

# Object-oriented Programming

**Inline Functions |**

**Chained Function Calls**

# Inline Function

- Placing the qualifier inline before a function's return type in definition **“advises”** the compiler to generate a copy of the function's code in place **(when appropriate)** to avoid a function call
- Compiler ignores this request unless the function does not have too much code
- Class member functions are implicitly inline

# Inline Function

```
inline void square(int a)
{
    cout << "Square of given number is: " << a * a;
}
```

```
int main()
{
    int a = 2;
    square(a);
}
```

*// This function call is likely to be replaced by code in the function's body*

# Inline Functions

- Compiler **does not** perform inlining when:
  - 1) If a function contains a loop
  - 2) If a function contains static variables
  - 3) If a function is recursive
  - 4) If a function return type is other than void, and the return statement doesn't exist in function body
  - 5) If a function contains switch or goto statement

# Advantages of Inline Functions

- 1) Function call overhead doesn't occur
- 2) It also saves overhead of a return call from a function
- 3) Inline function may be useful (if it is small) for embedded systems because inline can yield less code than the function call preamble and return



# Disadvantages of Inline Functions

- 1) The added variables from the inline function consumes additional registers
- 2) If you use too many inline functions then the size of the binary executable file will be large, because of the duplication of same code
- 3) Inline function may increase compile time overhead if someone changes the code inside the inline function then all the calling location has to be recompiled



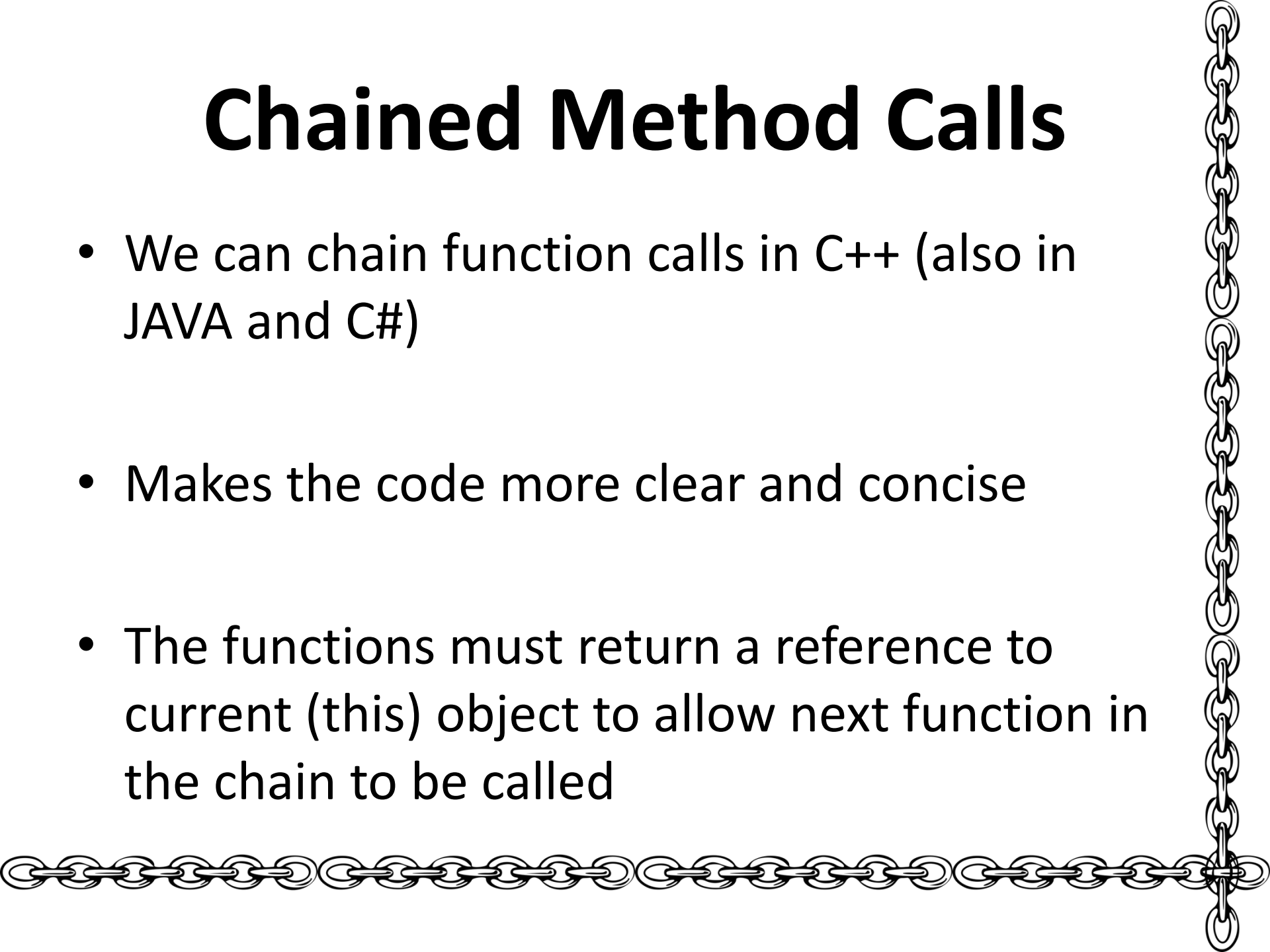
# Exploring *this* keyword

- this contains reference the current object, i.e. an object that is being active for the current call
- Can be used to identify class members



# Chained Method Calls

- We can chain function calls in C++ (also in JAVA and C#)
- Makes the code more clear and concise
- The functions must return a reference to current (this) object to allow next function in the chain to be called





# Chained Method Calls

```
class A
{
    int x, y;
    public:
    A( ) { }

    A& setX(int a)
    {
        x = a;
        return *this;
    }
    A& setY(int b)
    {
        y = b;
        return *this;
    }
};
```

```
int main( )
{
    A ob1;
    ob1.setX(5).setY(10);
    ob1.setY(100).setX(200);
}
```

# Chained Method Calls

- What if we remove reference from return type of the function?
  - *The function returns a temporary object instead of the current object!*
  - *As a result, the next call in the chain will be made through that temporary object*

