Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Általános Informatikai Tanszék

Programterezési Minták

Szak: Mérnök informatikus MSc
Kód: GEIAL517M
Félév: Tavasz

hét	Előadás	Gyakorlat
1	Bevezetés, OOP elvek, UML	Aktuális minták implementálása
2	SOLID Elvek	Aktuális minták implementálása
3	Tervezési Minták	Aktuális minták implementálása
4	Tervezési Minták	Aktuális minták implementálása
5	Structural Patterns	Aktuális minták implementálása
6	Wrappers	Aktuális minták implementálása
7	Viselkedési Minták	Aktuális minták implementálása
8	Viselkedési Minták	Aktuális minták implementálása
9	Viselkedési Minták	Aktuális minták implementálása
10	Ellenminták	Aktuális minták implementálása
11	Konzultáció	Féléves feladat bemutatása
12	Konzultáció	Féléves feladat bemutatása

Kötelező Irodalom:

- **Gamma, Erich. Design patterns: elements of reusable object-oriented software. Pearson Education India, 1995**
- **Freeman, Eric, et al. Head first design patterns. " O'Reilly Media, Inc.", 2004**
- **Kent Beck: Implementation Patterns**

Évközi számonkérések: A félév során 1 önálló feladatot kell elkészíteni, aminek a célja, hogy a hallgató bemutassa a félév során szerzett ismereteit.

Aláírás megszerzésének feltételei:

- A gyakorlati órák minimum 66%-án való aktív részvétel.
- A félév során 1 önálló feladat elkészítése, prezentlása és egy lehetőleg angol nyelvű beszámoló beadása az előírt formában. A feladat bemutatására a félév során folyamatosan, legkésőbb a szorgalmi időszak végéig lehetséges, az erre kijelölt időben.

Vizsga formája:

írásbeli és szóbeli
Vizsgára csak azon hallgató jelentkezhet, aki már megszerezte az aláírást.

Írásbeli
A vizsga írásbeli részében a 3 tanult mintát kell ismertetni.

Szóbeli
Az érdemjegy a szóbeli vizsgán kerül meghatározásra. A szóbeli vizsga a félév elméleti anyagából és gyakorlati anyagából áll. A szóbeli rész programozási feladatot is tartalmazhat.

Miskolc 2019. 09. 10 ……..

Sátán Ádám
Tanszéki mérnök

Név:……. **Neptun kód:……..**

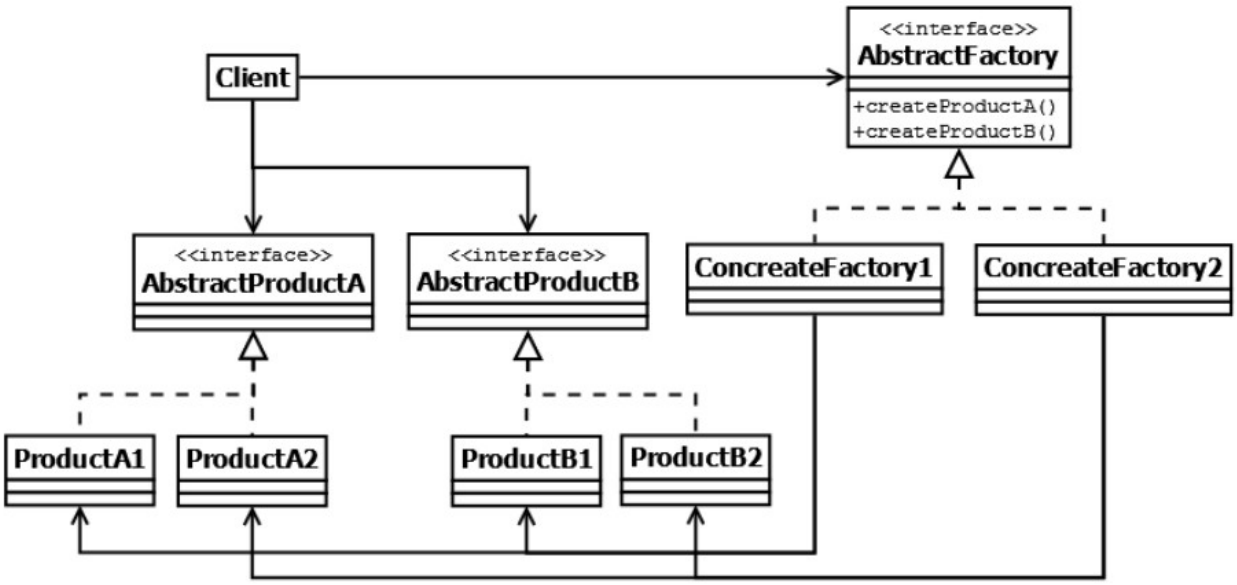
Programterezési Minták vizsga Kód: GEIAL517M

