Introduzione a Classi e Oggetti

Oggetto: concetto astratto

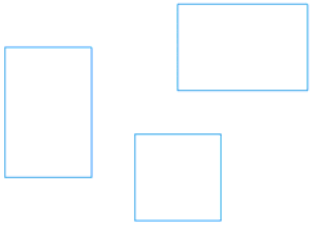
- Entità di un programma dotata di tre proprietà caratteristiche
  - stato
    - informazioni conservate nell'oggetto
    - condizionano il comportamento dell'oggetto nel futuro
    - può variare nel tempo per effetto di un'operazione sull'oggetto
    - Es. casella vocale:
      vuoto = nessun messaggio pieno = elevato # messaggi
      accetta un messaggio se e solo se non piena
  - comportamento
    - definito dalle operazioni che possono essere eseguite dall'oggetto (metodi) e che possono modificare lo stato dell'oggetto
    - Es. casella vocale:
      aggiungere un messaggio ad un elenco
      recuperare un messaggio memorizzato
  - identità
    - ogni oggetto ha una propria identità
    - Es. casella vocale:
      due caselle con gli stessi messaggi sono pur sempre distinguibili

Classe: concetto astratto

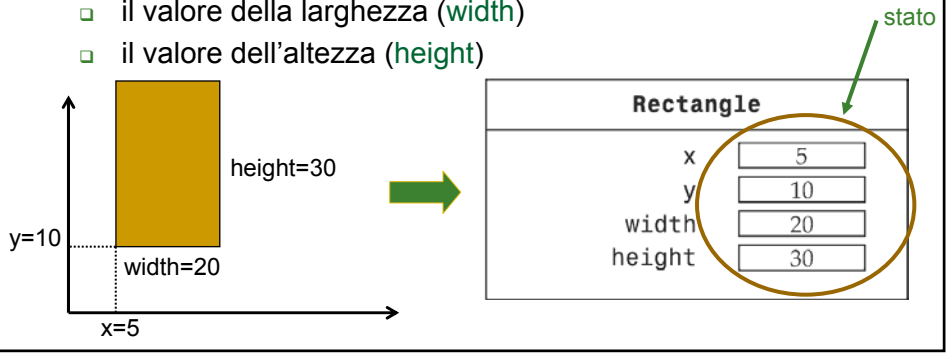
- Ogni oggetto appartiene a (è un'istanza di) una *classe* che ne determina il tipo
- Una classe descrive un insieme di oggetti caratterizzati dallo stesso insieme di
  - comportamenti possibili (metodi)
  - stati possibili (variabili istanza o campi)
- Es. tutte le caselle vocali di un certo tipo appartengono ad una stessa classe Mailbox

Esempio: Rectangle

- Oggetti di tipo rettangolo:



- Per descrivere un rettangolo posso specificare
  - L'ascissa x e l'ordinata y del suo angolo inferiore sinistro
  - il valore della larghezza (width)
  - il valore dell'altezza (height)



Esempio: Rectangle cont.

- Operazioni possibili:
  - traslazione del punto iniziale: translate(x,y)
  - recupero valore altezza: getHeight()
  - modifica ampiezza e altezza: resize(w,h)
  - intersezione con altro rettangolo: intersection(R)
  - test intersezione non vuota: intersects(R)
  - test uguaglianza: equals(O)
  - etc.

Classi in Java

- In Java il comportamento di un oggetto è descritto da una *classe*
  - Una classe ha
    - Un'interfaccia pubblica
      - Insieme di metodi (funzioni) che si possono invocare per manipolare l'oggetto
      - Es.: Rectangle(x_init,y_init,width_init,height_init) metodo dell'interfaccia che crea un rettangolo (costruttore)
    - Un'implementazione nascosta
      - codice e variabili usati per implementare i medodi dell'interfaccia e non accessibili all'esterno della classe.
      - Es.: x,y,width,height

Possibili stati: le variabili istanza

- Le variabili istanza (campi) memorizzano lo *stato* di un oggetto
- Ciascun oggetto di una certa classe ha la propria copia delle variabili istanza
- Le variabili istanza solitamente possono essere lette e modificate solo dai metodi della stessa classe

