COS 2021 – Fundamental Data Structures – Spring 2024

Homework 3 (Sorting)

Documentation

#include directive

* #include <iostream> - calling exceptions and console output
* #include <string> - using all properties of the type string
* #include <vector> - using vectors

using namespace std; - used for calling exceptions, checking whether there is a given GUID in the GUIDs vector, console output and to save time to not write everywhere ” std::” where it is required

enum DiscountType { amount, percentage }; - creating a enum for discount with two options – amount or percentage that can be called anywhere in the program.

vector<string> GUIDs; - used to check whether there is already a division with a specific GUID before creating a new one with the same GUID

class Division

* **private** members – the data slots
  + GUID (string)
  + Name (string)
  + PhoneNumber (string)
  + Description (long string)
  + Parent (Division)
  + void checkGUID(string guid) – checks whether there is already a division with a specific GUID in the vector GUIDs before creating this new one with the same GUID
    - If there is, it throws an invalid argument exception “There is already a division with this GUID.”.
    - If there is not, it proceeds to create the new division.

It is called in every constructor of the class Division.

* **public** members (using const )
  + Constructors (using the const keyword to prevent the new data from modification)
    - Standard (empty) constructor with default data for all data slots
      * GUID (string) – e.g. ABCD-EFGH-1234 through void checkGUID(“ABCD-EFGH-1234”)
      * Name (string) – e.g. Electronics
      * PhoneNumber (string) – e.g. 0888 123456
      * Description (long string) - empty
      * Parent (Division) – NULL pointer
    - Argument constructor – used to create a division with previously set data.
      * checkGUID(“”) is called here as well.
      * One of the arguments “Division\* parent” is by default = nullptr.
        + In case this constructor is used and the user

doesn’t define a parent, the division is said to not have any and gives the Parent value nullptr.

define a parent the Parent is given a value.

* + - Copy constructor – used to create a copy of another division in the currently generated new division
      * All of the data is being replicated, including the GUID.
  + Assignment operator= - it does the same operation just like the copy constructor

class Item – an abstract class

* **public** members
  + **virtual** double GetTotalPrice() const = 0;
  + **virtual** string GetName() const = 0;
  + **virtual** string GetCategory() const = 0;
    - using the **const** keyword to prevent data modification
    - virtual – allows us to get an access to Artifact’s GetName(), GetCategory() and GetTotalPrice() in the driver program, as well as Service’s ones (since Service inherits Artifact)
    - The **= 0** at the end of the function declaration signifies that the functions are pure virtual, meaning they have no implementation in the Item class itself. Pure virtual functions must be overridden by any concrete derived class before that class can be instantiated.

class Artifact (inherits Item)

* **private** members – the data slots
  + GUID (string) – we don’t need to check the GUID for Artifact since nowhere is required
  + Name (string)
  + Description (long string)
  + Category (string) – the artifact category
  + Division (Division) – division to which the artifact belongs; cannot be empty
  + Price (double) **- in Bulgarian leva.**
  + Discount (double) – used to reduce the price for the buyer
  + DiscountType (enum) – amount or percentage – depending of the type/reason the buyer gets
  + Quantity (number (int)) – the quantity that the person buys
* **public** members
  + Constructors (using the const keyword to prevent the new data from modification)
    - Standard (empty) constructor with default data for all data slots
      * GUID (string) - empty
      * Name (string) – e.g. Sony Vaio
      * Description (long string) - empty
      * Category (string) – the artifact category – e.g. Laptop
      * Division (Division) – new empty Division
      * Price (double) – e.g. 185.99 BGN
      * Discount (double) – e.g. 30
      * DiscountType (enum) – **amount (chosen** at random**)** or percentage
      * Quantity (number (int)) – 0
    - Argument constructor – used to create an artifact with previously set data.
      * Here we check whether the given price, discount and/or the quantity are less than 0. They cannot be negative numbers!
        + If any of them is, then the program throws an invalid argument exception.
        + If all of them are positive, the program continues.
    - Copy constructor – used to create a copy of another artifact in the currently generated new artifact
      * All of the data is being replicated, including the GUID.
  + Assignment operator= - it does the same operation just like the copy constructor
  + Methods (using the const keyword to prevent data modification)
    - double GetEffectivePrice()
      * Calculating the effective price in terms of what type of discount we have.
        + If it is an amount, we just reduce the price by it.
        + If it is a percentage, we calculate it by it.
      * Here we check whether the effective price is less than 0. It cannot be a negative number!
        + If it is, then the program throws an invalid argument exception.
        + If it is a positive, the program returns the effective price.
    - double GetTotalPrice()
      * Calculating the total price of the artifact by multiplying the effective price with the quantity the buyer requested.
      * Here we check whether the total price is less than 0. It cannot be a negative number!
        + If it is, then the program throws an invalid argument exception.
        + If it is a positive, the program returns the total price.
      * Abstraction
        + Artifact inherits Item. Therefore, when we call it from a pointer of Item with a data value of Artifact, this function will be used.
        + Also, there is no longer need to have **virtual** keyword in this instance of GetTotalPrice(). When Service calls the function, it automatically takes it to its instance.
        + When Service class inherits an Artifact, the requirements for getting the total price changes.