1, Ismertesse az Abstract Factory mintát
2, Ismertesse a Composite mintát
3, Ismertesse az Command viselkedési mintát

Programterezési Minták vizsga megoldás

1, Ismertesse az Abstract Factory mintát

Célja, hogy létrehozási interfészt biztosítson kapcsolódó vagy függő objektumok családjának a konkrét osztályok megadása nélkül, így a termékek különböző csoportokba sorolhatóak. A Factory minta kiterjesztése, ahol a termékek csoportokba rendezhetőek.
Akkor használjuk, ha a létrehozandó objektumokat családba rendezhetjük, azaz függőség van köztük. Általában a létrehozás kevés lépésből áll. A rendszernek függetlennek kell lennie attól, hogyan hozza létre, állítja össze és jeleníti meg a termékeit. A termékcsaládok közül egyet be kell állítania, ezeknek csak az interfészeiket fedhetjük fel, a megvalósításukat nem kell ismernie a kliensnek



Az AbstractFactory interfész az absztrakt termékobjektumokat létrehozó felületet deklarálja. Több is lehet belőle, mindegyiket ismernie kell a kliensnek. A ConcreateFactory osztályok a konkrét termékobjektumok létrehozására szolgáló műveleteket implementálják, több konkrét terméket is ismerhetnek. Az AbstractProduct interfészek a termékobjektumok egy típusának interfészét deklarálják. Több is létezhet belőle, a kliensnek szintén ismernie kell mindet. Végezetül a konkrét termékeket reprezentáló osztályok részei a modellnek, melyek definiálják a létrehozandó termékobjektumot a megfelelő ConcreateFactory mellett, valamint implementálják az AbstractProduct interfészt.
Futásidőben a ConcreateFactory egy példánya jön létre, amely ismeri a konkrét Product osztályokat. Ha a kliens más osztályú objektumot akar létrehozni, akkor másik ConcreateFactory-t kell használnia. A konkrét osztályok elszigeteltek, a kliensnek nem kell ismernie azokat. Ez megkönnyíti a termékcsaládok cseréjét, de megnehezíti az újak bevezetését.

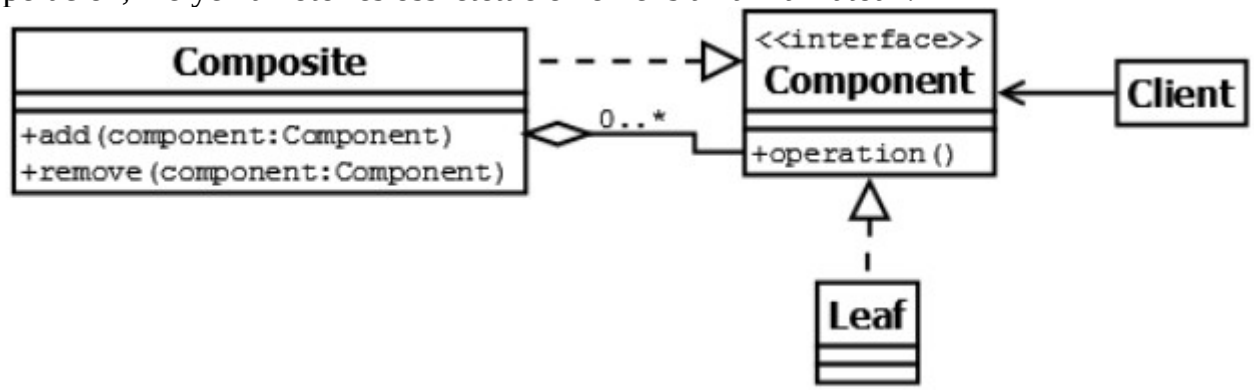
2, Ismertesse a Composite mintát

Célja az objektumok faszerkezetben való ábrázolásának megvalósítása és egyesíteni a kezelését az objektumnak és az összetételnek. A faszerkezet részei:

