

Studente extends Persona
ogni Studente è una Persona

Qualsiasi classe estende la classe Object, direttamente o indirettamente tramite ereditarietà, e quindi ne eredita i metodi, tipo `toString`, che possono poi essere sovrascritti nella classe specifica.

ARRAY

nomeClasse `arrayOfString[]`;
`arrayOfString = new nomeClasse[4];`
lista dell'array

LISTA
`LinkedList l;` → CREA VARIABILE
`l = new LinkedList();` → CREA OGGETTO
↳ ASSOCIA VAR RIF A OGGETTO

3o ZETTELLONE

LISTA DI PERSONE

Principale.java

```
public class Principale {  
    public static void main (String [] args) {  
        ha già contains(), size(), add(), add (index, elem) ...  
        LinkedList < Persona > e; } può essere fatto su 1 linea  
        e = new LinkedList < Persona > (); ↳ gli elementi possono essere oggetti Persona o oggetti che ESTENDONO Persona (es. Studente)  
        ... la classe Persona è nel file Persona.java, quella Studente in Studente.java  
        for (Persona p : e) {  
            ...  
        }  
        for su una lista elemento per elemento  
    }
```

HASHSET → non esistono indici

Principale.java

```
public class Principale {  
    public static void main (String [] args) {  
        HashSet < Persona > s;  
        s = new HashSet < Persona > ();  
        ...  
    }
```

Se vuoi fare una costruzione per cui viene lanciato eccezione (es. se si prova a settare il nome di una Persona = "")

```
public class Persona {  
    ... ↳ non è un costruttore, è per modificare  
    void setName (String n) {  
        if (n.equals (""))  
            throw new RuntimeException ("errore");  
        this.name = n  
    }
```

3 ottobre

//Principale.java
public class Principale {
 public static void main (String [] args) {
 }
 }
// Persona.java
public class Persona {
 String nome;
 int anno;
 //costruttore implicito
 // public Persona () {}
 // e/o costruttori con argomenti
 public Persona (String x, int y) {
 this.nome = x;
 this.anno = y;
 }
}

↳ costruttore implicito assegnazione ai campi valori di default:
int → 0
float → 0.0
double → 0.0
char → '/0'
boolean → false
class C → null

// Studente.java
public class Studente extends Persona {
 //costruttore con riferimento a super
 public Studente () {}
 super (null, 0);
}

CREAZIONE NUOVO PROGETTO

new → Java Project → nomeProgetto → ↳ DON'T CREATE (?)

new → CLASS → Principale → ↳ main

private rende un campo NON MODIFICABILE

fai un getCampo nella classe del campo per poterlo accedere in lettura

es.

//Principale.java
public class Principale {
 public static void main (String [] args) {
 Automobile a;
 a = new Automobile ("Panda", "Rosso");
 a.getModello (...);
 a.setTarga (...);
 ...
 }

// Automobile.java
public class Automobile {
 int modificaTarga = 0;
 private String targa;
 private String modello;
 String colore;
 Persona persona; ↳ questo costruttore puo' essere ampliato
 public Automobile (String modello, String colore) {
 this.modello = modello;
 this.colore = colore;
 }

public String getTarga () {
 this.targa;
}
public void setTarga (String targa) {
 if (this.modificaTarga == 0)
 this.targa = targa; ↳ voglio che la targa sia
 this.modificaTarga++; modificabile -> VOLTA SOLO
 else
 throw new RuntimeException ("errore");
}
public String getModello () {
 this.modello; ↳ nonostante modello sia
 privato per non farne
 modificare il contenuto,
 voglio poterlo LEGGERE
}

10 ottobre

metodo static NON ha oggetto di invocazione
stipendio, moltiplica(); → NON STATICO
assegnaColore (auto, rosso); → STATICO

10 ottobre
in hashset inserimento in O(1) e ricerca di un elem
tramite MODULO:
elemento cercato % base hashset = indice
ok finché non ci sono troppi conflitti
se usi hashset devi ridefinire equals e hashset
se usi TreeSet devi ridefinire equals e compone
persona implements Comparable { ...

```
...  
@override  
public int compare (Object o){  
    Persona p = (Persona o)  
    if this.anno > p.anno  
        return 1;  
    else if this.anno == p.anno  
        return 0;  
    else return -1;  
}
```

} puoi ordinare come vuoi non per forza per anno

12 ottobre

```
...  
@override  
public int compareTo (ContoCorrente c) {  
    if (this.equals (c)) → quando return 0 devi usare equals, non == !!!  
        return 0;  
    if (this.saldo < c.saldo)  
        return -1;  
    return 1;  
}
```

