**CS52 Lab 9: Exception Handling**

**Sources of Support**

1. Pages 895 to 926 of Savitch 10th edition discuss Exceptions and Error handling.
2. Exception Handling PowerPoint available in canvas in this module.
3. An example is contained in this document

Exception handling is used in many different ways, but its use in three situations is well suited.

1. A member function of a class or a stand-alone function is required to return a value, but due to error condition it cannot do that. In that case a return by exception mechanism informs user of the error condition, and allows them to take corrective action by placing a catch block in the code.
2. A file or data source is opened either for reading or writing. The data source crashes or is not available due to hardware problems. In that case exception mechanism informs user of the problem and closes files safely, and releases resources that were in use.
3. Validation of data input.

There are other situations of course, for example out of bound array access, illegal casting operators etc., which can be handled by exception handling.

**Exception Handling Use for Employee Class**

UML diagram of Employee class is given below.



Notice that it has a field hoursWorked, which is number of hours worked by the employee in a week. A week contains (24\*7 = 168), 168 hours. Obviously if user input for hours worked is greater than 168, then that is an error condition in the data input. We can also assume some other value of hoursWorked that is lesser than 168 as being in an error. In this situation we will just work with 168 hours.

The class also has a getInstance method which has the below header.

void getInstance (Employee & EMP);

This method takes user input for all employee data, including hoursWorked, from the keyboard. If user enters hours worked >168, then that is an exception/error condition that we need to correct. For that we need to write an exception class with name (related to the exception). We choose the class name to be: **HoursWorkedException**

**It is important to name the exception handling classes, in a way that the name is closely related to the exception being handled.**

Exception classes are similar to regular classes. They have data members, constructor(s) and generally only get methods. More often than not, the exception classes need not mutate, so they may not have set functions. We wish to design the HoursWorkedException class such that:

* It carries the invalid value of hoursWorked entered by the user.
* A message that can tell user the details of error.

Thus we add the following three data members to the **HoursWorkedException** class.

**const static int NUM\_HOURS\_WEEK = 168;**

**int hoursWorked; // Number of hours worked inputted by the user**

**string Message; // Message to be displayed to the user when exception is handled**

**NUM\_HOURS\_WEEK** stores the number of hours in a week. For class **HoursWorkedException**, we provide a dual role default and explicit constructor and getters for all three data members. The UML diagram for **HoursWorkedException class is shown below.**



The source code is shown in Table below.

|  |  |
| --- | --- |
| Line # | Source code for class HoursWorkedException |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | classHoursWorkedException**{**  private**:**  conststaticintNUM\_HOURS\_WEEK **= 168;**  doublehoursWorked**;**  stringMessage**;**  public**:**  HoursWorkedException**(**doublehrs**=0.0,** stringmsg **= "Message not set"):**  hoursWorked**(**hrs**),**Message**(**msg**){**  //no code needed  **}**  doublegetHoursWorked**()** const**{**  returnhoursWorked**;**  **}**  conststringgetMessage**()** const**{**  returnMessage**;**  **}**  staticintgetMaxHoursWeek**(){**  returnNUM\_HOURS\_WEEK**;**  **}**  **};** |
|  |  |

As mentioned earlier, the class has three data members, and three getters; one to return each of them. The default/explicit constructor initialize the hoursWorked and Message data members to default or user provided values.

**Use of HoursWorkedException class inside the getInstance method of Employee class**

If we wish to make the use of **HoursWorkedException** class, then an object of this class type need preparation and throwing at the appropriate location inside the getInstance method. To get the best use of this exception class, the code inside the getInstance has been rearranged. The code for modified getInstance is given in the table below.

|  |  |
| --- | --- |
| Line # | Source code for modified getInstance in Employee class |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | voidEmployee**::**getInstance**(**Employee **&** EMP**){**  //why do we need below if structure code?  if**(!**cin**){**  return**;**  **}**  //Get input for data that can throw exception first so that user does not have to  //enter other data until data subject to exception is finished first  cout**<<"Please enter employee hours worked [xx.yy] :";**  doublehrs**;**  cin**>>**hrs**;**  hrs **=** abs**(**hrs**);**  if **(**hrs**>**HoursWorkedException**::**getMaxHoursWeek**()) {**  stringmsg**= "Number of hours worked " +** to\_string**(**hrs**)+**  **" exceeds total hours in a week: " +**to\_string**(**HoursWorkedException**::**getMaxHoursWeek**())+**  **".\n";**  throwHoursWorkedException**(**hrs**,**msg**);**  **}**  cout**<<"Please enter employee social security number [digits only]: ";**  stringssn**;**  cin**>>**ssn**;**  cout**<<"Please enter employee first name: ";**  stringfirst**;**  cin**>>**first**;**  cout**<<"Please enter employee last name: ";**  stringlast**;**  cin**>>**last**;**  cout**<<"Please enter employee hourly pay rate [xx.yy] :";**  doublerate**;**  cin**>>**rate**;**  rate **=** abs**(**rate**);**  EMP **=** Employee**(**ssn**,**first**,**last**,**rate**,**hrs**);**  **}** |
|  |  |

In above table, the user inputted hours worked in a week completes on line 11. Line 12 then codes a conditional if to test whether hrs(hours worked), exceeds the value HoursWorkedException**::**getMaxHoursWeek**(),** latter being 168. [See code for function getMaxHoursWeek in the class HoursWorkedException in previous table]. Exception condition exists if the condition

hrs**>**HoursWorkedException**::**getMaxHoursWeek**())**

**evaluates to true.** In that case the body of if structure prepares a string msg which contains the description of exception condition [Line 14 to 16]. Then the object of type HoursWorkedException is prepared and thrown. The throw clause will terminate the execution of the getInstance and return the object constructed on line 17 to the calling block.

