

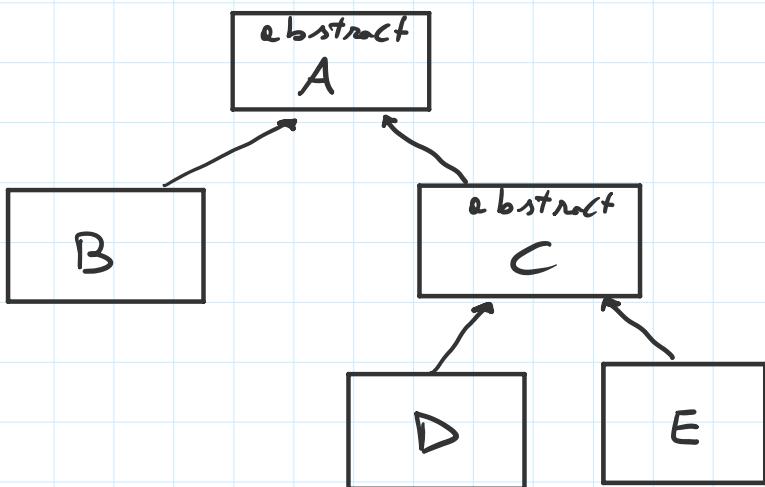
Classi Astratte

Una classe astratta non può essere istanziata

Può essere usata come classe base nella definizione di nuove classi

Utile quindi come "modelli" per le definizioni di nuove classi.

Poss. creare riferimenti di tipo classe astratta



Posso creare istanze di B, D, E

Non posso creare istanze di A, C

Posso creare riferimenti di tipo A, B, C, D, E

Esempio:

```
abstract class OggettoGrafico {  
    int x,y;  
    void sposta (int nuovax, int nuova y){  
        _
```

=
}
}
class Bottone extends OggettoGrafico {
:
}
class CasellaDiTesto extends OggettoGrafico {
:
}

Non possono esistere oggetti grafici che non siano bottoni o caselle di testo.

OggettoGrafico oggetto = new OggettoGrafico(); //ERRORE

OggettoGrafico oggetto = new Bottone(); //OK

oggetto.sposta(100, 200); //OK, ereditato
da OggettoGrafico

In una classe estesa alcuni metodi possono essere indicati come abstract

Per esempio:

abstract class OggettoGrafico {
 int x, y;
 void sposta (int nuovox, int nuovoy);

```
int x, y,  
void sposta (int nuovox, int nuovoy);  
=
```

{

abstract void disegna ();

}

Nota: c'è ; non { }

```
class Bottone extends OggettoGrafico {
```

```
void disegna () {
```

=

definizione di disegna

{

```
class CasellaDiTesto extends OggettoGrafico {
```

```
String testo;
```

```
void disegna () {
```

=

{

```
void altroMetodo () {
```

=

{

Se c'è almeno un metodo abstract la classe deve essere abstract.

```
OggettoGrafico oggetto = new Bottone();  
OggettoGrafico oggetto2 = new CasellaDiTesto();  
oggetto. sposta (100, 200);
```

Oggetto Guglio og2: new CassettaDiTesoro();
og2. sposta(100, 200);
og2. disegna(); // viene eseguito il disegno
d' Bottone

```
abstract class X {  
    void m1() {  
        =  
        m2(); // Posso chiamare  
        =  
    }  
    abstract void m2();  
};
```

Classi e metodi final

Una classe final non può essere estesa

```
final class Y {  
};
```

non posso scrivere

```
class X extends Y {  
};
```

errore

E' possibile definire metodi final

```
class C {
```

```
final void mimetodo () {  
    =
```

```
}
```

```
:
```

```
}
```

OK

```
class D extends C {
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

```
:
```

Interface

In Java un'interfaccia è una raccolta di metodi abstract

Un'interfaccia definisce un protocollo di comportamento e cui le classi possono decidere di aderire

Possiamo definire nuove interfacce

Possiamo scrivere classi che implementano interfacce

Per definire un'interfaccia si usa la parola chiave interface

Esempio:

```
public interface Comparable {
```

```
    int compare ( Object o );
```

```
}
```

↑
implicitamente public e abstract

↑
in Comparable.java

```
$ javac Comparable.java
```

```
produce Comparable.class
```

Una classe che implementa un'interfaccia deve fornire l'implementazione di tutti i metodi dell'interfaccia.

Esempio:

```
public class Data implements Comparable {
```

```
    int giorno;  
    int mese;  
    int anno;
```

```
    public Data (int g, int m, int a) {
```

```
        giorno = g;  
        mese = m;  
        anno = a;
```

```
}
```

```
    public int compare (Object o) {
```

```
        Data d = (Data) o;  
        if (anno != d.anno)  
            return anno - d.anno;  
        if (mese != d.mese)  
            return mese - d.mese;  
        return giorno - d.giorno;
```

