Argomenti POO Esposito

Definizioni/Semplici Spiegazioni

Definizione Information Hiding: Tramite l’information hiding scegliamo di nascondere i dettagli implementativi interni di una componente software, in questo modo riduciamo la dipendenza e miglioriamo la manutenibilità del codice. La parte esterna interagisce solo con un interfaccia pubblica esterna.

Lezione 2

Binding: L'azione in cui specifichiamo il valore di un attributo questo può avvenire
- in fase di definizione del linguaggio
- durante l'implementazione del linguaggio
- durante la traduzione del programma
- durante l'esecuzione del programma.

Parliamo di binding statico quando avviene prima dell'esecuzione del programma altrimenti binding dinamico.

Tipi di Conversione: Conversione implicita ovvero coercizione (conversione svolta dal compilatore), conversione esplicita ovvero il casting.

Classi Di Memorizzazione: Pre classe di memorizzazione di una variabile intendiamo visibilità spaziale, durata temporale e la sua allocazione di memoria durante l'esecuzione.
Per un processo abbiamo 4 aree
1) Area programmi e costanti, contiene le istruzioni e le costanti.
2) Area dati statici, contiene le variabili allocate staticamente e quelle esterne.
3) Area Heap, contiene le variabili dinamiche esplicite come un array.
4) Area Stack, contiene le variabili automatiche come gli interi e quelle definite all'interno di funzioni.

Tipi Wrapper: Sono oggetti che hanno l'unico scopo di contenere un valore primitivo, con l'espressione AutoBoxing intendiamo che le variabili primitive vengono convertite e usate come la loro controparte Wrapper.

Variabili Final: Sono delle variabili costanti anche se il loro valore non è necessariamente noto a tempo di compilazione. Con solo final avremo una "costante" per ogni istanza della classe mentre con static finale avremo una vera e propria costante.

Tipo Enumerativo: In Java un ENUM è una tipo di dato che rappresenta un insieme fisso di costanti, alla fine in Java sono come delle classi infatti possono avere anche un costruttore, metodi e variabili come una vera e propria classe.

Garbage Collector: Modulo che in maniera automatica recupera la memoria degli oggetti non più utilizzati, ovvero che non sono più riferiti. Libera la memoria dell'heap distruggendo gli oggetti non più raggiungibili. Il suo primo passo è controllare quale "pezzi di memoria" dell'heap sono in uso o meno. Il secondo passo consiste nell'eliminare quest'ultimi e compattare quelli rimanenti. Un oggetto appena creato va nella Generazione Giovane dove sono presenti tutti gli oggetti appena creati qui si attiva un minor Garbage Collector. I superstiti vanno nella Vecchia Generazione nella quale avremo un major Garbage Collector.
- Serial GC (più semplice e heap ridotti)
- Parllel (a più thread per carichi di lavoro grandi e con pause lunghe)
- Current Mark Sweep (minimizza le pause)
- Garbage-First(per heap > 4gb).

Lezione 3

Meccanismi di Astrazione: Astrazione sul controllo (astrae una funzionalità dai dettagli implementativi) e astrazione sui dati (astrae entità descritte in termini di strutture dati).

Metodologie di progetto: "top-down" e "bottom-up"

Variabile Static: Variabile allocata una sola volta indipendentemente dagli oggetti della classe e accessibile da tutti gli oggetti di quella classe.

Classi Interne: Quando posizioniamo la definizione di una classe all'interno di un'altra. Questa ha completa visibilità degli attributi e dei metodi che sono nella classe che la contiene. Un istanza di una classe interna è ottenibile solo a partire da un'istanza della classe esterna, possiamo comunque disciplinare la visibilità della classe con le parole chiave.

Classi Anonime: Sono delle sottoclassi specificate dopo l'operatore new, nel caso sia un interfaccia la classe anonima si considera implementazione dell'interfaccia. Non possono avere costruttori.