**Handling the Exception in main function**

The table below shows the main function where exception handler HoursWorkedExceptionis used to validate the user input for hours worked. The code that could throw an exception is placed inside a try block and catch block then gets the exception object, where it is used further.

|  |  |
| --- | --- |
| Line # | Code in the main function which uses the Exception Handler [All includes have been done] |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | intmain**(** **){**  EmployeeEMP**;**  booldone **=** false**;**  while **(!**done**) {**  try **{**  Employee**::**getInstance**(**EMP**);**  done **=** true**;**  **}** catch **(**HoursWorkedExceptionex**) {**  cout**<<**ex**.**getMessage**()<<**endl**;**  done **=** false**;**  **}**  **}**  cout**<<"Data of employee you just entered:\n";**  cout**<<**EMP**.**toString**();**  return **0;**  **}** |
|  | Typical output from the main function |
| 18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | Please enter employee hours worked [xx.yy] :200  Number of hours worked 200.000000 exceeds total hours in a week: 168.  Please enter employee hours worked [xx.yy] :300  Number of hours worked 300.000000 exceeds total hours in a week: 168.  Please enter employee hours worked [xx.yy] :55.0  Please enter employee social security number [digits only]: 11223344  Please enter employee first name: Sam  Please enter employee last name: Jones  Please enter employee hourly pay rate [xx.yy] :10.02  Data of employee you just entered:  Employee Social Security Number: 11223344  Employee Name: Sam Jones  Employee Hours Worked: 55.00 hours.  Employee pay rate: $10.02 per hour.  Employee salary: $626.25 |

In this code a flag controlled loop in addition to the try catch block are used for data validation for the hours worked. A flag controlled loop is started on line 5. Since getInstance can throw exception, the call to it is surrounded by a try block. If hours worked entered by the user exceeds 168, then line 17 of getInstance throws the exception object and rest of the try block is skipped. But since there is a catch block whose argument has the exception object that is thrown, we can make use of it to handle the invalid data input. When employee hour worked = 200 are entered (see output line 18 in above table), then exception object is thrown by the getInstance function. The line 8 inside the try block is skipped. the control then transfers to the line 10 in catch block. Inside this block we call the getMessage function of exception object ex. The output from this function is shown on line 19 of output part in above table. It clearly informs user that entering 200 hours is an error condition. Then line 11 sets done to false, and control returns to the top of the loop. The call to getInstance will keep repeating until the hours worked less than 168 are entered. That is the case when user inputs the hours worked = 55 as shown by the line 24 of output. In that case exception object is not thrown because in the getInstance function the condition

hrs**>**HoursWorkedException**::**getMaxHoursWeek**()**

evaluates to false . [See code of getInstance function]. The program is then able to complete the data input for rest of the employee and inputted employee is printed to console. [See code line 15, and output lines 30 to 34].

**Statement of Work You Need to do**

For BankAccount class one special situation that also deserves exception handling is in the getInstance method which has the following version.

static void getInstance(BankAccount & BA){

}

Understand that in this version at the end of the function, the constructor construct an object of BankAccount type which must be a legal object with balance > 0.0. It is reading data from the keyboard. What if user enters an account balance that is zero or negative? Obviously, the bank cannot afford to start an account with zero or negative balance. Thus you need to write an exception class to handle that situation. The class UML diagram of this class is given below.



The class has two data members. The double initBalance will store the initial balance entered by the user to construct a BankAccount object when getInstance function is called. Message stores the message to be displayed to the user when catch block executes. The class has two getters, one for each data member. The class also has the constructor, which folds both default and explicit constructors into one. That means that default values are provided for both constructor arguments.

**Proceed as Follows**

1. Create a C++ console application for this assignment.
2. For sake of simplicity you could create a single file.
3. Copy and paste all the includes, from your BankAccount.h on top of your source file. Delete all the include guards.
4. Code the full definition of class ZeroOrNegativeInitialBalanceException. See the model of Exception class I did for the Employee class in the table in previous pages.
5. Copy and paste the BankAccount.h file. Delete any end of include guard if you had in BankAccount.h.
6. Copy and paste your BankAccount.cpp. Delete the include statement on the top.
7. Modify the getInstance (BankAccount & BA) in your implementation of this function that function throws the object of type ZeroOrNegativeInitialBalanceException when user enters the 0 or a negative value for the initial balance. You can look at the mode of getInstance I did for Employee in the table in previous pages. Rest of the function getInstance remains unchanged.
8. Write the main function now so that inside the flag controlled loop and inside the try block the BankAccount getInstance that you modified is called. You can look for the model of main function in the table in previous pages where I applied it to Employee class.
9. The typical output from my program is given below. Your output should be qualitatively similar.

*Enter the balance or initial deposit: -2.0*

*Initial balance of negative dollars is not allowed.*

*Enter the balance or initial deposit: 0.0*

*Initial balance of zero dollars is insufficient*

*Enter the balance or initial deposit: 200.0*

*Enter the new (unique) account number: 1234*

*Enter the first name on account: Sam*

*Enter the last name on account: Jones*

*Bank Account object created:*

*Account Number: 1234 Name: Sam Jones Balance: $200.00*

**Submit this source file or your entire project (zipped) to dropbox.**