**Object-Oriented Application Development**

**Practical 12**

**Part A**

1. Run the following programs twice: first with valid input, then with invalid input.

|  |  |
| --- | --- |
| (a) | using System;  public class ExceptionExample1  {  public static void Main()  {  Console.Write("Enter an integer: ");  string input = Console.ReadLine();  int number = Convert.ToInt32(input);  Console.WriteLine("Number is: {0}\n", number);  }  } |
| (b) | using System;  public class ExceptionExample2  {  public static void Main()  {  Console.Write("Enter numerator: ");  int n1 = Convert.ToInt32(Console.ReadLine());  Console.Write("Enter denominator: ");  int n2 = Convert.ToInt32(Console.ReadLine());  int result = n1 / n2;  Console.WriteLine("Result: {0}\n", result);  }  } |

2. Rewrite the programs in Question 1 with try-catch blocks for exception handling. Run again with valid and invalid input.

public static void Main()

{

try {

Console.Write("Enter an integer: ");

string input = Console.ReadLine();

int number = Convert.ToInt32(input);

Console.WriteLine("Number is: {0}\n", number);

}

catch

{

Console.WriteLine("Meh");

}

}

3. Given the class definition for class Account:

|  |
| --- |
| using System; // for ArgumentOutOfRangeException  public class Account  {  private decimal balance = 0.0M;  public decimal Balance  {  get { return balance; }  set  {  if (value >= 0.0M)  balance = value;  else // throwing an exception  throw new ArgumentOutOfRangeException();  }  }  public Account(decimal initialBalance)  { Balance = initialBalance; }  public void Withdraw(decimal amount)  {  if (amount <= Balance)  Balance = Balance - amount;  else // throwing an exception  throw new ArgumentOutOfRangeException("amount",  "Amount cannot be greater than balance");  }  public void Deposit(decimal amount)  { Balance = Balance + amount; }  } |

Run the programs with the following Main() methods:

|  |  |
| --- | --- |
| (a) | using System;  public class AccountTest1  {  public static void Main()  {  Account account = new Account(-5.0M);  }  } |
| (b) | using System;  public class AccountTest2  {  public static void Main()  {  Account account = new Account(500.0M);  account.Withdraw(1000.0M);  }  } |

4. Rewrite the Main methods in Question 3 with try-catch blocks for exception handling. Run again.

**Part B**

1. Given the class definition for class NegativeNumberException, which extends the library (built-in) class ApplicationException from the System namespace:

|  |
| --- |
| public class NegativeNumberException **:** System.ApplicationException  {  public NegativeNumberException(string exceptionMessage)  **:** base (exceptionMessage)  {  // Empty body  }  } |

Run the program below with valid and invalid input.

|  |
| --- |
| using System;  public class Account  {  private decimal balance = 0.0M;  public decimal Balance  {  get { return balance; }  set  {  if (value >= 0.0M)  balance = value;  else // throwing an exception  throw new NegativeNumberException("Negative balance not allowed");  }  }  public Account(decimal initialBalance)  {  Balance = initialBalance;  }  public void Withdraw(decimal amount)  {  if (amount < 0.0M)  throw new NegativeNumberException("Negative amount not allowed");  else  {  if (amount <= Balance)  Balance = Balance - amount;  else  throw new ArgumentOutOfRangeException("amount",  "Amount cannot be greater than balance");  }  }  } |

|  |
| --- |
| using System;  public class AccountTest  {  public static void Main()  {  try  {  Console.Write("Enter balance: ");  decimal balance = Convert.ToDecimal(Console.ReadLine());  Account account = new Account(balance);  Console.Write("Enter amount to withdraw: ");  decimal amount = Convert.ToDecimal(Console.ReadLine());  account.Withdraw(amount);  }  catch (NegativeNumberException ex)  {  Console.WriteLine(ex.Message);  }  catch (ArgumentOutOfRangeException ex)  {  Console.WriteLine(ex.Message);  }  catch // general catch block  {  Console.WriteLine("Some other error");  }  }  } |

2. (a) Write a IllegalTransactionException class that extends the library (built-in) class ApplicationException from the System namespace.

(b) Modify the Account class in Question 1 so that the Withdraw method will throw an IllegalTransactionException if there is an attempt to make the balance below zero.

(c) Modify the Account class to add a Deposit method. The method should throw a NegativeNumberException if the amount to deposit is negative.

(d) Test your modified Account class with a Main() method that contains catch blocks to catch NegativeNumberException and IllegalTransactionException.

public class NegativeNumberException : System.ApplicationException

{

public NegativeNumberException(string exceptionMessage)

: base(exceptionMessage)

{

// Empty body

}

}

public class ITE : System.ApplicationException

{

public ITE(string exceptionMessage) : base(exceptionMessage)

{ }

}

public class Account

{

private decimal balance = 0.0M;

public decimal Balance

{

get { return balance; }

set

{

if (value >= 0.0M)

balance = value;

else // throwing an exception

throw new NegativeNumberException("Negative balance not allowed");

}

}

public Account(decimal initialBalance)

{

Balance = initialBalance;

}

public void Deposit(decimal amount)

{

if (amount < 0.0M)

throw new NegativeNumberException("Negative amount not allowed");

else

{

Balance = Balance + amount; }

}

public void Withdraw(decimal amount)

{

if (amount < 0.0M)

throw new NegativeNumberException("Negative amount not allowed");

else

{

if (amount <= Balance)

Balance = Balance - amount;

else

throw new ITE("Amount cannot be NEGATIVE");

}

}

}

public class AccountTest

{

public static void Main()

{

try

{

Console.Write("Enter balance: ");

decimal balance = Convert.ToDecimal(Console.ReadLine());

Account account = new Account(balance);

Console.Write("Enter deposit: ");

decimal deposit = Convert.ToDecimal(Console.ReadLine());

account.Deposit(deposit);

Console.Write("Enter amount to withdraw: ");

decimal amount = Convert.ToDecimal(Console.ReadLine());

account.Withdraw(amount);

}

catch (NegativeNumberException ex)

{

Console.WriteLine(ex.Message);

}

catch (ArgumentOutOfRangeException ex)

{

Console.WriteLine(ex.Message);

}

catch // general catch block

{

Console.WriteLine("Some other error");

}

}

}

3. Rewrite the application in Question 2 as a Windows-based application. Include exception handling for all possible exceptions that may occur in your application. The application should also check whether the text boxes are empty when the user clicks the button.