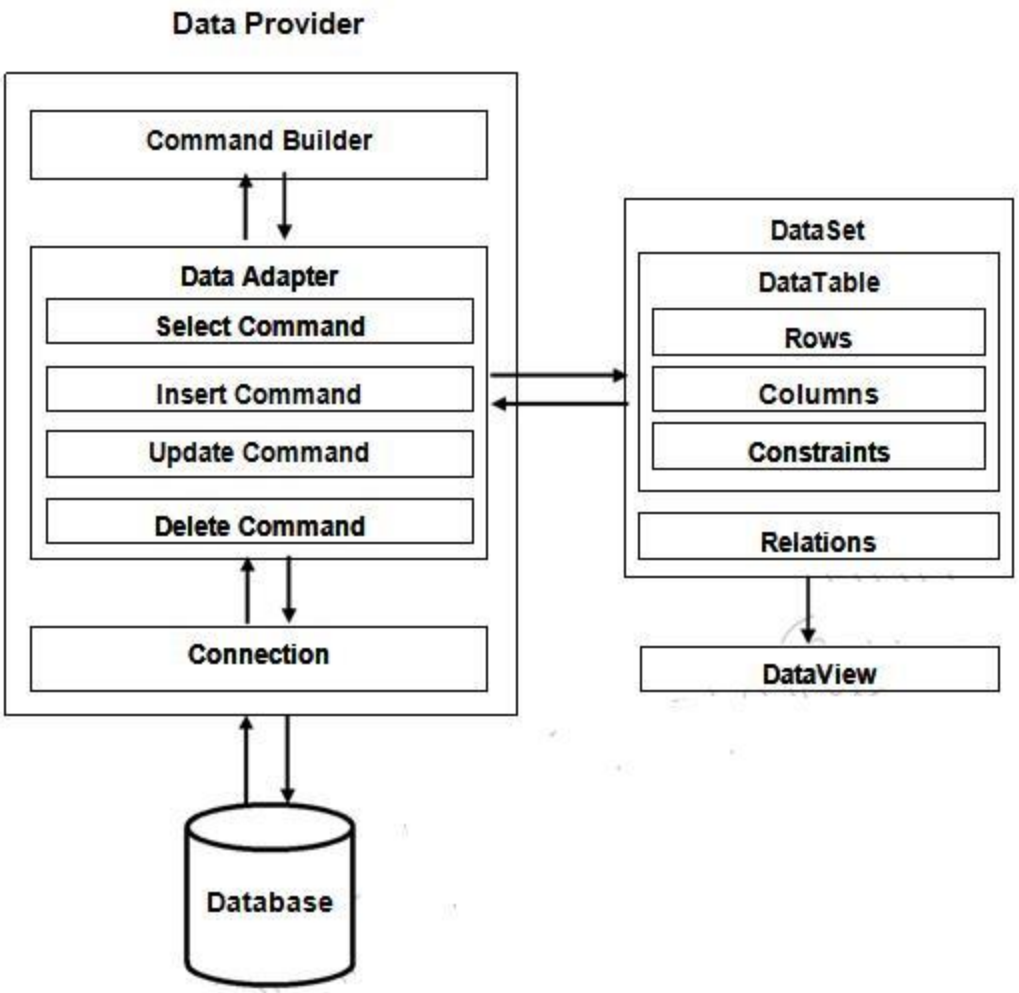
You can also get the value of particular column of the table by using the data reader object.

Disconnected Architecture

A connected mode of operation in ADO.Net is one in which the connection to the underlying database is alive throughout the lifetime of the operation. Meanwhile, a disconnected mode of operation is one in which ADO.Net retrieves data from the underlying database, stores the data retrieved temporarily in the memory, and then closes the connection to the database.
The architecture of ADO.net in which data retrieved from database can be accessed even when connection to database was closed is called as disconnected architecture. Disconnected architecture of ADO.net was built on classes connection, dataadapter, commandbuilder and dataset and dataview.

Connection : Connection object is used to establish a connection to database and connectionit self will not transfer any data.

DataAdapter : DataAdapter is used to transfer the data between database and dataset. It has commands like select, insert, update and delete. Select command is used to retrieve data from database and insert, update and delete commands are used to send changes to the data in dataset to database. It needs a connection to transfer the data.



CommandBuilder : by default dataadapter contains only the select command and it doesn'tcontain insert, update and delete commands. To create insert, update and delete commands for the dataadapter, commandbuilder is used. It is used only to create these commands for the dataadapter and has no other purpose.

DataSet : Dataset is used to store the data retrieved from database by dataadapter and make it available for .net application.

To fill data in to dataset **fill()** method of dataadapter is used and has the following syntax.

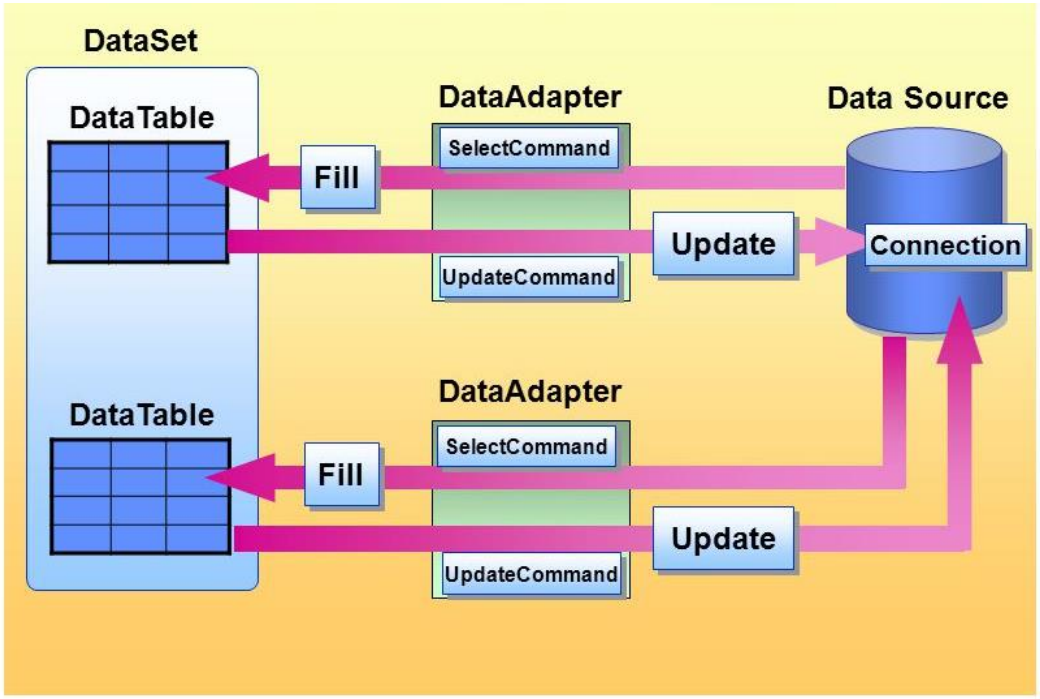
Da.Fill(Ds,"TableName");

When fill method was called, dataadapter will open a connection to database, executes select command, stores the data retrieved by select command in to dataset and immediately closes the connection.

As connection to database was closed, any changes to the data in dataset will not be directly sent to the database and will be made only in the dataset. To send changes made to data in dataset to the database, **Update()** method of the dataadapter is used that has the following syntax.

Da.Update(Ds,"Tablename");

When Update method was called, dataadapter will again open the connection to database, executes insert, update and delete commands to send changes in dataset to database and immediately closes the connection. As connection is opened only when it is required and will be automatically closed when it was not required, this architecture is called disconnected architecture. A dataset can contain data in multiple tables.

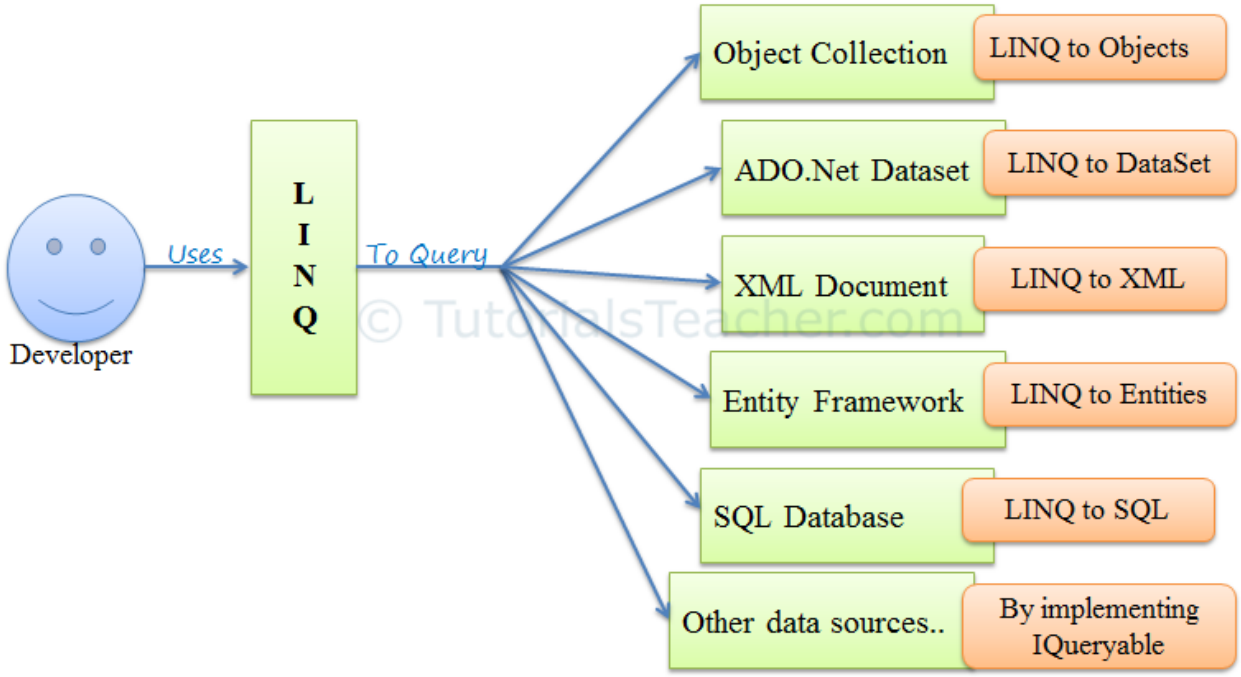


Difference between Connected and Disconnected Architecture

	Connected	Disconnected
Classes used	**Connection, Data Reader, Command**	**Connection, Command Builder, DataSet, DataAdapter, DataView**
Performance	Provide **faster** performance and speed.	Provides **slower** performance and speed when compared to connected arch.
Data Storage	DataReader can store only **one result Set**	DataSet Can store **multiple result sets.**
Data Accessing	**DataReader** object	**DataSet** object
Data Update	**DataReader** Can't update data directly, it follows read only manner.	Can update data using **Update ( )** of **DataAdapter**.
Data Persistence	Data Reader **can't persist** the data	Data Set **can persist** the data.
Data loosing	Data will not available once connection is lost.	**Data still available** in DataSet object even connection is lost.
No of Lines of code	No of lines of code will be more when using **DataReader** for accessing data.	Very few lines of code will be used for accessing data using DataSet.
Memory Usage	DataReader uses **less memory** and **system resources**	DataSet uses **more memory** and **system resources** when compared to DataReader.
Serializing	Serializing to XML or JSON format will become **complex** when using DataReader result.	Can **quickly serialize** DataSet to XML using **WriteXML( )**
Fetching huge data	**DataReader is most preferable** when trying to fetch millions or billions of records.	**Don't prefer DataSet** when fetching huge data.

What is LINQ?

LINQ is an acronym for Language Integrated Query, which is descriptive for where it's used and what it does. The *Language Integrated* part means that LINQ is part of programming language syntax. In particular, both C# and VB are languages that ship with .NET and have LINQ capabilities. Another programming language that supports LINQ is Delphi Prism. The other part of the definition, *Query*, explains what LINQ does; LINQ is used for querying data. Notice that I used the generic term "data" and didn't indicate what type of data. That's because LINQ can be used to query many different types of data, including relational, XML, and even objects. Another way to describe LINQ is that it is programming language syntax that is used to query data.



LINQ queries return results as objects. It enables you to uses object-oriented approach on the result set and not to worry about transforming different formats of results into objects.

We Already have ADO.NET, so Why Another Data Access Technology?

