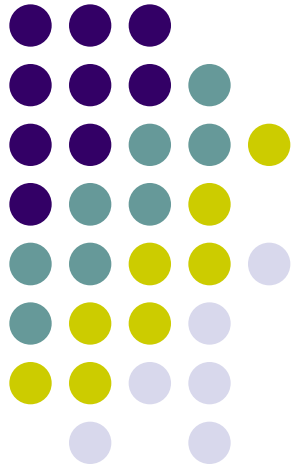
Chương 3

Lớp và đối tượng



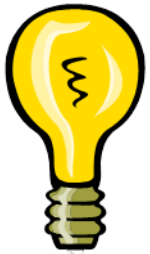
Nội dung

- Nhắc lại khái niệm đối tượng, lớp
- Khai báo và sử dụng lớp
- Khai báo và sử dụng đối tượng
- Các loại phương thức của lớp
- Thành phần tĩnh(static)
- Thành phần hằng(const)
- Con trỏ this
- Hàm bạn, lớp bạn

Object(đối tượng)

Object = Data + Methods

Object(đối tượng)



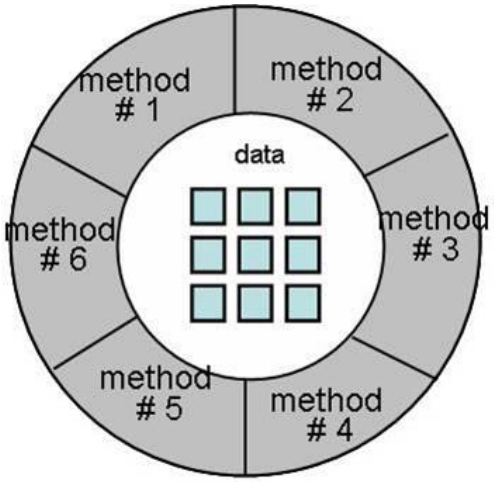
LightBulb

- **state/attributes**
  - on (true or false)
- **behavior**
  - switch on
  - switch off
  - check if on

- **Mỗi đối tượng là thể hiện của lớp**
- **Mỗi thể hiện sẽ có trạng thái khác nhau**
  - Ví dụ: hai tài khoản khác nhau sẽ có balance khác nhau
- **Đối tượng là công cụ hỗ trợ sự đóng gói dữ liệu**

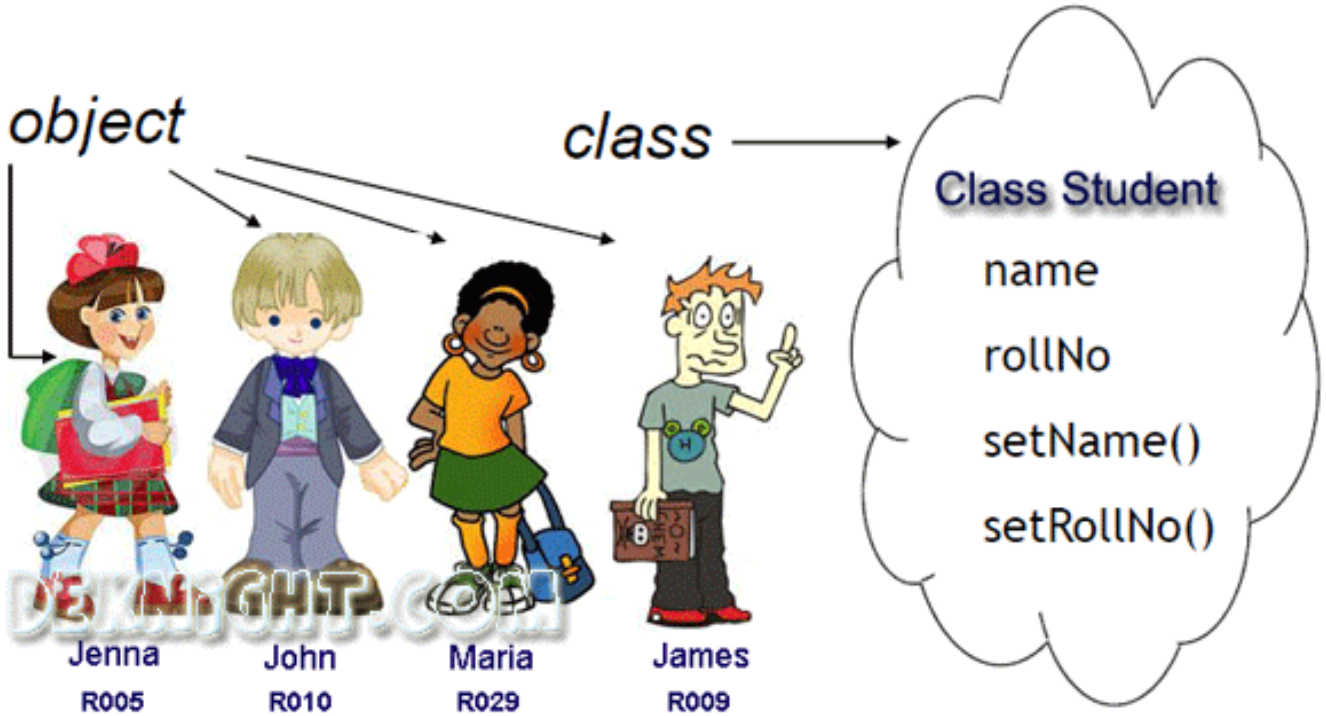
BankAccount

- **state/attributes**
  - balance
- **behavior**
  - deposit
  - withdraw
  - check balance

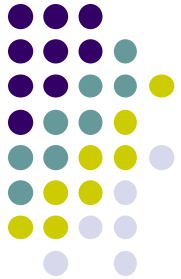


Class(lớp)

- Là đại diện cho một tập các đối tượng có cùng thuộc tính và hành vi
- Lớp là kiểu dữ liệu trừu tượng(ADT)



