Il paradigma a oggetti

1 Il paradigma a oggetti

- I concetti fondamentali:
  - oggetto
  - astrazione
  - classe
  - incapsulamento
  - ereditarietà
  - polimorfismo - late binding
  - delegazione

Oggetti

- Sono gli elementi di base del paradigma, e corrispondono a entità (non necessariamente "fisiche") del dominio applicativo
  - Esempi (in un'aula universitaria): le sedie, gli studenti che le occupano, il professore che tiene la lezione, il corso seguito dagli studenti
- *Un oggetto è un individuo sostanziale che possiede un identità e un insieme di proprietà, che ne rappresentano lo stato e il comportamento*
- Ogni oggetto è caratterizzato da:
  - una identità (OID, *Object IDentifier*) che gli viene associata all'atto della creazione, non può essere modificata ed è indipendente dallo stato corrente dell'oggetto
  - uno stato definito come l'insieme dei valori assunti a un certo istante da un insieme di attributi
  - un comportamento definito da un insieme di operazioni
- Poiché un oggetto può anche includere riferimenti ad altri oggetti, risulta possibile creare *oggetti complessi*.

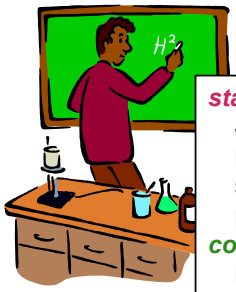
Oggetti (2)

stato:
esame di Analisi
svolto il 3.3.2001
voto 30/30
comportamento:
stampa verbale
trasmetti in segreteria

stato:
Mark
nato il 12.12.19
celibe
matricola XXX
comportamento:
mangia
dorme
si iscrive a cors
sostiene esami

stato:
Ann
nata il 22.2.198
nubile
matricola YYY
comportamento:
mangia
dorme
si iscrive a cors
sostiene esami

stato:
Carl
nato il 7.7.1979
sposato
matricola ZZZ
comportamento:
mangia
dorme
si iscrive a corso
sostiene esami



stato:
John
nato il 2.2.1968
sposato
prof. associato
comportamento:
mangia
dorme
tiene corso
tiene esami

gli attributi esistono nel mondo reale, gli identificatori no

Operazioni e interfaccia

- Ogni operazione, dichiarata da un oggetto, specifica il nome dell'operazione, gli oggetti che prende come parametri e il valore restituito (*signature*)
  - L'oggetto su cui l'operazione opera è definito implicitamente
- L'insieme di tutte le signature delle operazioni di un oggetto sono dette *interfaccia* dell'oggetto
  - L'interfaccia specifica l'insieme completo di tutte le richieste che possono essere inviate all'oggetto

Abstract Data Type

Tipo di dati astratto

"rappresentazione di oggetti simili" perché non si riferisce direttamente a una specifica implementazione fisica, ma piuttosto a un modello concettuale che descrive un insieme di oggetti che condividono caratteristiche e comportamenti comuni.

La struttura per i dati determina come sono organizzati i dati all'interno dell'oggetto. L'interfaccia definisce cosa puoi fare con quell'oggetto, senza preoccuparti dei dettagli implementativi.

- E' una rappresentazione di un insieme di oggetti "simili", caratterizzato da una struttura per i dati e da un'interfaccia che definisce quali sono le operazioni associate agli oggetti, ovvero l'insieme dei servizi implementati
- Un tipo è sottotipo di un supertipo se la sua interfaccia contiene quella del supertipo
  - Un sottotipo eredita l'interfaccia del suo supertipo
  - L'interfaccia non vincola l'implementazione del servizio offerto ovvero il comportamento effettivo
  - Oggetti con la stessa interfaccia possono avere implementazioni completamente diverse

l'ADT è come un contratto: definisce cosa un oggetto deve fare, lasciando aperto come deve farlo.

In pratica, mentre un ADT è una definizione teorica, la classe specifica come queste operazioni funzionano concretamente.

