Data Structures and Object Oriented Programming

Lecture 24

Dr. Naveed Anwar Bhatti

Webpage: naveedanwarbhatti.github.io

Class Templates

Motivation for Class Templates

- Consider almost identical complex number classes:

```
class ComplexDouble {
private:
    double x, y; // real and imaginary parts
public:
    ComplexDouble(double a = 0.0, double b = 0.0) : x(a), y(b) {}
    double real() { return x; }
    double imag() { return y; }
};
```

```
class ComplexFloat {
 private:
     float x, y; // real and imaginary parts
 public:
     ComplexFloat(float a = 0.0, float b = 0.0) : x(a), y(b) {}
     float real(){ return x; }
     float imag(){ return y; }
};
```

Again, would be nice if we did not have to repeatedly type, debug, test, and maintain nearly identical code

Class Templates

- Like function template, a class template is a common class that can represent various similar classes operating on data of different types.

- Once a class template is defined, we can create an object of that class using a specific basic or user-defined data types to replace the generic data types used during class definition.

Syntax of Class Template

```
template <class T>
class classname
{
   attributes;
   methods;
};
```

Example – Class Template

```
template< class T >
class Complex {
private:
    T x; // real part
    T y; // imaginary part

public:
    Complex(T a = T(0), T b = T(0)) : x(a), y(b) {}
    T real() { return x; }
    T imag() { return y; }

};

 Complex <int> zi;
 Complex <double> zd;
```

Another Example – Class Template

```
template<class T1, class T2>
class sample {
private
    T1 a; T2 b;
public:
    void getdata()
    {
       cout << "Enter a and b: "<<endl;
       cin>> a >> b;
    }
    void display()
    {
       cout<<"Displaying values"<<endl;
       cout<<"a = "<< a << endl;
       cout<<"b = "<< b << endl;
    }
};
```

```
int main() {

sample<int, int> s1;
sample<int, char> s2;
sample<int, float> s3;

cout << "Two Integer data" << endl;
s1.getdata();
s1.display();

cout << "Integer and Character data" << endl;
s2.getdata();
s2.display();

cout << "Integer and Float data" << endl;
s3.getdata();
s3.display();

return 0;
}
```

Member function definition outside class

```
template<class T1, class T2>
class sample {
private:
    T1 a; T2 b;
public:
    void getdata();
    void display();
};

void sample::getdata()
{
    cout << "Enter a and b: " << endl;
    cin >> a >> b;
}
```

Error: parameter list is missing

```
void sample::display()
{
    cout << "Displaying values" << endl;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
}
```

Error: parameter list is missing

Member function definition outside class

```
template<class T1, class T2>
class sample {
private:
    T1 a; T2 b;
public:
    void getdata();
    void display();
};

template<class T1, class T2>
void sample<T1,T2>::getdata()
{
    cout << "Enter a and b: " << endl;
    cin >> a >> b;
}

template<class T1, class T2>
void sample<T1, T2>::display()
{
    cout << "Displaying values" << endl;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
}
```

Default Type Parameters

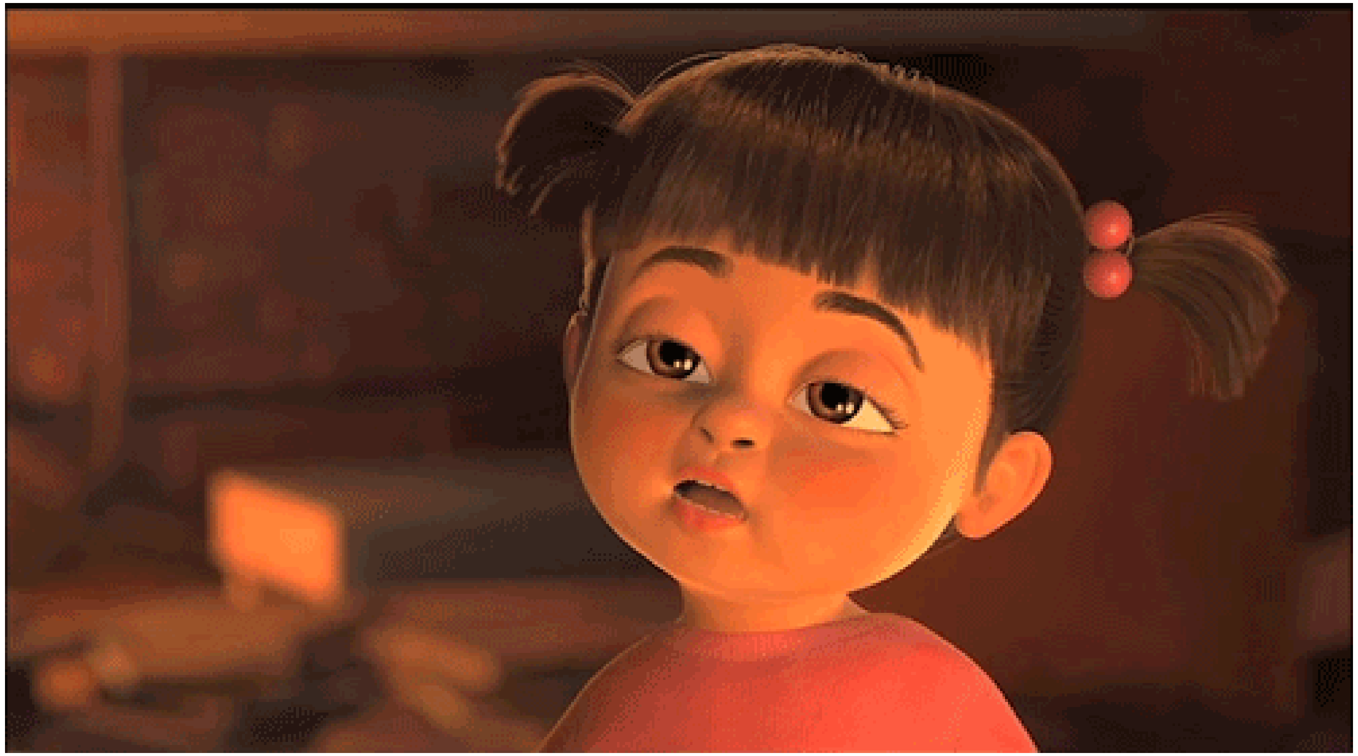
- A type parameter can specify a default type

- Example

```
template<class T = int>
class MyArray
{
    T data ;
};

MyArray <> a; // MyArray<int>
MyArray <double> b; // MyArray<double>
```

Thanks a lot



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