14 ottobre

```
t = new TreeSet<ContoCorrente>();  
questo TreeSet sarà ordinato secondo la compare descritta in ContoCorrente  
Come posso scoprire un ordinamento diverso? sfrutto una classe che uso solo per contenere un override di compareTo
```

```
// confrontaConto.java  
public class ConfrontaConto  
    implements Comparator<ContoCorrente> {  
    @Override  
    public int compare (ContoCorrente c1,  
                        ContoCorrente c2) {  
        return c1.codice.compareTo(c2.codice);  
    }  
}
```

```
// ContoCorrente.java  
ConfrontaConto c = new ConfrontaConto;  
L'oggetto c è detto FUNTORE, cioè contiene solo un metodo più eventuali parametri che voglio sfruttare  
t = new TreeSet<ContoCorrente>(c);  
questo TreeSet sfrutterà l'oggetto c che gli viene passato per usare un altro tipo di ordinamento, diverso da quello definito in ContoCorrente
```

Potrei avere più classi con più metodi overridden di compareTo in modo da poter creare vari TreeSet ordinati diversamente

Interface

new → project → name : Interface
new → class → name : Principale + Main
new → class (?) → name : Esempio
forse interface?
new → class → name classe → in Interface
fai implements Esempio
↓

//Esempio.java → Tutti gli oggetti che implementano questa interfaccia dovranno il metodo
public interface Esempio {
 int prova (int x);
}
x è di classe Esempio prova

//Principale.java
public class Principale
 public static void main (String [] args) {
 Esempio x;
 x = new Classe ();
 }

//Classe.java
import java.awt.event.ActionListener;
import java.util.LinkedList;
public class Classe
 extends LinkedList
 implements Esempio, ActionListener {
 ↗ quindi avrà il metodo prova

```
@Override  
public int prova (int x) {  
    return 0;  
}
```

17 ottobre

Package/src ↗ minuscolo x convenzione
new package → persone → all'interno si creano le classi es. Persona, Studente ...

getClass() viene usato soprattutto nella ridefinizione del metodo equals()

new class Principale (main) NON voglio metterlo in un package quindi cancello il campo package su eclipse

quando in principale richiamo persona dai estremità perché dal main non si vede l'interno del package. Puoi fare in 2 modi

① personnes.Persona p;
p = new personnes.Persona();

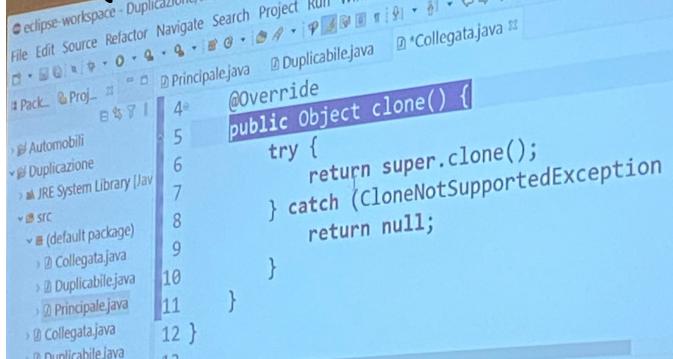
② import personnes.Persona;
...

Personna p = new Personna();
Accedere alla classe da fuori il package adesso è diverso. Gli attributi non sono scrivibili se non sono public.

Proteggere vuol dire che un campo è visibile all'interno di tutte le classi del package in cui è definita la classe in cui è E ANCHE nelle sottoclassi anche se non appartengono al package

19 Ottobre
 le classi copiabili devono implementare
 l'interfaccia cloneable con:
 public class Duplicabile extends Object
 implements Cloneable {
 ...
 @Override
 public Object clone() {
 ...
 }
 }

Collegata

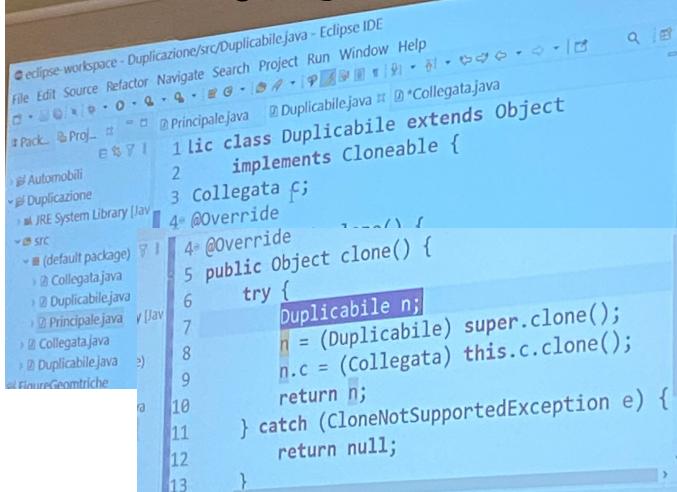


```

    4:     @Override
    5:     public Object clone() {
    6:         try {
    7:             return super.clone();
    8:         } catch (CloneNotSupportedException e) {
    9:             return null;
   10:        }
   11    }
   12  }
   13
  
