Sub-typing, Polymorphism

Daha önce Inheritance konusunu görmüştük. Bir classın bir classtan miras almasının, onun o classa ait olacağını söylemiştik. (Örneğin, Cat classı "Animal" dan miras aldığı için, bir "Cat" aynı zamanda bir "Animal" dır. Hayvanlardan çok sıkıldığımız için farklı bir örneğe bakalım.

```
class Character
{
  public:
    void sayHello(std::string const &target);
};

class Warrior : public Character
{
  public:
    void sayHello(std::string const &target);
};

class Cat
{
  // [...]
};
```

Parent classtaki bir fonksiyonu override etmek için onu child classta yeniden tanımlayabileceğimizi biliyoruz.

```
void Character::sayHello(std::string const &target)
{
  std::cout << "Hello " << target << " !" << std::endl;
}

void Warrior::sayHello(std::string const &target)
{
  std::cout << "F*** off " << target << ", I don't like you !" << std::endl;
}
```

GIPTA

```
int main()
{
    // Bunun çalışacağına şüphe yok sanırsam:
    Warrior* a = new Warrior();
    // Bir "Warrior" aynı zamanda bir "Character" olduğundan bu da çalışır:
    Character* b = new Warrior();
    // Tam tersi çalışmaz, çünkü her "Warrior" bir "Character" olmasına rağmen
    // her "Character" bir "Warrior" değildir. Compiler hatası alırsınız:
    // Warrior* c = new Character();
    // Bu da çalışmaz tabii ki çünkü Cat classının Character classıyla
    // hiçbir alakası yok zaten:
    Character* d = new Cat();
    a->sayHello("students");   → Burda ne olacağını çok kolay bir şekilde tahmin edebiliyorsun. Warrior::sayHello() çalışacak.

    b->sayHello("students");   → Peki ya burda ne olacak? b'nin data tipi "Character" ancak "Warrior" olarak initialize ettik.
}
```

Output:

```
F*** of students, I don't like you!
Hello students!
```

b için Character::sayHello nun çalıştığını görüyoruz. Peki b bir Warrior olmasına rağmen, neden böyle oldu?

b bir Warrior olsa bile, bu compilerın umrunda değil. Değişkenin tipini "Character" yaptığımız için derleyici bunu görüyor ve b'nin Warrior gibi davranmamasına sebep oluyor.

Ama ben böyle olmasını istemiyorum. b değişken tipi "Character" olmasına rağmen bir "Warrior" ve bu şekilde davranmasını istiyorum. Neyse ki C++ da bunu yapmanın bir yolu var.

```
class Character
{
  public:

    virtual void sayHello(std::string const &target);
};
```

İstediğimiz sonucu elde etmek için "virtual" keywordünü kullanıyoruz. Bunun neden işe yaradığını öğrenmeden önce, birkaç kavramı bilmeliyiz.

! "Compile time" ve "Runtime" kavramlarını araştırın.

İlk başta, virtual kullanmadığımda sayHello fonksiyonum statik bağlantılıydı. Bu da demektir ki, programımın hangi sayHelloyu çalıştıracağına derleme sırasında karar verir ve bu karar değişmez.

Derleyicimin objemin nasıl davranış göstereceğini bilmesinin tek yolu değişken tipidir. Statik bağlantılı bir fonksiyon çağırdığımda, asıl class ne olursa olsun değişken tipine göre karar verir. (Önceki örnekte, b bir Warrior olmasına rağmen, değişken bir "Character" pointeri olduğundan Character::sayHello çağırıldı.

Virtual keywordü, fonksiyonumun dinamik bağlantılı olmasını sağlar. Bu şu anlama geliyor: Hangi fonksiyonu çağıracağıma derleme zamanında değil çalışma zamanında karar verilir.

Bu sefer program b'nin asıl olarak hangi classa ait olduğuna bakar ve b→sayHello(...) davranışını buna göre belirler.

Ayrıca, fonksiyonumun dinamik bağlantılı olması, b→sayHello(...) davranışımın runtime zamanında değişebileceği anlamına gelir. Bu da oldukça kullanışlı bir şey.

Olur ki: b character pointerıma, daha sonra farklı bir şeye işaret edecek şekilde atama yaparsam (bir Character instancei olabilir, Characterden miras alan farklı bir şey olabilir) b→sayHello(...)nun davranışı program çalışırken değişebilir.

Character::sayHello ve Warrior::sayHello yu virtual yaptıktan sonra Output:

```
F*** off students, I don't like you!
F*** off students, I don't like you!
```

Abstract Classes

Pure virtual fonksiyon: Class içinde, 0'a eşitlenmiş ve tanımı olmayan fonksiyonlara denir. Bu classın her child classı, o fonksiyon için kendi implementasyonuna sahip olmalıdır. En az bir pure metoda sahip olan classlara Abstract Class denir. Abstract classların kendine ait class örneği olamaz. Yalnızca miras alınmak için kullanılırlar.

```
class ACharacter                    → kod okunabilirliği açısından abstract class
{                                     isimlerinin başında A olması iyi bir fikirdir.
  private:

    std::string _name;

  public:

    virtual void attack(std::string const & target) = 0;

    void sayHello(std::string const & target);

};
```

↓ pure virtual fonksiyon. bu fonksiyonun tanımı (definition) yoktur.

```
class Warrior : public ACharacter
{
  public:

    virtual void attack(std::string const & target);

};
```

↓ attack(), miras aldığım classta pure metod olduğu için, kendime ait Warrior::attack fonksiyonum tanımlı olmak zorundadır.

```
void ACharacter::sayHello(std::string const &target)
{
  std::cout << "Hello " << target << " !" << std::endl;
}


void Warrior::attack(std::string const & target)
{
  std::cout << "*attacks " << target << " with a sword" << std::endl;
}
```

Interfaces

Tamamen pure metodlardan oluşan classlara Interface denir. Interfaceler attributelara sahip olamazlar. Interfaceler, kendilerinden miras alan classlar için bir çeşit şablon görevi görmeleri için kullanılırlar.

```
class   ICoffeeMaker   → yine kod okunabilirliği açısından, interface
{                        isimlerini I ile başlatmayı tercih ediyorum.
  public:

    virtual void fillWaterTank(IWaterSource *src) = 0;
    virtual ICoffee* makeCoffee(std::string const & type) = 0;
};
```

Interfaceler, kendisinden miras alan classları içerisindeki davranışlara sahip olmaya zorlar. En azından fillWaterTank ve makeCoffee davranışlarına sahip olduktan sonra child classla istediğinizi yapabilirsiniz. Bu durumun, özellikle başkalarıyla kod yazarken çok kullanışlı olduğunu fark edeceksiniz.

```
int   main()
{
  ACharacter *a = new Warrior();   → ACharacter tipinde bir pointer, bir Warrior'a işaret ediyor. Sorun yok.

  // Ancak aşağıdaki gibi bir şey yapmaya çalışırsanız compile olmaz.
  // Çünkü ACharacter bir abstract class ve abstract classların
  // kendine ait class örneği olamaz.

  // ACharacter *b = new ACharacter();
                     ^^^^^^^^^^^^^^^^
                     illegal

  a->sayHello("students");
  a->attack("roger");

}
```