Lecture 10
OOP

Khola Naseem
khola.naseem@uet.edu.pk

Restriction on friend operator functions, =, [],(), ->

- **Reason?**
- So overload via member functions

Overload []

- In addition to the traditional operators, C++ allows you to change the way the [] symbols work.
- This gives you the ability to write classes that have array-like behaviors.
- For example,

  the string class overloads the [] operator so you can access the individual characters stored in string class objects. Assume the following definition exists in a program:

  ```
  string name = "William";
  ```

  The first character in the string, 'W,' is stored at name[0], so the following statement will display W on the screen.

  ```
  cout << name[0];
  ```

Overload []

- Class:

```
class IntArray
 {
 private:
    int *aptr; // Pointer to the array
    int arraySize; // Holds the array size
    void subscriptError(); // Handles invalid subscripts
public:
    IntArray(int); // Constructor
    IntArray(const IntArray &); // Copy constructor
    ~IntArray(); // Destructor
    int size() const // Returns the array size
    {
     return arraySize;
    }
    int &operator[](const int &); // Overloaded [] operator
 };
```

Credit: Khola Naseem

Overload []

- Class:

```
class IntArray
 {
 private:
    int *aptr; // Pointer to the array
    int arraySize; // Holds the array size
    void subscriptError(); // Handles invalid subscripts
public:
    IntArray(int); // Constructor
    IntArray(const IntArray &); // Copy constructor
    ~IntArray(); // Destructor
    int size() const // Returns the array size
    {
     return arraySize;
    }
    int &operator[](const int &); // Overloaded [] operator
 };
```

```
IntArray::IntArray(int s)
{
  arraySize = s;
  aptr = new int [s];
  for (int count = 0; count < arraySize; count++)
      *(aptr + count) = 0;
 }
```

Credit: Khola Naseem

Overload []

- Class:

```
class IntArray
 {
 private:
    int *aptr; // Pointer to the array
    int arraySize; // Holds the array size
    void subscriptError(); // Handles invalid subscripts
public:
    IntArray(int); // Constructor
    IntArray(const IntArray &); // Copy constructor
    ~IntArray(); // Destructor
    int size() const // Returns the array size
    {
     return arraySize;
    }
    int &operator[](const int &); // Overloaded [] operator
 };
```

```
IntArray::IntArray(const IntArray &obj)
{
  arraySize = obj.arraySize;
   aptr = new int [arraySize];
   for(int count = 0; count < arraySize; count++)
       *(aptr + count) = *(obj.aptr + count);
}
```

```
IntArray::~IntArray()
  {

 delete [] aptr;
 cout<<"memory free";
  }
```

Credit: Khola Naseem

Overload []

- Class:

```
class IntArray
 {
 private:
    int *aptr; // Pointer to the array
    int arraySize; // Holds the array size
    void subscriptError(); // Handles invalid subscripts
public:
    IntArray(int); // Constructor
    IntArray(const IntArray &); // Copy constructor
    ~IntArray(); // Destructor
    int size() const // Returns the array size
    {
     return arraySize;
    }
    int &operator[](const int &); // Overloaded [] operator
 };
```

```
void IntArray::subscriptError()
{
   cout << "ERROR: Subscript out of range.\n";
   exit(0);
}
```

```
int &IntArray::operator[](const int &sub)
 {
 if (sub < 0 || sub >= arraySize)
     subscriptError();
 return aptr[sub];
}
```

Credit: Khola Naseem

Overload []

- Class:

```
int main()
{
const int SIZE = 10; // Array size

IntArray table(SIZE);
for (int x = 0; x < SIZE; x++){
    table[x] = (x * 2);
}
for (int x = 0; x < SIZE; x++){
 cout << table[x] << " ";

}
cout << endl;
for (int x = 0; x < SIZE; x++){

    table[x] = table[x] + 5;
}
for (int x = 0; x < SIZE; x++){
    cout << table[x] << " ";
}

cout << endl;
for (int x = 0; x < SIZE; x++){
    table[x]++;
}
for (int x = 0; x < SIZE; x++){
    cout << table[x] << " "<<endl;
}
return 0;
}
```

```
0 2 4 6 8 10 12 14 16 18
5 7 9 11 13 15 17 19 21 23
6
8
10
12
14
16
18
20
22
24
memory free
```

Credit: Khola Naseem

Overload []