Classe

Abstract Data Type

- Fornisce una realizzazione di un tipo di dati astratto, specifica cioè un'implementazione per i metodi a esso associati
  - Esempi: classe delle sedie, degli studenti, dei professori, dei corsi

Un oggetto è sempre istanza di esattamente una classe

- Tutti gli oggetti di una classe hanno gli stessi attributi e metodi. Esistono metodi di due tipi: quelli che restituiscono astrazioni significative sullo stato dell'oggetto cui sono applicati, e quelli che ne alterano lo stato

1) Questi metodi forniscono informazioni sullo stato interno dell'oggetto, senza modificarlo. Servono a leggere o a calcolare qualcosa basandosi sui valori degli attributi dell'oggetto.
2) Questi metodi modificano gli attributi dell'oggetto, cambiando quindi il suo stato interno. Permettono di aggiornare lo stato dell'oggetto in base a operazioni o input dell'utente.

Classe

ESAME

attributi:
- corso
- data
- voto

operazioni:
- stampa verbale
- trasmetti in segreteria

STUDENTE

attributi:
- nome
- data nascita
- stato civile
- num. matricola

operazioni:
- mangia
- dorme
- si iscrive a corso
- sostiene esami

DOCENTE

attributi:
- nome
- data nascita
- stato civile
- ruolo

operazioni:
- mangia
- dorme
- tiene corso
- tiene esami

Incapsulamento

- Protegge l'oggetto nascondendo lo stato dei dati e l'implementazione delle sue operazioni
- Un oggetto incapsula i dati (attributi) e le procedure (operazioni) che li possono modificare
- *Il principio di incapsulamento sancisce che gli attributi di un oggetto possono essere letti e manipolati solo attraverso l'interfaccia che l'oggetto stesso mette a disposizione*
  - I dettagli dell'implementazione di una classe sono *privati*, cioè manipolabili direttamente solo dai metodi della classe e quindi protetti
  - L'accesso dall'esterno agli attributi della classe avviene attraverso una ristretta *interfaccia pubblica*, costituita da un sottoinsieme dei metodi della classe
  - Un oggetto esegue una operazione quando riceve una richiesta (messaggio) da un oggetto client

Vantaggi dell'incapsulamento

- Per l'utilizzo di una classe è sufficiente conoscerne l'interfaccia pubblica; i dettagli implementativi sono nascosti all'interno. La classe viene quindi vista come una "scatola nera"
- La modifica dell'implementazione di una classe non si ripercuote sull'applicazione, a patto che non ne venga variata l'interfaccia
- Poiché la manipolazione diretta degli attributi della classe avviene esclusivamente tramite i suoi metodi, viene fortemente ridotta la possibilità di commettere errori nella gestione dello stato degli oggetti
- Il debugging delle applicazioni è velocizzato, poiché l'incapsulamento rende più semplice identificare la sorgente di un errore

Operazioni e Metodi

Metodo: È l'implementazione concreta di una specifica operazione.

- Un metodo cattura l'implementazione di una operazione
- I metodi possono essere classificati in:
  - *costruttori,* per costruire oggetti a partire da parametri di ingresso restituendo l'OID dell'oggetto costruito
  - *distruttori,* per cancellare gli oggetti ed eventuali altri oggetti ad essi collegati
  - *accessori,* per restituire informazioni sul contenuto degli oggetti (proprietà derivate) Recuperano informazioni sullo stato di un oggetto, senza modificarlo.
  - *trasformatori,* per modificare lo stato degli oggetti e di eventuali altri oggetti ad essi collegati Modificano lo stato interno di un oggetto o degli oggetti collegati.
- I metodi possono essere:
  - *pubblici* Accessibili da qualsiasi altra classe o oggetto.
  - *protetti* Accessibili solo dalla classe stessa e dalle classi derivate (ereditarietà).
  - *privati* Accessibili solo all'interno della classe in cui sono definiti.

Ereditarietà

- Il meccanismo di ereditarietà permette di basare la definizione e implementazione di una classe su quelle di altre classi.
- E' possibile definire relazioni di specializzazione/ generalizzazione tra classi: la classe generalizzante viene detta *superclasse*, la classe specializzante viene detta *sottoclasse* o *classe derivata*
  - Esempio: le classi studente e professore sono entrambe derivate dalla classe persona
- Ciascuna sottoclasse eredita dalla sua superclasse la struttura ed i comportamenti, ovvero gli attributi, i metodi e l'interfaccia; può però specializzare le caratteristiche ereditate e aggiungere caratteristiche specifiche non presenti nella superclasse

Ereditarietà (2)

