**Pets shop project report**

**Group Members**

* Yousra Zaheer (1790-2023)
* Muhammad Sheharyar (1584-2023)
* Muhammad Hasnain Amjad (2013-2023)
* Syeda Maira Abid Gardazi (1981-2023)

**Introduction**

The code provided is a simulation of a pet shop management system implemented in C++. Its purpose is to manage information about different types of animals (dogs, cats, birds) that are available for sale, purchase, or viewing in the pet shop. Let's discuss how this code can be used by someone working at the front desk of a pet shop:

**Purpose of the Code:**

1. **Inventory Management**: The code allows the pet shop staff to maintain a digital inventory of animals. Each animal (dog, cat, bird) is represented by objects instantiated from respective classes (dog, cat, bird). These objects store detailed information such as name, breed, cost, age, size, etc.
2. **Customer Interaction**: It facilitates interactions with customers who visit the pet shop:
   * **Viewing Animals**: The staff can use the code to display information about available animals to customers.
   * **Buying Animals**: Customers can purchase animals, and the staff can update the inventory accordingly by adding new animal objects.
   * **Selling Animals**: The staff can also sell animals from the inventory, updating the availability and the financial records of the shop.
3. **Financial Management**: The code includes features for financial management:
   * It tracks the cost and sale prices of animals.
   * It calculates profits when animals are sold, considering a 10% profit margin in the code's current implementation.
   * It maintains a running balance of funds available to the pet shop, updating based on purchases and sales.
4. **User Interface**: Although not explicitly detailed in the provided code, the user interface (UI) would be a crucial part for the pet shop staff. This UI would interact with the backend logic (like the functions and classes provided) to provide a seamless experience for managing animals and transactions.

**Usage in a Pet Shop:**

For someone working at the desk of a pet shop, here's how they might use this code on a daily basis:

* **Displaying Animals**: When a customer inquires about available animals (dogs, cats, birds), the staff can use the code to quickly display detailed information such as breed, cost, and age.
* **Managing Sales**: During a sale, the staff can enter details like the customer's name, the animal's details, and the sale price. The code will then update the inventory and financial records automatically.
* **Adding New Animals**: When new animals arrive at the shop, the staff can add them to the inventory using the code. They input details like the animal's name, breed, cost, etc., and the system creates a new object to represent the animal.
* **Handling Customer Queries**: If a customer has specific preferences (e.g., looking for a watchdog or a friendly cat), the staff can filter and display relevant animals using the code's categorization (dogs, cats, birds).

**Benefits for Pet Shop Operations:**

* **Efficiency**: Automates and speeds up tasks such as inventory management and financial tracking, reducing manual errors.
* **Accuracy**: Ensures that all information about animals and transactions is recorded accurately and consistently.
* **Customer Satisfaction**: Provides quick access to detailed information about animals, helping customers make informed decisions.

Features Of OOPs Used in the Project

1.inheritance:

Inheritance in C++ allows one class (derived class) to inherit the properties and behaviors (member variables and functions) from another class (base class). In your provided code, inheritance is implemented as follows:

1. **Base Class (animal)**:

cpp

Copy code

class animal {

public:

string name, color, property, breed;

int cost, age, id;

float size;

// Constructor to initialize animal properties

animal(string n, string c, string p, string b, int co, int ag, int d, float s) {

name = n;

color = c;

property = p;

breed = b;

cost = co;

age = ag;

id = d;

size = s;

}

// Function to display animal information

void getdata() {

cout << endl;

cout << "Breed: " << breed << endl;

cout << "Unique Id: " << id << endl;

cout << "Name: " << name << endl;

cout << "Color: " << color << endl;

cout << "Property: " << property << endl;

cout << "Cost: " << cost << " Rs" << endl;

cout << "Life span: " << age << " years" << endl;

cout << "Size: " << size << " cm" << endl;

cout << endl;

}

};

* + animal is the base class here, which defines common properties (name, color, property, breed, cost, age, id, size) and methods (getdata()).
  + The constructor animal(string n, string c, string p, string b, int co, int ag, int d, float s) initializes these properties when an object of animal or its derived classes is created.

1. **Derived Classes (dog, cat, bird)**:

cpp

Copy code

class dog : public animal {

};

class cat : public animal {

};

class bird : public animal {

};

* + dog, cat, and bird are derived classes that inherit from the base class animal using the syntax class derived\_class\_name : access\_specifier base\_class\_name.
  + By inheriting from animal, each derived class (dog, cat, bird) inherits all the member variables (name, color, etc.) and member functions (getdata()) of the animal class.

1. **Usage in Main Function**:

cpp

Copy code

int main() {

dog a[100];

cat c[100];

bird b[100];

// Examples of using inherited functions and properties

a[1].showinfo("buddy", "Black & Tan", "watchdog", "German shepherd", 90000, 12, 1, 64.5);

a[2].showinfo("Duddle", "Brown", "protective", "Pitbull", 90000, 16, 2, 53);

// Similar usage for cats and birds

// Other operations like buying, selling, accessing data are handled based on the category

return 0;

}

* + In main(), you create arrays of objects (dog a[100];, cat c[100];, bird b[100];) based on the derived classes.
  + You then use member functions (showinfo(), getdata()) inherited from the base class animal to set and display information specific to each type of animal (dog, cat, bird).

