Disclaimer!

The solution provided is intended to be a reference material for students to aid in their understanding of the subject matter. However, it is not a substitute for studying textbooks, applying their own logic and ideas, and engaging in critical thinking. The solution is meant to provide additional insights and perspectives to help students grasp the concepts and principles covered in the material. It should not be solely relied upon as the sole source of information, and students should not depend on it entirely for their learning. The solution may not be applicable to all situations, and there may be alternative solutions and viewpoints that are equally valid. Therefore, students are encouraged to consult with their instructor, textbooks, and other resources if they have any doubts or questions regarding the solution.

This pdf includes

1. Sequence diagram and sequence diagram of ATM system.
2. Draw a CRC cards for Library System
3. Create classes called Vechile1, Vechile2 and Vechile3 with each of having one private member named price. Add member function to set a price(say setPrice()) one each class. Add one more function max() that is friendly to all classes. max() function should compare private member named price of three classes and show maximum among them. Create one-one object of each class and then set a value on them. Display the maximum price among them.
4. Create a class named Height with data members feet and inches. In main function create two objects of class Height. Initialize one object using parametrized constructor and copy this values to another object. Now finally perform the addition between these two objects
5. Define a class to represent a bank account .Include the following
   Data members
   - Name of the depositor
   - Account number
   - Type of account
   - Balance amount in the account

   Member functions
   - To assign Initial values
   - To deposit an amount
   - To withdraw an amount after checking the balance
   - To display name and balance

   Write main program to test the program

6. Create a class Employee with data members (id, name, Post, address, salary) and read information of 20 Employees and display the name and post of employee whose salary is greater than 50000.
7. Using class write a program that receives inputs principle amount, time and rate. Keeping rate 12% as the default argument , calculate the simple interest for five customers.
8. WAP to input marks of 35 students and find the highest marks using Dynamic memory allocation.
9. Create a class student with two data members represent name and roll. Use appropriate member function to read and print these data members name and roll. Derive a class marks from student that has additional data member sessional1, sessional2 to store sessional marks. Derive another class result from marks and add the sessional marks. Use appropriate member function to read and display data in the class.
10. Create a class student with two data members to represent name and age. Use member function to read and print those data. From this derive a class called boarder with member data to represent room number .Derive another class called day-scholar from class student with member data to represent address of student. In both derived class use function to read and print respective data.
11. Define two classes named 'polar' and 'rectangle' to represent points in polar and rectangle systems. Use conversion routines to convert from one system to another system using template
12. Write a program showing ++ and - - operator overloading.
13. How operator overloading support polymorphism? Explain it by overloading '+' Operator to concatenate two strings.
14. We have a class name complex which have a data member real and imaginary .Default and parameterized constructor are there to initialize data member .WAP to overloading unary operator '++' where real and imaginary data member will be incremented and '+' operator to add two complex number.
15. Can you have more than one constructor in a program? Write a program to find area of triangle (when sides are given) using the concept of overloaded constructor.
16. Why does this pointer is widely used than object pointer? Write a program to implement pure polymorphism.
17. In your opinion how does the object oriented thinking solve the software crisis? List the basic characters of object oriented programming and explain any two characteristics that are directly applicable for software maintenance.
18. Constructor is called in derived class but it can't be inherited. Support your answer with suitable example.
19. Explain Responsibility implies non-interface. Explain with example.
20. What is behavior in OOP?
21. What is the role of behavior in OOP?
22. What is Responsibility-Driven Design (RDD)?

23. How are object oriented Programs are designed and developed according to the concept of RDD? Describe entire process in brief.
24. Cohesion and coupling.

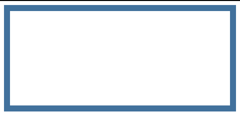
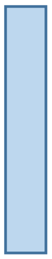
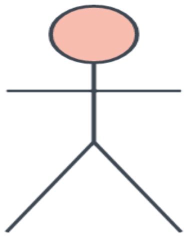
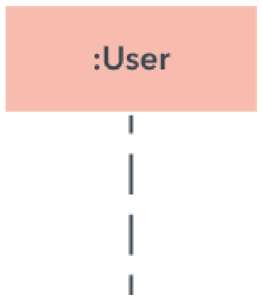
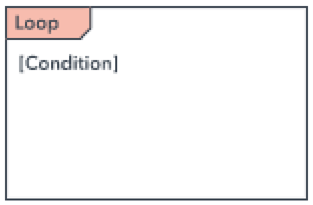
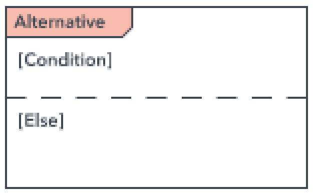
1) Sequence diagram and sequence diagram of ATM system

<u>Sequence diagram</u>

A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process.

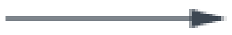
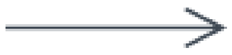
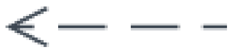
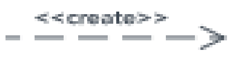
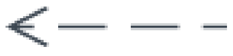
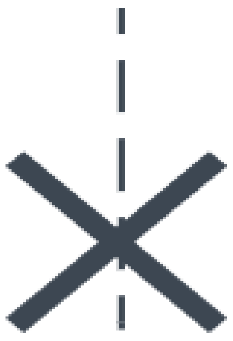
Basic symbols and components

To understand what a sequence diagram is, you should be familiar with its symbols and components. Sequence diagrams are made up of the following icons and elements:

