

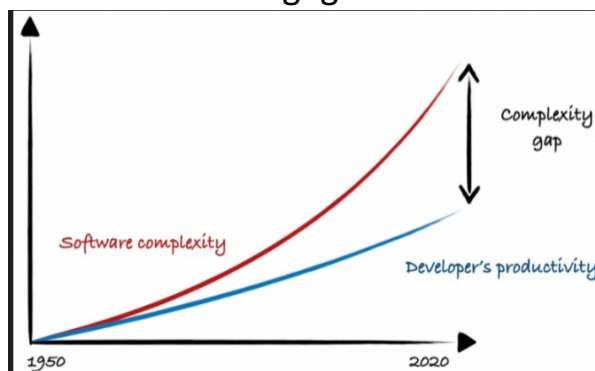
Domande

Saturday, 24 June 2023

14:50

Nota: le risposte potrebbero non essere giuste e le domande potrebbero essere confusionarie.

- 1) Cos'è il low coupling
E' un pattern grasp e si chiama basso accoppiamento
Una classe ha basso accoppiamento se non è troppo dipendente con altre classi
- 2) Cos'è l'high coision
Deve avere una coesione alta, ed una classe ha una coesione alta quando ha responsabilità di un numero non troppo eccessivo ed esse sono correlate tra di loro
- 3) Perché è stato dato il bisogno di un metodo alternativo al metodo cascata
Siccome esso è troppo rigido ai cambiamenti quindi è soggetto ad un alto tasso di fallimenti nel caso di progetti grandi siccome più grande è un progetto e più cambiano i requisiti
- 4) Nel processo iterativo incrementale ed evolutivo abbiamo scoperto che alla 2 iterazione abbiamo scoperto il 90% dei requisiti
No.
- 5) Perché è nata l'ingegneria del software



- 6) L'UML è usata obbligatoriamente nell'architettura logica
Falso
- 7) Differenza SD e SSD
SSD scatola chiusa,
SD scatola aperta
- 8) Differenza aggregazione e composizione
Sono associazioni
Simili tra di loro

Mentre nell'aggregazione sapendo che A è aggregata di B

Può essere indipendente da A

Mentre nella composizione A è molto dipendente da B, è una aggregazione più forte

9) Parlami con scrum

- Product backlog: requisiti del nostro sistema
- Spring backlog: i requisiti fatti in 1 sprint
- Sprint: E' una iterazione di 2-4 settimane
Ogni sprint viene creato un incremento del prodotto -> output che funziona
- Scrum: ogni 24 ore abbiamo dei meeting così tutto il team conosce tutto
- ScrumMaster: responsabile che tutto vada apposto con nessuna interferenza

E che scrum viene applicato correttamente

- Velocity: stima di quanto lavoro rimane in 1 singolo print
- Team sviluppo: devs con non più 7 persone
- Product owner: identificare caratteristiche/requisiti del prodotto e dare priorità

10) Spiegami le pre-condizioni e le post-condizioni (contratti)

Un contratto è un documento che descrive le operazioni di sistema

Pre condizioni = condizioni che devono essere vere affinché contratto deve essere avviato

Post condizioni = condizioni vere dopo

11) Dimmi i 2 formati di casi d'uso (sono 3)

- Breve
- Informale
- Dettagliato

Essi Storie scritte, fatti da un dialogo tra attore e sistema per svolgere un

sistema, scritte con l'intento di comprendere i requisiti ⇒ Ciò a cui servono

Aka un attore che usa il sistema per raggiungere obiettivi

12) Dove useresti il pattern GOF (che è un GRASP) Controller

13) Quale pattern useresti:

Si consideri un gioco per dispositivi mobili. Si vuole avere informazioni sullo stato dell'intero gioco, quali:

- *Numero del livello attuale*
- *Punteggio*
- *Tempo trascorso durante il gioco*
- *Punteggio più alto ottenuto durante la sessione di gioco*

Si vuole inoltre ottenere facilmente l'accesso a questo stato ovunque nel gioco e evitare la necessità di lunghe catene di scambio dati.

14) Spiegami gli eventi

- a. Chiamata
- b. Segnale
- c. Variazione
- d. Temporale

15) Principi dei metodi agili

- a. Sviluppo iterativo
- b. Consegna incrementali
- c. Semplice, leggero, facile comunicazione
- d. Pratiche agili: uml per abbozzi, programmazione a coppie

16) Cos'è il timeboxed

E' il tempo dove l'iterazione ha una durata fissa.

17) Come differisce l'UP dall'UP agile

- Piccolo insieme attività ed elaborati
- Applicare UML con abbozzi
- Pianificazione iterativa ed adattativa
- Requisiti e progettazione non vengono completati prima dell'implementazione

Ma emergono in modo adattativo durante le iterazione sulla base dei feedback

18) Fasi UP ed Iterazioni

- a. Ideazione
- b. Elaborazione
- c. Costruzione
- d. Transizione

E dentro abbiamo diverse fasi chiamate iterazioni:

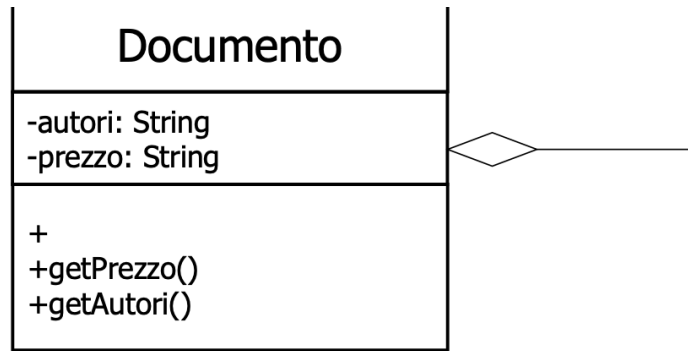
- a. Analisi
- b. Progettazione
- c. Test
- d. Release