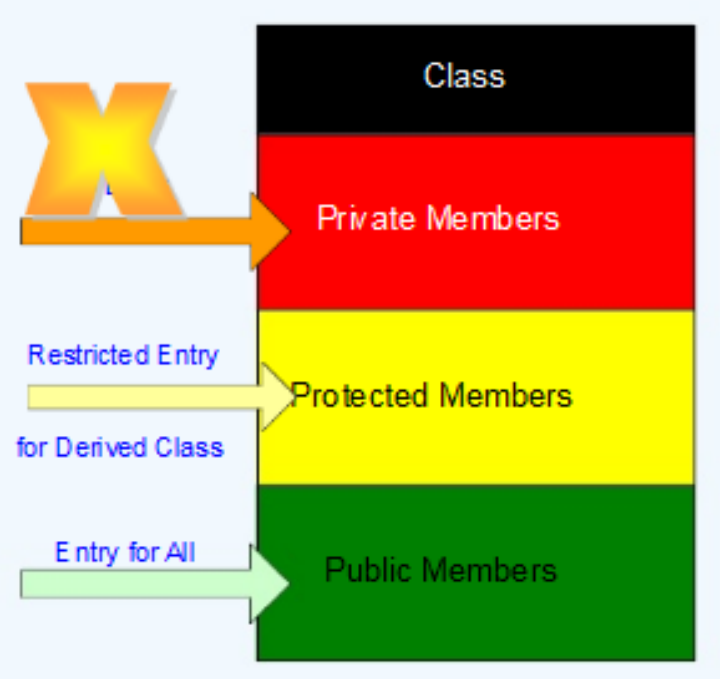
Khai báo Class(lớp)



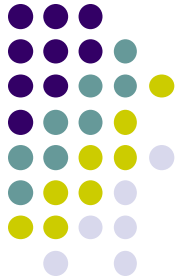
```
class   class_name
{
    access_specifier:
        member1;
    access_specifier:
        member2;
    ...
};
```

Khai báo Class(lớp)

- access_specifier: quyền truy cập, chỉ định mức độ cho phép truy cập(tính bảo mật)
- Các giới hạn truy cập:
  - public: mọi nơi nếu đối tượng tồn tại(trong và ngoài lớp)
  - private: trong phạm vi của lớp
  - protected: trong phạm vi của lớp và các lớp con thừa kế
- Ngầm định: private



Khai báo Class(lớp)



- Member
  - data members(variable)

    type name;

    double balance;
  - member functions(method)

    Khai báo:

    return_type func(type arg1, type arg2,…)

    Định nghĩa:

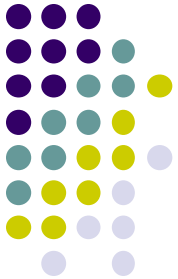
    return_type class_name::func(type arg1, type arg2,…)

    {

    //body of function

    }

Ví dụ: Class(lớp)



```
1  //Rectangle.h
2  #pragma once
3
4  class CRectangle
5  {
6  private:
7       int width, height;
8  public:
9       void setWidth(int _width);
10      int getWidth() const;
11      void setHeight(int _height);
12      int getHeight() const;
13      int area();
14 };
15
```

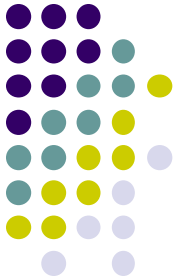
Ví dụ: Class(lớp)

```
1  //Rectangle.cpp
2  #include "Rectangle.h"
3  int CRectangle::getHeight() const
4  {
5      return height;
6  }
7  int CRectangle::getWidth() const
8  {
9      return width;
10 }
11 void CRectangle::setHeight(int _height)
12 {
13     height = _height;
14 }
15 void CRectangle::setWidth(int _width)
16 {
17     width = _width;
18 }
19 int CRectangle::area()
20 {
21     return width*height;
22 }
23
```

Phương thức getWidth() nằm trong phạm vi lớp

Thành viên dữ liệu "height" được phép truy cập trực tiếp trong phương thức thành viên setHeight()

Tạo object(đối tượng)



- Cú pháp

  class_name object1, object 2, …;
  class_name *object3;

  Ví dụ: Student s1, s2;
  CRectangle rect1, *rect2;

- Truy xuất thành viên

  object1.member1;
  object1.member2;
  object3→member1;

  Ví dụ: s1.input();
  rect2→setWidth(2);

Ví dụ

```
1  //main.cpp
2  #include <iostream>
3  #include "Rectangle.h"
4  using namespace std;
5
6  int main()
7  {
8      CRectangle rect;
9      rect.height = 2;
10     rect.setHeight(3);
11     rect.setWidth(4);
12     cout<<"Dien tich: "<< rect.area()<<endl;
13     return 0;
14 }
```

Tạo đối tượng rect

Truy xuất hàm thành viên

```
KP_CLASS, Configuration: Debug Win32 ------


main.cpp(9) : error C2248: 'CRectangle::height' : cannot access private member declared in class 'CRectangle'
```

Ví dụ

```
1  //main.cpp
2  #include <iostream>
3  #include "Rectangle.h"
4  using namespace std;
5  int main(){
6      CRectangle rect1, *rect2;
7      //rect1.height = 2;
8      rect1.setHeight(3);
9      rect1.setWidth(4);
10     cout<<"Height1: "<<rect1.getHeight()<<endl;
11     cout<<"Width1: "<<rect1.getWidth()<<endl;
12     cout<<"S1: "<< rect1.area()<<endl;
13     cout<<endl;
14     rect2 = new CRectangle();
15     rect2->setHeight(5);
16     rect2->setWidth(4);
17     cout<<"Height2: "<<rect2.getHeight()<<endl;
18     cout<<"Width2: "<<rect2->getWidth()<<endl;
19     cout<<"S2: "<< rect2->area()<<endl;
20     return 0;
21 }
```

Đối tượng là 1 con trỏ

Cấp bộ nhớ động cho con trỏ rect2

Truy xuất hàm thành viên

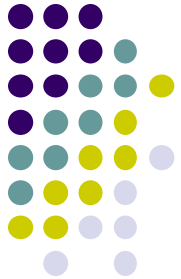
```
t from: Build
rs\user\desktop\exp_class\main.cpp(19) : error C2228: left of '.getHeight' must have class/struct/union
  type is 'CRectangle *'
  did you intend to use '->' instead?
```

Ví dụ

```
1  //main.cpp
2  #include <iostream>
3  #include "Rectangle.h"
4  using namespace std;
5  int main(){
6       CRectangle rect1, *rect2;
7       //rect1.height = 2;
8       rect1.setHeight(3);
9       rect1.setWidth(4);
10      cout<<"Height1: "<<rect1.getHeight()<<endl;
11      cout<<"Width1: "<<rect1.getWidth()<<endl;
12      cout<<"S1: "<< rect1.area()<<endl;
13      cout<<endl;
14
15      rect2 = new CRectangle();
16      rect2->setHeight(5);
17      rect2->setWidth(4);
18      cout<<"Height2: "<<rect2->getHeight()<<endl;
19      cout<<"Width2: "<<rect2->getWidth()<<endl;
20      cout<<"S2: "<< rect2->area()<<endl;
21      return 0;
22 }
```

```
Height1: 3
Width1: 4
S1: 12

Height2: 5
Width2: 4
S2: 20
Press any key to continue . . .
```

