Metodologie di Programmazione

Lezione 19: Le interfacce

Lezione 19: Sommario

UNITELMA SAPIENZA
SAPIENZA UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

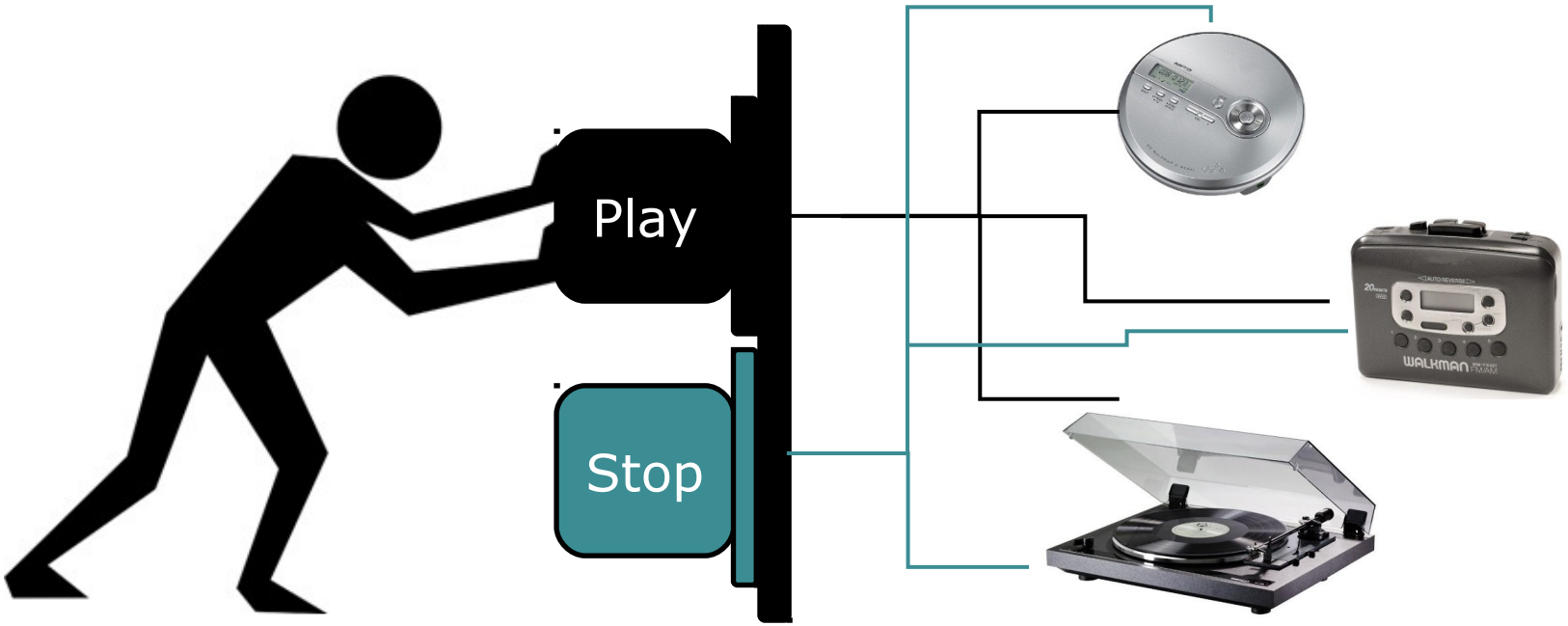
- Che cos'è un'interfaccia?
- Dichiarazione
- Implementazione
- Il contratto
- Differenze con le classi astratte
- Ereditarietà multipla
- Interfacce notevoli
- Callback mediante interfacce

Le interfacce in Java

- Le interfacce sono uno strumento che Java mette a disposizione per consentire a più classi di fornire e, in seguito, implementare un insieme di metodi comuni
- Le interfacce definiscono e standardizzano l'interazione fra oggetti tramite un insieme limitato di operazioni

Le interfacce in Java

- Esse specificano soltanto il **comportamento** che un certo oggetto deve presentare all'esterno, cioè **cosa** quell'oggetto può fare
- L'**implementazione** di tali operazioni, cioè **come** queste vengono tradotte e realizzate, rimane invece non definito



Le interfacce in uno slogan

Le interfacce sono classi astratte al 100%

Dichiarazione di un'interfaccia (1)

- Un'interfaccia è una classe che può contenere soltanto
  - Costanti
  - Metodi astratti
- Tutti i **metodi** dichiarati in un'interfaccia sono implicitamente **public abstract**
- Tutti i **campi** dichiarati in un'interfaccia sono implicitamente **public static final**
- Non è possibile specificare alcun dettaglio implementativo
  - non vi è alcun corpo di metodo o variabile di istanza
  - tranne nelle implementazioni di default (trattate in seguito)

