



CPP Module 03

Inheritance, bir sınıfın (base class) özelliklerinin ve metodlarının başka bir sınıf (derived class) tarafından devralınmasını sağlayan bir mekanizmadır. Bu mekanizma sayesinde, code reuse (kod tekrar kullanımı) ve polymorphism (çok biçimlilik) gibi avantajlar sağlanabilir.

Örnek olarak, bir base class olan `Animal` sınıfındaki özellikler (örneğin, "name" ve "age" özellikleri) ve metodlar (örneğin, "makeSound()" metodu) başka bir sınıf olan `Dog` sınıfı tarafından devralınabilir. `Dog` sınıfı, `Animal` sınıfındaki özellikleri ve metodları kullanabileceği gibi kendine özgü özellikler ve metodlar da ekleyebilir (örneğin, "breed" özelliği).

```
c Copy code  
  
class Animal {  
    public:  
        string name;  
        int age;  
        void makeSound() { cout << "Animal sound" << endl; }  
};  
  
class Dog : public Animal {  
    public:  
        string breed;  
        void makeSound() { cout << "Bark!" << endl; }  
};
```

Bu şekilde, `Animal` sınıfındaki özellikler ve metodlar `Dog` sınıfı tarafından devralınır ve kullanılabilir. Aynı zamanda, `Dog` sınıfı kendine özgü `breed`

özellği ve `makeSound()`

metodu eklemiştir. Bu mekanizma sayesinde, benzer özelliklere ve metodlara sahip sınıflar için kod tekrar kullanımı yapılabilir ve polymorphism (çok biçimlilik) kavramı kullanılarak farklı sınıflardaki metodlar aynı şekilde çağrılabilir.

Bir classtan başka classın private değişkenine asla erişemem ama protected ve public değişkenlerine erişebilirim.

private değişkenler sadece belirli sınıf tarafından erişilebilir ve diğer sınıflardan gizli tutulurlar. Eğer başka bir sınıf tarafından erişmek istiyorsanız, o sınıf tarafından tanımlanmış bir public fonksiyon kullanabilirsiniz. Bu fonksiyon, private değişkenlere erişim sağlar ve bu değişkenlerin değerlerini almak veya değiştirmek için kullanılabilir.
→ Getter ve setter

Polimorfizm

```
#include <iostream>
```

```
class Shape {  
public:  
virtual void Draw() {  
std::cout << "Drawing Shape\n";  
}  
};  
  
class Circle : public Shape {  
public:  
void Draw() {  
std::cout << "Drawing Circle\n";  
}  
};  
  
class Square : public Shape {  
public:  
void Draw() {  
std::cout << "Drawing Square\n";  
}  
};
```

```

int main() {
    Shape *shape1 = new Circle();
    Shape *shape2 = new Square();

    shape1->Draw();
    shape2->Draw();

    return 0;
}

```

Çıktı:

```

Drawing Circle
Drawing Square

```

Bu örnekte, **Shape** sınıfından türeyen **Circle** ve **Square** sınıfları aynı isimli **Draw()** fonksiyonunu tanımlar ve farklı işlevsellik sağlar. Polimorfizm, fonksiyonların farklı nesne tipleri için farklı şekilde çalışmasını mümkün kılar.

Inheritance →

```

#include <iostream>

class Vehicle {
public:
    void StartEngine() {
        std::cout << "Starting engine of Vehicle\n";
    }
};

class Car : public Vehicle {
public:
    void StartEngine() {
        std::cout << "Starting engine of Car\n";
    }
};

```

```
}  
};  
  
int main() {  
    Car car;  
    car.StartEngine();  
  
    return 0;  
}
```

Çıktı:

```
Starting engine of Car
```

Bu örnekte, `Vehicle` sınıfından türeyen `Car` sınıfı, `Vehicle` sınıfının `StartEngine()` fonksiyonunu kalıtır ve kendine özgü bir versiyonu tanımlar. Bu, tekrar kodun azaltılmasını ve sınıflar arasındaki hiyerarşiyi gösterir.

IMPORTANT

C++'da bir sınıftan kalıtım yapılırken, **kalıtılan sınıfın bütün özellikleri ve fonksiyonları türetilen sınıfta da bulunmalıdır**. Ayrıca, türetilen sınıf, kalıtılan sınıfın fonksiyonlarını üzerine yazabilir veya kalıtılan fonksiyonları kullanarak kendine özgü fonksiyonlar oluşturabilir.