Chuẩn hóa mã nguồn



- Khai báo class trong header file

  "Rectangle.h" (interface)

- Định nghĩa tất cả các phương thức trong file

  "Rectangle.cpp" (implementation)

- Để tránh include nhiều lần khai báo class(Rectangle.h) sử dụng:

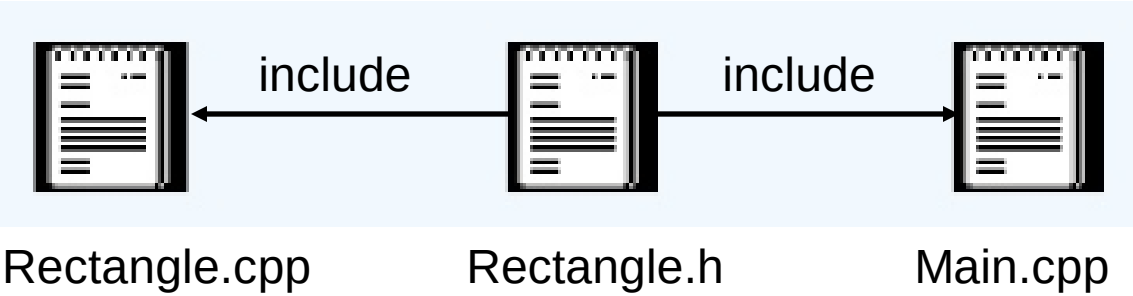
```
#ifndef _RECTANGLE_H
#define _RECTANGLE_H
//Class declaration
....
#endif
```

Hoặc

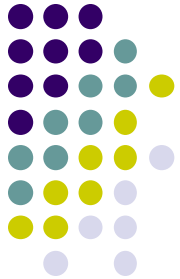
```
#pragma once
```

Chuẩn hóa mã nguồn

- Sử dụng class, tạo đối tượng trong file
  "Main.cpp" (client)



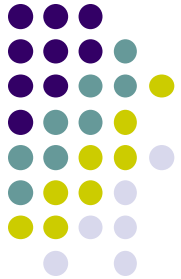
Ví dụ: Class(lớp)



- Ví dụ 1: Xây dựng và sử dụng lớp Student
  - Data members: mssv, name, averMark
  - Member functions: print(), input()

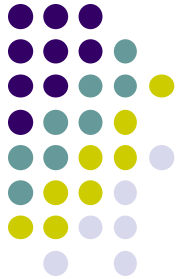
- Ví dụ 2: Xây dựng và sử dụng lớp Fraction
  - Data members: numerator, denominator
  - Member functions: print(), input(), add()

Các phương thức



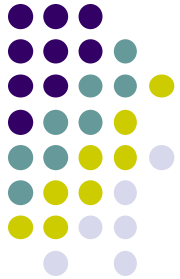
- Mỗi đối tượng thường có 4 phương thức cơ bản:
  - Phương thức khởi tạo đối tượng: constructor
  - Phương thức hủy đối tượng: destructor
  - Phương thức truy xuất dữ liệu: get
  - Phương thức cập nhật dữ liệu: set

Phương thức khởi tạo (constructor)



- Chức năng:
  - Khởi tạo các giá trị thành viên dữ liệu của đối tượng
  - Xin cấp phát bộ nhớ cho thành viên dữ liệu động
- Là hàm thành viên của lớp
- Nó được gọi tự động mỗi khi đối tượng được khai báo

Phương thức khởi tạo (constructor)



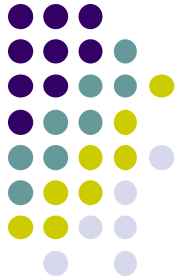
Khai báo

class_name(type arg1, type arg2,…)

Định nghĩa

```
class_name::class_name(type arg1, type arg2,…)
{
    //Than ham
}
```

Phương thức khởi tạo (constructor)



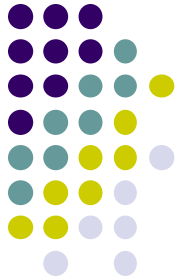
```
1  //Rectangle.h
2  #pragma once
3
4  class CRectangle
5  {
6  private:
7       int width, height;
8  public:
9       CRectangle(int, int);
10      void setWidth(int _width);
11      int getWidth() const;
12      void setHeight(int _height);
13      int getHeight() const;
14      int area();
15 };
16
```

Khai báo phương thức khởi tạo hai tham số

```
1  //Rectangle.cpp
2  #include "Rectangle.h"
3  CRectangle::CRectangle(int w, int h)
4  {
5       width = w;
6       height = h;
7  }
8  int CRectangle::getHeight() const
9  {
10      return height;
11 }
12 int CRectangle::getWidth() const
13 {
14      return width;
15 }
16 void CRectangle::setHeight(int _height)
17 {
18      height = _height;
19 }
```

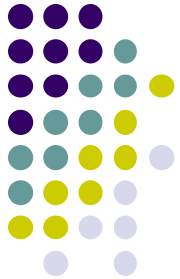
Định nghĩa phương thức khởi tạo hai tham số

Phương thức khởi tạo (constructor)



- Một số đặc điểm của phương thức khởi tạo:
  - Có cùng tên với tên lớp
  - Không có giá trị trả về
  - Có thể có nhiều phương thức khởi tạo trong cùng lớp(chồng hàm)
  - Có thể khai báo với tham số có giá trị ngầm định

Một số phương thức khởi tạo



- Default Constructor
  - Không có tham số
  - Chương trình tự động phát sinh nếu trong lớp không xây dựng phương thức khởi tạo nào
- Parameterized Constructor
  - Có một hoặc nhiều tham số
  - Đối số được dùng để khởi tạo đối tượng
- Copy Constructor
  - Có một tham số thuộc kiểu class đang khai báo
  - Sao chép thành viên dữ liệu của một đối tượng cho đối tượng khác

Ví dụ

```cpp
//main.cpp
#include <iostream>
#include "Rectangle.h"
using namespace std;
int main(){
    CRectangle rect1, *rect2;
    //rect1.height = 2;
    rect1.setHeight(3);
    rect1.setWidth(4);
    cout<<"Height1: "<<rect1.getHeight()<<endl;
    cout<<"Width1: "<<rect1.getWidth()<<endl;
    cout<<"S1: "<< rect1.area()<<endl;
    cout<<endl;

    rect2 = new CRectangle();
    rect2->setHeight(5);
    rect2->setWidth(4);
    cout<<"Height2: "<<rect2->getHeight()<<endl;
    cout<<"Width2: "<<rect2->getWidth()<<endl;
```

```cpp
//Rectangle.h
#pragma once

class CRectangle
{
private:
    int width, height;
public:
    CRectangle(int, int);
    void setWidth(int _width);
    int getWidth() const;
    void setHeight(int _height);
    int getHeight() const;
    int area();
};
```

```
lass\main.cpp(6) : error C2512: 'CRectangle' : no appropriate default constructor available
lass\main.cpp(15) : error C2512: 'CRectangle' : no appropriate default constructor available
```

