**SESSION PLANNING SHEET**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **SLG Leaders: Pratham Garg** | | | **Date: 18-03-2024** |
|  | **Course: OOP244** | | | **Week #: 10** |
| **OBJECTIVES:** *What does the group most need to get out of this session?*: | | | | |
| * Design polymorphic objects to amplify the reusability of code * Distinguish monomorphic and polymorphic objects * Describe the difference between early binding and dynamic dispatch * Introduce pure virtual functions | | | | |
| **CO-FACILITATION CHECKLIST:** | | | | |
| Check session plans in the Leader’s Manual for inspiration  Decide who will plan and lead each activity. What support do you need from your co-facilitator?  Prepare your PowerPoint session file  Check for accessibility & compatibility Upload your session planning sheet to MS Teams  Promote your session! | | | | |
| **FACILITATOR** | **OPENING ACTIVITY:**(*Consider the time of the semester, numbers anticipated, proximity of tests, etc.)* | | | |
|  | **Ice breaker:** General questions like how’s life going. What is their progress on milestones. Is the workshops finished. What topic they are covering in the lecture what they covered last week and which topic is the most confusing. | | | |
|  | **Content/Concept** | **Activity**  *Align learning strategy to content;*  *provide instructions for participants* | **Collaborative Technique**  *How will participants work on this task together?* | |
|  | **ACTIVITY 1:** | | | |
|  | Coding question to cover polymorphic object dynamic binding and early binding. | Imagine you're building a virtual zoo management system. Your system should handle different types of animals with unique behaviors and characteristics. Implement a hierarchy of animal classes using inheritance and dynamic memory allocation, ensuring that certain traits are protected while allowing flexibility for specific behaviors.  Your hierarchy should include the following classes:  **Animal**: Base class for all animals. It should have protected data members name (string) and age (integer). Implement a constructor to initialize these members. It should have a display function to display the details.  **Mammal**: Derived from Animal, representing mammals. It should have a private data member numLegs (integer). Implement a constructor to initialize numLegs and invoke the Animal constructor. It should have a display function to display the details.  **Bird**: Derived from Animal, representing birds. It should have a private data member canFly (boolean). Implement a constructor to initialize canFly and invoke the Animal constructor. It should have a display function to display the details.  **Zoo**: A class representing a zoo that contains a dynamic array of pointers to Animal. Implement a method addAnimal() to add a new animal to the zoo, and a method displayAnimals() to display information about all animals in the zoo.  Write a C++ program that demonstrates the usage of these classes. You should dynamically allocate memory for animals, add them to the zoo, and display their information.  All your code should be compiled using this command on matrix:  g++ -Wall -std=c++11 -g -o ws main.cpp  After compiling and testing your code, run your program as follows to check for possible memory leaks (assuming your executable name is ws):  valgrind --show-error-list=yes --leak-check=full --show-leak-kinds=all --track-origins=yes ws  int main() {  Zoo zoo(5);  Mammal elephant("Dumbo", 10, 4);  Bird parrot("Polly", 3, true);  Mammal lion("Simba", 5, 4);  zoo.addAnimal(&elephant);  zoo.addAnimal(&parrot);  zoo.addAnimal(&lion);  zoo.displayAnimals();  return 0;  } | Students will work individually and them paste their answer on single doc file after they are done. | |
|  | **ACTIVITY 2:** | | | |
|  | Quiz to check understanding: | **Q1. What is the purpose of virtual functions in object-oriented programming?**  A) To enhance encapsulation  B) To optimize code execution  C) To facilitate polymorphism  D) To improve inheritance  **Q2. What does polymorphism refer to in the context of object-oriented languages?**  A) The ability to hide implementation details  B) The ability to create objects from classes  C) The ability to reuse code across different types  D) The ability to define multiple constructors  **Q3. Which statement best describes the difference between early binding and dynamic dispatch?**  A) Early binding occurs at runtime, while dynamic dispatch occurs at compile time.  B) Early binding is based on the object's dynamic type, while dynamic dispatch is based on the object's static type.  C) Early binding occurs when a function call is resolved at compile time, while dynamic dispatch occurs when it's resolved at runtime.  D) Early binding always results in faster execution compared to dynamic dispatch.  **Q4. What is the significance of declaring a destructor virtual in a base class?**  A) It prevents memory leaks in derived classes.  B) It allows dynamic allocation of objects.  C) It ensures that the most derived destructor is called during object destruction.  D) It facilitates dynamic type checking.  **Q5. What is the primary purpose of using abstract base classes in object-oriented programming?**  a) To define a class that cannot be instantiated and contains pure virtual functions  b) To define a class that provides implementations for all its member functions  c) To define a class that can be instantiated and contains concrete member functions  d) To define a class that cannot be inherited and contains static member functions  **Q6. Which of the following statements best describes a pure virtual function?**  a) A function that has a definition but no declaration  b) A function that has a declaration but no definition  c) A function that has both declaration and definition  d) A function that has neither declaration nor definition  **Q7. How is a pure virtual function declared in C++?**  a) Using the keyword "pure" before the function declaration  b) Using the keyword "virtual" before the function declaration and assigning 1  c) Using the keyword "virtual" before the function declaration and assigning 0  d) Using the keyword "abstract" before the function declaration  **Q8. What is the significance of declaring a function as pure virtual in an abstract base class?**  a) It allows the function to be accessed directly without any inheritance  b) It requires all derived classes to provide an implementation for the function  c) It prevents the function from being accessed by any client code  d) It allows the function to be overloaded with multiple definitions  **Q9. According to the notes, what is the recommended approach for performing unit tests on an interface?**  a) Conduct unit tests on concrete classes directly  b) Modify the interface for each unit test iteration  c) Perform unit tests on an abstract base class representing the interface  d) Skip unit testing as interfaces are not testable  **Q10. How does using an array of pointers to objects facilitate polymorphism in C++?**  a) It allows for direct access to object data members  b) It allows for efficient memory allocation for objects  c) It enables accessing objects of different dynamic types through a common interface  d) It ensures type safety during object instantiation |  | |
|  | **ACTIVITY 3:** | | | |
|  | Coding question | If time left as them to implement a “pure interface” named as **iAnimal** and modify the previous code to utilize it. |  | |
|  | **CLOSING ACTIVITY:** | | | |
|  |  | | | |
| **POST-SESSION REFLECTION:** | | | | |
| Take 3-5 minutes to briefly summarize your session. You may address any of these questions:   * What went well? * What didn’t? * What did people say? * What would you do differently next time? * What content / learning strategies will you cover in your next session? | | | | |
|  | | | | |