(*incapsulamento dei dati)*

Possibili comportamenti: metodi

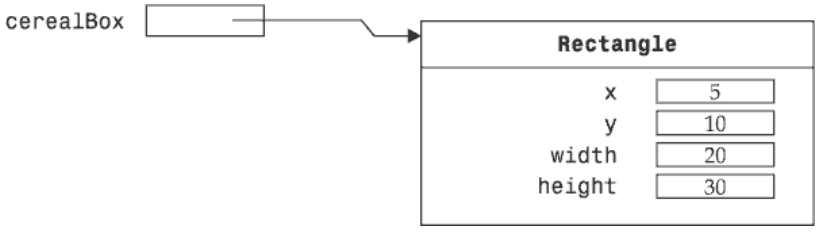
- I metodi di una classe esprimono la parte computazionale della classe
- somigliano a funzioni dei linguaggi procedurali tipo C
- possono utilizzare gli altri metodi della classe e manipolare il contenuto delle variabili istanza

Esempio: codice Java classe Rectangle

File Rectangle.java

```
public class Rectangle{
 //costruttore
  public Rectangle(int x_init,int y_init,int width_init, int height_init)  {
        x=x_init;
        y=y_init;
        width=width_init;
        height=height_init;
       }
 //metodo che sposta il rettangolo
  public void translate(int x_new,int y_new){
         x=x+x_new;
         y=y+y_new;
         return;
  }
… //altri metodi
 // variabili di istanza
     private int x,y,width,height;
}
```

Variabili oggetto



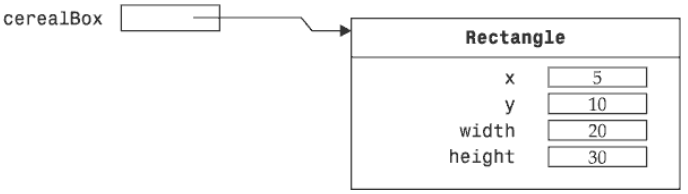
- La variabile cerealBox contiene un riferimento (indirizzo) ad un oggetto di tipo rettangolo.

Creazione di oggetti (1)

- Rectangle cerealBox;
  - Definisce una variabile oggetto rettangolo non inizializzata

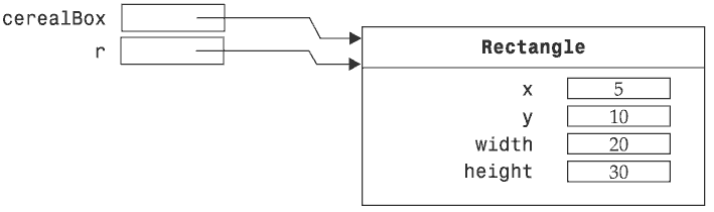
  - è buona norma inizializzare sempre le variabili oggetto

- cerealBox = new Rectangle(5, 10, 20, 30);
  - Crea un rettangolo e assegna il suo indirizzo a cerealBox



Creazione di oggetti (2)

- Rectangle cerealBox = new Rectangle(5, 10, 20, 30);
- Rectangle r = ceralBox; (assegnamento a variabile)
  - Si ottengono due variabili oggetto che si riferiscono allo stesso oggetto

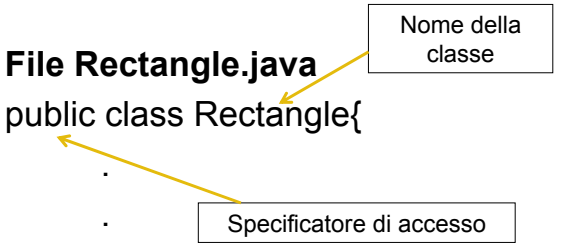


Creazione di Oggetti (3)

- Quando viene creato l'**oggetto** cerealBox con Rectangle cerealBox = new Rectangle(5, 10, 20, 30);

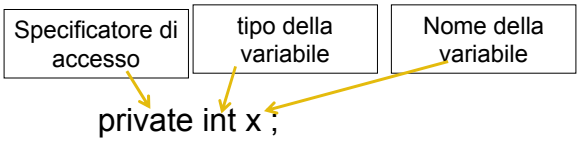
  viene allocato uno spazio di memoria in cui sono conservati
  - i valori di x, y, width e height
    - quindi ciascuna istanza di Rectangle ha le proprie copie delle variabili x, y, width e height
  - gli indirizzi dei **metodi** Rectangle, translate, etc.
    - quindi i metodi di tutte le istanze di Rectangle sono implementati con lo stesso codice

