Object-Oriented Programming

Misc. Concepts

C++ Misc. Concepts

- Nested class
- Abstract class
- Static members
- Final specifier
- Friends

Object-Oriented Programming

Nested Class

Nested Classes

- A nested class is declared inside another enclosing class.
- A nested class is a member of the enclosing class therefore it has access rights as the other class members.

```
class Enclosing {
   private:
       int x;

   /* start of Nested class declaration */
   class Nested {
      int y;
      void NestedFun(Enclosing *e) {
        cout<<e->x;  // works fine: nested class can access
                     // private members of Enclosing class
      }
   }; // declaration Nested class ends here
}; // declaration Enclosing class ends here
```

Object-Oriented Programming

Abstract Class

Abstract Classes

- Abstract classes are used *exclusively* as **base classes** for other classes.
- You cannot create an instance of abstract classes.

- You create an abstract class by defining at least one **pure virtual member function**.
- Classes that derive from abstract classes MUST implement the pure virtual function or they will be abstract too.

Abstract Classes

```
//abstract base class
class Weapon
{
public:
    virtual int calcDamage() = 0;//pure virtual
};


class Grenade : public Weapon
{
public:
    //required to implement
    virtual int calcDamage()
    {}
};
```

Object-Oriented Programming
Static

Static keyword

- The static keyword can be used in different scenarios…
  - Static variables in a function
  - Static class members

Object-Oriented Programming

Static Variables

Static variables

- When a variable in a function is marked static, that means
  - the variable is allocated once, even if the function is called multiple times
  - remains in memory for the lifetime of the application

```cpp
void demo()
{
    static int count = 0; //created once
    std::cout << count << " ";
    count++;
}

int main()
{
    for (size_t i = 0; i < 10; i++)
       demo();

    //prints 0 1 2 3 4 5 6 7 8 9
}
```

Object-Oriented Programming

Static Class Members

Static Members

- Data members and member functions can be marked with the **static** keyword.

Static Data Members

- If a **data member (field)** is marked as static, then the variable is created once and remains in memory for the lifetime of the application.
- The static data members are ***shared*** by all instances of the class.

```
class Car
{
public:
  Car(int year)
  {
      mModelYear = year;
      mNumberOfCarsMade++;
  }

  //each car has its own model year variable
  int mModelYear;

  //shared by ALL cars
  static int mNumberOfCarsMade;
};

//initialize with class name scoping
int Car::mNumberOfCarsMade = 0;
```

Static Member Functions

- Static methods can *only access static members* of the class.

```
static void reporting()
{
    std::cout << "Model year: " << mModelYear << "\n"; //ERROR! cannot access non-static members
    std::cout << "Number of cars made: " << mNumberOfCarsMade << "\n";
}
```

Non-Static Member Functions

- Non-Static methods can access static and non-static members of the class.
- Non-static methods have a hidden parameter called 'this'. It points to the object that the method was called on.
- Only use 'this->' in your method to eliminate ambiguity.

```
void vehicleInfo() //there's a hidden parameter called 'this'
{
    std::cout << "Model Year: " << this->mModelYear << "\n";
}
```

Object-Oriented Programming

Final Specifier

Final

- Final specifies that…
  - a virtual function cannot be overridden in a derived class or
  - that a class cannot be derived from.

Final Virtual Functions

- Typically, final is not used on base class virtual functions.
- You would mark a derived class override as final so that classes further down the hierarchy can’t override anymore.
- Sometimes you mark a base virtual function as final so that someone doesn’t accidentally ‘hide’ the method by defining a new one in a derived class.

```
class rocketEngine : public engine
{
public:
    virtual void ApplyThrust() final
    { }
};
```

Final Classes

- A final class prevents other classes from inheriting from it.
- Why? The compiler can optimize the code and remove virtualizations which would gain some performance benefits at runtime.

```
class rocketEngine final
{
};

class carEngine : public rocketEngine  //BUILD ERROR!
{
};
```

Object-Oriented Programming
Friends

Friend function

- A friend function can access the private and protected members of a class.
- A friend function can be:
  - A global function
  - A member function of another class

Friend function

- A global friend function

```
class Box
{
private:
    int width, height;
public:
    //grant render function access
    //to my private members
    friend void render(Box& box);
};
```

```
void render(Box& box)
{
    //accessing the private fields of box
    if (box.width > 0 && box.height > 0)
    { }
}
```

Friend function

- A friend function from another class…

```
class Box
{
private:
    int width, height;
public:
    //grant GraphicsEngine::Draw function
    //access to my private members
    friend void GraphicsEngine::Draw(Box& box);
};
```

```
class GraphicsEngine
{
public:
    void Draw(Box& box);
};

void GraphicsEngine::Draw(Box& box)
{
    //accessing the private fields of box
    if (box.width > 0 && box.height > 0)
    { }
}
```