

# Object Oriented Programming

## Lecture 7

Dr. Naveed Anwar Bhatti

Webpage: [naveedanwarbhatti.github.io](https://naveedanwarbhatti.github.io)

# *Static* variables



- Lifetime of static variable is **throughout the program life**
- They are declared and initialized **only once**
- If static variables are not explicitly initialized then they are **initialized to 0** of appropriate type



## Example

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

void func1(int i)
{
    static int staticInt = i;
    cout << staticInt << endl;
}

int main()
{
    func1(1);
    func1(2);
}
```

Output:

```
1
1
```



## Example (Default Values)

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

void func1(int i)
{
    static int staticInt;
    cout << staticInt << endl;
}

int main()
{
    func1(1);
    func1(2);
}
```

Output:

```
0
0
```



## Example (Static vs Const)

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

void func1(int i)
{
    static int staticInt;
    staticInt = i;
    cout << staticInt << endl;
}

int main()
{
    func1(1);
    func1(2);
}
```

Static variable is **not a constant variable**. Their values can be changed

Output:

1  
2



## Example (Life vs Access)

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

void func1(int i)
{
    static int staticInt;
    staticInt = i;
    cout << staticInt << endl;
}

int main()
{
    func1(1);
    func1(2);
    cout << staticInt;
}
```

Output:  
Error

Lifetime is throughout the program life.  
**But access is limited to function where they are defined**

## Definition

“A variable that is part of a class, yet is not part of an object of that class, **is called static data member**”