Bu şekilde, türetilen sınıf, kalıtılan sınıfın özelliklerini ve fonksiyonlarını kullanabilecek ve aynı zamanda kendine özgü özellikler ve fonksiyonlar ekleyebilecektir. Bu sayede, kod tekrarı azaltılabilir ve sınıflar arasındaki hiyerarşi ve işlevsellik düzenlenmiş olur.

`void engageFourWheelDrive() const` bir fonksiyon tanıımıdır ve aşağıdaki öğeleri içermektedir:

1. `void`: Fonksiyonun geri dönüş değeri. Bu durumda fonksiyon geri dönüş değeri olmadan sonlandırılacağı anlatılmıştır.
2. `engageFourWheelDrive`: Fonksiyonun adı.

3. `()`: Parametre listesi. Bu fonksiyon parametre almaz.
4. `const`: `const` anahtar kelimesi, fonksiyonun `this` pointer'ının constant olmasını belirtir. Yani, fonksiyon içinde `this` pointer'ına atama yapılamaz ve fonksiyon içinde nesnenin özelliği değiştirilemez.

VIRTUAL


`virtual` anahtar kelimesi, C++ dilinde polimorfizm kullanımını sağlar. `virtual` anahtar kelimesi ile tanımlanmış bir fonksiyon, derived sınıflarda override edilebilir ve run-time polymorphism kullanımını sağlar.

Aşağıdaki örnekte, `Vehicle` sınıfındaki `startEngine` fonksiyonu `virtual` olarak tanımlanmış

```
c Copy code  
  
class Vehicle {  
public:  
    Vehicle() {}  
    virtual ~Vehicle() {}  
    virtual void startEngine() const {  
        std::cout << "Starting engine of a vehicle." << std::endl;  
    }  
};
```

Bu sayede, `Vehicle` sınıfından kalıtılan bir derived sınıf, `startEngine` fonksiyonunu override ederek farklı bir işlevsellik sunabilir:

c

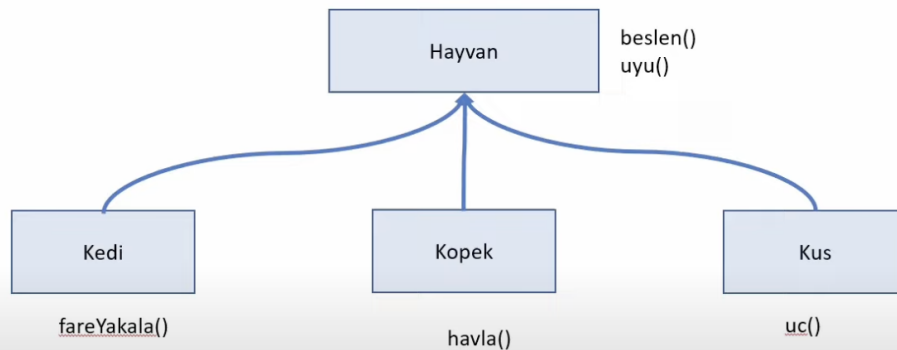
 Copy code

```
class Car : public Vehicle {  
public:  
    Car() {}  
    virtual ~Car() {}  
    void startEngine() const override {  
        std::cout << "Starting engine of a car." << std::endl;  
    }  
};
```


virtual anahtar kelimesi kullanılmazsa, polimorfizm kullanılamaz ve derived sınıflarda fonksiyonların override edilmesi mümkün olmaz.

Inheritance'da kedi kopek ve kusun hayvan classından kalıtım aldığı için hayvandaki fonksiyonları otomatik kullanabilirler. Bir daha tek tek yazmama gerek yok.

Inheritance




csharp

 Copy code

```
class Person {  
private:  
    string name;  
  
public:  
    void setName(string n) {  
        name = n;  
    }  
    string getName() {  
        return name;  
    }  
};
```

Bu sınıfın bir nesnesi oluşturulduğunda, private alan "name" sadece setName() ve getName() metodları aracılığıyla erişilebilir:

c

 Copy code

```
int main() {  
    Person p;  
    p.setName("John Doe");  
    cout << p.getName() << endl;  
    return 0;  
}
```

Programın çıktısı: "John Doe" olacaktır.

Wshadow flag, C ve C++ programlama dillerinde, yapılan kodun derlenmesi sırasında açıklama içeren hataları tespit etmeyi amaçlar. "Wshadow" kelimesi "warning shadow" kelimelerinin birleşiminden oluşmuştur ve programcıya, aynı isimde iki farklı değişkenin kullanılması durumunda oluşabilecek olası problemleri uyarmayı amaçlar.

