Copyright Warning

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

WARNING

This material has been reproduced and communicated to you by or on behalf of the University of Sydney pursuant to Part VB of the Copyright Act 1968 (**the Act**).

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.

Copyright © ViSLAB, The University of Sydney

Object Oriented Design

Week 4-2

Copyright © ViSLAB, The University of Sydney

Inheritance

- One thing you cannot avoid
  - the use of `class`.
- Two ways to use `class`
  - Create an instance
  - Inheritance

Copyright © ViSLAB, The University of Sydney

Creating an instance

```
class Monster {
        char name[64];
        int power;
        int status;
    public:
        void DisplayData();
};

int main(void) {
    class Monster aMon;

    ...
    return 0;
}
```

Copyright © ViSLAB, The University of Sydney

Define a new class through inheritance

```
class Monster {
    ...
};

class FlyingMonster : public Monster {
    ...
};
```

- The colon (:) represents ***inheritance***.
- `Monster` is the base (super) class
- The new class inherits things from the super class

Copyright © ViSLAB, The University of Sydney

```cpp
#include <string.h>
#include <iostream.h>

class Monster {
    public:
        char name[64];
        int power;
        int status;
        Monster();
        void DisplayData();
};

void Monster::DisplayData() {
    cout << name << "\n";
    cout << power << "\n";
    cout << status << "\n";

    return;
}

Monster::Monster() {
    strcpy(name, "Unnamed");
    power = 100;
    status = 0;

    return;
}
```

```cpp
class FlyingMonster : public Monster
{
    // members
    int flyingPower;
};

int main(void) {
    FlyingMonster aMon;

    strcpy(aMon.name, "Neko");
    aMon.power = 500;
    aMon.status = 5;
    aMon.flyingPower = 300;
    aMon.DisplayData();

    return 0;
}
```

Copyright © ViSLAB, The University of Sydney

```
#include <string.h>
#include <iostream.h>

class FlyingMonster {
    public:
        char name[64];
        int power;
        int status;
        int flyingPower;
        FlyingMonster();
        void DisplayData();
};

void
FlyingMonster::DisplayData() {
    cout << name << "\n";
    cout << power << "\n";
    cout << status << "\n";

    return;
}
```

```
FlyingMonster::FlyingMonster() {
    strcpy(name, "Unnamed");
    power = 100;
    status = 0;

    return;
}

int main(void) {
    FlyingMonster aMon;

    strcpy(aMon.name, "Neko");
    aMon.power = 500;
    aMon.status = 5;
    aMon.flyingPower = 300;
    aMon.DisplayData();

    return 0;
}
```

Copyright © ViSLAB, The University of Sydney

Access Specifiers - revisited

- There are three access specifiers
  - public:
  - private:
  - protected

Copyright © ViSLAB, The University of Sydney

private:

- Any members under private:
  - stay invisible to outside
    - objects of other classes
    - derived classes/objects
    - under both instantiation and inheritance.

Copyright © ViSLAB, The University of Sydney

public:

- Any members under public:
  - accessible from outside.

Copyright © ViSLAB, The University of Sydney

protected:

- won't be accessible from outside classes/objects

  but
- accessible from a derived classes/objects.

Copyright © ViSLAB, The University of Sydney

```
   class BaseClass {
public:
        int public_data;
private:
        int private_data;
protected:
        int protected_data;
};

int main() {
    BaseClass bclass;

    bclass.public_data = 123;
    bclass.private_data = 456;    // <- compile error
    bclass.protected_data = 789;  // <- compile error

    return 0;
}
```

Copyright © ViSLAB, The University of Sydney

```
   class BaseClass {
public:
        int public_data;
private:
        int private_data;
protected:
        int protected_data;
};

class DerivedClass : public BaseClass {
    public:
        void NewFunc();
};
```