19) Questo:

(Composite)

Si consideri un sistema per la stampa di documenti di una copisteria. Si estenda il seguente diagramma delle classi per rappresentare i seguenti requisiti:

- ***Un documento è composto da pagine, eventualmente organizzate in sezioni. Ogni sezione può contenere sezioni (una o più) e pagine semplici.***
- ***E' possibile stampare una pagina singola o una sezione.***



20) A
(Singleton)

Si consideri un gioco per dispositivi mobili. Si vuole avere informazioni sullo stato dell'intero gioco, quali:

- ***Numero del livello attuale***
- ***Punteggio***
- ***Tempo trascorso durante il gioco***
- ***Punteggio più alto ottenuto durante la sessione di gioco***

Si vuole inoltre ottenere facilmente l'accesso a questo stato ovunque nel gioco e evitare la necessità di lunghe catene di scambio dati.

21) Trasformarla in codice

```
public class StatoGioco
{
    private static StatoGioco uniqueInstance;
    private int livello, punteggio, tempoGioco, migliorPunteggio;

    private StatoGioco()
    {
        livello=punteggio=tempoGioco=migliorPunteggio=0;
    }

    public static synchronized StatoGioco getInstance()
    {
        if( uniqueInstance == null )
        {
            uniqueInstance = new StatoGioco();
        }
        return uniqueInstance ;
    }

    // getter and setter
    ...
}
```

22) (Adapter)

Si consideri una classe Documento, che non implementa l'interfaccia copiabile. Si vuole però poter eseguire un copia selettiva di un vettore di oggetti Documento, ma non si vuole assolutamente modificare la classe Documento. Si assuma che la classe Documento ha un metodo valido() che si può invocare per determinare se può essere copiato.

23) Strategy

Si consideri una libreria per la realizzazione di interfacce grafiche. Un oggetto textField di questa libreria rappresenta una casella di testo e può essere utilizzato per validare differenti tipologie di testo come email, cap e date. Ad ogni textField può essere associato un solo tipo di testo da validare.

24) Adapter

Un sistema per la produzione e stampa di ricette mediche deve interfacciarsi con una libreria, prodotta da Lombardia Telematica, per la lettura di dati da uno speciale lettore di smartcard per le tessere sanitarie distribuito dalla regione Lombardia. Volete far sì che il sistema possa dialogare con un lettore di smartcard collegato tanto al computer sul quale è installato il vostro sistema quanto su un computer diverso in rete, anche se la libreria non è fatta per funzionare in rete. Dal momento che la libreria sarà usata in molti punti all'interno del vostro sistema, volete progettare il sistema in modo che il codice che usa un lettore smartcard locale sia identico a quello che ne usa uno in remoto.

25) Factory

Si vuole modellare un polinomio, tenendo conto che il polinomio non è detto che sia sempre completo (ad esempio, potrebbe essere sparso, cioè con solo $ax^n + b$, o non avere tutti i termini). Solo a runtime si sa di che tipo sarà il polinomio. Si vuole progettare una soluzione che permetta di gestire in maniera efficace ed efficiente questa situazione: si identifichi il DP corretto da usare, e si crei il relativo class diagram

26) No Composite

Un tipo di colazione è composto da componenti semplici, eventualmente organizzate in contenitori di componenti. Ogni contenitore può contenere contenitori (una o più) e componenti semplici. Attraverso una stessa interfaccia sarà possibile aggiungere a un tipo di colazione sia contenitori che componenti semplici. A tal fine, identificare il design pattern GoF ritenuto più opportuno e proporre una soluzione descrivendone gli aspetti statici mediante un diagramma delle classi.

27) Differenza tra casi d'uso e requisiti

Caso d'uso è un testo che contiene una storia scritta degli attori che usano il sistema

Ed esso descrive un requisito

Ed il requisito sono delle specifiche che il sistema si deve attenere

28) Dimmi le attività principali incluse in ogni processo software

- a. Requisiti
- b. Analisi
- c. Progettazione
- d. Implementazione
- e. Validazione
- f. Rilascio e Installazione
- g. Manutenzione ed evoluzione
- h. Gestione del progetto

29) Un progetto dopo 1 mese di ideazione, studi economici approfonditi, una creazione di un semplicissimo glossario e la programmazione di un prototipo hanno notato che, alla fine hanno commesso un errore. Quale?

E' durato 1 mese, quindi è stato troppo approfondito.

