Sistemi ad oggetti

- Le classi incapsulano dati ed operazioni che manipolano i dati
- Gli oggetti possiedono uno stato e reagiscono eseguendo del codice quando ricevono una invocazione
- La comunicazione tra componenti (classi e oggetti) avviene tramite passaggio di messaggi (ovvero invocazioni di operazioni)

Qualità OO

- Semantica: cosa risolve una classe?
  - Scegliere il nome della classe in modo che esso sia significativo
  - Descrivere il contesto per cui la classe è implementata
  - Il lavoro di progettazione, modifica o riuso è facilitato se vi è corrispondenza tra ciò che la classe implementa e quello che il nome suggerisce
- Correttezza: quando una classe smette di comportarsi bene?
  - Una classe funziona correttamente solo sotto certe condizioni
  - Documentare tali condizioni, esplicitando dipendenze da variabili, (range di) valori dei parametri in ingresso e uscita, flusso di controllo
  - Quando si riusa la classe bisogna verificare se le ipotesi su cui l'implementazione si basa sono soddisfatte
- Responsabilità: non sviluppare "God classes"
  - La granularità per il riuso è la classe, classi piccole sono più riusabili
  - Separare compiti (o responsabilità) diversi in classi distinte permette di ottenere classi aventi alta ***coesione***
  - Alta coesione logica tra sotto-compiti implica maggiore facilità e probabilità di comprensione, riuso, evoluzione

Qualità OO

- Information hiding: quanto si conosce di una classe?
  - Ciò che non contribuisce alle caratteristiche essenziali di una classe deve essere nascosto (gli attributi ed alcuni metodi sono private). Sono i metodi che forniscono ciò che si vuol far conoscere all'esterno
  - Avere caratteristiche nascoste limita la propagazione dei cambiamenti all'esterno
- Coesione: quanti compiti ha una classe?
  - Indica la connessione degli elementi all'interno di un modulo
  - Una classe è coesa se implementa un solo piccolo compito, quindi tutti i suoi elementi (metodi) contribuiscono a realizzare il ***singolo compito*** (lo stesso vale per la coesione di un metodo)
  - Classi poco coese (usate ad es. per visualizzare, trasformare e registrare dati) sono difficili da comprendere e da riusare

Qualità OO

- Parlare solo con gli amici (Law of Demeter)
  - Una classe dovrebbe avere una conoscenza limitata delle altre classi: solo di quelle strettamente legate ad essa
  - L'accoppiamento lasco tra classi facilita la loro comprensione, sostituzione, riuso ed evoluzione
  - Qui l'accoppiamento è rivelato dal numero di classi usate
- Interazioni (***coupling***): quanto una classe dipende da altre classi?
  - Riusare o evolvere una classe risulta difficile se essa implementa molte invocazioni a metodi di altre classi: codice più difficile da comprendere e modificare
  - Qui l'accoppiamento è rivelato: dal numero di classi usate, dal numero di invocazioni a metodi diversi di altre classi, e dal numero di invocazioni a metodi (anche allo stesso metodo)
  - Eliminare una classe o evolvere la sua interfaccia implica modificare tutte le classi che la usano: tanto lavoro quando i suoi metodi sono invocati da molte classi

Rimedi per le classi

- Se una classe è poco coesa
  - Dividerla in più classi
- Se una classe invoca molte altre classi
  - Probabilmente fa molte cose, dividerla in più classi
- Se l'accoppiamento tra due classi è molto alto
  - Probabilmente hanno lo stesso obiettivo, fondere le due classi

Progettazione OO

- Partendo dalla descrizione dei requisiti
  - Definire il contesto del prodotto tramite casi d'uso
  - Progettare l'architettura (ispirandosi ad uno stile)
  - Identificare i sottosistemi principali
  - Per ogni sottosistema identificare classi ed oggetti
  - Sviluppare i diagrammi di ereditarietà, interazione, etc.
  - Specificare le interfacce delle classi
  - Considerare quali algoritmi sono necessari
  - Iterare!!!! (Raffinando ad ogni passo)