Also known as *Class Variable*



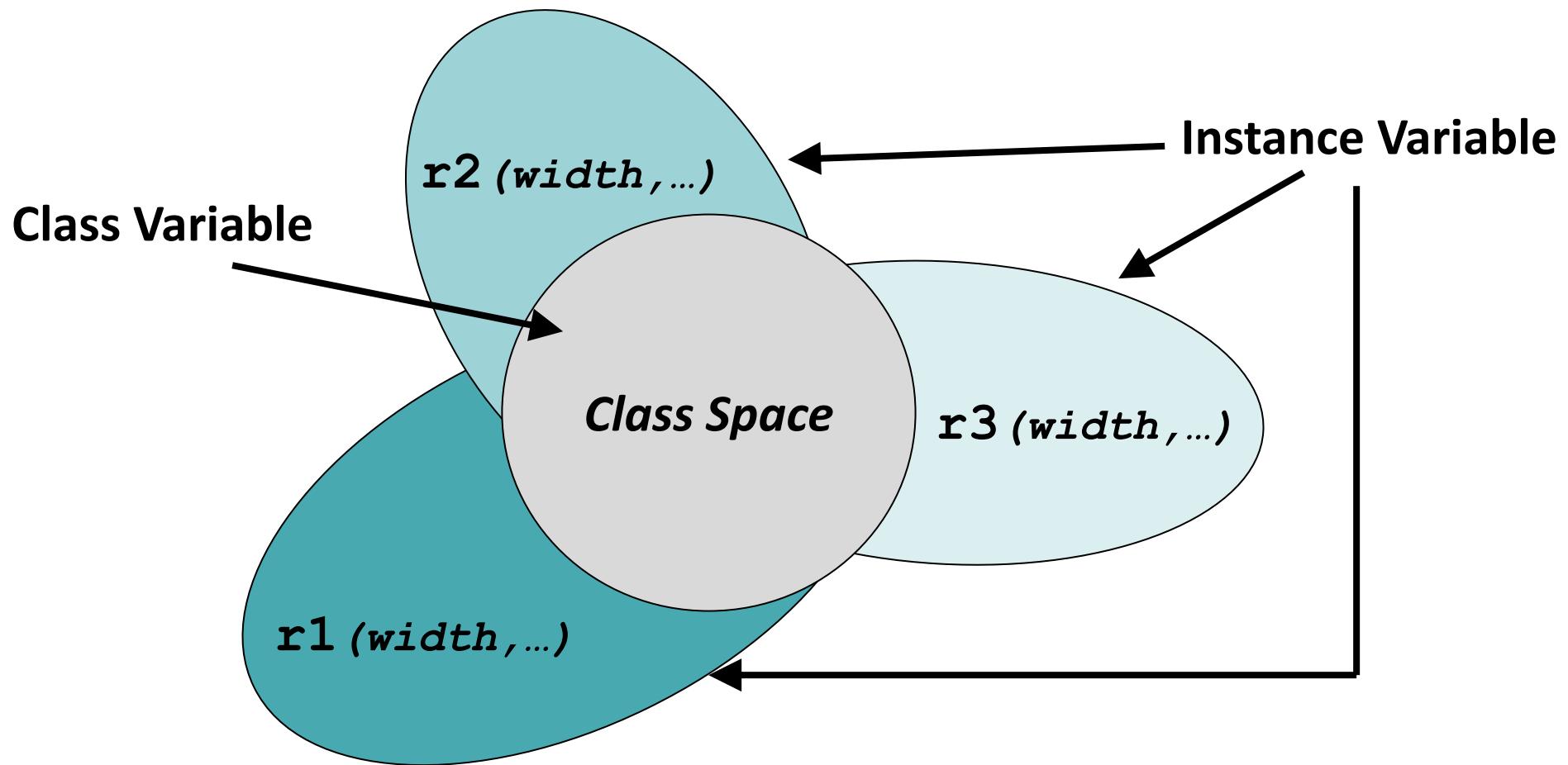
## Static Data Member

- They are shared by all instances of the class
- They do not belong to any particular instance of a class



## Class Variable vs. Instance Variable

- **Rectangle r1, r2, r3;**





## Static Data Member (Syntax)

- Keyword **static** is used to make a data member static

```
class ClassName{  
...  
static DataType VariableName;  
};
```



## Defining Static Data Member

- Static data member is declared inside the class
- But they are defined outside the class



## Example

```
class Rectangle{  
private:  
    static int noOfRectangles;  
public:  
    ...  
};  
int Rectangle::noOfRectangles = 0;  
/*private static member cannot be accessed  
outside the class except for initialization*/
```



## Initializing Static Data Member

- If static data members are not explicitly initialized at the time of definition then they are initialized to 0



## Example

```
int Rectangle::noOfRectangles ;
```

is equivalent to

```
int Rectangle::noOfRectangles =0;
```



## Accessing Static Data Member

---

- To access a static data member there are two ways
  - Access like a normal data member
  - Access using a scope resolution operator '::'



## Example

```
class Rectangle
{
    int width, height;
public:
    static int NoOfRectangles;
    Rectangle();
};

int Rectangle::NoOfRectangles=0;

int main()
{
    cout << Rectangle::NoOfRectangles;

    Rectangle r1;
    cout << r1.NoOfRectangles;
}
```

Access Method 1

Access Method 2



## Life of Static Data Member

- They are created even when there is no object of a class
- They remain in memory even when all objects of a class are destroyed



## Example

```
class Rectangle
{
    int width, height;
public:
    static int NoOfRectangles;
    Rectangle();
};

int Rectangle::NoOfRectangles=0;

int main()
{
    cout << Rectangle::NoOfRectangles;
}

}
```



## Uses

- They can be used to store information that is required by all objects, like global variables



## Exercise

- Modify the class ***Rectangle*** such that one can know the number of rectangles created in a system



## Example

```
class Rectangle
{
    int width, height;
public:
    static int NoOfRectangles;
    Rectangle();
    ~Rectangle();
};
```

```
Rectangle::Rectangle()
{
    NoOfRectangles++;
}
```

```
Rectangle::~Rectangle()
{
    NoOfRectangles--;
}
```



## Example

```
int main()
{
    cout << Rectangle::NoOfRectangles;

    Rectangle r1;

    cout << Rectangle::NoOfRectangles;

    Rectangle r2;

    cout << Rectangle::NoOfRectangles;

}
```

Output:

```
0  
1  
2
```



## Problem

- *noOfRectangles* is accessible outside the class
- Bad design as the local data member is kept public



### Definition:

“The function that needs access to the members of a class, yet does not need to be invoked by a particular object, is called **static member function**”



## Static Member Function

- They are used to **access static data members**
- Access mechanism for static member functions is same as that of static data members
- They **cannot** access any **non-static members**



## Example

```
class Rectangle
{
    int width, height;
    static int NoOfRectangles;
public:

    Rectangle(){};
    ~Rectangle(){};

    static int getTotalRectangles()
    {
        return NoOfRectangles;
    }

};

int Rectangle::NoOfRectangles;
```

```
int main()
{
    cout << Rectangle::getTotalStudent();
    Rectangle r1;
    cout << Rectangle::getTotalStudent();
}
```



