MECHTRON 2MD3

Data Structures and Algorithms for Mechatronics

Winter 2022

08 C++ Inheritance and Polymorphism – with Examples

Department of Computing and Software

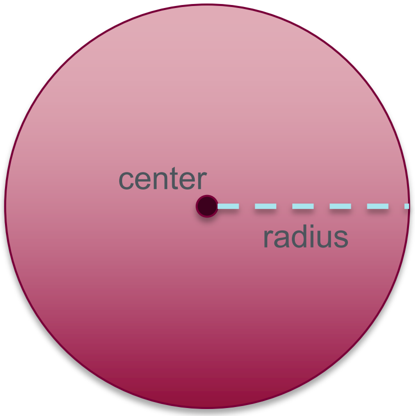
Instructor:

Omid Isfahanialamdari

January 31, 2022

Inheritance vs. Nested Classes

- Model real-life systems as close as possible



```
class circle {
        Point center;
        int radius;
        ...
};
```

Nested ✅

```
class circle : public Point {
        int radius;
        ...
};
```

Inheritance

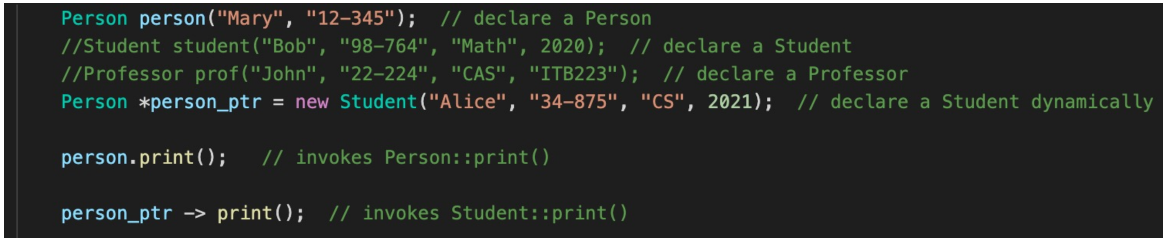
Nested	Has-a relation	A Circle has a Point
Inheritance	Is-a relation	A student is a Person

Inheritance

- Inheritance enables you to create a class that absorbs an existing class's capabilities, then customizes or enhances them. The existing class is called the **base class**, and the new class is referred to as the **derived class**.
- Every object of a derived class is also an object of that class's base class. However, a base-class object is not an object of derived classes.
- The **is-a** relationship represents inheritance. In an is-a relationship, an object of a derived class also can be treated as an object of its base class.
- Inheritance relationships form class hierarchies.
  - Person
    - Student
    - Professor

Inheritance and Polymorphism

- The ability for objects of different classes related by inheritance to respond differently to the same member function call.
- Polymorphism enables us to write programs that process objects of classes that are part of the same class hierarchy as if they were all objects of the hierarchy's base class.



```
Person person("Mary", "12-345");  // declare a Person
//Student student("Bob", "98-764", "Math", 2020);  // declare a Student
//Professor prof("John", "22-224", "CAS", "ITB223");  // declare a Professor
Person *person_ptr = new Student("Alice", "34-875", "CS", 2021);  // declare a Student dynamically

person.print();   // invokes Person::print()

person_ptr -> print();  // invokes Student::print()
```

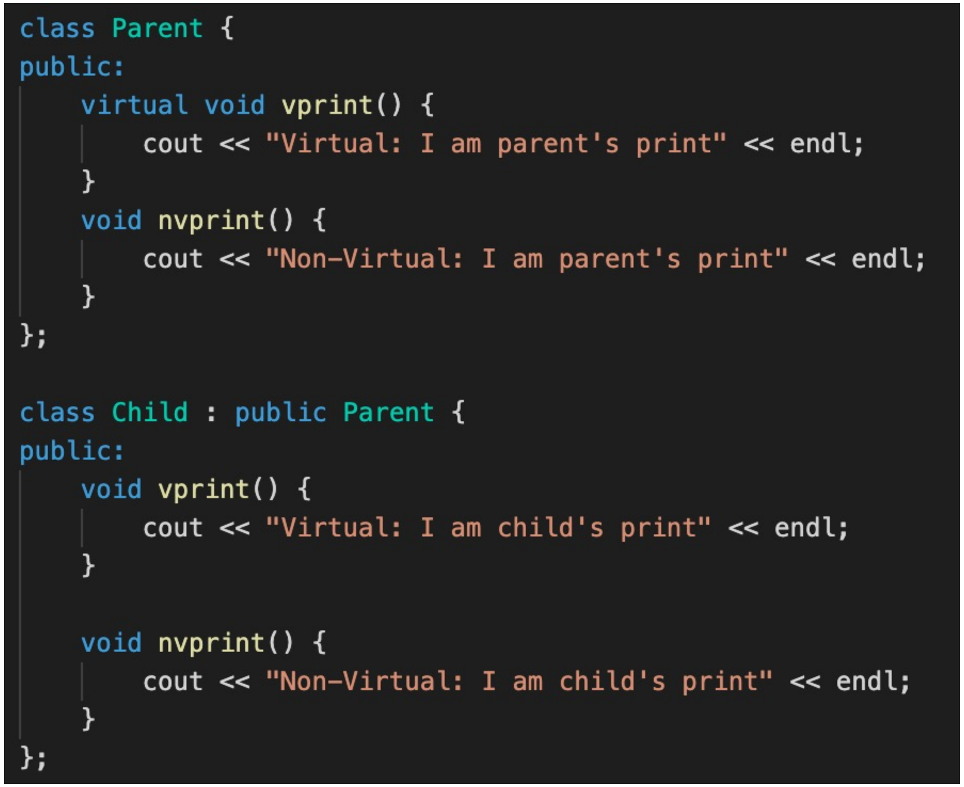
- With polymorphism, one function call can cause different actions to occur, depending on the type of the object on which the function is invoked.