Overloading: Possibilità di dare a funzioni diverse lo stesso nome a patto che abbiano liste di parametri diverse. Si usa quando si devono definire funzioni concettualmente simili da applicare a dati di tipo diverso oppure se l'algoritmo realizzato da ciascuna funzione è diverso in base al tipo di dato a cui la funzione deve essere applicata.

Lezione 4

Definizione Ereditarietà: L'ereditarietà consente di definire nuove classi ereditando le caratteristiche offerte da classi esistenti. Realizziamo un classe base che realizza un comportamento comune mentre le classi derivate realizzano comportamenti specializzati. Grazie all'eredità quando si chiama un metodo su un oggetto l'interprete cerca prima la definizione nella classe dell'oggetto stesso, se non la trova la cerca nella superasse e risale la gerarchia fino a trovarla. Nel caso esistano più metodi con lo stesso nome viene eseguito quello trovato per primo.

Ovveriding: Nella fase di specializzazione di una classe vengono ridefinite funzioni già implementate nelle classi padre.

Modalità di protezione classi: Esistono tre modalità di protezione.
1) Privati non accessibili all'esterno della classe che li ha definiti.
2) Pubblici accessibili all'esterno della classe da chiunque.
3) Protetti non accessibili all'esterno della classe ma accessibili alle classi derivate.

Visibilità attributi e metodi:
1) Privato, visibilità solo dall'ambito della stessa classe
2) Default, visibilità solo all'interno dello stesso package
3) Protetto, visibilità dalle sotto classe e dallo stesso package.
4) Pubblico, visibilità completa.

Final sui metodi e classi: La parola chiave final sui metodi non permette di fare l'ovveriding di quei metodi nelle sottoclassi, nelle classi invece non permette l'ereditarietà, ovvero che non la si può specializzare.

Classe Astratta: Una classe che può includere o meno metodi astratti, queste non possono essere istanziate ma possono essere sottoclassi. Un metodo si dice astratto quando è dichiarato senza un'implementazione. Quando si deriva un classe astratta la sottoclasse deve implementare tutti metodi astratti, altrimenti dovrà essere astratta.

Interfacce: Le interfacce sono delle classe astratte con tutti metodi astratti e attributi che sono o static o final. Una interfaccia può derivare da un'altra interfaccia o una classe astratta che non implementa

metodi. Da Java 8 si può fornire un implementazione di default dei metodi delle interfacce, in modo da non obbligare chi usa quella interfaccia a scrivervi il metodo. In una interfaccia si possono avere metodi con implementazione solo se sono static.

Definizione Polimorfismo: Per polimorfismo si intende la proprietà di un'entità di assumere forme diverse nel tempo.

Lezione 5

Benefici Generics rispetto Object:
- Un codice con i generics ha un controllo più forte sul tipo a tempo di compilazione.
- Permette di eliminare il casting.
- Consente l'implementazione di algoritmi generici.

Definizione Tipo Generico: Un tipo generico è una classe o un interfaccia che è stata parametrizzata rispetto ai tipi delle variabili che impiega. Le variabili <T> che utilizziamo per parametrizzare possono rappresentare ogni possibile tipo non primitivo.

Definizione Tipo Raw: Sono i tipi resi generici con l'impiego di Object.

Parametro Limitato: Limitiamo i parametri superiormente con la parola extends in questo modo limitiamo i tipi che possiamo usare al tipo dichiarato dopo extends e a tutte le sue sottoclassi, è possibile anche definire più di un vincolo sul parametro di tipo.

Inferenza di tipo: L'inferenza di tipo è la capacità del compilatore di guardare ogni invocazione di metodo e la relativa dichiarazione per determinare l'argomento di tipo che consente l'applicazione dell'invocazione. L'algoritmo di inferenza determina i tipi degli argomenti e se disponibile il tipo a cui il risultato è assegnato o ritornato.

Tipi Target: I tipi target rappresentano i tipi di dato che il compilatore si aspetta in funzione di dove l'espressione appare nel contesto di un programma.

Wildcard: Rappresenta un tipo sconosciuto e può essere usata in vari modi come tipo per un parametro, campo o variabile locale.