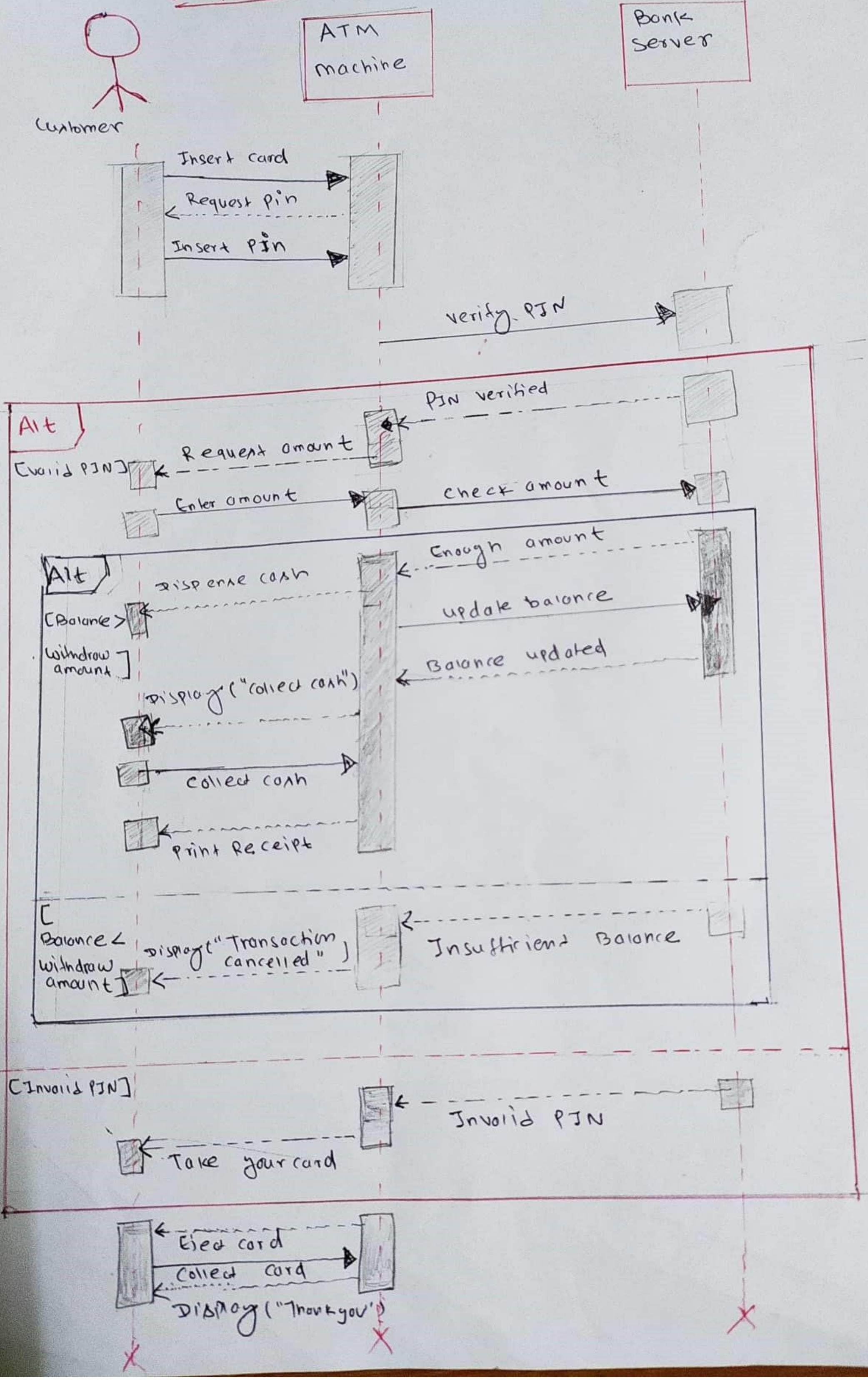
Symbol	Name	Description
	Object symbol	Represents a class or object in UML. The object symbol demonstrates how an object will behave in the context of the system.
	Activation box	Represents the time needed for an object to complete a task. The longer the task will take, the longer the activation box becomes.
	Actor symbol	Shows entities that interact with or are external to the system.
	Lifeline symbol	Represents the passage of time as it extends downward. This dashed vertical line shows the sequential events that occur to an object during the charted process. Lifelines may begin with a labeled rectangle shape or an actor symbol.
	Option loop symbol	Used to model if/then scenarios, i.e., a circumstance that will only occur under certain conditions.
	Alternative symbol	Symbolizes a choice (that is usually mutually exclusive) between two or more message sequences. To represent alternatives, use the labeled rectangle shape with a dashed line inside.

Common message symbols

Use the following arrows and message symbols to show how information is transmitted between objects. These symbols may reflect the start and execution of an operation or the sending and reception of a signal.

Symbol	Name	Description
	Synchronous message symbol	Represented by a solid line with a solid arrowhead. This symbol is used when a sender must wait for a response to a message before it continues. The diagram should show both the call and the reply.
	Asynchronous message symbol	Represented by a solid line with a lined arrowhead. Asynchronous messages don't require a response before the sender continues. Only the call should be included in the diagram.
	Asynchronous return message symbol	Represented by a dashed line with a lined arrowhead.
	Asynchronous create message symbol	Represented by a dashed line with a lined arrowhead. This message creates a new object.
	Reply message symbol	Represented by a dashed line with a lined arrowhead, these messages are replies to calls.
	Delete message symbol	Represented by a solid line with a solid arrowhead, followed by an X. This message destroys an object.

Sequence diagram for ATM System



2) Draw a CRC cards for Library management system.

(Note:This solution is minor modification on lecture notes)

Librarian	
Know all books Know all borrowers Search for borrower Check in book Check out book	Book Borrower

Book	
Know its title Know its author Know its Registration date Knows Borrower Knows due date Knows if late Know in or out	Date Borrower

Borrower	
Knows name Knows contact no. Knows set of books can borrow books	Book

Date	
Knows current date Can compute new date Compare two dates	

3) **Create classes called Vechile1, Vechile2 and Vechile3 with each of having one private member named price. Add member function to set a price(say setPrice()) one each class. Add one more function max() that is friendly to all classes. max() function should compare private member named price of three classes and show maximum among them. Create one-one object of each class and then set a value on them. Display the maximum price among them.**

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

class Vechile2;
class Vechile3;

class Vechile1
{
     private:
     long int price;public:
      void setPrice(long int p)
     {
      price=p;
     }
    friend void max(Vechile1,Vechile2,Vechile3);
};
class Vechile2
{
    private:
   long int price;
   public:
   void setPrice(long int p)
   {
     price=p;
    }
    friend void max(Vechile1,Vechile2,Vechile3);
};
```

```
class Vechile3
{
private:
long int price;
public:
void setPrice(long int p)
{
price=p;
}
friend void max(Vechile1,Vechile2,Vechile3);
};

void max(Vechile1 v1,Vechile2 v2,Vechile3 v3)
{
	if(v1.price>v2.price&&v1.price>v3.price)
	{
	cout<<"Maximum price="<<v1.price<<endl;
	}
	else if(v2.price>v3.price)
	{
	cout<<"Maximum price="<<v2.price<<endl;
	}
	else
	{
	cout<<"Maximum price="<<v3.price<<endl;
	}
}
int main()
{
Vechile1 v1;
Vechile2 v2;
Vechile3 v3;
v1.setPrice(5032323);
v2.setPrice(4543043);
v3.setPrice(1232343);
max(v1,v2,v3);
return 0;
}
```

4) Create a class named Height with data members feet and inches. In main function create two objects of class Height. Initialize one object using parametrized constructor and copy this values to another object. Now finally perform the addition between these two objects.