30) I requisiti non funzionali sono divisi in obiettivi e non verificabili. vero o

- 30) I requisiti non funzionali sono divisi in obiettivi e non verificabili, vero o falso?
- 31) Cos'è un interfaccia
E' un insieme di funzionalità pubbliche identificate da un nome
- 32) Spiegare i vari pattern GRASP:
- 33) Creator
- 34) Information expert
- 35) Low coupling
- 36) Controller
- 37) High cohesion
- 38) Pure fabrication
- 39) Polymorphism
- 40) Protected variations
- 41) Indirection
- 42) Spiegare i vari pattern GOF
- 43) Spiegare:
- 44) Singleton
- a. Si utilizza per le classi che hanno 1 istanza
E devono avere punto di accesso facilmente accessibile da ogni parte del sistema.
 - b. Non bisogna abusarne siccome aumenta il coupling
- 45) Factory (estensiva)
E' un pattern GOF
Bisogna creare oggetti con modalità di creazioni diversi, aka interfaccia con cose differenti
Allora bisogna creare oggetto pure fabrication che serve per mantenere il disaccoppiamento tra le entità che serve per distanziare le classi qui con una sua logica
Un man in the middle praticamente che crea gli oggetti e li fa comunicare
- 46) Adapter
- 47) Composite
- 48) Facade
- 49) Observer
- 50) Strategy
- 51) Quale varie categorie di pattern GOF esistono
- 52) Introducimi le classi concettuali
Rappresentano entità concettuali e non software ed essa può avere attributi ma non può avere operazioni e viene usato nel modello di dominio per descrivere cose alte del dominio del sistema
- 53) Introducimi le operazioni di sistema
Sono indicate negli SSD e vengono descritte da un contratto

sono indicate negli SD e vengono descritte da un contratto

- 54) Come passiamo dall'analisi alla progettazione
Avviene nel momenti in cui si passa dall'astrazione del mondo reale alle classi software
Questo passaggio avviene quando si passa dai contratti ai diagrammi di iterazione che vengono fatti a seconda dei contratti
Dove si vedono le iterazioni classi software
- 55) Vantaggi dello sviluppo guidato dai test
Serve per evitare di rendersi troppo tardi i bug e gestirli quando si verificano
Ed il programmatore sa già quali obiettivi raggiungere
Tipi di test:
- Unit
 - Integrazione
 - Sistema (scatola nera)
 - Accettazione (cliente)
- 56) **Quali sono i 3 metodi per dire se un caso d'uso è utile?**
- Test del capo
 - Il capo: che fai tutto il giorno
 - Tu: il login
 - Il capo sarà felice?
 - Test EBP
Serve per comprendere se un caso d'uso è un processo di business elementare:
 - Non è singolo passo
 - E' un attività singola in una sessione tra alcuni minuti ed 1 ora
 - Risposta ad un evento business
 - Aggiunge un valore di business misurabile
 - Lascia il sistema in uno stato stabile
 - Del della Dimensione
 - Né troppo breve
Quindi diversi passi
 - Formato dettagliato deve richiedere 3-10 pagine
- 57) Quale sono le tipologie di strati
- Rilassato
 - Stretto
- 58) Quale sono le 3 tipologie di visibilità di una variabile
- Pubblico
 - Private
 - Protected
 - Package
- 59) Differenza obiettivo e requisito

L'obiettivo è una intenzione, è un requisito non funzionale non misurabile

Un requisito è un obiettivo misurabile

60) Le 3 tipologie di attori (sono 4)

Qualunque cosa che abbia un comportamento

a. Primario

Quello che utilizza il sistema

b. Finale

Vuole che il sistema vuole essere utilizzato per i suoi fini

c. Supporto

Fornisce servizi al sistema

d. Esterno / Fuoriscena

Colui che ha interesse nel caso d'uso

61) Perché si usa il modello di dominio

Serve a rappresentare in maniera astratto le entità del mondo esterno

E serve per avere un'idea di possibili classi concettuali all'interno del nostro sistema

62) Perché si usa il diagramma interazioni (SD)

Permettono di mostrare lo scambio di messaggi tra classi software

63) Com'è strutturato un diagramma di iterazione

Si concentra molto su linee temporali, le classi software sono fatte da linee di vita

Quelle sopra avvengono prima di quelle sotto e queste rappresentano le chiamate di metodi

Si possono mettere frame che permettono di fare loop/scelte/simili

Le linee di vita possono essere create anche da questi diagrammi di sequenza, e partono da dove vengono create

64) Se ho bisogno di una operazione atomica?

Devi specificarlo con una nota siccome RSAD non lo permette di fare

Su non RSAD con un frame critical

65) Perché si usa il diagramma dei casi d'uso

È un diagramma che mostra in formato visivo i casi d'uso identificati

Li mette in relazione con gli attori.

Disposto a sinistra gli attori primari, a destra attori supporto

E mostra relazioni tra casi d'uso

66) Perché si usano i contratti

Sono un elaborato che serve per definire le operazioni di sistema che sono prese dall'SSD

Servono per costruire i diagrammi di sequenza siccome specificano prima e dopo l'operazione

67) È obbligatorio?

Nessun elaborato è obbligatorio, vanno fatti quando serve

68) Spiegami il refractoring

Insieme di tecniche che NON permettono di correggere i bug ma permettono di modificare struttura del codice senza modificare cosa il codice fa

69) La visibilità di default è privata

70) Cosa sono gli oggetti

Sono le classi istanziate

71) Cosa sono le classi concettuali

SONO classi non software ispirate al mondo reale

72) Perché usare diagramma dei package

L'architettura logica serve per dividere il software in macro-aree

E praticamente applicare low coupling e high coesion

I package sono in strati, ed i strati comunicano tra di loro o stretto o largo (guardare sopra)

73) Perché usiamo macchine a stati

Siccome vogliamo sapere come in una determinata classe il comportamento varia a seconda dello stato interno

