UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS110 - Program design: Introduction

Practical 5 Specifications: Polymorphism

Release date: 26-09-2022 at 06:00
Due date: 30-09-2022 at 23:59
Total Marks: 41

Contents

1 General instructions:

- This assignment should be completed individually, no group effort is allowed.
- Be ready to upload your assignment well before the deadline as no extension will be granted.
- You may not import any of C++'s built-in data structures. Doing so will result in a mark of zero. You may only make use of 1-dimensional native arrays where applicable. If you require additional data structures, you will have to implement them yourself.
- If your code does not compile you will be awarded a mark of zero. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- If your code experience a runtime error you will be awarded a mark of zero. Runtime errors are considered as unsafe programming.
- All submissions will be checked for plagiarism.
- Read the entire specification before you start coding.
- Ensure your code compiles with C++98

2 Plagiarism

The Department of Computer Science considers plagiarism as a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent) and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm (from the main page of the University of Pretoria site, follow the Library quick link, and then choose the Plagiarism option under the Services menu). **If you have any form of question regarding this, please ask one of the lecturers, to avoid any misunderstanding.** Also note that the OOP principle of code re-use does not mean that you should copy and adapt code to suit your solution.

3 Outcomes

The aim of this practical is to gain experience with polymorphism in a somewhat complex class hierarchy.

4 Introduction

Implement the UML diagram and functions as described on the following pages.