```
#include<iostream>
using namespace std;
class Height
{
        private:
        int feet;
        int inches;
        public:
        Height()
        { }
        Height(int f,float i)
        {
        feet=f;
        inches=i;
        }
        Height(Height &h)
        {
        feet=h.feet;
        inches=h.inches;
        }
        void add(Height h1,Height h2)
        {
        inches=h1.inches+h2.inches;
        feet=inches/12;
        inches=inches%12;
        feet=feet+h1.feet+h2.feet;
        }
        void display()
        {
        cout<<feet<<"feet and"<<inches<<"inches"<<endl;
        }
};
```

```
int main()
{
Height h1(5,7);
Height h2(h1);
Height h3;
h3.add(h1,h2);
cout<<"sum of height=";
h3.display();
return 0;
}
```

5) Define a class to represent a bank account .Include the following
Data members

- **Name of the depositor**
- **Account number**
- **Type of account**
- **Balance amount in the account**

Member functions

- **To assign Initial values**
- **To deposit an amount**
- **To withdraw an amount after checking the balance**
- **To display name and balance**

Write main program to test the program

```
#include<iostream>
#include<string.h>
using namespace std;
class Bank
{
 char name[20];
long int acno;
char acctype[20];
float bal;
public:
Bank(long int ano, char n[], char at[], float b) //Parameterized Constructor
{
acno=ano;
strcpy(name,n);
strcpy(acctype, at);
bal=b;
}
void deposit();
void withdraw();
void display();
};
void Bank::deposit()
{
int damt;
cout<<"Enter Deposit Amount"<<endl;
cin>>damt;
bal=bal+damt;
}
void Bank::withdraw()
{
int wamt;
cout<<"Enter Withdraw Amount"<<endl;
cin>>wamt;
if(wamt>bal)
{
cout<<"Cannot Withdraw Amount";
}
else
{
bal=bal-wamt;
}
}
void Bank::display()
{
cout<<"Account Number="<<acno<<endl;
cout<<"Name="<<name<<endl;
cout<<"Account Type:"<<acctype<<endl;
cout<<"Balance ="<<bal<<endl;
}
```

```
int main()
{
int acc_no;char name[100], acc_type[100];
float balance;
cout<<"Enter Details:"<<endl;
cout<<"Enter Account Number"<<endl;
cin>>acc_no;
cout<<"Enter Name"<<endl;
cin>>name;
cout<<"Enter Account Type"<<endl;
cin>>acc_type;
cout<<"Enter Balance"<<endl;
cin>>balance;
Bank b1(acc_no,name,acc_type,balance);
b1.deposit();
b1.withdraw();
b1.display();
return 0;
}
```

6) **Create a class Employee with data members (id, name, Post, address, salary) and read information of 20 Employees and display the name and post of employee whose salary is greater than 50000.**

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

class Employee
{
private:
int eid;
char name[20];
char post[20];
char address[20];
float salary;
public:
void getdata()
{
cout<<"Enter the Employee id"<<endl;
cin>>eid;
cout<<"Enter the Employee name"<<endl;
cin>>name;
cout<<"Enter the post "<<endl;
cin>>post;
cout<<"Enter the address"<<endl;
cin>>address;
cout<<"Enter the salary"<<endl;
cin>>salary;
}
void display()
{
	if(salary>50000)
	{
	cout<<"Employee id="<<eid<<endl;
	cout<<"Employee name="<<name<<endl;
	cout<<"Post="<<post<<endl;
	cout<<"Address="<<address<<endl;
	cout<<"Salary="<<salary<<endl;
	}
}
};
int main()
{
Employee e[20];
int i;
for(i=0;i<20;i++)
{
cout<<"Enter the information of Employee:"<<i+1<<endl;
e[i].getdata();
}
cout<<"Information of Employee who has salary greater than 50000 are:"<<endl;
for(i=0;i<20;i++)
{
e[i].display();
}
return 0;
}
```

7) Using class write a program that receives inputs principle amount, time and rate. Keeping rate 12% as the default argument , calculate the simple interest for five customers.

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

class customer
{
private:
float principle,rate,si;
int time;
public:
void setdata(float p,int t,float r=12);
void display();
};
void customer::setdata(float p,int t,float r)
{
principle=p;
time=t;
rate=r;
}

void customer::display()
{
si=(principle*time*rate)/100;
cout<<"Simple Interest="<<si<<endl;
}


int main()
{
float p;
int t,i;
customer c[5];
for(i=0;i<5;i++)
{
cout<<"Enter principle and time for customer"<<i+1<<endl;
cin>>p>>t;
c[i].setdata(p,t);
}
for(i=0;i<5;i++)
{
cout<<"for customer"<<i+1<<endl;
c[i].display();
}
return 0;
};
```

8) WAP to input marks of 35 students and find the highest marks using Dynamic memory allocation.