PERSONA

attributi:
nome
data nascita
stato civile

operazioni:
mangia
dorme

DOCENTE

attributi:
nome
data nascita
stato civile
ruolo

operazioni:
mangia
dorme
tiene corso
tiene esami

è un

è un

STUDENTE

attributi:
nome
data nascita
stato civile
num. matricola

operazioni:
mangia
dorme
si iscrive a corso
sostiene esami

Ereditarietà (3)

- Si parla di *ereditarietà multipla* quando una sottoclasse può essere derivata contemporaneamente da più superclassi
  - in caso di conflitti tra attributi o metodi ereditati da due superclassi, occorre individuare opportune strategie di risoluzione
- Poiché una classe derivata può essere ulteriormente specializzata, si vengono a formare *gerarchie* di classi, strutturate come alberi in caso di ereditarietà singola e come reticoli in caso di ereditarietà multipla
- Date due classi A e B di cui B è una sottoclasse di A, esiste di fatto la relazione B *is-a* A (B *è un* A)
  - gli oggetti istanze di B possano a tutti gli effetti essere utilizzati al posto di oggetti istanze di A (ad esempio, uno studente è una persona) perché B ha gli stessi metodi di A
  - *Non è vero il contrario* (non è detto che una persona sia uno studente) perché A potrebbe non avere dei metodi di B

Polimorfismo

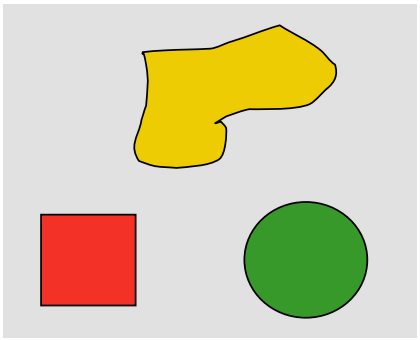
capacità di assumere forme molteplici

- Nel paradigma a oggetti si usa questo termine per alludere alla possibilità di creare metodi con lo stesso nome ma implementazioni differenti
- Tramite il meccanismo di *overload* è possibile definire, all'interno di una stessa classe, più metodi con lo stesso nome ma *signature* (insieme dei parametri) differenti
  - A fronte di un messaggio inviato per invocare il metodo, sarà il sistema a scegliere l'implementazione da considerare, sulla base della struttura del messaggio stesso

Polimorfismo (2)

- Possibilità di ridefinire, all'interno di una sottoclasse, l'implementazione di un metodo ereditato (*override*)

```
class figuraGeometrica
{  // attributi
   int posizioneX; int posizioneY;
   int coloreContorno;
   int coloreRiempimento;
   // metodi
   public:
   void trasla(int shiftX, int shiftY);
   void ruota(int angoloRotazione);
   . . . . . .
}
```



```
class quadrato:figuraGeometrica
{  int lato; int angolo
}
```

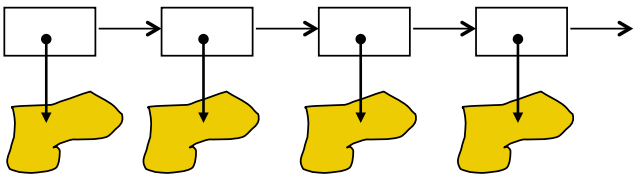
```
class cerchio:figuraGeometrica
{  int raggio;
}
```

ridefinizione di `trasla` e `ruota`

- Early Binding: L'associazione tra la chiamata a un metodo e la sua implementazione viene risolta durante la compilazione.
- Late Binding: L'associazione viene risolta durante l'esecuzione del programma. Ciò consente al programma di decidere quale metodo invocare in base al tipo effettivo dell'oggetto a runtime, supportando il polimorfismo e l'override dei metodi nelle classi derivate.