- levél: egyszerű objektum
- kompozíció: levelek együttese

A rész-egész viszony alkalmazásánál a felépítés lényege, hogy a felhasználó ne érezzen

különbséget komplex és egyszerű objektumok között.
Olyan grafikus alkalmazást készíthetünk, amely lehetővé teszi egyszerű alkotóelemekből bonyolult grafikus objektumok létrehozását. Például szimpla szöveg- és vonalelemeket kategorizálhatunk és azokat egy egységként kezelhetjük. Ilyen a flash szerkesztő azon része is, ahol különböző objektumokat, vonalakat, alakzatokat rakhatunk a rajzfelületre. A programnak viszont meg kell különböztetni az egyszerű alkotó- és az összetett elemeket, mert csak az utóbbihoz lehet elemeket adni illetve elvenni. Ezeken túl vannak olyan operációk, melyek alkotó- és összetett elemekre is alkalmazhatóak.



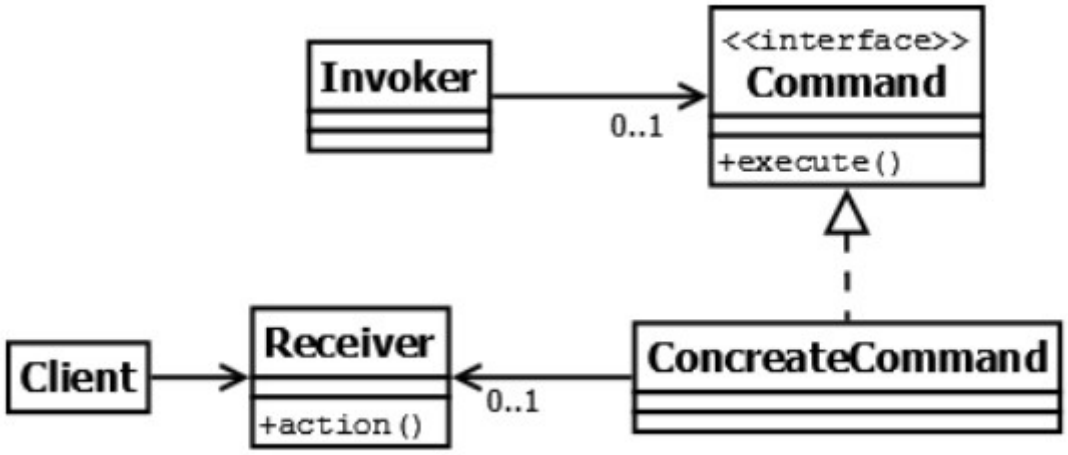
A minta részei az Elem felület (Component), melynek feladata az Összetétel (Composite) objektumok felületének kialakítása, megfelelő alapértelmezett viselkedést biztosít. A Levél osztály (Leaf), amely a fa szerkezet leveleit írja le (levél = nincs gyereke), az összetételt és az összetétel objektumainak viselkedését írja le. Az Összetétel osztály, amely a gyerekekkel rendelkező elemek viselkedését írja le, gyerekeket tárol, végrehajtja az Elem felület műveleteit. Végezetül az Ügyfél (Client) osztály, amely különböző műveleteket hajt végre az Összetétel osztály objektumaival az Elem interfészen keresztül.
A kliens az Elem osztály felületén keresztül kapcsolódik az objektumhoz. Ha a címzett egy Levél osztály, akkor végrehajtódik, vagy amennyiben a címzett egy Összetétel osztály, akkor továbbítódik az összes Levél osztályhoz.
A minta előnye, hogy olyan hierarchiát valósít meg, ami az alap objektumokat rekurzívan építi fel egyre bonyolultabb objektumokká, továbbá egyszerűbbé válik a kliens program, mert nem szükséges megkülönböztetni az alap és az összetett objektumokat, azokat egységesen kezeli szokásos esetekben. Ilyen szerkezettel könnyebb az új komponensek hozzáadása és nem kell a klienst megváltoztatni. Hátránya, hogy az új komponensek könnyebb hozzáadásával viszont túl általánossá válhat az alkalmazás.

