**Appunti Programmazione Object Oriented**

**Concetti base della OOP:**

* **Incapsulamento dei dati:** Il processo di nascondere i dettagli di definizione di oggetti, solo le interfacce sono visibili dall’esterno, in questo modo un programmatore conosce il comportamento dell’oggetto ma non la sua struttura interna**.**
* **Ereditarietà:** Gli oggetti sono definiti in una gerarchia ed ereditano dall’immediato padre caratteristiche comuni, che possono essere specializzate.
* **Astrazione:** Il meccanismo con cui si specificano le caratteristiche peculiari di un oggetto che lo differenzia da altri.
* **Polimorfismo:** Possibilità di eseguire metodi con la stessa firma ma con implementazioni differenti (per esempio tramite l’utilizzo di un’interfaccia, la scelta del metodo avviene durante la **fase di esecuzione**).

**Altri concetti:**

* **Information Hiding:** Una tecnica per lo sviluppo del software in cui le interfacce dei moduli mostrano il meno possibile del loro funzionamento interno e gli altri moduli sono prevenuti dall’usare informazioni del modulo che non sono definite nell’interfaccia.

**Concetto di:**

* **Oggetto:** è un entità di una classe dotata di tre proprietà distinte:
  + **Stato:** informazioni conservare nell’oggetto (Es. valori delle variabili d’istanza).
  + **Comportamento:** che viene definito tramite una serie di operazioni che l’oggetto può effettuare, tale operazioni vengono dette **metodi**.
  + **Identità:** Ogni oggetto ha un propria identità e viene indentificato tramite un identificatore univoco.
* **Classe:** ogni oggetto appartiene per definizione ad una classe ed è un istanza di essa, la classe definisce per l’oggetto:
  + **Comportamenti possibili (metodi);**
  + **Stati possibili (variabili di istanza o campi);**

**Classificazione dei metodi:** all’interno di un programma orientato agli oggetti, i metodi possono essere classificati in due categorie principali:

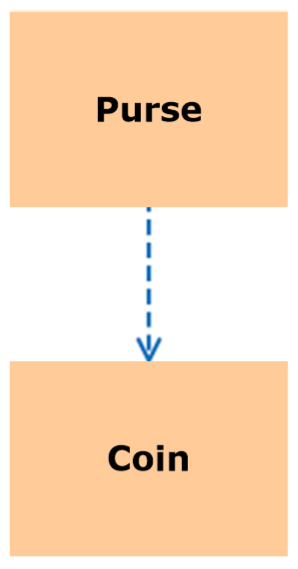
* **Metodi di accesso (get) :**  che non modificano lo stato del proprio **parametro implicito** (sta a sinistra).
* **Metodi modificatori (set) :** che modicano lo stato del proprio parametro implicito.

**Concetto di garbage:** quando un oggetto non ha più un riferimento non può essere più referenziato e quindi utilizzato, pertanto diventa inutile **(garbage)**, Java effettua automaticamente dei cicli di recupero della spazzatura in modo da liberare la memoria occupata dagli oggetti ormai diventati **garbage**.

**Concetto di:**

* **Overloading:** Possibilità di creare metodi con lo stesso nome ma con firma diversa (**fase di compilazione**)
* **Overwriting:** Possibilità di riscrivere i metodi di una superclasse all’interno di un sottoclasse (esempio metodo **toString, Clone**).

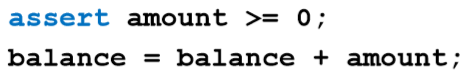
**Criteri utili per analizzare la qualità di una interfaccia pubblica di una classe:**

* **Coesione:** una classe deve rappresentare un singolo concetto, per tanto una classe è coesa se l’interfaccia contiene solo operazioni tipiche del concetto che la classe realizza
* **Accoppiamento:** una classe **A** dipende da una classe **B** se usa esemplari di **B**, in caso di molte classi che dipendono tra loro si ha una situazione di **accoppiamento elevato,** tale situazione può portare ai seguenti problemi:
  + Se una classe venisse modificata tutte le classi dipendenti da essa potrebbero necessitare di una modifica
  + Se si volesse usare una classe in un altro programma bisognerebbe usare anche tutte le classi da cui quella classe dipende.