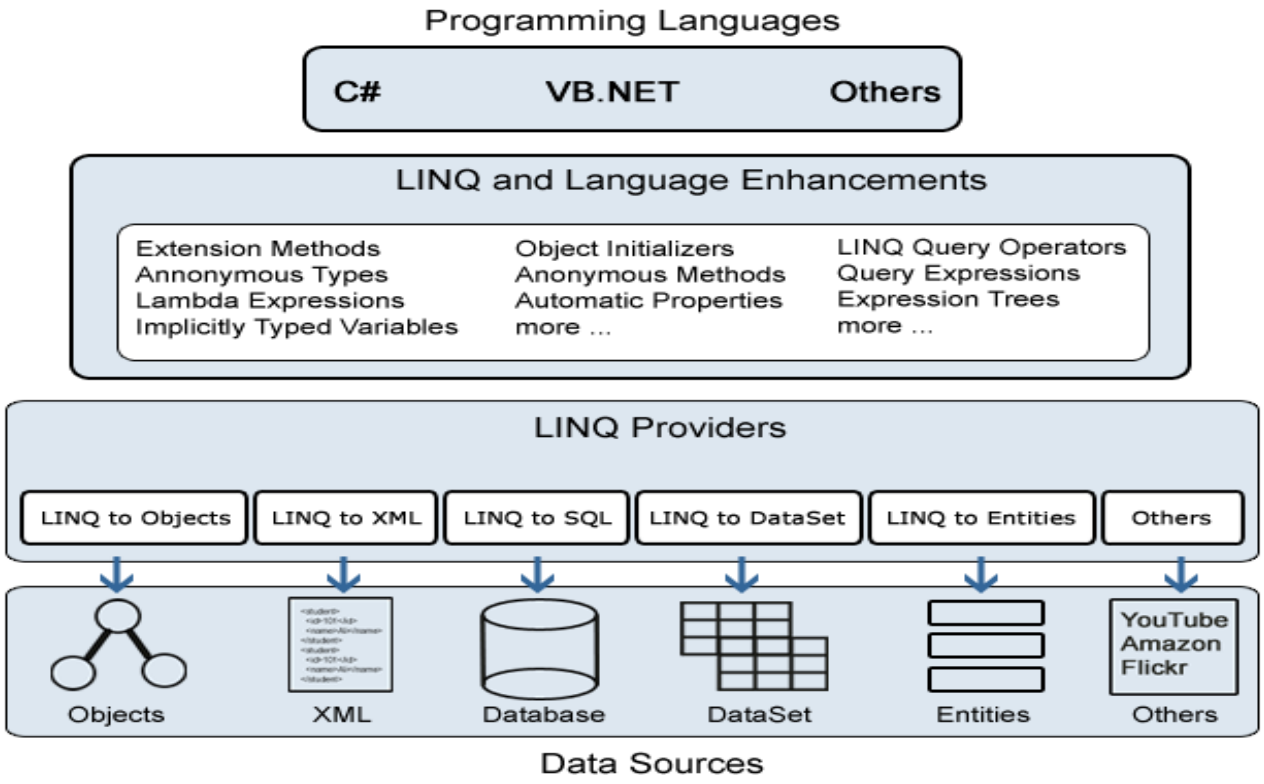
Most applications work with data in one form or another, meaning that data is very important to the work we do as software engineers. It's so important that the tools we use are constantly evolving, with the next generation building and improving upon the previous. This doesn't change with LINQ, which is the next giant leap in data development technology beyond ADO.NET.

ADO.NET is an object-oriented library, yet we must still reason about data from a relational perspective. In simple scenarios, we can bind ADO.NET objects directly to user interfaces (UI), but many other situations require the translation of the ADO.NET data into business objects with relationships, rules, and semantics that don't translate automatically from a relational data store. For example, a relational data store will model Orders and Customers with a foreign key from an Order table to a Customer table, but the object representation of this data is a Customer object with a collection of Order objects. Similar situations occur for other storage types such as hierarchical, multi-value, and flat data sources. This gap between the representation of data from the storage site to the objects you use in your applications is called an *Impedance Mismatch*. While ADO.NET is an object library for working with relational data, LINQ is a SQL-like syntax that produces usable objects. LINQ helps reduce this Impedance Mismatch.

Note: The operative term in the previous paragraph is "reduce". LINQ does not eliminate Impedance Mismatch, because you must still reason about your data store format. However, LINQ does remove a lot of the plumbing work you have to do to re-shape your data as an object.

More about Data Sources

LINQ is intended to make it easy to query data sources. One of the more popular uses of LINQ is to query relational databases. However, as you see here, LINQ can query objects. That's not all, the .NET Framework includes libraries that allow anyone to create a LINQ provider that can query any data source. Out of the box, Microsoft ships LINQ to Objects, LINQ to XML, LINQ to SQL (SQL Server), and LINQ to Entities (Entity Framework). There are also 3rd party LINQ providers that make it easy to query specialized data sources



In the query, we select elements from an array in descending order (high to low). We filter out elements <= 2. In the loop, we evaluate the expression and print the results.**Var**

C# program that uses query expression

```
using System;
using System.Linq;


class Program
{
   static void Main()
   {
```

```
      int[] array = { 1, 2, 3, 6, 7, 8 };
      // Query expression.
var elements = from element in array
                  orderby element descending
                  where element > 2
                  select element;
      // Enumerate.
foreach (var element in elements)
      {
         Console.Write(element);
         Console.Write(' ');
      }
      Console.WriteLine();
   }
}
```

Output

```
8 7 6 3
```

C# program that removes duplicate elements

```
using System;
using System.Linq;
class Program
{
   static void Main()
   {
      // Declare an array with some duplicated elements in it.
int[] array1 = { 1, 2, 2, 3, 4, 4 };
      // Invoke Distinct extension method.
var result = array1.Distinct();
      // Display results.
```

```
foreach (int value in result)
      {
         Console.WriteLine(value);
      }
   }
}
```

Output

```
1
2
3
4
```

C# program that uses Contains

```
using System;
using System.Collections.Generic;
using System.Linq;
class Program
{
   static void Main()
   {
      var list = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
      // Use extension method.
bool a = list.Contains<int>(7);
      // Use instance method.
bool b = list.Contains(7);
      Console.WriteLine(a);
      Console.WriteLine(b);
   }
}
```

Output TrueTrue

What is REST

REST is acronym for **RE**presentational **S**tate **T**ransfer. It is architectural style for **distributed hypermedia systems** and was first presented by Roy Fielding in 2000 in his famous dissertation.

Like any other architectural style, REST also does have it's own 6 guiding constraints which must be satisfied if an interface needs to be referred as **RESTful**. These principles are listed below.

Guiding Principles of REST

1. **Client–server** – By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.

2. **Stateless** – Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.

3. **Cacheable** – Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.

4. **Uniform interface** – By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.

5. **Layered system** – The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot "see" beyond the immediate layer with which they are interacting.

6. **Code on demand (optional)** – REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

Resource

The key abstraction of information in REST is a **resource**. Any information that can be named can be a resource: a document or image, a temporal service, a collection of other resources, a non-virtual object (e.g. a person), and so on. REST uses a **resource identifier** to identify the particular resource involved in an interaction between components.

The state of resource at any particular timestamp is known as **resource representation**. A representation consists of data, metadata describing the data and **hypermedia** links which can help the clients in transition to next desired state.

The data format of a representation is known as a media type. The media type identifies a specification that defines how a representation is to be processed. **A truly RESTful API looks like** *hypertext*. Every addressable unit of information carries an address, either explicitly (e.g., link and id attributes) or implicitly (e.g., derived from the media type definition and representation structure).

According to Roy Fielding:

> Hypertext (or hypermedia) mean the **simultaneous presentation of information and controls** such that the information becomes the affordance through which the user (or automaton) obtains choices and selects actions. Remember that hypertext does not need to be HTML (or XML or JSON) on a browser. Machines can follow links when they understand the data format and relationship types.

Further, **resource representations shall be self-descriptive**: the client does not need to know if a resource is employee or device. It should act on basis of media-type associated with resource. So in practice, you will end up creating lots of **custom media-types** – normally one media-type associated with one resource.

> Every media type defines a default processing model. For example, HTML defines a rendering process for hypertext and the browser behavior around each element. It has no relation to the resource methods GET/PUT/POST/DELETE/… other than the fact that some media type elements will define a process model that goes like “anchor elements with an href attribute create a hypertext link that, when selected, invokes a retrieval request (GET) on the URI corresponding to the CDATA-encoded href attribute.”

Resource Methods

Other important thing associated with REST is **resource methods** to be used to perform the desired transition. A large number of people wrongly relate resource methods to HTTP **GET/PUT/POST/DELETE** methods.

Roy Fielding has never mentioned any recommendation around which method to be used in which condition. All he emphasizes is that it should be **uniform interface**. If you decide HTTP POST will be used for updating a resource – rather than most people recommend HTTP PUT – it's alright and application interface will be RESTful.

Ideally, everything that is needed to change the resource state shall be part of API response for that resource – including methods and in what state they will leave the representation.

A REST API should be entered with no prior knowledge beyond the initial URI (bookmark) and set of standardized media types that are appropriate for the intended audience (i.e., expected to be understood by any client that might use the API). From that point on, all application state transitions must be driven by client selection of server-provided choices that are present in the received representations or implied by the user's manipulation of those representations. The transitions may be determined (or limited by) the client's knowledge of media types and resource communication mechanisms, both of which may be improved on-the-fly (e.g., code-on-demand).
[Failure here implies that out-of-band information is driving interaction instead of hypertext.]

Another thing which will help you while building RESTful APIs is that **query based API results should be represented by a list of links with summary information**, not by arrays of original resource representations because query is not a substitute for identification of resources.

REST and HTTP are not same !!

A lot of people prefer to compare HTTP with REST. **REST and HTTP are not same.**

REST != HTTP

Though, because REST also intend to make web (internet) more streamline and standard, he advocate to use REST principles more strictly. And that's from where people try to start comparing REST with web (HTTP). Roy fielding, in his dissertation, nowhere mentioned any implementation directive – including any protocol preference and HTTP. Till the time, you are honoring the 6 guiding principles of REST, you can call your interface RESTful.

In simplest words, in the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs). The resources are acted upon by using a set of simple, well-defined operations. The clients and servers exchange representations of resources by using a standardized interface and protocol – typically HTTP.

Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others. Metadata about the resource is available and used, for example, to control caching, detect transmission errors, negotiate the appropriate representation format, and perform authentication or access control.And most importantly, every interaction with a resource is stateless.

All these principles help RESTful applications to be simple, lightweight, and fast.

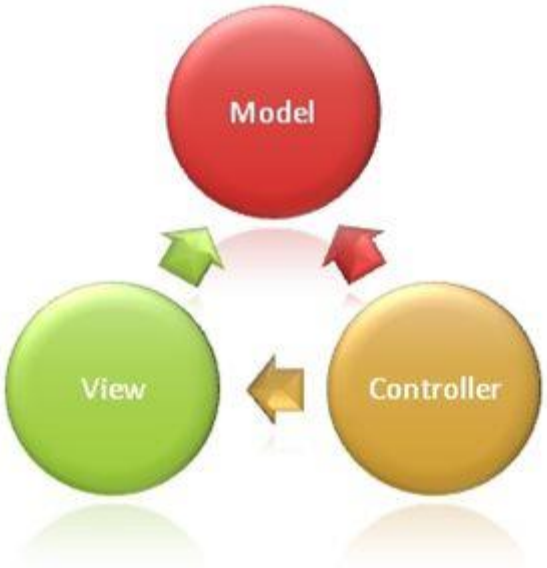
What is MVC (Model view controller)?

Model–view–controller (MVC) is a software architectural pattern for implementing user interfaces. It divides a given software application into three interconnected parts, so as to separate internal representation of information from the way that information is presented to or accepted from the user.

MVC is a framework for building web applications using a MVC (Model View Controller) design:

- The Model represents the application core (for instance a list of database records).
- The View displays the data (the database records).
- The Controller handles the input (to the database records).

The MVC model also provides full control over HTML, CSS, and JavaScript.



The MVC model defines web applications with 3 logic layers,

- The business layer (Model logic)

- The display layer (View logic)
- The input control (Controller logic)

The Model is the part of the application that handles the logic for the application data.

Often model objects retrieve data (and store data) from a database.

The View is the part of the application that handles the display of the data.

Most often the views are created from the model data.

The Controller is the part of the application that handles user interaction.