Soultions:

-------------------------------------------------------------------------------------------------------------------------------------------------------  
  
1.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <string.h>

using namespace std;

class Animal {

protected:

char\* name;

int age;

public:

Animal(const char\* \_name, int \_age) : name(new char[strlen(\_name) + 1]), age(\_age) { strcpy(name, \_name); }

~Animal() {

cout << "calling destructor of animal: " << name << endl;

delete[] name;

}

virtual void display() const {

cout << "Name: " << name << ", Age: " << age;

}

};

class Mammal : public Animal {

protected:

int numLegs;

public:

Mammal(const char\* \_name, int \_age, int \_numLegs) : Animal(\_name, \_age), numLegs(\_numLegs) {}

void display() const override {

Animal::display();

cout << ", Number of Legs: " << numLegs;

}

~Mammal() { cout << "calling destructor of Mammal: " << name << endl; }

};

class Bird : public Animal {

protected:

bool canFly;

public:

Bird(const char\* \_name, int \_age, bool \_canFly) : Animal(\_name, \_age), canFly(\_canFly) {}

void display() const override {

Animal::display();

if (canFly)

cout << ", Can Fly";

else

cout << ", Can't Fly";

}

~Bird() { cout << "calling destructor of Bird: " << name << endl; }

};

class Zoo {

private:

Animal\*\* animals;

int capacity;

int count;

public:

Zoo(int \_capacity) : capacity(\_capacity), count(0) {

animals = new Animal \* [capacity];

}

~Zoo() {

cout << "Starting destructor of zoo" << endl;

delete[] animals;

cout << "Ending destructor of zoo" << endl;

}

void addAnimal(Animal\* animal) {

if (count < capacity) {

animals[count++] = animal;

}

}

void displayAnimals() const {

cout << "Animals in the Zoo:\n";

for (int i = 0; i < count; ++i) {

animals[i]->display();

cout << endl;

}

}

};

Solution

3.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <string.h>

using namespace std;

class iAnimal {

public:

virtual void display()const = 0; // pure virtual function

virtual ~iAnimal() { cout << "calling destructor of iAnimal " << endl; } // Virtual destructor

};

class Animal : public iAnimal{

protected:

char\* name;

int age;

public:

Animal(const char\* \_name, int \_age) : name(new char[strlen(\_name) + 1]), age(\_age) { strcpy(name, \_name); }

~Animal() {

cout << "calling destructor of animal: " << name << endl;

delete[] name;

}

void display() const override {

cout << "Name: " << name << ", Age: " << age;

}

};

class Mammal : public Animal {

protected:

int numLegs;

public:

Mammal(const char\* \_name, int \_age, int \_numLegs) : Animal(\_name, \_age), numLegs(\_numLegs) {}

void giveBirth() const {

cout << name << " is giving birth!\n";

}

void display() const override {

Animal::display();

cout << ", Number of Legs: " << numLegs;

}

~Mammal() { cout << "calling destructor of Mammal: " << name << endl; }

};

class Bird : public Animal {

protected:

bool canFly;

public:

Bird(const char\* \_name, int \_age, bool \_canFly) : Animal(\_name, \_age), canFly(\_canFly) {}

void fly() const {

if (canFly)

cout << name << " is flying!\n";

else

cout << name << " can't fly.\n";

}

void display() const override {

Animal::display();

if (canFly)

cout << ", Can Fly";

else

cout << ", Can't Fly";

}

~Bird() { cout << "calling destructor of Bird: " << name << endl; }

};

class Zoo {

private:

iAnimal\*\* animals;

int capacity;

int count;

public:

Zoo(int \_capacity) : capacity(\_capacity), count(0) {

animals = new iAnimal \* [capacity];

}

~Zoo() {

cout << "Starting destructor of zoo" << endl;

delete[] animals;

cout << "Ending destructor of zoo" << endl;

}

void addAnimal(iAnimal\* animal) {

if (count < capacity) {

animals[count++] = animal;

}

else {

cerr << "Zoo is full, cannot add more animals." << endl;

}

}

void displayAnimals() const {

cout << "Animals in the Zoo:\n";

for (int i = 0; i < count; ++i) {

animals[i]->display();

cout << endl;

}

}

};

int main() {

Zoo zoo(5);

Mammal elephant("Dumbo", 10, 4);

Bird parrot("Polly", 3, true);

Mammal lion("Simba", 5, 4);

zoo.addAnimal(&elephant);

zoo.addAnimal(&parrot);

zoo.addAnimal(&lion);

zoo.displayAnimals();

return 0;

}