Khai báo phương thức khởi tạo hai tham số

Ví dụ

```
1  //main.cpp
2  #include <iostream>
3  #include "Rectangle.h"
4  using namespace std;
5  int main(){
6      CRectangle rect1(3,4);
7      //rect1.height = 2;
8      cout<<"Height1: "<<rect1.getHeight()<<endl;
9      cout<<"Width1: "<<rect1.getWidth()<<endl;
10     cout<<"S1: "<< rect1.area()<<endl;
11     cout<<endl;
12
13     CRectangle *rect2 = new CRectangle(5,4);
14     cout<<"Height2: "<<rect2->getHeight()<<endl;
15     cout<<"Width2: "<<rect2->getWidth()<<endl;
16     cout<<"S2: "<< rect2->area()<<endl;
17     return 0;
18 }
```

```
Height1: 4
Width1: 3
S1: 12

Height2: 4
Width2: 5
S2: 20
Press any key to continue . . .
```

phương thức
khởi tạo hai tham số
được gọi

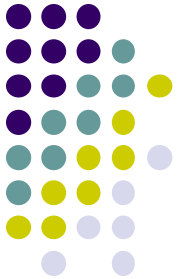
Ví dụ

```
1  //Rectangle.h
2  #pragma once
3
4  class CRectangle
5  {
6  private:
7      int width, height;
8  public:
9      CRectangle();
10     CRectangle(int, int);
11     void setWidth(int _width);
12     int getWidth() const;
13     void setHeight(int _height);
14     int getHeight() const;
15     int area();
16 };
17
```

```
1  //Rectangle.cpp
2  #include "Rectangle.h"
3  CRectangle::CRectangle()
4  {
5      width = 1;
6      height = 1;
7  }
8  CRectangle::CRectangle(int w, int h)
9  {
10     width = w;
11     height = h;
12 }
13 int CRectangle::getHeight() const
14 {
15     return height;
16 }
17 int CRectangle::getWidth() const
18 {
19     return width;
```

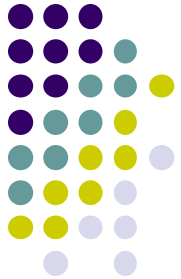
Hai phương thức
khởi tạo đối tượng

Ví dụ



```
4  using namespace std;
5  int main(){
6      CRectangle rect1(3,4);
7      //rect1.height = 2;
8      cout<<"Height1: "<<rect1.getHeight()<<endl;
9      cout<<"Width1: "<<rect1.getWidth()<<endl;
10     cout<<"S1: "<< rect1.area()<<endl;
11     cout<<endl;
12     CRectangle *rect2 = new CRectangle(5,4);
13     cout<<"Height2: "<<rect2->getHeight()<<endl;
14     cout<<"Width2: "<<rect2->getWidth()<<endl;
15     cout<<"S2: "<< rect2->area()<<endl;
16     cout<<endl;
17     CRectangle rect3;
18     cout<<"Height3: "<<rect3.getHeight()<<endl;
19     cout<<"Width3: "<<rect3.getWidth()<<endl;
20     cout<<"S3: "<< rect3.area()<<endl;
21     return 0;
22 }
```

Copy constructor



Khai báo

```
class_name(const  class_name  &arg)
```

Định nghĩa

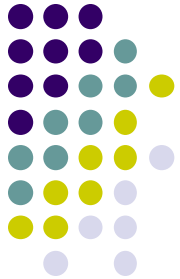
```
class_name::class_name(const  class_name  &arg)
{
    //thân hàm
    ….
}
```

Copy constructor

```
1  //Rectangle.cpp
2  #include "Rectangle.h"
3  CRectangle::CRectangle()
4  {
5      width = 1;
6      height = 1;
7  }
8  CRectangle::CRectangle(int w, int h)
9  {
10     width = w;
11     height = h;
12 }
13 CRectangle::CRectangle(const CRectangle &r)
14 {
15     width = r.width;
16     height = r.height;
17 }
```

Sao chép dữ liệu thành viên của đối tượng r cho đối tượng hiện thời

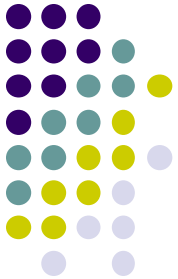
Copy constructor



- Được sử dụng để copy một phần dữ liệu thành viên của một đối tượng cho một đối tượng khác
- Copy constructor được gọi trong các tình huống sau:
  - CRectangle rect3 = rect1;
  - CRectangle rect3;

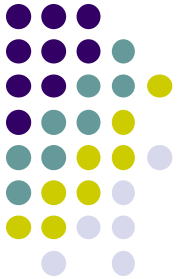
    rect3 = rect1;
  - CRectangle rect3(rect1);

Phương thức hủy đối tượng (destructor)



- Chức năng:
  - Hủy bỏ đối tượng khi không sử dụng nó nữa
  - Giải phóng bộ nhớ đã cấp phát động cho các thành viên dữ liệu
  - Đóng các file, hủy các file tạm
  - Đóng các kết nối mạng, kết nối cơ sở dữ liệu,…
- Là hàm thành viên của lớp
- Nó được gọi tự động mỗi khi đối tượng bị hủy bỏ

Phương thức hủy đối tượng (destructor)



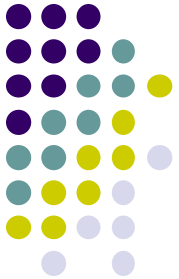
Khai báo

~class_name()

Định nghĩa

```
class_name::~class_name()
{
    //Than ham
}
```

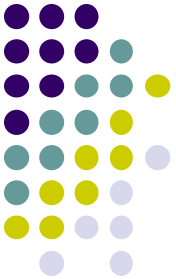
Phương thức hủy đối tượng (destructor)



- Một số đặc điểm của phương thức hủy:
  - Có cùng tên với tên lớp và bắt đầu bằng dấu ~
  - Không có giá trị trả về, không có tham số

    → mỗi lớp chỉ có duy nhất một phương thức hủy
  - Chương trình tự động phát sinh phương thức hủy nếu nó không được định nghĩa tường minh