Nihai olarak, çoklu miras ve virtual fonksiyonlar, çoklu soyutlama ve polimorfizmi sağlar.

inherit constructor



<https://stackoverflow.com/questions/347358/inherit-ing-constructors>

ScavTrap nesnesi oluşturulduğunda, ClapTrap nesnesinin üzerine inşa edilir, yani ClapTrap nesnesi ScavTrap nesnesinin bir parçasıdır. ScavTrap nesnesi yok edildiğinde, ters sırada gerçekleşir çünkü program öncelikle ScavTrap nesnesini yok etmeli ve daha sonra ona dayalı ClapTrap nesnesini yok etmelidir. Bu, bellekte doğru sırada serbest bırakılmasını ve potansiyel bellek sızıntılarını önlemeyi garantiler.


```
12
13 #include "ScavTrap.hpp"
14
15 ScavTrap::ScavTrap()
16 {
17     std::cout<<"CHILD Default Constructor called!"<<std::endl;
18 }
19
20 ScavTrap::ScavTrap(std::string sName) : ClapTrap(sName)
21 {
22     this->setName(sName);
23     this->setHit(100);
24     this->setEnergy(50);
25     this->setAttack(20);
26     std::cout<<"CHILD Constructor of " <<this->getName() <<" " <<std::endl;
27 }
28
29 ScavTrap::~ScavTrap()
30 {
31     std::cout<<"CHILD Destructor of " <<this->getName() <<" " <<std::endl;
32 }
33
34 ScavTrap::ScavTrap(const ScavTrap& stcopy) : ClapTrap()
35 {
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL bash - ex01

```
mtemel@DESKTOP-H8879H0:~/42cursus/cpp_folders/cpp/CPP_Module_03/ex01$ g++ 03.cpp & make exe
COMPILED SUCCESSFULLY
START EXECUTING
PARENT Constructor of AALLLLLLLLLLYYYYY called!
CHILD Constructor of AALLLLLLLLLLYYYYY called!
ScavTrap AALLLLLLLLLLYYYYY attacks ENNNNNEEEEEEMMMMMYYYYY, causing 20 points of damage!
ClapTrap AALLLLLLLLLLYYYYY takes damage, causing 200 points of damage!
Now has 0 hit point!
ClapTrap AALLLLLLLLLLYYYYY is dead!
CHILD Destructor of AALLLLLLLLLLYYYYY called!
PARENT Destructor of AALLLLLLLLLLYYYYY called!
DONE EXECUTING
mtemel@DESKTOP-H8879H0:~/42cursus/cpp_folders/cpp/CPP_Module_03/ex01$
```

```

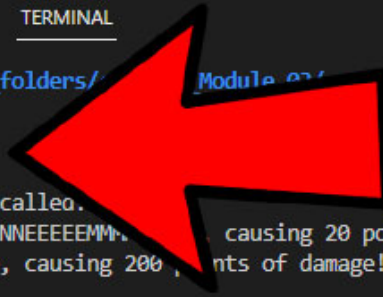
13 #include "ScavTrap.hpp"
14
15 ScavTrap::ScavTrap()
16 {
17     std::cout<<"CHILD Default Constructor called!"<<std::endl;
18 }
19
20 ScavTrap::ScavTrap(std::string sName) : ClapTrap(sName)
21 {
22     this->setName(sName);
23     this->setHit(100);
24     this->setEnergy(50);
25     this->setAttack(20);
26     std::cout<<"CHILD Constructor of "<<this->getName()<<" called!"<<std::endl;
27 }
28
29 ScavTrap::~ScavTrap()
30 {
31     std::cout<<"CHILD Destructor of "<<this->getName()<<" called!"<<std::endl;
32 }
33
34 ScavTrap::ScavTrap(const ScavTrap& stcopy) : ClapTrap()
35 {

```

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL

bash

● mtemel@DESKTOP-H8879H0:~/42cursus/cpp_folders/... Module 03/...
 COMPILED SUCCESSFULLY
 START EXECUTING
 PARENT DEFAULT CONSTRUCTOR CALLED
 CHILD Constructor of AALLLLLLLLLLYYYYY called.
 ScavTrap AALLLLLLLLLLYYYYY attacks ENNNNNEEEEEMM... causing 20 points of damage!
 ClapTrap AALLLLLLLLLLYYYYY takes damage, causing 200 points of damage!
 Now has 0 hit point!
 ClapTrap AALLLLLLLLLLYYYYY is dead!
 CHILD Destructor of AALLLLLLLLLLYYYYY called!
 PARENT Destructor of AALLLLLLLLLLYYYYY called!
 DONE EXECUTING
 ○ mtemel@DESKTOP-H8879H0:~/42cursus/cpp_folders/cpp/CPP_Module_03/ex01\$



DEFAULT
CONSTR