ACS231012
M- Shayan Khan

Q1

a

Pure Virtual Functions :-
A Virtual functions declared with =0 in its declaration, indicating it has no implementation in the base Class.

Purpose :
Enforce implementation in derived Classes, ensuring they provid concrete behaviors for abstract concepts.
Creat interfaces that derived classes must adhere to, promoting code consistency and maintability.

Abstract Base Classes (ABCs)
A Class containing at least one pure virtual function.
- cannot be instantiated directly.
- Serve as blueprints for derived classes, defining a common interface and shared functionality.

Interfaces :
A special Kind of abstract Class where all functions are pure virtual.

- Establish a contract for derived classes, specifying the methods they must implement without providing any default behavior.

Role of Abstract Classes in Class Hierarchy Design:-

- Encapsulating Commonality: Factor out shared attributes and methods into the abstract base class, reducing code duplication and promoting reusability.

Defining Interfaces: Establish a consistent interface for interacting with objects of different derived classes, enhancing code flexibility and polymorphism.

Enforcing Implementation: Ensure derived classes provide concrete implementations for essential method, preventing incomplete classes and errors.

~~Structing~~

Structuring Hierarchies: Organize classes into meaningful relationships, establishing clear inheritance patterns for better code comprehension and maintainability.

Q1

C

Association :-

A General relationship between two classes where objects of one class are connected to objects of another class.

Represents a "uses a" or "has a" relationship, but without ownership.

object can exist independently.

- A Student class and a course class might have an association, A student can enroll in multiple course, and a course can have multiple students enrolled, but they don't depend on each other for existence.

Aggregation :-

- A "Whole-Part" relationship where one class (the whole) is composed of objects of another class (the parts).
- Whole has ownership of its part, but parts can exist independently.

eg:- A car class might have an aggregation relationship with a wheel class. A car has wheels, but wheel can

exist independently of a car.

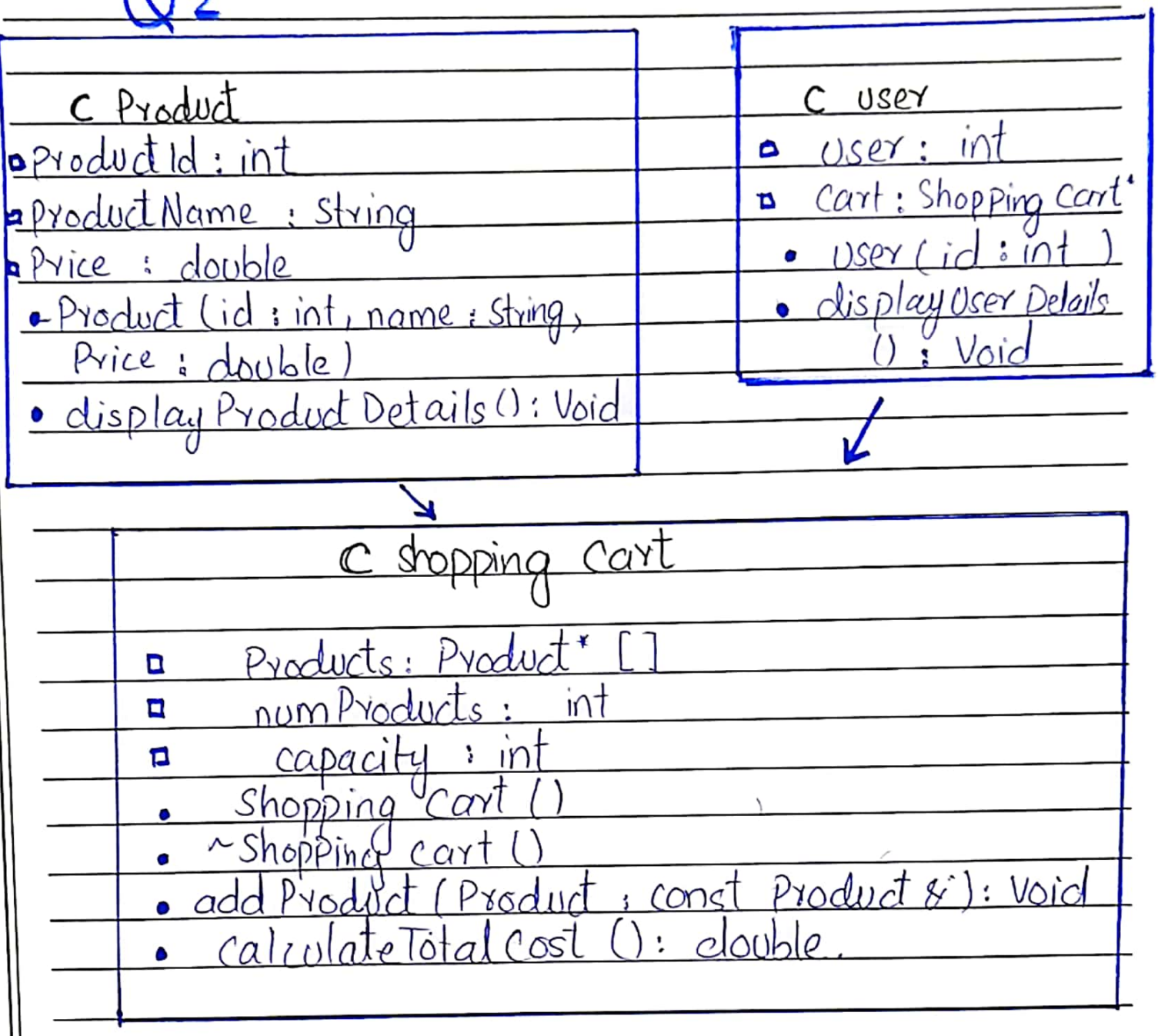
Composition :-

- A stronger form of aggregation where the parts cannot exist independently of the whole.
- Whole has a strict lifetime control over its parts.
- Parts are created when the whole is created and destroyed when the whole is destroyed.

eg :-

A House class might have a composition relationship with a Room class. A room cannot exist without a house, and if a house is demolished, its rooms cease to exist.

Q2


C Product
- ProductId : int
- ProductName : String
- Price : double
+ Product (id : int, name : String, Price : double)
+ displayProductDetails () : Void
C User
- User : int
- Cart : ShoppingCart*
+ User (id : int)
+ displayUserDelails () : Void
C ShoppingCart
- Products : Product* []
- numProducts : int
- capacity : int
+ ShoppingCart ()
+ ~ShoppingCart ()
+ addProduct (Product : const Product &) : Void
+ calculateTotalCost () : double.