

# Forum:[Hack Forums](#)

Thread: [D-Pointer Pattern](#)

**Remix** - November 30, 2015, 4:59 am

## Introduction

I'm creating this thread because I want to share this useful technique.  
The D-Pointer Pattern can be used to improve your classes and their design.

## Advantages

This pattern design will make classes better readable.  
Imagine you have a class with many lines and you want to find the member variables.  
This design will let you find them much easier.  
It will improve the translation time of the compiler as well.

## How to use it

Let's take this simple class as an example:

Code class Person

```
{
    std::string m_name;
    int m_age;

public:
    Person();
    std::string name() const;
    void setName(const std::string &name);
    int age() const;
    void setAge(int age);
};
```

As you can see everything is just declared in the basic class.

Imagine you would have 100+ attributes and getter/setter methods for every attribute.

Somebody who wants to use the class doesn't really need to know which attributes exist etc and he probably would waste time with searching them.

If you'd use the D-Pointer Pattern he would find the attributes easily.

The first step to use that pattern is the forward declaring of a new class:

Code class PersonPrivate;

class Person

```
{
    std::string m_name;
```

```
int m_age;
```

```
public:
```

```
    Person();  
    std::string name() const;  
    void setName(const std::string &name);  
    int age() const;  
    void setAge(int age);
```

```
};
```

The forward declaring is just used to make the typename available for other declarations.

Mostly the new class has the name of the actual class just with a "Private" at the end. Some people declare a nested class for it but there's no need at all.

The initialization file would look like that:

Code #include "person.h"

```
Person::Person()
```

```
{
```

```
}
```

```
std::string Person::name() const
```

```
{
```

```
    return m_name;
```

```
}
```

```
void Person::setName(const std::string &name)
```

```
{
```

```
    m_name = name;
```

```
}
```

```
int Person::age() const
```

```
{
```

```
    return m_age;
```

```
}
```

```
void Person::setAge(int age)
```

```
{
```

```
    m_age = age;
```

```
}
```

In that file you have to declare the class.

With the basic declaration the file content looks like that:

Code #include "person.h"

```
class PersonPrivate {
```

```
};
```

```
Person::Person()
```

```
{
```

```

}

std::string Person::name() const
{
    return m_name;
}

void Person::setName(const std::string &name)
{
    m_name = name;
}

int Person::age() const
{
    return m_age;
}

void Person::setAge(int age)
{
    m_age = age;
}

```

The next step would be declaring all the private attributes and methods of the Person class. In this case there are only attributes so it will look like that:

```

Code class PersonPrivate {
public:
    std::string name;
    int age;
};

```

Note that they have to be public in the new class because you will need to call them. And actually you can delete the m\_ from the names but that's up to you. Once they are in the new class you can delete them from the actual class. But you have to declare a pointer of the new class.

With the pointer of the new class it looks like this:

```

Code class PersonPrivate;

class Person
{
    PersonPrivate* d;

public:
    Person();
    std::string name() const;
    void setName(const std::string &name);
    int age() const;
    void setAge(int age);
};

```

Now you have to initialize the pointer in the constructor header:

```

Code Person::Person() :

```

```

    d(new PersonPrivate)
{
}

```

After that you should edit the getter/setter methods.

They are still calling the old member variables:

They need to call the variables with the pointer of the new class:

```

Code std::string Person::name() const
{
    return d->name;
}

```

```

void Person::setName(const std::string &name)
{
    d->name = name;
}

```

```

int Person::age() const
{
    return d->age;
}

```

```

void Person::setAge(int age)
{
    d->age = age;
}

```

At the end you should use a constructor which deletes the pointer of the new class to avoid memory leaks:

```

Code Person::~~Person()
{
    delete d;
}

```

After you did all the steps the classes should look like that:

.h

Code class PersonPrivate;

```

class Person
{
    PersonPrivate* d;

public:
    Person();
    ~Person();
    std::string name() const;
    void setName(const std::string &name);
    int age() const;
    void setAge(int age);
};

```

.cpp

Code #include "person.h"

```

class PersonPrivate {
public:
    std::string name;
    int age;
};

Person::Person() :
    d(new PersonPrivate)
{

}

Person::~~Person()
{
    delete d;
}

std::string Person::name() const
{
    return d->name;
}

void Person::setName(const std::string &name)
{
    d->name = name;
}

int Person::age() const
{
    return d->age;
}

void Person::setAge(int age)
{
    d->age = age;
}

```

Closing words

I hope you found that guide useful.  
Any criticism and suggestion is appreciated.

Greetings  
Remix

**Nymphetamin** - November 30, 2015, 5:25 am

Right. So you've never heard of "protected"? Or inheritance?

**Remix** - November 30, 2015, 6:18 am

(Nov 30, 2015 06:25 AM)Nymphetamin Wrote: [\[CITE\]](#) Right. So you've never heard of "protected"?

Or inheritance?

I did but it's not the point of it.

**0xDEAD10CC** - *November 30, 2015, 8:34 pm*

In terms of the pimpl idiom, d-pointers are a design pattern built around consistency, why not use a macro to identify the private d pointer like in the way Qt does it with `Q_DECLARE_PRIVATE`? -- One place for pre-compile-time modification and guaranteed consistency with naming. d-pointers aren't the only implementation of the idea of pimpl though albeit an early one. Lastly, in terms of advantages, you didn't once mention anything about binary compatibility.