Exercises for the Lecture Series

"Object-Oriented Programming for Scientific Computing"

Ole Klein
ole.klein@iwr.uni-heidelberg.de

To be handed in on 19. 05. 2015 before the lecture

EXERCISE 1 EXCEPTIONS AND DESTRUCTORS

What happens when a destructor throws an exception? Let's look at the program to the right.

- Augment the example with the definition of the class `my_exception`, which is derived from `std::exception`. The class `my_exception` should receive an error message in a `std::string` in its constructor. The virtual method `what()` should return this error message.
- What do you observe when executing the program?
- Try to explain the behavior.

```
1  #include <iostream>
2  #include <string>
3  #include <exception>
4
5  // class Foo throws in the destructor
6  class Foo {
7  public:
8      ~Foo () {
9          throw my_exception("Foo exception");
10     }
11 };
12
13 // class Bar throws in the constructor
14 class Bar {
15 public:
16     Bar () {
17         throw my_exception("Bar exception");
18     }
19 };
20
21 int main()
22 try {
23     Foo f;
24     Bar b;
25 }
26 catch (const std::exception & e){
27     std::cout << "ERROR:" << e.what() << std::endl;
28 }
```

- Section 15.2, point 3 of the C++ standard:

  3 The process of calling destructors for automatic objects constructed on the path from a try block to a throw-expression is called "stack *unwinding*." [Note: If a destructor called during stack unwinding exits with an exception, terminate is called (15.5.1). So destructors should generally catch exceptions and not let them propagate out of the destructor. —end note]

  Why is the behavior detailed in the *note* sensible?

Further reading: a collection of the possible options when destructors and exceptions meet can be found under `http://www.kolpackov.net/projects/c++/eh/dtor-1.xhtml` *8 Points*

EXERCISE 2 EXCEPTIONS AND SANITY CHECKS

The class `NumVector` on the last exercise sheet did not do bounds checking, i.e. it was possible to access indices smaller than zero or larger than the largest index. Modify the method `operator[]`, so that erroneous access results in an exception being thrown (comparable to the behavior of the method `std::vector<T>::at` instead of `std::vector<T>::operator[]`).

Also implement a method `operator*` that calculates the scalar product of two vectors. Exceptions that may result from incompatible lengths should be caught and a new, more detailed exception should be thrown.

The exceptions should be classes written by you and have distinct types. Write a program that tests both exceptions and prints a detailed error message in the case of errors without terminating the program.

8 Points

EXERCISE 3 FATHER AND SON

What do you think the Java code on the right will do? Translate it into equivalent C++ code (`extends` corresponds to `public` inheritance and `Throwable` to an exception).

Comment your program and write down in which order the lines are executed. Is this a valid C++ program? If yes, what does it do?

```
CLASS BALL EXTENDS THROWABLE {}
CLASS P{
  P TARGET;
  P(P TARGET) {
    THIS.TARGET = TARGET;
  }
  VOID AIM(BALL BALL) {
    TRY {
      THROW BALL;
    }
    CATCH (BALL B){
      TARGET.AIM(B);
    }
  }
  PUBLIC STATIC VOID MAIN(STRING[] ARGS) {
    P PARENT = NEW P(NULL);
    P CHILD = NEW P(PARENT);
    PARENT.TARGET = CHILD;
    PARENT.AIM(NEW BALL());
  }
}
```

Source: Randall Munroe (`xkcd.com`)

4 Points