Figure 1: Meme

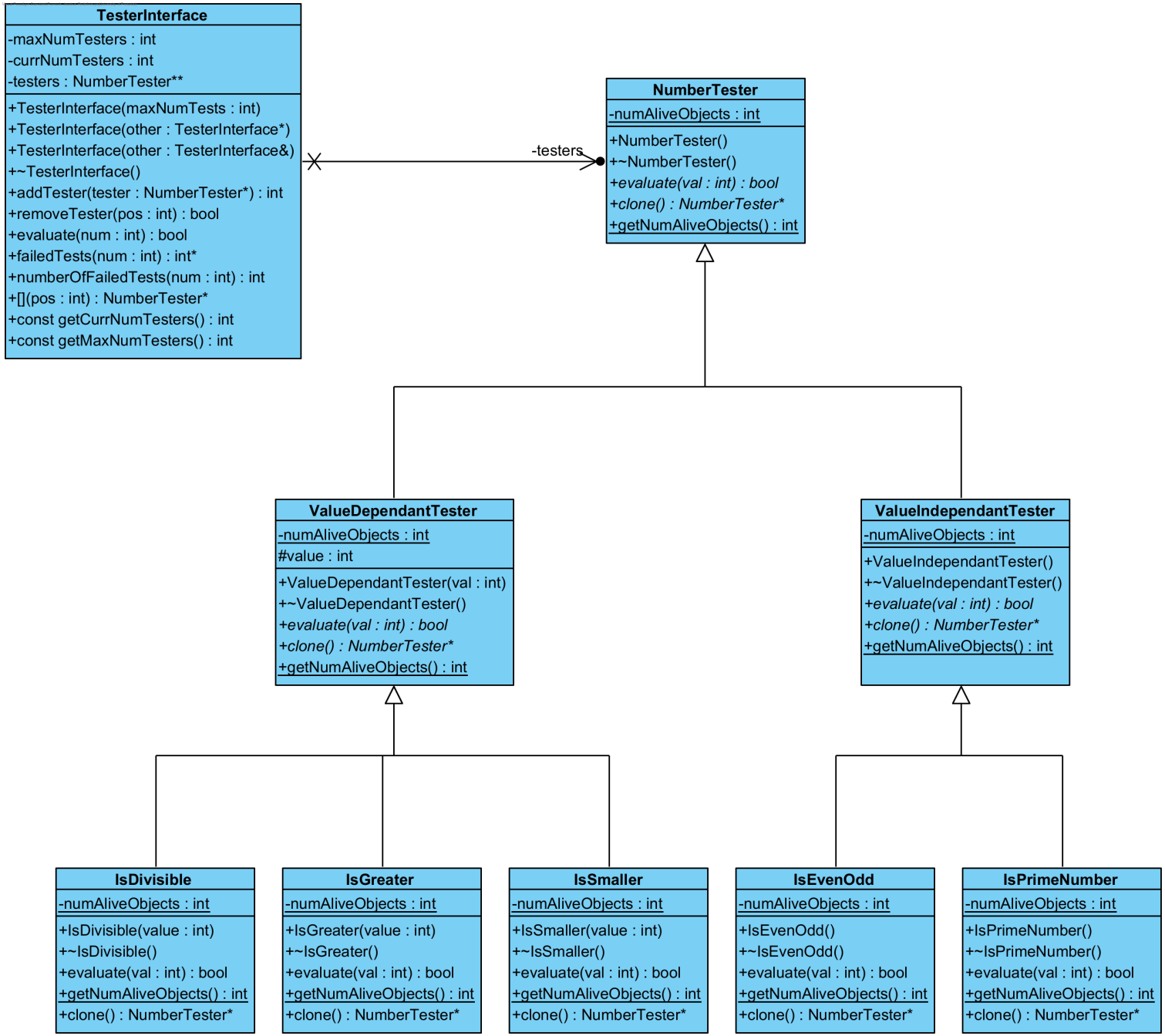

Figure 2: Class diagrams

5 Classes:

5.1 NumberTester:

- members:
  - numAliveObjects: int
    - This is a static variable of the NumberTester class.
    - This variable will keep track of the number of current instantiated NumberTester objects.
    - This variable should be initially initialized to a value of 0.
- functions:
  - NumberTester():
    - This is the constructor for the NumberTester class.
    - This function should increment the numAliveObjects variable.
  - ~NumberTester()
    - This is the destructor for the NumberTester class.
    - This function should decrement the numAliveObjects variable.
  - evaluate(val: int): bool
    - This is a pure virtual function.
  - clone(): NumberTester*
    - This is a pure virtual function.
  - getNumAliveObjects(): int
    - This is a static function.
    - This function should return the static numAliveObjects member of NumberTester class.

5.2 ValueDependantTester

This class has a **public inheritance** relationship with NumberTester.

- member:
  - numAliveObjects: int
    - This is a static variable of the ValueDependantTester class.
    - This variable will keep track of the number of current instantiated ValueDependantTester objects.
    - This variable should be initially initialized to a value of 0.
  - value: int
    - This is the value that will be used by objects of this class.
- functions:
  - ValueDependantTester(val: int)
    - This is the constructor for the ValueDependantTester class.
    - This function should initialize the value member with the passed in parameter.
    - This function should also increment the numAliveObjects variable.

  - ~ValueDependantTester()
    - This is the destructor for the ValueDependantTester class.
    - This function should decrement the numAliveObjects variable.
  - evaluate(val: int): bool
    - This is a pure virtual function.
  - clone(): NumberTester*
    - This is a pure virtual function.
  - getNumAliveObjects(): int
    - This is a static function.
    - This function should return the static numAliveObjects member of ValueDependantTester class.

5.3 ValueIndependantTester:

This class has a **public inheritance** relationship with NumberTester.

- members:
  - numAliveObjects: int
    - This is a static variable of the ValueIndependantTester class.
    - This variable will keep track of the number of current instantiated ValueIndependantTester objects.
    - This variable should be initially initialized to a value of 0.
- functions:
  - ValueIndependantTester():
    - This is the constructor for the ValueIndependantTester class.
    - This function should increment the numAliveObjects variable.
  - ~ValueIndependantTester()
    - This is the destructor for the ValueIndependantTester class.
    - This function should decrement the numAliveObjects variable.
  - evaluate(val: int): bool
    - This is a pure virtual function.
  - clone(): NumberTester*
    - This is a pure virtual function.
  - getNumAliveObjects(): int
    - This is a static function.
    - This function should return the static numAliveObjects member of ValueIndependantTester class.

5.4 IsDivisible

This class has a **public inheritance** relationship with ValueDependantTester.

- members:
  - numAliveObjects: int
    - This is a static variable of the IsDivisible class.
    - This variable will keep track of the number of current instantiated IsDivisible objects.
    - This variable should be initially initialized to a value of 0.
- functions:
  - IsDivisible(int value):
    - This is the constructor for the IsDivisible class.
    - This function should increment the numAliveObjects variable.
    - This function should also initialize the inherited value member with the passed in member
  - ~IsDivisible()
    - This is the destructor for the IsDivisible class.
    - This function should decrement the numAliveObjects variable.
  - evaluate(val: int): bool
    - This function should determine if the passed in value is divisible with the inherited value member.
    - If it is, then the function should return true else false.
  - clone(): NumberTester*
    - This function should return a new IsDivisible pointer initialized with the inherited variable member.
  - getNumAliveObjects(): int
    - This is a static function.
    - This function should return the static numAliveObjects member of ValueIndependantTester class.