**2.Data Hiding:**

Data hiding is a concept in object-oriented programming (OOP) that promotes encapsulation by controlling the visibility of class members (variables and methods). It helps in protecting the integrity of data by restricting access to certain parts of a class from the outside world.

**Implementing Data Hiding in the Code**

In C++, the private, protected, and public access specifiers play a crucial role in achieving data hiding:

* **Private**: Members declared as private are accessible only within the same class. They cannot be accessed directly by derived classes or by code outside the class.
* **Protected**: Members declared as protected are accessible within the same class and by derived classes. However, they are not accessible outside the class hierarchy.
* **Public**: Members declared as public are accessible from anywhere. They can be accessed freely by any part of the program.

**Refactoring Code with Data Hiding**

Let's refactor your animal class to demonstrate data hiding with appropriate access specifiers:

cpp

Copy code

#include <iostream>

#include <string>

using namespace std;

class animal {

public:

string name, color, property, breed;

int cost, age, id;

float size;

public:

// Constructor to initialize an animal object

animal(string n, string c, string p, string b, int co, int ag, int d, float s) {

name = n;

color = c;

property = p;

breed = b;

cost = co;

age = ag;

id = d;

size = s;

}

// Method to display information about the animal

void getdata() {

cout << endl;

cout << "Breed: " << breed << endl;

cout << "Unique Id: " << id << endl;

cout << "Name: " << name << endl;

cout << "Color: " << color << endl;

cout << "Property: " << property << endl;

cout << "Cost: " << cost << " Rs" << endl;

cout << "Life span: " << age << " years" << endl;

cout << "Size: " << size << " cm" << endl;

cout << endl;

}

};

**Explanation:**

1. **Private Members**: name, color, property, breed, cost, age, id, and size are now declared as private. This means they can only be accessed directly within the animal class.
2. **Constructor Usage**: The constructor animal initializes these private members. Clients of the animal class (or its derived classes) cannot directly modify or access these private members.
3. **Public Method**: getdata() remains public, allowing clients to retrieve and display information about an animal object. This method provides controlled access to the private data members.

**Using Inheritance with Data Hiding**

When you derive classes like dog, cat, and bird from animal, they inherit the public and protected members of animal. Private members of animal remain hidden from derived classes, maintaining encapsulation and preventing accidental misuse.

**Example Usage:**

cpp

Copy code

class dog : public animal {

public:

// Constructor chaining with base class constructor

dog(string n, string c, string b, int co, int ag, int d, float s)

: animal(n, c, "watchdog", b, co, ag, d, s) {

// Property is hardcoded as "watchdog" for dogs

}

};

int main() {

// Creating a dog object

dog a("Buddy", "Black & Tan", "German shepherd", 90000, 12, 1, 64.5);

// Accessing public method to display data

a.getdata();

return 0;

}

In this example:

* The dog class inherits the animal class publicly.
* The dog class constructor uses constructor chaining to initialize the base class (animal) with specific parameters, including private data members.
* getdata() method in animal is used to display the information about the dog.

**Benefits of Data Hiding:**

* **Encapsulation**: Protects the internal state of objects (name, color, etc.) from unintended access or modification.
* **Maintainability**: Easier to change internal implementations (like renaming variables) without affecting external code.
* **Security**: Prevents unintended bugs caused by direct modification of internal state by external code.

By integrating data hiding principles into your code, you enhance its readability, maintainability, and security, adhering to best practices in object-oriented design.

3.Encapsulation:

Encapsulation in object-oriented programming (OOP) is the bundling of data (attributes) and methods (functions) that operate on the data into a single unit, typically a class. It helps in controlling access to the data, ensuring that it's accessed and modified in a controlled manner. In your provided code, encapsulation is demonstrated to some extent, but there are improvements that can be made to adhere more closely to encapsulation principles.

**Encapsulation in Your Code:**

**1. Private Data Members:**

Encapsulation begins with declaring data members (name, color, property, etc.) as private in the animal class:

cpp

Copy code

class animal {

private:

string name, color, property, breed;

int cost, age, id;

float size;

public:

// Constructor to initialize an animal object

animal(string n, string c, string p, string b, int co, int ag, int d, float s) {

name = n;

color = c;

property = p;

breed = b;

cost = co;

age = ag;

id = d;

size = s;

}

// Method to display information about the animal

void getdata() {

cout << endl;

cout << "Breed: " << breed << endl;

cout << "Unique Id: " << id << endl;

cout << "Name: " << name << endl;

cout << "Color: " << color << endl;

cout << "Property: " << property << endl;

cout << "Cost: " << cost << " Rs" << endl;

cout << "Life span: " << age << " years" << endl;

cout << "Size: " << size << " cm" << endl;

cout << endl;

}

};

* **Private Members**: Attributes such as name, color, property, etc., are declared as private. This means they are accessible only within the animal class. This encapsulates the internal state of an animal object, preventing direct access or modification from outside the class.