74) Cos'è uno stato

Valori dei attributi di una classe

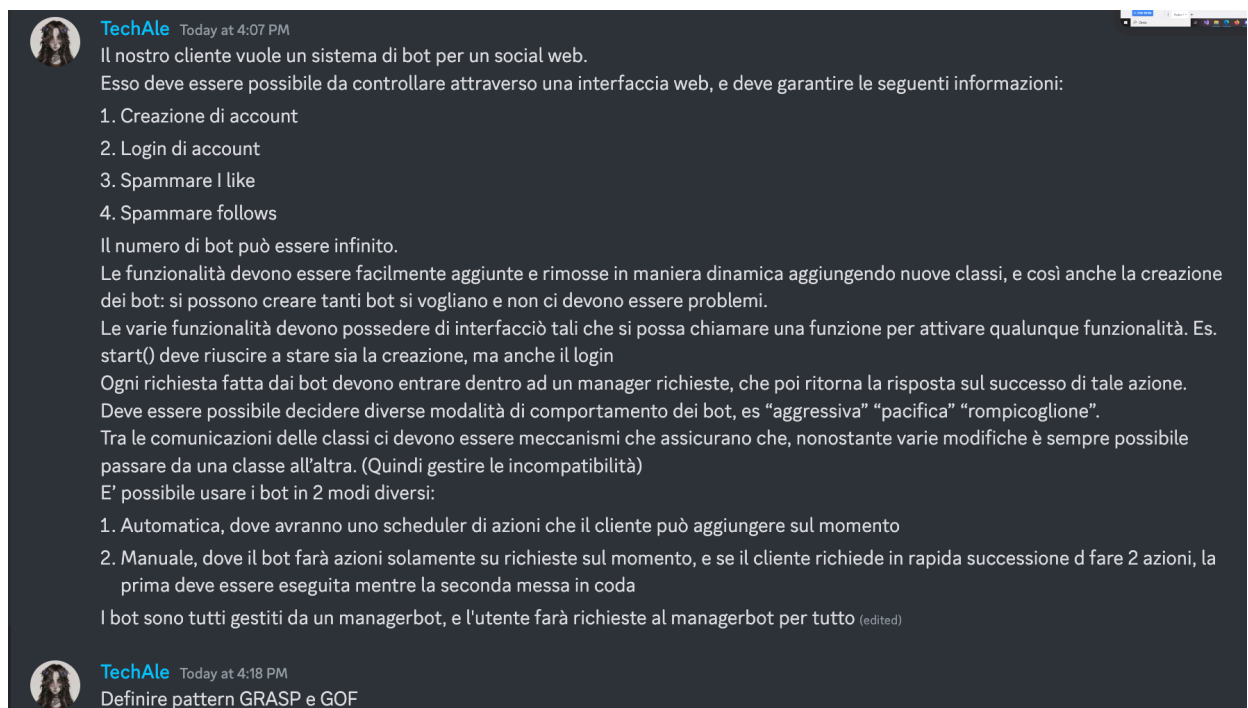
75) Differenza SCRUM e UP

UP è il processo unificato e si basa su processo iterativo incrementale iterativo

Quindi insieme best practice

SCRUM è qualcosa più specifica suddivisione compiti interno agile e singole iterazioni

76) F



TechAle Today at 4:07 PM

Il nostro cliente vuole un sistema di bot per un social web. Esso deve essere possibile da controllare attraverso una interfaccia web, e deve garantire le seguenti informazioni:

1. Creazione di account
2. Login di account
3. Spammare i like
4. Spammare follows

Il numero di bot può essere infinito.

Le funzionalità devono essere facilmente aggiunte e rimosse in maniera dinamica aggiungendo nuove classi, e così anche la creazione dei bot: si possono creare tanti bot si vogliono e non ci devono essere problemi.

Le varie funzionalità devono possedere di interfacciò tali che si possa chiamare una funzione per attivare qualunque funzionalità. Es. `start()` deve riuscire a stare sia la creazione, ma anche il login

Ogni richiesta fatta dai bot devono entrare dentro ad un manager richieste, che poi ritorna la risposta sul successo di tale azione. Deve essere possibile decidere diverse modalità di comportamento dei bot, es "aggressiva" "pacifica" "rompicoglione".

Tra le comunicazioni delle classi ci devono essere meccanismi che assicurano che, nonostante varie modifiche è sempre possibile passare da una classe all'altra. (Quindi gestire le incompatibilità)

E' possibile usare i bot in 2 modi diversi:

1. Automatica, dove avranno uno scheduler di azioni che il cliente può aggiungere sul momento
2. Manuale, dove il bot farà azioni solamente su richieste sul momento, e se il cliente richiede in rapida successione di fare 2 azioni, la prima deve essere eseguita mentre la seconda messa in coda

I bot sono tutti gestiti da un managerbot, e l'utente farà richieste al managerbot per tutto (edited)