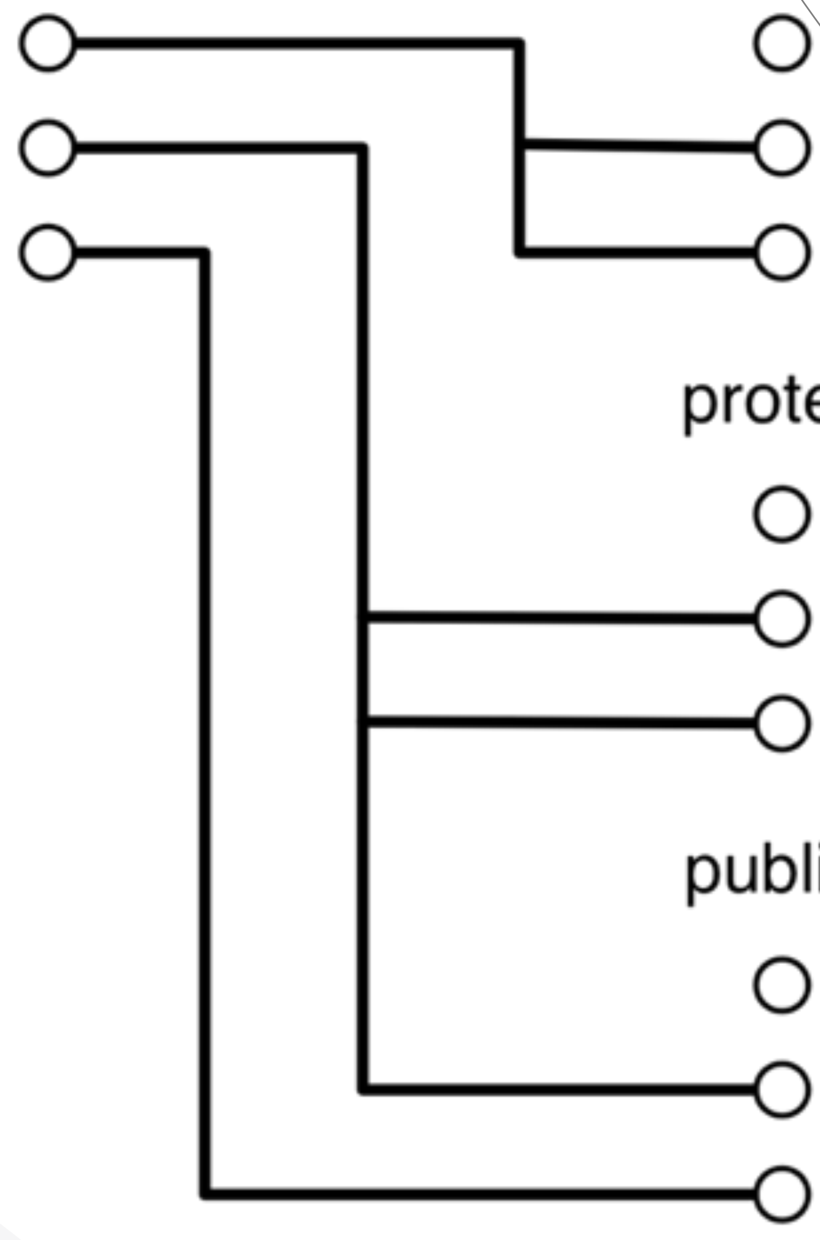
```
void DerivedClass::NewFunc() {
    public_data = 123;
    private_data = 456;    // c.error
    protected_data = 789;

    return;
}

int main() {
    DerivedClass dclass;

    dclass.NewFunc();

    return 0;
}
```

Copyright © ViSLAB, The University of Sydney

combination of a.s.

