Homework 5

UML Sequence Diagrams

You may draw these diagrams using a UML editor or by hand (provided they’re easily readable).

Draw a sequence diagram describing the execution of main(). Although main() is not really an object, you may show it as an object in the diagram (no class name needed). (Use multiple pages for your diagram—you’ll need them.)

|  |
| --- |
| **Main.cpp** |
| #include <iostream>  #include "EagerAverager.h"  #include "LazyAverager.h"  #include "ScoreAveragerUnitTest.h"  using namespace std;  int main(int argc, char\* argv[]) {  IScoreAverager\* eager = new EagerAverager();  ScoreAveragerUnitTest testEager(eager);  ScoreAveragerUnitTest testLazy(new LazyAverager());  cout << "Eager Averager" << endl;  cout << "\tTest 1 ";  if (testEager.runTest1()) cout << "passed" << endl;  else cout << "failed" << endl;  cout << "\tTest 2 ";  if (testEager.runTest2()) cout << "passed" << endl;  else cout << "failed" << endl;  cout << endl << "Lazy Averager" << endl;  cout << "\tTest 1 ";  if (testLazy.runTest1()) cout << "passed" << endl;  else cout << "failed" << endl;  cout << "\tTest 2 ";  if (testLazy.runTest2()) cout << "passed" << endl;  else cout << "failed" << endl;  system("pause");  return (0);  } |

|  |
| --- |
| **ScoreAveragerUnitTest.h** |
| #ifndef SCORE\_AVERAGER\_UNIT\_TEST\_H  #define SCORE\_AVERAGER\_UNIT\_TEST\_H  class IScoreAverager;  class ScoreAveragerUnitTest {  public:  ScoreAveragerUnitTest(IScoreAverager\*);  ~ScoreAveragerUnitTest(void);  bool runTest1(); // pre: no scores have been added  bool runTest2(); // pre: no scores have been added  private:  bool checkValues(double avg, int num);  IScoreAverager\* \_scoreAverager;  };  #endif |

|  |
| --- |
| **ScoreAveragerUnitTest.cpp** |
| #include "ScoreAveragerUnitTest.h"  #include "IScoreAverager.h"  ScoreAveragerUnitTest::ScoreAveragerUnitTest(IScoreAverager\* a) {  \_scoreAverager = a;  }  ScoreAveragerUnitTest::~ScoreAveragerUnitTest(void) { }  bool ScoreAveragerUnitTest::checkValues(double avg, int num) {  double epsilon = 0.0000001;  double computedAvg = \_scoreAverager->getAverage();  if ((computedAvg < (avg-epsilon)) ||  (computedAvg > (avg+epsilon)))  return false;  if (\_scoreAverager->getNumScores() != num)  return false;  return true;  }  bool ScoreAveragerUnitTest::runTest1() {  return (checkValues(0, 0));  }  bool ScoreAveragerUnitTest::runTest2() {  \_scoreAverager->addScore(78);  \_scoreAverager->addScore(90);  if (!checkValues((78+90)/2.0, 2))  return false;  \_scoreAverager->addScore(89);  return(checkValues((78+90+89)/3.0, 3));  } |

|  |
| --- |
| **IScoreAverager.h** |
| #ifndef ISCORE\_AVERAGER\_H  #define ISCORE\_AVERAGER\_H  // This is the specification from Hwk 3  class IScoreAverager  {  public:  // constructor  // postcondition: getNumScores()' == 0  virtual void addScore(double score) = 0;  // precondition: score >= 0  // postcondition:  // getNumScores()' == getNumScores() + 1  // getAverage()' == (getAverage() \* getNumScores() + score) / (getNumScores()')  virtual double getAverage() = 0;  // precondition: getNumScores() >= 0  // postcondition:  // getAverage()' == getAverage()  // getNumScores()' == getNumScores()  virtual int getNumScores() = 0;  // postcondition:  // getAverage()' == getAverage()  // getNumScores()' == getNumScores()  virtual ~IScoreAverager(void) { };  };  #endif |

|  |  |
| --- | --- |
| **PartialAverager.h** | **PartialAverager.cpp** |
| #ifndef PARTIAL\_AVERAGER\_H  #define PARTIAL\_AVERAGER\_H  #include "IScoreAverager.h"  class PartialAverager :  public IScoreAverager {  public:  PartialAverager(void);  ~PartialAverager(void);  virtual int getNumScores();  protected:  virtual void incrementNumScores();  virtual void resetNumScores();  private:  int \_numScores;  };  #endif | #include "PartialAverager.h"  PartialAverager::PartialAverager(void) {  resetNumScores();  }  PartialAverager::~PartialAverager(void) { }  int PartialAverager::getNumScores() {  return \_numScores;  }  void PartialAverager::incrementNumScores() {  \_numScores = \_numScores + 1;  }  void PartialAverager::resetNumScores() {  \_numScores = 0;  } |

|  |  |
| --- | --- |
| **EagerAverager.h** | **EagerAverager.cpp** |
| #ifndef EAGER\_AVERAGER\_H  #define EAGER\_AVERAGER\_H  #include "PartialAverager.h"  // This is impl. 5. from Hwk 3  class EagerAverager :  public PartialAverager {  public:  EagerAverager(void);  ~EagerAverager(void);  virtual void addScore(double);  virtual double getAverage();  private:  double \_sumScores;  };  #endif | #include "EagerAverager.h"  EagerAverager::EagerAverager(void) {  \_sumScores = 0;  }  EagerAverager::~EagerAverager(void) { }  void EagerAverager::addScore(double score) {  incrementNumScores();  \_sumScores += score;  }  double EagerAverager::getAverage() {  if (getNumScores() > 0)  return (\_sumScores / getNumScores());  else  return 0;    } |

|  |  |
| --- | --- |
| **LazyAverager.h** | **LazyAverager.cpp** |
| #ifndef LAZY\_AVERAGER  #define LAZY\_AVERAGER  #include "PartialAverager.h"  // This is impl. 2. from Hwk 3  class LazyAverager :  public PartialAverager {  public:  LazyAverager(void);  ~LazyAverager(void);  virtual void addScore(score);  virtual double getAverage();  private:  double \_avgScore;  };  #endif | #include "LazyAverager.h"  LazyAverager::LazyAverager(void) {  \_avgScore = 100;  }  LazyAverager::~LazyAverager(void) { }  void LazyAverager::addScore(double score) {  if (score >= 0)  {  double previousScoreTotal  = \_avgScore \* getNumScores();  incrementNumScores();  \_avgScore = (previousScoreTotal + score) /  getNumScores();  }  }  double LazyAverager::getAverage() {  if (getNumScores() > 0)  return \_avgScore;  else  return 0;  } |