University of Technology, Jamaica
School of Computing and Information Technology

OOP Principles Lab Practical Eight – Exception Handling

Objective:

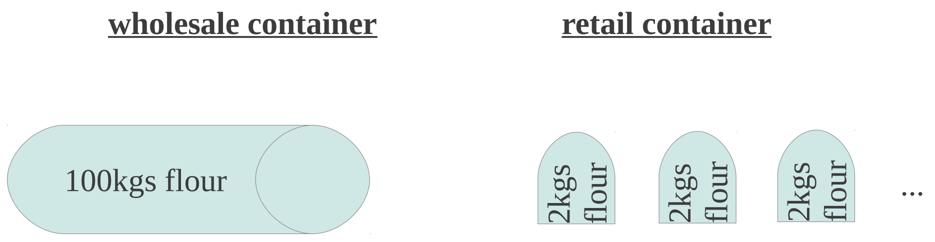
The objectives of this lab practical are to allow students to be able to:

- practice defensive programming using excepting handling
- gain experience using try, catch, throw and finally
- work with exception specifications/declarations and re-throwing exceptions

Exercise One

Enter the following program files into your text editor or IDE, save them then compile them. There are two versions – C++ and Java and both perform the same task. The idea behind both programs is to facilitate a retailer who purchases wholesale goods and repackages them into smaller retail quantities. Each program prompts the user to enter the size of the wholesale container and the size of the retail container, then calculates the total amount of retail containers needed to repackage the item.

Run both programs. Enter some sample data to ensure they run OK. For example, if you enter 100 for the wholesale container size, and 2 for the retail container size, then the program should output the answer that 50 retail containers in total would be needed.



The purpose of the following C++ and Java programs is to determine how many retail containers would be needed to repackage the contents of the wholesale container into retail containers of the given size.

C++

```cpp
//ExceptionExample.cpp
//By David White 2009 Oct 19
#include <iostream>
using namespace std;

class ContainersNeeded
{
	public:
	/* Demonstrate() calls two functions to enter
	 the size of the wholesale and retail containers
	  respectively, then calls a 3rd function to calculate
	   the number of retail containers needed and displays
	    this result */
	void Demonstrate()
	{
		int WholesaleSize;  //size of ws container
		int RetailSize; //size of rt container
		int NumberRetailNeeded;//num of containers needed
		char ans; //single char needed to store 'y' or 'n'

		cout << "Now in Demonstrate()" << endl;

		while ( true )
		{
			WholesaleSize = EnterSizeOfWholesaleContainer();
			RetailSize = EnterSizeOfRetailContainer();
			NumberRetailNeeded =
			 CalculateRetailContainersNeeded(WholesaleSize, RetailSize);
			cout << "Back in Demonstrate()" << endl;
			cout << "The number of retail containers"
				<< " needed is " <<
				 NumberRetailNeeded << endl;
			cout << "Perform another calculation?"
				<< " [y/n] ";
			ans = ' ';
			while (ans != 'y' && ans != 'Y' && ans != 'n' && ans != 'N')
				cin >> ans;
			cout << endl;
			if (ans == 'n' || ans == 'N')
				break;
		}
	}

	//enters the size of the wholesale container
	int EnterSizeOfWholesaleContainer()
	{
		int size;
		cout << "Now in EnterSizeOfWholesaleContainer()" << endl;
		cout << "Enter the size of the wholesale container - ";
		cin >> size;
		cout << endl;
		return size;
	}
	//enters the size of the retail container
	int EnterSizeOfRetailContainer()
	{
		int size;
		cout << "Now in EnterSizeOfRetailContainer()"
			<< endl;
		cout << "Enter the size of the retail container - ";
		cin >> size;
		cout << endl;
		return size;
	}

	//calculates the number of retail containers needed
	// and returns the result
	int CalculateRetailContainersNeeded(int wcSize, int rcSize)
	{
			cout << "Now in CalculateRetailContainersNeeded()" << endl;
			return wcSize / rcSize;
	}


};


//main method - program starts running here
int main()
{
	ContainersNeeded *X = new ContainersNeeded;

	cout << "Calling Demonstrate() from main()" << endl;

	X->Demonstrate();

	cout << "Returned from Demonstrate(), now back in main()" << endl;

	return 0;
}
```

