**LAB 11**

**QUESTION 1:**

**This Q1. You are tasked to develop a reusable array class where accessing an invalid index must throw an exception.**

**Requirements:**

**● Create a class template SafeArray<T>.**

**● Add methods:**

**○ void set(int index, T value)**

**○ T get(int index)**

**● Throw an OutOfBoundsException if:**

**○ Index < 0**

**○ Index >= size**

**● Catch the exception in main() and print "Invalid array index access.".**

**● The class should be able to work with different types (int, double, string).**

**PROGRAM**:

#include<iostream>

#include<stdexcept>

using namespace std;

class OutOfBoundsException : public exception{

    public:

    const char\* what() const noexcept override{

        return "out of bound index";

}

};

template<typename T>

class SafeArray{

    private:

    T array[5];

    int size=5;

    public:

    void set(int *index*, T *value*){

            array[*index*]=*value*;

    }

     T get(int *index*){

               if(*index* < 0 || *index* > size){

                throw OutOfBoundsException();

               }

            return array[*index*];

     }

};

int main(){

    class SafeArray<int> s;

    int value;

    for(int i=0;i<5;i++){

        cout<<"Enter value for "<<i+1<<" :";

        cin>>value;

        s.set(i,value);

    }

    try{

     cout<<s.get(3)<<endl;

    }

    catch(OutOfBoundsException e){

         cout<<e.what()<<endl;

    }

     try{

        cout<<s.get(-1)<<endl;

       }

       catch(OutOfBoundsException e){

            cout<<e.what()<<endl;

       }

}

**RESULT:**

**QUESTION#2**

**Implement a stack data structure for any data type that throws an error when trying to pop from an empty stack.**

**Requirements:**

**● Create a class template Stack<T>.**

**● Implement methods:**

**○ void push(T item)**

**○ T pop()**

**● Throw a custom StackUnderflowException if pop is called on an empty stack.**

**● Handle the exception in main() and display "Stack is empty. Cannot pop.".**

**PROGRAM**:

#include<iostream>

#include <stdexcept>

using namespace std;

class StackUnderflowException : public exception{

    public:

    const char\* what() const noexcept override{

        return "stack is empty";

}

};

template <class T>

class Stack{

    private:

        T array[5];

        int count;

    public:

        Stack(){

            count=0;

        }

        void push(T *item*){

           array[count]=*item*;

           count++;

        }

        T pop(){

            int temp=0;

                if(count==0){

                    throw StackUnderflowException();

                }

           count--;

           return array[count];

        }

};

int main(){

    Stack<int> s;

    s.push(2);

    s.push(3);

try{

    cout<<s.pop()<<endl;

}

catch(StackUnderflowException e){

    cout<<e.what()<<endl;

}

try{

    cout<<s.pop()<<endl;

}

catch(StackUnderflowException e){

    cout<<e.what()<<endl;

}

try{

    cout<<s.pop()<<endl;

}

catch(StackUnderflowException e){

    cout<<e.what()<<endl;

}

}

A black background with white text

AI-generated content may be incorrect.

**QUESTION#3**

**Develop a generic matrix multiplication function. Before multiplying, ensure the**

**matrices are compatible.**

**Requirements:**

**● Create a class template Matrix<T> that holds a 2D array.**

**● Write a function multiply(Matrix<T> a, Matrix<T> b):**

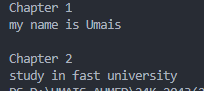
**○ Throw DimensionMismatchException if number of columns in a != number**

**of rows in b.**

**● Catch the exception and print "Matrix dimensions incompatible for multiplication.".**

**● Use the template to multiply matrices of int and float.**

#include<iostream>



**QUESTION#4**

**Q4. Build a file reader that can read and store data of different types from a file.**

**Requirements:**

**● Create a function template readFromFile<T>(string filename).**

**● If the file does not exist or cannot be opened, throw a FileNotFoundException.**

**● Handle the exception by printing "Error: Unable to open file.".**

**● Read data (e.g., integers, floats, or strings) into an array<T>.**

**● Demonstrate with a sample file.**

#include <iostream>

#include <fstream>

#include <stdexcept>

using namespace std;

class FileNotFoundException: public exception{

    public:

    const char\* what() const noexcept override{

        return "unable to open to";

    }

    };

class FileReader{

    public:

template <typename T>

void readfromFile(string *filename*){

    string line;

    ifstream file(filename);

     if(!file){

             throw FileNotFoundException();

     }

     while(getline(file,line)){

        cout<<line<<endl;

     }

     int length;

     length=line.length();

    T a[line.length()+1]=line;

    return a;

}

};

int main(){

   FileReader F;

  try{

   F.readfromFile("file.txt");

  }

  catch(FileNotFoundException e){

    cout<<e.what()<<endl;

  }

}

**QUESTION#5**

**You are designing a banking application that can handle different types of account**

**balances (float, double).**

**Requirements:**

**● Create a class template BankAccount<T>.**

**● Implement methods:**

**○ void deposit(T amount)**

**○ void withdraw(T amount)**

**● Throw an InsufficientFundsException if withdrawal amount > current balance.**

**● Catch and handle the exception with a message like "Withdrawal failed: Insufficient**

**funds.".**

**● Allow deposit and withdrawal operations through user input.**

#include<iostream>

using namespace std;

class InsufficientFundsException: public exception{

    public:

    const char\* what() const noexcept override{

        return "Insufficient Funds";

    }

    };

template <typename T>

class BankAccount{

    private:

   T amount;

   public:

   BankAccount(){

    amount=0;

   }

   void deposit(T *Amount*){

      amount+=*Amount*;

      cout<<"Amount added sucessfully"<<endl;

   }

   void withdrawal(T *Amount*){

    if(*Amount* > amount){

     throw InsufficientFundsException();

    }

    amount-=*Amount*;

    cout<<"Amount withdrawal sucessfully"<<endl;

   }

};

int main(){

    BankAccount<int> B;

    B.deposit(20000);

    try{

        B.withdrawal(1500);

    }

    catch(InsufficientFundsException e){

        cout<<e.what();

    }

    try{

        B.withdrawal(30000000);

    }

    catch(InsufficientFundsException e){

        cout<<e.what()<<endl;

    }

}