TechAle Today at 4:18 PM

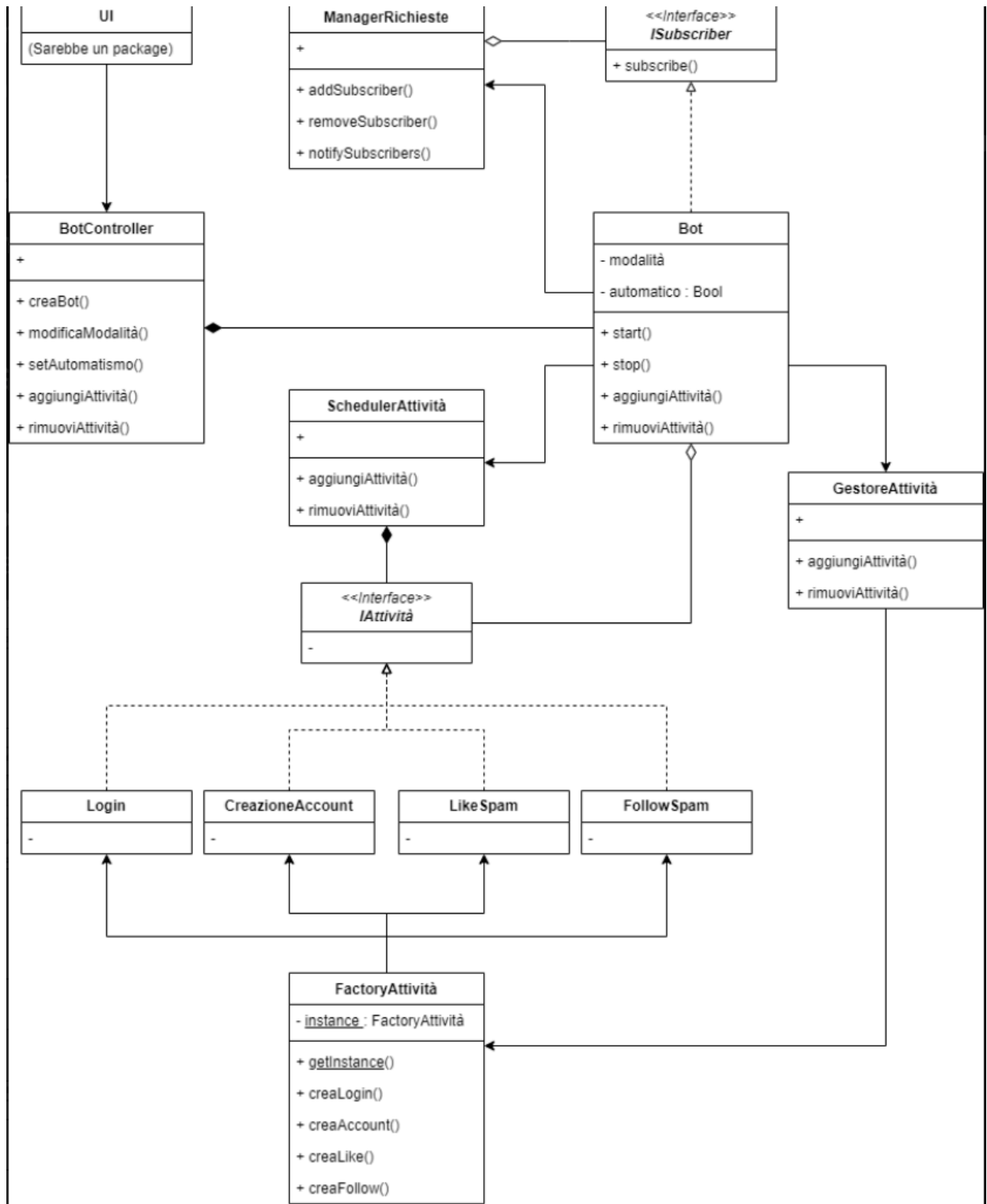
Definire pattern GRASP e GOF



Okami Today at 5:40 PM

GRASP: Li diciamo a voce perchè siamo pigri 😊

GOF: Facade, Factory, Observer, Singleton, Strategy (edited)



- 77) Prendete il file main.py
E cercate di fare un refactoring pesante.
Trovate tutti i code smell, segnateveli.

Questo file main.py è un qualcosa che ho creato nel 2019 quando avevo appena iniziato a fare progetti di programmazione seri, quindi troverete tanti errori semantici.

Ho appena letto che il riganelli fa domande stronzissime all'esame...

78) Dimmi le discipline del metodo agile

Le discipline si differenziano dalle fasi siccome

Le fasi riguardano tutto processo software

Discipline invece singole iterazioni

Inanzitutto è processo unificato.

- **Modellazione del business**

- Riguarda la stesura del modello di dominio

- Che è il modello delle classi concettuali

- Che è ispirato alle entità del mondo reale

- **Requisiti**

- Riguarda casi d'uso e la stesura

- Glossario, diagrammi sequenza di sistema ed i contratti

- **Progettazione**

- Stesura dei diagrammi di iterazioni e diagramma delle classi

- **Implementazione**

- Scrittura del codice e testing

- **Gestione del progetto**

79) Cosa sono i pattern grasp e fammi l'elenco

80) Soluzioni pattern grasp

I pattern sono una coppia problema soluzione ben conosciuta per l'assegnazione delle responsabilità

- a. **Creator**

- A chi si assegna la responsabilità creare l'oggetto?

- Viene dato a chi ha le informazioni

- Viene fatto dalla factory

- b. **Information expert**

- E' un pattern grasp che riguarda l'assegnazione delle responsabilità di conoscere ad una classe che ha le informazioni sull'altra classe

- c. **Low coupling**

- Accoppiamento tra le varie classi è il basso possibile

- E per accoppiamento intendo quanto le suddette classi sono indipendenti tra di loro.