- Class:

```
int main()
 {
const int SIZE = 10; // Array size

IntArray table(SIZE);
IntArray table2(SIZE);
for (int x = 0; x < SIZE; x++){
    table[x] = (x * 2);
}
for (int x = 0; x < SIZE; x++){
 cout << table[x] << " ";

}
cout << endl;
for (int x = 0; x < SIZE; x++){

    table[x] = table[x] + 5;
}
for (int x = 0; x < SIZE; x++){
    cout << table[x] << " ";
}

cout << endl;
for (int x = 0; x < SIZE; x++){
    table[x]++;
}
for (int x = 0; x < SIZE; x++){
    cout << table[x] << " "<<endl;
}
table2=table;
for (int x = 0; x < SIZE; x++){
 cout << table2[x] << " ";

}
return 0;
 }
```

```
0 2 4 6 8 10 12 14 16 18
5 7 9 11 13 15 17 19 21 23
6
8
10
12
14
16
18
20
22
24
6 8 10 12 14 16 18 20 22 24 memory free
--------------------------------
```

Credit: Khola Naseem

Overload []

- Class:

```
int main()
 {
const int SIZE = 10; // Array size
IntArray table(SIZE);
for (int x = 0; x < SIZE; x++){
    table[x] = (x * 2);
}
for (int x = 0; x < SIZE; x++){
 cout << table[x] << " ";
}
cout << endl;
for (int x = 0; x < SIZE; x++){
    table[x] = table[x] + 5;
}
for (int x = 0; x < SIZE; x++){
    cout << table[x] << " ";
}

cout << endl;
for (int x = 0; x < SIZE; x++){
    table[x]++;
}
for (int x = 0; x < SIZE; x++){
    cout << table[x] << " "<<endl;
}
IntArray table2=table;
table[4]=88;
cout<<"table 2"<<endl;
for (int x = 0; x < SIZE; x++){
 cout << table2[x] << " ";
}
cout<<"table 1"<<endl;
for (int x = 0; x < SIZE; x++){
 cout << table[x] << " ";

}
return 0;
 }
```

```
0 2 4 6 8 10 12 14 16 18
5 7 9 11 13 15 17 19 21 23
6
8
10
12
14
16
18
20
22
24
table 2
6 8 10 12 14 16 18 20 22 24 table 1
6 8 10 12 88 16 18 20 22 24 memory freememory free
--------------------------------
```

Credit: Khola Naseem

Composition

- A Student object needs to calculate marks, so why not include a mark object as a member of the Student class? Such a capability is called composition (or aggregation) and is sometimes referred to as a has-a relationship a class can have objects of other classes as members
- It is also known as containment, part-whole, or has-a relationship.
- You've actually been using composition. For example class Student contained a string object as a data member
- Composition in C++ is achieved by using objects and classes; therefore, it is referred to as object composition.
- Purpose is to design complex classes by using simpler and smaller manageable parts also provide reusability.
- Complex objects are the objects that are built from smaller or a collection of objects. For example, a mobile phone is made up of various objects like a camera, battery, screen, sensors, etc.

Composition

- The object that is a part of another object is known as a sub-object. When a C++ Composition is destroyed, then all of its subobjects are destroyed as well. Such as when a car is destroyed, then its motor, frame, and other parts are also destroyed with it.

```
class Number
{
    private:
    int num1;
    public:
    void set_value(int k)
    {
        num1=k;

    }
    void show_sum(int n)
    {
        cout<<"sum of "<<num1<<" and "<<n<<" is :  "<<num1+n<<endl;

    }
};
class Add
{
    public:
    Number n;
    void print_result()
    {
        n.show_sum(5);

    }

};
int main()
{
  Add ad;
  ad.n.set_value(20);
  ad.n.show_sum(100);
  ad.print_result();
}
```

Output:

```
sum of 20 and 100 is :  120
sum of 20 and 5 is :  25
```

Credit: Khola Naseem

Composition

- Calling the constructor:

```
class A {
public:
    int x;
    A() { x = 0; }
    A(int a)
    {
        cout << "Constructor A(int a) is invoked" << endl;
        x = a;
    }
};
// Complex class
class B {
    int data;
    A objA;
public:
    B(int a)
    {
        objA(a);
        data = a;
    }
    // Function to print values of data members in class  A and B
    void display()
    {
        cout << "Data in object of class B = " << data << endl;
        cout << "Data in member object of "<< "class A in class B = " << objA.x;
    }
};
// Driver code
int main()
{
    // Creating object of class B
    B objb(25);
    // Invoking display function
    objb.display();
    return 0;
}
```

Error

```
In constructor 'B::B(int)':
[Error] no match for call to '(A) (int&)'
recipe for target 'composition.o' failed
```

Credit: Khola Naseem

Composition

- Calling the constructor:

```
class A
{
   // body of a class
};
class B
{
   A objA;
   public:
  B(arg-list) : objA(arg-list1);
};
```