3, Ismertesse az Command viselkedési mintát

Célja a kérelmek és azok paramétereinek objektumba ágyazása, ezáltal a klienseknek különböző parancsokat adhatunk át, amit naplózhatunk, sorba rendezhetünk és visszavonást kezelhetünk le, tehát az egyes kérelmek teljesen kivizsgálhatóak és hatálytalaníthatóak lesznek. Mivel objektumba vannak ágyazva a kérelmek, így lehetőség van a kérések ideiglenes tárolására.
Néha muszáj, hogy kéréseket küldjünk egyes objektumokhoz úgy, hogy bármi ismeretünk lenne a kért folyamatról vagy a kérést fogadóról. A felhasználói felületek programozására gyakran használt könyvtárak, eszközkészletek többek között olyan objektumokat, menüket és gombokat tartalmazhatnak, amelyek egy felhasználói esemény hatására indítanak el valamilyen műveletet. Konkrét implementációt a menü, vagy a gomb objektumok nem képesek megvalósítani, mert csakis az eszközkészleteket használó alkalmazás tudja, mi mit csinál ilyenkor. Az eszközkészlet tervezői nem tudják előre, hogy a konkrét tevékenységet végül milyen objektum és hogyan hajtja végre. A mintával ezen kérések az

elemkészlet objektumok számára megvalósíthatóvá válnak oly módon, hogy a kéréseket is objektumként kezelik.
A minta alkalmazhatósága:

- Commit támogatás: a műveletek ismét végrehajtásának támogatása
- Undo támogatás: a műveletek visszavonásának támogatása, unexecute művelet szükséges hozzá
- Naplózás: a változások naplózása rendszerösszeomlás, helyreállítás esetén



A Parancs (Command) felület létrehoz egy interfészt a művelethez. A KonkrétParancs (ConcreateCommand) osztály egymáshoz rendeli a Fogadó (Receiver) osztályt és egy végrehajtandó műveletet. A Fogadó osztály megfelelő műveletének hívásával implementálja a végrehajtó műveletet. Undo megvalósítása esetén a KonkrétParancs osztály állapotát is tárolni kell. Az Ügyfél (Client) osztály létrehoz egy KonkrétParancs osztályt és beállítja a Fogadó osztályt. A Hívó (Invoker) osztály felkéri a Parancs felületet a kérelem teljesítésére úgy, hogy meghívja ennek execute() metódusát. A Fogadó osztály a kérést fogadja, birtokában van az adott kérelemhez kapcsolódó műveletek végrehajtásához szükséges tudásnak, bármelyik osztály lehet fogadó.
Az Ügyfél osztály létrehoz egy KonkrétParancs osztályt és meghatározza annak fogadóját. Valamelyik Hívó osztály elraktározza a KonkrétParancs osztályt. A Hívó osztály kérelmet bocsát ki, a KonkrétParancs osztály pedig műveleteket hív meg a fogadóján.
Előnye, hogy a minta alkalmazása feloldja a kapcsolatot a műveletet kezdeményező és az azt végrehajtó objektum között. A Parancs objektumok is ugyanúgy használhatóak és kibővíthetőek, tipikus objektumok. A parancsok összetett parancsokká rendezhetőek, például makrót képezve. Parancsot osztálymódosítás nélkül vehetünk fel. A minta lehetővé teszi a párhuzamos feldolgozást. Hátránya, hogy sok kis osztályt kell írni és sok kis objektum keletkezik.