```
#include <iostream>
using namespace std;
int main ()
{
int i;
float *ptr,high=0;
ptr= new float[35];
for (i=0; i<35; i++)
{
cout <<"Enter the marks of student"<<i+1<<endl;
cin >> ptr[i];
}
for (i=0; i<35; i++)
{
        if(ptr[i]>high)
        {
        high=ptr[i];
        }
}
cout<<"Highest marks="<<high<<endl;
delete[] ptr;
return 0;
}
```

9) Create a class student with two data members represent name and roll. Use appropriate member function to read and print these data members name and roll. Derive a class marks from student that has additional data member sessional1, sessional2 to store sessional marks. Derive another class result from marks and add the sessional marks. Use appropriate member function to read and display data in the class.

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

class Student
{
protected:
char name[20];
int roll;
public:
void getdata()
{
cout<<"Enter the name"<<endl;
cin>>name;
cout<<"Enter the rollno"<<endl;
cin>>roll;
}
void putdata()
{
cout<<"Name="<<name<<endl;
cout<<"Roll="<<roll<<endl;
}
};
class Marks:public Student
{
protected:
float sessional1;
float sessional2;public:
void getmarks()
{
cout<<"Enter sessional1 marks"<<endl;
cin>>sessional1;
cout<<"Enter sessional2 marks"<<endl;
cin>>sessional2;
}
void putmarks()
{
cout<<"sessional1="<<sessional1<<endl;
cout<<"sessional2="<<sessional2<<endl;
}
};

class Result:public Marks
{
private:
float total;
public:
void display()
{
total=sessional1+sessional2;
cout<<"Total marks="<<total<<endl;
}
};
int main()
{
Result r;
r.getdata();
r.getmarks();
r.putdata();
r.putmarks();
r.display();
return 0;
}
```

10) Create a class student with two data members to represent name and age. Use member function to read and print those data. From this derive a class called boarder with member data to represent room number .Derive another class called day-scholar from class student with member data to represent address of student. In both derived class use function to read and print respective data.

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

class Student
{
protected:
char name[20];
int age;
public:
void read()
{
cout<<"Enter name"<<endl;
cin>>name;
cout<<"Enter age"<<endl;
cin>>age;
}
void print()
{
cout<<"Name="<<name<<endl;
cout<<"Age="<<age<<endl;
}
};

class Boarder:public Student
{
private:
int roomnumber;
public:
void getroomno()
{
cout<<"Enter room number"<<endl;
cin>>roomnumber;
}
void putroomno()
{
cout<<"Room number="<<roomnumber<<endl;
}
};
```

```
class Day_scholar:public Student
{
private:
char address[20];
public:
void getaddress()
{
cout<<"Enter address"<<endl;
cin>>address;
}
void putaddress()
{
cout<<"Address="<<address<<endl;
}
};

int main()
{
Boarder b;
Day_scholar d;
b.read();
b.getroomno();
d.read();
d.getaddress();
b.print();
b.putroomno();
d.print();
d.putaddress();
return 0;
}
```

11) Define two classes named ‘polar’ and ‘rectangle’ to represent points in polar and rectangle systems. Use conversion routines to convert from one system to another system using template.

```
#include <iostream>
#include <math.h>
using namespace std;

template<class T>
```

```
class Polar
{
private:
T radius;
T angle;
public:
Polar()
{
radius=0.0;
angle=0.0;
}
Polar(T r,T a)
{
radius=r;
angle=a;
}
void display()
{
cout<<"("<<radius<<","<<angle<<")"<<endl;
}
T getr()
{
return radius;
}
T geta()
{
return angle;
}
};

template<class T>
class Rectangle
{
private:
T xco;
T yco;
public:Rectangle()
{
xco=0.0;
yco=0.0;
}
```

```
Rectangle(T x,T y)
{
xco=x;
yco=y;
}
void display()
{
cout<<"("<<xco<<","<<yco<<")"<<endl;
}
operator Polar<T>()
{
float a=atan(yco/xco);
float r=sqrt(xco*xco+yco*yco);
return Polar<T>(r,a);
}
Rectangle(Polar<T> p)
{
float r=p.getr();
float a=p.geta();
xco=r*cos(a);
yco=r*sin(a);
}
};

int main()
{
cout<<"Conversion from Polar to Rectangle coordinates"<<endl;
Polar<float> p1(10.0,0.785398);
Rectangle<float> r1;
r1=p1;
cout<<"Polar coordinates=";
p1.display();
cout<<"Rectangle coordinates=";
r1.display();
cout<<"Conversion from Rectangle to Polar coordinates"<<endl;
Rectangle<float> r(7.07107,7.07107);
Polar<float> p;
p=r;
cout<<"Rectangle coordinates=";r.display();
cout<<"Polar coordinates:=";
p.display();
```

```
return 0;
}
```

12) Write a program showing ++ and - - operator overloading.

```
#include <iostream>
using namespace std;
class counter
{
private:
int count ;
public:
void getdata (int x)
{
count=x ;
}
void showdata()
{
cout<<"count="<<count<<endl;
}
counter operator ++()
{
counter temp;
temp.count=++count;
return temp;
}
counter operator --()
{
counter temp;
temp.count=--count;
return temp;
}

};

int main()
{
counter c1,c2;
c1.getdata(3) ;
c1.showdata() ;
cout<<"After overloading ++ operator"<<endl;
c2=++c1 ;
c1.showdata();
c2.showdata();
cout<<"After overloading -- operator"<<endl;
c2=--c1 ;
c1.showdata();
c2.showdata();
return 0;
}
```

13) How operator overloading support polymorphism? Explain it by overloading ‘+’ Operator to concatenate two strings.

As we know Polymorphism means ‘One name-multiple forms’.
let us consider an example, operator symbol ‘+’ is used for arithmetic operation between two numbers, however by overloading (means given additional job) it can be used for difference purposes such as addition of currency that has Rs and Paisa as its attributes, addition of complex number that has real part and imaginary part as attribute etc.
By overloading same operator ‘+’ can be used for different purpose like concatenation of strings too.

Here we illustrate the program that concatenates two strings by overloading ‘+’ operator.

