ZH2 előkészítő - mintafeladatok

1. **Készítsen** egy sablon osztályt, amely szállításra való konténerek (*Container*) reprezentálására képes. Minden konténernek van egy azonosítója (*id*). Az osztályt úgy szeretnénk megírni, hogy bármilyen azonosító típussal működhessen (egész szám, sztring, tetszőleges saját osztály stb.), és az azonosítója lekérdezhető legyen (*getId()*).
Készítsen egy (több konténerből álló) rakományokat reprezentáló osztálysablont (*Cargo)*, amely bármilyen, azonos azonosító-típusú konténereket tud nyilvántartani (például csupa egész számmal azonosított *Container*). Nem tudjuk előre, mi lesz a tárolt konténerek azonosító típusa. A szállítmányhoz tetszőleges számú konténert adhatunk egyesével (*az addContainer* tagfüggvénnyel), kivenni viszont nem kell tudni belőle.
Célunk, hogy a Cargo osztályból lekérdezzük egy adott azonosítójú konténert (visszakapjunk a megfelelő, eltárolt objektumra pointert) a *getContainerWithId* tagfüggvénnyel. Ha nincs megfelelő azonosítójú konténer, nullptr-t adjunk vissza. (Példa: myCargo.getContainerWithId("ABCD123")).
Milyen követelményeket támaszt az azonosító típusával szemben? Mutassa meg a megfelelő kódrészletet.
Egy teljes példán mutassa be konténerek létrehozását egy szabadon választott azonosító típussal. Hozzon létre hozzá egy szállítmányt, helyezze el benne az elemeket, majd kérjen vissza a szállítmánytól egy adott azonosítójú konténer elemet. A feladat során ne használjon STL-t.

```
template<class T>
class Container {
        T id; //nem const, hogy lehessen op= !!!
public:
        Container(T idparam=0) :id(idparam) {} //T masolokonstr.! (vagy const ref)
        T getId() const { return id; } //T masolokonstruktor!
};

 ////////////////////////////////////////////////////////////////////////////
template<class U>
class Cargo {
        Container<U>* containers;
        int count;
public:
        Cargo() : count(0), containers(nullptr) {}
        ~Cargo() {
                delete[] containers;
        }
        void addContainer(Container<U>& param)
        {
                Container<U>* temp = new Container<U>[count + 1];
                for (int i = 0; i < count; i++)
                {
                        temp[i] = containers[i]; //op= a T-re is!
                }
                temp[count] = param;
                count++;
                delete[] containers;
                containers = temp;
        }

        Container<U>* getContainerWithId(const U& pid) {
                for (int i = 0; i < count; i++)
                {
                        if(containers[i].getId() == pid) //op==
                        {
                                return &containers[i];
                        }
                }
                return nullptr;
        }
};
```

```
int main(int argc, char* argv[]) {

	Container<int> c1(1);
	Container<int> c2(2);
	Container<int> c3(3);
	Cargo<int> cargo;
	cargo.addContainer(c1);
	cargo.addContainer(c2);
	cargo.addContainer(c3);

	Container<int>* result = cargo.getContainerWithId(2);
if(result != nullptr)
	cout << result->getId();


	getchar();
	return 0;

}
```

4. Egy erdei kisvasutat üzemeltető cég különböző árkategóriájú jegyeket (*Ticket*) ad el. Jelenleg három típust értékesítenek: normál (*StandardTicket*), diák (*StudentTicket*) és nyugdíjas jegy (*SeniorTicket*). A jegyek ára a konkrét távolságtól függ, amelyet kilométerben megadva állítanak be egy-egy jegyhez. Egy kiadott jegyen utólag az érvényes távolságot nem lehet módosítani. A távolságdíj normál alapértéke (*basePrice*) kilométerenként 30 Ft., amely a programfutás során nem változik. A különböző kategóriájú jegyek esetén az ár az alapérték konstansszorosa, értéke rendre 1.0 (*normalFactor*), 0.5 (*studentFactor*) és 0.3 (*seniorFactor*). A jegyeket reprezentáló osztályoktól a konkrét ár lekérdezhető (*getPrice() fv.* adja vissza az árat). Szoftverünk az adott napon kiadott jegyeket egy kategóriától független közös tömbben tárolja. A célunk ez alapján a tömb alapján összegezni és kiírni a napi bevételt.

- **Tervezze** meg és vázolja fel az osztályok öröklési hierarchiáját! Használja fel a fenti dőlt betűs osztály-, függvény- és változóneveket! Az osztályok téglalapjaiban tüntesse fel a kategóriával és árral kapcsolatos tagváltozók és/vagy tagfüggvények deklarációját és láthatóságát! Ügyeljen az elegáns OO megoldásokra!
- **Implementálja** az osztályokat és konstansokat, figyelve arra, hogy esetlegesen egyes konstansokat is tagként vagy statikus tagként érdemes implementálni. Ne legyen egy függvénytörzsben sem felesleges, nem használt kód! Egy új jegytípus esetleges felvételéhez ne kelljen a már meglévő osztályokat módosítani!

Írjon egy egyszerű programrészletet (nem dinamikus) tömbbel, ami megmutatja a három különböző típusú jegy felvételét, valamint egy ciklusban kiszámolja, majd kiírja a tömbben tárolt jegyek összegét.

```
class Ticket {
protected:
        const unsigned distance;
        static const unsigned basePrice;
public:
        Ticket(unsigned pdist) :distance(pdist) {}
        virtual double getPrice() = 0;

};
const unsigned Ticket::basePrice = 30;
-------------------------------------------------------------------------
class StandardTicket :public Ticket {
        static const double normalFactor;
public:
        StandardTicket(unsigned pdist) :Ticket(pdist) {}
        double getPrice() { return distance*basePrice*normalFactor; }
};
const double StandardTicket::normalFactor = 1.0;
-------------------------------------------------------------------------
class StudentTicket :public Ticket {
        static const double studentFactor;
public:
        StudentTicket(unsigned pdist) :Ticket(pdist) {}
        double getPrice() { return distance*basePrice*studentFactor; }
};
const double StudentTicket::studentFactor = 0.5;
-------------------------------------------------------------------------

class SeniorTicket :public Ticket {
        static const double seniorFactor;
public:
        SeniorTicket(unsigned pdist) :Ticket(pdist) {}
        double getPrice() { return distance*basePrice*seniorFactor; }

};
const double SeniorTicket::seniorFactor = 0.3;
-------------------------------------------------------------------------
```

```
int main(int argc, char* argv[]) {

	Ticket* tickets[3];
	double sum = 0;
	tickets[0] = new StandardTicket(100);
	tickets[1] = new SeniorTicket(200);
	tickets[2] = new StudentTicket(300);
	for (int i = 0; i < 3; i++)
	{
		sum += tickets[i]->getPrice();
		delete tickets[i];
	}
	cout << sum << endl;

	getchar();
	return 0;

}
```