Concludendo l’accoppiamento va bene ma non ce ne deve essere tanto. L’accoppiamento all’interno dei diagrammi UML si rappresenta nel seguente modo:

**Concetto di effetto collaterale:** con il termine **effetto collaterale**, si intendono tutte quelle modifiche che possono essere osservate al di fuori del metodo, una volta che quest’ultimo viene eseguito. È buona norma ridurre al minimo i metodi con effetti collaterali, perché possono portare a dei **comportamenti inattesi.**

**Pre-condizioni:** con il termine precondizioni si intendono una serie di requisiti che devono essere soddisfatti prima che un metodo venga invocato, tali **precondizioni** vengono utilizzate principalmente per:

* Restringere il campo di parametri di un metodo
* Per richiedere che un metodo venga invocato solo quando l’oggetto si trova in un determinato stato

In alternativa alla precondizioni è possibile usare le **asserzioni.**

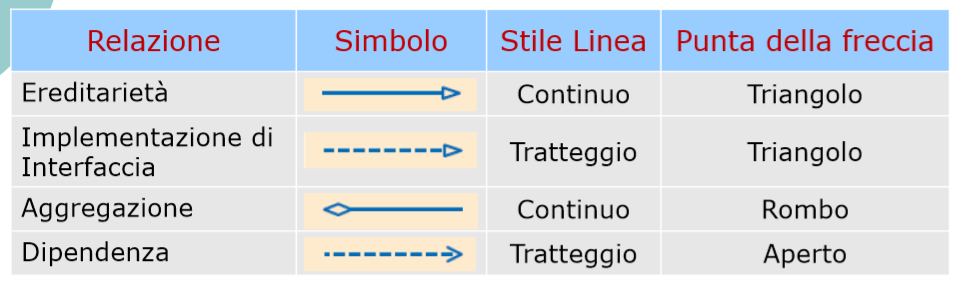
**Post-condizioni:** stesso funzionamento delle precondizioni solo applicato dopo che il metodo è stato eseguito.

**L’interfaccia:** è un tipo di costrutto che dichiara una collezione di metodi elencando le loro firme, ma non fornisce alcuna implementazione di metodi, tale implementazione e delegata alle classi che implementano tale interfaccia, classi che appunto dovranno fornire l’impetrazione di tutti i metodi presenti nell’interfaccia.

**Ereditarietà:**  è un meccanismo che permette di estendere le classi esistenti aggiungendo altri metodi e attributi, la classe più generica è detta **superclasse**, la classe più specifica è detta **sottoclasse.** Il concetto di ereditarietà si rappresenta graficamente nel seguente modo:

**Metodo astratto (abstract):** è un metodo presente nella superclasse di cui tutte le sottoclassi devono obbligatoriamente fornire un’implementazione.

**Notazioni per i diagrammi UML:**

****

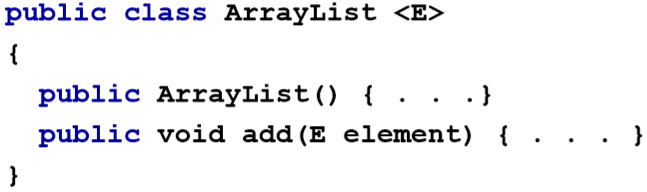
**Procedure di sviluppo per un programmo OO:**

* Collezionare i requisiti;
* Utilizzare le schede **CRC** (responsabilità e collaboratori) per le classi;
* Utilizzare diagrammi **UML** per registrare le relazioni tra classi;
* Utilizzare **JavaDoc** per documentare il comportamento dei metodi;
* Implementare il programma;

