**Test 2**

Design a Vehicle Rental Management System that can handle different types of vehicles.

Class Hierarchy (**3p**)

Create an abstract class Vehicle with the following protected attributes:

- model (string): The model of the vehicle.

- year (int): The manufacturing year.

- dailyRate (double): The daily rental rate, with getters and setters. If the rate is less than 10, throw *std::invalid\_argument* with the text: "Invalid daily rate!": throw std::invalid\_argument{Invalid daily rate!"};

Implement an **abstract** method *display*() to show the vehicle's details.

Overload the stream insertion operator (<<) to call the *display*() method.

Create a concrete class Car, inheriting from Vehicle, with the following additional attributes:

- seats (int): Number of seats in the car. If the number of seats is greater than 10, this function will throw a std::runtime\_error with the text “Invalid number of seats.

- fuelType (string): Type of fuel the car uses (e.g., gasoline, diesel, electric).

Override the *display*() method to show car-specific details including seats and fuel type.

Create a concrete class Bike, also inheriting from Vehicle, with the following additional attribute bikeType (string): e.g., road, mountain. Override the display() method to include bike-specific details.

Rental Agency (**3p**)

Create a class RentalAgency which can contain a collection of Vehicle objects (both Car and Bike).

- Implement the stream insertion operator to display all vehicles.

- Implement a function to add new vehicles to the inventory.

- Implement a function that displays all available vehicles for rent, filtering by vehicle type (car or bike).

*Main Function* (**3p**):

Create a RentalAgency object and add at least three cars and one bike.

Develop a console-based user interface that allows users to:

- Display all vehicles.

- Read info about a vehicle (car or bike) and add it to the agency's inventory.

- Display all vehicles of a type (car or bike). The type is specified by the user.

- Select a vehicle and display its rental charge for a given number of days.

- Write the agency's vehicle inventory in a text file.

- Exit the application.

Exception handling should cover *invalid\_argument* for pricing errors, then *std::exception*, and then catch any other exceptions.

Implement the rule of three only if and where needed.

Default (**1p**)

To use STL exceptions you must include the stdexcept header: #include<stdexcept>

Optionally, you can STL containers:

|  |
| --- |
| // Initialize the vector with 3 values  std::vector<int> myVector = {10, 20, 30};  // Add an element to the vector  myVector.push\_back(40);  // Display all the elements in the vector  for (int value : myVector) {  std::cout << value << " ";  }  for(size\_t i = 0; i < myVector.size(); ++i){  std::cout << myVector.at(i)<< " "<<myVector[i];  } |

**Grading**

If your program does not compile, the grade is 2.

Ways to lose points (max 1 point), even if your program works perfectly:

* Your classes ignore encapsulation. (-0.2 p)
* You don’t use meaningful names for your class members and methods: for example, your methods are called void m1(), void m2(char \* s) (-0.2 p)
* You don’t use indentation and you don’t separate the implementation into a header and a source file (-0.2 p)
* You don`t mark the required methods as const in their declaration, to ensure that you cannot change any member data through that method. (-0.2 p)
* You don’t use a consistent coding style. (-0.2p)