Dichiarazione di un'interfaccia (2)


```
public interface SupportoRiscrivibile
{
    int TIMES = 1000;

    void leggi();
    void scrivi();
}
```

Parola chiave interface: indica la definizione di una nuova interfaccia

Costante: implicitamente final, public e static

Dichiarazione di metodi: anche se non esplicitato, sono public e abstract

E' vietato definire qualunque implementazione (**tranne** con metodi di default da Java 8 in poi)

Implementare un'interfaccia (1)


```
public class Nastro implements SupportoRiscrivibile
{
    private Pellicola pellicola;

    @Override
    public void leggi()
    {
        attivaTestina();
        muoviTestina();
    }

    @Override
    public void scrivi()
    {
        attivaTestina();
        caricaTestina();
        muoviTestina();
        scaricaTestina();
    }

    public void attivaTestina() {}
    public void caricaTestina() {}
    public void scaricaTestina() {}
    public void muoviTestina() {}
```

- Per realizzare un'interfaccia è necessario che una classe la **implementi**, tramite la parola chiave **implements**
- Una classe che implementa una interfaccia decide di voler **esporre** pubblicamente all'esterno il comportamento descritto dall'interfaccia
- E' obbligatorio che ciascun metodo abbia esattamente la stessa intestazione che esso presenta nell'interfaccia

Implementare un'interfaccia (2)


```
public class MemoriaUsb implements SupportoRiscrivibile
{
    private CellaMemoria[] celle;

    @Override
    public void leggi()
    {
        // Leggi la cella corretta
    }

    @Override
    public void scrivi()
    {
        // Modifica la cella corretta
    }
}
```

Anche la classe MemoriaUsb implementa l'interfaccia SupportoRiscrivibile, definendo i propri metodi leggi() e scrivi()

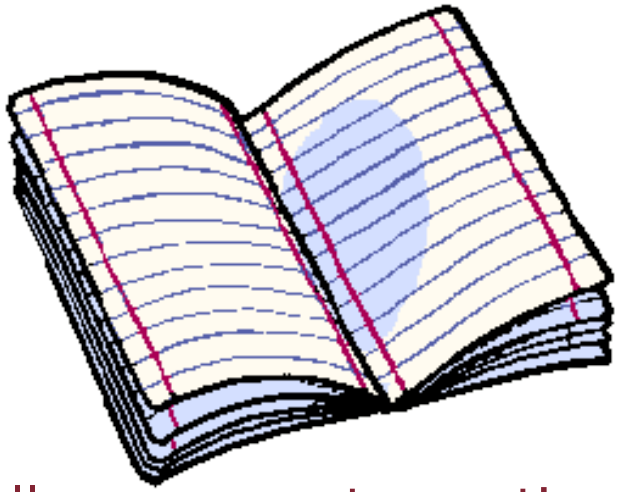
Implementare un'interfaccia (3)


```
public class Quaderno implements SupportoRiscrivibile
{
    private Foglio[] fogli;

    @Override
    public void leggi()
    {
        // Leggi la pagina corrente
    }

    @Override
    public void scrivi()
    {
        // Scrivi sulla pagina corrente
    }
}
```

Perfino un quaderno può essere visto come un supporto che è possibile leggere e scrivere!



Le interfacce permettono di modellare comportamenti comuni a classi che **non sono necessariamente** in relazione gerarchica (**is-a**, **è-un**)

Un esempio: l'interfaccia Iterabile (1)

- Ci sono molte classi di **natura diversa** che rappresentano sequenze di elementi
  - Es. array, liste, stringhe, ecc.
- Tuttavia, le sequenze hanno qualcosa in comune: è possibile iterare sui loro elementi:

```
public interface Iterabile
{
    boolean hasNext();
    Object next();
    void reset();
}
```

Un esempio: l'interfaccia Iterabile (2)

- Ciascuna classe implementerà i metodi **a suo modo**:

```
public class MyIntegerArray implements Iterabile
{
    private Integer[] array;
    private int k = 0;

    public MyIntegerArray(Integer[] array)
    {
        this.array = array;
    }

    @Override
    public boolean hasNext()
    {
        return k < array.length;
    }

    @Override
    public Object next()
    {
        return array[k++];
    }

    @Override
    public void reset() { k = 0; }
}
```

```
public class MyString implements Iterabile
{
    private String s;
    private int k = 0;

    public MyString(String s)
    {
        this.s = s;
    }

    @Override
    public boolean hasNext()
    {
        return k < s.length();
    }

    @Override
    public Object next()
    {
        return s.charAt(k++);
    }

    @Override
    public void reset() { k = 0; }
}
```

