Inheritance

What Is Inheritance?

- **Inherit Definition** -

  Derive quality and characteristics from parents or ancestors. Like you inherit features of your parents.

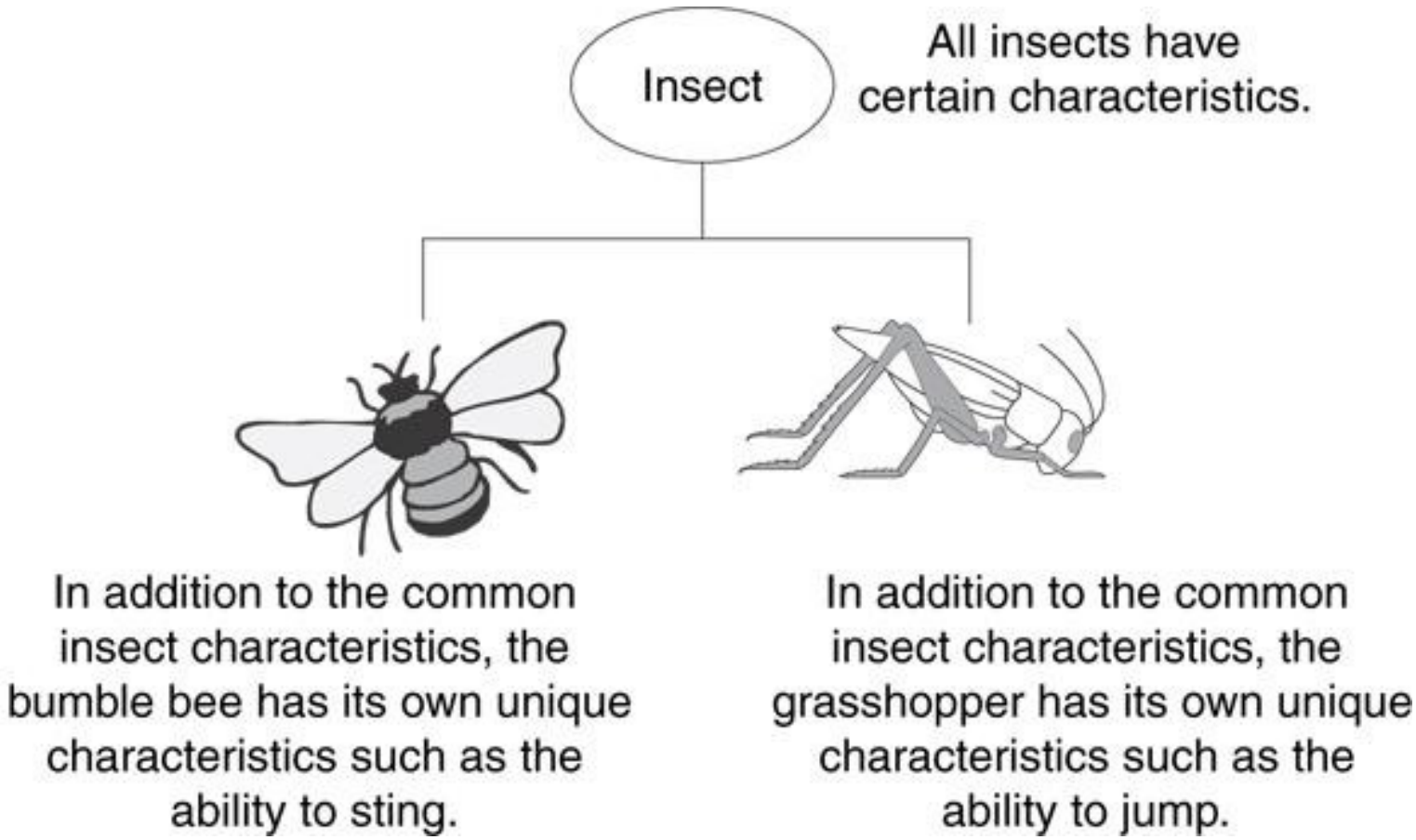
- **Example:**

  "She had inherited the beauty of her mother"

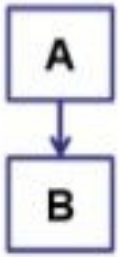
- Inheritance in Object Oriented Programming can be described as a process of creating new classes from existing classes.

- New classes inherit some of the properties and behavior of the existing classes. An existing class that is "parent" of a new class is called a **base class**. New class that inherits properties of the base class is called a **derived class(“child class”)**.
- Inheritance is a technique of code reuse. It also provides possibility to extend existing classes by creating derived classes.

