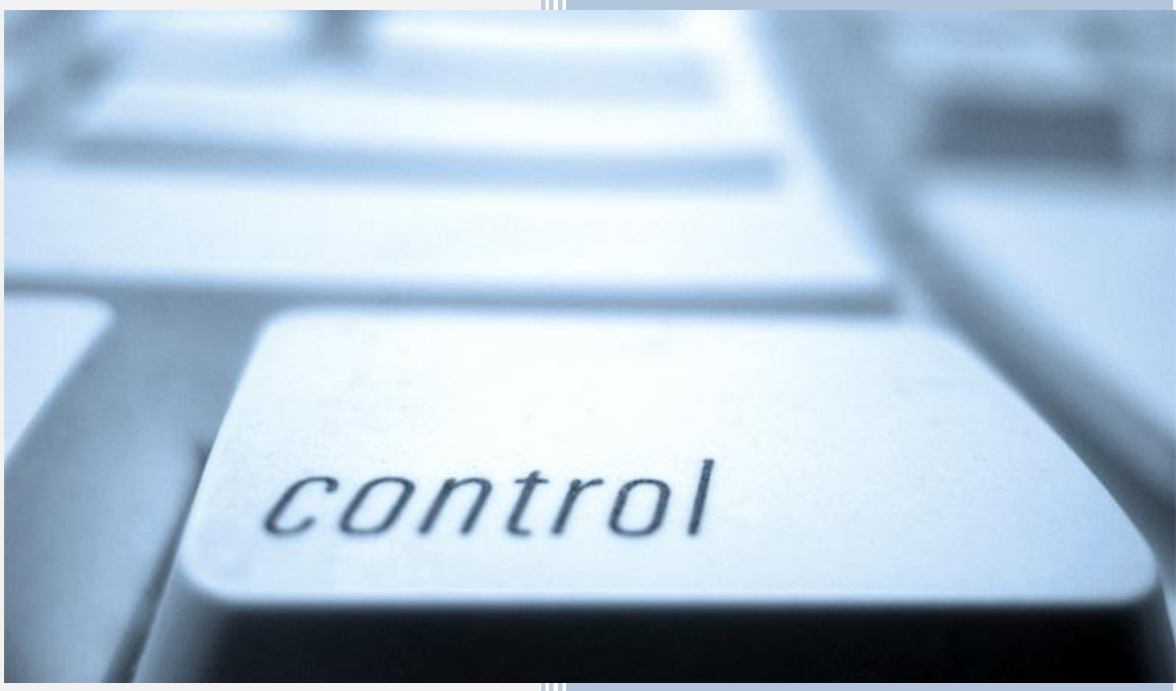
# Futár program dokumentációja




damsalevente@gmail.com

Dámsa Levente

2016.05.13.

