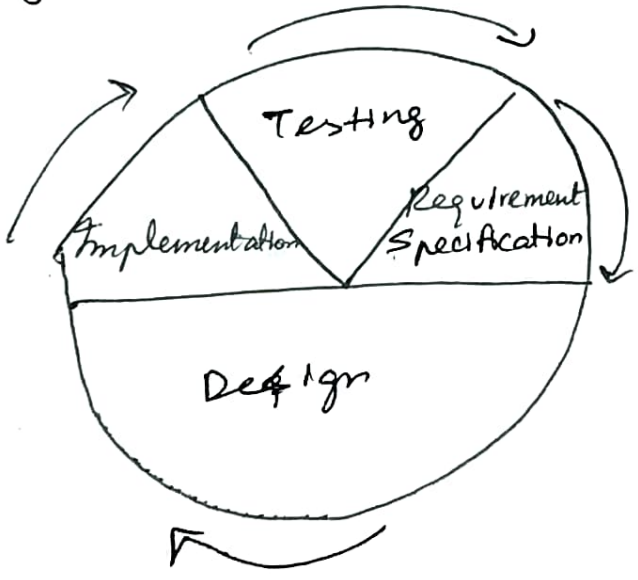
Part B

1.

Abstract Class	Interface
→ Abstract key word is used to declare abstract class	→ Interface keyword. is used to declare interface.
→ Doesn't support multiple inheritance	→ Supports multiple inheritance.
→ Can provide implementation of interface	→ Can't provide implementation of abstract class.
→ Contains both incomplete & complete member	→ Contains only incomplete member.

3. object oriented software Development life cycle model.

=>



Requirement specification

=> Description of function of software & expected performance.

Design

=> Mechanism to transform user requirements into suitable form, which help the programmer in software coding and implementation.

Implementation

=> Transformation of the software technical data package into one or more configuration items that are ready for software acceptance testing.

Testing

=> Method to check whether the actual software product matches expected requirements and to ensure that software product is defect free.

5.

Pop

→ Divided into functions
→ Top down approach
→ Less secure
→ Deals with algorithms

ex: C

OOP

Divided into objects
Bottom up approach.
more secure.
Deals with data

ex: C++.

⑦

Visibility	Public	Private
→ class in same package.	✓	✗
→ subclass in same package.	✓	✗
→ subclass outside same package	✓	✗
→ Non-subclass outside same package	✓	✗.

⑧ Static keyword ⇒ used to give special characteristics to an element
- They have scope till end of program.

① Static variables → Variables in a function, class

② Static Members of class → class objects and function in a class.

ex: static int x;

① syntax :- static datatype variable = value;

② syntax :- static return_type functionname { }.

10. Type conversion example program.

=>
```
#include <iostream>
using namespace std;
int main()
{ double x = 1.2;
  int sum = (int)x + 1;
  cout << sum;
  return 0; }.
```

→ explicit conversion from ~~float to~~ double to int.

14. Example for pointer & explain:-

=>
```
#include <iostream>
using namespace std;
int main()
{ int v[] = {10};
    int *ptr;
    ptr = v;
    cout << *ptr; }.
```

output will be => 10.

pointer => Symbolic representation of addresses.
=> call-by-reference.
=> create/manipulate dynamic data structures.

syntax :- datatype *var_name;

15. Fibonacci series.

```
#include <iostream.h>
using namespace std;
int main()
{ int n1 = 0, n2 = 1, n3, i, no;
  cout << "no. of elements";
  cin >> no;
  cout << n1 << n2;
  for(i = 2; i < no; ++i)
    { n3 = n2 + n1
      cout << n3;
      n1 = n2;
      n2 = n3; }.
      return 0; }
```

19)

Include

→ Base use case is incomplete

→ It is mandatory

→ No explicit inclusion location

→ No explicit inclusion condition

ex: Bank ATM transaction

<<include>>

Customer Authentication

Extend

→ Base use case is complete

→ Not mandatory

→ Has explicit extension location

→ Has/may have explicit extension condition.

ex: Bank ATM transaction

<<extend>>

Help.

23) Objects passed as function parameters :-

⇒ Write object name as argument while calling the function.

syntax :- function_name(object_name).

ex

```
#include <iostream>
using namespace std;
class Example
{ public : int a;
void add(Example E)
{a = a + E.a;}
};
```