Quando conviene non limitare una Wildcard: Non conviene limitare una Wildcard se si sta scrivendo un metodo che è implementabile con i metodi definiti nella classe Object oppure quando il codice sta usando metodi nella classe generica che non dipendono sul parametro di tipo.

Cattura Wildcard: Quando il compilatore determina il tipo di una Wildcard in base al codice.

Cancellazione del tipo: Processo mediante il quale le annotazioni di tipo esplicite vengono rimosse da un programma prima che venga eseguito. Per implementare i Generics il compilatore Java applica la cancellazione del tipo per sostituire tutti i parametri di tipo nei tipi generici con i vincoli o Object, il bytecode contiene quindi solo classi interfacce e metodi ordinari. Inoltre si occupa anche di inserire il cast di tipo e di inserire dei metodi ponte(metodo che serve per garantire che la sotto-tipizzazione funzioni come ci si aspetti) per preservare il polimorfismo in tipi generici estesi.

Reificazione: Processo Attraverso il quale un concetto astratto viene reso in un modello dei dati o oggetto computabile detto risorsa attraverso istruzione messe a disposizione dal linguaggio di programmazione. Reificare un concetto significa renderlo come un tipo, un oggetto che supporti le operazioni più comuni.

Tipo reificabile: Un tipo reificabile è un tipo le cui informazioni sono pienamente disponibili a tempo di esecuzione e include tipi primitivi, non generici, tipi raw e Wildcard non vincolate. I non tipi reificabili sono tipi le cui informazioni sono state rimosse a tempo di compilazione per effetto della cancellazione di tipo

come List<String>.

Heap Pollution: Fenomeno che si verifica quando una variabile di un tipo parametrizzato si riferisce ad un oggetto che non è dello stesso tipo parametrizzato. Ad esempio quando proviamo ad aggiungere un intero ad una lista di stringhe. Problematico con i VarArgs usati con i generics (T... nomeArray).

Restrizioni Generics:
1) Non è possibile istanziare tipi generici con tipi primitivi (NO List<int>)
2) Non è possibile istanziare i parametri di tipo (su una classe parametrizzata su E NO "E oggetto = new E();")
3) Non è possibile dichiarare dei campi statici i cui tipi sono parametri di tipo (su una classe parametrizzata su E NO "private static E;")
4) Non è possibile usare cast o istanceof, poiché il compilatore cancella tutti i parametri di tipo
5) Non è possibile creare array di tipi parametrizzata.
6) Non è possibile intercettare o sollevare oggetti di tipi parametrizzata, una classe generica non può estendere Throwable né direttamente né indirettamente.
7) Non è possibile fare overloading su un metodo dove i tipi dei parametri formali di ogni sovraccarico cancellano lo stesso tipo raw.

Lezione 6

Definizione Espressione Lambda: La funzione Lambda è una funzionalità che consente allo sviluppatore di rendere una funzionalità come un argomento di un metodo o il codice come un dato, con una sintassi migliore e più compatta rispetto alle classi anonime.

Definizione Interfaccia funzionale: Un'interfaccia che contiene uno e un solo metodo astratto, può comunque avere molteplici metodi di default e statici.

Riferimento a metodo: Sono tipi speciali di espressioni lambda che eseguono un solo metodo.

Quando usare classi interne, classi locali, anonime , espressioni lambda:

1) Classi interne: Vanno usate se i requisiti sono simili alle classe locali e si vuole rendere la classe maggiormente disponibile e non si richiede accesso a variabili locali o parametri di metodo.
2) Classe locale: Va usata per creare più istanze di una classe, accedere al suo costruttore e introdurre nuovi tipi
3) Classe anonime: Va usata se lo sviluppatore richiede di dichiarare campi o metodi addizionali.
4) Espressioni Lambda: Vanno usate se si intende incapsulare una singola unità di esecuzione da passare a un altro codice oppure per avere un singola istanza di un interfaccia funzionale.