Definizione di una classe



- *specificatore di accesso* **public** indica che la classe Rectangle è utilizzabile anche al di fuori del *package* di cui fa parte la classe
- una classe pubblica deve essere contenuta in un file avente il suo stesso nome
  - Es.: la classe Rectangle è memorizzata nel file Rectangle.java

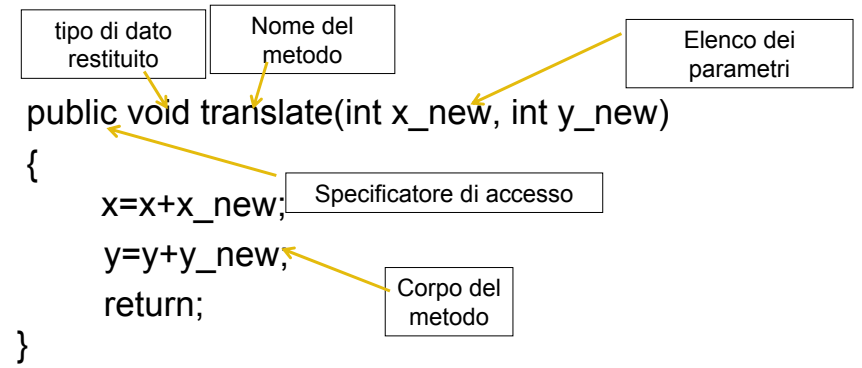
Definizione di una variabile istanza



- Lo *specificatore di accesso* indica la visibilità *(scope)* della variabile
  - **private** indica che la variabile istanza può essere letta e modificata solo dai metodi della classe
    - dall'esterno è possibile accedere alle variabili istanza **private** solo attraverso i metodi pubblici della classe
    - Solo raramente le variabili istanza sono dichiarate **public**
- Il tipo delle variabili istanza può essere
  - una classe, Es.: String
  - un array
  - un tipo primitivo, Es.: int

  } Tipi riferimento

Definizione di un metodo



- Lo *specificatore di accesso* indica la visibilità (scopo) del metodo
  - **public** indica che il metodo può essere invocato anche dai metodi esterni alla classe Rectangle (e anche da quelli esterni al package a cui appartiene la classe Rectangle)

Invocazione dei metodi

- Per invocare un metodo di un certo **oggetto** bisogna specificare il nome del metodo preceduto dal nome dell'oggetto e da un punto
  - Es.: cerealBox.translate(1,34);

  (Eseguiamo il metodo *translate* sull'oggetto **cerealBox** passandogli i valori 1 e 34)
- L'oggetto funge da *parametro implicito* nell'invocazione del metodo
  - E' come passare a *translate* come terzo parametro cerealBox

Parametri impliciti

- All'interno di un metodo per riferirsi esplicitamente al parametro implicito si può usare la parola chiave this
- Il nome di una variabile istanza o di un metodo all'interno di un metodo di una classe si riferisce alla variabile istanza o al metodo del parametro implicito
- A volte per chiarezza si usa this.nomeMetodo o this.nomeVariabile
  - ad es., in un metodo che esegue un'operazione con un altro oggetto della stessa classe

    this.x = paramRect.getX();

    (paramRect è un oggetto della classe Rectangle passato come parametro al metodo)

Il metodo costruttore

```
public Rectangle(int x_init,int y_init,int width_init, int height_init)  {
    x=x_init;
    y=y_init;
    width=width_init;
    height=height_init;
  }
```

- Una **classe** può implementare un **metodo** particolare, detto **costruttore,** che serve a inizializzare i nuovi oggetti
- Quando esiste un **costruttore** deve chiamarsi come la **classe**
- Per creare un oggetto di una classe deve essere invocato un costruttore della classe in combinazione con l'operatore **new**
- Se una variabile di istanza non è inizializzata dal costruttore allora è inizializzata automaticamente a 0 se si tratta di un tipo numerico o a null se si tratta di un riferimento

Overloading (sovraccarico)

- Più metodi con lo stesso nome
  - Consentito se i parametri li distinguono, cioè hanno firme diverse

