

implementare una carta del gioco del poker

Card.java (classe concreta)

```

import java.util.Random;

public class Card {
    // Il valore della carta.
    private final int value;

    // Il seme della carta.
    private final int suit;

    // Genera una carta a caso con un valore da min (incluso) in su.
    // Param: min - il valore minimo (0-12) della carta che può essere generata
    public Card(int min) {      //costruttore → assegna i valori alle proprietà e non ha return
        //ha lo stesso nome della classe e possono essercene più di uno,
        //si differenziano dal numero di parametri che ricevono in input
        Random random = new Random();
        int valore;
        do {
            valore = random.nextInt(bound: 13); //genera un numero random tra 0 e bound escluso
        } while (valore < min);

        value = valore;

        // value = random.nextInt(13 - min) + min;

        suit = random.nextInt(bound: 4);
    }

    // Genera una carta a caso con un valore da 0 (incluso) in su.
    public Card() {
        // this(0);      //si può richiamare il primo costruttore passando il parametro più generico

        Random random = new Random();
        value = random.nextInt(bound: 13);

        suit = random.nextInt(bound: 4);
    }

    public int getValue() { return value; }

    public int getSuit() { return suit; }

    // Ritorna una descrizione della carta sotto forma di stringa, del tipo 10♠ oppure J♥.
    public String toString() {
        String[] valori = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "1"};
        String[] semi = {"♠", "♣", "♦", "♥"};

        return valori[value] + semi[suit];
    }
}

```

```

// Determina se questa carta è uguale ad other.
// Param: other - l'altra carta con cui confrontarsi
// Return: true se e solo se le due carte sono uguali
public boolean equals(Card other) {
    return value == other.value && this.suit == other.suit;      //ritorna il risultato
                                                               della condizione
}

// Ordina le carte, prima in base al valore, poi in base al seme.
// Param: other - l'altra carta con cui confrontarsi
// Return: -1 se la carta è più piccola della carta other, 0 se sono uguali, 1 se la carta è più grande della carta other.
public int compareTo(Card other) {
    if (this.value != other.value) {
        if (this.value > other.value) {
            return 1;
        } else {
            return -1;
        }
    } else {
        if (this.suit != other.suit) {
            if (this.suit > other.suit) {
                return 1;
            } else {
                return -1;
            }
        } else {
            return 0;
        }
    }
}

```

Main.java

Crea una carta *card1* a caso, quindi crea ripetutamente una carta *card2* a caso finché non risulta che *card1* è *equals* con *card2*. A quel punto termina.

Sia *card1* che tutte le *card2* dovranno venire stampate sul video mano che vengono generate.

```

Public class Main {
    public static void main(String[] args) {
        String card1 = (new Card()).toString();
        System.out.println(card1);
        String card2;
        do {
            card2 = new Card().toString();
            System.out.println(card2);
        } while (!card1.equals(card2));

        System.out.println("-----");
        main2();
    }
}

```

```

public static void main2() {
    Card card1 = new Card(min: 12);
    System.out.println(card1);
    System.out.println(card1.toString()); //il toString() nel println è ridondante perché chiama già il toString della classe più vicina

    Card card2 = null;
    do {
        card2 = new Card(min: 12);
        System.out.println(card2);
    } while (!card1.equals(card2));

    System.out.println("-----");
}

main3();
}

public static void main3() { //seconda parte del lab (Card2 - enumerazioni)
    Card2 card1 = new Card2(Value.DUE);
    System.out.println(card1);
    Card2 card2;
    do {
        card2 = new Card2(Value.DUE);
        System.out.println(card2);
    } while (!card1.equals(card2));
}
}

```

Definire due enumerazioni *Value* e *Suit*, che rappresentano rispettivamente il valore delle carte e il loro seme.

Value.java (enumerazione)

```

public enum Value {
    DUE,
    TRE,
    QUATTRO,
    CINQUE,
    SEI,
    SETTE,
    OTTO,
    NOVE,
    DIECI,
    J,
    Q,
    K,
    ASSO;
}

```

Suit.java (enumerazione)

```

public enum Suit {
    PICCHE,
    FIORI,
    QUADRI,
    CUORI;
}

```

Card2.java (classe concreta)

Usa le queste enumerazioni al posto degli interi come valore e seme delle carte.

```
import java.util.Random;

public class Card2 {
    // Il valore della carta.
    private final int value;

    // Il seme della carta.
    private final int suit;

    // Genera una carta a caso con un valore da min (incluso) in su.
    // Param: min - il valore minimo (0-12) della carta che può essere generata
    public Card2(int min) {
        Random random = new Random();
        int valore;
        do {
            valore = random.nextInt(bound: 13);
        } while (valore < min.ordinal()); //ordinal() ritorna l'indice della posizione del "segnaposto"
                                            all'interno dell'enumerazione

        value = (Value.values())[valore]; // values() ritorna a modi array i valori dentro Enum

        /*
        switch (valore) {
            case 0:
                value = Value.DUE;
                break;
            case 1:
                value = Value.TRE;
                break;
            // .....
        }
        */

        // value = random.nextInt(13 - min) + min;
        suit = Suit.values()[random.nextInt(bound: 4)];
    }

    // Genera una carta a caso con un valore da 0 (incluso) in su.
    public Card2() { this(Value.DUE); }

    public Value getValue() { return value; }

    public Suit getSuit() { return suit; }

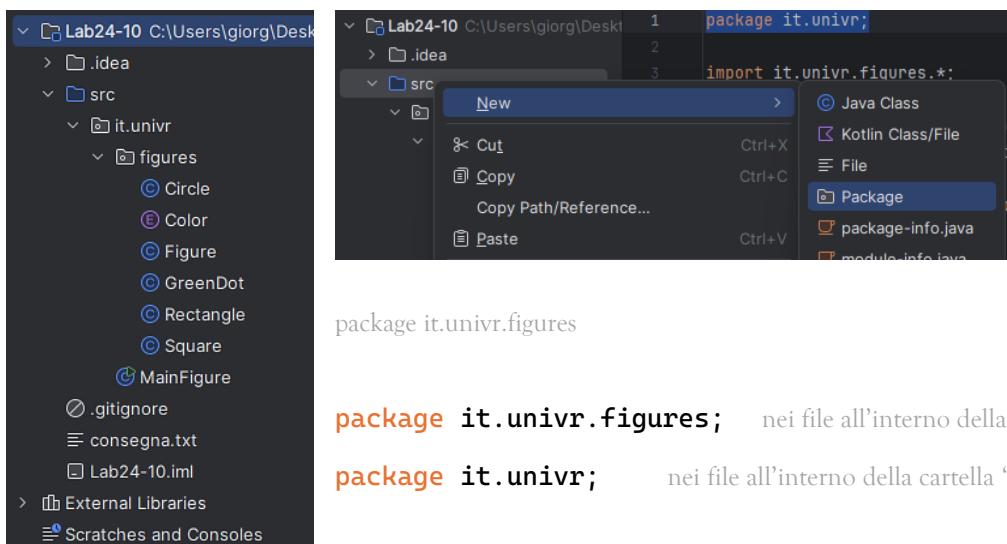
    // Ritorna una descrizione della carta sotto forma di stringa, del tipo 10♠ oppure J♥.
    public String toString() {
        String[] valori = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "1"};
        String[] semi = {"♠", "♣", "♦", "♥"};

        return valori[value.ordinal()] + semi[suit.ordinal()];
    }
}
```

```
// Determina se questa carta è uguale ad other.  
// Param: other - l'altra carta con cui confrontarsi  
// Return: true se e solo se le due carte sono uguali  
public boolean equals(Card other) {  
    return value == value.equals(other.value) && suit.equals(other.suit);  
}  
}
```

`==` → controlla se è lo stesso oggetto (punta alla stessa cella di memoria)

`equals()` → controlla se un oggetto è uguale a un altro



Color.java (enumerazione)

```
package it.univr.figures;

public enum Color {
    GIALLO,
    ROSSO,
    BLU,
    VERDE,
    NERO;
}
```

Figure.java (classe concreta)

Rappresenta una figura geometrica colorata.

```
package it.univr.figures;

public class Figure {
    private Color colore;

    public Figure (Color colore) { this.colore = colore; }

    public double perimeter() { return 0; }

    public double area() { return 0; }

    @Override      //indica che si sta andando a sovrascrivere il metodo di una classe superiore
    public String toString() {
        return "area: " + this.area() + ", perimeter: " + this.perimeter() +
               ", color: " + colore;
    }

    protected Color getColore() { return colore; }
}
```

Rectangle.java (classe concreta)

Rectangle di Figure, rappresenta un rettangolo.

```
package it.univr.figures;

public class Rectangle extends Figure{      //extends → Rectangle è figlia di Figure
    private double base;
    private double altezza;

    public Rectangle(Color colore, double base, double altezza) {
        super(colore);   //richiama il costruttore della superclasse e assegna il valore alla rispettiva proprietà,
                          //già presente nella superclasse (quindi non si deve ri-istanziare nella sottoclasse).
                          //va richiamato nella prima riga di codice del costruttore,
                          //se non si scrive, è implicito super();
        this.base = base;
        this.altezza = altezza;
    }

    @Override
    public double perimeter() { return (base + altezza)*2; }

    @Override
    public double area() { return base*altezza; }

    @Override
    Ridefinito per ritornare la stringa "Rectangle of " seguita dalla chiamata al toString() della superclasse.
    public String toString() { return "Rectengle of " + super.toString(); }
}
```

Square.java (classe concreta)

Square di Rectangle, rappresenta un quadrato. I metodi double *perimeter()* e double *area()* non vengono ridefiniti.

```
package it.univr.figures;

public class Square extends Rectangle{

    public Square(Color colore, double lato) { super(colore, lato, lato); }

    @Override
    public String toString() { return "Square, a " + super.toString(); }
}
```

Circle (classe concreta)

Circle di Figure, rappresenta un cerchio.

```
package it.univr.figures;

import it.univr.MainFigure;

public class Circle extends Figure{
    private double raggio;

    public Circle(Color colore, double raggio) {
        super(colore);
        this.raggio = raggio;
    }

    @Override
    public double perimeter() { return 2*Math.PI*raggio; }

    @Override
    public double area() { return Math.PI*raggio*raggio; }

    @Override
    public String toString() { return "Circle of " + super.toString(); }
}
```

GreenDot.java (classe concreta)

GreenDot di Circle, rappresenta un cerchio di raggio 1 e di colore verde.

```
package it.univr.figures;

public class GreenDot extends Circle{
    public GreenDot() { super(Color.VERDE, 1); }
}
```

MainFigure.java

```
package it.univr;

import it.univr.figures.*;

public class MainFigure {
    public static void main(String[] args) {
        Figure rettangolo = new Rectangle(Color.BLU, base: 10, altezza: 11);
        Figure quadrato = new Square(Color.BLU, lato: 10);
        print(rettangolo);

        Figure cerchio = new Circle(Color.GIALLO, raggio: 5);
        print(cerchio);

        //in questa classe non è possibile chiamare il metodo getColor() sulle figure,
        //perché il metodo è protected e la classe main non si trova nella stessa cartella del file che contiene il metodo
        //getColor() da richiamare
    }

    public static void print(Figure figure) {          //l'oggetto più generico può contenere gli oggetti più
                                                    //specifici, quindi posso passare Rectangle, che è una
                                                    //sottoclasse di Figure,
        if (figure instanceof Rectangle)           //ma viene poi utilizzata sotto il tipo di Figure,
                                                    //per averla sotto il tipo di Rectangle, dopo aver controllato che
                                                    //effettivamente è un'istanza di Rectangle,
                                                    //si può castare, ovvero creare una nuova variabile, definendola col
                                                    //tipo più specifico,
                                                    //oppure si può dire di considerarla del tipo più specifico,
                                                    //a sì deve fare ogni volta che si utilizza
            return;
        System.out.println(figure);
    }
}
```

SoccerPlayer.java (**interfaccia**) comprende tutti i metodi di una classe, senza implementarli

Specifica un giocatore di calcio

```
public interface SoccerPlayer {
    String toString(); // ritorna il nome del giocatore
    boolean canUseHands(); // determina se il giocatore può usare le mani
}
```

AbstractSoccerPlayer.java (**classe astratta**) mette insieme la classe concreta e l'*interfaccia*, comprende quindi metodi già implementati, e metodi non implementati (da implementare nelle sottoclassi), contrassegnati dalla parola **abstract**

```
public abstract class AbstractSoccerPlayer implements SoccerPlayer { //implements →
    private String name;
    protected AbstractSoccerPlayer(String name) { this.name = name; }

    @Override
    public final String toString() { return name; }

    public abstract boolean canUseHands(); //metodo da implementare
}
```

Sottoclassi *Forward*, *Midfield*, *Defence* e *GoalKeeper*, (solo il *GoalKeeper* può usare le mani). ...