- d. **Controller**

E' la prima classe del modello di dominio che si interfaccia con l'interfaccia utente

Serve per separare il layer dell'interfaccia utente dall'interfaccia di domini

e. High cohesion

Riguarda l'assicurarsi di mantenere un alta coesione con una classe
Aka assicurarsi che una classe non si occupi di troppe funzionalità diverse

Per mantenerlo ci si deve assicurare che il compito è specifico

f. Pure fabrication

Serve quando dobbiamo assegnare delle responsabilità ma non a nessun'altra classe

E qui creiamo un'altra classe

g. Poliformismo

h. Indirection

Pattern GRASP che riguarda il problema di disaccoppiare 2 classi, e per fare ciò viene introdotta una 3 entità, di norma adapertr, che serve per connettere le 2

i. Protected variations

I cambiamenti di 1 classe si riversano su 1'altra classe

E viene garantito con una interfaccia stabile

81) Pattern GOF

Sono una soluzione progettuale comune a un problema di progettazione ben noto

a. Adapter

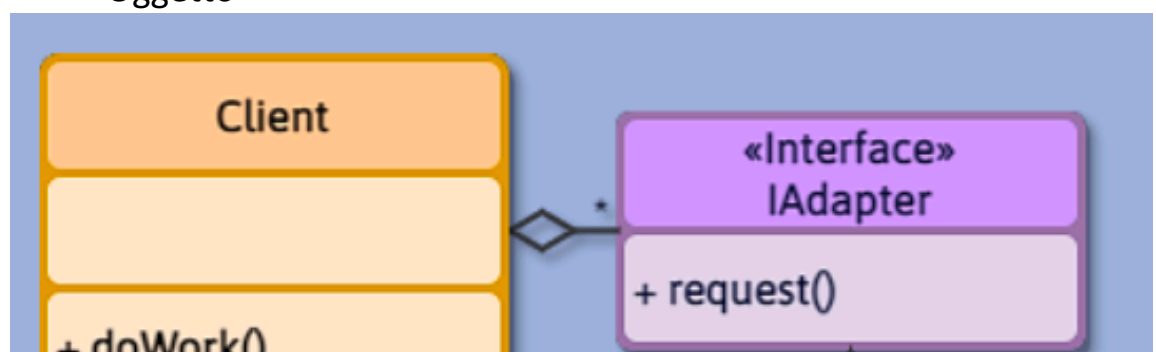
Serve per risolvere il problema di offrire un'interfaccia stabile a diverse classi che supportano interfacce incompatibili.

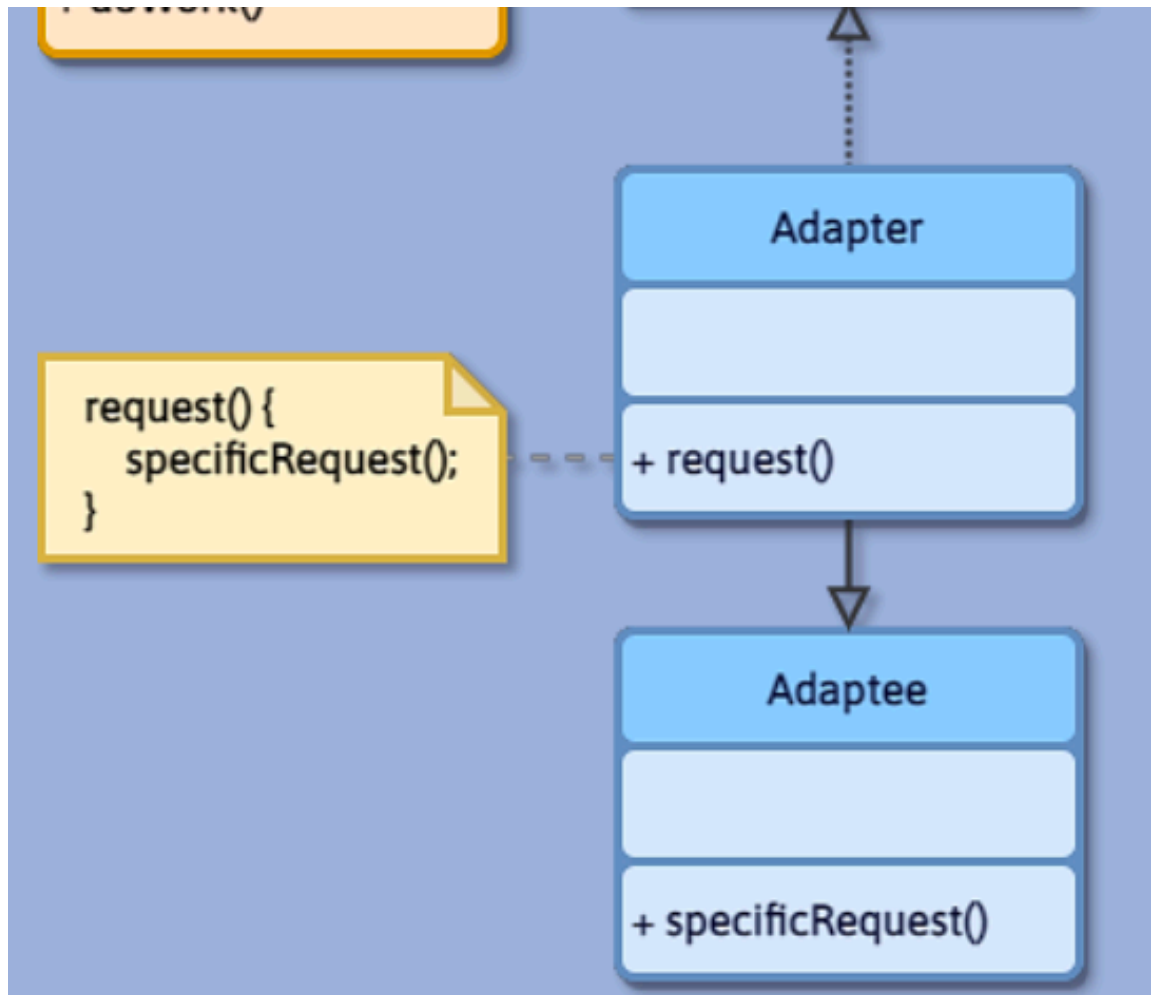
Funziona da intermezzo per adattare una richiesta specifica da interfacce differenti