```
}
```

```
}
```

Posso creare riferimenti al tipo interfaccia.
Non posso creare oggetti al tipo interfaccia.

Riferimenti al tipo interfaccia possono puntare a oggetti che sono istanze di classi che implementano l'interfaccia.

Data d1 = new Data(10, 5, 1950);

Data d2 = new Data(11, 6, 2000);

Comparabile e1 = d1;

int r = e1.compara(d2);

→ posso chiamare i soli
metodi dell'interfaccia

Anche la classe ContoBancario puo' aderire
allo stesso protocollo di comportamento

public class ContoBancario implements Comparabile {

=

public int compara(Object o) {

ContoBancario c = (ContoBancario) o; ↗

if (bilancio < c.bilancio)

return -1;

if (bilancio > c.bilancio)

return 1;

return 0;

} }

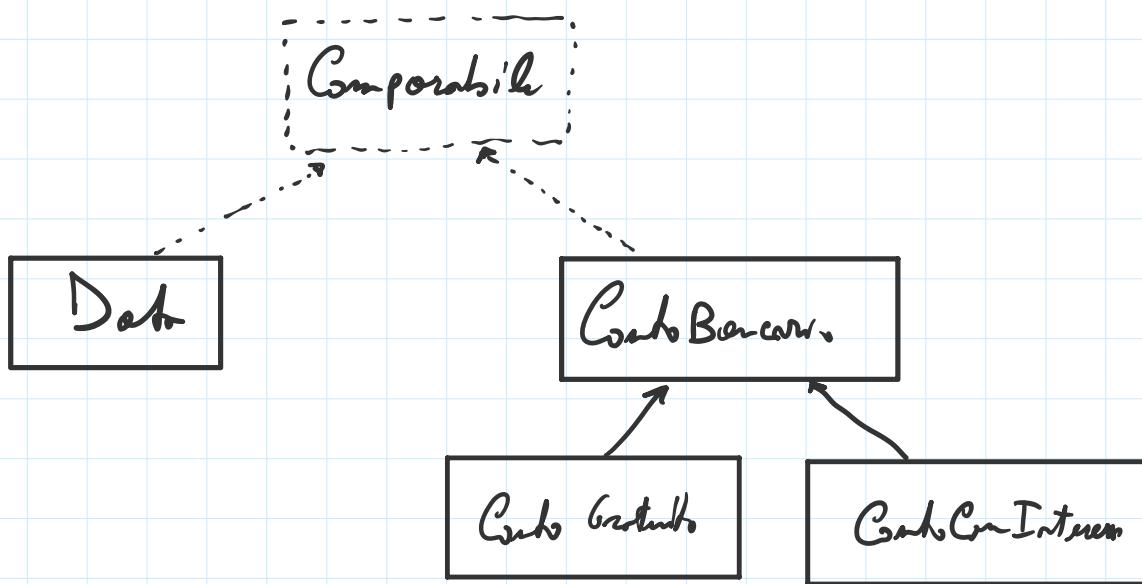
Comparabile x = new ContoBancario(...);

ContoBancario y = new ContoBancario(...);

int w = x.compara(y);

`X.Compara (new Data (...));`

ClassCastException : una Data
non può essere convertita a ContoBancario



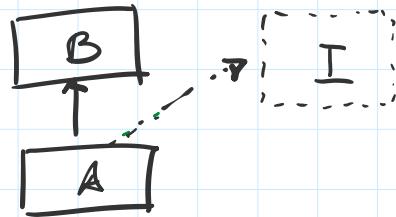
Una classe può implementare tutte le interfacce che vuole

```
class X implements Interf1, Interf2, Interf3 {  
    :  
}
```

Una classe può estendere un'altra e al tempo stesso implementare una o più interfacce

```
class A extends B implements I {
```

}



Se una classe implementa più interfacce
dove fornire una implementazione di Tutti:
metodi di tutte le interfacce

public interface Interf1 {

void m1();

}

public interface Interf2 {

void m2();

int m3(int x);

}

public interface Interf3 {

float m4(float f1, float f2);

}

class X implements Interf1, Interf2, Interf3 {

void m1() {

=

}

void m2() {

=

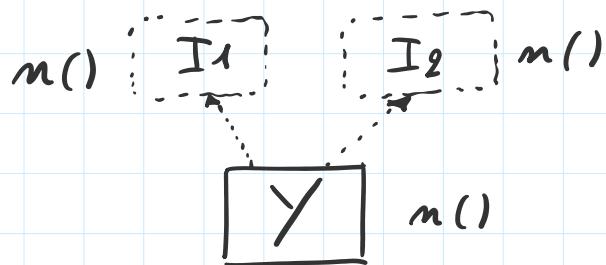
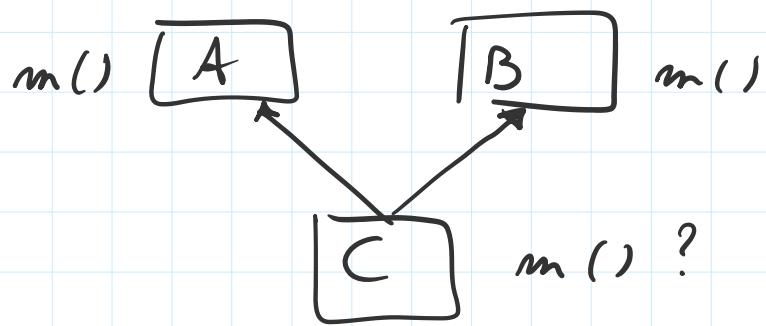
```