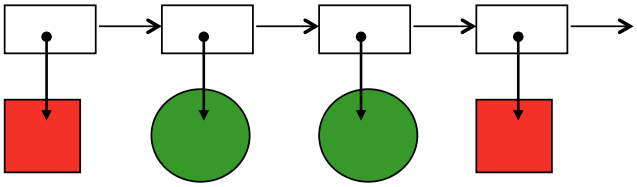
Istanziamento dinamico (*late binding*)

- Il polimorfismo, abbinato all'istanziamento dinamico, permette a ciascun oggetto di rispondere a uno stesso messaggio in modo appropriato a seconda della classe da cui deriva



compile-time

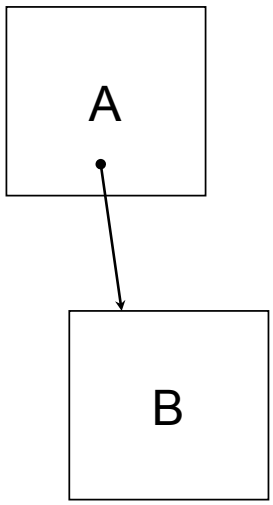
run-time



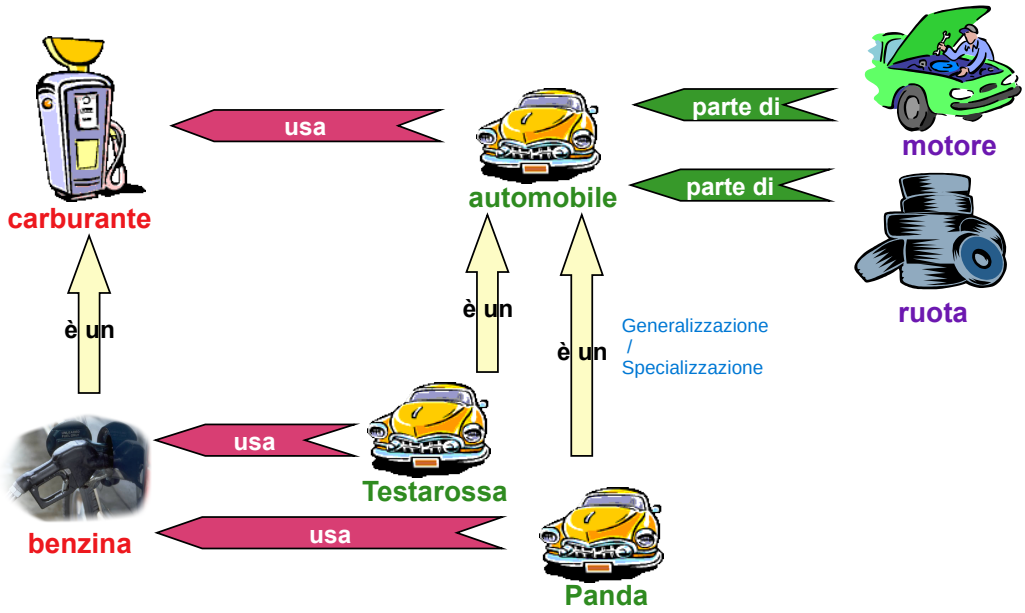
fino a run-time non si è vincolati a una particolare implementazione

Delegazione

- Si parla di delegazione quando un oggetto A contiene al suo interno un riferimento a un altro oggetto B, cosicché A (che risulta essere in questo caso un *oggetto complesso*) può delegare alcune funzioni alla classe ~~a cui appartiene~~ B
  - Esempio: Dovendo definire una classe persona, gli attributi nome, cognome e indirizzo saranno dichiarati come puntatori a oggetti di classe stringa, delegando così a quest'ultima classe le operazioni di manipolazione
- La delegazione costituisce il meccanismo fondamentale per implementare associazioni tra classi
  - Esempio: per rappresentare l'associazione di inclusione tra un aeroplano e il suo motore, si includerà in ogni oggetto di classe aeroplano un puntatore a un oggetto di classe motore



Un esempio



2 Lo sviluppo di sistemi a oggetti

Imparare una nuova tecnica di progettazione è molto più difficile che imparare un nuovo linguaggio, poiché richiede di modificare sostanzialmente il nostro modo di pensare

- Il bisogno di sviluppare e mantenere sistemi di grandi dimensioni e complessi in ambienti dinamici crea un forte interesse in nuovi approcci al problema del design
- L'obiettivo principale dell'approccio orientato agli oggetti (OO, *object-oriented*) per lo sviluppo di sistemi software è migliorare la produttività aumentando l'estendibilità e la riusabilità del software e controllando la complessità e i costi della manutenzione

I sistemi OO sono progettati per essere facilmente ampliati con nuove funzionalità senza dover modificare radicalmente il codice esistente.

Gli oggetti e le classi possono essere progettati una sola volta e riutilizzati in progetti futuri.