SubClass

private
protected
public



private SuperClass

private
protected
public

protected SuperClass

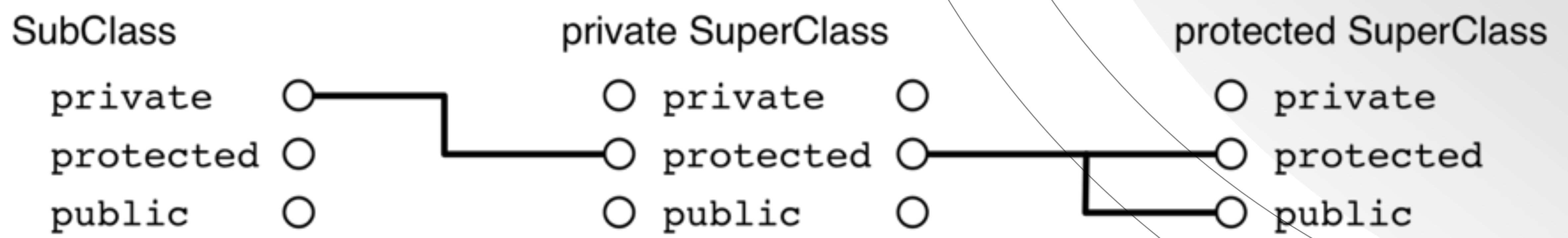
private
protected
public

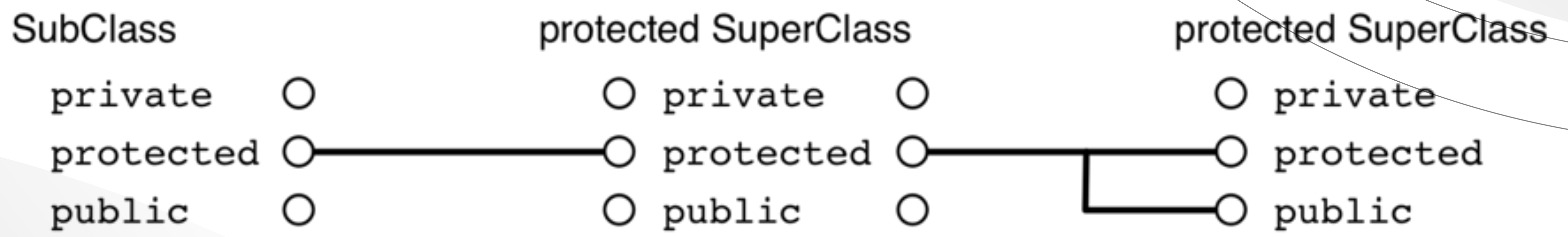
public SuperClass

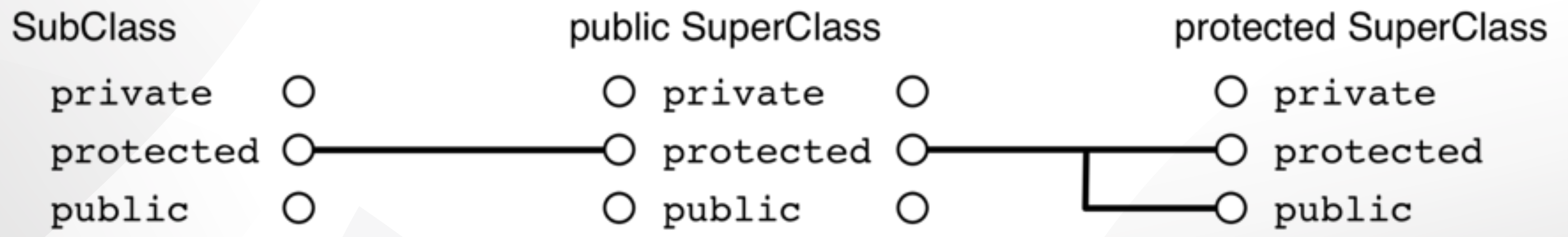
private
protected
public

Copyright © ViSLAB, The University of Sydney

combination of a.s.







Copyright © ViSLAB, The University of Sydney

Inheritance for Reuse

- Main objective
  - efficient programming (not execution)
  - you don’t have things already coded

Copyright © ViSLAB, The University of Sydney

Inheritance for Reuse

- OOP is for large-scale program dev.
    - programmers
        - use classes
        - build classes

Copyright © ViSLAB, The University of Sydney

Inheritance for Reuse

- When you build classes, you need to think...
  - will your classes be used to
    - create instances?
    - create new classes through inheritance.
  - Sometimes, you would build classes, which can be used for both.