Phương thức hủy đối tượng (destructor)



```
#pragma once

class Trainee
{
private:
     int id;
     char *name;
public:
     int getID() const;
     char* getName() const;
     void setID(int _id);
     void setName(char *n);
     Trainee(void);
     Trainee(int _id, char *n);
     ~Trainee(void);
};
```

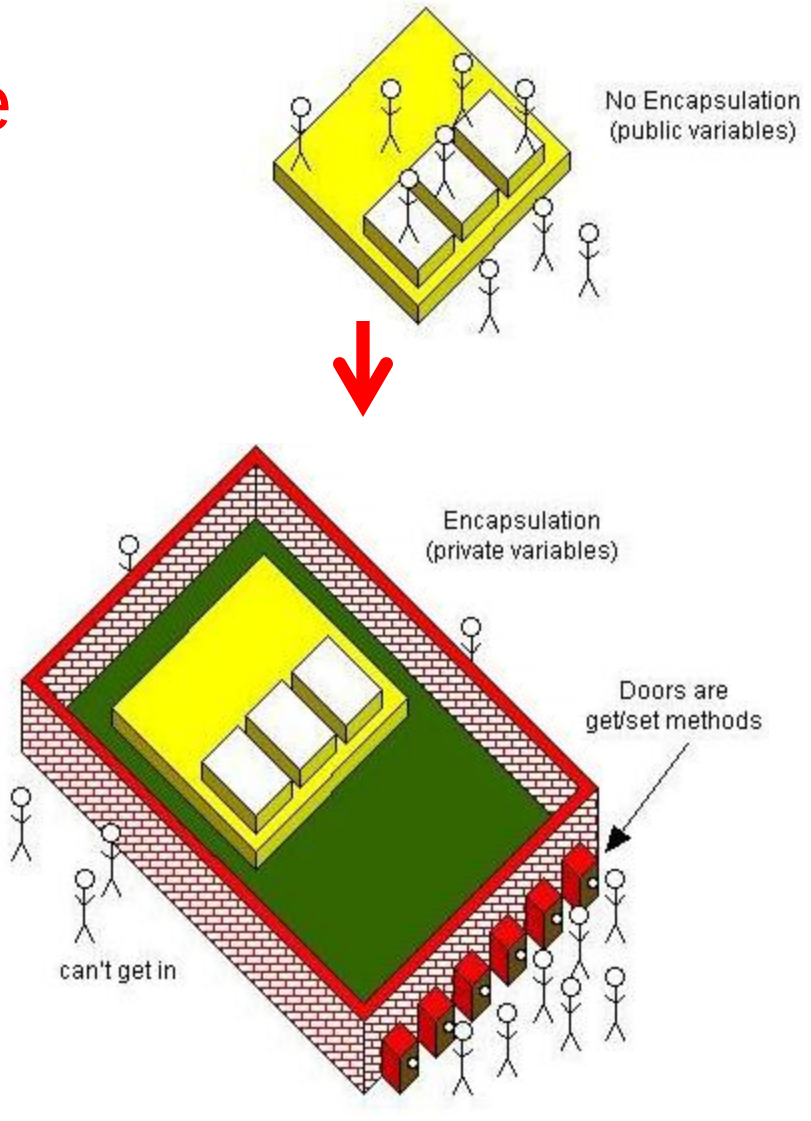
```
#include <string.h>
Trainee::Trainee(void)
{
     id = 0;
     name = NULL;
}
Trainee::Trainee(int _id, char *n)
{
     id = _id;
     name = new char[strlen(n)+1];
     strcpy(name,n);
}
Trainee::~Trainee(void)
{
     if(name != NULL)
          delete []name;
}
int Trainee::getID() const
{
```