```

↳ implementa cloneable

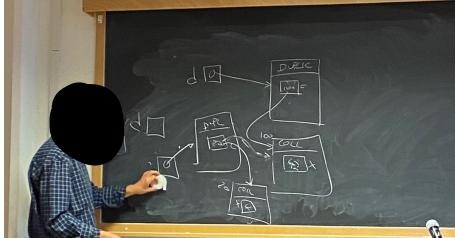
Duplicabile con @override di clone



```

    1: ublic class Duplicabile extends Object
    2:      implements Cloneable {
    3:          Collegata f;
    4:          @Override
    5:          public Object clone() {
    6:              try {
    7:                  Duplicabile n;
    8:                  n = (Duplicabile) super.clone();
    9:                  n.c = (Collegata) this.c.clone();
   10:                 return n;
   11:             } catch (CloneNotSupportedException e) {
   12:                 return null;
   13:             }
   14  }
   15
  
```

Prima fa una clone superficiale, poi
 "va in profondità" e fa una copia
 profonda, ovvero sdoppia la parte
 "COLL" e ne collega ogni doppione
 ad un DUPLIC



21 ottobre
 Sintesi
 eccezioni
 classi generiche

24 ottobre
 Project: GUI

vedi JPanel

// Main.java

```

    import java.awt.*;
    import javax.swing.*;
    public class Main {
        public static void main (String [] args) {
            JFrame finestra = new JFrame ();
            JButton b = new JButton ("ok");
            finestra.add (b);
            JButton c = new JButton ("no");
            finestra.add (c, BorderLayout.SOUTH);
            JTextArea au = new JTextArea (10, 10);
            finestra.add (au, BorderLayout.NORTH);
            a.append ("inizio");
            finestra.pack (); → naturale
            finestra.setVisible (true); → rende visibile
            finestra.setDefaultCloseOperation (
                JFrame.EXIT_ON_CLOSE );
  
```

Agisci o = new Agisci ();
 b.addActionListener (o) in questo oggetto
funziona ci sono
la funzione che
voglio eseguire in
caso di eventi
Questo oggetto
implementerà
ActionListener

AgisciNo n = new AgisciNo ();
 c.addActionListener (n); PASSAGGIO DATI DA
MAIN A LISTENER
 Agisci o = new Agisci ();
 o.texto = au
 b.addActionListener (o)

? { c.addActionListener (o);
 c.addActionListener (n);

System.out.println ("fine main");
 fini serve a inizializzare la finestra ma
 non fa eseguire il programma, che avviene
 e' interazione con l'utente

// Agisci.java

```

    import java.awt.event.ActionEvent;
    import java.awt.event.ActionListener;
    public class Agisci implements ActionListener {
        JTextField testo;
        @Override
        public void actionPerformed (ActionEvent e) {
            System.out.println ("qualcosa");
            this.texto.append ("\n ok");
        }
    }
  
```

puoi fare: "\n" + e.getActionCommand ()
 stampa il testo alle pulsioni

// AgisciNo.java

```

    import java.awt.event.ActionEvent;
    import java.awt.event.ActionListener;
    public class AgisciNo implements ActionListener {
        @Override
        public void actionPerformed (ActionEvent e) {
            System.out.println ("altra ");
        }
    }
  
```

26 ottobre

UML → DIAGRAMMA DELLE CLASSI
Data: esame 20 OTT 2022

```

classDiagram
    class Strumento {
        <<partizione delle sottoclassi>>
    }
    class Macchinario {
        <<ogni macchinario ha 10 + strum.>>
    }
    class Fresa {
        <<1..*>>
    }
    class Manipolazione {
        <<1..*>>
    }
    class Programma {
        <<0..*>>
    }

    Strumento "1..*" --> Macchinario : implicito
    Strumento "1..*" --> Fresa : ordered?
    Strumento "1..*" --> Manipolazione : ordered?
    Macchinario "1..1" --> Programma : non più di una volta
    Macchinario "1..1" --> Programma : ogni programma è un solo macchinario
  
```

Ogni strumento è un insieme di insieme non ordinato può essere poi implementato come un HasUser

28 ottobre slides 1 - Analisi diagrammi Classi UML - 2 up

2 Novembre SANTATO

ne hai una copia
usa quella in Materiale 2022

4 Novembre

Puoi passare all'oggetto Scanner

```

graph TD
    Scanner --> InputStream
    InputStream --> String
    String --> Object
  