Lezione 7

Definizione Stream: Uno stream è un'interfaccia che restituisce un flusso di dati finito o infinito su cui è possibile fare varie operazioni. Non è una raccolta di oggetti ma un modo per manipolare i dati in maniera dichiarativa e compositiva. Lo stream è composto da 3 parti:
1) Sorgente una raccolta come una Collection
2) O o più funzioni intermedie
3) Operazione di terminazione, operazioni che producono in uscita un valore.

Definizione Pipeline: Una serie di operazioni aggregate.

Collezioni Immutabili: Collezioni nelle quali se si prova ad aggiungere, aggiornare o eliminare un elemento viene sollevate un'eccezione.

Differenza tra generate() e iterate(): Iterate assume un valore iniziale e una espressione lambda per generare i valori successivi, generate produce anch'esso un stream infinito ma in modo non sequenziale.

Differenze Stream e Collection: La differenza principale sta nel quando gli elementi sono elaborati. Una Collection è una struttura dati in memoria che mantiene i suoi valori, uno stream è una struttura concettualmente fissa i cui elementi sono computati su richiesta. Le Collection sono costituite in maniera eager, ovvero risolvono una espressione non appena questa viene legata ad una variabile, gli stream sono costruiti in maniera lazy ovvero l'espressione può essere valutata solo quando si richiede il suo valore.

Optionl: L'optional è un tipo a cui è possibile associare funzioni per trasformare il valore al suo interno.

Riduzioni: Operazioni che aggregano gli elementi in unico risultato, le principali sono reduce() come sum oppure collect() che crea una lista.

Quando usare il parallelismo: Si deve considerare il suo uso quando bisogna gestire un grande set di dati da elaborare o il flusso di dati è divisibile.

Lezione 8

Definizione Reflection: La Reflection permette la manipolazione di classi, metodi e attributi in maniera non convenzionale al paradigma della programmazione Object Oriented. Le classi si trovano all'interno del package reflection sono pensate per l'interrogazione dinamica di istanze di classi java. Per reflection si intende l'abilità di un programma di interrogare e modificare il suo stato in maniera simile ai suoi dati durante la sua esecuzione.

Introspection: Abilità di un programma di interrogare o ottenere informazioni su se stesso.

Intercession: Abilità di un programma di modificare il suo stato di esecuzione, alterando anche la sua interpretazione.

Suddivisione Reflection: Le reflection si suddivide in due categorie (In Java Intro>Inter)
1) Strutturale, ovvero l'abilita di un programma di interrogare l'implementazione dei suoi dati e del suo codice (Introspection) oppure l'abilita di modificare o creare nuove strutture dati e codice (Intercession)
2) Comportamentale, l'abilità di un programma di ottenere informazioni sul suo contesto di esecuzione (Introspection) , oppure l'abilita di modificare il suo contesto di esecuzione (Intercession)

Letterale di una classe: Un letterale di una classe è il nome di una classe seguito da un punto e dalla parola class. Il riferimento al letterale si può avere partendo da ogni istanza utilizzando il metodo getClass(). Un campo di una classe è rappresentato da un'istanza di Field. Per i metodi abbiamo la classe Method. Per i costruttori Constructor.

Annotazioni: Le annotazioni forniscono informazioni circa un programma che non è parte del programma e non hanno diretto effetto sul comportamento del codice che esse annotano. Le annotazioni danno informazioni per il compilatore, elaborazioni a tempo di compilazione e elaborazione a tempo di esecuzione. Le annotazioni sono seguite da @ e possono anche includere degli elementi.

Extra
-
-
-

Argomenti POO La Torre

Definizioni/Semplici Spiegazioni

Lezione Polimorfismo

Polimorfismo vs Overloading: Entrambi invocano metodi distinti con lo stesso nome ma con l'overloading la scelta del metodo appropriato avviene in fase di compilazione (early binding) esaminando il tipo dei parametri, mentre con il polimorfismo la scelta avviene in fase di esecuzione dalla JVM (late binding)

Come viene scelto il metodo grazie al polimorfismo: A runtime dallo spazio dell'oggetto si segue il link al codice da eseguire.