Composition

- Calling the constructor:

```
class A {
public:
    int x;
    A() { x = 0; }
    A(int a)
    {
        cout << "Constructor A(int a) is invoked" << endl;
        x = a;
    }
};
// Complex class
class B {
    int data;
    A objA;
public:
    B(int a) : objA(a)
    {
        data = a;
    }
    // Function to print values of data members in class  A and B
    void display()
    {
        cout << "Data in object of class B = " << data << endl;
        cout << "Data in member object of "<< "class A in class B = " << objA.x;
    }
};
int main()
{
    // Creating object of class B
    B objb(25);
    // Invoking display function
    objb.display();
    return 0;
}
```

```
Constructor A(int a) is invoked
Data in object of class B = 25
Data in member object of class A in class B = 25
```

Credit: Khola Naseem

Composition

- Types of Object Compositions:
  - There are two basic subtypes of object composition:
    1. **Composition**
    2. **Aggregation**

Composition

- **1. Composition**
- The composition relationships are part-whole relationships where a part can only be a part of one object at a time. This means that the part is created when the object is created and destroyed when the object is destroyed.
- A person's body and heart is a good example of a part-whole relationship where if a heart is part of a person's body, then it cannot be a part of someone else's body at one time.
- To qualify as a composition, the object and a part must have the following relationship-
  - The part or child component (referred to as a member) belongs to a single object (also called class).
  - The part component can show its presence with the help of an object.
  - The part(member) component is the element of the object.
  - The part component needs to learn about the object's presence.

Composition

- Composition

```cpp
#include <iostream>

using namespace std;
class Engine {
public:
    void start() {
        cout<<"The car has an engine"<<endl;
    }
};
class Car {
public:
    Car() : engine(new Engine()) {}
    void startCar() {
        engine->start();
    }
private:
    Engine* engine;
};

int main() {
    Car car;
    car.startCar();
    return 0;
}
```

```
E:\UET\Spring 23\OOP\Class\Composition_2.exe
The car has an engine

--------------------------------
```

Aggregation

- Unlike composition, in the aggregation process, the part component can simultaneously belong to more than one object. It is also a part-whole relationship.
- The object(class) will not be responsible for the existence or presence of the parts. To be qualified as aggregation, the part and object must follow the relationship described below:
  - The part component or child component (also referred to as a member) simultaneously belongs to more than one object (also referred to as a class).
  - The part component does not show its presence with the help of an object.
  - The part(member) component is the element of the object.
  - The part component does not know about the object's presence.

Aggregation

- Example:

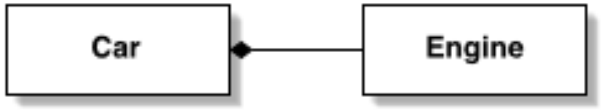
```
#include<iostream>
using namespace std;
class Address {
    private:
    string street;
    string city;
    string state;
    string zip;
public:
    Address(string street, string city, string state, string zip): street(street), city(city), state(state), zip(zip) {}
    void display()
    {
        cout<<street;
    }
};
class Person {
    private:
    string name;
    Address* address;
public:
    Person(std::string name) : name(name), address(NULL) {}
    void setAddress(Address* address) {
        this->address = address;
        address->display();
    }
};
int main() {
    Address* address = new Address("123 Main St.", "Anytown", "CA", "12345");
    Person person("John Doe");
    person.setAddress(address);
    return 0;
}
```

- Output:

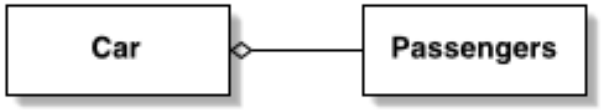
```
E:\UET\Spring 23\OOP\Class\Aggregation.exe
123 Main St.
--------------------------------
Process exited after 0.1599 seconds with return value 0
Press any key to continue . . .
```

Credit: Khola Naseem

Composition vs aggregation



Composition: every car has an engine.



Aggregation: cars may have passengers, they come and go

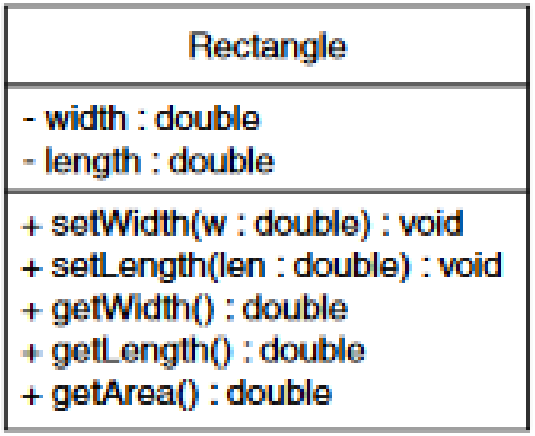
Credit: Khola Naseem

UML Class diagram:

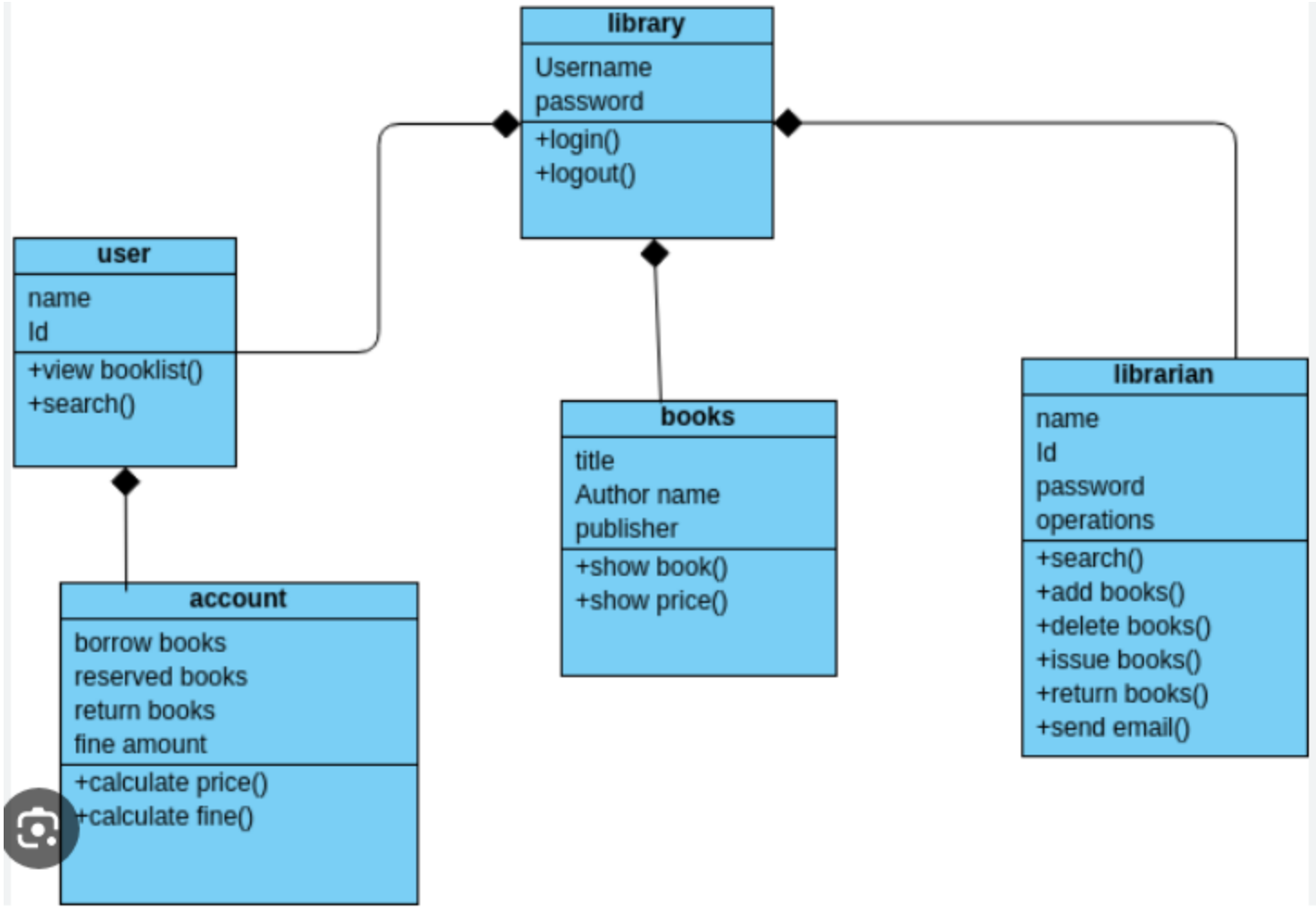
- Class diagram:

Class Name
Data Members
Member Functions

- Example:



Class diagram:



Credit: Khola Naseem

Reference material

For Practice Questions, refer to these books

- C++ Programming From Problem Analysis To Program Design, 5th Edition, D.S.Malik. Chapter 12.
- C++ How to Program, Deitel & Deitel, 5th Edition, Prentice Hall.
- Object Oriented Programming in C++ by Robert Lafore.
- Object Oriented Software Construction, Bertrand Meyer's
- Object-Oriented Analysis and Design with applications, Grady Booch et al, 3Rd Edition, Pearson, 2007
- Web