Typically controllers read data from a view, control user input, and send input data to the model.

The MVC separation helps you manage complex applications, because you can focus on one aspect a time. For example, you can focus on the view without depending on the business logic. It also makes it easier to test an application.

The MVC separation also simplifies group development. Different developers can work on the view, the controller logic, and the business logic in parallel.

What are the advantages of MVC?

- *Multiple view support*
  Due to the separation of the model from the view, the user interface can display multiple views of the same data at the same time.

- *Change Accommodation*
  User interfaces tend to change more frequently than business rules (different colors, fonts, screen layouts, and levels of support for new devices such as cell phones or PDAs) because the model does not depend on the views, adding new types of views to the system generally does not affect the model. As a result, the scope of change is confined to the view.

SoC – Separation of Concerns

Separation of Concerns is one of the core advantages of ASP.NET MVC . The MVC framework provides a clean separation of the UI, Business Logic, Model or Data.

More Control

The ASP.NET MVC framework provides more control over HTML, JavaScript and CSS than the traditional Web Forms.

Testability

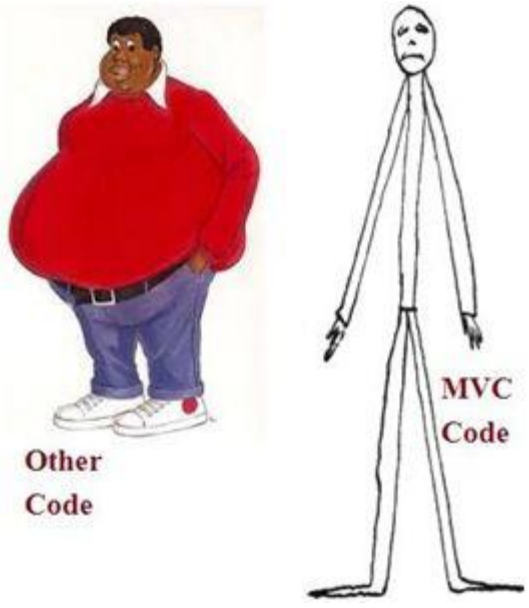
ASP.NET MVC framework provides better testability of the Web Application and good support for the test driven development too.

Lightweight

ASP.NET MVC framework doesn't use View State and thus reduces the bandwidth of the requests to an extent.

Full features of ASP.NET

One of the key advantages of using ASP.NET MVC is that it is built on top of ASP.NET framework and hence most of the features of the ASP.NET like membership providers, roles, etc can still be used.



Explain MVC application life cycle?

Any web application has two main execution steps, first understanding the request and depending on the type of the request sending out appropriate response. MVC application life cycle is not different it has two main phases, first creating the request object and second sending our response to the browser.

Creating the request object,

The request object creation has four major steps. The following is the detailed explanation of the same.

Step 1 - Fill route

MVC requests are mapped to route tables which in turn specify which controller and action to be invoked. So if the request is the first request the first thing is to fill the route table with routes collection. This filling of route table happens in the global.asax file.

Step 2 - Fetch route

Depending on the URL sent "UrlRoutingModule" searches the route table to create "RouteData" object which has the details of which controller and action to invoke.

Step 3 - Request context created

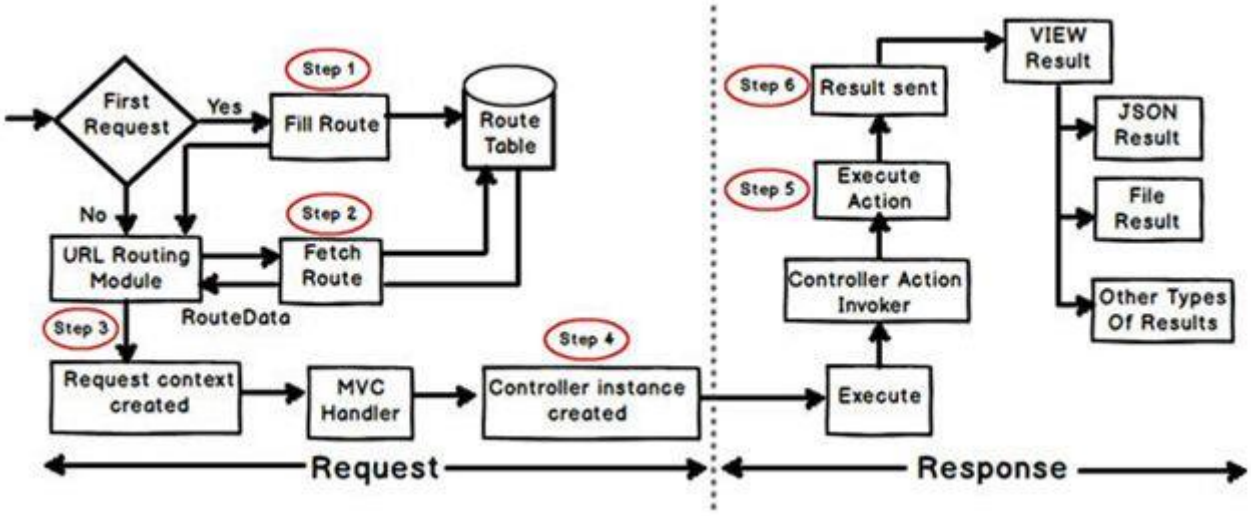
The "RouteData" object is used to create the "RequestContext" object.

Step 4 - Controller instance created

This request object is sent to "MvcHandler" instance to create the controller class instance. Once the controller class object is created it calls the "Execute" method of the controller class.

Creating Response object

This phase has two steps executing the action and finally sending the response as a result to the view.



List out different return types of a controller action method?

There are total nine return types we can use to return results from controller to view.

The base type of all these result types is ActionResult.

1. *ViewResult (View)*
   This return type is used to return a webpage from an action method.

2. *PartialviewResult (Partialview)*
   This return type is used to send a part of a view which will be rendered in another view.

3. *RedirectResult (Redirect)*
   This return type is used to redirect to any other controller and action method depending on the URL.

4. *RedirectToRouteResult (RedirectToAction, RedirectToRoute)*
   This return type is used when we want to redirect to any other action method.

5. *ContentResult (Content)*
   This return type is used to return HTTP content type like text/plain as the result of the action.

6. *jsonResult (json)*
   This return type is used when we want to return a JSON message.

7. *javascriptResult (javascript)*
   This return type is used to return JavaScript code that will run in browser.

8. *FileResult (File)*
   This return type is used to send binary output in response.

9. *EmptyResult*
   This return type is used to return nothing (void) in the result.

What are Filters in MVC?

In MVC, controllers define action methods and these action methods generally have a one-to-one relationship with UI controls such as clicking a button or a link, etc. For example, in one of our previous examples, the UserController class contained methods UserAdd, UserDelete, etc.

But many times we would like to perform some action before or after a particular operation. For achieving this functionality, ASP.NET MVC provides feature to add pre and post action behaviors on controller's action methods.

Types of Filters

ASP.NET MVC framework supports the following action filters,

- *Action Filters*
  Action filters are used to implement logic that gets executed before and after a controller action executes. We will look at Action Filters in detail in this chapter.

- *Authorization Filters*
  Authorization filters are used to implement authentication and authorization for controller actions.

- *Result Filters*
  Result filters contain logic that is executed before and after a view result is executed. For example, you might want to modify a view result right before the view is rendered to the browser.

- *Exception Filters*
  Exception filters are the last type of filter to run. You can use an exception filter to

handle errors raised by either your controller actions or controller action results. You can also use exception filters to log errors.

Action filters are one of most commonly used filters to perform additional data processing, or manipulating the return values or cancelling the execution of action or modifying the view structure at run time.

What are Action Filters in MVC?

Action Filters are additional attributes that can be applied to either a controller section or the entire controller to modify the way in which action is executed. These attributes are special .NET classes derived from System.Attribute which can be attached to classes, methods, properties and fields.

ASP.NET MVC provides the following action filters,

- *Output Cache*
  This action filter caches the output of a controller action for a specified amount of time.

- *Handle Error*
  This action filter handles errors raised when a controller action executes.

- *Authorize*
  This action filter enables you to restrict access to a particular user or role.

Now we will see the code example to apply these filters on an example controller ActionFilterDemoController. (ActionFilterDemoController is just used as an example. You can use these filters on any of your controllers.)

Output Cache

Code Example

Specifies the return value to be cached for 10 seconds.

```
1. publicclassActionFilterDemoController: Controller
2. {
3.     [HttpGet]
4.     OutputCache(Duration = 10)]
5. publicstringIndex()
6. {
```

```
7.      returnDateTime.Now.ToString("T");
8.
9.  }
10. }
```

Explain what is routing in MVC? What are the three segments for routing important?

Routing is a mechanism to process the incoming url that is more descriptive and give desired response. In this case, URL is not mapped to specific files or folder as was the case of earlier days web sites.

There are two types of routing (after the introduction of ASP.NET MVC 5).

1. Convention based routing - to define this type of routing, we call MapRoute method and set its unique name, url pattern and specify some default values.
2. Attribute based routing - to define this type of routing, we specify the Route attribute in the action method of the controller.

Routing is the URL pattern that is mappped together to a handler,rounting is responsible for incoming browser request for particular MVC controller. In other ways let us say routing help you to define a URL structure and map the URL with controller. There are three segments for routing that are important,

1. ControllerName
2. ActionMethodName
3. Parammeter

ControllerName/ActionMethodName/{ParamerName} and also route map coding written in a Global.asax file.

What is Route in MVC? What is Default Route in MVC?

A route is a URL pattern that is mapped to a handler. The handler can be a physical file, such as a .aspx file in a Web Forms application. A handler can also be a class that processes the request, such as a controller in an MVC application. To define a route, you create an instance of the Route class by specifying the URL pattern, the handler, and optionally a name for the route.

You add the route to the application by adding the Route object to the static Routes property of the RouteTable class. The Routesproperty is a RouteCollection object that stores all the routes for the application.
You typically do not have to write code to add routes in an MVC application. Visual Studio

project templates for MVC include preconfigured URL routes. These are defined in the Mvc Application class, which is defined in the Global.asax file.

Route definition	Example of matching URL
{controller}/{action}/{id}	/Products/show/beverages
{table}/Details.aspx	/Products/Details.aspx
blog/{action}/{entry}	/blog/show/123
{reporttype}/{year}/{month}/{day}	/sales/2008/1/5
{locale}/{action}	/US/show
{language}-{country}/{action}	/en-US/show

Default Route

The default ASP.NET MVC project templates add a generic route that uses the following URL convention to break the URL for a given request into three named segments.

URL: "{controller}/{action}/{id}"

This route pattern is registered via call to the MapRoute() extension method of RouteCollection.