"name" được cấp phát bộ nhớ động ở hàm tạo và giải phóng vùng nhớ ở hàm hủy

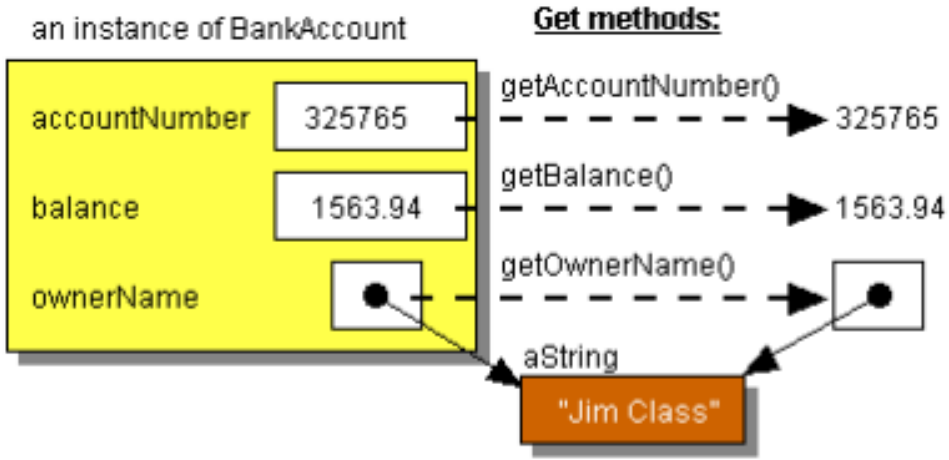
Phương thức get/set

- Dữ liệu thành viên: private
- Phương thức thành viên: public

→ Cần định nghĩa 2 phương thức get/set

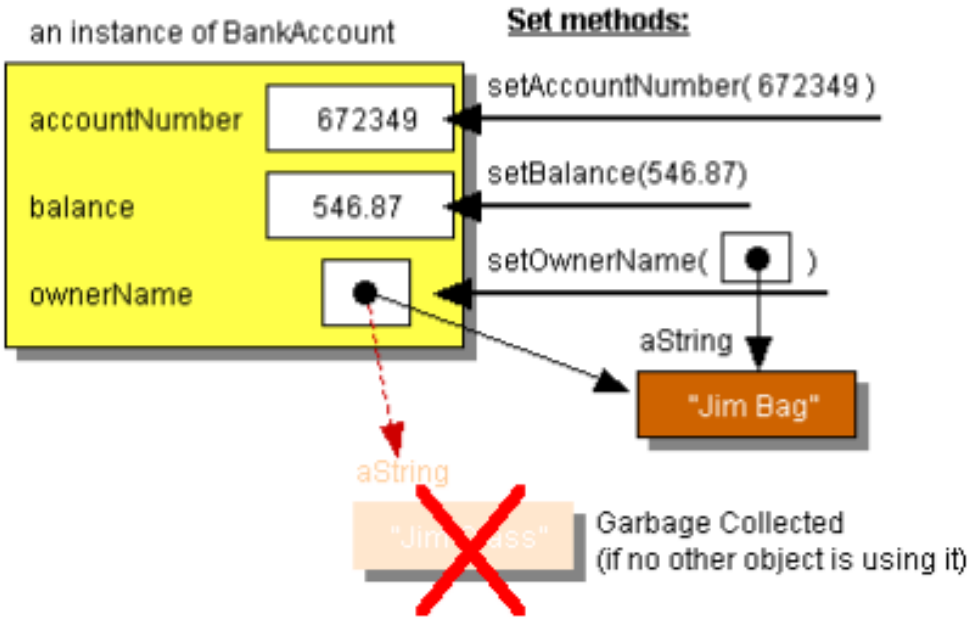


Phương thức get/set



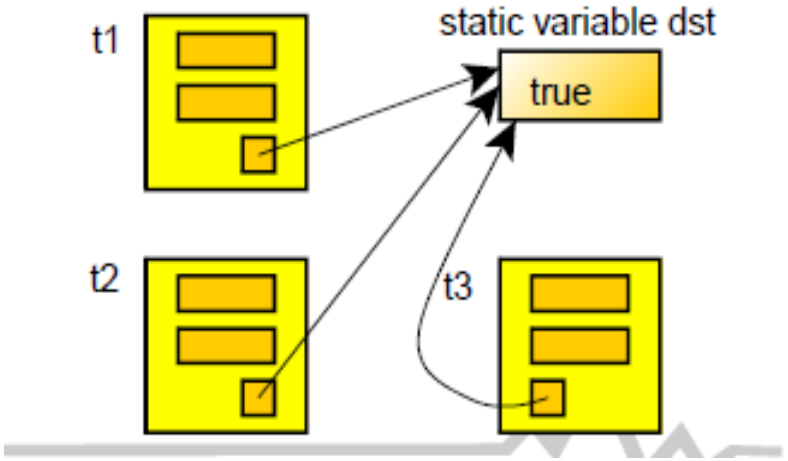

```
<fieldType> get<FieldName>() {
        return <fieldName>;
}
```

```
void set<FieldName>(<fieldType>
    <paramName>)
{
     <fieldName> = <paramName>;
}
```

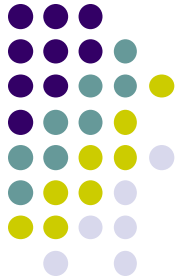


Các thành phần tĩnh(static)

- **Thành phần dữ liệu tĩnh**
  - Giá trị của thành viên dữ liệu được chia sẻ cho tất cả các đối tượng của lớp
  - Dữ liệu tĩnh được cấp phát bộ nhớ 1 lần duy nhất
  - Phải được khởi tạo bên ngoài khai báo lớp, ngoài tất cả các hàm



Các thành phần tĩnh(static)



- **Thành viên dữ liệu tĩnh**
  - Khai báo:

    static datatype var;

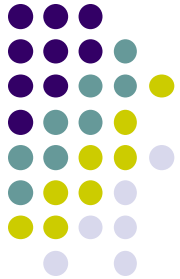
    static int count;
  - Khởi tạo:

    datatype class_name::var = value
  - Truy xuất:
    - Theo đối tượng:

      CRectangle a;

      a.count = 0;
    - Theo lớp: CRectangle::count = 1;

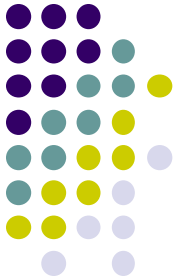
Các thành phần tĩnh(static)



- **Phương thức thành viên tĩnh**
  - Được dùng chung cho tất cả các đối tượng của lớp
  - Có thể được gọi mà không cần tạo ra đối tượng
  - Chỉ có thể truy xuất thành viên tĩnh
  - Phương thức không tĩnh(non-static) có thể truy xuất thành viên dữ liệu tĩnh
  - Khai báo:

    static return_type func (ds tham số)

Các thành phần tĩnh(static)



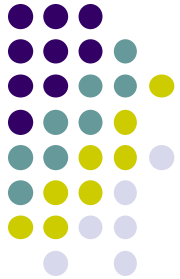
```
3  class Trainee
4  {
5  private:
6       int id;
7       char *name;
8       static int totalTrainees;
9  public:
10      Trainee(void);
11      Trainee(int _id, char *n);
12      static int getTotalTrainees();
13      ~Trainee(void);
14      int getID() const;
15      char* getName() const;
16      void setID(int _id);
17      void setName(char *n);
18
19 };
```

```
10 int Trainee::totalTrainees = 0;
11 Trainee::Trainee(int _id, char *n)
12 {
13      id = _id;
14      name = new char[strlen(n)+1];
15      strcpy(name,n);
16      totalTrainees++;
17 }
18 int Trainee::getTotalTrainees()
19 {
20      return totalTrainees;
21 }
22 Trainee::~Trainee(void)
23 {
24      if(name != NULL)
25          delete []name;
26      totalTrainees--;
27 }
```

Khai báo và định nghĩa phương thức static

Khởi tạo thành phần dữ liệu tĩnh

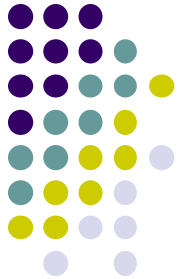
Các thành phần tĩnh(static)



```
So luong Trainee hien co: 0
So luong Trainee hien co: 1
So luong Trainee hien co: 2
So luong Trainee hien co: 1
So luong Trainee hien co: 0
Press any key to continue . . .
```

```
1  #include <iostream>
2  #include "Trainee.h"
3  using namespace std;
4
5  int main()
6  {
7      Trainee *t1, *t2;
8      cout<<"So luong Trainee hien co: "<<Trainee::getTotalTrainees()<<endl;
9      t1 = new Trainee(1,"Lena");
10     cout<<"So luong Trainee hien co: "<<Trainee::getTotalTrainees()<<endl;
11     t2 = new Trainee(2,"Anna");
12     cout<<"So luong Trainee hien co: "<<Trainee::getTotalTrainees()<<endl;
13     delete t1;
14     cout<<"So luong Trainee hien co: "<<Trainee::getTotalTrainees()<<endl;
15     delete t2;
16     cout<<"So luong Trainee hien co: "<<Trainee::getTotalTrainees()<<endl;
17     return 0;
18 }
```

Thành viên dữ liệu const



- Thành viên dữ liệu const sẽ không thay đổi giá trị trong suốt thời gian sống của đối tượng
- Các object khác nhau sẽ có giá trị thành viên dữ liệu const khác nhau
- Được khởi tạo giá trị trong hàm tạo
- Khai báo

```
const   datatype   cdata1, cdata2;
```