Lezione Interfacce

Definizione Interfaccia: Un interfaccia dichiara un insieme di metodi elencandone le firme con il tipo di valore restituito, il tutto non fornendo nessuna implementazione. Le variabili che contiene vengono considerate final e static. Un interfaccia esprime una caratteristica che può essere comune a concetti diversi, non è un concetto in sé. Una classe può realizzare (implementare) più interfacce. I metodi di un interfaccia sono polimorfici poiché cambiano in basse alla classe che l'implementa.

Variabile di tipo interfaccia: Una variabile che può contenere il riferimento ad oggetti delle classi che implementano l'interfaccia.

Differenze SuperClasse e Interfacce: Entrambe definiscono super-tipi e consentono di definire metodi polimorfici. Un'interfaccia non è una classe visto che non ha uno stato né un comportamento è solo un elenco di metodi da implementare. Una super-classe è una classe. Inoltre un'interfaccia non può avere variabili di istanza.

Cosa può estendere un interfaccia: Un'interfaccia può estendere altre interfacce.

Differenze e analogie tra classi astratte e interfacce: Le classi astratte e le interfacce sono simili perché catturano astrazioni che possono raggruppare aspetti comuni di concetti differenti, tuttavia le classi in generale forniscono un'implementazione di alcuni metodi. Le classi astratte sono fatte per concetti strettamente correlati, le interfacce per enfatizzare un aspetto comune di concetti eterogenei.

Conversione tra tipi: Risulta sempre possibile convertire dal tipo di una classe al tipo di un'interfaccia implementata dalla classe. Eccezione in caso di errore ClassCastException. Operatore "instance of" ci permette (grossolanamente) di vedere se quell'oggetto appartiene a quel determinato tipi.

Lezione scrivere codice Riutilizzabile

Cosa si intende per riutilizzo del codice: Scrivere un'unica soluzione algoritmica in diversi contesti.

Programmazione Generica: Creazione di costrutti che possono essere utilizzati con tipi di dati diversi. Ci semplifica poiché non dobbiamo fare manualmente il casting

Tipi generici: Sono dichiarati tra le parentesi <> e sono indicati con la lettera maiuscola, sono utilizzati come tipi per dichiarare le variabili.

Problemi del riutilizzo del codice: In alcune classi non possiamo implementare le nostre interfacce generiche, implementando poi una singola interfaccia possiamo appunto dare una singola

implementazione.

Lezione Eccezioni

Errori: La superclasse di tutti gli errori è throwable e definisce il caso base di ogni cosa che può essere lanciata.

Classe Error: Utilizzata per indicare una situazione anomala in esecuzione fuori dal controllo dell'applicazione stessa (esaurimento risorse JVM , versioni obsolete). Problema serio non prevedibile.

Definizione eccezione: Un evento che interrompe la normale esecuzione del programma.

Classe Exception: Casi di errori più comuni gestibili dal programma. Quando si verifica un'eccezione il controllo passa ad un gestore delle eccezioni.

Differenza tra eccezioni controllate e non: Le eccezioni non controllate derivano da RunTimeException e riguardano errori che sono evitabili con un attenta programmazione. Le eccezioni controllate derivano da exception(escluso il ramo di RunTimeException) e riguardano errori causati da eventi esterni o situazioni che richiedono l'attenzione del programmatore.

Come gestire le eccezioni: Abbiamo due possibilità segnalare esplicitamente che il gestore delle eccezioni deve eseguire solo le operazioni di routine (throws nella firma) oppure utilizzare un blocco try-catch. Se non catturiamo le eccezioni queste provocheranno l'arresto del programma.

Clausola finally: Questa clausola serve per indicare un'istruzione che va eseguita sempre, indipendentemente dall'esito del try-catch.