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}",
        defaults: new { controller = "Home", action = "Index",
            id = UrlParameter.Optional }
    );
}
```

Ignore routes that end with axd extension

Route Name

URL Pattern

Default Values

Mention what is the difference between Temp data, View, and View Bag?

In ASP.NET MVC there are three ways to pass/store data between the controllers and views.

ViewData

1. ViewData is used to pass data from controller to view.
2. It is derived from ViewDataDictionary class.
3. It is available for the current request only.
4. Requires typecasting for complex data type and checks for null values to avoid error.
5. If redirection occurs, then its value becomes null.

ViewBag

1. ViewBag is also used to pass data from the controller to the respective view.
2. ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0
3. It is also available for the current request only.
4. If redirection occurs, then its value becomes null.
5. Doesn't require typecasting for complex data type.

TempData

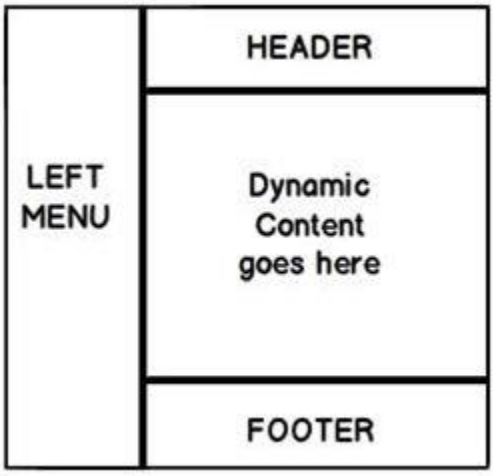
1. TempData is derived from TempDataDictionary class
2. TempData is used to pass data from the current request to the next request
3. It keeps the information for the time of an HTTP Request. This means only from one page to another. It helps to maintain the data when we move from one controller to another controller or from one action to another action
4. It requires typecasting for complex data type and checks for null values to avoid error. Generally, it is used to store only one time messages like the error messages and validation messages

What is Partial View in MVC?

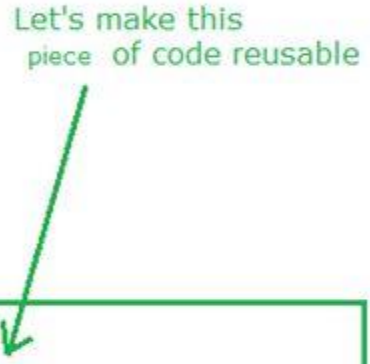
A partial view is a chunk of HTML that can be safely inserted into an existing DOM. Most commonly, partial views are used to componentize Razor views and make them easier to build and update. Partial views can also be returned directly from controller methods. In this case, the browser still receives text/html content but not necessarily HTML content that makes up an entire page. As a result, if a URL that returns a partial view is directly invoked from the address

bar of a browser, an incomplete page may be displayed. This may be something like a page that misses title, script and style sheets. However, when the same URL is invoked via script, and the response is used to insert HTML within the existing DOM, then the net effect for the end user may be much better and nicer.

Partial view is a reusable view (like a user control) which can be embedded inside other view. For example, let's say all the pages of your site have a standard structure with left menu, header, and footer as in the following image,



```
@model IEnumerable<MvcApplication8.Models.Product>

@{
    ViewBag.Title = "Index";
}

<h2>Products</h2>

<table>
    <tr>
        <th>
            Name
        </th>
        <th>
            Description
        </th>
        <th>
            Price
        </th>
        <th>
            EDD
        </th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @item.Name
        </td>
        <td>
            @item.Description
        </td>
        <td>
            @(item.Price/10)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.ProductId }) |
            @Html.ActionLink("Details", "Details", new { id=item.ProductId }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.ProductId })
        </td>
    </tr>
}
</table>
```



Explain what is the difference between View and Partial View?

View

- It contains the layout page.
- Before any view is rendered, viewstart page is rendered.
- View might have markup tags like body, html, head, title, meta etc.
- View is not lightweight as compare to Partial View.

Partial View

- It does not contain the layout page.
- Partial view does not verify for a viewstart.cshtml.We cannot put common code for a partial view within the viewStart.cshtml.page.
- Partial view is designed specially to render within the view and just because of that it does not consist any mark up.
- We can pass a regular view to the RenderPartial method.

What are HTML helpers in MVC?

With MVC, HTML helpers are much like traditional ASP.NET Web Form controls.

Just like web form controls in ASP.NET, HTML helpers are used to modify HTML. But HTML helpers are more lightweight. Unlike Web Form controls, an HTML helper does not have an event model and a view state.

In most cases, an HTML helper is just a method that returns a string.

With MVC, you can create your own helpers, or use the built in HTML helpers.

Standard HTML Helpers

HTML Links

The easiest way to render an HTML link in is to use the HTML.ActionLink() helper.With MVC, the Html.ActionLink() does not link to a view. It creates a link to a controller action.

ASP Syntax

```
1. <%=Html.ActionLink("About this Website", "About")%>
```

The first parameter is the link text, and the second parameter is the name of the controller action.

The Html.ActionLink() helper above, outputs the following HTML:

```
1. <a href="/Home/About">About this Website</a>
```

The Html.ActionLink() helper has several properties:

- Property Description.
- .linkText The link text (label).
- .actionName The target action.
- .routeValues The values passed to the action.
- .controllerName The target controller.
- .htmlAttributes The set of attributes to the link.
- .protocol The link protocol.
- .hostname The host name for the link.
- .fragment The anchor target for the link.

HTML Form Elements

There following HTML helpers can be used to render (modify and output) HTML form elements:

- BeginForm()
- EndForm()
- TextArea()
- TextBox()
- CheckBox()
- RadioButton()
- ListBox()
- DropDownList()
- Hidden()
- Password()

ASP.NET Syntax C#

```
1. <%= Html.ValidationSummary("Create was unsuccessful. Please correct the errors and tr
   y again.") %>
2.     <% using (Html.BeginForm()){%>
3.         <p>
4.             <label for="FirstName">First Name:</label>
5.             <%= Html.TextBox("FirstName") %>
6.                 <%= Html.ValidationMessage("FirstName", "*") %>
7.         </p>
8.         <p>
9.             <label for="LastName">Last Name:</label>
10.            <%= Html.TextBox("LastName") %>
11.                <%= Html.ValidationMessage("LastName", "*") %>
12.        </p>
13.        <p>
14.            <label for="Password">Password:</label>
```

```
15.          <%= Html.Password("Password") %>
16.             <%= Html.ValidationMessage("Password", "*") %>
17.       </p>
18.       <p>
19.          <label for="Password">Confirm Password:</label>
20.          <%= Html.Password("ConfirmPassword") %>
21.             <%= Html.ValidationMessage("ConfirmPassword", "*") %>
22.       </p>
23.       <p>
24.          <label for="Profile">Profile:</label>
25.          <%= Html.TextArea("Profile", new {cols=60, rows=10})%>
26.       </p>
27.       <p>
28.          <%= Html.CheckBox("ReceiveNewsletter") %>
29.             <label for="ReceiveNewsletter" style="display:inline">Receive Newsletter?</la
    bel>
30.       </p>
31.       <p>
32.          <input type="submit" value="Register" />
33.       </p>
34.       <%}%>
```

Explain attribute based routing in MVC?

In ASP.NET MVC 5.0 we have a new attribute route,cBy using the "Route" attribute we can define the URL structure. For example in the below code we have decorated the "GotoAbout" action with the route attribute. The route attribute says that the "GotoAbout" can be invoked using the URL structure "Users/about".

```
1.  public class HomeController: Controller
2.  {
3.     [Route("Users/about")]
4.     publicActionResultGotoAbout()
5.     {
6.        return View();
7.     }
8.  }
```

What is TempData in MVC?

TempData is a dictionary object to store data temporarily. It is a TempDataDictionary class type and instance property of the Controller base class.

TempData is able to keep data for the duration of a HTP request, in other words it can keep live data between two consecutive HTTP requests. It will help us to pass the state between action methods. TempData only works with the current and subsequent request. TempData uses a session variable to store the data. TempData Requires type casting when used to retrieve data.

TempDataDictionary is inherited from the IDictionary<string, object>, ICollection<KeyValuePair<string, object>>, IEnumerable<KeyValuePair<string, object>> and IEnumerable interfaces.

Example

```
1.  public ActionResult FirstRequest()
2.  {
3.      List < string > TempDataTest = new List < string > ();
4.      TempDataTest.Add("Tejas");
5.      TempDataTest.Add("Jignesh");
6.      TempDataTest.Add("Rakesh");
7.      TempData["EmpName"] = TempDataTest;
8.      return View();
9.  }
10. public ActionResult ConsecutiveRequest()
11. {
12.     List < string > modelData = TempData["EmpName"] as List < string > ;
13.     TempData.Keep();
14.     return View(modelData);
15. }
```

What is Razor in MVC?

ASP.NET MVC has always supported the concept of "view engines" - which are the pluggable modules that implement different template syntax options. The "default" view engine for ASP.NET MVC uses the same .aspx/.ascx/. master file templates as ASP.NET Web Forms. Other popular ASP.NET MVC view engines are Spart&Nhaml.

MVC 3 has introduced a new view engine called Razor.

Why is Razor?

1. Compact & Expressive.
2. Razor minimizes the number of characters and keystrokes required in a file, and enables a fast coding workflow. Unlike most template syntaxes, you do not need to interrupt your coding to explicitly denote server blocks within your HTML. The parser is smart enough to infer this from your code. This enables a really compact and expressive syntax which is clean, fast and fun to type.
3. Easy to Learn: Razor is easy to learn and enables you to quickly be productive with a minimum of effort. We can use all your existing language and HTML skills.
4. Works with any Text Editor: Razor doesn't require a specific tool and enables you to be productive in any plain old text editor (notepad works great).
5. Has great Intellisense:
6. Unit Testable: The new view engine implementation will support the ability to unit test views (without requiring a controller or web-server, and can be hosted in any unit test project - no special app-domain required).

Differences between Razor and ASPX View Engine in MVC?

Razor View Engine	ASPX View Engine (Web form view engine)
The namespace used by the Razor View Engine is System.Web.Razor	The namespace used by the ASPX View Engine is System.Web.Mvc.WebFormViewEngine
The file extensions used by the Razor View Engine are different from a web form view engine. It uses cshtml with C# and vbhtml with vb for views, partial view, templates and layout pages.	The file extensions used by the Web Form View Engines are like ASP.Net web forms. It uses the ASPX extension to view the aspc extension for partial views or User Controls or templates and master extensions for layout/master pages.
The Razor View Engine is an advanced view engine that was introduced with MVC 3.0. This is not a new language but it is	A web form view engine is the default view engine and available from the beginning of MVC

markup.	
Razor has a syntax that is very compact and helps us to reduce typing.	The web form view engine has syntax that is the same as an ASP.Net forms application.
The Razor View Engine uses @ to render server-side content.	The ASPX/web form view engine uses "<%= %>" or "<%: %>" to render server-side content.
By default all text from an @ expression is HTML encoded.	There is a different syntax ("<%: %>") to make text HTML encoded.
Razor does not require the code block to be closed, the Razor View Engine parses itself and it is able to decide at runtime which is a content element and which is a code element.	A web form view engine requires the code block to be closed properly otherwise it throws a runtime exception.
The Razor View Engine prevents Cross Site Scripting (XSS) attacks by encoding the script or HTML tags before rendering to the view.	A web form View engine does not prevent Cross Site Scripting (XSS) attack.
The Razor Engine supports Test Driven Development (TDD).	Web Form view engine does not support Test Driven Development (TDD) because it depends on the System.Web.UI.Page class to make the testing complex.
Razor uses "@* â€¦ *@" for multiline comments.	The ASPX View Engine uses "<!--...-->" for markup and "/* â€¦ */" for C# code.
There is only three transition characters with the Razor View Engine.	There are only three transition characters with the Razor View Engine.

The Razor View Engine is a bit slower than the ASPX View Engine.

What are the Main Razor Syntax Rules?

Razor code blocks are enclosed in @{ ... }

- Inline expressions (variables and functions) start with @
- Code statements end with semicolon
- Variables are declared with the var keyword
- Strings are enclosed with quotation marks
- C# code is case sensitive
- C# files have the extension .cshtml

C# Example

```
<!-- Single statement block -->
@ {
    varmyMessage = "Hello World";
}
<!-- Inline expression or variable -->
< p > The value of myMessage is: @myMessage < /p>
    <!-- Multi-statement block -->
@ {
    var greeting = "Welcome to our site!";
    varweekDay = DateTime.Now.DayOfWeek;
    vargreetingMessage = greeting + " Here in Huston it is: " + weekDay;
} < p > The greeting is: @greetingMessage < /p>
```

How do you implement Forms authentication in MVC?

Authentication is giving access to the user for a specific service by verifying his/her identity using his/her credentials like username and password or email and password. It assures that the correct user is authenticated or logged in for a specific service and the right service has been provided to the specific user based on their role that is nothing but authorization.

ASP.NET forms authentication occurs after IIS authentication is completed. You can configure forms authentication by using forms element with in web.config file of your application. The default attribute values for forms authentication are shown below,