- Khởi tạo

```
class_name(ds tham so): cdata1(doi so), cdata2(doi so),…
{
    …..
}
```

Thành viên dữ liệu const

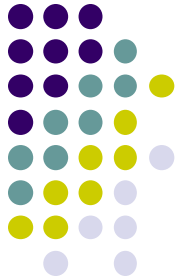
```
3  class Trainee_List
4  {
5  private:
6      const int size;
7  public:
8      Trainee_List(int s);
9      ~Trainee_List(void);
10     int getSize() const;
11 };
```

```
1  #include "Trainee_List.h"
2
3  Trainee_List::Trainee_List(int s):size(s)
4  {
5      //.....
6  }
7
8  int Trainee_List::getSize() const
9  {
10     return size;
11 }
```

Khởi tạo thành viên dữ liệu const

```
6  int main()
7  {
8      //demo constant data member of class
9      Trainee_List list1(100);
10     Trainee_List list2(150);
11     cout<<"List1 has "<<list1.getSize()<<" trainees\n";
12     cout<<"List2 has "<<list2.getSize()<<" trainees\n";
13
```

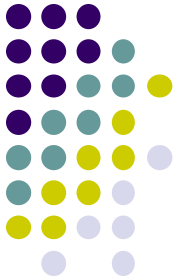
Phương thức thành viên Const



- **Mục đích**
  - Ngăn chặn sự thay đổi của thành viên dữ liệu bên trong phương thức thành viên
- Phương thức get thường được khai báo với const
- Khai báo:

```
return_type   Func(ds tham số)  const;
```

Phương thức thành viên Const

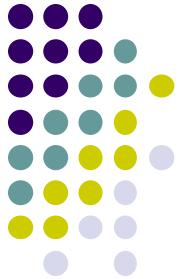


```
29  int Trainee::getID() const
30  {
31      id = id + 1;
32      return id;
33  }
34  char* Trainee::getName() const
35  {
36      return name;
37  }
38  void Trainee::setID(int _id)
39  {
40      id = _id;
41  }
42  void Trainee::setName(char *n)
```

char *Trainee::getName(void) const

```
x86)\microsoft visual studio 8\vc\include\string.h(74) : see declaration of 'strcpy'
: hanh so 1\exp2\trainee.cpp(31) : error C2166: l-value specifies const object
```

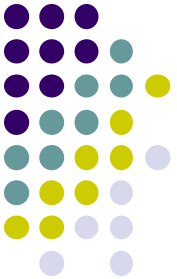
Đối tượng hằng(const object)



- Là đối tượng không thể thay đổi giá trị của dữ liệu thành viên
- Đối tượng hằng chỉ có thể gọi phương thức thành viên là tĩnh(static) hoặc hằng(const)
- Khai báo

```
const   class_name   obj;
```

Đối tượng hằng(const object)



```
5  int main()
6  {
7      //demo const object
8      const Trainee cTrainee(3,"Maria");
9
10     cTrainee.setID(4); //thay doi gia tri thanh vien du lieu --> error
11     cTrainee.printStandard(); //non-const function - error
12
13     cTrainee.getTotalTrainees(); //static function - ok
14     cout<<"Name: "<<cTrainee.getName()<<endl; //const function- ok
15
16     //demo phuong thuc static
17     Trainee *t1, *t2;
18     cout<<"So luong Trainee hien co: "<<Trainee::getTotalTrainees()<<endl;
19     t1 = new Trainee(1,"Lena");
20     cout<<"So luong Trainee hien co: "<<Trainee::getTotalTrainees()<<endl;
```

```
nh so 1\exp2\main.cpp(9) : error C2662: 'Trainee::setID' : cannot convert 'this' pointer from 'const Trainee' to 'Trainee &'
ifiers
nh so 1\exp2\main.cpp(10) : error C2662: 'Trainee::printStandard' : cannot convert 'this' pointer from 'const Trainee' to
```

Con trỏ this

- Được sử dụng làm tham số ngầm định trong tất cả các phương thức thành viên non-static của lớp
- Cho phép các đối tượng tham chiếu đến chính nó thông qua “this”
- Trỏ đến đối tượng hiện thời đang gọi phương thức thành viên
- Con trỏ this không thể được sử dụng trong phương thức tĩnh

Sử dụng con trỏ this

- Tránh sự mơ hồ

```
class Circle
{
private:
    int radius;
public:
    Circle(void);
    ~Circle(void);
    void setRadius(int radius);
    int getRadius() const;
};
```

```
void Circle::setRadius(int radius)
{
    //tham so va du lieu thanh vien trung ten
    this->radius = radius;
}
int Circle::getRadius() const
{
    return this->radius;//or return radius;
}
```

```
int main()
{
    Circle c1;
    c1.setRadius(2);
    cout<<"Radius of Circle: "<<c1.getRadius()<<endl;
    return 0;
}
```

Sử dụng con trỏ this

- Phương thức trả về đối tượng

```
class Circle
{
private:
    int radius;
    const float PI;
public:
    Circle(void);
    ~Circle(void);
    void setRadius(int radius);
    int getRadius() const;
    Circle scaleUp(int iTimes);
    float area();
};
```

```
float Circle::area()
{
    return radius*radius*PI;
}
Circle Circle::scaleUp(int iTimes)
{
    this->radius = radius*iTimes;
    return (*this);
}
Circle::Circle(void):PI(3.14)
{
}
```

```
int main()
{
    Circle c1;
    c1.setRadius(2);
    cout<<"Area of Circle1: "<<c1.area()<<endl;
    Circle c2 = c1.scaleUp(2);
    cout<<"Area of Circle2: "<<c2.area()<<endl;
    return 0;
}
```

Sử dụng con trỏ this

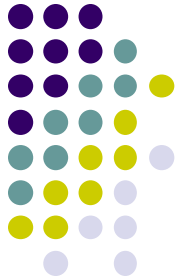
- Phương thức trả về tham chiếu đến đối tượng

```
class Circle
{
private:
    int radius;
    const float PI;
public:
    Circle(void);
    ~Circle(void);
    void setRadius(int radius);
    int getRadius() const;
    Circle& scaleUp(int iTimes);
    float area();
};
```

```
float Circle::area()
{
    return radius*radius*PI;
}
Circle& Circle::scaleUp(int iTimes)
{
    this->radius = radius*iTimes;
    return (*this);
}
Circle::Circle(void):PI(3.14)
{
}
```

```
int main()
{
    Circle c1;
    c1.setRadius(2);
    cout<<"Area of Circle1: "<<c1.area()<<endl;
    Circle c2 = c1.scaleUp(2);
    cout<<"Area of Circle2: "<<c2.area()<<endl;
    return 0;
}
```