Forward.java (classe concreta)

```
public class Forward extends AbstractSoccerPlayer{  
  
    protected Forward(String name) { super(name); } //anche il costruttore della superclasse è  
    //protected, cambiando visibilità non si va a  
    //sovrascrivere, ma se ne crea uno nuovo  
  
    @Override  
    public boolean canUseHands() { return false; }  
}
```

Midfield.java (classe concreta)

```
public class Midfield extends AbstractSoccerPlayer{  
  
    protected Midfield(String name) { super(name); }  
  
    @Override  
    public boolean canUseHands() { return false; }  
}
```

Defence.java (classe concreta)

```
public class Defence extends AbstractSoccerPlayer{  
  
    protected Defence(String name) { super(name); }  
  
    @Override  
    public boolean canUseHands() { return false; }  
}
```

GoalKeeper.java (classe concreta)

```
public class GoalKeeper extends AbstractSoccerPlayer{  
  
    protected GoalKeeper(String name) { super(name); }  
  
    @Override  
    public boolean canUseHands() { return true; }  
}
```

Formation.java (classe concreta)

```
public class Formation {  
  
    private SoccerPlayer[] players;  
  
    public Formation(SoccerPlayer[] players) {  
        this.players = players;  
        if (!isValid())  
            throw new IllegalArgumentException("invalid formation"); //lancia un errore (rosso)
```

```

}

// ritorna true se e solo se la formazione è fatta da 11 giocatori, di cui esattamente uno è un portiere
protected boolean isValid() {
    int portieri = 0;

    /*
    for (int i = 0; i < players.length; i++) {
        if (players[i].canUseHands()) {
            portieri++;
        }
        if (portieri > 1) {
            return false;
        }
    }
    */

    for (SoccerPlayer player : players) { //for each → SoccerPlayer player = players[i];
        //viene istanziata una variabile, in questo caso di tipo
        //SoccerPlayer
        //bisogna prendere il tipo della proprietà della singola
        //cella dell'array → ad esempio con String si prende char),
        //che passa una a una ogni cella dell'array
        if (player.canUseHands()) {
            portieri++;
        }
        if (portieri > 1) {
            return false;
        }
    }

    return players.length == 11 && portieri == 1;
}

// ritorna i giocatori di questa formazione
protected SoccerPlayer[] getPlayers() { return this.players; }

@Override
// ritorna i nomi dei giocatori della formazione, separati da virgola
public final String toString() {
    String giocatori = "";

    for (int i = 0; i < players.length; i++) {
        if (i == 0) {
            giocatori = players[i].toString();
        } else {
            giocatori = giocatori + ", " + players[i].toString();
        }
    }

    return giocatori;
}
}

```

Formation433.java (classe concreta)

Sottoclasse concreta di *Formation*.

```
public class Formation433 extends Formation{

    public Formation433(SoccerPlayer[] players) { super(players); }

    // ritorna true se e solo se la formazione è fatta da 11 giocatori,
    // di cui 4 difensori, 3 centrocampisti e 3 attaccanti, e un portiere
    protected boolean isValid() {
        if (!super.isValid()) { return false; }

        int difensori = 0;
        int centrocampisti = 0;
        int attaccanti = 0;

        /*
        for (SoccerPlayer player : getPlayers()) {
            if (player instanceof Defence) {
                difensori++;
                if (difensori > 4) { return false; }
            } else if (player instanceof Midfield) {
                centrocampisti++;
                if (centrocampisti > 3) { return false; }
            } else if (player instanceof Forward) {
                attaccanti++;
                if (attaccanti > 3) { return false; }
            }
        }
        */

        return true;
    }

    for (SoccerPlayer player : getPlayers()) {
        if (player instanceof Defence) {
            difensori++;
        } else if (player instanceof Midfield) {
            centrocampisti++;
        } else if (player instanceof Forward) {
            attaccanti++;
        }
    }

    return difensori == 4 && centrocampisti == 3 && attaccanti == 3;
}
```

Main.java

```
import java.util.LinkedList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        String[] difensori = {"Alex Sandro", "Rugani", "Chiellini", "Dani Alves"};
        String[] centrocampisti = {"Fabinho", "Iniesta", "Pjanic"};
        String[] attaccanti = {"Dybala", "Higuain", "Bernardeschi"};
        String[] portiere = {"Szczesny"};
        String[][] giocatori = {difensori, centrocampisti, attaccanti, portiere};

        SoccerPlayer[] players = new SoccerPlayer[12]; //array di SoccerPlayer
                                                        con dimensione definita
                                                        //12 → test per vedere se fallisce
                                                        (dimensione valida → 11)
        List<SoccerPlayer> playerList = new LinkedList<SoccerPlayer>(); //LinkedList
        test(playerList);           //LinkedList

        int index = 0;
        for (int i = 0; i < giocatori.length; i++) {
            for (int j = 0; j < giocatori[i].length; j++) {
                switch (i) {
                    case 0:
                        players[index] = (new Defence (difensori[j]));
                        index++;
                        break;
                    case 1:
                        players[index] = (new Midfield (centrocampisti[j]));
                        index++;
                        break;
                    case 2:
                        players[index] = (new Forward (attaccanti[j]));
                        index++;
                        break;
                    case 3:
                        players[index] = (new Goalkeeper (portiere[j]));
                        index++;
                        break;
                }
            }
        }

        players[index] = (new Goalkeeper("Tizio")); //test per vedere se fallisce
```

```

//try - catch - finally
try { //prova il pezzo di codice all'interno,
    se durante l'esecuzione vengono generate delle eccezioni all'interno del try,
    e sono catturabili da uno dei catch, esegui il catch corrispondente all'eccezione (errore)
    System.out.println(new Formation433(players).isValid());
    System.out.println(new Formation433(players));
} catch (ExceptionInInitializerError e) { //esegue in catch sse l'eccezione (errore)
    generata è del tipo che riceve come parametro
    o sua "figlia"
    System.out.println("Errore nella creazione della formazione.");
    //System.exit(-1); // -1 → codice di uscita con errore
} catch (IllegalArgumentException e) {
    System.out.println(e);
    //System.exit(-1);
} finally { //a prescindere esegui,
    se non viene terminata prima l'esecuzione (System.exit, errori non catturati, return, ...)
    System.out.println("Try catch finito.");
    System.exit(-1);
}
}

public static void test (List<SoccerPlayer> playerList) {} //LinkedList
//riesce a entrare perché la classe LinkedList è figlia
dell'interfaccia List
//e il metodo è static come il main
}

```

Number.java (interfaccia)

```
public interface Number extends Comparable<Number> { //Comparable<T>
    int getValue(); // restituisce il valore di questo numero
}
```

AbstractNumber.java (classe astratta)

Implementazione astratta di un Number. Fornisce le funzionalità comuni a tutti i numeri.

```
public abstract class AbstractNumber implements Number {
    private final int value;

    // se value è negativo, esegue throw new IllegalArgumentException(); altrimenti inizializza il campo value
    protected AbstractNumber(int value) {
        if (value < 0) {
            throw new IllegalArgumentException();
        }
        this.value = value;
    }

    // restituisce il valore di questo numero
    public final int getValue() { return this.value; }

    // restituisce la base di numerazione di questo numero
    protected abstract int getBase();

    // restituisce il carattere che rappresenta la cifra "digit" nella base di numerazione
    // di questo numero. Sarà sempre vero che 0 <= digit < getBase();
    // per esempio, in base sedici si avrà getCharForDigit(10) == 'A' e
    // in base otto si avrà getCharForDigit(7) == '7'
    protected abstract char getCharForDigit(int digit);

    // restituisce una stringa che rappresenta il numero nella sua base di numerazione
    @Override
    public String toString() {
        String string = "";
        int val = this.value;
        do {
            string = getCharForDigit(val % getBase()) + string;
            val = val/getBase();
        } while (val > 0);

        return string;
    }
}
```

```

// due numeri sono uguali se e solo se hanno lo stesso valore
@Override
public final boolean equals(Object other) {
    if (!(other instanceof Number)) { //controlla se other è istanza di Number,
        // se non lo è si sa già che non è uguale
        return false;
    }

    //other = (Number) other;    NO → ridondante perché other resta di tipo Object

    //((Number) other).getValue(); scrivere ogni volta il tipo da considerare

    Number otherNumb = (Number) other; //cast
    return this.value == otherNumb.getValue();
}

// ordinamento fra i Number è quello crescente per valore
@Override
public final int compareTo(Number other) {
    if (this.value < other.getValue()) {
        return -1; //ritorna -1 se this < other
    } else if (this.value == other.getValue()) {
        return 0; //ritorna 0 se this = other
    } else {
        return 1; //ritorna 1 se this > other
    }
}
}

```

Sottoclassi concrete *DecimalNumber*, *BinaryNumber*, *OctalNumber*, *HexNumber* e *Base58Number* di *AbstractNumber*.
Il metodo *toString()* ereditato da *AbstractNumber* funziona per tutte queste sottoclassi.

DecimalNumber.java (classe concreta)

```

public class DecimalNumber extends AbstractNumber{
    public DecimalNumber(int value) { super(value); }

    @Override
    protected int getBase() { return 10; }

    @Override
    protected char getCharForDigit(int digit) {
        return string.charAt(digit); //ritorna il char (della stringa string) che corrisponde all'indice in input (digit)
    }

    private static final String string = "0123456789";
}

```

BinaryNumber.java (classe concreta)

```
public class BinaryNumber extends AbstractNumber{
    public BinaryNumber(int value) { super(value); }

    @Override
    protected int getBase() { return 2; }

    @Override
    protected char getCharForDigit(int digit) { return string.charAt(digit); }

    private static final String string = "01";
}
```

OctalNumber.java (classe concreta)

```
public class OctalNumber extends AbstractNumber{
    public OctalNumber(int value) { super(value); }

    @Override
    protected int getBase() { return 8; }

    @Override
    protected char getCharForDigit(int digit) { return string.charAt(digit); }

    private static final String string = "012345678";
}
```

HexNumber.java (classe concreta)

```
public class HexNumber extends AbstractNumber{
    public HexNumber(int value) { super(value); }

    @Override
    protected int getBase() { return 16; }

    @Override
    protected char getCharForDigit(int digit) { return array[digit]; }

    private static final char[] array =
        {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
}
```

Base58Number.java (classe concreta)

```
public class Base58Number extends AbstractNumber{
    public Base58Number(int value) { super(value); }

    @Override
    protected int getBase() { return 58; }

    @Override
    protected char getCharForDigit(int digit) { return string.charAt(digit); }

    private static final String string =
        "123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
}
```

BinaryNumberWithParity.java (classe concreta)

Sottoclasse concreta di *BinaryNumber*.

Il numero binario viene esteso con un'ulteriore cifra binaria di controllo, in modo da rendere pari il numero totale di cifre 1: se la quantità pari di 1, si aggiungerà uno 0, sennò si aggiungerà un 1.

```
public class BinaryNumberWithParity extends BinaryNumber{
    public BinaryNumberWithParity(int value) {
        super(value);
    }

    @Override
    public String toString(){
        String string = super.toString();
        int counter1 = 0;

        for (int i = 0; i < string.length(); i++) {
            //char c = string.charAt(i);
            if (string.charAt(i) == '1') {
                counter1++;
            }
        }
        /*
        for (char stringa : string) {      //non si può fare il for each sulle stringhe
            if (stringa == 1) {
                counter1++;
            }
        }
        */

        return string + (counter1 % 2);    //se dispari il resto è 1, senno è 0
                                            //viene concatenato alla stringa del numero binario già esistente
    }
}
```

MainNumbers.java

Chiede all'utente di inserire un numero non negativo n, quindi crea il numero in base 10, 2, 2 con parità, 8, 16 e 58.

```
import java.util.Scanner;

public class MainNumbers {
    public static void main(String[] args) {
        System.out.print("inserire un numero non negativo: ");
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();

        System.out.println(new DecimalNumber(n) + "\n" + new BinaryNumber(n) + "\n" +
                           new BinaryNumberWithParity(n) + "\n" + new OctalNumber(n) + "\n" +
                           new HexNumber(n) + "\n" + new Base58Number(n));
    }
}
```

MainNumbersSort.java

Crea un array[6] di Number:

2024 in base 10

113 in base 2

158 in base 2 con parità

827 in base 8

2066 in base 16

8092 in base 58

```
import java.util.Arrays;  
  
public class MainNumbersSort {  
    public static void main(String[] args) {  
        Number[] array = {new DecimalNumber(2024), new BinaryNumber(113),  
            new BinaryNumberWithParity(158), new OctalNumber(827), new HexNumber(2066),  
            new Base58Number(8092)};  
  
        Arrays.sort(array); //ordina l'array, secondo il metodo compareTo (ordinamento crescente) → Comparable<T>  
        System.out.println(Arrays.toString(array)); // [1110001, 100111101, 1473, 2024, 812, 3QX]  
                                                // qui non è ridondante perché bisogna chiamare il  
                                                // toString degli array  
    }  
}
```

List.java (classe concreta)