  (firma = nome del metodo + lista tipi dei parametri nell'ordine in cui compaiono)
  - Il tipo restituito non conta
- Frequente con costruttori
  - Devono avere lo stesso nome della classe
  - Es.: aggiungiamo a Rectangle il costruttore

```
public Rectangle(int x_init,int y_init)  {
        x=x_init;
        y=y_init;
}
```

- Usato anche quando dobbiamo agire diversamente a seconda del tipo passato
  - Ad es., *println* della classe *PrintStream*

Usare classi di altri pacchetti

- Le classi non contenute nel pacchetto java.lang devono essere importate
  - Es.: import java.awt.Rectangle;
- Se interessano tutte le classi di un pacchetto si può usare il costrutto ***import nomePacchetto.****
  - Es.: import java.awt.*
- In alternativa, si può specificare tutto il nome di ogni classe.
  - Es.: java.awt.Rectangle cerealBox = new java.awt.Rectangle(5, 10, 20, 30);

Esempio: importazione di Rectangle

File MoveTest.java

```
//importa la classe Rectangle del pacchetto java.awt
import java.awt.Rectangle;

public class MoveTest
   {
      public static void main(String[] args)
      {
         Rectangle cerealBox = new Rectangle(5, 10, 20, 30);

         // sposta il rettangolo
         cerealBox.translate(15, 25);

         // stampa il rettangolo spostato
         System.out.println(cerealBox);
   }
}
```

Categorie di variabili

- Variabili istanza
  - Appartengono all'oggetto
  - Esistono finché l'oggetto esiste
  - Hanno un valore iniziale di default
- Variabili locali
  - Appartengono al metodo
  - Vengono create all'attivazione del metodo e cessano di esistere con esso
  - Non hanno valore iniziale se non inizializzate
- 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

Progettazione ad oggetti

- Caratterizzazione attraverso le **classi** delle entità (oggetti) coinvolte nel problema da risolvere (individuazione classi)
  - identificazione delle classi
  - identificazione delle responsabilità (operazioni) di ogni classe
  - individuazione delle relazioni tra le classi
    - dipendenza (usa oggetti di altre classi)
    - aggregazione (contiene oggetti di altre classi)
    - ereditarietà (relazione sottoclasse/superclasse)
- Realizzazione delle classi

Realizzazione di una classe

1. individuazione dei metodi dell'interfaccia pubblica:
   - determinazione delle operazioni che si vogliono eseguire su ogni oggetto della classe
2. individuazione delle variabili istanza:
   - determinazione dei dati da mantenere
3. individuazione dei costruttori
4. Codifica dei metodi
5. Collaudo del codice

Incapsulamento dei Dati (1)

- In genere quando si definisce una classe, i metodi dell'interfaccia pubblica sono dichiarati **public**, mentre i metodi di servizio e tutti i campi sono dichiarati **private**
  - Si rende accessibile al mondo esterno solo l'interfaccia dell'oggetto e si nasconde la sua implementazione (astrazione delle caratteristiche della classe)
  - L'interfaccia determina il modo in cui l'oggetto comunica con l'ambiente circostante
- Questo processo di nascondere i dettagli di definizione degli oggetti si chiama incapsulamento dei dati ed è uno dei concetti base della programmazione a oggetti

Incapsulamento dei Dati (2)

L'incapsulamento dei dati presenta due notevoli vantaggi

- Gli oggetti sono protetti da un utilizzo errato o fraudolento
  - Solo i metodi definiti per la classe possono alterare i campi di un oggetto
- Le classi sono facilmente riutilizzabili
  - Alcuni programmatori possono utilizzare classi già definite
    - Non hanno bisogno di sapere come è stata implementata la classe
    - Possono creare oggetti appartenenti alla classe e invocarne i metodi dell'interfaccia pubblica
    - Possono creare nuove classi che utilizzano la classe come membro

Programmi Java

- Un programma Java consiste di una o più classi
- Per poter eseguire un programma bisogna definire una classe pubblica che contiene un metodo main(String[ ] args)

Esempio: Classe Conto Corrente

Una **classe** che rappresenti dei conti correnti bancari potrebbe contenere

- Un campo in cui memorizzare il saldo
  - Un float (tipo primitivo o **oggetto**)
- Un metodo **costruttore**
  - Inizializza il saldo del nuovo conto corrente alla cifra con cui viene aperto
- Un metodo che legge il saldo del conto
- Un metodo che preleva denaro
- Un metodo che deposita denaro