Copyright © ViSLAB, The University of Sydney

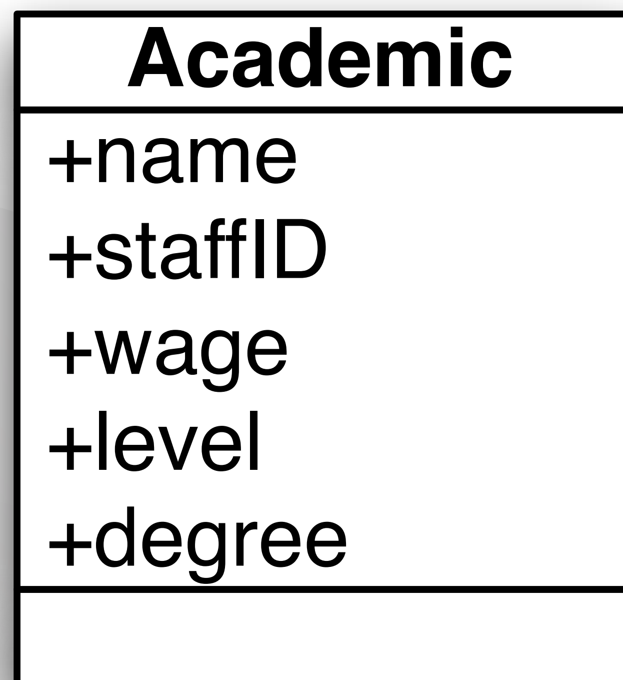
Generalization and Inheritance

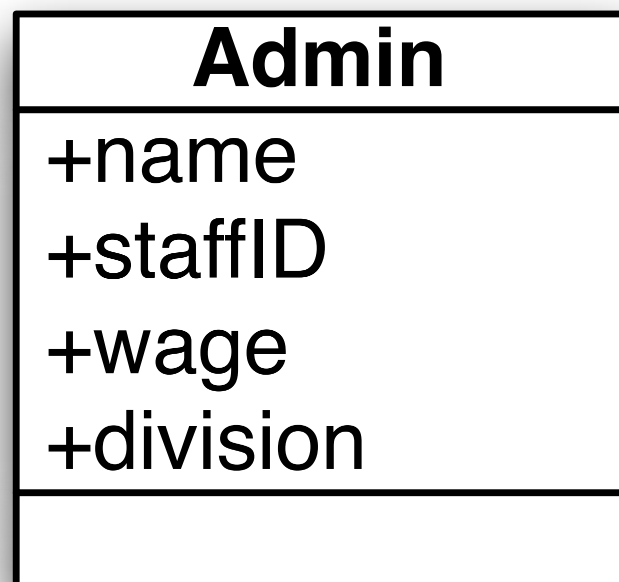
- How do we come across with inheritance?

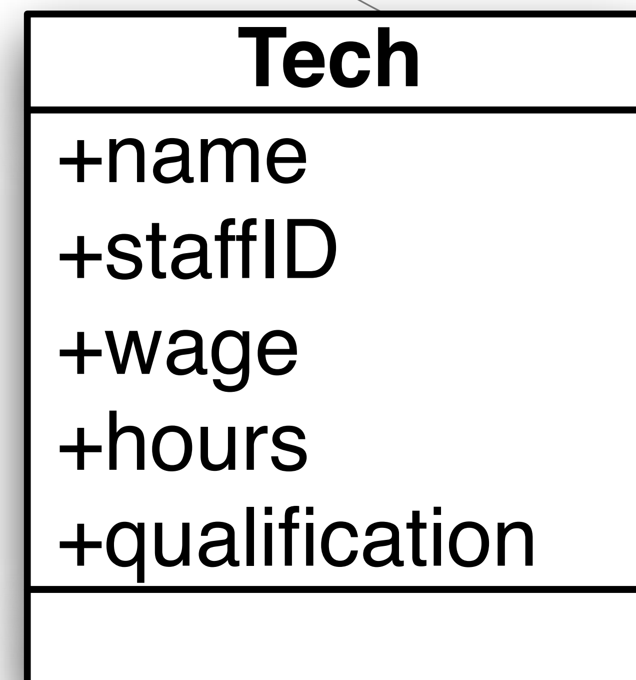
- HR system for uni.
  - Academic Staff
  - Admin Staff
  - Technical Staff

Copyright © ViSLAB, The University of Sydney

- after listing out all the facts,
- after deciding the view point
- design the first set of classes