Rappresenta una lista non vuota di T (può essere rimpiazzato con qualsiasi tipo (*obj*))
 (≠ da interfaccia List, o dalle classi *LinkedList*, *ArrayList*, ...)

```
package it.univr.lists;

import java.io.IOException;
import java.io.PrintWriter;

public class List<T> {
    private final T head;          //testa della lista
    private final List<T> tail;     //il resto della lista

    // crea una lista con la testa e la coda indicate
    public List(T head, List<T> tail) {
        this.head = head;           //testa
        this.tail = tail;           //elemento successivo (next)
    }

    // crea una lista contenente la testa indicata, seguita dagli elementi indicati
    public List(T head, T... elements) { //passa in input in numero indefinito di elements di tipo T
        this.head = head;           //assegnata la testa ("value")

        /*
         T[] elementi = new T[elements.length]; //non si può istanziare un array di tipo T
         for (int i = 0; i < elements.length-1; i++) {
             elementi[i] = elements[i+1];
         }
         this.tail = new List<T>(elements[0], elementi);
        */

        List<T> list = null;

        for (int i = (elements.length - 1); i >= 0 ; i--) { //metodo ricorsivo
            //abbiamo già la testa (this.head = head);
            //parte dall'ultimo elemento della lista, costruendo ogni nodo
            //tramite il primo costruttore, e salvata mano a mano in list
            list = new List<T>(elements[i], list); //prende in input "testa" e "elemento successivo"
        }

        this.tail = list;           //alla fine "collega" la testa alla coda
        (la coda è l'elemento successivo ("next"))
    }

}
```

```
public class List<T> { 10 usages 1 inheritor ▾ GiorgiaZanini
    private final List<T> tail; 10 usages          tail: null

    // crea una lista con la testa e la coda indicate
    public List(T head, List<T> tail) { 10 usages ▾ GiorgiaZanini      head: "are"      tail: "you?"
        this.head = head;  head: "are"      head: "are"
        this.tail = tail; tail: "you?"    tail: null
    }
```

```

// restituisce una descrizione di questa lista, fatta dai toString()
// dei suoi elementi separati da virgole
public String toString() {
    String string = head.toString();

    if (tail == null)
        return string;

    List<T> tempTail = tail;
    while (tempTail != null) {
        string = string + ", " + tempTail.head;      //aggiunge il valore del nodo alla stringa
        tempTail = tempTail.tail;                  //passa al nodo successivo
    }

    return string;
}

// restituisce il numero di elementi di questa lista
public int length() {
    int counterNodes = 1;
    if (tail == null)
        return counterNodes;          //se c'è solo un nodo ritorna i counter dei nodi a 1

    List<T> tempTail = tail;
    while (tempTail != null) {
        counterNodes++;           //conta il nodo corrente
        tempTail = tempTail.tail;   //passa al nodo successivo
    }

    return counterNodes;
}

// scrive gli elementi di questa lista (cioè il loro toString())
// dentro il file testuale col nome indicato (un PrintWriter vi aiuterà)
public void dump(String fileName) throws IOException{           //dump = "buttare fuori"
    PrintWriter printWriter = new PrintWriter(fileName);          //In questo caso scrivere su file
                                                               //struttura dati salvata nella ram)
                                                               //per sola scrittura su file

    printWriter.print(this.head + " ");
    List<T> tempTail = tail;
    while (tempTail != null) { //come il toString(), cicla sui nodi e li scrive mano mano
        printWriter.print(tempTail.head + " ");
        tempTail = tempTail.tail;
    }

    printWriter.close();    //chiude il file (in scrittura)
}

```

IntList.java

Rappresenta una lista di interi (sottoclasse di List<T>)

```
package it.univr.lists;

import java.io.FileReader;
import java.io.IOException;
import java.util.NoSuchElementException;
import java.util.Scanner;

public class IntList extends List<Integer> {

    public IntList(Integer head, IntList tail) { super(head, tail); }
    public IntList(Integer head, Integer... elements) { super(head, elements); }

    // restituisce una lista di interi letta dal file testuale indicato;
    // in caso di errore di lettura, lancia una IOException; uno Scanner vi aiuterà
    public static IntList readFrom(String fileName) throws IOException {
        Scanner scanner = new Scanner(new FileReader(fileName)); //crea un oggetto Scanner che
                                                               legge da file
        //return readFrom(scanner);
        try {
            return new IntList(scanner.nextInt(), readFrom(scanner)); // **
        } catch (NoSuchElementException e) { // *
            throw new IOException(e);
        }
    }

    private static IntList readFrom(Scanner scanner) throws IOException {
        try {
            //return new IntList(scanner.nextInt(), scanner.hasNextInt() ?
            readFrom(scanner) : null); //new IntList(1, new IntList(2, new IntList(3, null)));
                                         //soluzione, ricorsiva, ma ritorna, il primo parametro (head) al
                                         costruttore, prima di fare il controllo scanner.hasNextInt(),
                                         (quindi ritorna un errore, perché non controlla il primo ma
                                         ritorna direttamente, e poi fa il controllo sui successivi),
                                         e fa il controllo sul secondo parametro (tail)
                                         //aggiungo il ritorno del primo senza controllo nel readFrom(String) *
                                         //soluzione → ritorna il primo parametro (scanner.nextInt()),
                                         poi per il secondo (tail) →
                                         if il file ha un intero successivo (scanner.hasNextInt() ?)
                                         se è true, ritorna leggi (readFrom(scanner)),
                                         sennò ( : ) ritorna null (null)
            //ricorsivo → implementata la soluzione per esteso **
            if (scanner.hasNextInt()) { //prima di leggere, controlla se quello che andrà a leggere è un intero
                return new IntList(scanner.nextInt(), readFrom(scanner));
            } else {
                return null;
            }

        } catch (NoSuchElementException e) { //se l'elemento letto non è un intero ...
            throw new IOException(e); //... lancia un'eccezione (di tipo IOException(e))
        }
    }
}
```

Main.java

```
package it.univr.lists;

import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        try {
            List<String> l1 = new List<String>(head: "hello", ...elements: "how", "are", "you?");
            System.out.println(l1 + " di lunghezza " + l1.length());
            l1.dump(fileName: "l1.txt");

            IntList l2 = new IntList(head: 11, ...elements: 13, 42, 9, -5, 17, 13);
            System.out.println(l2 + " di lunghezza " + l2.length());
            l2.dump(fileName: "l2.txt");

            IntList l3 = IntList.readFrom(fileName: "l2.txt");
            System.out.println(l3 + " di lunghezza " + l3.length());

            IntList.readFrom(fileName: "l1.txt"); // fallisce perché l1.txt contiene stringhe, non interi
        }
        catch (IOException e) {
            //System.out.println(e);
            System.out.println("Errore di I/O");
        }
    }
}
```

Output main:

```
hello, how, are, you? di lunghezza 4
11, 13, 42, 9, -5, 17, 13 di lunghezza 7
11, 13, 42, 9, -5, 17, 13 di lunghezza 7
Errore di I/O
```

Lab 6 6/12

```

1 package it.univr.cards;
2
3 public enum Value { 9 usages
4     DUE, 2 usages
5     TRE, no usages
6     QUATTRO, no usages
7     CINQUE, 1 usage
8     SEI, no usages
9     SETTE, 1 usage
10    OTTO, no usages
11    NOVE, no usages
12    DIECI, no usages
13    J, no usages
14    Q, no usages
15    K, no usages
16    ASSO; no usages
17 }

```



```

1 package it.univr.cards;
2
3 public enum Suit { 3 usages ± GiorgiaZanini
4     PICCHE, no usages
5     FIORI, no usages
6     QUADRI, no usages
7     CUORI; no usages
8 }
9

```

```

1 package it.univr.cards;
2
3 > import ...;
4
5
6 public class Deck { 6 usages ± GiorgiaZanini
7     private SortedSet<Card> deck = new TreeSet<Card>(); 5 usages
8
9
10    | Costruisce un mazzo di size carte, tutte diverse, il cui valore è da min in su.
11
12    public Deck(int size, Value min) { 6 usages ± GiorgiaZanini
13        boolean aggiunto;
14        for (int i = 0; i < size; i++) {
15            do {
16                | aggiunto = deck.add(new Card(min));
17            } while (!aggiunto);
18        }
19    }
20
21
22    | Restituisce una stringa che descrive le carte in questo mazzo, costruita come la loro stampa in
23    | sequenza crescente per valore, separate da virgola, con parentesi quadre agli estremi.
24 @↑
25
26    public String toString() { ± GiorgiaZanini
27        String stringDeck = "[";
28        for (Card card : deck) {
29            if (!(card.equals(deck.last())))
30                | stringDeck += (card + ", ");
31            else
32                | stringDeck += card;
33        }
34        return stringDeck + "]";
35    }
36
37
38    | Restituisce la carta più alta di questo mazzo (senza rimuoverla).
39    > public Card best() { return deck.last(); }
40
41
42    | Rimuove la carta più alta di questo mazzo.
43    > public void removeBest() { deck.remove(best()); }
44
45
46
47
48 }

```

```
© Suit.java   © Value.java   © Card.java ×

1 package it.univr.cards;
2
3 import java.util.Random;
4
5 public class Card implements Comparable<Card>{ 14 usages  ± GiorgiaZanini
6     // Il valore della carta.
7     private final Value value; 10 usages
8
9     // Il seme della carta.
10    private final Suit suit; 10 usages
11
12    // Genera una carta a caso con un valore da min (incluso) in su.
13    Params: min – il valore minimo (0-12) della carta che può essere generata
14
15    @
16    public Card(Value min) { 10 usages  ± GiorgiaZanini
17        Random random = new Random();
18        value = Value.values()[random.nextInt( bound: 13 - min.ordinal()) + min.ordinal()];
19        suit = Suit.values()[random.nextInt( bound: 4)];
20    }
21
22    // Genera una carta a caso con un valore da 0 (incluso) in su.
23
24    > public Card() { this(Value.DUE); }
25
26    > public Value getValue() { return value; }
27
28    > public Suit getSuit() { return suit; }
29
30    // Ritorna una descrizione della carta sotto forma di stringa, del tipo 10♣ oppure J♥.
31
32    @
33    public String toString() {  ± GiorgiaZanini
34        String[] valori = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "1"};
35        String[] semi = {"♠", "♦", "◆", "♥"};
36
37        return valori[value.ordinal()] + semi[suit.ordinal()];
38    }
39
40    @
41    public boolean equals(Object other) {  ± GiorgiaZanini
42        if (!(other instanceof Card))
43            return false;
44
45        return value.equals(((Card) other).value) && suit.equals(((Card) other).suit);
46    }
47
48    @Override  ± GiorgiaZanini
49    public int hashCode() {
50        //return 0;
51        return (value.ordinal() * 10) + suit.ordinal();
52    }
53
54    @Override  ± GiorgiaZanini
55    public int compareTo(Card other) {
56        if (value.ordinal() < other.value.ordinal())
57            return -1;
58        else if (value.ordinal() > other.value.ordinal())
59            return 1;
60
61        if (suit.ordinal() < other.suit.ordinal())
62            return -1;
63        else if (suit.ordinal() > other.suit.ordinal())
64            return 1;
65
66        return 0;
67    }
68
69
70
71
72    @
73    public int compareTo(Card other) {
74        if (value.ordinal() < other.value.ordinal())
75            return -1;
76        else if (value.ordinal() > other.value.ordinal())
77            return 1;
78
79        if (suit.ordinal() < other.suit.ordinal())
80            return -1;
81        else if (suit.ordinal() > other.suit.ordinal())
82            return 1;
83
84        return 0;
85    }
86
87
88
89
90
91
92
93
94
95
96
```

```
⑥ Suit.java      ⑥ Value.java      ⑥ Card.java      ⑥ MainCards.java ×

1 package it.univr.cards;
2
3 import java.util.HashSet;
4 import java.util.TreeSet;
5
6 public class MainCards { ± GiorgiaZanini
7     public static void main(String[] args) { ± GiorgiaZanini
8         HashSet<Card> hashSet = new HashSet<>(); //1733942564908 ms
9                                         //con hashCode costante: 1733942828775 ms
10        TreeSet<Card> treeSet = new TreeSet<>(); //1733942481488 ms
11                                         //con hashCode costante: 1733942739247 ms
12
13        int counterHashSet = mainHashSet(hashSet);
14        int counterTreeSet = mainTreeSet(treeSet);
15
16        System.out.println("dimensione hashSet : " + counterHashSet);
17        System.out.println("dimensione treeSet : " + counterTreeSet);
18        if (hashSet.equals(treeSet))
19            System.out.println("i due insiemi sono uguali (equals())");
20        else
21            System.out.println("i due insiemi sono diversi");
22        System.out.println("tempo richiesto : " + System.currentTimeMillis() + " ms");
23    }
24
25    public static int mainHashSet(HashSet hashSet) { 1 usage ± GiorgiaZanini
26        boolean aggiunto;
27        int counterHashSet = 0;
28
29        for (int counterCards = 0; counterCards < 100000000; counterCards++) {
30            aggiunto = hashSet.add(new Card());
31            if (aggiunto)
32                counterHashSet++;
33        }
34        System.out.println("hashSet: " + hashSet);
35
36        return counterHashSet;
37    }
38
39    public static int mainTreeSet(TreeSet treeSet) { 1 usage ± GiorgiaZanini
40        boolean aggiunto;
41        int counterTreeSet = 0;
42
43        for (int counterCards = 0; counterCards < 100000000; counterCards++) {
44            aggiunto = treeSet.add(new Card());
45            if (aggiunto)
46                counterTreeSet++;
47        }
48        System.out.println("treeSet: " + treeSet);
49
50        return counterTreeSet;
51    }
52}
53}
```

```
④ Suit.java      ④ Value.java      ④ Card.java      ④ MainCards.java      ④ MainDeck.java ×

1 package it.univr.cards;
2
3 ▷ public class MainDeck { ± GiorgiaZanini
4 ▷     public static void main(String[] args) { ± GiorgiaZanini
5         Deck deck1 = new Deck( size: 5,Value.SETTE);
6         System.out.println(deck1);
7         System.out.println("best of deck1: " + deck1.best());
8         System.out.print("deck1 without " + deck1.best() + ": ");
9         deck1.removeBest();
10        System.out.println(deck1);
11
12        System.out.println();
13
14        Deck deck2 = new Deck( size: 5,Value.CINQUE);
15        System.out.println(deck2);
16        System.out.println("best of deck2: " + deck2.best());
17
18        System.out.println();
19
20        Deck deck3 = new Deck( size: 52,Value.DUE);    // 52 -> dimensione di un deck completo
21        System.out.println(deck3);
22        System.out.println("best of deck3: " + deck3.best());
23    }
24}
25
```

Lab 7

The screenshot shows a Java code editor with two tabs open: `Genre.java` and `Book.java`.