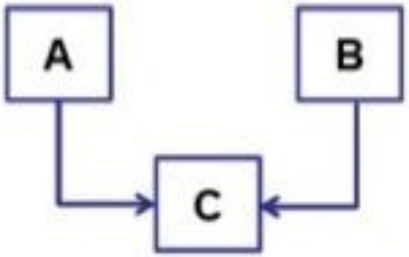
Example: Insect Taxonomy



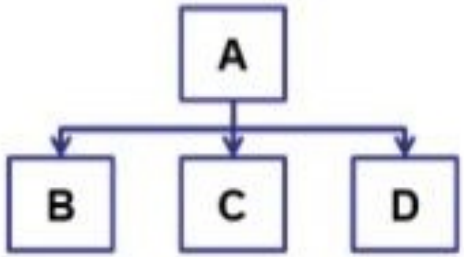
Types of Inheritance



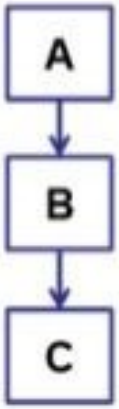
Single Inheritance



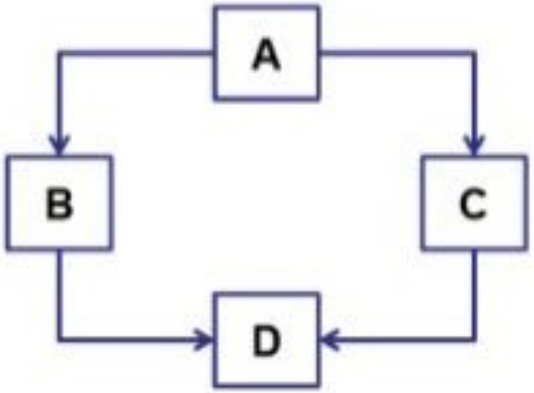
Multiple Inheritance



Hierarchical Inheritance



Multilevel Inheritance



Hybrid Inheritance

- **Single Inheritance:** It is the inheritance hierarchy wherein one derived class inherits from one base class.
- **Multiple Inheritance:** It is the inheritance hierarchy wherein one derived class inherits from multiple base class(es).
- **Hierarchical Inheritance:** It is the inheritance hierarchy wherein multiple subclasses inherit from one base class.
- **Multilevel Inheritance:** It is the inheritance hierarchy wherein subclass acts as a base class for other classes.
- **Hybrid Inheritance:** The inheritance hierarchy that reflects any legal combination of other four types of inheritance.

Inheritance – Terminology and Notation in C++

- Base class (or parent) – inherited from
- Derived class (or child) – inherits from the base class
- Notation:

```
class Base              // base class
{
    . . .
};
class Derived : public Base
{  // derived class
    . . .
};
```

What Does a Child Have?

An object of the derived class has:

- all members defined in child class
- all members declared in parent class

An object of the derived class can use:

- all public members defined in child class
- all public members defined in parent class

```
#include <iostream>
using namespace std;
class base {
int i, j;
public:
void set(int a, int b)
 { i=a; j=b; }
void show()
{ cout << i << " " << j << "\n"; }
};
class derived : public base {
int k;
public:
derived(int x) { k=x; }
void showk() { cout << k << "\n";
    }
};
```

```
int main()
{
derived ob(3);
ob.set(1, 2); // access member of
    base
ob.show(); // access member of
    base
ob.showk(); // uses member of
    derived class
return 0;
}
```

```
// This program won't
   compile.
#include <iostream>
using namespace std;
class base {
int i, j;
public:
void set(int a, int b) { i=a; j=b; }
void show() { cout << i << " "
   << j << "\n";}
};
// Public elements of base are
   private in derived.
class derived : private base {
int k;
public:
derived(int x) { k=x; }
void showk()
{ cout << k << "\n"; }
};
int main()
{
derived ob(3);
ob.set(1, 2);
// error, can't access set()
ob.show();
// error, can't access show()
return 0;
}
```

Protected Members and Class Access

- protected member access specification: like private, but accessible by objects of derived class
- Class access specification: determines how private, protected, and public members of base class are inherited by the derived class

Class Access Specifiers

public – object of derived class can be treated as object of base class (not vice-versa)

protected – more restrictive than public, but allows derived classes to know details of parents

private – prevents objects of derived class from being treated as objects of base class.

Inheritance vs. Access

Base class members

How inherited base class members appear in derived class

```
private: x
protected: y
public: z
```

base class

private

```
x is inaccessible
private: y
private: z
```

```
private: x
protected: y
public: z
```

base class

protected

```
x is inaccessible
protected: y
protected: z
```

```
private: x
protected: y
public: z
```

base class

x is inaccessible

public

```
protected: y  public: z
```

Inheritance vs. Access

class Grade

private members:

```
char letter;
float score;
void calcGrade();
```

public members:

```
void setScore(float);
float getScore();
char getLetter();
```

class Test : public Grade

private members:

```
int numQuestions;
float pointsEach;
int numMissed;
```

public members:

```
Test(int, int);
```

When `Test` class inherits from `Grade` class using `public` class access, it looks like this:

private members:

```
int numQuestions:
float pointsEach;
int numMissed;
```

public members:

```
Test(int, int);
void setScore(float);
float getScore();
char getLetter();
```