```
<system.web>
```

```
2.     <authenticationmode="Forms">
3.         <formsloginUrl="Login.aspx" protection="All" timeout="30" name=".ASPXAUTH" pa
       th="/" requireSSL="false" slidingExpiration="true" defaultUrl="default.aspx" cookieless=
       "UseDeviceProfile" enableCrossAppRedirects="false" />
4.         </authentication>
5.   </system.web>
```

The FormsAuthentication class creates the authentication cookie automatically when SetAuthCookie() or RedirectFromLoginPage() methods are called. The value of authentication cookie contains a string representation of the encrypted and signed FormsAuthenticationTicket object.

What is Node.js?

Node.js is an open-source, cross-platform runtime environment used for development of server-side web applications. Node.js applications are written in JavaScript and can be run on a wide variety of operating systems.

Node.js is based on an event-driven architecture and a non-blocking Input/Output API that is designed to optimize an application's throughput and scalability for real-time web applications.

Over a long period of time, the framework available for web development were all based on a stateless model. A stateless model is where the data generated in one session (such as information about user settings and events that occurred) is not maintained for usage in the next session with that user.

A lot of work had to be done to maintain the session information

var fs = require('fs'); fs.readFile("Sample.txt",function(error,data) { console.log("Reading Data completed"); });

between requests for a user. But with Node.js there is finally a way for web applications to have a real-time, two-way connections, where both the client and server can initiate communication, allowing them to exchange data freely.

Why use Node.js?

We will have a look into the real worth of Node.js in the coming chapters, but what is it that makes this framework so famous. Over the years, most of the applications were based on a stateless request- response framework. In these sort of applications, it is up to the developer to ensure the right code was put in place to ensure the state of web session was maintained while the user was working with the system.

But with Node.js web applications, you can now work in real-time and have a 2-way communication. The state is maintained, and the either the client or server can start the communication.

Features of Node.js

Let's look at some of the key features of Node.js

1. Asynchronous event driven IO helps concurrent request handling – This is probably the biggest selling points of Node.js. This feature basically means that if a request is received by Node for some Input/Output operation, it will execute the operation in the background and continue with processing other requests.

This is quite different from other programming languages. A simple example of this is given in the code below

The above code snippet looks at reading a file called Sample.txt. In other programming languages, the next line of processing would only happen once the entire file is read. But in the case of Node.js the important fraction of code to notice is the declaration of the function ('function(error,data)'). This is known as a callback function. So what happens here is that the file reading operation will start in the background. And other processing can happen simultaneously while the file is being read. Once the file read operation is completed, this anonymous function will be called and the text "Reading Data completed" will be written to the console log.

2. Node uses the V8 JavaScript Runtime engine, the one which is used by Google Chrome. Node has a wrapper over the JavaScript engine which makes the runtime engine much faster and hence processing of requests within Node also become faster. 3. Handling of concurrent requests – Another key functionality of Node is the ability to handle concurrent connections with a very minimal overhead on a single process. 4. The Node.js library used JavaScript – This is another important aspect of development in Node.js. A major part of the development community are already well versed in javascript, and hence, development in Node.js becomes easier for a developer who knows javascript. 5. There are an Active and vibrant community for the Node.js framework. Because of the active community, there are always keys updates made available to the framework. This helps to keep the framework always upto-date with the latest trends in web development.

Who uses Node.js

Node.js is used by a variety of large companies. Below is a list of a few of them.

Paypal – A lot of sites within Paypal have also started the transition onto Node.js. LinkedIn - LinkedIn is using Node.js to power their Mobile Servers, which powers the iPhone, Android, and Mobile Web products. Mozilla has implemented Node.js to support browser APIs which has half a billion installs. Ebay hosts their HTTP API service in Node.js

When to Use Node.js

Node.js is best for usage in streaming or event-based real-time applications like

1. Chat applications 2. Game servers – Fast and high-performance servers that need to processes thousands of requests at a time, then this is an ideal framework. 3. Good for collaborative environment – This is good for environments which manage document. In document management environment you will have multiple people who post their documents and do constant changes by checking out and checking in documents. So Node.js is good for these environments because the event loop in Node.js can be triggered whenever documents are changed in a document managed environment. 4. Advertisement servers – Again here you could have thousands of request to pull advertisements from the central server and Node.js can be an ideal framework to handle this. 5. Streaming servers –

Another ideal scenario to use Node is for multimedia streaming servers wherein clients have request's to pull different multimedia contents from this server.

Node.js is good when you need high levels of concurrency but less amount of dedicated CPU time.

Best of all, since Node.js is built on javascript, it's best suited when you build client-side applications which are based on the same javascript framework.

When to not use Node.js

Node.js can be used for a lot of applications with various purpose, the only scenario where it should not be used is if there are long processing times which is required by the application.

Node is structured to be single threaded. If any application is required to carry out some long running calculations in the background. So if the server is doing some calculation, it won't be able to process any other requests. As discussed above, Node.js is best when processing needs less dedicated CPU time.

Running your first Hello world application in Node.js

```
var http = require('http');

http.createServer(function (req, res) { res.writeHead(200, {'Content-Type': 'text/html'}); res.end('Hello World!'); }).listen(8080);
```

Once you have downloaded and installed Node.js on your computer, lets try to display "Hello World" in a web browser.

Create file Node.js with file name firstprogram.js

Code Explanation:

1. The basic functionality of the "require" function is that it reads a JavaScript file, executes the file, and then proceeds to return an object. Using this object, one can then use the various functionalities available in the module called by the require function. So in our case, since we want to use the functionality of http and we are using the require(http) command. 2. In this 2nd line of code, we are creating a server application which is based on a simple function. This function is called, whenever a request is made to our server application. 3. When a request is received, we are asking our function to return a "Hello World" response to the client. The writeHead function is used to send header data to the client and while the end function will close the connection to the client. 4. We are then using the server.listen function to make our server application listen to client requests on port no 8080. You can specify any available port over here.

Executing the code

1. Save the file on your computer: C:\Users\Your Name\ firstprogram.js

2. In the command prompt, navigate to the folder where the file is stored. Enter the command Node firstprogram.js

3. Now, your computer works as a server! If anyone tries to access your computer on port 8080, they will get a “Hello World!” message in return!

4. Start your internet browser, and type in the address: http://localhost:8080

What is Event-driven programming?

Event-driven programming is building our application based on and respond to events. When an event occurs, like click or keypress, we are running a callback function which is registered to the element for that event.

Event driven programming follows mainly a publish-subscribe pattern.

What is *Event loop* in Node.js work? And How does it work?

The *Event loop* handles all async callbacks. Node.js (or JavaScript) is a single-threaded, event-driven language. This means that we can attach listeners to events, and when a said event fires, the listener executes the callback we provided.

Whenever we are call setTimeout, http.get and fs.readFile, Node.js runs this operations and further continue to run other code without waiting for the output. When the operation is finished, it receives the output and runs our callback function.

So all the callback functions are queued in an loop, and will run one-by-one when the response has been received.

What is the difference between Asynchronous and Non-blocking?

Asynchronous literally means not synchronous. We are making HTTP requests which are asynchronous, means we are not waiting for the server response. We continue with other block and respond to the server response when we received.

The term Non-Blocking is widely used with IO. For example non-blocking read/write calls return with whatever they can do and expect caller to execute the call again. Read will wait until it has some data and put calling thread to sleep.

What is package.json? What is it used for?

This file holds various metadata information about the project. This file is used to give information to npm that allows it to identify the project as well as handle the project's dependencies.

Some of the fields are: name, name, description, author and dependencies.

When someone installs our project through npm, all the dependencies listed will be installed as well. Additionally, if someone runs npm install in the root directory of our project, it will install all the dependencies to ./node_modules directory.

What is EventEmitter in Node.js?

All objects that emit events are instances of the EventEmitter class. These objects expose an eventEmitter.on() function that allows one or more functions to be attached to named events emitted by the object.

When the EventEmitter object emits an event, all of the functions attached to that specific event are called *synchronously*.

Is Node.Js Entirely Based On A Single-Thread?

Answer.

Yes, it's true that Node.js processes all requests on a single thread. But it's just a part of the theory behind Node.js design. In fact, more than the single thread mechanism, it makes use of events and callbacks to handle a large no. of requests asynchronously.

Moreover, Node.js has an optimized design which utilizes both JavaScript and C++ to guarantee maximum performance. JavaScript executes at the server-side by Google Chrome v8 engine. And the C++ lib UV library takes care of the non-sequential I/O via background workers.

To explain it practically, let's assume there are 100s of requests lined up in Node.js queue. As per design, the main thread of Node.js event loop will receive all of them and forwards to background workers for execution. Once the workers finish processing requests, the registered callbacks get notified on event loop thread to pass the result back to the user.

Can You Create HTTP Server In Nodejs, Explain The Code Used For It?

Answer.

Yes, we can create HTTP Server in Node.js. We can use the <**http-server**> command to do so.
Following is the sample code.

```
var http = require('http');

var requestListener = function (request, response) {

response.writeHead(200, {'Content-Type': 'text/plain'});

response.end('Welcome Viewers\n');

}

var server = http.createServer(requestListener);

server.listen(8080); // The port where you want to start with.
```

What Is The Difference Between Nodejs, AJAX, And JQuery?

Answer.
The one common trait between Node.js, AJAX, and jQuery is that all of them are the advanced implementation of JavaScript. However, they serve completely different purposes.

Node.Js –

It is a server-side platform for developing client-server applications. For example, if we've to build an online employee management system, then we won't do it using client-side JS. But the Node.js can certainly do it as it runs on a server similar to Apache, Django not in a browser.

AJAX (Aka Asynchronous Javascript And XML) –

It is a client-side scripting technique, primarily designed for rendering the contents of a page without refreshing it. There are a no. of large companies utilizing AJAX such as Facebook and Stack Overflow to display dynamic content.

JQuery –

It is a famous JavaScript module which complements AJAX, DOM traversal, looping and so on. This library provides many useful functions to help in JavaScript development. However, it's not mandatory to use it but as it also manages cross-browser compatibility, so can help you produce highly maintainable web applications.

What Are Globals In Node.Js?

Answer.
There are three keywords in Node.js which constitute as Globals. These are Global, Process, and Buffer.

Global.

The Global keyword represents the global namespace object. It acts as a container for all other <global> objects. If we type <console.log(global)>, it'll print out all of them.

An important point to note about the global objects is that not all of them are in the global scope, some of them fall in the module scope. So, it's wise to declare them without using the var keyword or add them to Global object.

Variables declared using the var keyword become local to the module whereas those declared without it get subscribed to the global object.

Process.

It is also one of the global objects but includes additional functionality to turn a synchronous function into an async callback. There is no boundation to access it from anywhere in the code. It is the instance of the EventEmitter class. And each node application object is an instance of the Process object.

It primarily gives back the information about the application or the environment.

- **<process.execPath>** – to get the execution path of the Node app.
- **<process.Version> –** to get the Node version currently running.
- **<process.platform> –** to get the server platform.

Some of the other useful Process methods are as follows.

- **<process.memoryUsage> –** To know the memory used by Node application.
- **<process.NextTick> –** To attach a callback function that will get called during the next loop. It can cause a delay in executing a function.

Buffer.

The Buffer is a class in Node.js to handle binary data. It is similar to a list of integers but stores as a raw memory outside the V8 heap.

We can convert JavaScript string objects into Buffers. But it requires mentioning the encoding type explicitly.

- **<ascii> –** Specifies 7-bit ASCII data.

- **<utf8> –** Represents multibyte encoded Unicode char set.
- **<utf16le> –** Indicates 2 or 4 bytes, little endian encoded Unicode chars.
- **<base64> –** Used for Base64 string encoding.
- **<hex> –** Encodes each byte as two hexadecimal chars.

Here is the syntax to use the Buffer class.

```
> var buffer = new Buffer(string, [encoding]);
```

The above command will allocate a new buffer holding the string with <utf8> as the default encoding. However, if you like to write a <string> to an existing buffer object, then use the following line of code.

```
> buffer.write(string)
```

This class also offers other methods like <readInt8> and <writeUInt8> that allows read/write from various types of data to the buffer.

How To Load HTML In Node.Js?

Answer.

To load HTML in Node.js we have to change the “Content-type” in the HTML code from text/plain to text/html.

Let’s see an example where we have created a static file in web server.

```
fs.readFile(filename, "binary", function(err, file) {
if(err) {
response.writeHead(500, {"Content-Type": "text/plain"});
response.write(err + "\n");
response.end();
return;
}

response.writeHead(200);
response.write(file, "binary");
response.end();
});
```

Now we will modify this code to load an HTML page instead of plain text.

