Exercises week 2

Klaas Isaac Bijlsma
s2394480

David Vroom
s2309939

November 26, 2017

Exercise 11

Learn to appreciate catching references when throwing exceptions

A simple class `Object` is made. It has a data member `d_name` that stores an internal name. If an object is made via the copy constructor, 'copy' is added to this internal name. The constructor, copy constructor and destructor print what they did together with the internal name. A function `hello()` is added that says hello and prints the internal name.

object/object.h

```
#ifndef INCLUDED_OBJECT_
#define INCLUDED_OBJECT_

#include <string>

class Object
{
    std::string d_name;

    public:
        Object(std::string const &name);        // 1
        Object(Object const &other);            // 2
        ~Object();
        void hello();
};

#endif
```

object/object.ih

```
#include "object.h"
#include <iostream>
#include <string>

using namespace std;
```

object/destructor.cc

```
#include "object.ih"

Object::~Object()
{
    cout << "Destructed '" << d_name << "'\n";
}
```

object/hello.cc

```
#include "object.ih"

void Object::hello()
{
    cout << "Hello, this is '" << d_name << "'\n";
}
```

object/object1.cc

```
#include "object.ih"

Object::Object(string const &name)
:
    d_name(name)
{
    cout << "Constructed '" << d_name << "'\n";
}
```

object/object2.cc

```
#include "object.ih"

Object::Object(Object const &other)
:
    d_name(other.d_name + " (copy)")
{
    cout << "Copy constructed '" << d_name << "'\n";
}
```

Below a main function (`main1.cc`) is shown, in which within a try block, an object of the class `Object` is made. This object is then thrown. The exception handler catches an object of the class `Object` (by value). The output of the program is given below the code of main1. We see that the object is properly constructed and says hello. Then when it is thrown, first a copy is made and the original object is destructed. The copy is passed to the exception handler. Here an additional copy is made, because it receives the object by value. Therefore, within the exception handler, the copy of the copy of the object says hello.

main1.cc

```
#include "object/object.h"
#include <iostream>
using namespace std;

int main()
{
    try
    {
        Object object{ "object" };
        object.hello();
        throw object;
    }
    catch (Object caughtObject)
    {
        cout << "Caught exception\n";
        caughtObject.hello();
    }
}
```

Output of main1.cc

```
Constructed 'object'
Hello, this is 'object'
Copy constructed 'object (copy)'
Destructed 'object'
Copy constructed 'object (copy) (copy)'
Caught exception
Hello, this is 'object (copy) (copy)'
Destructed 'object (copy) (copy)'
Destructed 'object (copy)'
```

The following main function (`main2.cc`) does the same as the previous, except that the exception handler catches *a reference* to an object of the class `Object`. From the output we see that no second copy is made. This is more efficient and therefore exception handlers should catch references to objects.

main2.cc

```cpp
#include "object/object.h"
#include <iostream>
using namespace std;

int main()
{
    try
    {
        Object object{ "object" };
        object.hello();
        throw object;
    }
    catch (Object &caughtObject)
    {
        cout << "Caught exception\n";
        caughtObject.hello();
    }
}
```

Output of main2.cc

```
Constructed 'object'
Hello, this is 'object'
Copy constructed 'object (copy)'
Destructed 'object'
Caught exception
Hello, this is 'object (copy)'
Destructed 'object (copy)'
```

In the previous two programs, we saw that a copy of the object is thrown. This is because the original object is a local object that only lives inside the try block. The same is true when a reference to an object is thrown, as can be seen from the output that the following code produces:

main3.cc

```
#include "object/object.h"
#include <iostream>
using namespace std;

int main()
{
    try
    {
        Object object{ "object" };

        Object &ref = object;
        ref.hello();
        throw ref;
    }
    catch (Object &caughtObject)
    {
        cout << "Caught exception\n";
        caughtObject.hello();
    }
}
```

Output of main3.cc

```
Constructed 'object'
Hello, this is 'object'
Copy constructed 'object (copy)'
Destructed 'object'
Caught exception
Hello, this is 'object (copy)'
Destructed 'object (copy)'
```

The following main function (`main4.cc`) has two exception levels. In the inner level, an object of the class `Object` is thrown and caught as a reference. Then it is rethrown to a more shallow level where it is again caught as a reference. From the shown output, we conclude that '`throw;`' results in throwing the currently available exception and not a copy of that exception.

main4.cc

```
#include "object/object.h"
#include <iostream>
using namespace std;

int main()
{
    try
    {
        try
        {
            Object object{ "object" };
            object.hello();
            throw object;
        }
        catch (Object &caughtObject)
        {
            cout << "Caught exception in inner block\n";
            caughtObject.hello();
            throw;
        }
    }
    catch (Object &caughtObject)
    {
        cout << "Caught exception in outer block\n";
        caughtObject.hello();
    }
}
```

Output of main4.cc

```
Constructed 'object'
Hello, this is 'object'
Copy constructed 'object (copy)'
Destructed 'object'
Caught exception in inner block
Hello, this is 'object (copy)'
Caught exception in outer block
Hello, this is 'object (copy)'
Destructed 'object (copy)'
```

Exercise 12

Exercise 13

Exercise 14

Exercise 15

Exercise 16

Exercise 17

Exercise 18

Exercise 19