```

oggetto FileWriter:

```

FileWriter r;
r = new FileWriter("Phova.txt");
r.write("abcd");
r.close();
  
```

oggetto PrintWriter:

```

PrintWriter p;
p = new PrintWriter(r);
p.println("bacd");
p.flush(); → invia effettivamente il buffer da scrivere, quando questo è pieno, nel canale di rete
//...
p.close();
  
```

7 NOVEMBRE

Rete

//Client.java

```

import java.net.Socket;
public class Client {
    public static void main (String[] args) {
        Socket canale;
        canale = new Socket ("www.google.it", 8080);
        PrintWriter p = new PrintWriter (canale.getOutputStream());
        Scanner s = new Scanner (canale.getInputStream());
        p.print ("GET /\n\n");
        p.print ("\n\n");
        p.print ("doo");
        p.flush();
        String r;
        while (s.hasNextLine()) {
            r = s.nextLine();
            System.out.println(r);
        }
        canale.close();
    }
}

u Client:
1. si connette ad un computer ad una porta corrispondente ad un programma specifico tramite newSocket che prende ritorno una socket
2. legge e scrive sulla socket usando scanner (lettura) e PrintWriter (scrittura)
3. p.println ("A") + FLUSH
4. s.nextLine() → "B"! :)
```

Aggiungi throws exception ...

X IL LOCALE: (string) null
"127.0.0.1"
"localhost"

9 NOVEMBRE

Metodo 1:

```

class Manager {
    Guidazione g;
    Auto a;
}
class Guidazione {
    Manager m;
}
class Auto {
    Manager m;
}
  
```

questo campo NON è accessibile fuori dal package così per gestire il campo di associazione tra auto e guidazione DEVO passare dal manager che owner i suoi metodi

Metodo 2:

classe con costruttore privato :

```

class ES {
    private ES() { ... }
    public static ES metodo() {
        return new ES();
    }
}
```

ES.metodo() viene chiamata (può essere chiamata) fuori dalla classe ES

Nel caso Auto - Guidazione - Manager :

```

class Guidazione {
    private Auto a;
    public Auto getAuto() { return this.auto; }
    public void setAuto(Auto a, Manager m) {
        if (m != null) this.auto = a;
    }
}
class Auto {
    Manager m;
}
class Manager {
    private Manager() { ... }
    static void inserisci(Guidazione g, Auto a) {
        Manager m = new Manager();
        g.setAuto(a, m);
        a.setGuidazione(g, m);
    }
}
```

Solo qui può essere creato il token Manager che poi viene usato per avere accesso ai metodi get e set

in pseudo codice:

```
class Playlist {
    void fired (Event e) {
        switch (this.state) {
            case ATTESA: // STATO ATTESA
                if (e.get class == Play.class) {
                    Play pl = (Play)e;
                    if (e.get Brami().size() != 0) {
                        // PLAYLIST NON VUOTA
                        Player p = pl.get Player();
                        INVIA EVENTO PLAYSONG A P
                        QUESTO FORSE STA QUI
                        p.played(s);
                    }
                }
            case ESECUZIONE:
                ...
        }
    }
}
```

~~QUESTO FORSE STA QUI~~

cou s eventoPlaysong
playsong s = new playsong (this.get Brami ().get(0))

IN GENERALE:

OGNI OGGETTO DINAMICO DEVE AVERE UN METODO DI RICEZIONE, CON UNO SWITCH SULLO STATO. ALL'INTERNO CI SONO IF IN BASE ALL'EVENTO RICEVUTO, SU CUI FARCI IL COST X OTTENERE IL CONTENUTO E Poi ESEGUIRE LE AZIONI DA ESEGUIRE

2 dicembre

3-STATI-TRANSIZIONI-UML-REALIZZAZIONE-2UP

OGGETTO CHE RICEVE UN EVENTO → fired ()
vedi slide

dalle slide

- diagramma delle classi e degli oggetti; **strutturale**
- diagramma delle attività; **comportamentali**
- diagramma degli stati e delle transizioni;

Io un ruolo

Diciamo che una classe C ha responsabilità sull'associazione A, quando, per ogni oggetto x che è istanza di C vogliamo poter eseguire opportune operazioni sulle istanze di A a cui x partecipa, che hanno lo scopo di: **conoscere, aggiungere, cancellare, aggredire**

In particolare, quando una classe C partecipa ad un'associazione A con un vincolo di:

1. molteplicità massima finita, oppure
2. molteplicità minima diversa da zero,

la classe C ha necessariamente responsabilità su A.