```
#include<iostream>
#include<string.h>
using namespace std;
class stringc
{
private:
char str[50];
public:
stringc()
{ }
stringc(char s[])
{
strcpy(str,s);
}
void display()
{
cout<<str<<endl;
}
stringc operator +(stringc s2)
{
stringc s3;
strcpy(s3.str,str);
strcat(s3.str,s2.str);
return s3;
}
};

int main()
{
stringc s1("pokhara");
stringc s2("university");
stringc s3;
cout<<"s1=";
s1.display();
cout<<"s2=";
s2.display();
s3=s1+s2;
cout<<"s1+s2=";
s3.display();
return 0;
}
```

Output:

```
s1=pokhara
s2=university
s1+s2=pokharauniversity
```

In this way operator overloading supports polymorphism.

14) **We have a class name complex which have a data member real and imaginary .Default and parameterized constructor are there to initialize data member .WAP to overloading unary operator '++' where real and imaginary data member will be incremented and '+' operator to add two complex number.**

```
#include<iostream>
using namespace std;
class Complex
{
   int real, imag;
  public:
   Complex()
          {
                    real=0;
                    imag=0;
          }
   Complex(int r, int i)
   {
          real=r;
          imag=i;
   }
   Complex operator++()
  {
     real++;
     imag++;
     return *this;
   }
   Complex operator +(Complex c2)
  {
     Complex temp;
     temp.real = real + c2.real;
     temp.imag = imag + c2.imag;
     return temp;
   }
   void display()
   {
     cout << real << "+" << imag <<"i"<<endl;
   }
};
```

```
int main()
{
  Complex c1(10, 5);
  Complex c2(2, 4);
  Complex c3;
  ++c1;
  ++c2;
  c3= c1 + c2;
  c3.display();
  return 0;
}
```

15) **Can you have more than one constructor in a program? Write a program to find area of triangle (when sides are given) using the concept of overloaded constructor.**

Yes, it is possible to have more than one constructor in a program. In C++, constructor overloading is implemented by defining multiple constructors with different parameter lists. Each constructor has a unique signature that is determined by the number, types, and order of its parameters. When you create an object of the class, the compiler determines which constructor to call based on the arguments that you pass to it.

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

class Triangle {
  private:
    float a,b,c,s,area;

  public:
    // Default constructor
    Triangle() {
       a = 6;
       b = 8;
       c = 10;
    }
    // Constructor with three arguments
    Triangle(float s1, float s2, float s3) {
       a = s1;
       b = s2;
       c= s3;
    }

    // Method to calculate area of triangle
    float calculate()
           {
       s=(a+b+c)/2;
       area=sqrt(s*(s-a)*(s-b)*(s-c));
       return area;
    }
};

int main() {
    Triangle t1(3, 4, 5);
    cout << "Area of triangle with sides 3, 4, 5 ="<<t1.calculate()<<endl;
    Triangle t2;
    cout << "Area of triangle with sides 6, 8, 10 = "<<t2.calculate()<<endl;
    return 0;
}
```

In this program, the Triangle class has two constructors - a default constructor with no arguments, and a constructor with three arguments for the lengths of the sides. The calcualte() method calculates the area of the triangle using the formula sqrt(s(s-a)(s-b)(s-c)), where s is the semiperimeter of the triangle, and a, b, and c are the lengths of the sides.

In the main() function, two Triangle objects are created using the two different constructors. The first triangle has sides 3, 4, and 5, and is created using the three-argument constructor. The second triangle has sides 6, 8, and 10, and is created using the default constructor. The output of the program is the areas of both triangles, which are calculated using the calculate() method.

16) **Why does this pointer is widely used than object pointer? Write a program to implement pure polymorphism.**

In C++, the this pointer is a special pointer that points to the object that the member function belongs to. It is used to access the member variables and methods of the current object.

The this pointer is widely used in C++ for several reasons:

1. This pointer can be used to refer current class instance variable

When a member variable or method has the same name as a local variable or parameter, the this pointer can be used to disambiguate between them. For example, consider the following code:

```
class MyClass {
  private:
    int x;

  public:
    void setX(int x) {
        this->x = x;
    }
};
```

In this code, the setX method takes a parameter x that has the same name as the member variable x. The this pointer is used to refer to the member variable x, so that it can be assigned the value of the parameter.

2. **this pointer return the objects it points to.**

For example, the statement
return *this;
inside a member function will return the object that invoked the function.

Example:

```
#include<iostream>
#include<string.h>
using namespace std;
```

```
class person
{
char name[20];
float age;
public:
person()
{ }
person (char n[],float a)
{
strcpy(name,n);
age=a;
}
person& greater(person &x)
{
if(x.age>=age)
{
return x;
}
else
{
return *this;
}
}
void display()
{
cout<<"Name:"<<name<<endl;
cout<<"Age:"<<age<<endl;
}
};
int main()
{
person p1("Ram",52);
person p2("Hari",24);
person p3;
p3=p1.greater(p2);
cout<<"Elder person is:"<<endl;
p3.display();
return 0;
}
```

Program to implement pure polymorphism

The polymorphic variable (or assignment polymorphism) is a variable that is declared as one type but in fact holds a value of different type.

Parent p=new child();//declared as parent ,holding a child value
When a polymorphic variable is used as an argument ,this resulting function is said to exhibit pure polymorphism.

```
#include<iostream>
using namespace std;
class parent
{
        public:
                virtual string toString()
                {
                        return "parent";
                }
};
class child:public parent
{
        public:
                string toString()
                {
                        return "child";
                }
};
class StringBuffer
{
        public:
                string append(parent* value)
                {
                        return value->toString();
                }
};
```

```
int main()
{
        parent* p=new child();
        StringBuffer buffer;
        cout<<buffer.append(p)<<endl;
        return 0;
}
```

Things to be understand

In this example **'parent'** class has a virtual method '**toString() '**that returns "parent" as string. The '**child** ' class inherits from' **parent '** and overrides the **toString()** to return "child" as string.

The 'StringBuffer' class has a method 'append' that takes a pointer of type 'parent*' as argument. Inside the 'append()' method it calls the 'toString()' method of the passed object.

In the main function ,we have a pointer 'p' of type 'parent*' that is holding an object of type 'child'. When we pass this pointer to 'append' method, it calls the overridden 'toString()' method of 'child' class and prints "child".

The apped() method can take any object regardless of it's actual class, and it will call the appropriate 'toString()' method.

17) In your opinion how does the object oriented thinking solve the software crisis? List the basic characters of object oriented programming and explain any two characteristics that are directly applicable for software maintenance.
(Students are encouraged to study about software crisis first ***Ref: E. Balaguruswamy book)***

In my opinion, object-oriented thinking can help solve the software crisis by providing a more modular and flexible approach to software design, development, and maintenance. Object-oriented programming (OOP) emphasizes the use of objects, classes, inheritance, and polymorphism to create reusable and scalable software components. By breaking down complex systems into smaller, more manageable pieces, OOP can help reduce the complexity and improve the maintainability of software.