# Tartalomjegyzék:

# Probléma rövid ismertetése

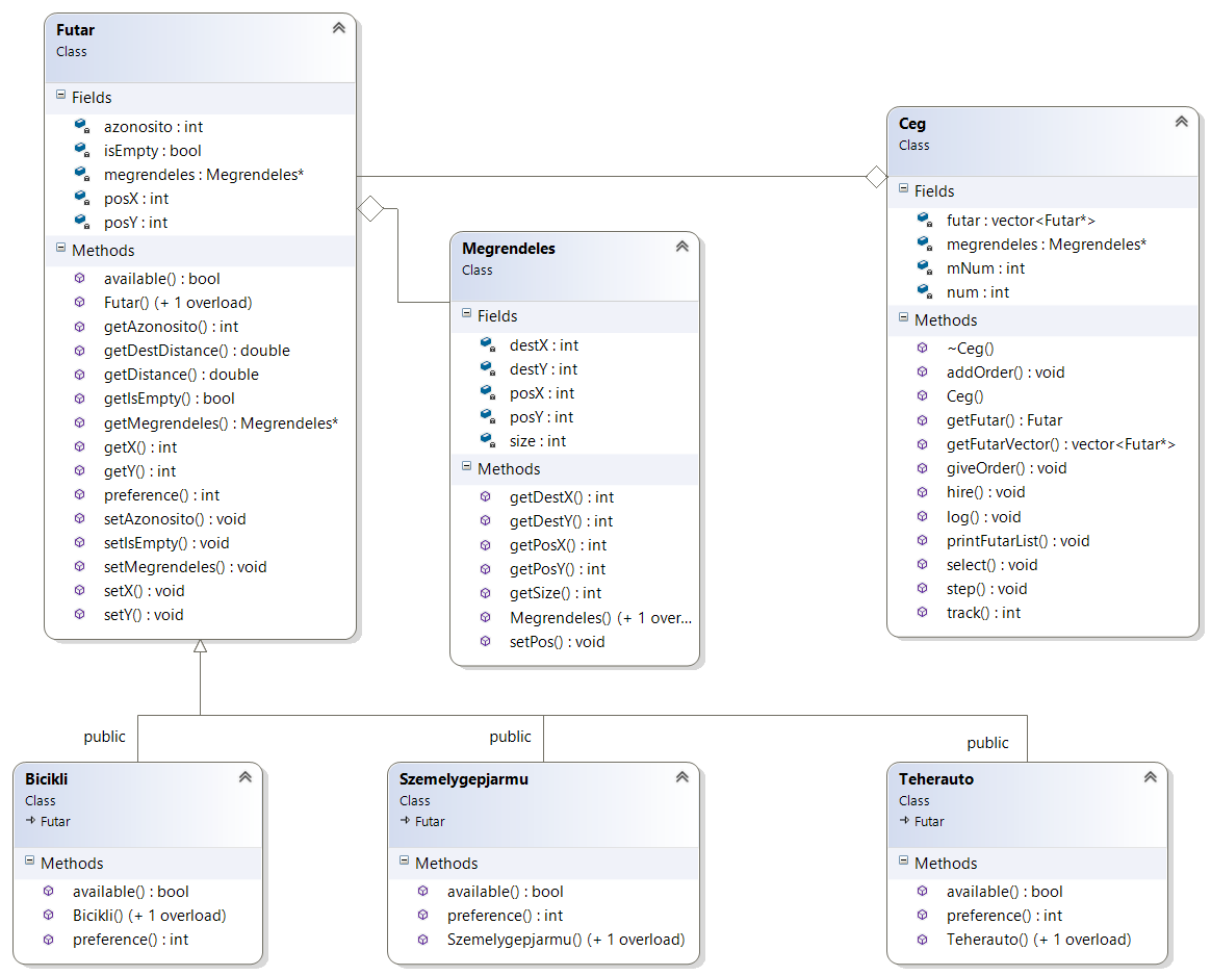
Készítsük el ez futárszolgálat irányítási programját. A futárszolgálathoz több futár csatlakozik, lehetnek biciklis, autós és teherautós futárok is. A futárszolgálathoz beérkező megrendelések alapján az irányító rendszer a legközelebbi futárt küldi a helyszínre, amely képes elvinni az adott csomagot. A biciklis futárok, a kisebb csomagokat 15km-es körzetben maguk kézbesítik, míg távolabbra csak a legközelebbi autós/teherautós futárnak adják át. Az autós futárok csak egy csomagot kézbesítenek, míg a teherautókra több csomagot is fel lehet rakni. A megrendelések alapján a program irányítsa a futárok munkáját, valamint lehessen nyomon követni a megrendeléseket.

# Feladat értelmezése, specifikáció

A futárszolgálat a következőképpen épül föl: biciklis, személygépjármű és teherautó típusú futárok vannak a cégnél, melyeket egy Irányítórendszer oszt be amennyiben üresek, illetve ő a legalkalmasabb arra, hogy elszállítsa a csomagot. A csomagokat nem fogja a biciklis cserélgetni, a három jármű feladata a következő:

- A biciklis a 15km-nél közelebbit, és a nem túl távolra viendő csomagot szereti
- A személygépjármű a viszonylag távoli, viszonylag távoli célokat szereti
- A teherautó a távoli kiszállításokra való, illetve csak ő képes nehezebb csomagokat elvinni

# UML diagram


Futar
Class
Fields
azonosito : int
isEmpty : bool
megrendeles : Megrendeles*
posX : int
posY : int
Methods
available() : bool
Futar() (+ 1 overload)
getAzonosito() : int
getDestDistance() : double
getDistance() : double
getIsEmpty() : bool
getMegrendeles() : Megrendeles*
getX() : int
getY() : int
preference() : int
setAzonosito() : void
setIsEmpty() : void
setMegrendeles() : void
setX() : void
setY() : void
Megrendeles
Class
Fields
destX : int
destY : int
posX : int
posY : int
size : int
Methods
getDestX() : int
getDestY() : int
getPosX() : int
getPosY() : int
getSize() : int
Megrendeles() (+ 1 over...
setPos() : void
Ceg
Class
Fields
futar : vector<Futar*>
megrendeles : Megrendeles*
mNum : int
num : int
Methods
~Ceg()
addOrder() : void
Ceg()
getFutar() : Futar
getFutarVector() : vector<Futar*>
giveOrder() : void
hire() : void
log() : void
printFutarList() : void
select() : void
step() : void
track() : int
public
public
public
Bicikli
Class
Futar
Methods
available() : bool
Bicikli() (+ 1 overload)
preference() : int
Szemelygepjarmu
Class
Futar
Methods
available() : bool
preference() : int
Szemelygepjarmu() (+ 1 overload)
Teherauto
Class
Futar
Methods
available() : bool
preference() : int
Teherauto() (+ 1 overload)