La modularità dell'approccio OO (organizzare il software in unità indipendenti e autonome) semplifica l'identificazione e la risoluzione dei problemi.

paradigma funzionale nel contesto dello sviluppo software

Dall'approccio funzionale…

Nel paradigma funzionale, l'analisi si concentra su ciò che il sistema deve fare (le funzioni o i task).

- La *decomposizione funzionale* è un'analisi di tipo top-down tradizionalmente impiegata nel paradigma procedurale, basata sui concetti di procedura e flusso di dati
  - La domanda fondamentale è: *cosa fa il sistema, qual è la sua funzione?*
  - Ad alto livello di astrazione, il sistema viene caratterizzato tramite *un'unica funzionalità* Scendendo, lo si scompone in funzioni più specifiche, fino a ottenere i dettagli implementativi.
  - I blocchi di base dell'applicazione sono i task (*compiti*), che durante l'implementazione daranno luogo a procedure, e sono legati alla specifica soluzione proposta
- Principali problemi:
  - Nessun modello unificante per integrare le diverse fasi: c'è una forte discrepanza tra concetto di flusso di dati utilizzato nell'analisi e concetto di gerarchia di compiti utilizzato nella progettazione
  - Mancanza di iterazione nella progettazione: si adotta il *modello a cascata*, in cui le attività sono viste come una progressione lineare Non c'è spazio per tornare indietro e rivedere decisioni prese in fasi precedenti.
  - Mancanza di estendibilità: non si considerano le possibili evoluzioni del sistema
  - Poca attenzione al problema della riusabilità: ogni sistema viene ricostruito a partire da zero, per cui i costi di manutenzione sono alti
  - La progettazione dei dati viene trascurata, poiché le strutture dati sono determinate dalle strutture procedurali L'attenzione è tutta sul flusso delle operazioni (le procedure)

L'analisi si basa sul concetto di flusso di dati: come le informazioni passano da una parte all'altra del sistema.
La progettazione, invece, si concentra sulla gerarchia dei task, ossia come le funzioni sono organizzate e collegate.
Questo crea una discrepanza tra le fasi di sviluppo, perché l'analisi e la progettazione usano approcci e rappresentazioni diversi.

Non integra bene analisi e progettazione. Non supporta la gestione di cambiamenti o evoluzioni del sistema.
Non favorisce il riutilizzo del codice. Spesso trascura la progettazione dei dati.

paradigma ad oggetti nel contesto dello sviluppo software

...all'approccio a oggetti

Obiettivo: Identificare cosa il sistema deve fare.

- ANALISI: va dall'inizio del progetto fino all'analisi delle specifiche utente e allo studio di fattibilità (*cosa* il sistema deve fare)

Obiettivo: Progettare come il sistema deve fare ciò che è stato stabilito nella fase di analisi.

- DESIGN: progettazione logica e fisica del sistema (*come* lo deve fare)
- IMPLEMENTAZIONE: scrittura del codice, test di verifica, validazione, manutenzione
  - I confini tra le fasi non sono più distinti, infatti il centro di interesse è lo stesso: gli oggetti e le loro interrelazioni
  - Il processo di sviluppo OO è iterativo: si adotta il *modello a fontana*, in cui lo sviluppo raggiunge un alto livello per poi ritornare a un livello precedente e risalire di nuovo Lo sviluppo può tornare a una fase precedente per apportare correzioni. Successivamente, si procede nuovamente verso la fase successiva.
  - L'ereditarietà permette di aggiungere nuove caratteristiche a un sistema riducendo i costi di manutenzione (estendibilità), e di costruire nuove funzionalità a partire dall'esistente (riusabilità) riscrivendo solo quella parte di codice inadeguato e solo per gli oggetti che ne hanno bisogno

Benefici dell'approccio a oggetti

- La decomposizione è orientata alla modellazione
  - I blocchi di base dell'applicazione sono entità che interagiscono, modellate come classi di oggetti, e sono legate alla formulazione originale del problema formando una rappresentazione coerente del problema.
  - I risultati dell'analisi non sono un semplice input del design, ma ne sono parte integrante: analisi e design lavorano insieme per sviluppare un modello del dominio del problema
- Il progetto dettagliato è rimandato nel tempo e nascosto all'interno di ciascuna classe
  - Algoritmi e strutture dati non sono più "congelati" a un alto livello del progetto
  - Si ha più flessibilità, poiché un cambiamento nell'implementazione non implica variazioni consistenti alla struttura del sistema

Il sistema è suddiviso in classi, che rappresentano entità reali o concettuali legate al problema da risolvere.

