

Polymorphism

Ali Haider

syedalihaider.ciit@gmail.com

Department of Computer Science IUB

Overview

- Polymorphism
- Compile time polymorphism
- Runtime polymorphism
- Virtual Function
- Pure Virtual Functions

Polymorphism

- The word **polymorphism** means having many forms.
- Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.
- C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.
- In C++ polymorphism is mainly divided into two types:
 - Compile time Polymorphism
 - Runtime Polymorphism

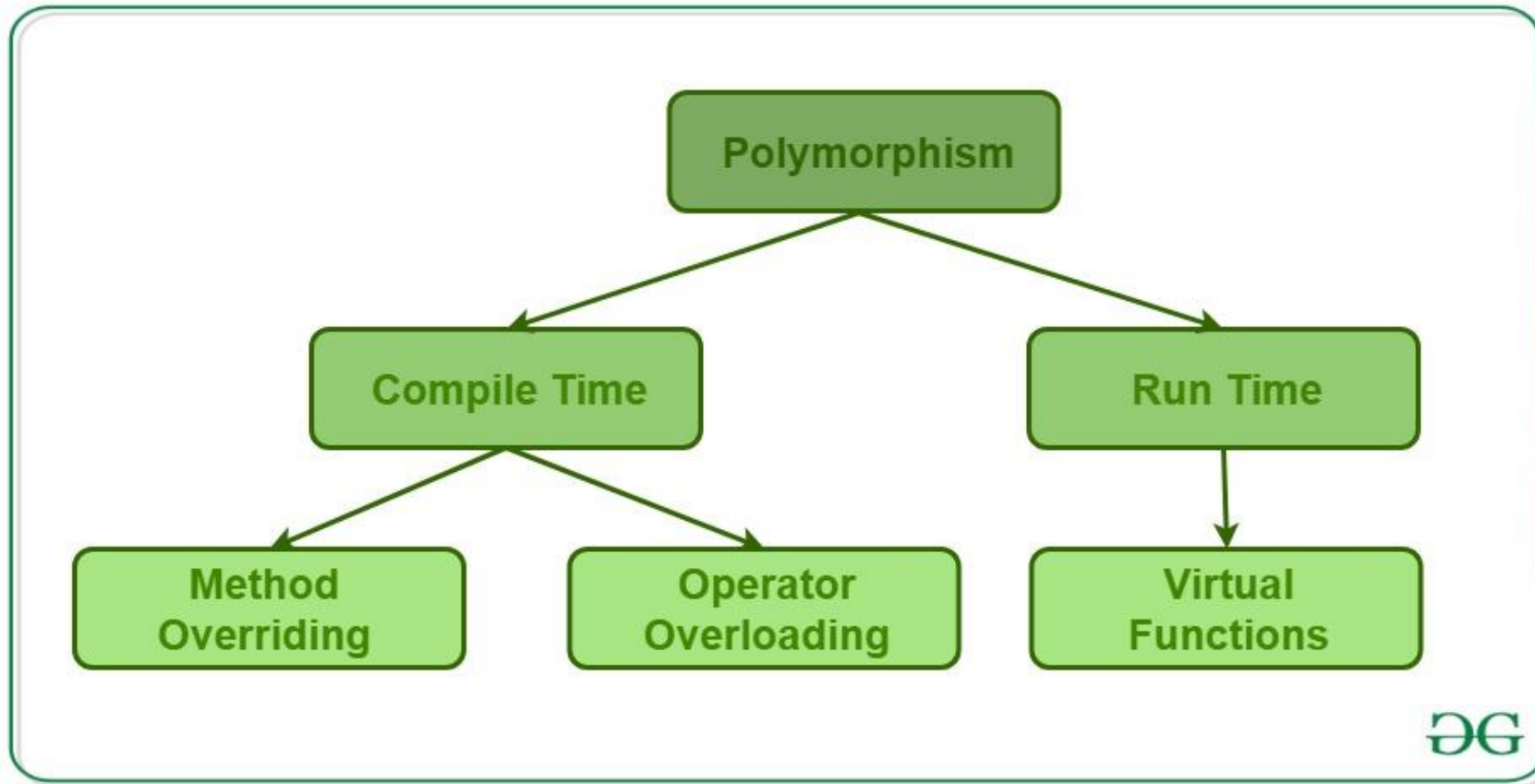
Compile time polymorphism

- This type of polymorphism is achieved by function overloading or operator overloading.
- **Function Overloading:** When there are multiple functions with same name but different parameters then these functions are said to be overloaded.
- Functions can be overloaded by change in number of arguments or/and change in type of arguments.
- **Operator Overloading:** C++ also provide option to overload operators.
- For example, we can make the operator ('+') for string class to concatenate two strings.
- We know that this is the addition operator whose task is to add two operands.
- So a single operator '+' when placed between integer operands , adds them and when placed between string operands, concatenates them.

Runtime polymorphism

- This type of polymorphism is achieved by Function Overriding.
- **Function overriding** on the other hand occurs when a derived class has a definition for one of the member functions of the base class.
- That base function is said to be overridden.

Cont...



Virtual Function

- A **virtual** function is a function in a base class that is declared using the keyword **virtual**.
- Defining in a base class a virtual function, with another version in a derived class, signals to the compiler that we don't want static linkage for this function.
- What we do want is the selection of the function to be called at any given point in the program to be based on the kind of object for which it is called.
- This sort of operation is referred to as **dynamic linkage**, or **late binding**.

Example-1

```
#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 () //print () is already virtual function in derived class
    { cout<< "print derived class" <<endl; }

    void show ()    { cout<< "show derived class" <<endl; }
};
//main function
int main() {
    base *bptr;
    derived d;
    bptr = &d;
    //virtual function, binded at runtime (Runtime polymorphism)
    bptr->print();
    // Non-virtual function, binded at compile time
    bptr->show();
    return 0; }
```

Example-2

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

class Shape {
protected:
    int width, height;

public:
    Shape( int a = 0, int b = 0){
        width = a;
        height = b;
    }
    virtual int area() {
        cout << "Parent class area :" <<endl;
        return 0;
    }
};

class Rectangle: public Shape {
public:
    Rectangle( int a = 0, int b = 0):Shape(a, b) { }

    int area () {
        cout << "Rectangle class area :" <<endl;
        return (width * height);
    }
};
```

1

```
class Triangle: public Shape {
public:
    Triangle( int a = 0, int b = 0):Shape(a, b) { }

    int area () {
        cout << "Triangle class area :" <<endl;
        return (width * height / 2);
    }
};

// Main function for the program
int main() {
    Shape *shape;
    Rectangle rec(10,7);
    Triangle tri(10,5);
    // store the address of Rectangle
    shape = &rec;
    // call rectangle area.
    shape->area();
    // store the address of Triangle
    shape = &tri;
    // call triangle area.
    shape->area();
    return 0;
}
```

2

Pure Virtual Functions

- It is possible that you want to include a virtual function in a base class so that it may be redefined in a derived class to suit the objects of that class, but that there is no meaningful definition you could give for the function in the base class.
- The = 0 tells the compiler that
- the function has no body and
- above virtual function will be called
- **pure virtual function.**

```
class Shape {
protected:
    int width, height;

public:
    Shape(int a = 0, int b = 0) {
        width = a;
        height = b;
    }

    // pure virtual function
    virtual int area() = 0;
};
```