CLASSI

DEFINIZIONE DI CLASSE

- Fabbrica di oggetti (istanze)
- Può contenere oggetti o metodi statici
- Sintassi:

```
tipoAccesso class nomeClasse {
    costruttori
    metodi
    variabili (o campi) di istanza
}
```

DEFINIZIONE DI METODI

METODO DI ISTANZA

- Funzione presente in una classe
- Firma = intestazione metodo

```
public tipoVariabileRestituito nomeMetodo(tipoParametro nomeParametro)
```

- Metodo d'**accesso**
  - restituisce informazioni
  - NON modifica l'oggetto
- Metodo **modificatore**
  - altera lo stato di un oggetto
- Metodo **predicativo**
  - restituisce un valore booleano
  - di solito iniziano con "is" o "has"

METODO STATICO

- Non agisce su oggetti
  - accetta solo parametri espliciti
  - viene chiamato sulla classe

```
public static nomeTipo nomeMetodo(parametri) // definizione metodo statico
NomeClasse.nomeMetodo(parametri) // chiamare metodo statico
```

COSTRUTTORI

- Sintassi:

```
tipoAccesso NomeClasse(parametri) {
    // realizzazione del costruttore
}
```

- Obiettivo
  - inizializzare oggetto di una classe
- Costruttore di default
  - **inizializza** automaticamente tutte le variabili di istanza
    - numerico -> 0
    - boolean -> false
    - oggetti -> null

OVERLOADING

SOVRACCARICO DEL COSTRUTTORE

- Una classe può avere **più costruttori** con lo **stesso nome**
- Il compilatore decide quale costruttore invocare
  - basandosi sul numero e sul tipo dei parametri forniti nell'invocazione

SOVRACCARICO DEI METODI

- È possibile creare **più metodi** con lo **stesso nome**
  - varia tipo di parametri espliciti accettati

VARIABILI DI ISTANZA (O DI ESEMPLARE)

- Sintassi:

```
tipoAccesso TipoVariabile nomeVariabile;
```

- Memorizzano lo stato di un oggetto

INCAPSULAMENTO

- È buona pratica impostare come private le variabili interne
  - e inserire un metodo getNomeVariabile() per accedervi
- Vantaggi:
  - impedisce accesso incontrollato allo stato di un oggetto

IL RIFERIMENTO NULL

- Permette di inizializzare una variabile con riferimento a nessun oggetto valido
  - non si possono invocare metodi (**NullPointerException**)

COLLAUDO

COME VERIFICARE CHE UNA CLASSE FUNZIONI FUNZIONI?

- Definisco una classe eseguibile
  - creo oggetti con la classe da collaudare
  - provo tutti i metodi della classe da collaudare

PROGRAMMA CON PIÙ CLASSI

- Ciascuna classe in un file diverso
  - tutti nella **stessa cartella**
  - ogni file ha il nome della classe
- Tutte le classi in un unico file
  - solo **UNA** classe pubblica (contenente metodo main)
  - deve avere il nome della classe pubblica

COMMENTI DI DOCUMENTAZIONE

- Sintassi

```
/**
    Descrizione metodo
    @param nomeParametro descrizione
    @return descrizione
*/
public void myMethod(parametri) {}
```

- Crea documentazione

```
javadoc NomeClasse.java -d ./output-directory
```

PACKAGE

- Classe della libreria standard -> raccolte in pacchetti
  - organizzate per argomento o finalità
- Pacchetto `java.lang` importato di default
- Se vengono importate + classi con lo stesso nome: errore di **riferimento ambiguo**

```
import nomePacchetto.nomeClasse;
import nomePacchetto.*; // importa tutte le classi di un pacchetto
```

Documentazione

ALTERNATIVA ALL'IMPORTANTE

```
java.math.BigInteger a = new java.math.BigInteger("123456789");
```

USO DEL PARAMETRO IMPLICITO

- `this` indica l'oggetto attuale
  - viene aggiunto automaticamente all'interno dei metodi se non indicato

MEMBRI DI CLASSE "STATICI"

- Classi di utilità
  - non servono per creare oggetti
  - contengono costanti e metodi statici

VARIABILI STATICHE

- Sono condivise tra tutti gli oggetti
- È bene inizializzarle quando si dichiarano
- Consigliato che siano private

```
private static type nomeVariabile = valoreIniziale;
```

COSTANTI STATICHE

- Valori pubblici e accessibili

```
public static final type NOME_COSTANTE = valoreIniziale;
```

VISIBILITÀ SOVRAPPOSTE

- Cosa succede quando si definisce variabile locale e di istanza/statica con lo stesso nome?
  - NO errore compilazione
  - prevale variabile locale (effetto di **shadowing**)
  - se voglio utilizzare variabile di istanza/statica, devo aggiungere prefisso `this.`

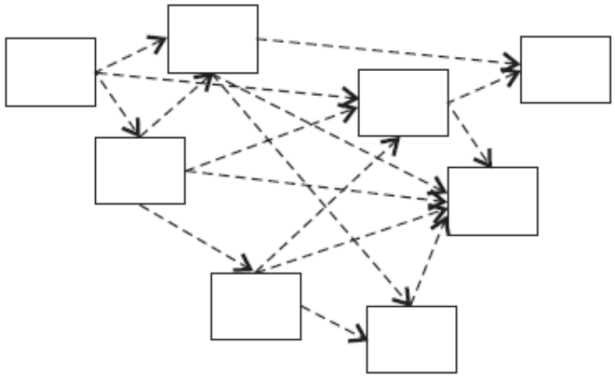
STRUTTURA PROGETTO

COESIONE

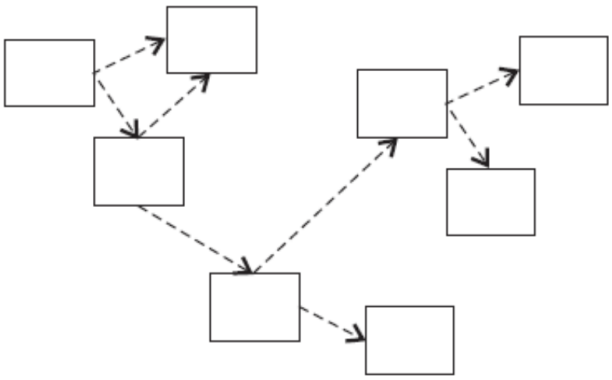
- Una classe dovrebbe rappresentare un singolo concetto
  - metodi e costanti strettamente correlati al concetto rappresentato

ACCOPPIAMENTO

- Una classe dipende da un'altra classe
- Relazione di dipendenza rappresentata con UML (Unified Modeling Language)
  - rettangoli = classi
  - freccia con linea tratteggiata = dipendenza
- Quando modifichi una classe, potresti dover modificare anche le classi su cui dipendono



Elevato accoppiamento



Basso accoppiamento

Good practice

Ottimizzare il codice con basso Accoppiamento

SIDE EFFECT (EFFETTO COLLATERALE)

- Qualsiasi comportamento **osservabile** al di fuori del metodo stesso
- Esempi di effetto collaterale
  - metodo che modifica parametro implicito
  - metodo che modifica parametro esplicito
  - visualizzazione di dati in uscita