# Accessing non static data members

```
class Rectangle
{
    int width, height;
    static int NoOfRectangles;
public:

    Rectangle() {};
    ~Rectangle() {};

    static int getTotalRectangles()
    {
        return width;
    }
};

int Rectangle::NoOfRectangles;
```

```
int main()
{
    cout << Rectangle::getTotalStudent();
    Rectangle r1;
    cout << Rectangle::getTotalStudent();
}
```

Error: Can **ONLY**  
access static data



## this Pointer

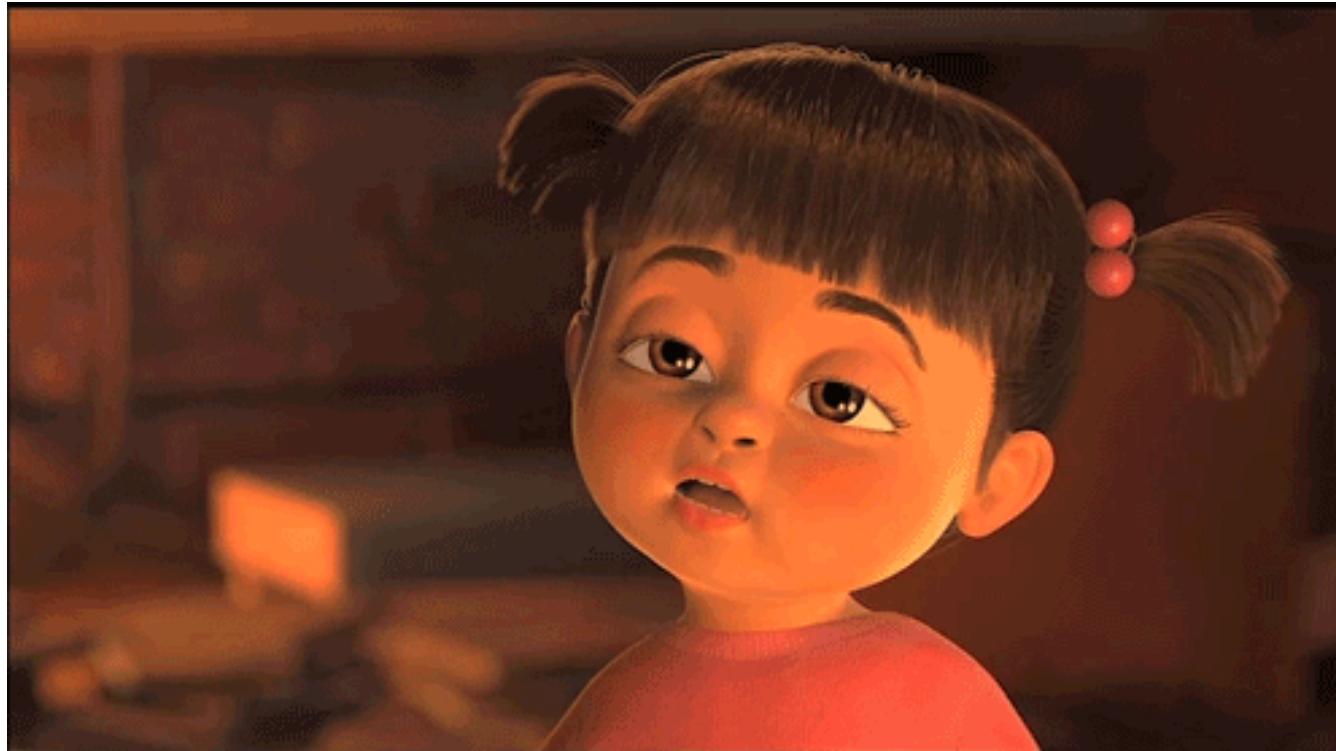
- *this* pointer is passed implicitly to member functions
- *this* pointer is **not passed** to static member functions
- Reason is static member functions cannot access non static data members



# Global Variable vs. Static Members

- Alternative to static member is to use global variable
- Global variables are accessible to all entities of the program
  - Against information hiding

# Thanks a lot



If you are taking a Nap, **wake up.....Lecture Over**