Inheritance and Polymorphism

- Programs can be written to process objects of types that may not exist when the program is under development (**run-time polymorphism**)
- With Polymorphism, we can design and implement systems that are easily extensible
  - new classes can be added with little or no modification to the general portions of the program.
  - The only parts of a program that must be altered to accommodate new classes are those that require direct knowledge of the new classes that you add to the hierarchy.
- Polymorphism is implemented via **virtual functions** and **dynamic binding**

Recall: Virtual vs Non-Virtual Functions

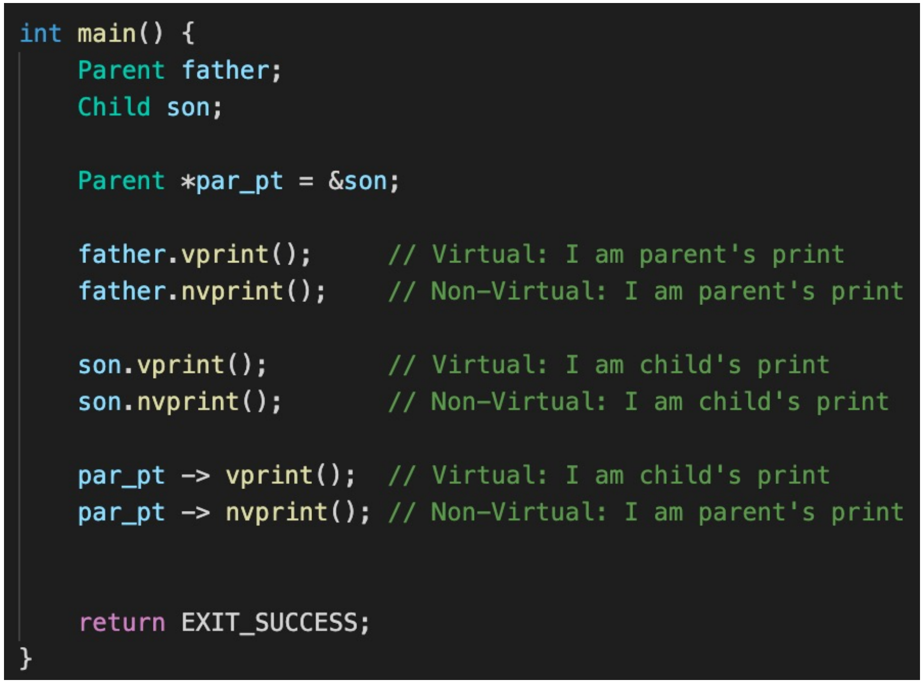
- Dynamic (run-time) binding vs Static (compile-time) binding



```cpp
class Parent {
public:
    virtual void vprint() {
        cout << "Virtual: I am parent's print" << endl;
    }
    void nvprint() {
        cout << "Non-Virtual: I am parent's print" << endl;
    }
};

class Child : public Parent {
public:
    void vprint() {
        cout << "Virtual: I am child's print" << endl;
    }

    void nvprint() {
        cout << "Non-Virtual: I am child's print" << endl;
    }
};
```



```cpp
int main() {
    Parent father;
    Child son;

    Parent *par_pt = &son;

    father.vprint();     // Virtual: I am parent's print
    father.nvprint();    // Non-Virtual: I am parent's print

    son.vprint();        // Virtual: I am child's print
    son.nvprint();       // Non-Virtual: I am child's print

    par_pt -> vprint();  // Virtual: I am child's print
    par_pt -> nvprint(); // Non-Virtual: I am parent's print


    return EXIT_SUCCESS;
}
```

Output:

```
Virtual: I am parent's print
Non-Virtual: I am parent's print
Virtual: I am child's print
Non-Virtual: I am child's print
Virtual: I am child's print
Non-Virtual: I am parent's print
```

See the code

- People in University code
  - Available in General > Class Materials > Lecture Slides > code
  - uni-example.cpp

- We will see how virtual functions realize polymorphism in C++

Review Progression Code

- Numerical Progression example code:
  - Available in General > Class Materials > Lecture Slides > code
  - progression.cpp

- Arithmetic progression (increment 1) 0,1,2,3,4,5,...

- Arithmetic progression (increment 3) 0,3,6,9,12,...

- Geometric progression (base 2) 1,2,4,8,16,32,...

- Geometric progression (base 3) 1,3,9,27,81,...

- Fibonacci progression (first = 0, second = 1) 0,1,1,2,3,5,8,...

Questions?