**Create a class hierarchy (e.g., animals with different sounds) and manage object lifetimes and relationships using smart pointers. Include error handling to gracefully handle situations where resources might not be available**.

#include <iostream>

#include <memory>

#include <string>

class Animal {

protected:

std::string name;

public:

Animal(const std::string& name) : name(name) {}

virtual ~Animal() {

std::cout << "Animal " << name << " destroyed." << std::endl;

}

virtual void makeSound() const = 0; // Pure virtual function

};

class Dog : public Animal {

public:

Dog(const std::string& name) : Animal(name) {}

~Dog() {

std::cout << "Dog " << name << " destroyed." << std::endl;

}

void makeSound() const override {

std::cout << "Dog " << name << " says Woof!" << std::endl;

}

};

class Cat : public Animal {

public:

Cat(const std::string& name) : Animal(name) {}

~Cat() {

std::cout << "Cat " << name << " destroyed." << std::endl;

}

void makeSound() const override {

std::cout << "Cat " << name << " says Meow!" << std::endl;

}

};

std::shared\_ptr<Animal> createAnimal(const std::string& type, const std::string& name) {

if (type == "dog") {

return std::make\_shared<Dog>(name);

} else if (type == "cat") {

return std::make\_shared<Cat>(name);

} else {

throw std::invalid\_argument("Unknown animal type");

}

}

int main() {

try {

std::shared\_ptr<Animal> dogPtr = createAnimal("dog", "Buddy");

std::shared\_ptr<Animal> catPtr = createAnimal("cat", "Whiskers");

dogPtr->makeSound();

catPtr->makeSound();

} catch (const std::exception& e) {

std::cerr << "Error: " << e.what() << std::endl;

return 1;

}

return 0;

}

OUTPUT:

Dog Buddy says Woof!

Cat Whiskers says Meow!

Cat Whiskers destroyed.

Animal Whiskers destroyed.

Dog Buddy destroyed.

Animal Buddy destroyed.