{
int m3(int x) {
    =
}

{
float m4(float f1, float f2) {
    =
}

}

```



```

public interface Closable {
    void close();
}

```

```

public class Fluo implements Closable {
    =
    public void close() {
        =
    }
}

```

{

public class FluxoFile extends Fluxo {

Potrei scrivere close() ma
non è obbligatorio.

}

public abstract class Fluxo implements Closeable {

=

}

public class FluxoFile extends Fluxo {

public void close() {

=

{

:

}

Un'interfaccia può contenere delle costanti

Esempio:

public interface Verbose {

int SILENT = 0;
int NORMAL = 1;
int VERBOSE = 8;

} sono implicitamente
public, static e final

void setVerbosity(int level);

int getVerbosity();

}

una classe che implementa un'interfaccia
"eredita" le costanti dell'interfaccia e le può usare.

```
public class Logger implements Verbose {
    private int level;
    :
    public void setVerbosity(int l) {
        level = l;
    }
    public int getVerbosity() {
        return level;
    }
    public void log(String m) {
        if (level == SILENT)
            return;
        if (level == NORMAL)
            :
    }
}
```

posso usare
il nome semplice
della costante

In una classe che non implementa l'interfaccia
può usare lo stesso

NameDell'Interfaccia . nomeDellaCostante

per esempio

Verbose . SILENT

Il linguaggio prende l'interfaccia
java.lang.Comparable

int compareTo(-)

java.util.Arrays

public static void sort(Object[] v)

Tipi enumerazione

Assomigliano ai tipi enumerazione del C/C++

Tutti i valori sono noti nel momento in cui il tipo viene dichiarato

Il numero di valori possibili è non grandissimo

Nella forma più semplice sono molto simili a C/C++

```
class Main {  
    enum Colore {VERDE, GIALLO, ROSSO}; ← viene definito il tipo enum  
    public static void main(String[] args) { ← Colore  
        Colore c1 = Colore.VERDE;  
        // Colore c2 = GIALLO; //ERRORE,  
        usare Colore.GIALLO  
        switch(c1) {  
            case VERDE:  
                System.out.println("Vai");  
                break;  
            case GIALLO: ← qui niente Colore.  
                System.out.println("Rallenta");  
                break;  
            case ROSSO:  
                System.out.println("Fermati");  
        }  
    }  
}
```

valori possibili:
VERDE, GIALLO, ROSSO

In effetti gli enum sono delle classi
che estendono java.lang.Enum

Ereditano da tale classe un certo numero di metodi:

public String toString()

restituisce il
nome dell'enumeratore
(per esempio "GIALLO")

public static E[] values()

restituisce un array
contenente tutti i valori

public static EvalueOf(String n)

restituisce l'enumeratore
corrispondente alla
stringa n

E è il tipo
enumerazione

Se non c'è corrispondenza
lancia IllegalArgumentException

public int ordinal()

restituisce il n° d'ordine
(0 per VERDE, 1 per
GIALLO, -)

Esempio

```
import java.util.Scanner;  
  
class Main {  
  
    enum Colore {VERDE, GIALLO, ROSSO};  
  
    public static void main(String[] args) {  
        Colore c1 = Colore.VERDE;  
        System.out.println(c1);  
        for(Colore c2: Colore.values())  
            System.out.println(c2 + " " + c2.ordinal());  
        Scanner sc = new Scanner(System.in);  
        String parola = sc.next();  
        Colore c3 = Colore.valueOf(parola);  
    }  
}
```

Stampa VERDE

restituisce
un array
di valori

```

Scanner sc = new Scanner(System.in);
String parola = sc.next();
Colore c3 = Colore.valueOf(parola);
System.out.println("Hai inserito: " + c3);
}
}

```

racchiamo
un array
con i valori
di colore

PA2021 enum2

The screenshot shows an IDE interface with two tabs: 'Main.java' and 'Console'. The code in 'Main.java' is:

```

import java.util.Scanner;
class Main {
    enum Colore {VERDE, GIALLO, ROSSO};
    public static void main(String[] args) {
        Colore c1 = Colore.VERDE;
        System.out.println(c1);
        for(Colore c2: Colore.values())
            System.out.println(c2 + " " + c2.ordinal());
        Scanner sc = new Scanner(System.in);
        String parola = sc.next();
        Colore c3 = Colore.valueOf(parola);
        System.out.println("Hai inserito: " + c3);
    }
}

```

The 'Console' tab shows the output of running the program:

```

> javac -classpath ./run_dir/junit-4.12.jar:/run_dir/hamcrest-core-1.3.jar:/run_dir/json-simple-1.1.1.jar -d . Main.java
> java -classpath ./run_dir/junit-4.12.jar:/run_dir/hamcrest-core-1.3.jar:/run_dir/json-simple-1.1.1.jar Main
VERDE
VERDE 0
GIALLO 1
ROSSO 2
VERDE
Hai inserito: VERDE
> []

```

Essendo i tipi enum delle classi è possibile ridefinire il comportamento di metodi.

Qui ridefiniamo `toString()` in modo da avere un output a video che ci piace di più.

```

enum Colore2 {
    VERDE, GIALLO, ROSSO;

    public String toString() {
        switch(this) {
            case VERDE:
                return "Verde";
            case GIALLO:
                return "Giallo";
            case ROSSO:
                return "Rosso";
            default:
                return null;
        }
    }
}

```

forma particolare
di definizione del classe

valori possibili

ridefinisco `toString()`

in una funzione

per ogni valore

```

class Main {
    public static void main(String[] args) {
        Colore2 c = Colore2.ROSSO;
        System.out.println("Colore " + c);
    }
}

```

Colore2.java

PA2021 enum3

The screenshot shows an IDE interface with three main panes:

- Files** pane: Shows files Main.java and Colore2.java.
- Main.java** code editor:

```

1 class Main {
2     public static void main(String[] args) {
3         Colore2 c = Colore2.ROSSO;
4         System.out.println("Colore " + c);
5     }
6 }
7 
```
- Console** pane: Shows the command-line output of the Java application:

```

javac -classpath ./run_dir/junit-4.12.jar:/run_dir/hamcrest-core-1.3.jar:/run_dir/json-simple-1.1.1.jar -d . Colore2.java Main.java
java -classpath ./run_dir/junit-4.12.jar:/run_dir/hamcrest-core-1.3.jar:/run_dir/json-simple-1.1.1.jar Main
Colore Rosso

```

Essendo delle classi è possibile aggiungere stato e/o aggiungere metodi.
Esempio enum ColoreRGB che mantiene delle informazioni e ha metodi in più.

```

enum ColoreRGB {
    VERDE(0, 1, 0), GIALLO(1, 1, 0), ROSSO(1, 0, 0);

    private final double[] componenti;

    ColoreRGB(double r, double g, double b) {
        componenti = new double[]{r, g, b};
    }

    public double luminosita(){
        return 0.2126*componenti[0] +
               0.7152*componenti[1] +
               0.0722*componenti[2];
    }

    private String comp() {
        return "(R=" + componenti[0] +
               ", G=" + componenti[1] +
               ", B=" + componenti[2] + ")";
    }
}

```

enum:
valori possibili
valori da passare
al costruttore

costruttore

```

public String toString() {
    String s = null;
    switch(this) {
        case VERDE: s = "Verde"; break;
        case GIALLO: s = "Giallo"; break;
        case ROSSO: s = "Rosso"; break;
    }
    return s + comp();
}

```

PA2021 enum4

The screenshot shows an IDE interface with the following components:

- Files:** Shows Main.java (selected), jdt.ls-java-project, and ColoreRGB.java.
- Main.java:** Contains the following code:


```

1 class Main {
2     public static void main(String[] args) {
3
4         for(ColoreRGB x: ColoreRGB.values())
5             System.out.println(x + " lum.= " + x.luminosita());
6
7     }
8 }
```
- Console:** Shows the terminal output of the application's execution:


```

> javac -classpath ./run_dir/junit-4.12.jar:/run_dir/hamcrest-1.3.jar:/run_dir/json-simple-1.1.1.jar -d . ColoreRGB.java Main.java
> java -classpath ./run_dir/junit-4.12.jar:/run_dir/hamcrest-core-1.3.jar:/run_dir/json-simple-1.1.1.jar Main
Verde(R=0.0, G=1.0, B=0.0) lum.=0.7152
Giallo(R=1.0, G=1.0, B=0.0) lum.=0.9278
Rosso(R=1.0, G=0.0, B=0.0) lum.=0.2126
> 
```