Homework Assignment

Class:	CS202	**Semester:**	Fall 2019
Assignment type:	Homework assignment	**Due date:**	9/30/19
Assignment topic:	Classes: objects and inheritance	**Assignment no.**	3
Delivery:	WebCampus – cpp files		

Goal

Practice the classes and inheritance, constructors, destructors.

Input to the program

- Input is internal: classes are used to create objects

Procedure for the implementation

- There are two stages in this assignment.
- Each stage is a separate cpp file
- Stage 2 requires code from stage 1

Stage 1

In this stage the class for Coffee Machine is implemented. This coffee machine has resources (water and ground coffee) that are used to prepare the product – cup of coffee.

Declare global const integer value: DEFAULT_CAPACITY with value 10.
Resources required to produce one cup of coffee:

- 1 cup of water
- ***coffee_spoons_per_cup****spoon_of_coffee

This means that in order to produce 1 cup of coffee, the machine needs to utilize 1 cup of water and 1 spoon of ground coffee (assuming ***coffee_spoons_per_cup*** is set to 1 at this point).

Implement *CoffeeMachine* class, with the following members (*DECAF* is used as an example value of *name* member). Implement all the functions outside of the class definition. Class members:

- public:
  - ***string name:*** the name of the coffee machine
  - ***int makeCups(int)***: coffee machine makes *int* cups of coffee. Invokes *makeSingleCup()* function to make single cup of coffee (separate call for each cup). For example, we can *order* 5 cups of coffee using this function. Function prints:
    `Ordered int cups of coffee (DECAF) of strength 1`

    - If there's not enough supplies to prepare the order, then the order is rejected, and the following message is displayed
      `NOT ENOUGH RESOURCES!`
      and function *makeCups* returns 0.
    - If there are enough resources, then function executes *makeSingleCup* multiple times to prepare ordered coffees and at the end function returns 1. Strength = value of *coffee_spoons_per_cup*
    - Summary of return values:

0 = not enough resources to prepare order
1 = enough resources, order prepared.

- ***void addWater(int)*:** adds *int* cups of water to the water container. Checks if not overflow – if requested more than the current free space in the container, then fills to the full capacity
- ***void addCoffee(int)*:** adds *int* spoons of coffee to the coffee container. Checks if not overflow – if requested more than the current free space in the container, then fills to the full capacity
- ***void setCoffeeSpoonsPerCup(int)* –** sets the *coffee_spoons_per_cup* member to a new *int* value
- ***displayCM()*:** prints the following output:

```
Current state of CM: Decaf
 WATER:  10 / 10 (cups)
 COFFEE: 10 / 10 (spoons)
 STRENGTH: 1 coffee spoons per cup
```

- private:
  - ***void makeSingleCup()*:** produces single cup of coffee. Does not check if the supplies are available (this is done by *makeCups(int)* function). This function modifies *curr_water* and *curr_coffee* members. Outputs the following message:
    `...made cup of coffee... (DECAF)`
  - ***int water_capacity***: the capacity of the water container, measured in cups.
  - ***int coffee_capacity***: the capacity of the coffee container, measured in spoons.
  - ***int curr_water*:** current amount of water in the water container, measured in cups.
  - ***int curr_coffee*:** current amount of coffee in the coffee container, measured in spoons.
- protected:
  - ***int coffee_spoons_per_cup*** : integer value saying how many spoons of coffee is used to prepare a single cup of coffee

Declare the following constructors:

- default constructor, no parameters. Initializes *curr_water* and *curr_coffee* to 0, *water_capacity* and *coffee_capacity* to *DEFAULT_CAPACITY*, *coffee_spoons_per_cup* to 1. Name is set to "*UNTITLED*". Outputs the message:
  `Created Coffee Machine UNTITLED with empty resources.`

- constructor that just initializes the *name* of the coffee machine and rest of the values same way as in default constructor. Outputs the message:
  `Starting up Coffee Machine DECAF with empty resources.`

- constructor that initializes the *CoffeeMachine* object setting the following parameters: *name*, *water_capacity* and *coffee_capacity* – set to values given as parameters to the constructor. Values of *curr_water* and *curr_coffee* are set to 0. Value of *coffee_spoons_per_cup* is set to 1. Outputs the message:

```
Starting up Coffee Machine DECAF with empty resources and capacities:
water_capacity=X
coffee_capacity=Y
```

Declare the class destructor that will give the following output message:

```
Shutting down Coffee Machine DECAF with the following resources:
 water:  (curr_water value)
 coffee: (curr_coffee value)
```

DECAF in this example is the name stored in *name* string member.

