NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

CL 217 – Object Oriented Programing Lab

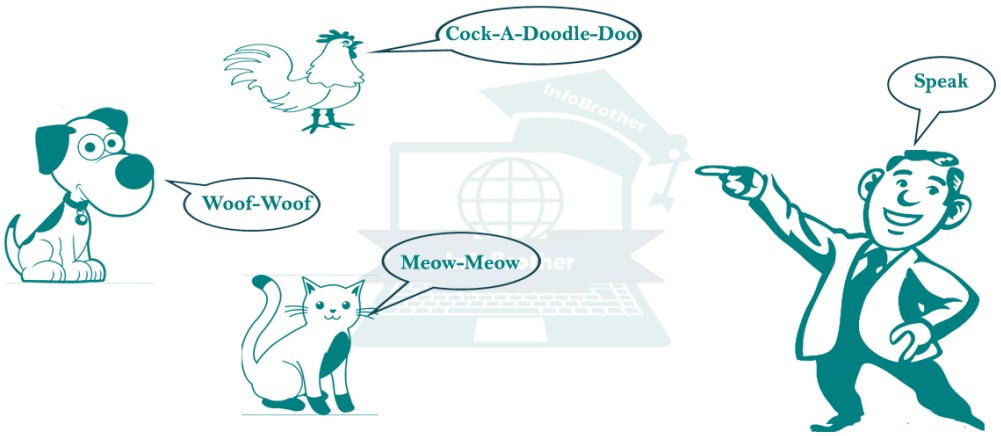
Lab 07

Outline

- Basic Concept of polymorphism
- Types of polymorphism
  - Compile time polymorphism
  - Run time polymorphism
- Examples
- Exercise

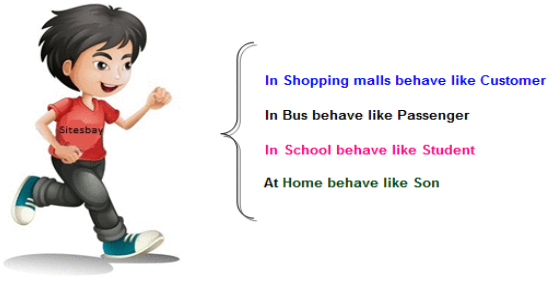
Polymorphism

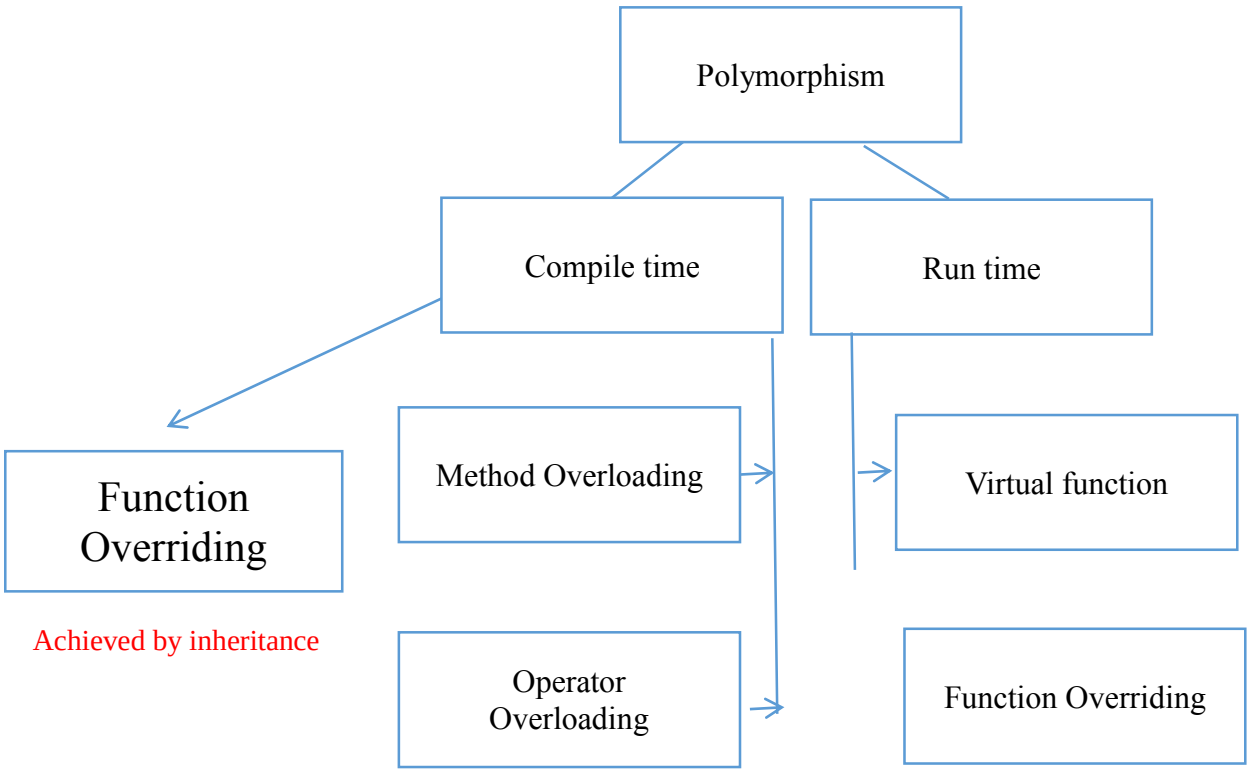
- Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and morphs means forms. So polymorphism means many forms.
- Polymorphism is a feature of OOPs that allows the object to behave differently in different conditions.
- we can define polymorphism as the ability of a message to be displayed in more than one form.
- First concept given by Hindley and Milner.
- The basic idea behind the polymorphism is that compiler does not which function to call at compile time.



