

CPSC 327

Project 5

Teams: None, please work individually on this project

References:

1. Building and linking to a Static Library lectures and projects
2. Pointers Memory lectures and projects
3. Classes, Objects lectures and projects

Sample Code:

See 2 starter projects on course website

Topics covered by this project;

- Creating and using a static library
- Using pointers to manipulate objects
- Using vectors to hold objects and pointers
- Class Heiarchies
- Abstract Base Classes
- Polymorphism
- Composition

Class Heiarchy

You are developing a class hierarchy for this project. An Abstract Base Class (ABC), 'Smalltalk' defines the hierarchy behaviour. Most of your work in this class structure will take place in Smalltalk.

Classes derived from Smalltalk must implement `populatePhrases()`. A function that initializes the baseclass vector with phrases that are unique to that class type. For instance, `Smalltalk_American` will populate its internal vector of strings with the american phrases found in `constants.h`.

Additionally you are given a complete watch object. You may give or take a watch from any instance of `Smalltalk_American`, `ST_American_DonutEnthusiest` or `Smalltalk_Brit`. Note that watches cannot be created out of thin air, if you give one to

an instance you no longer have that watch, the instance does. So this is one case where a shallow pointer copy is appropriate. See Smalltalk.h for further guidance.

Please also provide a function (as specifies in Functions.h and outlined in Functions.cpp) that generates a vector of unique pointers to objects derived from smalltalk. Please pay attention to the hints I've left you in the implementation.

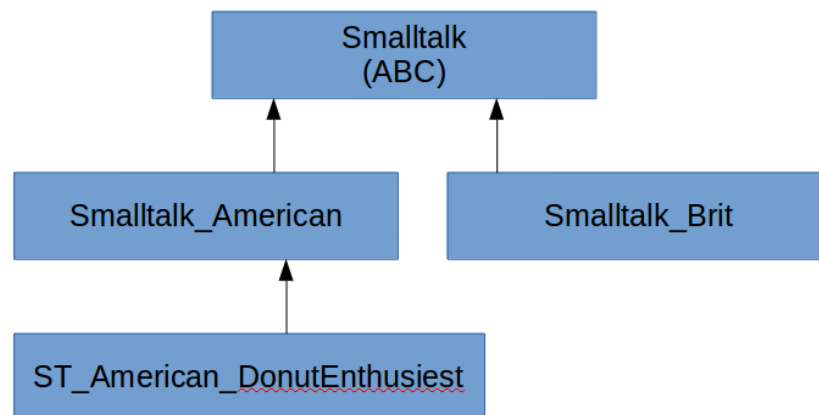
Please compile both projects using the C++11 language standard.

Library

I would like you to develop a static library with the following name and file structure.

- 327_Proj5_Lib
 - Archives
 - Includes
 - includes
 - constants.h
 - Functions.h
 - Smalltalk_American.h
 - Smalltalk_Brit.h
 - Smalltalk.h
 - ST_American_DonutEnthusiest.h
 - Watch.h
 - Debug
 - Functions.cpp
 - Smalltalk_American.cpp
 - Smalltalk_Brit.cpp
 - Smalltalk.cpp
 - ST_American_DonutEnthusiest.cpp
 - Watch.cpp

All classes inherit publicly. The class hierarchy is as follows;



I have given you the header files and some of the implementation.

Testing

Please develop a test application that has the following name and file structure:

```
▼ [C] > 327_Proj5
  ▶ [B] Binaries
  ▶ [I] Includes
  ▶ [D] > Debug
  ▼ [C] > src
    ▶ [C] > 327_Proj5.cpp
```

This application should link statically to the above library. The projects as they appear in the eclipse workspace.

```
▶ [C] > 327_Proj5
▶ [C] > 327_Proj5_Lib
```

Please be sure to test your library thoroughly. I will use my own test harness.

Submission:

Only the following 6 files. Please do not zip them together, or embed them in a directory structure. Just the 6 files.

327_Proj5.cpp

Functions.cpp

Smalltalk_American.cpp

Smalltalk_Brit.cpp

Smalltalk.cpp

ST_American_DonutEnthusiest.cpp

Grading:

For each concrete class (remember to push as much common functionality as possible into base classes! This cuts down on repetitive code in derived classes)

5% Submission instructions followed

25% getPeople populated correctly with unique pointers

25% populatePhrases implemented correctly

saySomething cycles correctly through available phrases

25% takeWatch, giveWatch, getTime correct

10% valgrind returns no errors

10% Code style (refactor repetitive code into functions, no magic numbers, no large blocks of empty space, no large chunks of commented out code, pushing as much functionality in base class as possible)

Helpful Bits

If it helps here is what the main of my tester looks like, it should give you an idea of how I will test.

```
//enum nationality{ST_A,ST_A_DE,ST_B};
int main() {
    //test getPeople
    test_getPeople(0,0);    //test with 0 people 0 watches
    test_getPeople(4,1);    //3*4 = 12 people 1 watches
    test_getPeople(4,10);   //3*4 = 12 people 12 watches
    test_getPeople(4,20);   //3*4 = 12 people 20 watches
    total_points +=1;        //free point

    //test order of phrasing
    Smalltalk_American sta;
    testPhrasing(ST_A,&sta);    //run through utterances, make sure they
                                //iterate in the right order and correctly
                                //restart at beginning once all phrases uttered

    ST_American_DonutEnthusiast stade;
    testPhrasing(ST_A_DE,&stade); //same as above but will go through both
                                //ST_A then ST_DE utterances
                                //before it restarts at beginning

    Smalltalk_Brit stb;
    testPhrasing(ST_B,&stb);

    //testWatchMove with just 1 watch, giving and taking it from all 3 derived
    //classes. Verify correct pointer movement, verify derived class responds
    //appropriately when given a watch and when watch is taken, (and when watch taken twice)
    testWatchMove();
}
```