Esempio: Classe Conto Corrente

```
public class ContoCorrente {

   // variabili di istanza
   private float saldo;
   // Costruttore
   public ContoCorrente(float saldoIniziale) {
            saldo = saldoIniziale; }
//metodi dell'interfaccia pubblica
public float leggiSaldo(void) {
            return saldo; }
public void preleva(float quantita) {
            saldo = saldo - quantita; }
   public void deposita(float quantita) {
            saldo = saldo + quantita; }
}
```

Esempio: Classe Conto Corrente

Un codice equivalente in C si servirebbe delle parole chiave **struct** e **typedef** e definirebbe le relative funzioni

```
/* File sorgente conto_corrente.h */

typedef struct conto_corrente {
  float saldo;
};

conto_corrente *
new_conto_corrente(float);
float leggi_saldo(conto_corrente *);
void preleva(conto_corrente *, float);
void deposita(conto_corrente *, float);
```

```
/* File sorgente conto_corrente.c */

conto_corrente * new_conto_corrente(float s)
{  conto_corrente *c=malloc(sizeof(conto_corrente));
    c->saldo=s;
    return c; }

float leggi_saldo(conto_corrente *c)
{  return c->saldo; }

void preleva(conto_corrente *c, float p)
{  c->saldo=c->saldo+s; }

void deposita(conto_corrente *c, float d)
{  c->saldo=c->saldo-s; }
```

Esempio: Classe Conto Corrente

- Per gestire il conto corrente del signor Rossi, possiamo creare un **oggetto** di tipo *ContoCorrente*:

  ContoCorrente contoRossi = new ContoCorrente(5000);

- Per tenere traccia dei movimenti che il signor Rossi fa sul suo conto corrente possiamo chiamare i relativi **metodi**:

  contoRossi.preleva(100); **// preleva 100 euro**

  contoRossi.deposita(200); **// deposita 200 euro**

  Per chiamare il **metodo** *preleva* inviamo il **messaggio** *preleva* all'**oggetto** *contoRossi*

Funzionamento di *contoRossi*

- Quando viene creato l'**oggetto** *contoRossi*, viene allocato uno spazio di memoria in cui sono conservati il valore del *saldo* e gli indirizzi dei **metodi** *preleva*, *deposita* e *leggiSaldo*
- L'uso dell'oggetto *contoRossi* avviene attraverso i metodi dell'interfaccia pubblica, non abbiamo bisogno di conoscere l'implementazione sottostante (astrazione)

Parallelo con il C

- Per fare la stessa cosa in C, possiamo creare una elemento di tipo conto_corrente, chiamando **new_conto_corrente**

  conto_corrente *conto_rossi = new_conto_corrente(5000);

- Per tenere traccia dei movimenti che il signor Rossi fa sul suo conto corrente possiamo chiamare le relative funzioni:

  prelievo(conto_rossi,100); // preleva 100 euro
  deposito(conto_rossi,200); // deposita 200 euro

  Per chiamare la funzione *preleva* dobbiamo usare *conto_rossi* come argomento!!!

Esempio: Classe Cliente

Definiamo la classe *cliente* che utilizza la classe *contoCorrente*

```
public class Cliente {
  private String datiAnagrafici;
  private ContoCorrente conto;
  public Cliente(String dati, float saldo) {
        datiAnagrafici = dati;
        conto = new ContoCorrente(saldo) } // Costruttore
  public String leggiDati(void) {
        return datiAnagrafici; }
  public float saldo(void) {
        return conto.leggiSaldo(); }
  }
```

Esercizio

- Implementare in Java una classe Punto che contiene le coordinate di un punto nel piano e ha come interfaccia pubblica i metodi per leggere e modificare le coordinate.
- Implementare la classe Rettangolo che differisce da Rectangle in quanto usa un'istanza di Punto come vertice in basso a sinistra invece delle due coordinate. (Considerare per questa implementazione di Rettangolo un insieme di metodi a scelta che fanno uso del punto iniziale)
- Aggiungere alla classe ContoCorrente un metodo che aggiunga al saldo gli interessi annuali calcolati ad un tasso passato come parametro.
- Scrivere un breve programma per collaudare le classi:
  - ContoCorrente e Cliente
  - Punto
  - Rettangolo