Real life example of Polymorphism

Suppose if you are in class room that time you behave like a student, when you are in market at that time you behave like a customer, when you at your home at that time you behave like a son or daughter, Here one person have different-different behaviors.





Static / Compile time polymorphism

- It is also called Early Binding
- It happens where more than one methods share the same name with different parameters or signature and different return type.
- It is known as Early Binding because the compiler is aware of the functions with same name and also which overloaded function is to be called is known at compile time.

```
using namespace std;
class Geeks
{
   public:

   // function with 1 int parameter
   void func(int x)
   {
      cout << "value of x is " << x << endl;
   }

   // function with same name but 1 double parameter
   void func(double x)
   {
      cout << "value of x is " << x << endl;
   }

   // function with same name and 2 int parameters
   void func(int x, int y)
   {
      cout << "value of x and y is " << x << ", " << y << endl;
   }
};
```

```
int main() {
   Geeks obj1;

    obj1.func(7);

   // The second 'func' is called
   obj1.func(9.132);

   // The third 'func' is called
   obj1.func(85,64);
   return 0;
}
```

Function-overloading-compile time

```cpp
#include <iostream>
using namespace std;
// Function with 2 int parameters
int sum(int num1, int num2) {
    return num1 + num2;
}
// Function with 2 double parameters
double sum(double num1, double num2) {
    return num1 + num2;
}
// Function with 3 int parameters
int sum(int num1, int num2, int num3) {
    return num1 + num2 + num3;
}
int main() {
    // Call function with 2 int parameters
    cout << "Sum 1 = " << sum(5, 6) << endl;

    // Call function with 2 double parameters
    cout << "Sum 2 = " << sum(5.5, 6.6) << endl;

    // Call function with 3 int parameters
    cout << "Sum 3 = " << sum(5, 6, 7) << endl;
    return 0;
}
```

- Here, we have created 3 different sum() functions with different parameters (number/type of parameters).
- And, based on the arguments passed during a function call, a particular sum() is called.
- It's a compile-time polymorphism because the compiler knows which function to execute before the program is compiled.

Function overriding-compile time

```cpp
#include <iostream>
using namespace std;
class A {
public:
  void disp(){
     cout<<"Super Class Function"<<endl;
  }
};
class B: public A{
public:
  void disp(){
     cout<<"Sub Class Function";
  }
};
int main() {
  //Parent class object
  A obj;
  obj.disp();
  //Child class object
  B obj2;
  obj2.disp();
  return 0;
}
```

In case of function overriding we have two definitions of the same function, one is parent class and one in child class.