**Correzione degli errori:** all’interno dei linguaggi OO esistono vari metodi per la gestione degli errori:

* **Program Trace:** Messaggi che mostrano l’andamento del programma;
* **Logging:** stesso funzionamento del program trace, con il vantaggio che possono essere zittiti un volta terminata la fase di correzione degli errori.
* **Asserzioni:** se un asserzione non è valida il programma si interrompe con l’errore **AssertError.**
* **Debbuger:** programma sviluppato appositamente per scovare errori all’interno del codice. Fornisce la possibilità di visualizzare lo stato delle variabili durante l’esecuzione passo passo.

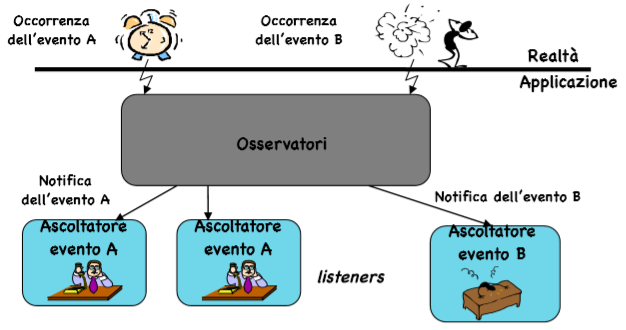
**Programmazione generica:** è una tecnica di programmazione che permette di creare algoritmi e strutture dati che possono essere utilizzate con tipi diversi di dati, tale tipo di programmazione si può realizzare con:

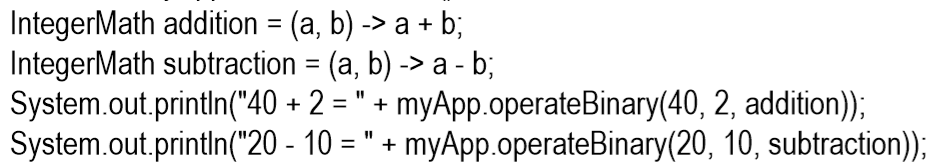
* **Ereditarietà**
* **Variabili di tipo:** variabili a cui si può assegnare un tipo non primitivo e che possono essere usate come tipi nelle dichiarazioni.

**Categorie di eccezioni:**

* **Eccezioni non controllate:** Eccezioni dovute ad errori che il programmatore può evitare correggendo il programma (principalmente errori di sintassi), questo tipo di eccezioni vengono lanciate senza essere controllate.
* **Eccezioni controllate:** Dovute a circostanze che il programmatore non può evitare, questa tipologia di eccezioni si divide, questo tipo di eccezioni viene controllato tramite un **try catch**, in questo caso il compilatore sa che operazioni eseguire nel caso in cui si verifichi quella determinata eccezione.