```
fs.readFile(filename, "binary", function(err, file) {
if(err) {
response.writeHead(500, {"Content-Type": "text/html"});
response.write(err + "\n");
response.end();
return;
}

response.writeHead(200, {"Content-Type": "text/html"});
response.write(file);
```

```
response.end();
});
```

Explain the concept of URL module.

The **URL module** of Node.js provides various utilities for **URL** resolution and parsing. It is a built-in module that helps in splitting up the web address into a readable format:

var url = require('url');
For example:

```
var url = require('url');
var adrs = 'http://localhost:8082/default.htm?year=2019&month=april';
var q = url.parse(adr, true);
console.log(q.host); //returns 'localhost:8082'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?year=2019 and month=april'
var qdata = q.query; //returns an object: { year: 2019, month: 'april' }
console.log(qdata.month); //returns 'april'
```

Explain the purpose of ExpressJS package?

Express.js is a framework built on top of Node.js that facilitates the management of the flow of data between server and routes in the server-side applications. It is a lightweight and flexible framework that provides a wide range of features required for the web as well as mobile application development. Express.js is developed on the middleware module of Node.js called ***connect***. The connect module further makes use of **http** module to communicate with Node.js. Thus, if you are working with any of the connect based middleware modules, then you can easily integrate with Express.js.

Explain the reason as to why Express 'app' and 'server' must be kept separate.

Express 'app' and 'server' must be kept separate as by doing this, you will be separating the API declaration from the network related configuration which benefits in the below listed ways:

- It allows testing the API in-process without having to perform the network calls
- Faster testing execution
- Getting wider coverage metrics of the code
- Allows deploying the same API under flexible and different network conditions
- Better separation of concerns and cleaner code

API declaration should reside in app.js:

```
var app = express();
app.use(bodyParser.json());
```

```
app.use("/api/events", events.API);
app.use("/api/forms", forms);
```

Server network declaration should reside in /bin/www:

```
var app = require('../app');
var http = require('http');
//Get port from environment and store in Express
var port = normalizePort(process.env.PORT || '8000');
app.set('port', port);
//Create HTTP server.
var server = http.createServer(app);
```

What do you mean by Express JS and what is its use?

Answer:
Express JS is an application framework that is light-weighted node JS. Variety of versatile, helpful and vital options are provided by this JavaScript framework for the event of mobile additionally as internet applications with the assistance of node JS. Express JS Use – Express.js could be a light-weight internet application that helps in organizing the net application into MVC design on the server aspect.

Write the steps for setting up an Express JS application?

Answer:
Following are the steps accustomed for An Express JS application: –

1. A folder with a constant name because the project name is made.
2. A file named package.json is made within the folder created.
3. "npm install" command is run on the electronic communication. It installs all the libraries gift in package.json.
4. A file named server.js is made.
5. "Router" file is made within the package that consists of a folder named index.js.
6. "App" is made within the package that has the index.html file.

What function are arguments available to Express JS route handlers?

Answer:The arguments which are available to an Express JS route handler-function are-

- Req – the request object
- Res – the response object
- Next (optional) – a function that is employed to pass management to 1 of the following route handlers.

The third argument is optional and should be omitted, however, in some cases, it's helpful wherever there's a series of handlers and management will be passed to 1 of the following route handlers skipping this one.

How to allow CORS in Express JS? Explain with an example?

Answer:
In order to permit CORS in Express.js, add the subsequent code in server.js:
For Example –

```
app.all('*', function(req, res, next) {
res.set('Access-Control-Allow-Origin', '*');
res.set('Access-Control-Allow-Methods', 'GET, POST, DELETE, PUT');
res.set('Access-Control-Allow-Headers', 'X-Requested-With, Content-Type');
if ('OPTIONS' == req.method) return res.send(200);
next();
});
```

What is the use of next in Express JS?

Answer:
Next -It passes management to a consecutive matching route. OR a operate to pass management to 1 of the following route handlers.
The argument could also be omitted, however, is beneficial in cases wherever you have got a series of handlers and you'd wish to pass management to 1 of the following route handlers, and skip this one.

```
app.get('/user details/:id?', function(req, res, next));
```

Req and Res – It represents the request and response objects
Next – It passes management to a consecutive matching route.

How to Redirect 404 Errors to A Page In ExpressJS?

Answer:
In server.js add the subsequent code to send 404 errors back to a page in our ExpressJS App:

```
/* Define fallback route */
app.use(function(req, res, next) {
res.status(404).json({errorCode: 404, errorMsg: "route not found"});
});
```

What is Angular Framework?

Angular is a TypeScript-based open-source front-end platform that makes it easy to build applications with in web/mobile/desktop. The major features of this framework such as declarative templates, dependency injection, end to end tooling, and many more other features are used to ease the development.

.What is the difference between AngularJS and Angular?

Angular is a completely revived component-based framework in which an application is a tree of individual components.

What is TypeScript?

TypeScript is a typed superset of JavaScript created by Microsoft that adds optional types, classes, async/await, and many other features, and compiles to plain JavaScript. Angular built entirely in TypeScript and used as a primary language. You can install it globally as

npm install -g typescript

Let's see a simple example of TypeScript usage,

```
function greeter(person: string) {
   return "Hello, " + person;
}
let user = "abc";

document.body.innerHTML = greeter(user);
```

The greeter method allows only string type as argument.

.

Write a pictorial diagram of Angular architecture?

The main building blocks of an Angular application is shown in the below diagram ScreenShot

.

What are the key components of Angular?

Angular has the below key components,

Component: These are the basic building blocks of angular application to control HTML views.

Modules: An angular module is set of angular basic building blocks like component, directives, services etc. An application is divided into logical pieces and each piece of code is called as "module" which perform a single task.

Templates: This represent the views of an Angular application.

Services: It is used to create components which can be shared across the entire application.

Metadata: This can be used to add more data to an Angular class.

.

What are directives?

Directives add behaviour to an existing DOM element or an existing component instance.

```
import { Directive, ElementRef, Input } from '@angular/core';

@Directive({ selector: '[myHighlight]' })
export class HighlightDirective {
   constructor(el: ElementRef) {
      el.nativeElement.style.backgroundColor = 'yellow';
   }
}
```

Now this directive extends HTML element behavior with a yellow background as below

```
<p myHighlight>Highlight me!</p>
```

.

What are components?

Components are the most basic UI building block of an Angular app which formed a tree of Angular components. These components are subset of directives. Unlike directives, components always have a template and only one component can be instantiated per an element in a template. Let's see a simple example of Angular component

```
import { Component } from '@angular/core';

@Component ({
  selector: 'my-app',
  template: ` <div>
```

```
    <h1>{{title}}</h1>
    <div>Learn Angular6 with examples</div>
  </div> `,
})

export class AppComponent {
  title: string = 'Welcome to Angular world';
}
```

.

What are the differences between Component and Directive?

In a short note, A component(@component) is a directive-with-a-template.

Some of the major differences are mentioned in a tabular form

Component

To register a component we use @Component meta-data annotation

Components are typically used to create UI widgets

Component is used to break up the application into smaller components

Only one component can be present per DOM element

@View decorator or templateurl/template are mandatory

Directive

To register directives we use @Directive meta-data annotation

Directive is used to add behavior to an existing DOM element

Directive is use to design re-usable components

Many directives can be used per DOM element

Directive doesn't use View

.

What is a template?

A template is a HTML view where you can display data by binding controls to properties of an Angular component. You can store your component's template in one of two places. You

can define it inline using the template property, or you can define the template in a separate HTML file and link to it in the component metadata using the @Component decorator's templateUrl property. Using inline template with template syntax,

```
import { Component } from '@angular/core';
@Component ({
  selector: 'my-app',
  template: '
    <div>
      <h1>{{title}}</h1>
      <div>Learn Angular</div>
    </div>
  '
})

export class AppComponent {
  title: string = 'Hello World';
}
```

Using separate template file such as app.component.html

```
import { Component } from '@angular/core';

@Component ({
  selector: 'my-app',
  templateUrl: 'app/app.component.html'
})

export class AppComponent {
  title: string = 'Hello World';
```

}

.

What is a module?

Modules are logical boundaries in your application and the application is divided into separate modules to separate the functionality of your application. Lets take an example of app.module.ts root module declared with @NgModule decorator as below,

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent }  from './app.component';

@NgModule ({
  imports:      [ BrowserModule ],
  declarations: [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

The NgModule decorator has three options

The imports option is used to import other dependent modules. The BrowserModule is required by default for any web based angular application

The declarations option is used to define components in the respective module

The bootstrap option tells Angular which Component to bootstrap in the application

.

What are lifecycle hooks available?

Angular application goes through an entire set of processes or has a lifecycle right from its initiation to the end of the application. The representation of lifecycle in pictorial representation as follows,

The description of each lifecycle method is as below,

ngOnChanges: When the value of a data bound property changes, then this method is called.

ngOnInit: This is called whenever the initialization of the directive/component after Angular first displays the data-bound properties happens.

ngDoCheck: This is for the detection and to act on changes that Angular can't or won't detect on its own.

ngAfterContentInit: This is called in response after Angular projects external content into the component's view.

ngAfterContentChecked: This is called in response after Angular checks the content projected into the component.

ngAfterViewInit: This is called in response after Angular initializes the component's views and child views.

ngAfterViewChecked: This is called in response after Angular checks the component's views and child views.

ngOnDestroy: This is the cleanup phase just before Angular destroys the directive/component.

.

What is a data binding?

Data binding is a core concept in Angular and allows to define communication between a component and the DOM, making it very easy to define interactive applications without worrying about pushing and pulling data. There are four forms of data binding(divided as 3 categories) which differ in the way the data is flowing.

From the Component to the DOM: Interpolation: {{ value }}: Adds the value of a property from the component

Name: {{ user.name }}

Address: {{ user.address }}

Property binding: [property]=”value”: The value is passed from the component to the specified property or simple HTML attribute

<input type="email" [value]="user.email">

From the DOM to the Component: Event binding: (event)=”function”: When a specific DOM event happens (eg.: click, change, keyup), call the specified method in the component

<button (click)="logout()"></button>

Two-way binding: Two-way data binding: [(ngModel)]=”value”: Two-way data binding allows to have the data flow both ways. For example, in the below code snippet, both the email DOM input and component email property are in sync

<input type="email" [(ngModel)]="user.email">

.

What is metadata?

Metadata is used to decorate a class so that it can configure the expected behavior of the class. The metadata is represented by decorators

Class decorators, e.g. @Component and @NgModule

```
import { NgModule, Component } from '@angular/core';


@Component({
  selector: 'my-component',
  template: '<div>Class decorator</div>',
})
export class MyComponent {
  constructor() {
    console.log('Hey I am a component!');
  }
}


@NgModule({
  imports: [],
  declarations: [],
})
export class MyModule {
  constructor() {
    console.log('Hey I am a module!');
  }
}
```

Property decorators Used for properties inside classes, e.g. @Input and @Output

```
import { Component, Input } from '@angular/core';


@Component({
    selector: 'my-component',
    template: '<div>Property decorator</div>'
```

```
})

export class MyComponent {
   @Input()
   title: string;
}
```

Method decorators Used for methods inside classes, e.g. @HostListener

```
import { Component, HostListener } from '@angular/core';

@Component({
   selector: 'my-component',
   template: '<div>Method decorator</div>'
})
export class MyComponent {
   @HostListener('click', ['$event'])
   onHostClick(event: Event) {
      // clicked, `event` available
   }
}
```

Parameter decorators Used for parameters inside class constructors, e.g. @Inject

```
import { Component, Inject } from '@angular/core';
import { MyService } from './my-service';

@Component({
   selector: 'my-component',
   template: '<div>Parameter decorator</div>'
})
export class MyComponent {
   constructor(@Inject(MyService) myService) {
      console.log(myService); // MyService
   }
```

}

.

What is angular CLI?

Angular CLI(Command Line Interface) is a command line interface to scaffold and build angular apps using nodejs style (commonJs) modules. You need to install using below npm command,

npm install @angular/cli@latest

Below are the list of few commands, which will come handy while creating angular projects

Creating New Project: ng new

Generating Components, Directives & Services: ng generate/g The different types of commands would be,

ng generate class my-new-class: add a class to your application

ng generate component my-new-component: add a component to your application

ng generate directive my-new-directive: add a directive to your application

ng generate enum my-new-enum: add an enum to your application

ng generate module my-new-module: add a module to your application

ng generate pipe my-new-pipe: add a pipe to your application

ng generate service my-new-service: add a service to your application

Running the Project: ng serve

.

What is the difference between constructor and ngOnInit?

TypeScript classes has a default method called constructor which is normally used for the initialization purpose. Whereas ngOnInit method is specific to Angular, especially used to define Angular bindings. Even though constructor getting called first, it is preferred to move all of your Angular bindings to ngOnInit method. In order to use ngOnInit, you need to implement OnInit interface as below,

```
export class App implements OnInit{
  constructor(){
    //called first time before the ngOnInit()
  }

  ngOnInit(){
    //called after the constructor and called  after the first ngOnChanges()
```

```
  }
}
```

.

What is a service?

A service is used when a common functionality needs to be provided to various modules. Services allow for greater separation of concerns for your application and better modularity by allowing you to extract common functionality out of components. Let's create a repoService which can be used across components,

```
import { Injectable } from '@angular/core';
import { Http } from '@angular/http';


@Injectable({ // The Injectable decorator is required for dependency injection to work
  // providedIn option registers the service with a specific NgModule
  providedIn: 'root',  // This declares the service with the root app (AppModule)
})
export class RepoService{
  constructor(private http: Http){
  }

  fetchAll(){
    return this.http.get('https://api.github.com/repositories');
  }
}
```

The above service uses Http service as a dependency.

.

What is dependency injection in Angular?

Dependency injection (DI), is an important application design pattern in which a class asks for dependencies from external sources rather than creating them itself. Angular comes with its own dependency injection framework for resolving dependencies(services or objects that a class needs to perform its function).So you can have your services depend on other services throughout your application.

.

What is the purpose of async pipe?

The AsyncPipe subscribes to an observable or promise and returns the latest value it has emitted. When a new value is emitted, the pipe marks the component to be checked for changes. Let's take a time observable which continuously updates the view for every 2 seconds with the current time.