Noi quindi convertiamo una richiesta generica che chiamiamo request che la converte in una richiesta specifica

Esistono 2 tipologie:

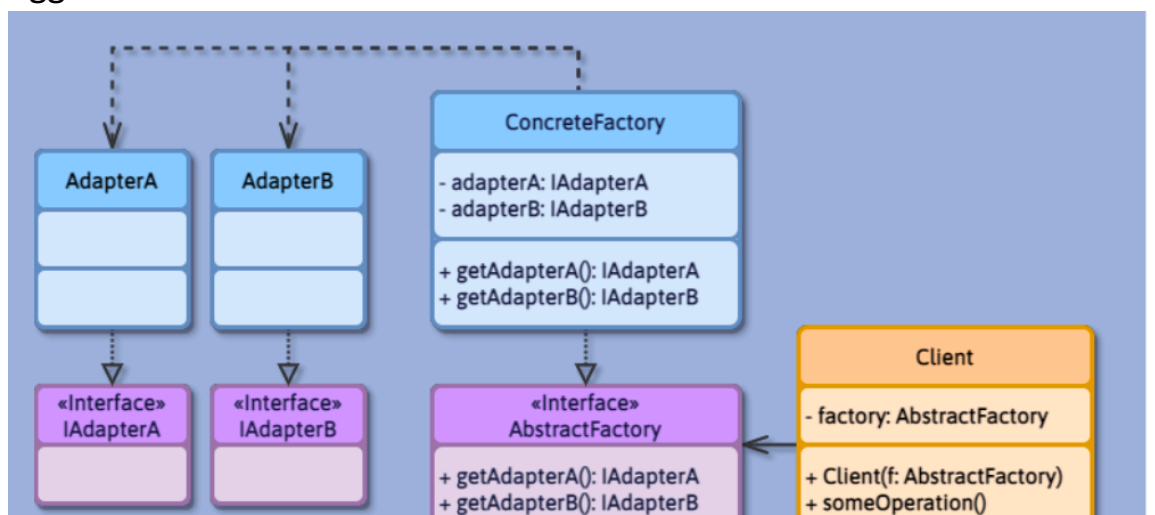
- Classe
- Oggetto





b. Factory

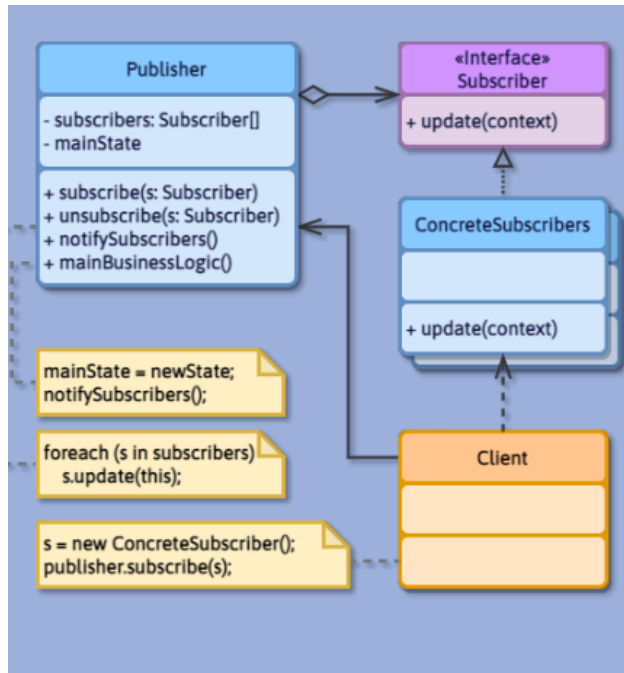
Creare oggetti con meccanismi creazione meccanismi complessi
Viene creata classe pure fabrication che serve da mediatore per creare oggetti necessari



c. Observer

Diverse classi vogliono reagire a determinati eventi che vengono generati da un'altra classe chiamata publisher
E gli altri sono chiamati subscriber



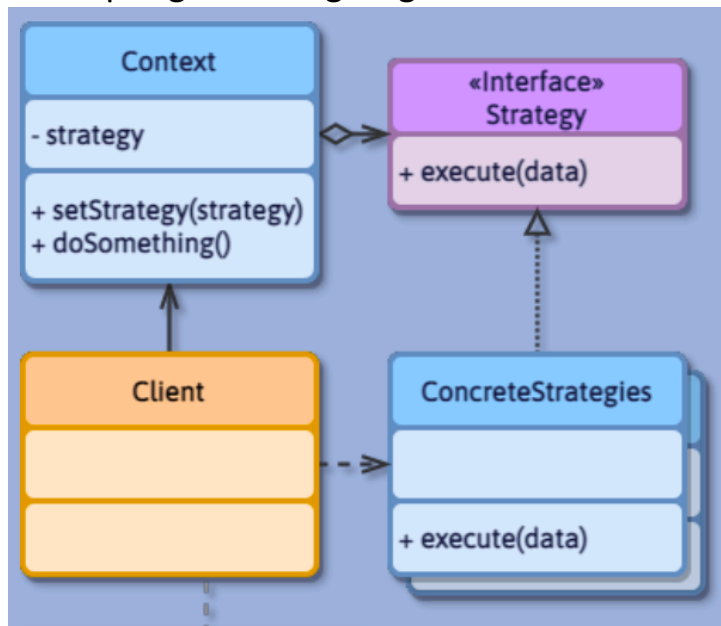


d. Facade

E' la prima classe che si interfaccia con l'interfaccia utente, e questa andrà a chiamare altri controller di supporto. Serve come un punto di incontro