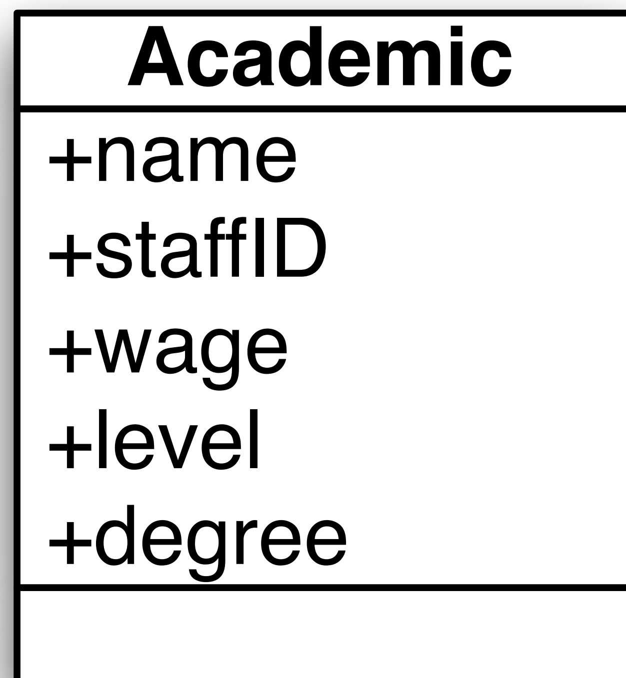


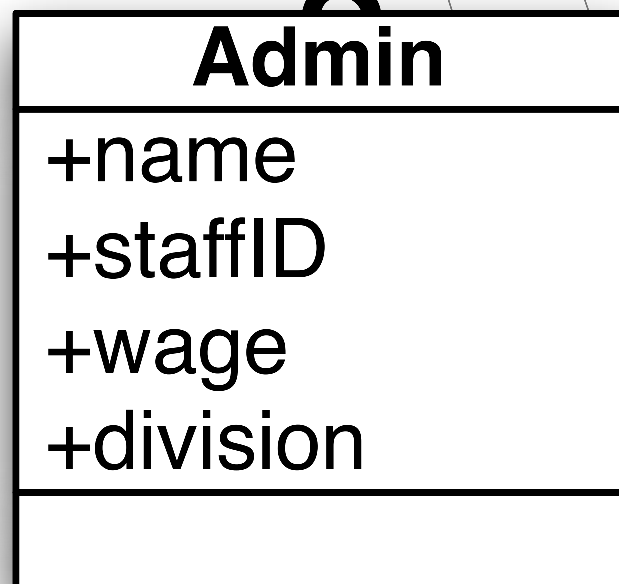
Copyright © ViSLAB, The University of Sydney

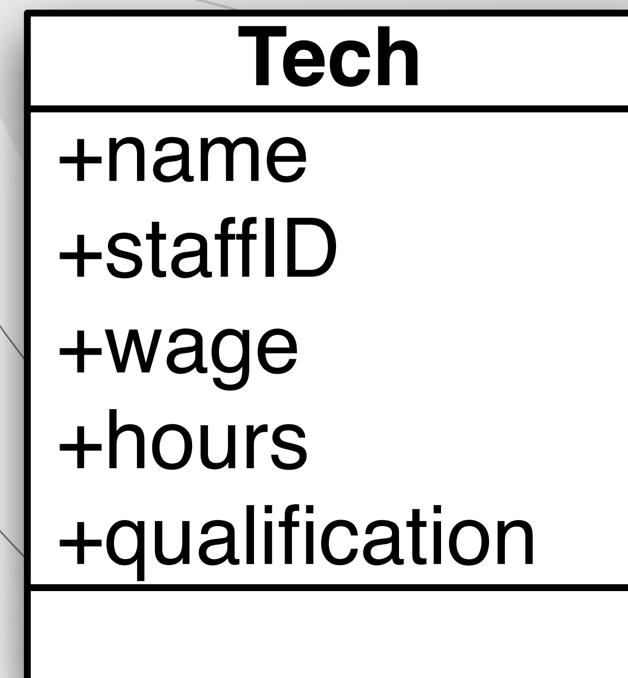
```
  class Academic {
public:
  string name;
  int staffID;
  float wage;
  int level;
  string degree;
};

class Admin {
public:
  string name;
  int staffID;
  float wage;
  int division;
};
```

```
class Tech {
public:
  string name;
  int staffID;
  float wage;
  int hours;
  string
qualification;
};
```

