OLQ13

Practice and Review

Dynamic Casting

```
for (auto it : MyShapes)
{
   it->Hello();
   cout << it->getName() << "'s area is " << it->getarea() << endl;

   Circle* IamaCircle = dynamic_cast<Circle*>(it);
   if (IamaCircle != nullptr)
   {
      cout << "My diameter is " << IamaCircle->getDiameter() << endl;
   }
}
```

`dynamic_cast` returns a value of `nullptr` if the input object is not of the requested type. In this for loop, `dynamic_cast` will not be equal to `nullptr` when it is a `Circle`. `IamaCircle` is then a pointer to `Circle` that can call `Circle`'s member function `getDiameter()`.

For the quiz, be able to add a member function to a derived class and be able to call it even when utilizing polymorphism.

Class Templates

Review the example given in the slides. Here is another example.

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

template <class T>
class Test
{
      private:
            T count{0};
      public:
            Test()
            {
                  count++;
            }
            T getcount()
            {
                  return count;
            }
};

int main(void)
```

```
{
      Test<int> a;
      Test<double> c;

      cout << a.getcount() << endl;
      cout << c.getcount() << endl;

      return 0;
}
```

The class templates specializations that the compiler would create would be

```
class Test
{
   private:
      int count{0};
   public:
      Test()
      {
         count++;
      }
      int getcount()
      {
         return count;
      }
};
```

```
class Test
{
   private:
      double count{0};
   public:
      Test()
      {
         count++;
      }
      double getcount()
      {
         return count;
      }
};
```

You should be able to create a class template given a class and you should be able to show/list the class template specializations given a class template.