Here are some basic characteristics of object-oriented programming:

Abstraction: OOP allows developers to create abstract representations of real-world objects and concepts, and to focus on the essential features and behaviors of those objects. This can help simplify the design and implementation of software systems.

Encapsulation: OOP allows developers to encapsulate data and behavior within objects, and to control access to those objects through well-defined interfaces. This can help improve the security, reliability, and maintainability of software systems.

Inheritance: OOP allows developers to define new classes based on existing ones, and to inherit and reuse their properties and methods. This can help reduce code duplication and improve code reuse.

Polymorphism: OOP allows developers to create multiple implementations of the same interface or method, and to choose the appropriate implementation at runtime based on the type of object being used. This can help make software systems more flexible and extensible.

Two characteristics of OOP that are directly applicable for software maintenance are inheritance and polymorphism.

Inheritance: This allows developers to create new classes based on existing ones, and to reuse and extend the properties and methods of those classes. This can help reduce code duplication and improve code reuse, which in turn can help simplify maintenance tasks. For example, if a bug is found in a base class, fixing it in the base class can automatically fix it in all the derived classes that use it. Similarly, if a new feature is added to a base class, it can be automatically available to all the derived classes. This can help reduce the amount of code that needs to be maintained and tested, and can also help improve the consistency and correctness of the code.

Polymorphism: This allows developers to create multiple implementations of the same interface or method, and to choose the appropriate implementation at runtime based on the type of object being used. This can help make software systems more flexible and extensible. For example, if a new type of object is added to the system, it can be integrated seamlessly with existing code as long as it conforms to the same interface or method. This can help simplify maintenance tasks because changes can be made without affecting other parts of the program that rely on the interface or method.

Polymorphism is a fundamental concept in object-oriented programming (OOP) and it plays a constructive role in application development in a number of ways:

Code reusability: Polymorphism allows developers to create code that can be used in a variety of different contexts without having to write new code every time. By using polymorphism, developers can create a base class or interface that defines a common set of behaviors, and then create derived classes that implement those behaviors in different ways.

Flexibility: Polymorphism allows developers to write code that is more flexible and adaptable to different situations. By using polymorphism, developers can write code that can handle different types of objects, even if those objects have different behaviors or interfaces.

Simplification: Polymorphism can help simplify complex code by encapsulating complex logic into a simple, reusable interface. By using polymorphism, developers can create objects that can be used in a wide range of contexts, without needing to know the details of how those objects work internally.

Extensibility: Polymorphism can make it easier to extend existing code by allowing developers to add new behaviors to existing objects without needing to modify the original code. By using polymorphism, developers can create new classes that inherit from existing classes, and then override or extend the behaviors of those classes as needed.

In summary, polymorphism plays a constructive role in application development by enabling code reusability, flexibility, simplification, and extensibility. By using polymorphism, developers can create code that is more modular, easier to maintain, and more adaptable to changing requirements.

18) Constructor is called in derived class but it can’t be inherited. Support your answer with suitable example.

In object-oriented programming, a constructor is a special method that is called when an object is created. The constructor initializes the object's data members and sets the object's initial state. In C++, constructors are not inherited by derived classes. However, the derived class can call the constructor of the base class to initialize the base class members.

Here is an example that demonstrates this concept:

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

class Base
{
private:
int a;
public:
   Base(int x)
          {
                    a=x;
      cout << "Base constructor called "<<endl;
   }
   void showbase()
   {
          cout<<"value initialzied in base constructor="<<a<<endl;
          }
};

class Derived : public Base
{
private:
int b;
public:
   Derived(int x,int y) : Base(x)
          {
      b=y;
      cout << "Derived constructor called"<<endl;
   }
   void showderived()
   {
          cout<<"value initialzied in derived constructor="<<b<<endl;
          }
};
int main() {
   Derived d(5,10);
   d.showbase();
   d.showderived();
   return 0;
}
```

Output:

```
Base constructor called
Derived constructor called
value initialized in base constructor=5
value initialized in derived constructor=10
```

In this example, we have a base class ‘Base’ and a derived class ‘Derived’. The Base class has a constructor that takes one integer parameter, and the Derived class has a constructor that takes two integer parameter and calls the Base constructor by passing one parameter.

When we create an object of the Derived class in the main function and pass the integer value of 5 and 10 to its constructor i.e. Derived d(5,10).
Now, derived class called the base class constructor by supplying first value .i.e 5 is initialized to base class data member ‘a’ ,followed by the constructor of the Derived class constructor executed and initialized the derived class data member with value 10.

Thus, even though the Derived class does not inherit the constructor of the Base class, it can still call the constructor of the Base class explicitly in its own constructor to initialize the base class members.

19) Explain Responsibility implies non-interface. Explain with example.

When we make an object (be it a child or a software system) responsible for specific actions, we expect a certain behavior, at least when the rules are observed. Responsibility implies a degree of independence or noninterference.
If we tell a child that she is responsible for cleaning her room, we do not normally stand over her and watch while that task is being performed-that is not the nature of responsibility. Instead, we expect that, having issued a directive in the correct fashion, the desired outcome will be produced. Similarly, if a system is responsible for managing and maintaining a database, it should be able to do so without interference from other systems.
In case of, conventional programming we tend to actively supervise the child while she’s performing a task. In case of OOP we tend to handover to the child responsibility for that performance
Conventional programming proceeds largely by doing something to something else .for example, modifying a record or updating an array. Thus, one portion of code in a software system is often intimately tied by control and data connections to many other sections of the system. Such dependencies can come about through the use of global

variables, through use of pointer values or simply through inappropriate use of and dependence on implementation details of other portions of code.
A responsibility-driven design attempts to cut these links, or at least make them as unobtrusive as possible
A real-life example of responsibility and non-interference in programming can be seen in the design of a web application.
Consider a web application that has multiple components such as a user management system, a shopping cart, and a payment gateway. Each of these components has a specific responsibility and should be designed to work independently without interfering with the other components. For example, the user management system is responsible for handling user registration, login, and profile management. It should be designed to work independently and should not interfere with the functioning of the shopping cart or the payment gateway. Similarly, the shopping cart is responsible for managing the items that a user adds to their cart and should not interfere with the user management system or the payment gateway.

20) What is behavior in OOP?

In object-oriented programming (OOP), behavior refers to the actions or operations that an object can perform. In OOP, objects are instances of classes, and classes define the behavior that objects of that class can perform.
Behavior is implemented through methods, which are functions that are associated with a class. Methods define the actions that an object can perform and the way it responds to certain events.For example, in a class for a "Dog" object, there may be methods such as "bark()" and "wagTail()" that define the behavior of the dog object. An object of the Dog class can call these methods to bark or wag its tail.

Behavior in OOP is an important concept because it allows objects to interact with each other and the environment in a meaningful way. It also allows for encapsulation of data and behavior, which improves code maintainability and flexibility.

21) What is the role of behavior in OOP?

Before the development of object-oriented programming (OOP), software development methodologies focused on ideas such as characterizing the basic data structures or the overall structure of function calls. These methodologies often involved creating a formal specification of the desired application, which could be difficult to understand for both programmers and clients. The lack of structure and organization made it difficult to maintain and modify these systems as they grew larger.

In contrast, OOP emphasizes the role of behavior in software development. By modeling real-world objects and their interactions, OOP allows for more natural and intuitive understanding of the problem domain. This makes it easier for both programmers and clients to understand the requirements and design of the software. Additionally, OOP provides a way to encapsulate data and behavior within objects, which improves code maintainability and flexibility.

22) What is Responsibility-Driven Design (RDD)?

Responsibility-Driven Design (RDD), developed by Rebecca Wirfs-Brock, is an object-oriented design technique that is driven by an emphasis on behavior at all levels of development.
It is a software design methodology that focuses on identifying the responsibilities of classes and objects in a system, and then designing them to fulfill those responsibilities. RDD focuses on what action must get accomplished and which object will accomplish them.
The RDD approach focuses on modeling object behavior and identifying patterns of communication between objects.
One objective of object oriented design is first to establish who is responsible for each action to be performed. The design process consists of finding key objects during their role and responsibilities and understanding their pattern of communication. RDD initially focuses on what should be accomplished not how. RDD tries to avoid dealing with details. If any particular action is to happen, someone must be responsible for doing it. No action takes place without an agent.

23) How are object oriented Programs are designed and developed according to the concept of RDD? Describe entire process in brief.

To illustrate the concept how object oriented Programs are designed and developed according to the concept of RDD, we will discuss with a case study of Interactive Intelligent Kitchen Helper (IIKH) .ie. how IIKH can be developed according to concept of RDD.

1. Brief introduction

The Interactive Intelligent Kitchen Helper (IIKH) is a PC-based application that serves as a replacement for traditional index-card recipe systems. It has the ability to assist users in meal planning for an extended period, such as a week. The user can browse the database of recipes and interactively create a series of menus. The IIKH can also automatically adjust the recipes to any number of servings, print out menus for the entire week, for a particular day, or for a particular meal and also print an integrated grocery list of all the items needed for the recipes for the entire period.

2. Working through scenarios

The first task in software development is to refine initial specifications, which are often ambiguous. The design should accommodate potential changes easily, and scenarios can be created to understand the fundamental behavior of the system. High-level decisions, such as mapping activities to components, can be made at this stage. To understand the fundamental behavior of the system, the design team creates scenarios by acting out the use of the application as if it already exists.

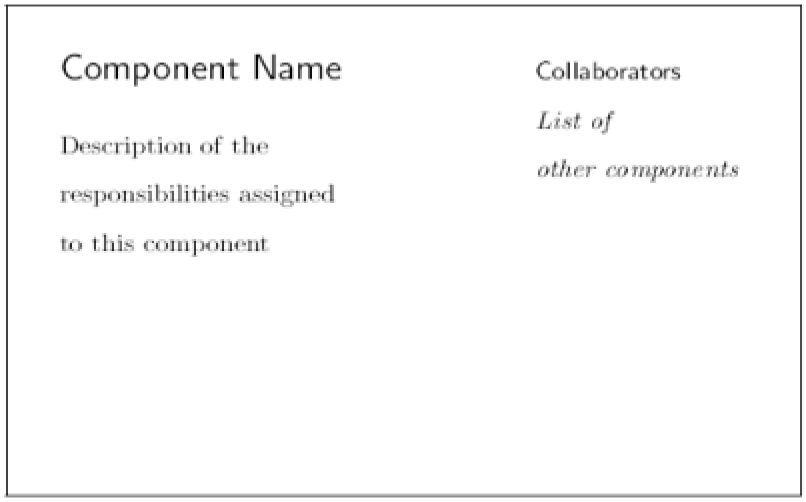
3. Identification of components

A component is simply an abstract entity that can perform tasks-that is, fulfill some responsibilities. At this point, it is not necessary to know exactly the eventual representation for a component or how a component will perform a task. A component may ultimately be turned into a function, a structure or class, or a collection of other components. At this level of development there are just two important characteristics:

- A component must have a small well-defined set of responsibilities.