Genre.java:

```
1 package it.univr.library;
2
3 /**
4  * I generi dei libri.
5  */
6 public enum Genre { 14 usages ▲ GiorgiaZanini
7     COMPUTING, 1 usage
8     FICTION, 3 usages
9     GUIDE, 1 usage
10    HISTORY 4 usages
11 }
```

Book.java:

```
1 package it.univr.library;
2
3 import java.util.Comparator;
4
5 /**
6  * Un libro.
7  */
8 @public abstract class Book implements Comparable<Book> { 35 usages 2 inheritors ▲ GiorgiaZanini *
9
10    private String title = ""; 10 usages
11    private String author= ""; 10 usages
12    private int year = -1; 8 usages
13    private Genre genre = null; 4 usages
14
15 /**
16  * Crea un libro.
17  *
18  * Params: title – il titolo del libro
19  *          author – l'autore del libro
20  *          year – l'anno di pubblicazione del libro
21  *          genre – il genere del libro
22  *
23  * Throws: IllegalArgumentException – se qualche parametro e' null
24
25    protected Book(String title, String author, int year, Genre genre) { 4 usages ▲ GiorgiaZanini *
26        try {
27            if (title != null)
28                this.title = title;
29
30            if (author != null)
31                this.author = author;
32
33            if (year < 0)
34                throw new IllegalArgumentException();
35            this.year = year;
36
37            if (genre != null)
38                this.genre = genre;
39
40        } catch (NullPointerException e) {
41            throw new IllegalArgumentException();
42        }
43    }
44 }
```

The code editor interface includes tabs for other files like `AudioBook.java`, `Book.java`, `Corso.java`, `Studente.java`, `Catalog.java`, and `Catalo`. It also shows line numbers and some annotations like `▲ GiorgiaZanini` and `*`.

[AudioBook.java](#)[Book.java](#) ×[Catalog.java](#)[CatalogWithStatistics.java](#)[PaperBook.java](#)[Main.java](#)

```
8     public abstract class Book implements Comparable<Book> { 35 usages 2 inheritors ▾ GiorgiaZanini
43
44         /**
45          * Restituisce il titolo del libro.
46         */
47         > public String getTitle() { return title; }
48
49         /**
50          * Restituisce l'autore del libro.
51         */
52         > public String getAuthor() { return author; }
53
54         /**
55          * Restituisce il genere del libro.
56         */
57         > public Genre getGenre() { return genre; }
58
59         /**
60          * Restituisce l'anno di pubblicazione del libro.
61         */
62         > public int getYear() { return year; }
63
64
65         /**
66          * Ritorna il titolo fra doppi apici seguito dall'autore e poi (fra parentesi) anno di pubblicazione e
67          * genere.
68
69         */
70
71
72
73
74
75
76         @Override 2 overrides ▾ GiorgiaZanini
77         > public String toString() { return "\"" + title + "\" by " + author + " (" + year + ", " + genre + ")"; }
78
79
80
81         /**
82          * Due libri sono equals se hanno stesso titolo, autore e anno di pubblicazione. Il genere non viene
83          * preso in considerazione.
84
85         */
86         @Override 2 overrides ▾ GiorgiaZanini
87         > public boolean equals(Object other) {
88             if (!(other instanceof Book))
89                 return false;
90
91             Book otherBook = (Book) other;
92             if (title.equals(otherBook.title) && author.equals(otherBook.author) && (year == otherBook.year))
93                 return true;
94
95             return false;
96         }
97
98         @Override ▾ GiorgiaZanini
99         > public int hashCode() {
100             // return title.length() + author.length() + year + genre.ordinal();
101             return title.hashCode() + author.hashCode() + year + genre.hashCode();
102         }
103
104
105         /**
106          * Due libri sono ordinati per titolo; a parità di titolo, sono ordinati per autore; a parità anche di
107          * autore, sono ordinati per anno di pubblicazione.
108
109         */
110         @Override 2 overrides ▾ GiorgiaZanini
111         > public int compareTo(Book other) {
112             if (this.equals(other))
113                 return 0;
114
115             if (!(title.equals(other.title)))
116                 return title.compareTo(other.title);
117
118             if (!(author.equals(other.author)))
119                 return author.compareTo(other.author);
120
121             if (year < other.year)
122                 return -1;
123             return 1;
124         }
125     }
```

```
© AudioBook.java × © Catalog.java © CatalogWithStatistics.java © PaperBook.java © Main.java
```

1 package it.univr.library;

2

3 | Un audio-libro.

6 public class AudioBook extends Book { 8 usages ▾ GiorgiaZanini

7

8 private int minutes = -1; 7 usages

9

10 |
11 | Crea un audio-libro.

12 | Params: title – il titolo del libro

13 | author – l'autore del libro

14 | year – l'anno di pubblicazione del libro

15 | genre – il genere del libro

16 | minutes – la durata del libro in minuti

17 |
18 | Throws: [IllegalArgumentException](#) – se qualche parametro e' null o se minutes e' negativo

20 public AudioBook(String title, String author, int year, Genre genre, int minutes) { 4 usages ▾ GiorgiaZanini

21 super(title, author, year, genre);

22 if (minutes < 0)

23 throw new [IllegalArgumentException\(\)](#);

24 this.minutes = minutes;

25 }

26

27 | Ritorna il numero di minuti dell'audio-libro.

30 > public int getMinutes() { return minutes; }

33

34 | Si comporta come il [toString](#) di Book ma aggiunge in fondo la durata in minuti fra parentesi quadre.

38 @Override ▾ GiorgiaZanini

39 public String [toString](#)() { return super.[toString](#)() + " [" + minutes + " min]"; }

42

43 | Si comporta come l'[equals](#) di Book ma in piu' richiede che other sia un AudioBook.

46 @Override ▾ GiorgiaZanini

47 public boolean [equals](#)(Object other) {

48 if (!super.[equals](#)(other))

49 return false;

50

51 return other instanceof [AudioBook](#);

52 }

53

54 | Si comporta come il [compareTo](#) di Book ma, a parità di titolo, autore e anno, mette prima i PaperBook e poi gli AudioBook.

58 @Override ▾ GiorgiaZanini

59 public int [compareTo](#)(it.univr.library.Book other) {

60 if (super.[compareTo](#)(other) != 0)

61 return super.[compareTo](#)(other);

62

63 if (!(other instanceof [AudioBook](#)))

64 return -1;

65

66 if (minutes < (([AudioBook](#)) other).minutes)

67 return -1;

68 else if (minutes > (([AudioBook](#)) other).minutes)

69 return 1;

70 return 0;

71 }

72 }

```
© Catalog.java © CatalogWithStatistics.java © PaperBook.java × © Main.java
1 package it.univr.library;
2
3     Un libro cartaceo.
4
5     public class PaperBook extends Book { 13 usages ▲ GiorgiaZanini
6
7         private int pages = -1; 7 usages
8
9
10        Crea un libro cartaceo.
11
12        Params: title – il titolo del libro
13            author – l'autore del libro
14            year – l'anno di pubblicazione del libro
15            genre – il genere del libro
16            pages – il numero di pagine del libro
17
18        Throws: IllegalArgumentException – se qualche parametro e' null o se pages e' negativo
19
20        public PaperBook(String title, String author, int year, Genre genre, int pages) { 9 usages ▲ GiorgiaZanini
21            super(title, author, year, genre);
22            if (pages < 0)
23                throw new IllegalArgumentException();
24            this.pages = pages;
25        }
26
27        ⓘ
28
29        Ritorna il numero di pagine del libro.
30        public int getPages() { return pages; }
31
32
33        Si comporta come il toString di Book ma aggiunge in fondo il numero di pagine fra parentesi quadre.
34
35        @Override ▲ GiorgiaZanini
36        public String toString() { return super.toString() + " [" + pages + " pages]"; }
37
38
39        ⓘ
40        Si comporta come l'equals di Book ma in piu' richiede che other sia un PaperBook.
41
42        @Override ▲ GiorgiaZanini
43        public boolean equals(Object other) {
44            if (!super.equals(other))
45                return false;
46
47            return other instanceof PaperBook;
48        }
49
50
51        /**
52         * Si comporta come il compareTo di Book ma, a parità di titolo, autore e anno,
53         * mette prima i PaperBook e poi gli AudioBook.
54         */
55
56        @Override ▲ GiorgiaZanini
57        public int compareTo(Book other) {
58            if (super.compareTo(other) != 0)
59                return super.compareTo(other);
60
61            if (!(other instanceof PaperBook)) {
62                return 1;
63            }
64
65            if (pages < ((PaperBook) other).pages)
66                return -1;
67            else if (pages > ((PaperBook) other).pages)
68                return 1;
69            return 0;
70        }
71    }
72
73 }
```