# Funkciók, fontosabb függvények részletezése

## Futár, és alosztályai

Az itt felsorolt függvények csak a legfontosabbak, amelyek a megoldás megértését segítik.

Futár:

| Név | Visszatérési érték | Megjegyzés |
|---|---|---|
| virtual available(const megrendeles &) | bool | minden esetben azt adja meg, hogy elérhető-e a futár az adott csomaghoz |
| getMegrendeles() | Megrendeles* | Visszaadja a megrendelése címét, ha nincs, akkor nullptr |
| getDistance(const Megrendeles &) | double | 2-normán alapuló távolság (euklideszi metrika) |
| virtual preference(const Megrendeles &) | int | megadja mennyire preferált az adott futár |

A futár fontosabb tagfüggvényei

## Megrendelés

A megrendelés osztály egyetlen nem szokványos (értem ez alatt, hogy nem getter, setter ) tagfüggvénye a setPos, amely x és y poziciót a cél x és cél y pontokra állítja be

## Cég

A cég osztály a program agya. Ő osztja ki és tárolja a Megrendeléseket és a Futárokat dinamikusan. A futárokat egy vektorban gyűjti, a megrendeléseket pedig egy dinamikus tömbben.

| Név | Megjegyzés |
|---|---|
| void hire(Futar *a); | beállítja az utolsó helyre az új futárt, default konstruktorként hívja, a felhasználótól elvárja hogy inicializálja |
| void addOrder(const Megrendeles &m) | felvesz egy új megrendelést |
| Futar getFutar(int index) const | A megadott indexű Futár pointer értékét adja vissza |
| void printFutarList() const | kiírja a standart kimenetre a futárokat |
| void log()const | kiírja a futárok listáját |
| void select(Futar &a, Megrendeles *m) | a futár megkapja az (m) megrendelést |
| int track(Megrendeles &m) const | megadja az azonosítót, ha nem viszi senki, akkor 0 |
| void giveOrder(Megrendeles *m) | megkeresi a legjobb futárt a feladathoz<br>használja a setpreference és a select függvényt-->nem constans |
| vector <Futar * > getFutarVector() const | Visszaadja a futar vectort |
| void step() | időben léptetés. minden futár a neki megadott megrendelés destinációjára ugrik |

# Hiányosságok és jövőbeli fejlesztések

Maga a legjobb futár beosztása elég kezdetleges, nem elég hatékony. Emellett maga a felhasználhatósága érdekében a step függvényt ki kell bővíteni, különben nem lehet folyamatosan használni a programot.

A tesztelések során az volt a tapasztalatom, hogy a beépített függvényekkel elég jól meg lehet oldani feladatokat, ha jól használja a felhasználó, viszont macerás. Egy csomó feladat megoldását meg lehetne oldani egy-egy külön megírt tagfüggvényként.

A jövőben szeretnék egy queue típusú futárbeosztás-rendszert készíteni, amely képes lesz váratni a megrendeléseket mindaddig, amíg azt valaki el nem tudja szállítani.

A template-ek megismerése után felmerült bennem az az ötlet, hogy a megrendelés lenne a program „beleégetett" rész. Ezek után a futár egy Template class lenne, amelyet paraméterként kapna a megrendelés. Így a feladat megoldása megfordulna: nem futárokat rendelnék megrendelésekhez, hanem rendeléseket futárokhoz. Talán elegánsabb megoldás lenne.

Javítás: „Magic" constansok helyett bevezettem a Futarhoz két const static int tagváltozót, amelyek a publikus állományban szerepelnek. Az auto és a teher távolságot jelölik, az autó distance 30km, míg a teher dist-jé 60km.

Hozzáadtam másoló konstruktorokat, habár a feladatomban nem volt rá szükségem, de követelmény.

Operator int cégnél kiadja a rendelések és a futárok számát összesen