More About Classes

By: **Annamalai A**

`this` Pointer

- A special pointer that is passed as a parameter to each member function
- It holds the address of the object that invoked it.
- It distinguishes between local variables (like function parameters) and class data members when they have the same name.
- **Example**: `this->real = real;` (where `this->real` is the class member and `real` is the parameter).

Example:

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

class Person {
    int age;
public:
    void setAge(int age) {
        // 'this->age' refers to the class member, 'age' r
efers to the local parameter
        this->age = age;
    }
    void show() { cout << "Age: " << age << endl; }
};

int main() {
    Person p;
    p.setAge(20);
    p.show();
```

```
    return 0;
}
```

Friend Functions and Classes

- **Friend Class**: Declared inside another class using the `friend` keyword, allowing the friend class to access both `private` and `protected` members of the class where it is declared.
- **Friend Function**: A function that is not a member of a class but is granted access to its `private` and `protected` members.
  - **Global Friend**: A global function can be declared as a friend inside a class.
  - **Member Friend**: A member function of one class can be declared as a friend of another.
  - **Bridging Classes**: Friend functions can act as a "bridge" to access data from multiple unrelated classes simultaneously.

  Example:

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

class Base {
private:
    int secret = 100;
    friend class RemoteControl; // Friend Class declaratio
n
    friend void reveal(Base b); // Friend Function declara
tion
};

void reveal(Base b) {
    cout << "Friend Function access: " << b.secret << end
l;
}
```

```
class RemoteControl {
public:
    void access(Base b) {
        cout << "Friend Class access: " << b.secret << end
l;
    }
};

int main() {
    Base obj;
    reveal(obj);
    RemoteControl rc;
    rc.access(obj);
    return 0;
}
```

`const` Objects and Member Functions

- `const` **Object**: An instance of a class whose data members cannot be modified after the constructor finishes executing.
  - **Syntax**: `const Square square1(6, 6);`
- `const` **Member Function**: Guarentees it will not modify any data members of the object in the member function.
  - **Syntax**: `int getLength() const { return length; }`

Example:

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

class Value {
    int val;
public:
```

```
    Value(int v) : val(v) {}
    // const member function
    void display() const {
        cout << "Value: " << val << endl;
    }
};

int main() {
    const Value obj(10); // const object
    obj.display();       // Can only call const functions
    return 0;
}
```

`static` Class Member

- `static`: A keyword that says it does not differ based on the object. It has nothing to do with the `this` pointer
- `static` **Data Member**: Shared by all objects in the class. Only one copy exists in memory.
- `static` **Data Function**: Can be called without an object using the class name and can only access other static member

Example:

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

class Counter {
public:
    static int count; // Static data member
    Counter() { count++; }
    static void showCount() { // Static member function
        cout << "Objects created: " << count << endl;
    }
```

```
};

int Counter::count = 0; // Initialization outside the clas
s

int main() {
    Counter c1, c2, c3;
    Counter::showCount(); // Called using class name
    return 0;
}
```