Friend



- Dữ liệu được khai báo “private” trong lớp không thể được truy xuất từ bên ngoài
- Tuy nhiên, trong một số trường hợp các hàm/lớp bên ngoài muốn truy xuất dữ liệu “private”

→ C++ hỗ trợ một ngoại lệ: “friend”

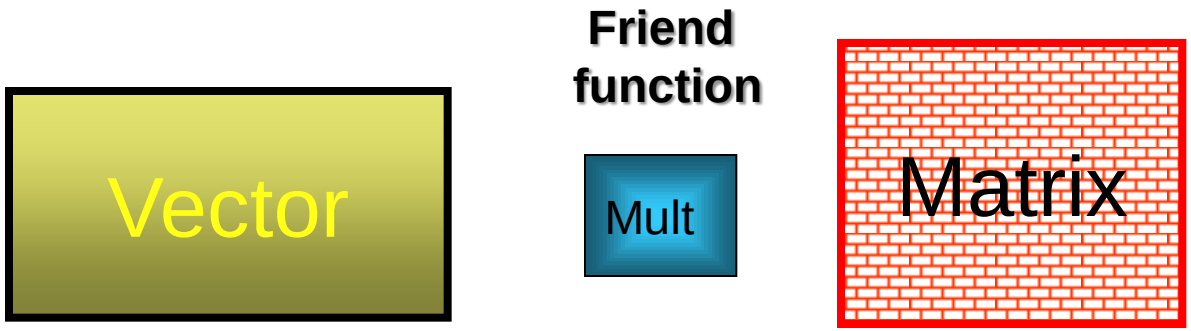
→ C++ cho phép khai báo “friend” với hàm/lớp

Friend

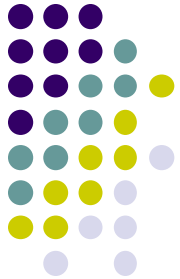
- **Tính chất của quan hệ "friend":**
  - Được cho, không được nhận
  - Không đối xứng(nếu B là bạn của A thì A không nhất thiết phải là bạn của B)
  - Không bắc cầu(nếu A là bạn của B, B là bạn của C, thì A không nhất thiết là bạn của C)

Hàm bạn(friend function)

- Hàm bạn của một lớp không phải là hàm thành viên của lớp đó
- Được phép truy xuất đến các thành phần "private" của lớp



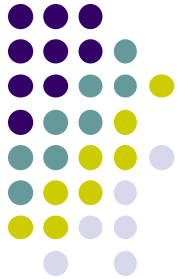
Hàm bạn(friend function)



- **Tính chất của hàm bạn:**
  - Khai báo nguyên mẫu hàm bên trong khai báo lớp với từ khóa friend
  - Được định nghĩa bên ngoài phạm vi lớp
  - Được gọi giống như hàm không thành viên
- **Khai báo:**

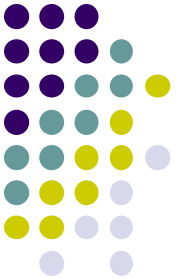
  friend <returntype> <func_name>(list_param);

Hàm bạn(friend function)



- **Các kiểu bạn bè:**
  - Hàm tự do là bạn của một lớp
  - Hàm thành viên của một lớp là bạn của một lớp khác
  - Hàm bạn của nhiều lớp

Hàm bạn(friend function)



- **Sử dụng hàm bạn trong trường hợp:**
  - Cần có một hàm có thể truy xuất thành viên “private” của hai hoặc nhiều lớp khác nhau.
  - Tạo ra các hàm xuất/nhập
  - Thiết kế một vài toán tử

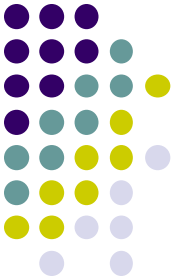
Hàm bạn(friend function)

```cpp
#pragma once

class Point
{
private:
    int x;
    int y;
public:
    Point(int x = 0, int y = 0);
    friend bool same(Point p1, Point p2);
    ~Point(void);
};
```

```cpp
#include <iostream>
#include "Point.h"
using namespace std;

//dinh nghia ham same ben ngoai pham vi lop Point
bool same(Point p1, Point p2)
{
    return ((p1.x == p2.x) && (p1.y == p2.y));
}
int main()
{
    Point p1(3,0), p2(3),p3;
    if(same(p1,p2)) cout<<"p1 trung voi p2"<<endl;
    else c p2"<<endl;
    if(same(p1,p3))  cout<<"p1 trung voi p3"<<endl;
    else  cout<<"p1 khac p3"<<endl;
}
```

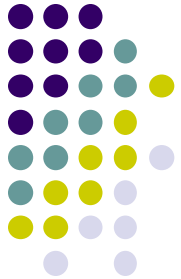
bool same(Point p1, Point p2)

Hàm bạn(friend function)



```
class Matrix;
class Vector {
  friend Vector multiply(const Matrix&,
                         const Vector&);
  ...  }
class Matrix {
  friend Vector multiply(const Matrix&,
                         const Vector&);
  ...  }
Vector multiply(const Matrix& ml,
                const Vector& vl) {
  ... }
```

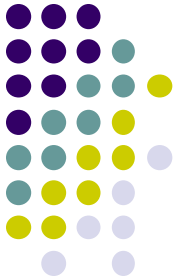
Lớp bạn(friend class)



- Friend class: tất cả các hàm thành viên của lớp A có thể truy xuất dữ liệu “private” của lớp B → A là bạn của B

```
class Point {
  ...
  friend class Line;
};
```

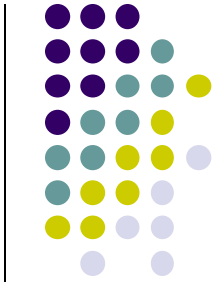
Lớp bạn(friend class)



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

class CSquare;

class CRectangle {
    int width, height;
  public:
    int area ()
      {return (width * height);}
    void convert (CSquare a);
};

class CSquare {
  private:
    int side;
  public:
    void set_side (int a)
      {side=a;}
    friend class CRectangle;
};

void CRectangle::convert (CSquare a) {
  width = a.side;
  height = a.side;
}
```

```
int main () {
  CSquare sqr;
  CRectangle rect;
  sqr.set_side(4);
  rect.convert(sqr);
  cout << rect.area();
  return 0;
}
```

Nhận xét



- Hàm/lớp bạn phá vỡ tính đóng gói, bảo mật dữ liệu
- Chỉ nên sử dụng khi thực sự cần thiết