

Forum:[Hack Forums](#)

Thread: [Create object instance based on enumeration value](#)

chrizator - May 11, 2018, 12:25 pm

Hey folks, quick question.

Let's assume I have an Enum called 'AnimalType'. And a lot of animals inheriting from a base 'Animal' class (class Dog : public Animal).

Now, based on what 'AnimalType' I have, I want to create the corresponding derivation.

Let's assume (IDK if the shit below compiles, just writing it in HF editor):

Code enum class AnimalType

```
{  
    DogAnimal,  
    CatAnimal  
}
```

class Animal {}

class Dog : public Animal {}

class Cat : public Animal {}

I'd like to have a method that does the following:

Code Animal* getAnimalFromEnum(AnimalType type)

```
{  
    switch (type)  
    {  
        case (AnimalType::DogType):  
            return new Dog()  
            break;  
        case (AnimalType::CatType):  
            return new Cat()  
            break;  
    }  
}
```

The code above will probably work, but I find it kind of silly. Especially when working with A LOT of Enum values, this is gonna get ugly.

I'd like to have a method using unordered map or something similiar. I don't know how to achieve that. Still new to C++.

Thanks for taking the time! ?

Annabelle Lee - May 11, 2018, 2:46 pm

Using an unordered map, you can do something like this...

Code #include <iostream>

#include <functional>

#include <unordered_map>

#include <memory>

```

enum class AnimalType {
    DogAnimal,
    CatAnimal
};

// Abstract class. Let's make it require a function that prints type of animal.
class Animal {
public:
    virtual void myType() = 0;
};

class Dog : public Animal {
public:
    void myType() {
        std::cout << "I'm a dog!" << std::endl;
    }
};

class Cat : public Animal {
public:
    void myType() {
        std::cout << "I'm a cat!" << std::endl;
    }
};

namespace {
    std::unordered_map<AnimalType, std::function<std::unique_ptr<Animal>()>> const
animalCreators{
        { AnimalType::DogAnimal, []{ return std::make_unique<Dog>(); } },
        { AnimalType::CatAnimal, []{ return std::make_unique<Cat>(); } },
    };
}

std::unique_ptr<Animal> getAnimalFromEnum(AnimalType type) {
    return animalCreators.at(type)();
}

int main(int argc, char const *argv[]) {
    std::unique_ptr<Animal> myPet = getAnimalFromEnum(AnimalType::DogAnimal);
    myPet->myType();
    std::unique_ptr<Animal> mySecondPet = getAnimalFromEnum(AnimalType::CatAnimal);
    mySecondPet->myType();
    return 0;
}

```

Gorzoid - May 11, 2018, 4:37 pm

(May 11, 2018 02:46 PM)Annabelle Lee Wrote: [\[CITE\]](#) Using an unordered map, you can do something like this...

```

Code #include <iostream>
#include <functional>
#include <unordered_map>

```

```
#include <memory>
```

```
enum class AnimalType {  
    DogAnimal,  
    CatAnimal  
};
```

```
// Abstract class. Let's make it require a function that prints type of animal.
```

```
class Animal {  
public:  
    virtual void myType() = 0;  
};
```

```
class Dog : public Animal {  
public:  
    void myType() {  
        std::cout << "I'm a dog!" << std::endl;  
    }  
};
```

```
class Cat : public Animal {  
public:  
    void myType() {  
        std::cout << "I'm a cat!" << std::endl;  
    }  
};
```

```
namespace {  
    std::unordered_map<AnimalType, std::function<std::unique_ptr<Animal>()>> const  
animalCreators{  
        { AnimalType::DogAnimal, []{ return std::make_unique<Dog>(); } },  
        { AnimalType::CatAnimal, []{ return std::make_unique<Cat>(); } },  
    };  
}
```

```
std::unique_ptr<Animal> getAnimalFromEnum(AnimalType type) {  
    return animalCreators.at(type());  
}
```

```
int main(int argc, char const *argv[]) {  
    std::unique_ptr<Animal> myPet = getAnimalFromEnum(AnimalType::DogAnimal);  
    myPet->myType();  
    std::unique_ptr<Animal> mySecondPet = getAnimalFromEnum(AnimalType::CatAnimal);  
    mySecondPet->myType();  
    return 0;  
}
```

+1 unique_ptr, +1 for at(), -1 no virtual destructor doesn't affect your code, but probably would in OP's case

OP, it's pretty much either this or a switch statement, you just gotta pick your poison since there is no runtime reflection of types in C++

Annabelle Lee - May 11, 2018, 6:02 pm

(May 11, 2018 04:37 PM)Gorzoid Wrote: [\[CITE\]](#) +1 unique_ptr, +1 for at(), -1 no virtual destructor

doesn't affect your code, but probably would in OP's case

OP, it's pretty much either this or a switch statement, you just gotta pick your poison since there is no runtime reflection of types in C++

Virtual destructor was left out on purpose, since I simply wanted to illustrate the point. Also recently discovered `unique_ptr/make_unique` and smart pointers, so I've been having a field day playing with them.

6577657534735 - *May 11, 2018, 10:11 pm*

switch case is better than a map because with all warnings enabled the compiler will warn you if you add a new enum but forget to define a new creation method