```
@Component({
  selector: 'async-observable-pipe',
  template: `<div><code>observable|async</code>:
       Time: {{ time | async }}</div>`
})
export class AsyncObservablePipeComponent {
  time = new Observable(observer =>
    setInterval(() => observer.next(new Date().toString()), 2000)
  );
}
```

.

What is the option to choose between inline and external template file?

You can store your component's template in one of two places. You can define it inline using the template property, or you can define the template in a separate HTML file and link to it in the component metadata using the @Component decorator's templateUrl property. The choice between inline and separate HTML is a matter of taste, circumstances, and organization policy. But normally we use inline template for small portion of code and external template file for bigger views. By default, the Angular CLI generates components with a template file. But you can override that with the below command,

ng generate component hero -it

.

What is the purpose of ngFor directive?

We use Angular ngFor directive in the template to display each item in the list. For example, here we iterate over list of users,

```
<li *ngFor="let user of users">
  {{ user }}
</li>
```

The user variable in the ngFor double-quoted instruction is a template input variable

.

What is the purpose of ngIf directive?

Sometimes an app needs to display a view or a portion of a view only under specific circumstances. The Angular ngIf directive inserts or removes an element based on a truthy/falsy condition. Let's take an example to display a message if the user age is more than 18,

<p *ngIf="user.age > 18">You are not eligible for student pass!</p>

Note: Angular isn't showing and hiding the message. It is adding and removing the paragraph element from the DOM. That improves performance, especially in the larger projects with many data bindings.

.

How do you categorize data binding types?

Binding types can be grouped into three categories distinguished by the direction of data flow. They are listed as below,

From the source-to-view

From view-to-source

View-to-source-to-view

The possible binding syntax can be tabularized as below,

Data direction Syntax Type

From the source-to-view(One-way) 1. {{expression}} 2. [target]="expression" 3. bind-target="expression" Interpolation, Property, Attribute, Class, Style

From view-to-source(One-way) 1. (target)="statement" 2. on-target="statement" Event

View-to-source-to-view(Two-way) 1. [(target)]="expression" 2. bindon-target="expression" Two-way

.

What are pipes?

A pipe takes in data as input and transforms it to a desired output. For example, let us take a pipe to transform a component's birthday property into a human-friendly date using date pipe.

import { Component } from '@angular/core';

```
@Component({
  selector: 'app-birthday',
  template: `<p>Birthday is {{ birthday | date }}</p>`
})
export class BirthdayComponent {
  birthday = new Date(1987, 6, 18); // June 18, 1987
}
```

.

What is a bootstrapping module?

Every application has at least one Angular module, the root module that you bootstrap to launch the application is called as bootstrapping module. It is commonly known as AppModule. The default structure of AppModule generated by AngularCLI would be as follows,

```
/* JavaScript imports */
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';

import { AppComponent } from './app.component';

/* the AppModule class with the @NgModule decorator */
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule
  ],
  providers: [],
```

```
  bootstrap: [AppComponent]
})
export class AppModule { }
```

.

What are observables?

Observables are declarative which provide support for passing messages between publishers and subscribers in your application. They are mainly used for event handling, asynchronous programming, and handling multiple values. In this case, you define a function for publishing values, but it is not executed until a consumer subscribes to it. The subscribed consumer then receives notifications until the function completes, or until they unsubscribe.

.

What is HttpClient and its benefits?

Most of the Front-end applications communicate with backend services over HTTP protocol using either XMLHttpRequest interface or the fetch() API. Angular provides a simplified client HTTP API known as HttpClient which is based on top of XMLHttpRequest interface. This client is avaialble from @angular/common/http package. You can import in your root module as below,

```
import { HttpClientModule } from '@angular/common/http';
```

The major advantages of HttpClient can be listed as below,

Contains testability features

Provides typed request and response objects

Intercept request and response

Supports Observalbe APIs

Supports streamlined error handling

.

Explain on how to use HttpClient with an example?

Below are the steps need to be followed for the usage of HttpClient.

Import HttpClient into root module:

```
import { HttpClientModule } from '@angular/common/http';
@NgModule({
  imports: [
```

```
  BrowserModule,
  // import HttpClientModule after BrowserModule.
  HttpClientModule,
 ],
 .....
 })
export class AppModule {}
```

Inject the HttpClient into the application: Let's create a userProfileService(userprofile.service.ts) as an example. It also defines get method of HttpClient

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

const userProfileUrl: string = 'assets/data/profile.json';

@Injectable()
export class UserProfileService {
 constructor(private http: HttpClient) { }

 getUserProfile() {
  return this.http.get(this.userProfileUrl);
 }
}
```

Create a component for subscribing service: Let's create a component called UserProfileComponent(userprofile.component.ts) which inject UserProfileService and invokes the service method,

```
fetchUserProfile() {
 this.userProfileService.getUserProfile()
  .subscribe((data: User) => this.user = {
     id: data['userId'],
     name: data['firstName'],
     city:  data['city']
```

```
  });
}
```

Since the above service method returns an Observable which needs to be subscribed in the component.

.

How can you read full response?

The response body doesn't may not return full response data because sometimes servers also return special headers or status code which which are important for the application workflow. Inorder to get full response, you should use observe option from HttpClient,

```
getUserResponse(): Observable<HttpResponse<User>> {
 return this.http.get<User>(
  this.userUrl, { observe: 'response' });
}
```

Now HttpClient.get() method returns an Observable of typed HttpResponse rather than just the JSON data.

.

How do you perform Error handling?

If the request fails on the server or failed to reach the server due to network issues then HttpClient will return an error object instead of a successful reponse. In this case, you need to handle in the component by passing error object as a second callback to subscribe() method. Let's see how it can be handled in the component with an example,

```
fetchUser() {
 this.userService.getProfile()
  .subscribe(
   (data: User) => this.userProfile = { ...data }, // success path
   error => this.error = error // error path
  );
}
```

It is always a good idea to give the user some meaningful feedback instead of displaying the raw error object returned from HttpClient.

.

What is RxJS?

RxJS is a library for composing asynchronous and callback-based code in a functional, reactive style using Observables. Many APIs such as HttpClient produce and consume RxJS

Observables and also uses operators for processing observables. For example, you can import observables and operators for using HttpClient as below,

```
import { Observable, throwError } from 'rxjs';

import { catchError, retry } from 'rxjs/operators';
```

.

What is subscribing?

An Observable instance begins publishing values only when someone subscribes to it. So you need to subscribe by calling the subscribe() method of the instance, passing an observer object to receive the notifications. Let's take an example of creating and subscribing to a simple observable, with an observer that logs the received message to the console.

Creates an observable sequence of 5 integers, starting from 1

```
const source = range(1, 5);


// Create observer object

const myObserver = {

  next: x => console.log('Observer got a next value: ' + x),

  error: err => console.error('Observer got an error: ' + err),

  complete: () => console.log('Observer got a complete notification'),

};


// Execute with the observer object and Prints out each item

myObservable.subscribe(myObserver);

// => Observer got a next value: 1

// => Observer got a next value: 2

// => Observer got a next value: 3

// => Observer got a next value: 4

// => Observer got a next value: 5

// => Observer got a complete notification
```

.

What is an observable?

An Observable is a unique Object similar to a Promise that can help manage async code. Observables are not part of the JavaScript language so we need to rely on a popular

Observable library called RxJS. The observables are created using new keyword. Let see the simple example of observable,

```
import { Observable } from 'rxjs';


const observable = new Observable(observer => {
  setTimeout(() => {
    observer.next('Hello from a Observable!');
  }, 2000);
});
```

.

What is an observer?

Observer is an interface for a consumer of push-based notifications delivered by an Observable. It has below structure,

```
interface Observer<T> {
  closed?: boolean;
  next: (value: T) => void;
  error: (err: any) => void;
  complete: () => void;
}
```

A handler that implements the Observer interface for receiving observable notifications will be passed as a parameter for observable as below,

```
myObservable.subscribe(myObserver);
```

Note: If you don't supply a handler for a notification type, the observer ignores notifications of that type.

.

What is the difference between promise and observable?

Below are the list of differences between promise and observable,

Observable Promise

Declarative: Computation does not start until subscription so that they can be run whenever you need the result Execute immediately on creation

Provide multiple values over time Provide only one

Subscribe method is used for error handling which makes centralized and predictable error handling Push errors to the child promises

Provides chaining and subscription to handle complex applications Uses only .then() clause

.

How do you perform error handling in observables?

You can handle errors by specifying an error callback on the observer instead of relying on try/catch which are ineffective in asynchronous environment. For example, you can define error callback as below,

```
myObservable.subscribe({
  next(num) { console.log('Next num: ' + num)},
  error(err) { console.log('Received an errror: ' + err)}
});
```

.

What is the short hand notation for subscribe method?

The subscribe() method can accept callback function definitions in line, for next, error, and complete handlers is known as short hand notation or Subscribe method with positional arguments. For example, you can define subscribe method as below,

```
myObservable.subscribe(
  x => console.log('Observer got a next value: ' + x),
  err => console.error('Observer got an error: ' + err),
  () => console.log('Observer got a complete notification')
);
```

.

What are dynamic components?

Dynamic components are the components in which components location in the application is not defined at build time.i.e, They are not used in any angular template. But the component is instantiated and placed in the application at runtime.

.

What are the various kinds of directives?

There are mainly three kinds of directives.

Components — These are directives with a template.

Structural directives — These directives change the DOM layout by adding and removing DOM elements.

Attribute directives — These directives change the appearance or behavior of an element, component, or another directive.

.

What is Angular Router?

Angular Router is a mechanism in which navigation happens from one view to the next as users perform application tasks. It borrows the concepts or model of browser's application navigation.

.

What is the purpose of base href tag?

The routing application should add element to the index.html as the first child in the tag inorder to indicate how to compose navigation URLs. If app folder is the application root then you can set the href value as below

<base href="/">

.

What are the router imports?

The Angular Router which represents a particular component view for a given URL is not part of Angular Core. It is available in library named @angular/router to import required router components. For example, we import them in app module as below,

import { RouterModule, Routes } from '@angular/router';

.

What is router outlet?

The RouterOutlet is a directive from the router library and it acts as a placeholder that marks the spot in the template where the router should display the components for that outlet. Router outlet is used like a component,

<router-outlet></router-outlet>

<!-- Routed components go here -->

.

What are router links?

The RouterLink is a directive on the anchor tags give the router control over those elements. Since the navigation paths are fixed, you can assign string values to router-link directive as below,

```
<h1>Angular Router</h1>
<nav>
  <a routerLink="/todosList" >List of todos</a>
  <a routerLink="/completed" >Completed todos</a>
</nav>
<router-outlet></router-outlet>
```

.

What are active router links?

RouterLinkActive is a directive that toggles css classes for active RouterLink bindings based on the current RouterState. i.e, the Router will add CSS classes when this link is active and and remove when the link is inactive. For example, you can add them to RouterLinks as below

```
<h1>Angular Router</h1>
<nav>
 <a routerLink="/todosList" routerLinkActive="active">List of todos</a>
 <a routerLink="/completed" routerLinkActive="active">Completed todos</a>
</nav>
<router-outlet></router-outlet>
```

.

What are router events?