Catalog.java × CatalogWithStatistics.java

Main.java

```
1 package it.univr.library;
2
3 import java.util.*;
4
5
6     Un catalogo contiene dei libri (senza ripetizioni).
7
8     public class Catalog implements Iterable<Book> { 5 usages 1 inheritor ▲ GiorgiaZanini
9
10        private final SortedSet<Book> catalog; 6 usages
11
12
13        Costruisce un catalogo. L'ordinamento sara' quello del compareTo definito in Book.
14
15        Params: books – i libri contenuti nel catalogo. Anche se ci fossero ripetizioni, il catalogo dovrà
16            contenere una sola istanza del libro ripetuto
17
18        public Catalog(Book... books) { 3 usages ▲ GiorgiaZanini
19            catalog = new TreeSet<>();
20            Collections.addAll(catalog, books);
21        }
22
23
24        Costruisce un catalogo. L'ordinamento sara' quello del comparatore fornito.
25
26        Params: comparator – l'oggetto che determina l'ordine fra due libri
27            books – i libri contenuti nel catalogo. Anche se ci fossero ripetizioni, il catalogo dovrà
28            contenere una sola istanza del libro ripetuto
29
30        public Catalog(Comparator<Book> comparator, Book... books) { 5 usages ▲ GiorgiaZanini
31            catalog = new TreeSet<>(comparator);
32            Collections.addAll(catalog, books);
33        }
34
35
36        @Override ▲ GiorgiaZanini
37        public Iterator<Book> iterator() { return catalog.iterator(); }
38
39
40        /**
41         * Restituisce il toString dei libri contenuti in questo catalogo, nel loro ordine,
42         * andando a capo dopo ciascuno di essi.
43         */
44
45        @Override ▲ GiorgiaZanini
46        public String toString() {
47            String string = "";
48
49            for (Book book : catalog) {
50                string += book + "\n";
51                // catalog.last()
52            }
53
54            return string.trim();    // trim -> toglie " ", "\n" (davanti e in fondo alla stringa)
55        }
56
57    }
```

```
© CatalogWithStatistics.java x © Main.java
1 package it.univr.library;
2
3 import java.util.Iterator;
4
5
6 Un catalogo che stampa anche statistiche sui libri contenuti.
7
8 public class CatalogWithStatistics extends Catalog { 1 usage ± GiorgiaZanini
9
10
11 Costruisce un catalogo con statistiche. L'ordinamento sara' quello del compareTo definito in Book.
12
13 Params: books - i libri contenuti nel catalogo. Anche se ci fossero ripetizioni, il catalogo dovrà
14 contenere una sola istanza del libro ripetuto
15
16 public CatalogWithStatistics(Book... books) { super(books); }
17
18
19 Si comporta come il toString() di Catalog, ma alla fine aggiunge una riga del tipo: "This catalog
20 contains paper books for a total of XXX pages and audiobooks for a total of YYY minutes"
21
22 @Override ± GiorgiaZanini
23 public String toString() {
24     int allPages = 0;
25     int allMinutes = 0;
26
27     Iterator<Book> iterator = super.iterator(); // ottieni un iteratore dai libri del catalogo
28     Book book;
29
30     while (iterator.hasNext()) {
31         book = iterator.next();
32
33         if (book instanceof PaperBook)
34             allPages += ((PaperBook) book).getPages();
35
36         if (book instanceof Audiobook)
37             allMinutes += ((Audiobook) book).getMinutes();
38
39     }
40
41     return super.toString() + "\nThis catalog contains paper books for a total of " + allPages + " pages and audiobooks for a total of " + allMinutes + " minutes";
42
43 }
44
45 }
```

```

1 import it.univr.library.*;
2
3 import java.util.Comparator;
4 import java.util.function.Function;
5
6 public class Main { // GiorgiaZanini
7
8     public static void main(String[] args) { // GiorgiaZanini
9         Book jj = new PaperBook( title: "The joy of Java", author: "John Stack", year: 2018, Genre.COMPUTING, pages: 557);
10        Book cr = new AudioBook( title: "Cappuccetto rosso", author: "Charles Perrault", year: 1697, Genre.FICTION, minutes: 13);
11        Book ps = new AudioBook( title: "I promessi sposi", author: "Alessandro Manzoni", year: 1840, Genre.FICTION, minutes: 1223);
12        Book ps2 = new PaperBook( title: "I promessi sposi", author: "Alessandro Manzoni", year: 1840, Genre.FICTION, pages: 622);
13        Book gl = new PaperBook( title: "Sentieri in Lessinia", author: "Giovanni Gamba", year: 2015, Genre.GUIDE, pages: 233);
14        Book sv = new PaperBook( title: "Gli Scala di Verona", author: "Roberta Guidi", year: 2012, Genre.HISTORY, pages: 380);
15        Book sv2 = new PaperBook( title: "Gli Scala di Verona", author: "Roberta Guidi", year: 2012, Genre.HISTORY, pages: 380);
16
17        // crea e poi stampa un catalogo con statistiche che contiene jj, cr, ps, ps2, gl, sv, sv2, ordinato secondo il compareTo fra i libri
18        System.out.println("Ordinamento naturale:\n" + new CatalogWithStatistics(jj, cr, ps, ps2, gl, sv, sv2) + "\n");
19
20        // crea e poi stampa un catalogo che contiene jj, cr, ps, ps2, gl, sv, sv2, ordinato per autore e, a parita' di autore, secondo il compareTo fra i libri
21        Comparator<Book> comparatorAuthor = new Comparator<Book>() { // GiorgiaZanini
22            @Override // GiorgiaZanini
23            public int compare(Book thisBook, Book otherBook) {
24                if (!thisBook.getAuthor().equals(otherBook.getAuthor()))
25                    return thisBook.getAuthor().compareTo(otherBook.getAuthor());
26                return thisBook.compareTo(otherBook);
27            }
28        };
29        System.out.println("Ordinamento per autore:\n" + new Catalog(comparatorAuthor, jj, cr, ps, ps2, gl, sv, sv2) + "\n");
30
31        // crea e poi stampa un catalogo che contiene jj, cr, ps, ps2, gl, sv, sv2, ordinato per anno e, a parita' di anno, secondo il compareTo fra i libri
32        Comparator<Book> comparatorYear = new Comparator<Book>() { // GiorgiaZanini
33            @Override // GiorgiaZanini
34            public int compare(Book thisBook, Book otherBook) {
35                if (thisBook.getYear() != otherBook.getYear()) {
36                    if (thisBook.getYear() < otherBook.getYear())
37                        return -1;
38                    return 1;
39                }
40                return thisBook.compareTo(otherBook);
41            }
42        };
43        System.out.println("Ordinamento per anno di pubblicazione:\n" + new Catalog(comparatorYear, jj, cr, ps, ps2, gl, sv, sv2) + "\n");
44
45        try {
46            new PaperBook( title: "Gli Scala di Verona", author: "Roberta Guidi", year: 2012, Genre.HISTORY, pages: -380);
47            System.out.println("legale");
48        }
49        catch (IllegalArgumentException e) {
50            System.out.println("illeagle");
51        }
52
53        try {
54            new PaperBook( title: "Gli Scala di Verona", author: null, year: 2012, Genre.HISTORY, pages: 380);
55            System.out.println("legale");
56        }
57        catch (IllegalArgumentException e) {
58            System.out.println("illeagle");
59        }
60
61        Catalog oggettoCatalogo = new Catalog(comparatorAuthor, jj, cr, ps, ps2, gl, sv, sv2);
62        for (Book book : oggettoCatalogo) {
63            //....
64            System.out.println(book);
65        }
66    }
67 }
68

```

Lab 8

```
© Calendar.java × © MainCalendar.java
1 import java.util.Iterator;
2 import java.util.Objects;
3
4 public class Calendar implements Iterable<Calendar.Date> { 13 usages ± GiorgiaZanini *
5     private final int year; // l'anno del calendario 3 usages
6     //private static final int[] daysForMonth = {30, 27, 30, 29, 30, 29, 30, 30, 29, 30, 29, 30};
7     // ogni cella contiene il numero massimo di giorni per ogni mese, ridotto di 1 (anno non bisestile)
8     private static final int[] daysForMonth = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}; 3 usages
9
10    // costruisce il calendario per l'anno indicato
11    > public Calendar(int year) { this.year = year; }
12
13    // determina se il calendario è per un anno bisestile
14    > public boolean isLeapYear() { return year % 4 == 0; }
15
16    // restituisce la prima data del calendario (primo gennaio)
17    > public Date getStart() { return new Date( daysFromStart: 0); }
18
19    // restituisce l'ultima data del calendario (31 dicembre)
20    public Date getEnd() { 2 usages ± GiorgiaZanini
21        if (isLeapYear())
22            return new Date( daysFromStart: 365);
23        return new Date( daysFromStart: 364);
24    }
25
26    @Override ± GiorgiaZanini
27    public Iterator<Calendar.Date> iterator() {
28        return new Iterator<Calendar.Date>() { ± GiorgiaZanini
29            private Date date = getStart(); 3 usages
30            int day = 0; 2 usages
31
32            public boolean hasNext() { return !date.equals(getEnd()); }
33
34            public Date next() { ± GiorgiaZanini
35                date = new Date(day);
36                day++;
37                return date;
38            }
39        };
40    }
41
42    };
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
311
312
313
313
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396

```

```
4  public class Calendar implements Iterable<Calendar.Date> { 13 usages ± GiorgiaZanini *
50
51      // classe interna (inner class)
52      public class Date { 16 usages ± GiorgiaZanini
53          // 0 = primo gennaio, 364 = 31 dicembre (per i bisestili: 365 = 31 dicembre)
54          private final int daysFromStart;    // giorno nel range da trasformare in data 4 usages
55          private int iMonth; // indice dell'array, corrisponde al mese ridotto di 1 8 usages
56          private int currentDay; 3 usages
57
58          private Date(int daysFromStart) { 4 usages ± GiorgiaZanini
59              this.daysFromStart = daysFromStart;
60              rangeDaysForMonth();
61          }
62
63          /*
64             gennaio --> 1-31 -----> 0-30
65             febbraio --> 1-28 | 1-29 --> 31-58 | 31-59
66             marzo -----> 1-31 -----> 59-89 | 60-90
67             aprile ----> 1-30 -----> 90-119 | 91-120
68             maggio ----> 1-31 -----> 120-150 | 121-150
69             giugno ----> 1-30 -----> 150-180 | 151-181
70             luglio ----> 1-31 -----> 181-211 | 182-212
71             agosto ----> 1-31 -----> 212-242 | 213-243
72             settembre -> 1-30 -----> 242-272 | 244-273
73             ottobre --> 1-31 -----> 273-303 | 274-304
74             novembre --> 1-30 -----> 304-333 | 305-334
75             dicembre --> 1-31 -----> 334-364 | 335-365
76             */ // tabella mesi-giorni
77         private void rangeDaysForMonth() { 1 usage ± GiorgiaZanini
78             int daysFromStart = this.daysFromStart + 1;
79             for (iMonth = 0; iMonth < 12; iMonth++) {
80                 if (isLeapYear() && iMonth == 1) {
81                     if (daysFromStart <= daysForMonth[iMonth] + 1) {
82                         currentDay = daysFromStart;
83                         break;
84                     }
85                     daysFromStart--;
86                 } else {
87                     if (daysFromStart <= daysForMonth[iMonth]) {
88                         currentDay = daysFromStart;
89                         break;
90                     }
91                 }
92                 daysFromStart -= daysForMonth[iMonth];
93             }
94         }
95
96         // ritorna il giorno di questa data, tra 1 e 31
97     >     public int getDay() { return currentDay; }
98
99
100        // ritorna il mese di questa data tra 1 e 12
101    >     public int getMonth() { return Math.min(iMonth+1, 12); }
102
103
104        // ritorna l'anno di questa data
105    >     public int getYear() { return year; }
106
107
108        public String toString() { return String.format("%d/%d/%d", getDay(), getMonth(), getYear()); }
109
110
111        @Override ± GiorgiaZanini
112        public boolean equals(Object other) {
113            if (!(other instanceof Date))
114                return false;
115            return ((Date) other).daysFromStart == this.daysFromStart;
116        }
117    }
118
119
120
121
122
123 }
```

© Calendar.java

© MainCalendar.java ×

```
1 ⌂ public class MainCalendar { ↗ GiorgiaZanini
2 ⌂     public static void main(String[] args) { ↗ GiorgiaZanini
3         Calendar cal = new Calendar( year: 2024);
4         System.out.println("start: " + cal.getStart());
5         System.out.println("end: " + cal.getEnd());
6         for (Calendar.Date date: cal)
7             System.out.println(date);
8     }
9
10
11
```

Lab 9

```
① NextAliveProcessor.java ×
1  public interface NextAliveProcessor { 9 usages 3 implementations ✎ GiorgiaZanini
2  boolean isAliveNextAt(int x, int y); 1 usage 3 implementations ✎ GiorgiaZanini
3  }
4

② NextAliveProcessor.java      © Board.java ×
1  import java.util.Random;
2
3  public class Board { 19 usages ✎ GiorgiaZanini
4      private final int width; 6 usages
5      private final int height; 6 usages
6      private boolean[][] current; 6 usages
7      // ...
8
9
10     Inizializza la tavola alle dimensioni indicate, in modo che contenga esattamente howManyAlive
11     cellule vive, posizionate casualmente nella tavola.
12
13     Params: width - la larghezza della tavola
14         height - l'altezza della tavola
15         howManyAlive - il numero di cellule vive della tavola
16
17     Throws: IllegalArgumentException - se width o height sono negative, oppure se
18         howManyAlive è negativo o maggiore del numero di cellule della tavola
19
20     public Board(int width, int height, int howManyAlive) { 13 usages ✎ GiorgiaZanini
21         if (howManyAlive < 0 || howManyAlive > width*height)
22             throw new IllegalArgumentException("il numero di cellule vive non è nel range previsto");
23
24         if (width < 0)
25             throw new IllegalArgumentException("width è negativo");
26         this.width = width;
27
28         if (height < 0)
29             throw new IllegalArgumentException("height è negativo");
30         this.height = height;
31
32
33         this.current = new boolean[width][height];
34
35         for (int i = 0; i < width; i++)
36             for (int j = 0; j < height; j++)
37                 current[i][j] = false;
38
39         Random random = new Random();
40         int randomWidth;
41         int randomHeight;
42         for (int i = 0; i < howManyAlive; i++) {
43             do {
44                 randomWidth = random.nextInt(width);
45                 randomHeight = random.nextInt(height);
46             } while (current[randomWidth][randomHeight]);
47             current[randomWidth][randomHeight] = true;
48         }
49     }
50     /*...*/
51 }
```

NextAliveProcessor.java Board.java

```
3  public class Board { 19 usages ± GiorgiaZanini
62
63 >     public int getWidth() { return width; }
64
65 >     public int getHeight() { return height; }
66
67
68 |     Determina se la cellula (x,y) è viva o morta.
69
70
71 >     public boolean isAliveAt(int x, int y) { return current[x][y]; }
72
73
74 >     public String toString() { ± GiorgiaZanini
75         String result = "";
76         for (int y = 0; y < height; y++) {
77             for (int x = 0; x < width; x++) {
78                 result += isAliveAt(x, y) ? '*' : ' ';
79                 //result += isAliveAt(x, y) ? "|*" : "| ";
80                 result += "\n";
81                 //result += "\\\n";
82             }
83             return result;
84         }
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
```

Stampa ripetutamente questa tavola, chiamando ogni volta next() per farla passare alla generazione successiva. L'unico modo per terminare questo metodo è con CTRL+C. In Eclipse, tale segnale viene inviato cliccando sul pulsante quadrato rosso della console.

