**C.W**

**Task 1: Car and Engine (Basic Composition)**

**Description**

Create two classes: Car and Engine. Demonstrate **composition** by ensuring:

* **Car has-a Engine** as a **direct member** (e.g., Engine engine; inside Car).
* The Engine object is fully contained within Car (i.e., the Car controls the Engine’s lifetime).

**Requirements**

1. **Engine**
   * Has a start() and stop() function that just prints messages (e.g., "Engine started", "Engine stopped").
   * Has a member variable (e.g., bool isRunning) to indicate if the engine is running.
2. **Car**
   * Declares a private Engine engine; as a **composition** member.
   * Provides a startCar() and stopCar() method that delegates to engine.start() and engine.stop().
   * In main() or a test function, create a Car object and call startCar() and stopCar().

**Task 2: Key Points to Demonstrate when coding**

* The Engine object is constructed **automatically** when the Car is constructed, and destroyed when the Car is destroyed (no pointers involved).
* Show how the Car fully owns the Engine’s lifetime. (Demonstrate an example)

**Garage and Cars (Aggregation with Raw Pointers)**

**Description**

Implement a Garage class that stores pointers to one or more Car objects. This setup is **aggregation**:

* The Garage does **not** own the lifetime of the cars.
* Cars can exist independently of the Garage.

**Requirements**

1. **Car**
   * Can reuse the Car class from the previous example, or create a simplified version (just a name and an ID).
2. **Garage**
   * Has a std::vector<Car\*> cars; (raw pointers) or a std::vector<std::reference\_wrapper<Car>> to store references to external cars.
   * A parkCar(Car\* car) method that adds a Car pointer to its collection.
   * A listCars() method that prints the IDs/names of all parked cars.
3. In main() (or a test function):
   * Create several Car objects **on the stack**.
   * Create one Garage object.
   * Call garage.parkCar(&car1) etc.
   * Observe that when main() ends, the cars (stack objects) are destroyed first or last? (Note that if the Garage tries to delete these pointers, it leads to undefined behavior.)

**Key Points to Demonstrate**

* **Aggregation**: The cars are **not** created or destroyed by the Garage. The Garage merely refers to them.
* Potential pitfalls if the cars go out of scope while the Garage still holds pointers. (Demonstrate this ability)

**Task 3: Library System (Composition for Catalog, Aggregation for Books)**

**Description (Multi-Class Project)**

**Design a mini library system that contains a Catalog and Library, where:**

1. **Catalog (composition)**
   * **The Library has exactly one Catalog object that is integral to its function.**
   * **The Catalog might store or manage indexes for quick search (like a map of ISBN -> Book location).**
2. **Book**
   * **Stored externally (on the heap or as a shared resource).**
   * **Each Book can belong to multiple libraries, or at least can be “loaned” between them.**
3. **Library**
   * **Has-a Catalog catalog; (composition).**
   * **Also has-a collection of Book\* or std::shared\_ptr<Book> (aggregation) to represent the books it currently owns or references.**
   * **Provide functions like addBook(...), removeBook(...), findBookInCatalog(isbn).**
   * **Possibly store the library’s address, name, etc.**
4. **Demonstrate usage in main():**
   * **Create a Library object.**
   * **Create some Book objects, add them to the library.**
   * **Show that the library’s Catalog is automatically constructed inside the Library.**
   * **Searching the catalog for a book by ISBN, etc.**

**Key Points**

* **Hybrid approach:**
  + **The Library is composed of exactly one Catalog.**
  + **The Library aggregates multiple Book objects, each possibly existing outside the Library.**

H.W

Task 4: Write a C++ program that has a class called Juice Maker with the method blend juice and grind juice which composes two classes Blend and grind. The blend juice should blend the juice (i.e. select fruits and loop the class function in the blend for about 4 to 5 seconds) and do the same for the grind class only this time you should sleep the program for 5 seconds (to mimick the grinding process).

Task 5: Write a C++ Program which consists of a class named car with attributes engine horsepower, seating\_capacity and no\_of\_speakers. You must change the values of the public attributes but via the constant function.

Task 6: Write a class called CoffeeShop, which has two data members and seven member functions:  
**Constant public Name:** a string (basically, of the shop)  
**Public Menu:** an array of items.  
**addOrder:** adds the name of the item to the end of the orders array if it exists on the menu. Otherwise, return “This item is currently unavailable!”  
**fulfillOrder:** if the orders array is not empty, return “The {item} is ready!”. If the orders array is empty, return “All ordershave been fulfilled”  
**listOrders:** returns the list of orders taken, otherwise, an empty array.  
**dueAmount:** returns the total amount due for the orders taken.  
**cheapestItem:** returns the name of the cheapest item on the menu.  
**drinksOnly:** returns only the item names of type drink from the menu.  
**foodOnly:** returns only the item names of type food from the menu.

Task 7: Write a program that uses a constant variable but is not assigned a value instead it declared in the class Student. Your task is to assign the constant value via a function. Note the value must be taken at run-time. Also you can take a student class example of this approach where the const variable is the student’s roll\_no.