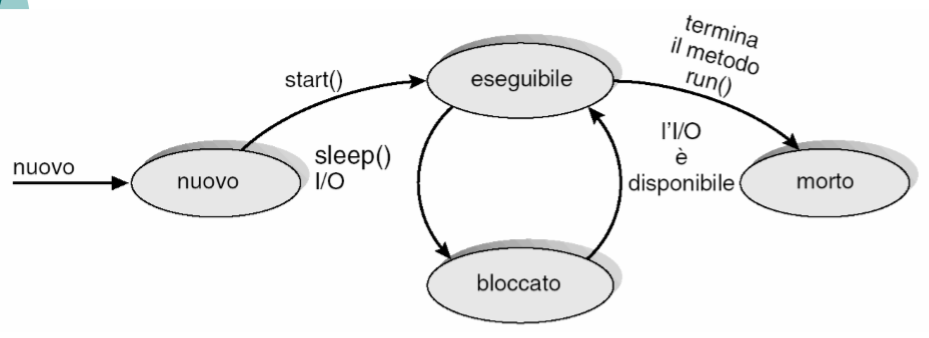
**Eventi:**

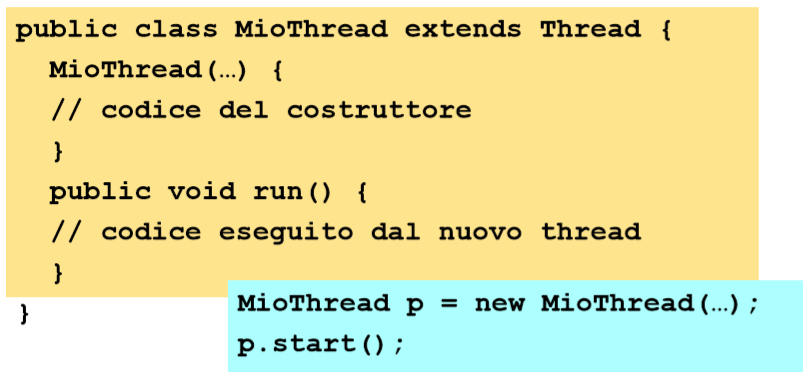
* **Ricevitore (listener):** riceve una notifica quando un evento accade, i suoi metodi descrivono le azioni da eseguire quando si verificano gli eventi, una programma sceglie quali eventi trattare in base ai vari tipi di listener che vengono in stanziati.
* **Sorgente (source):** La componente del programma che genere l’evento e si occupa di notificare a tutti i ricevitori che tale evento è accaduto.

**Espressioni Lambda:** permettono di descrivere un metodo nel punto in cui viene utilizzato, esempio:

**Thread:** un thread è una versione leggera di un processo, utilizzato all’interno di un programma per diversi scopi:

* Migliorare l’interazione con il programma
* Per simulare attività simultanee
* Per sfruttare sistemi multiprocessore

In java i thread sono associati ad istanze della classe **java.lang.Thread**. Gli oggetti istanza di tale classe svolgono la funzione di interfaccia verso la JVM che è l’unica capace di creare effettivamente nuovi thread.

L’esecuzione parallela di più thread all’interno dello stesso programma può portare a situazioni di deadlock, per evitare tali situazioni java mette a disposizione dei lock(li abbiamo visti in sistemi), tramite i metodi **synchronized** che possono essere anche per sincronizzare variabili. E gli oggetti di tipo **Lock**.

**Definizione di Stringa:** Una variabile di tipo stringa è un riferimento ad un oggetto di tipo stringa che modella una sequenza di caratteri

**Categorie di variabili**

1) Variabili di istanza:

* Appartengono all’oggetto
* Esistono finché l’oggetto esiste
* Hanno un valore iniziale di default

2) Variabili locali

* Appartengono al metodo
* Vengono create all’attivazione del metodo e cessano di esistere con esso
* Non hanno valore iniziale se non inizializzate

3) Parametri formali

* Appartengono al metodo
* Vengono create all’attivazione del metodo e cessano di esistere con esso
* Valore iniziale è il valore del parametro reale al momento dell’invocazione

**Visibilità delle variabili**

Campo di visibilità di una variabile (scope): parte del programma in cui si può fare riferimento alla variabile mediante il suo nome. Campo di visibilità di una variabile locale: dalla sua dichiarazione alla fine del blocco. Nell’ambito di visibilità di una variabile locale non è possibile definirne un’altra avente lo stesso nome (nomi non si possono ridefinire in blocchi annidati). I campi di visibilità di una variabile locale e di una variabile di istanza possono sovrapporsi. La variabile locale oscura la variabile di istanza con lo stesso nome. Se in un metodo si vuole fare riferimento ad una variabile di istanza che ha lo stesso nome di una variabile locale allora occorre usare il riferimento this.

**Le interfacce differiscono con l’ereditarietà perché:**

Un’interfaccia non è una classe, non ha né uno stato né un comportamento, ed è un elenco di metodi Una sottoclasse invece è una classe tutti gli effetti, ha uno stato, un comportamento che sono ereditati dalla superclasse.