Java

```
//ExceptionExample.cpp
//By David White 2009 Oct 19
import java.io.IOException;
import java.util.Scanner;

public class ContainersNeeded
{
      /*Demonstrate() calls two functions to enter
       the size of the wholesale and retail containers
        respectively, then calls a 3rd function to calculate
         the number of retail containers needed and displays this result*/
      public void Demonstrate() throws IOException
      {
            int WholesaleSize;  //size of ws container
            int RetailSize; //size of rt container
            int NumberRetailNeeded;//num of containers needed
            char ans; //single char needed to store 'y' or 'n'

            System.out.println("Now in Demonstrate()");

            while ( true )
            {
                  WholesaleSize = EnterSizeOfWholesaleContainer();
                  RetailSize = EnterSizeOfRetailContainer();
                  NumberRetailNeeded =
                   CalculateRetailContainersNeeded(WholesaleSize, RetailSize);
                  System.out.println("Back in Demonstrate()");
                  System.out.println("The number of retail containers"+
                         " needed is " +
                         NumberRetailNeeded);
                  System.out.print("Perform another calculation?"
                        + " [y/n] ");
                  ans = ' ';
                  while (ans != 'y' && ans != 'Y' && ans != 'n' && ans != 'N')
                        ans = (char) System.in.read();
                  System.out.println(" ");
                  if (ans == 'n' || ans == 'N')
                        break;
            }
      }

      //enters the size of the wholesale container
      public int EnterSizeOfWholesaleContainer()
      {
            Scanner in = new Scanner(System.in);
            int size;
            System.out.println("Now in EnterSizeOfWholesaleContainer()");
            System.out.println("Enter the size of the wholesale"
                  + " container - ");
            size = in.nextInt();
            System.out.println(" ");
            return size;
      }
```

```
	//enters the size of the retail container
	public int EnterSizeOfRetailContainer()
	{
		Scanner in = new Scanner(System.in);
		int size;
		System.out.println("Now in EnterSizeOfRetailContainer()");
		System.out.println("Enter the size of the retail container - ");
		size = in.nextInt();
		System.out.println(" ");
		return size;
	}

	//calculates the number of retail containers needed
	// and returns the result
	public int CalculateRetailContainersNeeded(int wcSize, int rcSize)
	{
			System.out.println("Now in CalculateRetailContainersNeeded()");
			return wcSize / rcSize;
	}

	//main method – program starts running here
	public static void main(String[] args) throws IOException
	{
		ContainersNeeded X = new ContainersNeeded();

		System.out.println("Calling Demonstrate() from main()");

		X.Demonstrate();

		System.out.println("Returned from Demonstrate()," +
		"  now back in main()");
	}

}
```

Exercise Two

The code in the programs above work most of the time. The problem with the code is that in the CalculateRetailContainersNeeded() method, if the retail container size (rcSize) is zero, then both programs will crash, because integer division by zero is not allowed in either of these two OOP languages. The method is reprinted below in both languages. Run the programs again and enter 0 as the retail container size and see the result for yourself.

C++

```cpp
      //calculates the number of retail containers needed
      // and returns the result
      int CalculateRetailContainersNeeded(int wcSize, int rcSize)
      {
                  cout << "Now in CalculateRetailContainersNeeded()" << endl;
                  return wcSize / rcSize;
      }
```

Java

```java
      //calculates the number of retail containers needed
      // and returns the result
      public int CalculateRetailContainersNeeded(int wcSize, int rcSize)
      {
                  System.out.println("Now in CalculateRetailContainersNeeded()");
                  return wcSize / rcSize;
      }
```

Rewrite the CalculateRetailContainersNeeded() method and make use of exception handling so that the programs do not crash or abort abnormally. Do not change any other method (handle the exception inside the method). Run the programs again and observe the results this time.

Exercise Three

Rewrite the CalculateRetailContainersNeeded() method again but this time do not handle the exception in the method. Ensure that you use an exception specification to declare the type of exception that may be thrown by the method. Do not change any other method. Will the programs run this time? If so, what will occur when 0 is entered? If it does not run, why not?

Exercise Four

Rewrite any method that calls the CalculateRetailContainersNeeded() method directly, using exception handling to handle any exceptions thrown by the CalculateRetailContainersNeeded() method. Rethrow any exceptions handled. Will the program still run now?

Exercise Five

Rewrite the main() method so that all exceptions are handled, including those exceptions that occur in methods called by main(). Ensure that the programs compile successfully and run correctly.

Exercise Six

Rewrite the CalculateRetailContainersNeeded() method once more, this time checking if the retail container size (rcSize) is zero. If it is zero, explicitly throw a customized designed exception called MyException. In the C++ version, let MyException inherit from runtime_error, while in the Java version let MyException inherit from Exception. Again, ensure that the programs compile successfully and run correctly.

Homework

a) Write a method called GetDayOfMonth() which prompts and accepts the day of the month (an integer) from the user then returns this number. Use exception handling to indicate that an exception occurred if the day of the month is less than one or greater than thirty-one, but do not handle the exception in this method.

b) Write a method called ProcessDay() that calls the GetDayOfMonth() method and displays the day returned by the method. Handle any exceptions that may occur in the ProcessDay() or passed up to it from methods called. Rethrow any exception that occurs.

c) Write a main() method which calls the ProcessDay method. Handle all exceptions that may occur.

d) Ensure that you program compiles and runs correctly, both when there is good input (0 < day < 32) or invalid input (day < 1 or day > 31).