During each navigation, the Router emits navigation events through the Router.events property allowing you to track the lifecycle of the route. The sequence of router events is as below,

NavigationStart,

RouteConfigLoadStart,

RouteConfigLoadEnd,

RoutesRecognized,

GuardsCheckStart,

ChildActivationStart,

ActivationStart,

GuardsCheckEnd,

ResolveStart,

ResolveEnd,

ActivationEnd

ChildActivationEnd

NavigationEnd,

NavigationCancel,

NavigationError

Scroll

.

What is activated route?

ActivatedRoute contains the information about a route associated with a component loaded in an outlet. It can also be used to traverse the router state tree. The ActivatedRoute will be injected as a router service to access the information. In the below example, you can access route path and parameters,

```
@Component({...})
class MyComponent {
  constructor(route: ActivatedRoute) {
    const id: Observable<string> = route.params.pipe(map(p => p.id));
    const url: Observable<string> = route.url.pipe(map(segments => segments.join('')));
    // route.data includes both `data` and `resolve`
    const user = route.data.pipe(map(d => d.user));
  }
}
```

.

How do you define routes?

A router must be configured with a list of route definitions. You configures the router with routes via the RouterModule.forRoot() method, and adds the result to the AppModule's imports array.

```
const appRoutes: Routes = [
```

```
 { path: 'todo/:id',      component: TodoDetailComponent },
 {
   path: 'todos',
   component: TodosListComponent,
   data: { title: 'Todos List' }
 },
 { path: '',
   redirectTo: '/todos',
   pathMatch: 'full'
 },
 { path: '**', component: PageNotFoundComponent }
];

@NgModule({
 imports: [
   RouterModule.forRoot(
     appRoutes,
     { enableTracing: true } // <-- debugging purposes only
   )
   // other imports here
 ],
 ...
})
export class AppModule { }
```

.

Do I need a Routing Module always?

No, the Routing Module is a design choice. You can skip routing Module (for example, AppRoutingModule) when the configuration is simple and merge the routing configuration directly into the companion module (for example, AppModule). But it is recommended when the configuration is complex and includes specialized guard and resolver services.

.

What are different types of compilation in Angular?

Angular offers two ways to compile your application,

Just-in-Time (JIT)

Ahead-of-Time (AOT)

.

What is JIT?

Just-in-Time (JIT) is a type of compilation that compiles your app in the browser at runtime. JIT compilation is the default when you run the ng build (build only) or ng serve (build and serve locally) CLI commands. i.e, the below commands used for JIT compilation,

ng build

ng serve

.

Why do we need compilation process?

The Angular components and templates cannot be understood by the browser directly. Due to that Angular applications require a compilation process before they can run in a browser. For example, In AOT compilation, both Angular HTML and TypeScript code converted into efficient JavaScript code during the build phase before browser runs it.

.

What are the restrictions of metadata?

In Angular, You must write metadata with the following general constraints,

Write expression syntax with in the supported range of javascript features

The compiler can only reference symbols which are exported

Only call the functions supported by the compiler

Decorated and data-bound class members must be public.

.

How do you describe various dependencies in angular application?

The dependencies section of package.json with in an angular application can be divided as follow,

Angular packages: Angular core and optional modules; their package names begin @angular/.

Support packages: Third-party libraries that must be present for Angular apps to run.

Polyfill packages: Polyfills plug gaps in a browser's JavaScript implementation.

.

What is the purpose of common module?

The commonly-needed services, pipes, and directives provided by @angular/common module. Apart from these HttpClientModule is available under @angular/common/http.

.

What is Style function?

The style function is used to define a set of styles to associate with a given state name. You need to use it along with state() function to set CSS style attributes. For example, in the close state, the button has a height of 100 pixels, an opacity of 0.8, and a background color of green.

```
state('close', style({
  height: '100px',
  opacity: 0.8,
  backgroundColor: 'green'
})),
```

Note: The style attributes must be in camelCase.

.

What are the case types in Angular?

Angular uses capitalization conventions to distinguish the names of various types. Angular follows the list of the below case types.

camelCase : Symbols, properties, methods, pipe names, non-component directive selectors, constants uses lowercase on the first letter of the item. For example, "selectedUser"

UpperCamelCase (or PascalCase): Class names, including classes that define components, interfaces, NgModules, directives, and pipes uses uppercase on the first letter of the item.

dash-case (or "kebab-case"): The descriptive part of file names, component selectors uses dashes between the words. For example, "app-user-list".

UPPER_UNDERSCORE_CASE: All constants uses capital letters connected with underscores. For example, "NUMBER_OF_USERS".

.

What are the class decorators in Angular?

A class decorator is a decorator that appears immediately before a class definition, which declares the class to be of the given type, and provides metadata suitable to the type The following list of decorators comes under class decorators,

@Component()

@Directive()

@Pipe()

@Injectable()

@NgModule()

.

What are class field decorators?

The class field decorators are the statements declared immediately before a field in a class definition that defines the type of that field. Some of the examples are: @input and @output,

@Input() myProperty;

@Output() myEvent = new EventEmitter();

.

What is declarable in Angular?

Declarable is a class type that you can add to the declarations list of an NgModule. The class types such as components, directives, and pipes comes can be declared in the module.

.

What are the restrictions on declarable classes?

Below classes shouldn't be declared,

A class that's already declared in another NgModule

Ngmodule classes

Service classes

Helper classes

.

What is platform in Angular?

A platform is the context in which an Angular application runs. The most common platform for Angular applications is a web browser, but it can also be an operating system for a mobile device, or a web server. The runtime-platform is provided by the @angular/platform-* packages and these packages allow applications that make use of

@angular/core and @angular/common to execute in different environments. i.e, Angular can be used as platform-independent framework in different environments, For example,

While running in the browser, it uses platform-browser package.

When SSR(server-side rendering) is used, it uses platform-server package for providing web server implementation.

.

What happens if I import the same module twice?

If multiple modules imports the same module then angular evaluates it only once (When it encounters the module first time). It follows this condition even the module appears at any level in a hierarchy of imported NgModules.

.

How do you pass headers for HTTP client?

You can directly pass object map for http client or create HttpHeaders class to supply the headers.

```
constructor(private _http: HttpClient) {}

this._http.get('someUrl',{
  headers: {'header1':'value1','header2':'value2'}
});
```

(or)

```
let headers = new HttpHeaders().set('header1', headerValue1); // create header object

headers = headers.append('header2', headerValue2); // add a new header, creating a new object

headers = headers.append('header3', headerValue3); // add another header

let params = new HttpParams().set('param1', value1); // create params object

params = params.append('param2', value2); // add a new param, creating a new object

params = params.append('param3', value3); // add another param

return this._http.get<any[]>('someUrl', { headers: headers, params: params })
```

Is Angular supports dynamic imports?

Yes, Angular 8 supports dynamic imports in router configuration. i.e, You can use the import statement for lazy loading the module using loadChildren method and it will be understood by the IDEs(VSCode and WebStorm), webpack, etc. Previously, you have been written as below to lazily load the feature module. By mistake, if you have typo in the module name it still accepts the string and throws an error during build time.

{path: ‘user’, loadChildren: ‘./users/user.module#UserModulee’},

This problem is resolved by using dynamic imports and IDEs are able to find it during compile time itself.

{path: ‘user’, loadChildren: () => import(‘./users/user.module’).then(m => m.UserModule)};

.

What is lazy loading?

Lazy loading is one of the most useful concepts of Angular Routing. It helps us to download the web pages in chunks instead of downloading everything in a big bundle. It is used for lazy loading by asynchronously loading the feature module for routing whenever required using the property loadChildren. Let's load both Customer and Order feature modules lazily as below,

```
const routes: Routes = [
 {
   path: 'customers',
   loadChildren: () => import('./customers/customers.module').then(module =>
module.CustomersModule)
 },
 {
   path: 'orders',
   loadChildren: () => import('./orders/orders.module').then(module =>
module.OrdersModule)
 },
 {
   path: '',
   redirectTo: '',
```

```
   pathMatch: 'full'
  }
];
```

.

How do you upgrade angular version?

The Angular upgrade is quite easier using Angular CLI ng update command as mentioned below. For example, if you upgrade from Angular 7 to 8 then your lazy loaded route imports will be migrated to the new import syntax automatically.

```
$ ng update @angular/cli @angular/core
```

.

.

What is NgUpgrade?

NgUpgrade is a library put together by the Angular team, which you can use in your applications to mix and match AngularJS and Angular components and bridge the AngularJS and Angular dependency injection systems.

How do you test Angular application using CLI?

Angular CLI downloads and install everything needed with the Jasmine Test framework. You just need to run ng test to see the test results. By default this command builds the app in watch mode, and launches the Karma test runner. The output of test results would be as below,

```
10% building modules 1/1 modules 0 active
...INFO [karma]: Karma v1.7.1 server started at http://0.0.0.0:9876/
...INFO [launcher]: Launching browser Chrome ...
...INFO [launcher]: Starting browser Chrome
...INFO [Chrome ...]: Connected on socket ...
Chrome ...: Executed 3 of 3 SUCCESS (0.135 secs / 0.205 secs)
```

Note: A chrome browser also opens and displays the test output in the "Jasmine HTML Reporter".

How to use polyfills in Angular application?

The Angular CLI provides support for polyfills officially. When you create a new project with the ng new command, a src/polyfills.ts configuration file is created as part of your project folder. This file includes the mandatory and many of the optional polyfills as JavaScript import statements. Let's categorize the polyfills,

Mandatory polyfills: These are installed automatically when you create your project with ng new command and the respective import statements enabled in 'src/polyfills.ts' file.

Optional polyfills: You need to install its npm package and then create import statement in 'src/polyfills.ts' file. For example, first you need to install below npm package for adding web animations (optional) polyfill.

npm install --save web-animations-js

and create import statement in polyfill file.

import 'web-animations-js';

.

What are the differences of various versions of Angular?

There are different versions of Angular framework. Let's see the features of all the various versions,

1. Angular 1
   - Angular 1 (AngularJS) is the first angular framework released in the year 2010.
   - AngularJS is not built for mobile devices.
   - It is based on controllers with MVC architecture.
2. Angular 2
   - Angular 2 was released in the year 2016. Angular 2 is a complete rewrite of Angular1 version.
   - The performance issues that Angular 1 version had has been addressed in Angular 2 version.
   - Angular 2 is built from scratch for mobile devices unlike Angular 1 version.
   - Angular 2 is components based.
3. Angular 3

   The following are the different package versions in Angular 2.

- @angular/core v2.3.0
- @angular/compiler v2.3.0
- @angular/http v2.3.0
- @angular/router v3.3.0

The router package is already versioned 3 so to avoid confusion switched to Angular 4 version and skipped 3 version.

4. Angular 4

- The compiler generated code file size in AOT mode is very much reduced.
- With Angular 4 the production bundles size is reduced by hundreds of KB's.
- Animation features are removed from angular/core and formed as a separate package.
- Supports Typescript 2.1 and 2.2.

5. Angular 5

- Angular 5 makes angular faster. It improved the loading time and execution time.
- Shipped with new build optimizer.
- Supports Typescript 2.5.

6. Angular 6

- It is released in May 2018.
- Includes Angular Command Line Interface (CLI), Component Development KIT (CDK), Angular Material Package.

7. Angular 7

- It is released in October 2018.
- TypeScript 3.1
- RxJS 6.3
- New Angular CLI
- CLI Prompts capability provide an ability to ask questions to the user before they run. It is like interactive dialog between the user and the CLI
- With the improved CLI Prompts capability, it helps developers to make the decision. New ng commands ask users for routing and CSS styles types(SCSS) and ng add @angular/material asks for themes and gestures or animations.

What are the security principles in angular?

You should avoid direct use of the DOM APIs.

You should enable Content Security Policy (CSP) and configure your web server to return appropriate CSP HTTP headers.

You should Use the offline template compiler.

You should Use Server Side XSS protection.

You should Use DOM Sanitizer.

You should Preventing CSRF or XSRF attacks.

.

How do you find angular CLI version?

Angular CLI provides it's installed version using below different ways using ng command

ng v

ng version

ng -v

ng --version

and the output would be as below,

Angular CLI: 1.6.3

Node: 8.11.3

OS: darwin x64

Angular:

...

.

What is the browser support for Angular?

Angular supports most recent browsers which includes both desktop and mobile browsers.

Browser	Version
Chrome	latest
Firefox	latest
Edge	2 most recent major versions
IE	11, 10, 9 (Compatibility mode is not supported)
Safari	2 most recent major versions

IE Mobile 11

iOS 2 most recent major versions

Android 7.0, 6.0, 5.0, 5.1, 4.4