Un esempio: l'interfaccia Iterabile (3)

- Infine, testiamo le due classi:

```
public class InterfacceChePassione
{
    static public void main(String[] args)
    {
        Iterabile i1 = new MyIntegerArray(new Integer[] {10, 20, 30, 40});
        Iterabile i2 = new MyString("abcdefghi");

        for (Iterabile i : new Iterabile[] { i1, i2 })
        {
            while(i.hasNext())
                System.out.println(i.next());
        }
    }
}
```

Assegniamo un oggetto di una classe che implementa l'interfaccia a una variabile-riferimento a interfaccia

Possiamo creare un array di riferimenti a interfaccia!

Abbiamo ora un meccanismo generale per iterare su **sequenze** che implementano l'interfaccia

Implementare un'interfaccia: il contratto (1)

- Implementare un'interfaccia equivale a **firmare un contratto** con il compilatore che stabilisce l'impegno ad implementare tutti i metodi specificati dall'interfaccia o a dichiarare la classe abstract

Implementare un'interfaccia: il contratto (2)

- 3 possibilità per una classe che implementa un'interfaccia:
  - fornire un'implementazione concreta di tutti i metodi, definendone il corpo
  - fornire l'implementazione concreta solo per un sottoinsieme proprio dei metodi dell'interfaccia
  - decidere di non fornire alcuna implementazione concreta

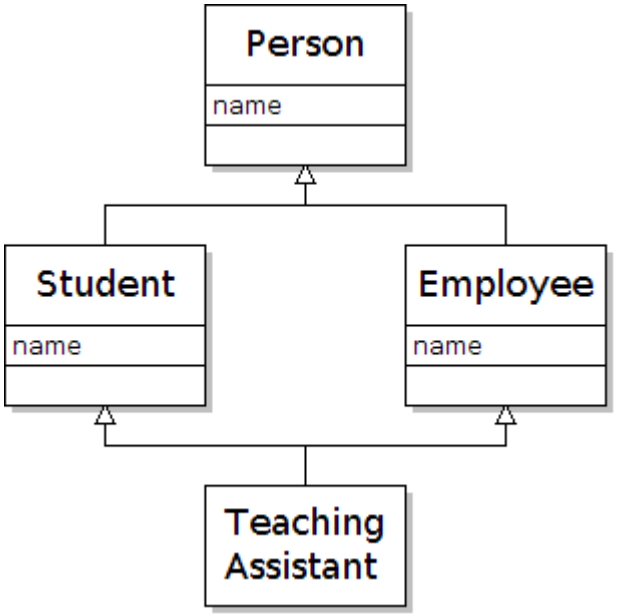
- Negli ultimi due casi, però, la classe va dichiarata abstract

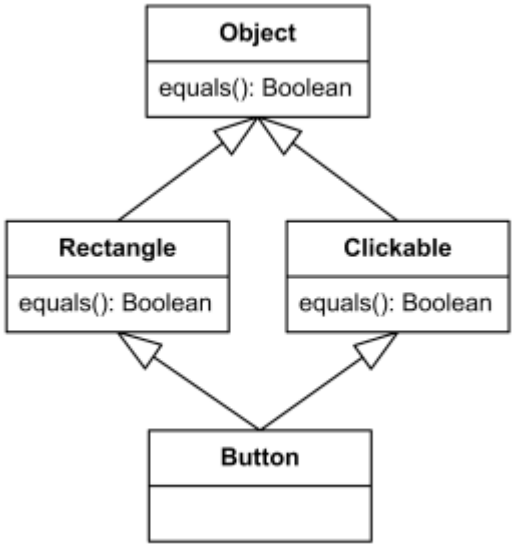
Interfacce vs. classi astratte

- **Una domanda nasce spontanea:** Se implementando un'interfaccia **devo** dichiarare tutti i metodi in essa definiti, perché non ricorrere ad una classe astratta?