Other variables are needed which are not presented in Artifact when inherited by Service

Not to break the calculations of service and use only a part of the data.

* + - string GetName() - returning the name of the Artifact
    - string GetCategory() - returning the category of the Artifact

class Service (Inherits Artifact)

* **private** members – the data slots
  + Since the class inherits Artifact, we don’t need to implement again the previous data members (GUID, Name, etc,).
  + Duration (double) (service execution duration in hours)
  + Rate (double)
  + RateDiscount (double)
  + RateDiscountType – amount or percentage
* **public** members
  + Constructors (using the const keyword to prevent the new data from modification)
    - Standard (empty) constructor with default data for all data slots
      * It also inherits the Artifact empty constructor as we don’t need to present default values of the other data slots.
      * Duration (double) – e.g. 1.5 hours
      * Rate (double) – e.g. 8.50
      * RateDiscount (double) – e.g. 30
      * RateDiscountType – amount or **percentage (chosen** at random**)**
    - Argument constructor – used to create a service with previously set data.
      * It also inherits the Artifact argument constructor as we don’t need to present values of the other data slots.
      * Here we check whether the given duration, rate and/or the rateDiscount are less than 0. They cannot be negative numbers!
        + If any of them is, then the program throws an invalid argument exception.
        + If all of them are positive, the program continues.
    - Copy constructor – used to create a copy of another service in the currently generated new artifact
      * It also inherits the Artifact copy constructor as we don’t need to present values of the other data slots.
      * All of the data is being replicated, including the GUID.
  + Assignment operator= - it does the same operation just like the copy constructor
  + Methods (using the const keyword to prevent data modification)
    - double GetEffectivePrice() and string GetName() are inherited and there is no need to modify them.
      * That’s the reason why they are not virtual in Artifact class.
    - double GetEffectiveRate()
      * Calculating the effective rate in terms of what type of rate discount we have.
        + If it is an amount, we just reduce the rate by it.
        + If it is a percentage, we calculate it by it.
      * Here we check whether the effective rate is less than 0. It cannot be a negative number!
        + If it is, then the program throws an invalid argument exception.
        + If it is a positive, the program returns the effective rate.
    - double GetTotalPrice() – inheriting the one in Artifact class
      * Calculating the total price of the service by getting the total price through the instance of the method in the Artifact parent and adding the effective rate multiplied by the duration.
        + from virtual – when Service class inherits an Artifact, the requirements for getting the total price changes.

Other variables are needed which are not presented in Artifact when inherited by Service

Not to break the calculations of service and use only a part of the data.

* + - * Here we check whether the total price is less than 0. It cannot be a negative number!
        + If it is, then the program throws an invalid argument exception.
        + If it is a positive, the program returns the total price.

Implementing an Insertion Sort and Quicksort algorithms (according to the presentation from our class on 02/13/2024) so that they can be used to sort the vector of Items based on their total price.

* void insertionSort(vector<Item\*>& a)
  + Insertion Sort works by iterating through an array, selecting one element at a time, and placing it in its correct position relative to the elements already sorted. It repeatedly picks the next unsorted element and inserts it into the sorted portion of the array until the entire array is sorted.
  + Taking a vector of Item pointers so that we can order the elements in relation to a common variable in them - their total price.
* void quicksort(vector<Item\*>& a, int low, int high)
  + Quicksort is a divide-and-conquer algorithm that selects a 'pivot' element and partitions the array into two sub-arrays, with elements smaller than the pivot on one side and larger on the other. It then recursively applies this process to each sub-array.
  + Using the Insertion Sort algorithm for small subarrays. Else, we go for Quicksort.
  + Taking a vector of Item pointers so that we can order the elements in relation to a common variable in them - their total price.
    - Also taking the beginning index of the vector as “low” and the end index as “high”.

int main() – testing the program as it is required by the task.

1. Creating a vector pointer of the respected class – Item. Artifact inherits Item and Service inherits Artifact.
2. Creating three division pointers with sample data.
   * The GUIDs are generated from a website for examples.
   * The second division has the first one as its parent.
     + To show that the argument constructor for divisions work with and without a value for a parent.
3. Creating three artifact pointers that have a correct Division from the available and instantiated divisions in the previous step.
4. Creating three service pointers, that have a valid Division.
5. Adding the three artifacts to the Items list.
6. Adding the three services to the Items list.
7. Calling the Quicksort algorithm to order the Items in the vector based on their total price.
8. Use a loop to show the Name, Category and Total Price of each artifact and service (the items) in the store.
   * Making the loop so that you can see the requested data as well as the number of the item added to the collection.
   * Rounding the total price to the 2 decimal.
9. Items.clear(); - Clearing the vector
10. Calling the *delete* operator so that all unneeded data (division1, division2, division3, artifact1, artifact2, artifact3, service1, service2, service3 and Items) gets collected as a garbage.
    * The program may be terminating after that but it is still a good practice.

*The program is tested with different values of different types of value. With the presented one, it should not give any exceptions. If it is changed so that exceptions are required, it will do so.*