5.5 IsGreater

This class has a **public inheritance** relationship with ValueDependantTester.

- members:
  - numAliveObjects: int
    - This is a static variable of the IsGreater class.
    - This variable will keep track of the number of current instantiated IsGreater objects.
    - This variable should be initially initialized to a value of 0.

- functions:
  - IsGreater(int value):
    - This is the constructor for the IsGreater class.
    - This function should increment the numAliveObjects variable.
    - This function should also initialize the inherited value member with the passed in member
  - ~IsGreater()
    - This is the destructor for the IsGreater class.
    - This function should decrement the numAliveObjects variable.
  - evaluate(val: int): bool
    - This function should determine if the passed in value is strictly greater than the inherited value member.
    - If it is then, the function should return true else false.
  - clone(): NumberTester*
    - This function should return a new IsGreater pointer initialized with the inherited variable member.
  - getNumAliveObjects(): int
    - This is a static function.
    - This function should return the static numAliveObjects member of IsGreater class.

5.6 IsSmaller

This class has a **public inheritance** relationship with ValueDependantTester.

- members:
  - numAliveObjects: int
    - This is a static variable of the IsSmaller class.
    - This variable will keep track of the number of current instantiated IsSmaller objects.
    - This variable should be initially initialized to a value of 0.
- functions:
  - IsSmaller(int value):
    - This is the constructor for the IsSmaller class.
    - This function should increment the numAliveObjects variable.
    - This function should also initialize the inherited value member with the passed in member
  - ~IsSmaller()
    - This is the destructor for the IsSmaller class.
    - This function should decrement the numAliveObjects variable.
  - evaluate(val: int): bool
    - This function should determine if the passed in value is strictly smaller than the inherited value member.
    - If it is, then the function should return true else false.

  - clone(): NumberTester*
    - This function should return a new IsSmaller pointer initialized with the inherited variable member.
  - getNumAliveObjects(): int
    - This is a static function.
    - This function should return the static numAliveObjects member of IsSmaller class.

5.7 IsEvenOdd:

This class has a **public inheritance** relationship with ValueIndependantTester.

- members:
  - numAliveObjects: int
    - This is a static variable of the IsEvenOdd class.
    - This variable will keep track of the number of current instantiated IsEvenOdd objects.
    - This variable should be initially initialized to a value of 0.
- functions:
  - IsEvenOdd():
    - This is the constructor for the IsEvenOdd class.
    - This function should increment the numAliveObjects variable.
  - ~IsEvenOdd()
    - This is the destructor for the IsEvenOdd class.
    - This function should decrement the numAliveObjects variable.
  - evaluate(val: int): bool
    - This function should test if the passed in value is even or odd.
    - If it is even the function should return true else false.
  - clone(): NumberTester*
    - This function should return a new IsEvenOdd pointer.
  - getNumAliveObjects(): int
    - This is a static function.
    - This function should return the static numAliveObjects member of IsEvenOdd class.

5.8 IsPrimeNumber:

This class has a **public inheritance** relationship with ValueIndependantTester.

- members:
  - numAliveObjects: int
    - This is a static variable of the IsPrimeNumber class.
    - This variable will keep track of the number of current instantiated IsPrimeNumber objects.
    - This variable should be initially initialized to a value of 0.

- functions:
  - IsPrimeNumber():
    - This is the constructor for the IsPrimeNumber class.
    - This function should increment the numAliveObjects variable.
  - ~IsPrimeNumber()
    - This is the destructor for the IsPrimeNumber class.
    - This function should decrement the numAliveObjects variable.
  - evaluate(val: int): bool
    - This function should test if the passed in value is a prime number or not.
    - If it is a prime number the function should return true else false.
  - clone(): NumberTester*
    - This function should return a new IsPrimeNumber pointer.
  - getNumAliveObjects(): int
    - This is a static function.
    - This function should return the static numAliveObjects member of IsPrimeNumber class.