Sample test for stage 1

Please note, that this is a sample test. Your program will be tested with this and other cases. Use the test below only as an example. Perform other tests that you consider suitable. The following test steps can be placed in the *main()* function. For your tests, include manipulating the value of *coffee_spoons_per_cup* – so the machine makes stronger coffee using more ground coffee.

- Set DEFAULT_CAPACITY to 10
- Declare object *cm1* of *CoffeeMachine* type, using parameterless constructor
- Fill water to 8 cups, add 8 spoons of coffee using appropriate member functions
- Order 5 cups of coffee
- Display state of machine
- Declare object *cm2* of *CoffeeMachine* type, using constructor with single parameter. Use "*DECAF*" as the name.
- Fill water and coffee resources to maximum
- Display machine state
- Order 14 cups of coffee
- Declare object *cm3* of *CoffeeMachine* type, using the constructor with multiple parameters. Use *"BLEND"* as the name, water capacity = 15, coffee capacity = 20
- Fill water to 14, coffee to 20
- Display machine state
- Order 12 cups of coffee
- Display machine state
- Order 5 cups of coffee

Submission for stage 1: submit file *rebelmailid_1.cpp* file that implements all the above functionality.

Sample output for stage 1

```
created coffee machine UNTITLED with empty resources
Current state of CM: UNTITLED
 WATER:  8 / 10 (cups)
 COFFEE: 8 / 10 (spoons)
 STRENGTH: 1 coffee spoons per cup

 ordered 5 cups of coffee of strength 1
...made cup of coffee UNTITLED...
...made cup of coffee UNTITLED...
...made cup of coffee UNTITLED...
...made cup of coffee UNTITLED...
...made cup of coffee UNTITLED...

Current state of CM: UNTITLED
 WATER:  3 / 10 (cups)
 COFFEE: 3 / 10 (spoons)
 STRENGTH: 1 coffee spoons per cup

created coffee machine DECAF with empty resources
Current state of CM: DECAF
 WATER:  10 / 10 (cups)
 COFFEE: 10 / 10 (spoons)
 STRENGTH: 1 coffee spoons per cup

 ordered 14 cups of coffee of strength 1
NOT ENOUGH RESOURCES!

Starting up Coffee Machine BLEND with empty resources and capacities:
        water_capacity=15
        coffee_capacity=10

Current state of CM: BLEND
 WATER:  14 / 15 (cups)
 COFFEE: 20 / 20 (spoons)
 STRENGTH: 1 coffee spoons per cup

 ordered 12 cups of coffee of strength 1
...made cup of coffee BLEND...
...made cup of coffee BLEND...
...made cup of coffee BLEND...
...made cup of coffee BLEND...
...made cup of coffee BLEND...
...made cup of coffee BLEND...
...made cup of coffee BLEND...
...made cup of coffee BLEND...
...made cup of coffee BLEND...
...made cup of coffee BLEND...
...made cup of coffee BLEND...

Current state of CM: BLEND
 WATER:  2 / 15 (cups)
 COFFEE: 8 / 20 (spoons)
 STRENGTH: 1 coffee spoons per cup

 ordered 5 cups of coffee of strength 1
NOT ENOUGH RESOURCES! ABORT.

shutting down Coffee Machine BLEND with the following resources left:
 water: 2
 coffee: 8

shutting down Coffee Machine DECAF with the following resources left:
 water: 10
 coffee: 10

shutting down Coffee Machine UNTITLED with the following resources left:
 water: 3
 coffee: 3
```

Stage 2

This stage is valid only with stage 1 submitted. Create this stage as the separate file that extends stage 1 file. This means, that the cpp file for stage 2 will include:

- Code of the class *CoffeeMachine* from stage 1
- Code of the derived class described in stage 2
- *main()* function with your tests.