Copyright © ViSLAB, The University of Sydney

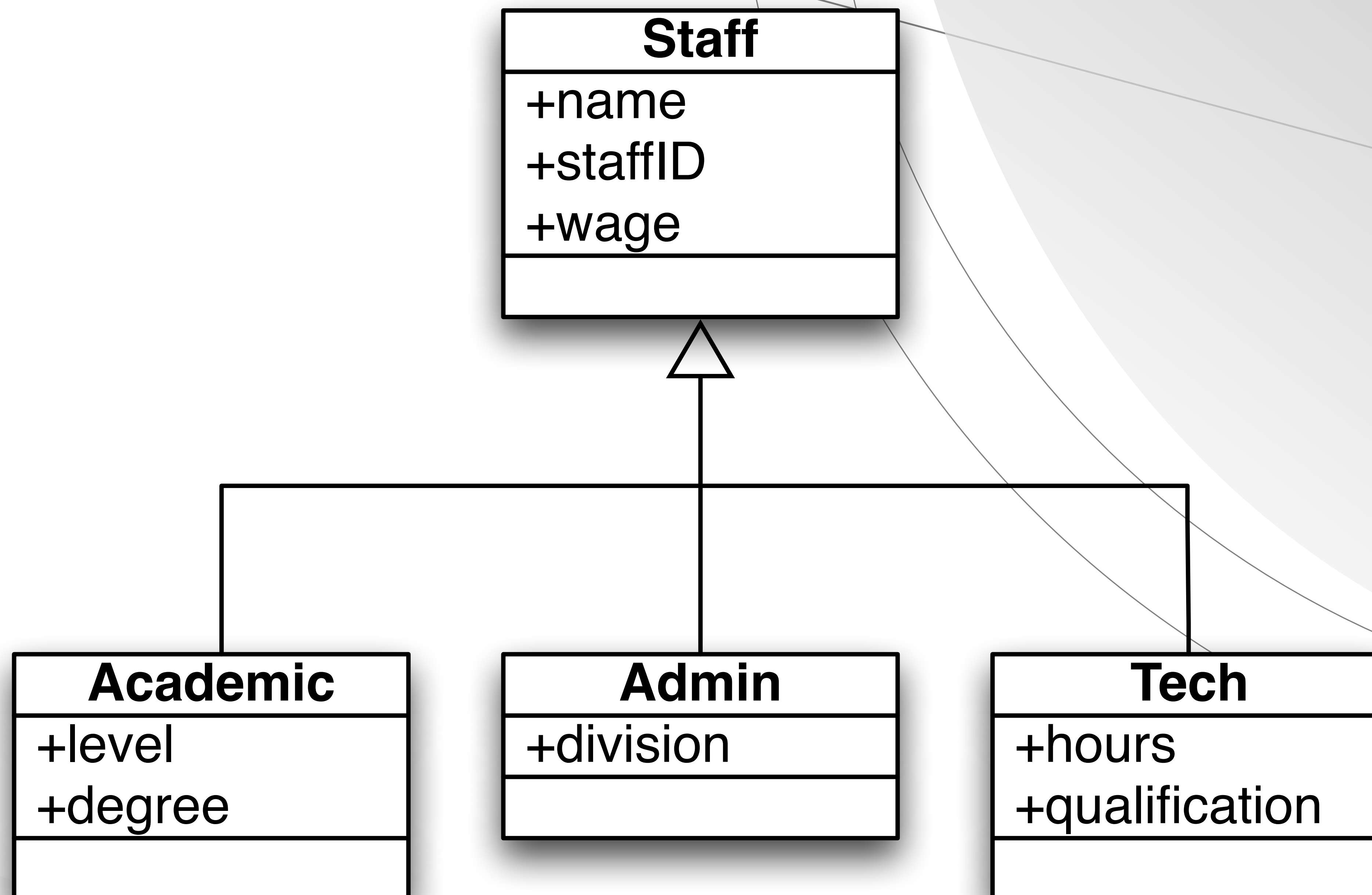






- name, staffID and wage are common members
- gather these three members and create a class to store them.
- Generalization