**3. Accessor Method (getdata()):**

Encapsulation includes providing controlled access to the internal state of an object through methods. In your code, getdata() serves as an accessor method:

cpp

Copy code

void getdata() {

cout << endl;

cout << "Breed: " << breed << endl;

cout << "Unique Id: " << id << endl;

cout << "Name: " << name << endl;

cout << "Color: " << color << endl;

cout << "Property: " << property << endl;

cout << "Cost: " << cost << " Rs" << endl;

cout << "Life span: " << age << " years" << endl;

cout << "Size: " << size << " cm" << endl;

cout << endl;

}

* **Controlled Access**: getdata() allows external code to retrieve information about an animal object (name, color, etc.) in a formatted manner. This method provides controlled, read-only access to the private data members, maintaining encapsulation by not exposing the internal representation of the object directly.

**Enhancing Encapsulation:**

While your code demonstrates some aspects of encapsulation, here are additional improvements to strengthen it further:

* **Private Methods**: Besides data members, encapsulation can also involve private methods that perform operations on the data, keeping the implementation details hidden from external users.
* **Access Control**: Review access specifiers (private, protected, public) to ensure that only necessary components are exposed to the outside world, while keeping the rest private or protected.
* **Immutable Properties**: Consider making properties read-only by removing setter methods and initializing them only through constructors, if modification is not intended.

**Example Usage:**

cpp

Copy code

int main() {

// Creating an animal object

animal myAnimal("Bella", "Golden", "Guard Dog", "Golden Retriever", 60000, 12, 3, 61);

// Accessing and displaying information using getdata() method

myAnimal.getdata();

return 0;

}

In this example:

* An object myAnimal of type animal is created with specific attributes using the constructor.
* getdata() method is used to retrieve and display information about myAnimal.

**Functions in the Code:**

1. **void showinfo(string n, string c, string p, string b, int co, int ag, int d, float s) (in animal class):**
   * **Purpose:** This function sets the attributes (name, color, property, breed, cost, age, id, size) of an animal object.
   * **Usage:** Used to initialize attributes when creating instances of dog, cat, or bird objects.
2. **void getdata() (in animal class):**
   * **Purpose:** This function prints out the details (attributes) of an animal object.
   * **Usage:** Used to display the attributes of a specific animal.
3. **main() function:**
   * **Purpose:** The main function is the entry point of the program and orchestrates the interaction with the user and data handling.
   * **Usage:** It manages the menu-driven interface where users can view, buy, sell, and access information about dogs, cats, and birds.
4. **Various sections in main() for different categories (dogs, cats, birds):**
   * **Purpose:** These sections handle specific operations related to different categories of animals (dogs, cats, birds).
   * **Usage:** They initialize sample data (showinfo() calls), handle user inputs for buying or selling animals, and display information about the animals.
5. **Operations for buying (f == 1), selling (f == 2), and accessing (f == 3):**
   * **Purpose:** These operations within each category allow users to perform specific action (buying, selling, accessing information).
   * **Usage:** They manage the inventory of animals (a[], c[], b[] arrays), update the balance, and display relevant information based on user inputs.

* **Total Functions:** The primary functions are showinfo() and getdata() in the animal class, and the main() function with various sections handling operations for different animal categories.
* **Purpose:** The functions collectively serve the purpose of managing a pet shop inventory, facilitating buying and selling of animals, and providing access to detailed information about animals based on user interactions.

The code demonstrates basic inventory management functionalities for a pet shop, allowing users to interact with different categories of animals (dogs, cats, birds) through a command-line interface.

### Conclusion

In conclusion, the implemented pet shop management system provides a comprehensive solution for managing the inventory and operations of a pet store. The system allows customers to interact with different categories of animals including dogs, cats, and birds through a user-friendly menu-driven interface.

#### Key Features and Capabilities:

* **Category-based Interaction:** Customers can select and view details of dogs, cats, and birds separately, facilitating easy browsing and decision-making.
* **Buying and Selling:** The system supports buying and selling operations, enabling customers to add new pets to the inventory or sell existing ones. This functionality includes updating the inventory and calculating financial transactions in real-time.
* **Detailed Information Display:** Each animal's attributes such as name, breed, color, cost, age, and unique ID are stored and displayed accurately, providing transparency and clarity to customers.
* **Financial Tracking:** The system tracks financial transactions, ensuring that balances are updated correctly after every purchase or sale. This feature helps in maintaining financial records and monitoring profitability.

#### Benefits:

* **Efficiency:** By automating inventory management and transaction processing, the system improves operational efficiency within the pet shop.
* **Customer Satisfaction:** Customers benefit from a structured and organized system that enhances their shopping experience by providing detailed information and facilitating smooth transactions.
* **Financial Management:** The system aids in financial management by providing clear insights into sales and revenue, supporting informed decision-making and business planning.