Programmazione Object-Oriented

- Per realizzare sistemi software di grandi dimensioni occorre avere supporti adeguati
- Il linguaggio di programmazione deve consentire di
  - Esprimere le attività da svolgere
  - Dare una struttura adeguata (modulare, per componenti) al software
    - Rendendo facile la scelta e la descrizione della struttura, ma anche imponendo vincoli per ottenere modularità
  - Coordinare le attività di sviluppo
- Costruzione del software per componenti
  - Ogni componente è un modulo che può essere esteso
  - Funzioni e procedure non possono essere intese come componenti
  - I file che contengono funzioni non sono componenti

Dimensioni dei sistemi

- Per sistemi di piccole dimensioni (size < 1000 LOC)
  - L'elemento fondamentale è l'algoritmo
  - I programmi sono strutturati (strutture dati e di controllo)
  - Funzioni e procedure sono usate per avere l'astrazione di istruzioni complesse
- Per sistemi di medie e grandi dimensioni
  - La decomposizione in funzioni non basta, poiché:
    - Non c'è accoppiamento tra dati e funzioni
    - Il passaggio di parametri è complesso
    - Esistono tante variabili globali
    - L'accoppiamento (stretto o lasco) tra funzioni non è evidente dalla struttura
    - Il sistema diventa complesso e difficile da comprendere, quindi gli errori sono difficili da trovare

Sistemi ad oggetti

- Concetti chiave
  - Tipi di dato astratto (ADT)
    - Es. il numero complesso, il libro, il conto corrente
  - Stato
  - Modello client-server
  - Oggetti e classi

Astrazione sui dati

- Dati ed operazioni sono raggruppati in un componente
  - Struttura dati nascosta (incapsulamento e information hiding)
  - Operazioni che agiscono sui dati
- Vantaggi
  - Accoppiamento tra dati e funzioni
    - Modularità
    - Limita il numero ed il tipo di funzioni che agiscono sui dati
  - Integrità dei dati
    - Solo le funzioni autorizzate agiscono sui dati
    - Se un errore è presente sui dati, la ricerca dell'errore è localizzata solo sulle funzioni che agiscono su quei dati
  - Creazione di componenti autonomi (o quanto più autonomi)
  - Astrazione, ci allontaniamo dal linguaggio di programmazione
  - Entità simili (con valori diversi) possono essere considerate dello stesso tipo

Esempio di sistema classico

```
void main() {
  …
  int balance;
  …
  if (balance > amount)
     balance = balance - amount;
  …
}
```

- Non vi è astrazione
  - Gestione a basso livello delle variabili
  - Non abbiamo nel codice il concetto di balance
  - Non possiamo riusare il concetto
  - Riusare il codice significa parametrizzare e inserire variabili globali

Esempio di classe Java

```
// Questa classe tiene informazioni sul conto corrente
// La classe e' public quindi accessibile dall'esterno
public class Account {  // classe di nome Account
   private int balance = 0; // attributo non accessibile dall'esterno
   // il metodo deposito non restituisce nulla
   public void deposito(int amount) {  // metodo accessibile
      balance = balance + amount;  // aggiorno il valore di balance
   }
   // il metodo check restituisce un boolean
   public boolean check(int amount) {
      if (amount > 0) return true;
      return false;
   }
}
```


Classe Account

- Account raggruppa
  - Il dato: balance
  - I metodi: deposito e check
- Account è un componente
  - Molto più che un insieme di dati o di funzioni
- Account è un nuovo tipo (non primitivo), può essere inteso come una nuova categoria
- In un sistema ad oggetti avremo tanti nuovi tipi
  - Durante lo sviluppo di un sistema ad oggetti ci chiederemo
    - Cosa mettiamo dentro ciascun tipo
    - Dove mettiamo ad es. le variabili costo, numero, etc.

Sintassi Java

- Modificatori
  - public e private indicano cosa è accessibile o non-accessibile, rispettivamente dall'esterno della classe stessa
  - È buona norma che i dati (o attributi) siano private, quindi nascosti alle altre classi
  - Le operazioni (o metodi) sono in genere public, quindi accessibili dalle altre classi
- Valori di ritorno dei metodi
  - E' possibile dichiarare e restituire un valore di ritorno (es. int, boolean) oppure nulla (allora si usa void)
  - Si restituisce un valore al chiamante tramite return
- Valori iniziali
  - Nell'esempio, l'attributo balance è dichiarato int e riceve subito il valore zero

Compilazione

- Classi Java
  - Ciascuna classe Java va conservata in un file a sé avente nome uguale al nome della classe ed estensione .java (es. nome file Account.java)
  - Il codice sorgente (Account.java) va trasformato in codice eseguibile tramite il compilatore javac (es. javac Account.java)
  - Il compilatore genera un file eseguibile (es. Account.class)
  - L'esecuzione dei .class avviene tramite la JVM
    - La JVM deve ricevere come argomento una classe che contiene il metodo main
- JVM
  - java.sun.com
  - J2SE

Esecuzione

- La classe TestAccount
  - Contiene il metodo main
    - Da questo metodo inizia l'esecuzione del codice
  - Può essere eseguita dalla JVM (es. java TestAccount) dopo essere stata compilata

```
// Classe per il test della classe Account
public class TestAccount {
    // il metodo main deve essere presente per poterla eseguire
    public static void main(String[] args) {
        Account unAccount = new Account();
    }
}
```


Classe Account vers. 2

```
public class Account {
   private int balance = 0;

   public void deposito(int amount) {
      if (check(amount))       // chiamo il metodo check della stessa classe
         balance = balance + amount;
   }

   public boolean check(int amount) {
      if (amount > 0) return true;
      return false;
   }
}
```