Come il programma gestisce il lancio delle eccezioni: Al lancio di una eccezione viene interrotta l'esecuzione normale del programma e il controlla passa al gestore delle eccezioni. Se si è all'interno di un modulo try vengono analizzate le clausole catch se viene eseguita una clausola catch il gestore delle eccezioni termina e passa il controllo al modulo principale della JVM. Se non c'è un catch si ritorna al chiamante si controlla se ci sono eventualmente clausole finally e si arresta il programma e si stampa l'errore.

Vantaggi delle eccezioni: Separano il trattamento degli errori dal codice.

Lezione I/O

Come sono definiti in Java input e output: In Java input e output sono definiti in termini di flussi (stream), abbiamo due tipi di flussi, quelli di dati binari e quelli di caratteri.

Eccezione Lanciata nel caso: IOException, FileNotFounException

Classi per i flussi di byte: FileInputStream e FileOutputStream + PrintStream (NO EXCEPTION) che aggiunge a FileOutputStream i metodi per semplificare la stampa dei dati.

Classi per leggere i flussi di caratteri: FileReader, questi leggono un carattere per volta se vogliamo leggere righe o stringhe possiamo utilizzare un oggetto di tipo Scanner, per la scrittura abbiamo PrintWriter (metodi come PrintStream)

Classe File: Astrae il concetto di File può essere manipolato per file esistenti ma non può creare un file se non esiste. Non possiamo utilizzarlo direttamene per leggere/scrivere. [Metodi delete, renameTo, lenght, exists]

Differenze accesso sequenziale e acceso casuale: Nell'accesso sequenziale viene elaborato un byte alla volta , nell'accesso casuale possiamo accedere a posizioni arbitrarie all'interno del file, questo solo sui file del disco. Per l'accesso casuale usiamo RandomAccessFile (IOException). Memorizza i dati in binario.

Flussi di oggetti: Consentono di operare su interi oggetti. Le classi sono ObjectOutputStream e ObjectInputStream.

Serializzazione: La serializzazione è la trasformazione di un oggetto in un flusso di byte, oltre ai valori delle variabili di istanza vengono inserite anche delle informazioni per individuare la classe corrispondente

Interfaccia Serializable: Come clonable è un interfaccia di marcatura serve solo a verificare che siamo a conoscenza della serializzazione. (Può lanciare ObjectStreamException)

Cos'è il serial version UID: Identificatore associato ad ogni classe serializzabile. Viene utilizzato nella deserializzazione per controllare che la versione della classe individuata nella JVM sia la stessa dell'oggetto.

Eccezione lanciata dalla serializzazione: Se il serial version UID non coincide viene lanciata la InvalidClassException

Requisiti per poter serializzare un oggetto: Tutte le variabili di istanza non primitive devono essere di tipo serializzabile, se una di esse non lo è, e non vogliamo serializzarla possiamo usare lo specificatore transit.

Lezione programmazione grafica

Frame per Thread: La JVM esegue ogni frame su una thread separata

---

Lezione programmazione a eventi

Cos'è un evento: Un qualcosa che viene generato ogni volta che l'utente esegue un'azione.

Interfacce Listner: Sono ricevitori di eventi ed esiste un'interfaccia listenener per ciascun tipo di evento.

Lezione Clonazione Oggetti Immutabili e altro

Problema della clone di Object: Crea una copia superficiale, ne copia il riferimento, crea una copia corretta solo se contiene valori primitivi e in generale oggetti immutabili

Perché la clone è protected: La clone è protected in modo tale da poterla usare solo nella classe di quello oggetto.

Quando la clone di object funziona (eccezione lanciata): La clone funziona quando implementiamo l'interfaccia di marcatura Clonable in caso contrario l'eccezione che viene lanciata è la CloneNotSupportedException, eccezione che va comunque catturata anche se implementiamo clonable.

Usare la clone per l'incapsulamento: Nei set assegniamo una clone dell'oggetto passato, nei get ritorniamo la clone dell'oggetto.