Params: `processor` – la specifica di come si passa da una generazione alla successiva

```
public void play(NextAliveProcessor processor) throws InterruptedException { 6 usages ± GiorgiaZanini
    while (true) {
        System.out.println(this);
        System.out.println("-----");
        //System.out.println("-----");
        next(processor);
        Thread.sleep( millis: 500); // aspetta mezzo secondo
    }
}

Modifica la tavola in modo che la cellula (x,y) sarà viva se e solo se processor.isAliveNextAt(x,y) è vero.

private void next(NextAliveProcessor processor) { 1 usage ± GiorgiaZanini
    //Board b = new Board(width, height, 0);
    //b.current[0][0] = true;
    boolean[][] tmpBoard = new boolean [width][height];
    for (int i = 0; i < width; i++)
        for (int j = 0; j < height; j++)
            tmpBoard[i][j] = false;

    for (int i = 0; i < width; i++)
        for (int j = 0; j < height; j++)
            if (processor.isAliveNextAt(i,j))
                tmpBoard[i][j] = true;

    current = tmpBoard;
}
```

① NextAliveProcessor.java ② Board.java ③ MainAtLeast3.java ×

```
1 ▶ public class MainAtLeast3 { ± GiorgiaZanini +1 *
2     private static Board board = new Board( width: 40, height: 20, howManyAlive: 100);  4 usages
3
4 ▶     public static void main(String[] args) throws InterruptedException { ± GiorgiaZanini +1
5         board.play(new Processor3());
6     }
7
8     private static class Processor3 implements NextAliveProcessor { 1 usage ± GiorgiaZanini +1 *
9         @Override 1 usage ± GiorgiaZanini +1 *
10        public boolean isAliveNextAt(int x, int y) {
11            int counterAlive = 0;
12
13            for (int i = (x-1); i <= (x+1); i++) {
14                for (int j = (y-1); j <= (y+1); j++) {
15                    if (i >= 0 && j >= 0 && i < board.getWidth() && j < board.getHeight() && (i!=x || j!=y) && board.isAliveAt(i,j))
16                        // /or -> per entrare devono essere entrambe "false"
17                        counterAlive++;
18                    if (counterAlive >= 3)
19                        return true;
20                }
21            }
22
23            return false;
24        }
25    }
26 }
27 }
```

① NextAliveProcessor.java ② Board.java ③ MainAtLeast3.java ④ MainAtLeast3_lambda.java ×

```
1 ▶ public class MainAtLeast3_lambda { ± GiorgiaZanini
2     private static Board board = new Board( width: 40, height: 20, howManyAlive: 100);  4 usages
3
4 ▶     public static void main(String[] args) throws InterruptedException { ± GiorgiaZanini
5         board.play(( int x, int y) -> {
6             int counterAlive = 0;
7
8             for (int i = (x-1); i <= (x+1); i++) {
9                 for (int j = (y-1); j <= (y+1); j++) {
10                     if (i >= 0 && j >= 0 && i < board.getWidth() && j < board.getHeight() && (i!=x || j!=y) && board.isAliveAt(i,j))
11                         counterAlive++;
12                     if (counterAlive >= 3)
13                         return true;
14                 }
15             }
16
17             return false;
18         });
19     }
20 }
21 }
```

① NextAliveProcessor.java ② Board.java ③ MainConstant.java ④ MainConstant_lambda.java

```
1 ▶ public class MainConstant { ± GiorgiaZanini
2   ● private static Board board = new Board( width: 40, height: 20, howManyAlive: 100); 2 usages
3
4 ▶   public static void main(String[] args) throws InterruptedException { ± GiorgiaZanini
5     |   board.play(new ProcessorEquals());
6   }
7
8   private static class ProcessorEquals implements NextAliveProcessor { 1 usage ± GiorgiaZanini
9     | @Override 1 usage ± GiorgiaZanini
10    |   public boolean isAliveNextAt(int x, int y) { return board.isAliveAt(x,y); }
11  }
12
13 }
14
15 }
```

① NextAliveProcessor.java ② Board.java ③ MainConstant.java ④ MainConstant_lambda.java

```
1 ▶ public class MainConstant_lambda { ± GiorgiaZanini
2 ▶   ● public static void main(String[] args) throws InterruptedException { ± GiorgiaZanini
3     |   Board board = new Board( width: 40, height: 20, howManyAlive: 100);
4
5     /**
6      |   board.play(new NextAliveProcessor() { // classe anonima
7        |     @Override
8        |     public boolean isAliveNextAt(int x, int y) {
9          |       return board.isAliveAt(x,y);
10         |     }
11       });
12     */
13
14     |   board.play(( int x, int y) -> board.isAliveAt(x,y));
15     |   //board.play(board::isAliveAt);
16   }
17
18 }
```

① NextAliveProcessor.java ② Board.java ③ MainGameOfLife.java

```
1 public class MainGameOfLife {   ▲ GiorgiaZanini
2     private static Board board = new Board( width: 40, height: 20, howManyAlive: 100);  5 usages
3     //private static Board board = new Board(5, 5, 10);
4
5     public static void main(String[] args) throws InterruptedException {   ▲ GiorgiaZanini
6         board.play(new ProcessorRules());
7     }
8
9     /*
10      Rules
11      - Qualsiasi cella viva con meno di due celle vive adiacenti muore, come per effetto d'isolamento;
12      - Qualsiasi cella viva con due o tre celle vive adiacenti sopravvive alla generazione successiva;
13      - Qualsiasi cella viva con più di tre celle vive adiacenti muore, come per effetto di sovrappopolazione;
14      - Qualsiasi cella morta con esattamente tre celle vive adiacenti diventa una cella viva, come per effetto di riproduzione.
15  */
16     private static class ProcessorRules implements NextAliveProcessor { 1 usage  ▲ GiorgiaZanini
17         @Override 1 usage  ▲ GiorgiaZanini
18         public boolean isAliveNextAt(int x, int y) {
19             int counterAlive = 0;
20
21             for (int i = (x-1); i <= (x+1); i++) {
22                 for (int j = (y-1); j <= (y+1); j++) {
23                     if (i >= 0 && j >= 0 && i < board.getWidth() && j < board.getHeight() && (i!=x || j!=y) && board.isAliveAt(i,j))
24                         counterAlive++;
25                 }
26             }
27
28             if (board.isAliveAt(x,y))
29                 return (counterAlive >= 2 && counterAlive <=3); // rimane viva se intorno ha da 2 a 3 celle vive
30             else
31                 return counterAlive == 3; // diventa viva se intorno ha esattamente 3 celle vive
32         }
33     }
34 }
```

① NextAliveProcessor.java ② Board.java ③ MainGameOfLife.java ④ MainGameOfLife_lambda.java

```
1 public class MainGameOfLife_lambda {  ▲ GiorgiaZanini
2     private static Board board = new Board( width: 40, height: 20, howManyAlive: 100);  5 usages
3
4     public static void main(String[] args) throws InterruptedException {  ▲ GiorgiaZanini
5         board.play(( int x, int y) -> {
6             int counterAlive = 0;
7
8             for (int i = (x-1); i <= (x+1); i++) {
9                 for (int j = (y-1); j <= (y+1); j++) {
10                     if (i >= 0 && j >= 0 && i < board.getWidth() && j < board.getHeight() && (i!=x || j!=y) && board.isAliveAt(i,j))
11                         counterAlive++;
12                 }
13             }
14
15             if (board.isAliveAt(x,y))
16                 return (counterAlive >= 2 && counterAlive <=3); // rimane viva se intorno ha da 2 a 3 celle vive
17             else
18                 return counterAlive == 3; // diventa viva se intorno ha esattamente 3 celle vive
19         });
20     }
21 }
```

Esame 1

```
⌚ StudenteIllegaleException.java × ⚖ Corso.java ⚖ Studente.java
1 package it.univr.corso;
2
3 public class StudenteIllegaleException extends IllegalArgumentException { 12 usages ± GiorgiaZanini
4 >     public StudenteIllegaleException() { super("studente incompatibile"); }
7
8 >     public StudenteIllegaleException(String string) { super("studente incompatibile: " + string); }
11 }
12
```

```
⌚ StudenteIllegaleException.java ⚖ Corso.java × ⚖ Studente.java
1 package it.univr.corso;
2
3 ⚡ In corso di laurea (per esempio, informatica), con nome e durata in anni.
4
5 public class Corso { 6 usages ± GiorgiaZanini
6     // aggiungete campi se servissero
7     private final String nome_corso; 3 usages
8     private final int durata; 2 usages
9
10
11     public Corso(String nome, int durata) { 3 usages ± GiorgiaZanini
12         this.nome_corso = nome;
13         this.durata = durata;
14     }
15
16     @Override ± GiorgiaZanini
17     public String toString() { return nome_corso; }
18
19     // aggiungete i metodi accessori pubblici getName() e getDurata()
20
21     public String getName() { return nome_corso; }
22
23     public int getDurata() { return durata; }
24 }
```

```
1 package it.univr.corso;
2
3     import java.util.Date; // rappresentare una data e un'ora (da Java 8)
4     import java.util.Calendar; // ottenere la data e l'ora attuali (da Java 8)
5     import java.time.LocalDateTime; // ottenere la data e l'ora attuali (da Java 8)
6     import java.util.Scanner;
7
```

Uno studente, che si puo' iscrivere agli esami di un corso di laurea.

```
11 @public class Studente implements Comparable<Studente> { 28 usages 1 inheritor ✎ GiorgiaZanini
12     // aggiungete campi se servissero
13     private final String nome; 2 usages
14     private final String cognome; 2 usages
15     private final int matricola; 8 usages
16     private final int annoDiImmatricolazione; 4 usages
17
```

Crea uno studente con il nome, cognome, matricola e anno di immatricolazione indicati.