Copyright © ViSLAB, The University of Sydney



- Generalization during OO Design
- Inheritance the base (super) class to ease the OO Programming

Copyright © ViSLAB, The University of Sydney

```
  class Staff {
public:
  string name;
  int staffID;
  float wage;
};

class Tech: public Staff {
public:
  int hours;
  string qualification;
};

class Academic: public Staff {
public:
  int level;
  string degree;
};

class Admin: public Staff {
public:
  int division;
};
```

Copyright © ViSLAB, The University of Sydney

Inheritance

- member variables/functions are inherited (public/protected)
- ...special member function?
  - Constructor(s)
  - Destructor

Copyright © ViSLAB, The University of Sydney

Con/Destructor not inherited!

- Constructors and Destructor are not inherited.
- However, superclasses' constructors and destructors are automatically called.

Copyright © ViSLAB, The University of Sydney

```cpp
#include <iostream.h>

class MyClass {
public:
    MyClass() {cout << "const of superclass" << endl;};
    ~MyClass() {cout << "dest of superclass" << endl;};
};

class NewClass : public MyClass {
public:
    NewClass() {cout << "const of subclass" << endl;};
    ~NewClass() {cout << "dest of subclass" << endl;};
};

int main() {
    NewClass obj;
    cout << "===============" << endl;
    return 0;
}
```

Copyright © ViSLAB, The University of Sydney

Constructor w/ args

- You cannot overload a destructor
  - after calling subclass's destructor, superclass's destructor will be called.
- How about constructors?
  - There might be many constructors (with different args).

Copyright © ViSLAB, The University of Sydney

Constructor w/ args

- You need to specify which constructor to call
  - specify the constructor after ":"
  - Initialization List

Copyright © ViSLAB, The University of Sydney

```
   class MyClass {
   int myData;
public:
   MyClass(){};
   MyClass(int m):myData(m){};
   ~MyClass() {};
};

class NewClass : public MyClass {
   int newData;
public:
   NewClass() {/*do something */};
   NewClass(int m, int n):MyClass(m),newData(n) {};
   ~NewClass() {};
};
```

Copyright © ViSLAB, The University of Sydney

overloading revisited

- in C, it does not work to have functions with the same name even though their args are different

```
int foo(char* a, int b) {...};

foo(5, 2.3, “test”);
```

Copyright © ViSLAB, The University of Sydney

overloading revisited

- in C, it does not work to have functions with the same name even though their args are different

```
int foo(char* a, int b) {...};

int foo(int a, int b) {...};
```

Copyright © ViSLAB, The University of Sydney

overloading revisited

- in C++, a function is identified with the combination of its name and args.

```
int foo(int a, int b);
int foo(char* a, int b);

int main() {
  foo(2, 5);
  foo("test", 6);
}

int foo(int a, int b) {...};
int foo(char* a, int b) {...};
```

Copyright © ViSLAB, The University of Sydney

overloading revisited

- in C++, type checking is much more strict.

```
main.c
int foo(int a, int b);

int main() {
  foo(2, 5);
  foo("test", 6);
}

foo.c
int foo(char* a, int b) {...};
```

Copyright © ViSLAB, The University of Sydney

overloading revisited

- A linker needs to look after producing code to jump to appropriate functions.
- a function call in `a.obj` can be linked to the function defined in `b.obj`
- How are multiple definitions dealt with?

Copyright © ViSLAB, The University of Sydney

overloading revisited

- Internally generate identifiable function name:

- foo(int, int) fooii
- foo(char*, int) fooiPc

- Mangling....actual implementation depends on a system.

Copyright © ViSLAB, The University of Sydney

overloading revisited

- To experience the effect of mangling...
- call a C function from a C++ code
- put the prototype declaration in `extern “C” {}`

```
extern "C" {
    void c_function(int, int);  // written in C
}

int main() {
   c_function(2, 3);
   return 0;
}
```

Copyright © ViSLAB, The University of Sydney