Oggetti immutabili: Sono oggetti che non modificano il loro stato, non hanno i metodi modificatori e restano sempre nello stato che assumono all'istanziazione. Utili perché semplificano l'incapsulamento dei dati e la clonazione degli oggetto, inoltre nei programmi concorrenti non generano problemi di thread e consistenza della memoria. Per definirli No Set, variabili private , e classe final, in alternativa costruttore privato. Usare la clone nei get.

Quando usare protected: La si usa per forzare l'uso di alcuni metodi solo da oggetti della stessa classe o in una sottoclasse. Di solito è una protezione per evitare un uso scorretto di alcuni metodi.

Covariant return type: Si può sovrascrivere un metodo usando la stessa firma ma cambiando il tipo di return con un sotto tipo.

Dove è messo il bytecode della classe interna: Il bytecode di una classe interna viene messo in un file separato, sotto questo punto di vista è trattata come una classe standard. Il file class delle classi esterne ha un nome che combina con il nome della classe ospitante e quello della classe interna separati dal $.

Cosa succede quando istanziamo un oggetto di una classe interna: Quando istanziamo un oggetto della classe interna, una variabile nascosta è assegnata con il riferimento all'oggetto in cui la classe interna è istanziata. In questo modo si ha un riferimento ai metodi, variabili di istanza e variabili statiche. Inoltre se la classe è definita in un metodo ogni variabile locale dell'ambiente esterno usata nella classe interna viene copiata in una variabile di istanza nascosta.

Classe Interna definita in un metodo: Ogni variabile locale dell'ambiente esterno usata nella classe interna viene copiata in una variabile di istanza nascosta, stessa cosa per i parametri espliciti. Dichiarare queste variabili final serve ad assicurare la consistenza.

Classe interne static: Sono definite come membri della classe con lo specificare static, rispetto alle classe interne non static devono essere istanziate necessariamente a partire da un oggetto della classe che lo contiene .

Importazione statica: Grazie a questa si può accedere all'interno di una classe ai nomi statici di una classe specificandone solo il nome. Va usata quando è richiesto un accesso frequente ai membri statici di una o più classi. Sarebbe preferibile importare singolarmente i membri statici di una classe.

Accoppiamento: Una classe A dipende da una classe B se usa esemplari di B (oggetti o metodi), questo viene detto accoppiamento. Se abbiamo un accoppiamento elevato ovvero una forte dipendenza tra più classi andremo incontro a problemi, infatti se una classe viene modificata tutte le classi che dipendono da essa potrebbero necessitare una modifica.

Effetto collaterale nei modificatori: Possono modificare altri oggetti

Pre-Condizioni: Requisiti che devono essere soddisfatti perché un metodo possa essere invocato, se le precondizioni di un metodo non vengono soddisfatte il metodo potrebbe avere un comportamento errato. Le pre-condizioni devono essere pubblicate nella documentazione. L'uso tipico è per restringere il campo dei parametri di un metodo o per richiedere che un metodo venga chiamato solo quando l'oggetto si trova in uno stato appropriato.

Asserzioni: Sono delle dichiarazioni che si presume debbano essere sempre vere nel codice. Compromesso tra il non fare nulla e lanciare le eccezioni che sono pesanti. Per inserire un asserzione scriviamo assert e la condizione da rispettare (Invoca un assert Error).

Post-Condizioni: Condizioni che devono essere soddisfatte al termine dell'esecuzione del metodo, ne abbiamo di due tipi , nel primo il valore restituito deve essere computato correttamente, nel secondo al termine dell'esecuzione del metodo l'oggetto con cui il metodo è invocato deve trovarsi in un determinato stato. In breve una post-condizione è una condizione che un garantisce quando il metodo termina.

Invariante: Condizione che vale sempre

Invariante di una classe: Una condizione che rimarrà vera riguardo i membri di una classe nelle varie istanze.

Programmazioni per contratti: Quando il programmatore rispetta le condizioni poste nella documentazione. Inoltre garantisce che se sono rispettate le pre-condizioni lo sono anche le invarianti e le post-condizioni.

Lezioni cenni progettazione orientata agli oggetti