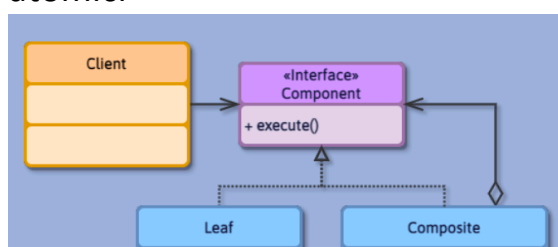
e. Strategy

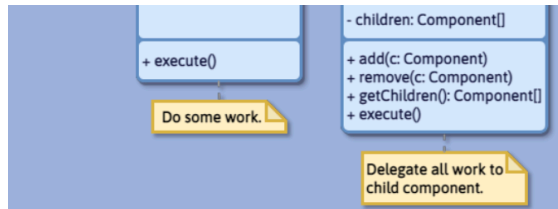
Serve per gestire degli algoritmi differenti sotto una interfaccia comune



f. Composite

Come gestire un gruppo di oggetti dello stesso tipo come se fossero atomici

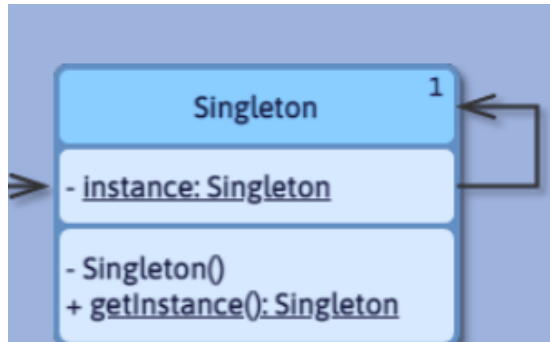




g. Singleton

Risponde necessarietà di 1 classe con 1 sola stanza con accesso a tutto il sistema facilmente.

Non bisogna abusarne siccome si rischia di aumentare l'accoppiamento complessivo delle classi del sistema



82) Quali sono le varie possibilità nelle post condizioni

- a. Creazione/Distruzione
- b. Modifica attributi
- c. Creazioni associazioni

83) Cos'è il diagramma di domini

E' una rappresentazione visuale del dominio, in particolare delle classi concettuali

84) Definizione code smelling

Sono cattive pratiche di scrittura del codice, ed essi non risolvono eventuali bug.

Indicano una serie di caratteristiche di cattive pratiche

Esempi:

- a. Feature envy, cioè un metodo di una sottoclassi accede più attributi di altre classi anziché sua classe
- b. Large class, quando una classe è troppo grande, ci sono troppi metodi/troppe responsabilità, si suddivide la classi in sotto classi, si potrebbe estrarre
- c. Long parameter list, una funzione ha troppi parametri, si potrebbe definire una parameter class (una classe che rappresenta 1 parametro)
- d. Shotgun surgery, un code smell nel quale si verifica che per effettuare 1 modifica si effettuano tante piccole modifiche. Per risolverlo si spostano queste funzionalità dentro 1 classe/si crea una nuova
- e. Switch statement. una serie di if else molto lunga e la notremmo

- e. ~~union statement~~, una serie di ~~if~~ ~~more~~ ~~range~~ e la ~~potremmo~~ risolvere creando sottoclassi/funzioni
- f. Refused bequest si ha quando una sottoclasse non utilizza tutti i metodi della superclasse
E si risolve modificando questa ereditarietà

85) Parlami della protected variations e come funziona, a cosa serve e come viene implementato

Serve per evitare che determinati cambiamenti si riversano su altre classi.

Per farlo si utilizzano interfacce stabili per accedere alle cose notevoli

Le classi concettuali sono un'idea/oggetto del mondo reale e sono definite:

- a. Simboli, parola/immagine
- b. Intensione
- c. Estensione -> Insieme oggetti

E lo usiamo

Nell'analisi per comprendere il dominio

E poi usato come fonte di ispirazione per lo strato di dominio

86) Definisci relazioni tra responsabilità e metodi

I metodi vengono creati per adempiere le responsabilità

87) Illustra i vari elaborati di requisiti

- a. Modello dei casi d'uso
- b. I casi d'uso dettagliati
- c. I diagrammi di sequenza di sistema
- d. I contratti
- e. Il glossario
- f. La visione economica

88) Disegna pattern ()

89) Come funziona UP

Sta per Unified process ed è un processo iterativo, incrementale ed evolutivo per lo sviluppo software.

È flessibile, pilotato dai casi d'uso e dai rischi ed incentrato sull'architettura.

Esso avviene in fasi, ogni fase è divisa in varie iterazioni è delimitata da un tempo chiamato timeboxe.

Le varie fasi sono: ideazione, elaborazione, costruzione e Transizione

Nelle prime iterazioni nell'UP bisogna affrontare i rischi maggiori ed una architettura coesa, usando intensivamente testing.

- Ideazione all'avvio del progetto, con una visione approssimata del progetto, studio economico, costi e tempi
- Elaborazione per la creazione del nucleo dell'architettura dove si identificano i requisiti
- Costruzione delle capacità iniziali e successiva preparazione al rilascio
- Transizione per il completamento del prodotto

Ogni iterazione è una finestra di tempo dove si completano mini progetti, e