Il problema del diamante

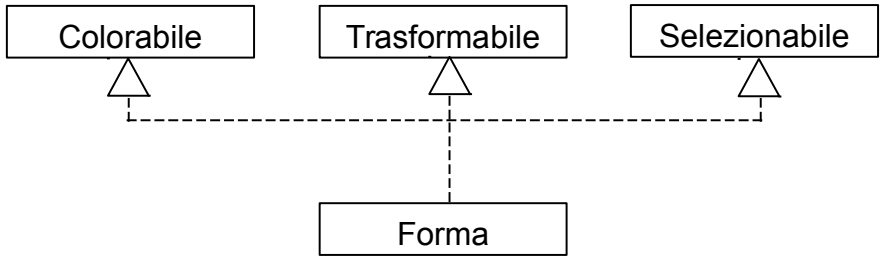
- Con l'ereditarietà multipla si possono creare situazioni **poco chiare** di duplicazione di metodi e campi





Ereditarietà multipla (1)

- In Java non è consentito estendere più di una classe alla volta: ovvero extends può essere seguito **solo da un unico nome di classe**
- Al contrario, una classe può implementare tutte le interfacce desiderate



- E' inoltre possibile estendere **1** classe e contemporaneamente implementare **n** interfacce diverse

Ereditarietà multipla (2)


```
public class Forma implements Colorabile, Trasformabile, Selezionabile
{

    @Override
    public void seleziona()
    {
    }

    @Override
    public void deseleziona()
    {
    }

    @Override
    public void sposta(Point point)
    {
    }

    @Override
    public void ruota(Angle angle)
    {
    }

    @Override
    public void ritaglia(Point upperLeft, Point bottoRight)
    {
    }

    @Override
    public void scala(double factor)
    {
    }

    @Override
    public void colora(Color c)
    {
    }

    @Override
    public void rimuoviColore()
    {
    }
}
```

La classe Forma implementa **3 interfacce** e definisce tutti i metodi da esse dichiarati

```
public interface Selezionabile
{
    void seleziona();
    void deseleziona();
}
```

```
public interface Colorabile
{
    void colora(Color c);
    void rimuoviColore();
}
```

```
public interface Trasformabile
{
    void sposta(Point point);
    void ruota(Angle angle);
    void ritaglia(Point upperLeft, Point bottoRight);
    void scala(double factor);
}
```

Relazione tra interfacce e classi che le implementano

- Nel momento in cui una classe C decide di implementare un'interfaccia I, tra queste due classi si instaura una relazione di tipo **is-a**, ovvero C è di tipo I
- Comportamento simile a quello dell'ereditarietà
- Quindi anche per le interfacce valgono le regole del polimorfismo
- Ad esempio, la seguente dichiarazione è lecita:

```
SupportoRiscrivibile supporto = new Nastro();
supporto.leggi();
```

- E ci consente di usare l'oggetto della classe Nastro **come fosse di tipo SupportoRiscrivibile**
- Con conseguente **restringimento della visibilità** ai soli metodi dell'interfaccia SupportoRiscrivibile

Interfacce notevoli

Interfaccia	Descrizione
Comparable	Impone un ordinamento naturale degli oggetti tramite il metodo: int compareTo(Object b), che restituisce un valore >, = o < 0 se l'oggetto è rispettivamente maggiore, uguale o minore di b
Cloneable	Le classi che implementano quest'interfaccia dichiarano al metodo clone() di Object che è legale effettuare una copia campo-a-campo delle istanze della classe Il metodo clone() invocato su oggetti di classi che non implementano Cloneable solleva una CloneNotSupportedException
Serializable	Quest'interfaccia non possiede metodi o campi e serve soltanto ad identificare il fatto che l'oggetto è serializzabile, cioè memorizzabile su un certo supporto

Esercizio: Successioni

- Progettare tre classi che realizzano diversi tipi di successioni
  - La successione $\{ i^2 \}$ (es. 0, 1, 4, 9, 16, ecc.)
  - La successione casuale (es. -42, 2, 5, 18, 154, ecc.)
  - La successione di Fibonacci (es. 1, 1, 2, 3, 5, 8, 13, ecc.)
- Elaborare un meccanismo generale per la generazione della successione indipendentemente dall'implementazione specifica

Esercizio: Animali

- Progettare una gerarchia di classi dei seguenti animali, ciascuno con determinate caratteristiche:
  - Uccello (vola, becca)
  - Pinguino (becca, nuota)
  - Aquila (vola, becca)
  - Pesce (nuota)
  - Pesce volante (nuota, vola)
  - Cane (salta, corre, fedele a, domestico)
  - Felino (salta, corre, fa le fusa)
  - Gatto (salta, corre, fa le fusa, domestico)
  - Uomo (salta, corre, pensa, nuota, vola?)

Passare funzioni in input mediante le interfacce

- Le interfacce permettono il passaggio in input di funzioni con una determinata intestazione
- Ad esempio:

```
public interface Callable
{
    Object esegui(Object o);
}
```