Throws: StudenteIllegalException – se l'anno di immatricolazione è nel futuro o se la matricola è negativa

```
24     public Studente(String nome, String cognome, int matricola, int annoDiImmatricolazione) throws StudenteIllegalException { 9 usages ✎ GiorgiaZanini
25         this.nome = nome;
26         this.cognome = cognome;
27
28         if (matricola < 0)
29             throw new StudenteIllegalException("numero di matricola negativo");
30         this.matricola = matricola;
31
32         Calendar calendar = Calendar.getInstance();
33         if (annoDiImmatricolazione > calendar.get(Calendar.YEAR)) // calendar.get(Calendar.YEAR) -> ritorna anno corrente
34             // Calendar.YEAR è un intero che serve per riferirsi al campo dell'anno in un oggetto Calendar
35             throw new StudenteIllegalException("anno futuro rispetto a quello corrente");
36         this.annoDiImmatricolazione = annoDiImmatricolazione;
37     }
38
```

```
⌚ StudenteIllegalException.java     ⌚ Corso.java     ⌚ Studente.java ×
11  public class Studente implements Comparable<Studente> {  28 usages  1 inheritor  ▲ GiorgiaZanini *
38
        Legge da tastiera i dati di uno studente, lo crea e lo restituisce. Già fatto, non modificate.
        Throws: StudenteIllegalException – se i dati letti sono illegali
45  @
46      public static Studente leggiDaTastiera(Scanner keyboard) throws StudenteIllegalException {  1 usage  ▲ GiorgiaZanini *
47          System.out.print("Nome: ");
48          String nome = keyboard.nextLine();
49          System.out.print("Cognome: ");
50          String cognome = keyboard.nextLine();
51          System.out.print("Matricola: ");
52          int matricola = keyboard.nextInt(); keyboard.nextLine();
53          System.out.print("Anno di immatricolazione: ");
54          int annoDiImmatricolazione = keyboard.nextInt(); keyboard.nextLine();
55
56          return new Studente(nome, cognome, matricola, annoDiImmatricolazione);
57      }
58
59
60      Due studenti sono uguali se e solo se hanno la stessa matricola.
61  @Override  ▲ GiorgiaZanini
62  ⚡
63      public boolean equals(Object other) {
64          if (!(other instanceof Studente))
65              return false;
66
67          return this.matricola == ((Studente) other).matricola;
68      }
69
70
71      Deve essere non banale e compatibile con equals.
72  @Override  ▲ GiorgiaZanini *
73  ⚡
74      public int hashCode() { return Integer.hashCode(matricola); }
75
76
77
78      Determina chi viene prima (per matricola) tra this e other.
79  @Override  ▲ GiorgiaZanini
80  ⚡
81      public int compareTo(Studente other) {
82          /*
83          if (this.matricola > other.matricola)
84              return 1;
85          if (this.matricola < other.matricola)
86              return -1;
87          return 0;
88          */
89          return Integer.compare(this.matricola, other.matricola);
90      }
91
92
93      // restituisce una stringa del tipo "34555 Giulio Andreotti immatricolato nel 2017"
94  @Override  ▲ GiorgiaZanini
95  ⚡
96      public String toString() {
97          return matricola + " " + nome + " " + cognome + " immatricolato nel " + annoDiImmatricolazione;
98      }
99
100
101     Determina se questo studente e' fuori corso rispetto al corso di laurea indicato: ovvero se si e'
102     immatricolato prima della durata del corso di laurea.
103  @
104      //return true -> fuori corso
105      public boolean fuoriCorso(Corso corso) {  1 usage  1 override  ▲ GiorgiaZanini
106          Calendar calendar = Calendar.getInstance();
107          return (calendar.get(Calendar.YEAR) - annoDiImmatricolazione) > corso.getDurata();
108      }
109
110
111     // aggiungete i metodi accessori pubblici getAnnoDiImmatricolazione() e getMatricola()
112
113     >     public int getAnnoDiImmatricolazione() { return annoDiImmatricolazione; }
114
115     >     public int getMatricola() { return matricola; }
116
117
118 }
```

```
④ StudenteIllegalException.java   ⑤ Corso.java      ⑥ Studente.java      ⑦ StudenteLavoratore.java x
  1 package it.univr.corso;
  2
  3 > import ...;
  5
  6
  7 Uno studente lavoratore è identico a uno studente ma finisce fuori corso nel doppio di anni rispetto a
  8 uno studente non lavoratore.
  9
 10 public class StudenteLavoratore extends Studente { 3 usages ± GiorgiaZanini
 11   // aggiungete campi se servissero
 12
 13   public StudenteLavoratore(String nome, String cognome, int matricola, int annoDiImmatricolazione) throws StudenteIllegalException {
 14     super(nome, cognome, matricola, annoDiImmatricolazione);
 15   }
 16
 17   @Override 1 usage ± GiorgiaZanini
 18   @⑧ public boolean fuoriCorso(Corso corso) {
 19     Calendar calendar = Calendar.getInstance();
 20     return (calendar.get(Calendar.YEAR) - getAnnoDiImmatricolazione()) > (corso.getDurata() * 2);
 21   }
 22
 23
```

```
④ StudenteIllegaleException.java      © Corso.java      © Studente.java      © StudenteLavoratore.java      © Esame.java  ×
1 package it.univr.corso;
2
3 > import ...
4
5
6     Un esame di un corso di laurea, con il nome dell'esame e il corso di laurea a cui appartiene.
7
8
9
10 public class Esame { 2 usages  ▲ GiorgiaZanini*
11     // aggiungete campi se servissero
12     private final String nome_esame; 2 usages
13     private final Corso corso; 2 usages
14     private SortedSet<Studente> iscritti = new TreeSet<>(); 5 usages
15
16
17
18     Crea un esame con il nome indicato, per il corso indicato, inizialmente senza iscritti.
19
20     public Esame(String nome, Corso corso) { 5 usages  ▲ GiorgiaZanini
21         nome_esame = nome;
22         this.corso = corso;
23     }
24
25
26     Iscrive lo studente indicato a questo esame.
27
28     Throws: StudenteIllegaleException - se ci fosse già uno studente iscritto a questo esame
29             con la stessa matricola
30
31
32     public void iscrivi(Studente studente) throws StudenteIllegaleException { 6 usages  ▲ GiorgiaZanini
33         if (!iscritti.isEmpty()) {
34             for (Studente s : iscritti) {
35                 if (s.getMatricola() == studente.getMatricola())
36                     throw new StudenteIllegaleException("numero matricola già registrata");
37             }
38         }
39         iscritti.add(studente);
40     }
41
42     // restituisce la stringa ottenuta concatenando tutti gli iscritti all'esame
43     // in ordine crescente per matricola; all'inizio riporta nome dell'esame e corso;
44     // si veda l'esempio nel testo del compito
45     /*...*/
46
47     @Override  ▲ GiorgiaZanini
48     public String toString() {
49         String allStudents = "Esame di " + nome_esame + " del corso di " + corso.getNome() + ":\n";
50
51
52         for (Studente s : iscritti) {
53             allStudents += s.toString() + "\n";
54         }
55
56
57         return allStudents.trim();
58     }
59
60
61
62     /**
63      * Esegue l'azione indicata per ogni studente iscritto che soddisfa
64      * la condizione indicata.
65      */
66
67     public void perOgniIscritto(Predicate<Studente> condizione, Consumer<Studente> azione) { 2 usages  ▲ Giorgi
68         for (Studente s : iscritti) {
69             if (condizione.test(s))
70                 azione.accept(s);
71         }
72     }
73
74 }
```

```
© Studente.java © StudenteLavoratore.java © Esame.java © MainEsame.java x
1 package it.univr.corso;
2
3 import java.util.InputMismatchException;
4 import java.util.Scanner;
5 import java.util.function.Consumer;
6 import java.util.function.Predicate;
7
8 public class MainEsame { ± GiorgiaZanini *
9
10    public static void main(String[] args) throws StudenteIllegaleException { ± GiorgiaZanini *
11        Corso informatica = new Corso( nome: "Informatica", durata: 3);
12        Esame esame = new Esame( nome: "Programmazione Quantistica", informatica);
13
14        // iscrive cinque studenti all'esame
15        esame.iscrivi(new Studente( nome: "Giulio", cognome: "Rossi", matricola: 111564, annoDilmmatricolazione: 2021));
16        esame.iscrivi(new Studente( nome: "Alessandra", cognome: "Allegrini", matricola: 178066, annoDilmmatricolazione: 2024));
17        esame.iscrivi(new StudenteLavoratore( nome: "Giordano", cognome: "Bruni", matricola: 98034, annoDilmmatricolazione: 2018));
18        esame.iscrivi(new StudenteLavoratore( nome: "Giulio", cognome: "Andreotti", matricola: 34555, annoDilmmatricolazione: 2017));
19        esame.iscrivi(new Studente( nome: "Antonietta", cognome: "Reale", matricola: 151535, annoDilmmatricolazione: 2020));
20
21        // crea uno scanner connesso alla tastiera, legge uno studente da tale scanner
22        // e lo iscrive all'esame; se si verifica una StudenteIllegaleException, allora ripete
23        // la lettura dello studente e il tentativo di iscrizione, ad oltranza;
24        // notate che per leggere uno studente da uno scanner esiste un metodo già fatto in Studente
25        try (Scanner Keyboard = new Scanner(System.in)) {
26            while (true) {
27                try {
28                    esame.iscrivi(Studente.leggiDaTastiera(Keyboard));
29                    break;
30                } catch (StudenteIllegaleException | InputMismatchException e) {
31                    System.out.println("Studente illegale, riprova");
32                }
33            }
34        }
35
36        // stampa l'esame con tutti gli studenti iscritti
37        System.out.println("\n" + esame);
38
39        System.out.println("\nMatricole degli studenti fuori corso:");
40        // chiamate perOgniIscritto per stampare le matricole degli studenti fuori corso iscritti all'esame
41
42        Predicate<Studente> p = new Predicate<Studente>() { ± GiorgiaZanini
43            @Override ± GiorgiaZanini
44            public boolean test(Studente studente) { return studente.fuoriCorso(informatica); }
45        };
46
47
48        Consumer<Studente> c = new Consumer<Studente>() { ± GiorgiaZanini
49            @Override ± GiorgiaZanini
50            public void accept(Studente studente) { System.out.println(studente.getMatricola()); }
51        };
52
53
54        esame.perOgniIscritto(p,c);
55
56
57        System.out.println("\nStudenti lavoratori:");
58        // chiamate perOgniIscritto per stampare gli studenti lavoratori iscritti all'esame
59
60        p = new Predicate<Studente>() { ± GiorgiaZanini
61            @Override ± GiorgiaZanini
62            public boolean test(Studente studente) { return studente instanceof StudenteLavoratore; }
63        };
64
65
66        c = new Consumer<Studente>() { ± GiorgiaZanini
67            @Override ± GiorgiaZanini
68            public void accept(Studente studente) { System.out.println(studente.getMatricola()); }
69        };
70
71
72        esame.perOgniIscritto(p,c);
73
74
75        esame.perOgniIscritto(p,c);
76    }
77
78 }
```

Esame 2

```
© AbstractComponent.java    ⓘ Component.java ×    © DirectoryComponent.java    © FileComponent.java    © MainFiles.java
1 package it.univr.corso;
2
3 import java.io.FileNotFoundException;
4 import java.util.List;
5
6 public interface Component { 18 usages 3 implementations ▲ GiorgiaZanini
7
8   /**
9    * Ritorna il nome della componente (nome del file o nome della directory).
10   */
11  String getName(); 9 usages 1 implementation ▲ GiorgiaZanini
12
13  /**
14   * Ritorna una stringa che rappresenta la componente e le sottocomponenti, ricorsivamente, come
15   * nel testo del compito. Il contenuto di una directory deve venire riportato in ordine alfabetico
16   * crescente per nome.
17   */
18  String toString(); 1 implementation ▲ GiorgiaZanini
19
20  /**
21   * Identico a toString() sopra, ma la stringa ritornata ha i caratteri nesting alla sua sinistra, in ogni
22   * riga.
23   */
24  String toString(String nesting); 2 usages 2 implementations ▲ GiorgiaZanini
25
26  /**
27   * Ritorna la dimensione totale, in byte, della componente e delle sue sottocomponenti. Per i file, e'
28   * la dimensione del file. Per le directory, e' 100 piu' la somma delle dimensioni del contenuto della
29   * directory, ricorsivamente.
30   */
31  int size(); 3 usages 2 implementations ▲ GiorgiaZanini
32
33  /**
34   * Ritorna le componenti di tipo file raggiungibili da questa componente, ricorsivamente.
35   */
36  List<FileComponent> getFiles(); 3 usages 2 implementations ▲ GiorgiaZanini
37
38  /**
39   * Restituisce una descrizione del persorso con cui si raggiunge un file con il nome indicato, se
40   * esiste a partire da questa componente. Il risultato separa i nomi delle componenti con il carattere
41   * / Lancia l'eccezione se non esiste nessun file con il nome cercato.
42   */
43  String find(String name) throws FileNotFoundException; 2 usages 2 implementations ▲ GiorgiaZanini
44
45  }
46 }
```

```
© AbstractComponent.java ×    ⓘ Component.java    © DirectoryComponent.java    © FileComponent.java    © MainFiles.java
1 package it.univr.corso;
2
3 @l public abstract class AbstractComponent implements Component { 2 usages 2 inheritors ▲ GiorgiaZanini
4   private final String name; 2 usages
5
6   /**
7    * Costruisce una componente con il nome indicato.
8   */
9  protected AbstractComponent(String name) { this.name = name; }
10
11
12  /**
13   * @Override 9 usages ▲ GiorgiaZanini
14   */
15  public final String getName() { return name; }
16
17
18  /**
19   * @Override ▲ GiorgiaZanini
20   */
21  public final String toString() { return toString( nesting: ""); }
22
23  // sotto potete aggiungere solo cose private, se servissero;
24  // quindi non aggiungete metodi public sotto
25 }
```

```
© DirectoryComponent.java × © FileComponent.java © AbstractComponent.java © MainFiles.java
1 package it.univr.corso;
2
3 import java.io.FileNotFoundException;
4 import java.util.*;
5
6 public class DirectoryComponent extends AbstractComponent { 8 usages ▲ GiorgiaZanini
7     // aggiungete campi, se servissero
8     private SortedSet<Component> children = new TreeSet<>( 5 usages
9         ( Component thisComponent, Component otherComponent ) -> thisComponent.getName().compareTo(otherComponent.getName()));
10
11
12     Costruisce una componente di tipo directory con il nome indicato e le sottocomponenti (figli)
13     indicate.
14
15     public DirectoryComponent(String name, Component... children) { 10 usages ▲ GiorgiaZanini
16         super(name);
17         this.children.addAll(Arrays.asList(children));
18         //Component c[] = children;
19         //Arrays.sort(c);
20     }
21
22     // implementate sotto i metodi public ancora astratti
23
24     @Override 2 usages ▲ GiorgiaZanini
25     public String toString(String nesting) {
26         String string = nesting + getName() + "/";
27
28         for (Component component : children) {
29             string += "\n" + component.toString(nesting + "\t");
30         }
31
32         return string;
33     }
34
35     @Override 3 usages ▲ GiorgiaZanini
36     public int size() {
37         int size = 100;
38
39         for (Component component : children) {
40             if (component instanceof DirectoryComponent)
41                 size += component.size();
42
43             if (component instanceof FileComponent)
44                 size += component.size();
45         }
46
47         return size;
48     }
49 }
```

```
6  public class DirectoryComponent extends AbstractComponent { 8 usages ▲ GiorgiaZanini
49
50      @Override 3 usages ▲ GiorgiaZanini
51  ↗  public List<FileComponent> getFiles() {
52      List<FileComponent> list = new LinkedList<>();
53
54      for (Component component : children) {
55          if (component instanceof DirectoryComponent) {
56              list.addAll(component.getFiles());
57              //return list;
58          }
59          if (component instanceof FileComponent)
60              list.add((FileComponent) component);
61      }
62
63      return list;
64  }
65
66  @Override 2 usages ▲ GiorgiaZanini
67  ↗  public String find(String name) throws FileNotFoundException {
68      for (Component component : getFiles()) {
69          if (component.getName().equals(name))
70              return this.getName() + "/" + findRecursive(name);
71      }
72
73      throw new FileNotFoundException();
74  }
75
76  private String findRecursive(String name) { 2 usages ▲ GiorgiaZanini
77      String string = "";
78
79      for (Component component : children) {
80          if (component instanceof DirectoryComponent) {
81              string = ((DirectoryComponent) component).findRecursive(name);
82              if (!string.equals(""))
83                  return component.getName() + "/" + string;
84          }
85          if (component instanceof FileComponent) {
86              if (component.getName().equals(name))
87                  return component.getName();
88          }
89      }
90
91      return string;
92  }
93 }
```