Inheritance vs. Access

class Grade

private members:

```
char letter;
float score;
void calcGrade();
```

public members:

```
void setScore(float);
float getScore();
char getLetter();
```

class Test : protected Grade

private members:

```
int numQuestions;
float pointsEach;
int numMissed;
```

public members:

```
Test(int, int);
```

When `Test` class inherits from `Grade` class using `protected` class access, it looks like this:

private members:

```
int numQuestions:
float pointsEach;
int numMissed;
```

public members:

```
Test(int, int);
```

protected members:

```
void setScore(float);
float getScore();
float getLetter();
```

Inheritance vs. Access

class Grade

private members:

```
char letter;
float score;
void calcGrade();
```

public members:

```
void setScore(float);
float getScore();
char getLetter();
```

class Test : private Grade

private members:

```
int numQuestions;
float pointsEach;
int numMissed;
```

public members:

```
Test(int, int);
```

When `Test` class inherits from `Grade` class using `private` class access, it looks like this:

private members:

```
int numQuestions:
float pointsEach;
int numMissed;
void setScore(float);
float getScore();
float getLetter();
```

public members:

```
Test(int, int);
```

Inheriting Multiple Base Classes

```
#include <iostream>
using namespace std;
class base1 {
protected:
int x;
public:
void showx() { cout << x << "\n"; }
};
class base2 {
protected:
int y;
Public:
void showy() {cout << y << "\n";}
};
```

```
// Inherit multiple base classes.
class derived: public base1, public
    base2 {
public:
void set(int i, int j) { x=i; y=j; }
};
int main()
{
derived ob;
ob.set(10, 20);
// provided by derived
ob.showx(); // from base1
ob.showy(); // from base2
return 0;
}
```

```
//EXAMPLE
#include <iostream>
using namespace std;
// Base class Shape
class Shape {
  public:
 void setWidth(int w) {
                    width = w; }
 void setHeight(int h) {
                    height = h; }
  protected:
    int width;
    int height;
};
// Base class PaintCost
class PaintCost {
  public:
  int getCost(int area) {
      return area * 70;
```

```
// Derived class
class Rectangle: public Shape, public
    PaintCost {
  public:
    int getArea() {
      return (width * height);
    } };
 int main(void) {
  Rectangle Rect;
  int area;
  Rect.setWidth(5);
  Rect.setHeight(7);
  area = Rect.getArea();
// Print the total cost of painting
  cout << "Total paint cost: $" <<
    Rect.getCost(area) << endl;
   return 0; }
```

Multi-level Inheritance

```
#include <iostream>
using namespace std;
//Base class
class base {
public:
void display1() {
 cout << "\nBase class content."; }
 };
//derived class
class derived : public base
{
public:
void display2()
{
cout << "1st derived class content."; }
    };
```

```
//derived2 class
class derived2 : public derived
{ public:
 void display3(){
 cout << "\n2nd Derived class
    content.";
 } };
 int main()
 {
derived2 D;
D.display3();
D.display2();
D.display1();
return(0);
}
```

Hierarchical Inheritance

```
#include <iostream>
#include <string.h>
using namespace
std;
//Base Class
class member {
char
gender[10];  int
age;
public:
 void get()
{ cout << "Age: "; cin >>
 age;  cout << "Gender: ";
 cin >>
```

```
void disp() {
 cout << "Age: " << age <<
    endl; cout << "Gender:
    "
    << gender << endl; }
};
//derived from member
class stud : public
member
 { char
level[20];
public:
void getdata() {
    member::get();
    cout << "Class: ";
```

```
void disp2()
 {
 member::disp();  cout << "Level:" << level;
 } };
//staff class derived
    from  member
class staff : public
member
{ float salary;
public:
 void getdata() {
    member::get()
    ;
cout << "Salary:
 Rs.";  cin >> salary; }
```

```
void disp3() {
    member::disp()
    ;
cout << "Salary: Rs."
    <<  salary << endl;
 } };

int main() {
//member
M;  staff S;
stud s;
s.getdata();
s.disp();
S.getdata();
S.disp();
return(0); }
```

Constructors and Destructors in Base and Derived Classes

- Derived classes can have their own constructors and destructors.
- When an object of a derived class is created, the base class's constructor is executed first, followed by the derived class's constructor.
- When an object of a derived class is destroyed, its destructor is called first, then that of the base class.

```
//EXAMPLE
#include<iostream>
Using namespace std;
//base class
class  base  {
public:
base()
{ cout << "Constructing base\n"; }
~base()
 { cout << "Destructing base\n"; }
};
//derived class
class derived: public base {
public:
derived()
{
 cout << "Constructing derived\n"; }
~derived()
{ cout << "Destructing derived\n"; }
};
int main()
{
derived ob;
// do nothing but construct and
    destruct ob
return 0;
}
```

Program Output
Constructing base
Constructing derived
Destructing derived
Destructing base

Thanks