Il risultato dell'analisi diventa parte integrante del design, contribuendo a creare un modello completo del dominio del problema.

Nei sistemi OO, i dettagli di implementazione sono nascosti all'interno delle classi. Questo concetto è noto come incapsulamento, e separa ciò che un oggetto fa (la sua interfaccia) da come lo fa (la sua implementazione).

Non è necessario definire tutti i dettagli di progettazione nelle fasi iniziali. Si può posticipare la decisione su come implementare un particolare algoritmo o struttura dati, lasciando spazio per esplorare diverse soluzioni.

Poiché i dettagli di implementazione sono nascosti all'interno delle classi, un cambiamento al loro interno non influisce sulla struttura generale del sistema.

Benefici dell'approccio a oggetti

- I sistemi sviluppati a oggetti risultano più stabili nel tempo di quelli progettati per decomposizione funzionale perché l'OO si basa sulla modellazione degli oggetti del dominio applicativo
  - Le caratteristiche dei domini applicativi variano più lentamente nel tempo rispetto alle funzionalità richieste ai sistemi
- La produttività è alta
  - Fasi diverse dell'analisi dei requisiti e del ciclo di vita possono essere svolte contemporaneamente
- C'è la possibilità di sviluppare rapidamente prototipi che possono risultare di valido ~~ausilio~~ aiuto per la certificazione dell'analisi dei requisiti Validare i requisiti: Gli utenti possono testare il prototipo e confermare se il sistema soddisfa le loro esigenze. Ridurre rischi: Identificare e risolvere problemi nelle prime fasi dello sviluppo.
- E' possibile che il design e l'implementazione a classi richiedano tempi elevati, volendo provvedere generalità e riusabilità; a fronte di ciò si ha però una drastica riduzione dei costi di manutenzione

Object-oriented analysis

"Di che cosa necessita il programma?"

"Quali classi saranno presenti?"

"Qual è la responsabilità di ciascuna classe?"

- Attività:
  - determinare la funzionalità del sistema — Viene definito l'insieme delle funzionalità richieste
  - creare una lista delle classi che sono parte del sistema
  - distribuire le funzionalità del sistema attraverso le classi individuate
- In una buona analisi …
  - le classi sono relativamente "piccole" e molte sono abbastanza generali da poter essere riusate in futuri progetti — Le classi devono avere una singola responsabilità ben definita (Single Responsibility Principle).
  - le responsabilità e il controllo sono distribuiti, in altre parole il progetto non ha un "centro" esplicito
  - ci sono poche assunzioni riguardo al linguaggio di programmazione da usare

Object-oriented design

"Come gestirà la classe le sue responsabilità?"

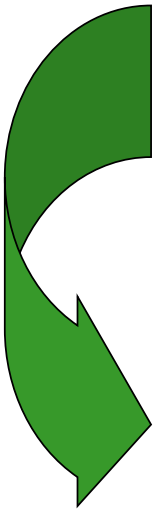
"Quali informazioni sono necessarie alla classe?"

"Come comunicheranno le classi tra loro?"

- Attività:
  - determinare metodi e attributi di ciascuna classe
  - progettare algoritmi per implementare le operazioni
  - progettare le associazioni — Si definiscono le relazioni tra le classi, come associazioni, aggregazioni o composizioni.
- In un buon design…
  - i percorsi di accesso ai dati sono ottimizzati — per minimizzare il consumo di risorse e i tempi di elaborazione.
  - le classi sono raggruppate in moduli — per migliorare la leggibilità e la manutenzione del sistema.

Alcuni approcci object-oriented

- Booch OOD
- Coad-Yourdon OOA/OOD
- Jacobson OOSE
- Rubin-Goldberg OBA
- Rumbaugh OMT
- Shlaer-Mellor OOA
- ……..



UML, acronimo di Unified Modeling Language, è un linguaggio di modellazione visiva standardizzato utilizzato principalmente per rappresentare, specificare, costruire e documentare le parti di un sistema. Basato sul paradigma orientato agli oggetti.