Ödev-03

1. Ödev-03

1.1. Factory Pattern

Aşağıdaki örnek bir oyun içerisindeki karakterlerin oluşturulması ile ilgilidir.

Karakterler 3 kategori olacak şekilde tanımlanmıştır (Savaşçı, Düşman ve Hayvan).

Her karakter türünün verdiği hasar farklıdır.

Bu hasarların farklılığının sağlanması için Factory Pattern kullanılmıştır.

```
#include <iostream>
#include <string>
#include <map>

class Karakter{                                                    ❶
public:
    Karakter(int can, int guc, const std::string &isim) : can(can),
 guc(guc), isim(isim) {}
    int can;
    int guc;
    std::string isim;
    virtual void saldir() const = 0;                               ❷
};

class Savasci : public Karakter{                                   ❸
public:
    Savasci(int can, int guc, const std::string &isim) : Karakter(can,
 guc, isim) {}

private:
    virtual void saldir() const{                                   ❹
        std::cout <<  this->isim << ": Verilen hasar: " << this->guc*3
 << std::endl;
    }
};
```

```
class Dusman : public Karakter{                                    ❺
public:
    Dusman(int can, int guc, const std::string &isim) : Karakter(can,
 guc, isim) {}

private:
    virtual void saldir() const{                                   ❻
        std::cout << this->isim << ": Verilen hasar: " << this->guc*2 <<
 std::endl;
    }
};

class Hayvan : public Karakter{                                    ❼
public:
    Hayvan(int can, int guc, const std::string &isim) : Karakter(can,
 guc, isim) {}

private:
    virtual void saldir() const{                                   ❽
        std::cout << this->isim << ": Verilen hasar: " << this->guc <<
 std::endl;
    }
};

class KarakterUretici{                                             ❾
public:
    Karakter* karakterUret(std::string tip)                        ❿
    {
        if(tip == "savasci")
        {
            return new Savasci(100,50,"savasci");
        }
        else if(tip == "dusman")
        {
            return new Dusman(100,50,"dusman");
        }
        else if(tip == "hayvan")
        {
            return new Hayvan(100,50,"hayvan");
        }
    }
};
```

```
int main()
{
    KarakterUretici *uretici = new KarakterUretici();                    11
    std::map<Karakter*, int> karakterler;                                12
    int i = 0;

    karakterler.insert(std::make_pair(uretici->karakterUret("savasci"),
 i++));
    karakterler.insert(std::make_pair(uretici->karakterUret("dusman"), i
++));
    karakterler.insert(std::make_pair(uretici->karakterUret("hayvan"), i
++));
    karakterler.insert(std::make_pair(uretici->karakterUret("dusman"), i
++));
    karakterler.insert(std::make_pair(uretici->karakterUret("savasci"),
 i++));
    karakterler.insert(std::make_pair(uretici->karakterUret("hayvan"), i
++));

    std::map<Karakter*, int>::iterator it = karakterler.begin();         13
    while(it != karakterler.end())
    {
        std::cout << "key: " << it->second << " ";
        it->first->saldir();
        it++;
    }

    return 0;
}
```

❶ **Karakter** sınıfının tanımlanması.

❷ Abstract sınıf oluşturulması için gerekli olan **saf sanal fonksiyonun** tanımlanması.

❸ **Savasci** sınıfının tanımlanması.

❹ **Karakter** sınıfında oluşturulan saf sanal fonksiyonun **override** edilmesi.

❺ **Dusman** sınıfının tanımlanması.

❻ **Karakter** sınıfında oluşturulan saf sanal fonksiyonun **override** edilmesi.

❼ **Hayvan** sınıfının tanımlanması.

❽ **Karakter** sınıfında oluşturulan saf sanal fonksiyonun **override** edilmesi.

❾ **KarakterUreticisi** sınıfının tanımlanması.

❿ Karakter ureten fonksiyonun tanımlanması. Bu fonksiyon girilen parametreye göre karakter tipi döndürmektedir.

11 KarakterUreticisi örneğinin (instance) oluşturulması.

12 Uretilecek karakterleri ve ID'lerini barındıracak olan **std::map** konteynerının oluşturulması.

13 **std::map** içindeki elemanları gezmek için kullanılacak **iterator** ın oluşturulması.

UML Diagramı