5.9 TesterInterface

- members:
  - maxNumTesters: int
    - This is the maximum number of testers that can be held by the interface.
    - This is also the size of the testers array.
  - currNumTesters: int
    - This is the current amount of tests held by the interface.
  - testers: NumberTester**
    - This is a dynamic array of dynamic NumberTester objects.
    - The array should initially be populated with nulls.
- functions:
  - TesterInterface(maxNumTests: int)
    - This is the parameterized constructor for the TesterInterface class and should initialize all appropriate member variables.
    - If maxNumTests is less than 1 then initialize the array with a size of 0 and initialize maxNumTests with 0.
  - TesterInterface(other: TesterInterface*)
    - This is a copy constructor for the TesterInterface class.
    - This function should make a deep copy of the passed in parameter
    - If the passed in parameter is NULL initialize all the int variables with a value of 0 and the array with a size of 0.
    - *Hint: remember there is a difference between a null array and an array with a size of 0*
  - TesterInterface(other: TesterInterface&)
    - This is a copy constructor for the TesterInterface class.
    - This function should make a deep copy of the passed in parameter

- ~TesterInterface()
  - This is the destructor for the TesterInterface class.
  - This function should deallocate all the dynamic memory allocated.
- addTester(tester: NumberTester*): int
  - This function should add a deep copy of the passed in NumberTester object to the first index containing null of the testers array and increment the currNumTesters.
  - The function should return the index that the new object was inserted into.
  - If the passed in parameter is null the function should return -1 and not alter the array.
  - If the array is full the function should return -1 and not alter the array.
  - *Hint: use the clone function to make a deep copy of the passed in parameter*
- removeTester(pos: int): bool
  - This function should remove the NumberTester* at the passed in parameter's index in the array and decrement the currNumTesters variable.
  - The NumberTester* object should be deleted and set to null.
  - If the function was able to succesfully remove the NumberTester then the function should return true.
  - If the passed in parameter's index in the array is null then the function should return false.
  - If the passed in parameter's index is outside the bounds of the array the function should return false.
- evaluate(num: int): bool
  - This function should iterate through all the NumberTesters currently in the testers array and pass the passed in parameter to the their evaluate function.
  - If all the NumberTesters in the array return true then the function should return true else it should return false.
  - If the array is empty the function should return false.
- failedTests(num: int): int*
  - This function should return an array containing all the indexes of tests that returned an evaluation result of false.
  - The array should be sized exactly to the number of tests that failed.
  - If no tests failed or if the array does not contain any NumberTesters the function should return an array of size 0.
- numberOfFailedTests(num: int): int
  - This function should return the amount of NumberTester's evaulations that failed.
  - If no tests failed or if the array is empty the function should return 0.
- [ ] (pos: int): NumberTester*
  - This is the overloaded form of the operator[ ].
  - This function should return the pointer at passed in index.
  - If the index is outside of the bounds of the array the function should return null.
- const getCurrNumTesters(): int
  - This is a const function and should return the currNumTesters variable.
  - *In the appropriate h file decalre the function as: int getCurrNumTesters() const.*
- const getMaxNumTesters(): int
  - This is a const function and should return the maxNumTesters variable.
  - *In the appropriate h file decalre the function as: int getMaxNumTesters() const.*

6 Source files

- IsDivisible.h,cpp
- IsEvenOdd.h, cpp
- IsGreater.h, cpp
- IsPrimeNumber.h, cpp
- IsSmaller.h, cpp
- NumberTester.h, cpp
- TesterInterface.h, cpp
- ValueDependantTester.h, cpp
- ValueIndependantTester.h, cpp

7 Allowed libraries

- cstddef
  - Note this is imported such that the NULL constant is defined.
  - Add this to all your h files.

8 Submission

You need to submit your source files, only the cpp files, on the Fitch Fork website (https://ff.cs.up.ac.za/). All methods need to be implemented (or at least stubbed) before submission. Place the above mentioned files in a zip named uXXXXXXXX.zip where XXXXXXXX is your student number. There is no need to include any other files or h files in your submission. Your code should be able to be compiled with the C++98 standard
For this practical you will have 10 upload opportunities. Upload your archive to the Practical 5 slot on the Fitch Fork website.