- A component should interact with other components to the minimal extent possible

4. CRC cards

The design team assigns every activity to a component as a responsibility, while representing components with CRC cards. The CRC cards list the name of the component, its responsibilities, and the names of other components it must interact with. As the team discovers component responsibilities, they are added to the CRC card


Component Name
Description of the responsibilities assigned to this component
Collaborators
List of other components

5.1 Give components physical representation

CRC cards are physical cards assigned to design team members, representing software components. The member holding the card records the component's responsibilities and acts as a "surrogate" for the software during scenario simulation, passing "control" to another member as needed. The physical separation of the cards helps emphasize the importance of logical separation between components and promotes understanding of cohesion and coupling.

5.2 What/Who cycle

During the design process, the team identifies components by asking what activity needs to be performed and who performs the action. It is similar to organizing a group of people, where every activity is assigned as a responsibility to a component.

5.3Documentation

Documentation is an essential part of any software system, and should include a user manual and system design documentation. The user manual describes the interaction with the system from the user's point of view and helps verify that the development team's conception of the application matches the client's. The design documentation records major decisions made during software design and should be produced while the decisions are fresh in the creators' minds.

6. Components and Behavior

6.1 Postponing Decisions

In case of IIKH Decisions that concerning how best to let the user browse the database such as presenting a list of categories, allowing users to search by keywords or ingredients, or using scroll bars or virtual book simulation etc. a can be deferred to a later stage. since they only affect a single component and do not impact the overall functioning of the system. The primary goal is to allow users to select a recipe, and the specific details of how this is done can be decided later.

6.2 Preparing for Change

Software is subject to uncertainty and change, and designers should anticipate this and plan accordingly. The goal is to minimize the impact of changes on the software system, by reducing the coupling between components, predicting likely sources of change, and isolating the effects of change on as few components as possible. It is also important to reduce the dependency of software on hardware to enable future portability. For example, the interface for recipe browsing in our application may depend in part on the hardware on which the system is running. Future releases may be ported to different platforms. A good design will anticipate this change.

6.3 Continuing the Scenario

Having walked through the various scenarios, the software design team eventually decides that all activities can be adequately handled by six components. The Greeter needs to communicate only with the Plan Manager and the Recipe Database components. The Plan Manager needs to communicate only with the Date component; and the Date agent, only with the Meal component. The Meal component communicates with the Recipe Manager and, through this agent, with individual recipes.

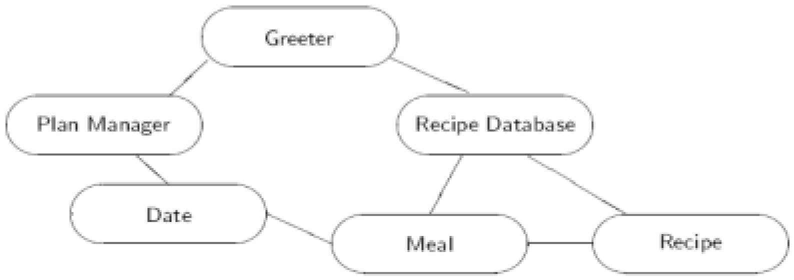


fig:Communication between the six components in the IIKH.

6.4 Interaction Diagrams

An interaction diagram is a better tool for illustrating the dynamic interactions between software components during scenario execution. It represents time moving forward from top to bottom and components as labeled vertical lines. Horizontal arrows represent message passing between components and returning control and result values back to the caller. The commentary on the side of the diagram provides a more detailed explanation of the interaction. With a time axis, the interaction diagram is able to describe better the sequencing of events during a scenario.

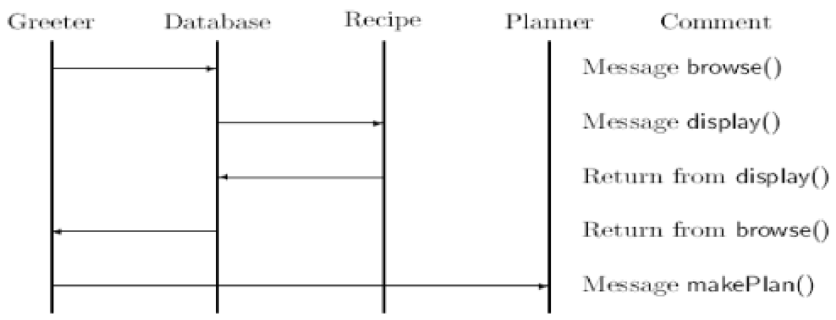


fig: An Example interaction diagram.

7. Formalize the Interface

The first step in this process is to formalize the patterns and channels of communication. A decision should be made as to the general structure that will be used to implement each component. Next, the information maintained within the component itself should be described.

8. Designing the Representation

To transform a software component description into a software system implementation, the design team can be divided into groups responsible for each component. The major part of this process is designing data structures that will maintain the state information required for the assigned responsibilities. The selection of data structures is central to the design process, and the wrong choice can lead to complex and inefficient programs. Descriptions of behavior must also be transformed into algorithms and matched against the expectations of each collaborating component to ensure that necessary data items are available to carry out each process.

9. Implementing Components

If the previous steps were correctly addressed, each responsibility or behavior will be characterized by a short description. The task at this step is to implement the desired activities in a computer language.

An important part of analysis and coding at this point is:

- Characterizing and documenting the necessary preconditions a software component requires to complete a task.
- Verifying that the software component will perform correctly when presented with legal input values.

10. Integration of Components

When software is developed, it is often divided into smaller parts or subsystems that can be designed and tested individually. Once these subsystems have been tested and confirmed to work correctly, they need to be integrated into the final software product.

The integration process involves gradually adding the subsystems to the software system, rather than doing it all at once. This helps ensure that each component works properly before it is combined with other components.

To achieve this gradual integration, stubs are used. Stubs are placeholder components that have no real behavior or have very limited behavior. They are used in place of components that have not yet been implemented or tested. When a stub is used, the rest of the system can still be tested to ensure that everything works correctly up to that point.

11. Maintenance and Evolution

The term software maintenance describes activities subsequent to the delivery of the initial working version of a software system. These activities can include correcting errors or bugs, updating the software to meet changing requirements or hardware environments, and addressing user expectations.

24) Cohesion and coupling.

Cohesion refers to the degree to which the elements within a single module or component work together to achieve a single, well-defined purpose. In other words, it measures how closely related the functions within a module are to each other.
Coupling refers to the degree to which one module or component depends on or is connected to another module or component. It measures the strength of the interdependence between two modules or components.
Here is an example to illustrate the concepts of cohesion and coupling:
Suppose you have a module in a software system that is responsible for sending emails. This module has the following functions:

validate_email_address(): Validates the format of an email address.

compose_email(): Composes an email message with a subject and body.

send_email(): Sends the email to the specified recipient.
In this example, the module has high cohesion because all of the functions within it are related to the task of sending emails. There is a clear purpose to the module, and the functions within it work together to achieve that purpose.

On the other hand, suppose that the send_email() function also has a dependency on another module in the system that is responsible for authenticating users. In this case, there is a coupling between the email sending module and the user authentication module because the email sending module depends on the user authentication module to function properly.