The call to the function is determined at runtime to decide which definition of the function is to be called, that's the reason it is called runtime polymorphism.

Dynamic / Run time polymorphism

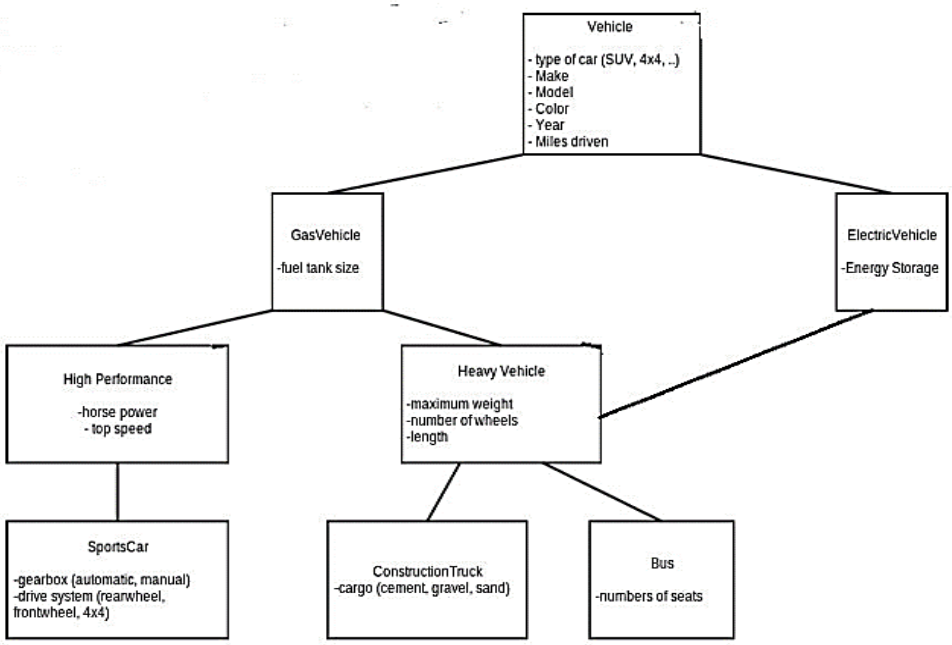
- This refers to the entity which changes its form depending on circumstances at runtime.
- Method Overriding uses runtime Polymorphism.
- Runtime Polymorphism is done using virtual and inheritance.
- It is also called Late Binding.

```
include <bits/stdc++.h>
using namespace std;

class base
{
public:
   virtual void print ()
   { cout<< "print base class" <<endl; }

   void show ()
   { cout<< "show base class" <<endl; }
};

class derived:public base
{
public:
   void print ()    { cout<< "print derived class" <<endl; }
   void show ()
   { cout<< "show derived class" <<endl; }
};
```

```
//main function
int main()
{
   base *bptr;
   derived d;
   bptr = &d;
   bptr->print();
   bptr->show();

   return 0;
}
```

Activity

1. Write a program that will create a a function to find the area rectangle. Use overloading to provide for calculating real and integer values.
2. Write an overloaded function "min" that takes either two or three parameters of type int and returns the smallest of them. That is you create two sub-functions: 1.int min(int,int) and int min(int, int, int). 2. both return minimum 3.write also the main program that call the two functions.
3. Write program to calculate the volume of sphare and cylindrical shape using polymorphism. Parameter of the volume must be pass through main function to "set-volume".
    1. Hint: base class pointer is used for both set value separately
4. All the values are required to be set through constructor's parameter.
    1. Provide necessary accessor functions where required.
    2. Create an object of class bus by initializing it through parameterized constructor in the main function and display
    3. All data members by calling display function of class bus.



5. Using static binding create two classes student and academic. The class academic is derived from class student. The two member function getdata and display are defined for both the classes. *obj is defined for class student, the address of which is stored in the object of the class academic.

Notes The functions getdata ( ) and display ( ) of student class are invoked by the pointer to the class