In this stage, you define the *MilkCoffeeMachine* class (derived) that extends the *CoffeeMachine* class (base). The new class is for a machine, that prepares coffee with milk. This way, there is an additional resource – milk – and also the new integer variable ***milk_spoons_per_cup*** that tells how many spoons of milk is used per each cup of coffee. New class derives from base class and uses all the functionalities of *CoffeeMachine* class.
You can imagine that you have a physical working coffee machine (object of a base class) and then you physically attach the milk installation – so you have the milk coffee machine now (object of a derived class).

Resources required to produce one cup of coffee with milk:

- 1 cup of water
- ***coffee_spoons_per_cup*** spoons of coffee
- ***milk_spoons_per_cup*** spoons of milk

Derive new class *MilkCoffeeMachine* from *CoffeeMachine* using public inheritance

- Add private members:
  - ***int milk_capacity***: the capacity of the milk container, measured in spoons.
  - ***int curr_milk*****:** current amount of milk in the milk container, measured in spoons.
- Add protected members:
  - ***int milk_spoons_per_cup*** spoons of milk per each cup of coffee
- Add public members:
  - ***void addMilk(int)*****:** adds *int* spoons of milk to the milk container. Checks if not overflow – if requested more than the current free space in the container, then fills to the full capacity
  - ***void setMilkSpoonsPerCup(int)* –** sets the *milk_spoons_per_cup* member to a new *int* value
  - Add constructor that will set name, *water_capacity*, *coffee_capacity*, *milk_capacity*. To set *name*, *water_capacity* and *coffee_capacity* – add call to the appropriate constructor from the base class. Set the *milk_spoons_per_cup* to value of 1.
  - Override ***makeCups(int)*** function:
    - Call the *makeCups* function from the base class (to prepare plain coffee without milk). Test the return value to see if there are enough resources. If plain coffee is prepared successfully (there were enough water and milk) then proceed with adding milk to each cup (*milk_spoons_per_cup* per each cup)
    - If you add milk to coffee, display (1 is a value of *milk_spoons_per_cup*):
      ```
      Adding 1 spoons of milk per cup
      ```

- While adding milk to the cups of plain coffee, machine can run out of milk. If this happens, then display message:
  `X of Y cups have milk added. Insufficient milk for Z cups.`
  This is related to the situation, where Y cups of coffee with milk were ordered, but there was milk enough just for X cups. Z cups remain plain coffee.

- Override ***displayCM()*** function, so it includes milk

```
Current state of CM: Decaf
 WATER:  10 / 10 (cups)
 COFFEE: 10 / 10 (spoons)
 STRENGTH: 1 coffee spoons per cup
 MILK:   5 / 10 (spoons)
 MILK PART: 1 milk spoons per cup
```

The derived class won't have access to the private members of the base class. It's not allowed to modify the base class.

Sample test for stage 2:

- Declare object *cm1* of *MilkCoffeeMachine* type, using constructor with parameters. Set name to DECAF, set all capacities to 8.
- Fill water to 8 cups, add 8 spoons of coffee, add 8 spoons of milk using appropriate member functions
- Order 8 cups of milk coffee
- Declare object *cm2* of *MilkCoffeeMachine* type, using constructor with parameters, *name* = BLEND, *water_capacity*=15, *coffee_capacity*=30, *milk_capacity* = 26
- Set *milk_spoons_per_cup* to 2
- Display machine state
- Order 10 milk coffees
- Display machine state
- Order 5 milk coffees
- Display machine state

Submission for stage 2: submit file *rebelmailid_2.cpp* file that implements all the Stage 2 functionality (Stage 1 code + Stage 2 modifications).

Submission:

Include the following elements in your submission: (rid = your rebel id)

Problem	Element	File
1	Code of your program (for stage 1)	rid_1.cpp file
2	Code of your program (for stage 2)	rid_2.cpp file
	Summary of the submission	
	Summary: 2 cpp files, submit them to the WebCampus. Remember about proper names of the files!	