(ricorsione)

© FileComponent.java × © AbstractComponent.java © MainFiles.java

```
1 package it.univr.corso;
2
3 import java.io.FileNotFoundException;
4 import java.util.List;
5
6 public class FileComponent extends AbstractComponent { 14 usages ▲ GiorgiaZanini
7     // aggiungete campi, se servissero
8     private final int size; 2 usages
9
10    /**
11     * Costruisce una componente di tipo file con il nome indicato
12     * e la dimensione in byte indicata.
13     */
14    public FileComponent(String name, int size) { 14 usages ▲ GiorgiaZanini
15        super(name);
16        this.size = size;
17    }
18
19    // implementate sotto i metodi public ancora astratti
20
21    @Override 2 usages ▲ GiorgiaZanini
22    public String toString(String nesting) { return nesting + getName(); }
23
24    @Override no usages ▲ GiorgiaZanini
25    public int size() { return size; }
26
27    @Override 3 usages ▲ GiorgiaZanini
28    public List<FileComponent> getFiles() { return null; }
29
30    @Override 2 usages ▲ GiorgiaZanini
31    public String find(String name) throws FileNotFoundException {
32        return "";
33    }
34}
```

© MainFiles.java ×

```
1 package it.univr.corso;
2
3 import java.io.FileNotFoundException;
4
5 public class MainFiles { ▲ GiorgiaZanini
6
7    public static void main(String[] args) throws FileNotFoundException { ▲ GiorgiaZanini
8        Component f1 = new FileComponent( name: "cat1.jpg", size: 34590 );
9        Component f2 = new FileComponent( name: "dog.gif", size: 12422 );
10       Component f3 = new FileComponent( name: "cat2.jpg", size: 52402 );
11       Component images = new DirectoryComponent( name: "images", f1, f2, f3 );
12       Component music = new DirectoryComponent( name: "music" ); // directory vuota
13       Component f4 = new FileComponent( name: "Pippo.java", size: 3255 );
14       Component f5 = new FileComponent( name: "Paperino.c", size: 44341 );
15       Component work = new DirectoryComponent( name: "work", f4, f5 );
16       Component f6 = new FileComponent( name: "passwords.txt", size: 3233 );
17       Component root = new DirectoryComponent( name: "root", work, images, f6, music );
18       System.out.println(root);
19       System.out.println();
20       System.out.println("total size: " + root.size() + " bytes");
21       System.out.println("files: " + root.getFiles());
22       System.out.println("dog.gif si trova come " + root.find( name: "dog.gif" ));
23       System.out.println("Pluto.c si trova come " + root.find( name: "Pluto.c" ));
24    }
25 }
```

Esame 4

```
② Emoji.java ×
1 package it.univr.sudoku;
2
3 // enumerazione di 9 emoji: non modificate
4 public enum Emoji { 3 usages ▲ GiorgiaZanini
5     GRINNING( s: "😁" ), no usages
6     MONEY( s: "💰" ), no usages
7     THINKING( s: "🤔" ), no usages
8     LAUGHING( s: "🤣" ), no usages
9     TONGUE( s: "👅" ), no usages
10    HEARTS( s: "❤️" ), no usages
11    MELTING( s: "融化" ), no usages
12    HALO( s: "光环" ), no usages
13    UPSIDE_DOWN( s: "🙃" ); no usages
14
15    private final String s; 2 usages
16
17    >     private Emoji(String s) { this.s = s; }
18
19    @Override
20    >     public String toString() { return s; }
21
22 }
```

```
② Emoji.java      ③ Sudoku.java ×
1 package it.univr.sudoku;
2
3 import java.util.Random;
4 import java.util.function.IntFunction;
5
6 // un sudoku in cui i numeri da 1 a 9 vengono rappresentati con elementi di tipo E
7 public class Sudoku<E> { 5 usages ▲ GiorgiaZanini
8
9     // le caselle del sudoku: contengono numeri tra 1 e 9,
10    // oppure 0, che indica una casella vuota;
11    // la casella (0,0) è quella in alto a sinistra;
12    // la casella (8,8) è quella in basso a destra
13    private final int[][] matrix = new int[9][9]; 12 usages
14
15    // una funzione che dato un numero tra 1 e 9 restituisce
16    // l'elemento di tipo E per rappresentare quel numero
17    private final IntFunction<E> generator; 2 usages
18
19    // la dimensione della stampa di un elemento di tipo E:
20    // si assume che tutti gli elementi di tipo E abbiano
21    // la stessa dimensione di stampa; questa informazione vi
22    // risulterà utile quando dovete stampare le caselle
23    // vuote del sudoku, per capire quanti spazi dovrete fare
24    private final int elementSize; 3 usages
25
26    public Sudoku(int empty, IntFunction<E> generator) { 5 usages ▲ GiorgiaZanini
27        // completare lanciando una IllegalArgumentException se empty
28        // non fosse tra 0 e 61 inclusi
29        if (empty < 0 || empty > 61)
30            throw new IllegalArgumentException("numero caselle vuote non valido");
31
32        this.generator = generator;
33
34        // calcoliamo quanto e' lunga la stampa dell'elemento che rappresenta 1:
35        // stiamo assumendo che la stampa di tutti gli elementi abbia la stessa lunghezza
36        this.elementSize = generator.apply( value: 1 ).toString().length();
37
38        // genera un Sudoku casuale completo
39        generate();
40
41        // cancella empty caselle a caso (mettendoci 0)
42        hide(empty);
43    }
```

```
Emoji.java      Sudoku.java ×
 7  public class Sudoku<E> {  5 usages  ▲ GiorgiaZanini
44
45      // restituisce una stringa che descrive il sudoku;
46      // si tratta della stampa della matrice 9x9,
47      // i cui elementi da 1 a 9 vengono prima trasformati
48      // nell'oggetto di tipo E corrispondente, usando la funzione generator,
49      // e poi trasformati in stringhe;
50      // si inseriscano anche le barrette di separazione orizzontale
51      // e verticale fra le 9 regioni del sudoku
52  ↗  public String toString() {  ▲ GiorgiaZanini
53      String result = "";
54      // completare
55      for (int j = 0; j < 9; j++) {
56          for (int i = 0; i < 9; i++) {
57              if (matrix[i][j] == 0) {
58                  result += ".repeat(elementSize);
59              } else {
60                  result += generator.apply(matrix[i][j]);
61              }
62
63              if (i == 2 || i == 5)
64                  result += "|";
65
66              result += "\n";
67              if (j == 2 || j == 5)
68                  result += "-".repeat( count: 9 * elementSize + 2 ) + "\n";
69          }
70
71      return result;
72  }
73
74  private final static Random random = new Random();  3 usages
75
76      // nasconde (cioè pone a 0) esattamente howMany caselle del sudoku,
77      // scelte a caso fra quelle che non sono già a 0
78  ↗  private void hide(int howMany) {  1 usage  ▲ GiorgiaZanini
79      // completare
80      int i;
81      int j;
82      for (int n = 0; n < howMany; n++) {
83          do {
84              i = random.nextInt( bound: 9 );
85              j = random.nextInt( bound: 9 );
86          } while (matrix[i][j] == 0);
87          matrix[i][j] = 0;
88      }
89
90
91      // genera un sudokSudoku risolto (già fatto, non dovete modificare nulla)
92  >  private void generate() { generate( x: 0, y: 0 ); }
93
94
95      // funzione ausiliaria per generare un sudoku risolto (non modificate)
96  ↗  private boolean generate(int x, int y) {  2 usages  ▲ GiorgiaZanini
97      int start = random.nextInt( bound: 9 );
98      int nextX = (x + 1) % 9;
99      int nextY = (nextX == 0) ? y + 1 : y;
100
101      for (int num = start; num < start + 9; num++) {
102          matrix[x][y] = (num % 9) + 1;
103          if (isLegal(x, y) && (nextY == 9 || generate(nextX, nextY)))
104              return true;
105      }
106
107      matrix[x][y] = 0;
108
109      return false;
110  }
111
112
113      // determina se l'elemento alle coordinate x,y è legale (non modificate)
114  ↗  private boolean isLegal(int x, int y) {  1 usage  ▲ GiorgiaZanini
115      return isHorizontallyUnique(x, y)
116          && isVerticallyUnique(x, y)
117          && isUniqueInRegion(x, y);
118  }
119
```

```
Emojo.java Sudoku.java
7 public class Sudoku<E> { 5 usages ± GiorgiaZanini
119
120     // determina se l'elemento alle coordinate x,y è unico nella sua riga
121     private boolean isHorizontallyUnique(int x, int y) { 1 usage ± GiorgiaZanini
122         // completare
123         for (int i = 0; i <= 8; i++) {
124             if (i != x) {
125                 if (matrix[i][y] == matrix[x][y])
126                     return false;
127             }
128         }
129         return true;
130     }
131
132     // determina se l'elemento alle coordinate x,y è unico nella sua colonna
133     private boolean isVerticallyUnique(int x, int y) { 1 usage ± GiorgiaZanini
134         // completare
135         for (int i = 0; i <= 8; i++) {
136             if (i != y) {
137                 if (matrix[x][i] == matrix[x][y])
138                     return false;
139             }
140         }
141         return true;
142     }
143
144     // determina se l'elemento alle coordinate x,y è duplicato nella sua regione
145     private boolean isUniqueInRegion(int x, int y) { 1 usage ± GiorgiaZanini
146         // completare
147         /*...*/
148
149         // determina numero quadrante
150         int i = x/3;
151         int j = y/3;
152
153         for (int h = 0; h < 3; h++) {
154             for (int k = 0; k < 3; k++) {
155                 if (((i * 3) + h) != x && ((j * 3) + k) != y) {
156                     if (matrix[(i * 3) + h][(j * 3) + k] == matrix[x][y]) // (i * 3) + h -> cella corrente da controllare nel quadrante i
157                         return false;
158                 }
159             }
160         }
161         return true;
162     }
163 }
164 }
```

```
Emojo.java Sudoku.java Main.java
1 package it.univr.sudoku;
2
3 import java.util.Arrays;
4 import java.util.function.IntFunction;
5
6 public class Main { ± GiorgiaZanini
7
8     public static void main(String[] args) { ± GiorgiaZanini
9         IntFunction<Integer> integer = new IntFunction<Integer>() { ± GiorgiaZanini
10            @Override ± GiorgiaZanini
11            public Integer apply(int i) { return i; }
12        };
13
14        System.out.println("Un sudoku di interi (1-9) con 61 caselle nascoste");
15        System.out.println(new Sudoku<Integer>(empty: 61, integer)); // completare
16        System.out.println("Un sudoku di interi (1-9) con 0 caselle nascoste");
17        System.out.println(new Sudoku<Integer>(empty: 0, integer)); // completare
18        System.out.println("Un sudoku di caratteri (A-I) con 30 caselle nascoste");
19        System.out.println(new Sudoku<Character>(empty: 30, int i -> (char) ('A'+i-1))); // completare //A+i-1
20        System.out.println("Un sudoku di emoji " + Arrays.toString(Emoji.values()) + " con 20 caselle nascoste");
21        System.out.println(new Sudoku<Emoji>(empty: 20, int i -> Emoji.values()[i-1])); // completare
22        System.out.println("Un sudoku di interi (1-9) con 62 caselle nascoste");
23        System.out.println(new Sudoku<Integer>(empty: 62, integer)); // completare, va in eccezione
24
25    }
26 }
```

Esame 5

```
IllegalProgramException.java ×  
1 package it.univr.instructions;  
2  
3 public class IllegalProgramException extends RuntimeException { ↗ GiorgiaZanini  
4 >     public IllegalProgramException(String message) { super(message); }  
5 }  
6
```

```
IllegalProgramException.java Instruction.java ×  
1 package it.univr.instructions;  
2  
3 import java.util.List;  
4  
5 public interface Instruction { 7 implementations ↗ GiorgiaZanini  
6  
    Esegue questa istruzione con lo stack indicato. Tipicamente, l'esecuzione modifichera' lo stack.  
    Params: stack - lo stack su cui si esegue l'istruzione  
    Throws: IllegalProgramException - se l'esecuzione dell'istruzione fallisce  
8  
    void execute(List<Integer> stack) throws IllegalProgramException; 7 implementations  
9  
10  
    Restituisce una descrizione stringa dell'istruzione.  
11  
12     String toString(); 7 implementations ↗ GiorgiaZanini  
13 }  
14
```

```
IllegalProgramException.java Instruction.java ADD.java ×  
1 package it.univr.instructions;  
2  
3 import java.util.List;  
4  
5 public class ADD implements Instruction{ 3 usages ↗ GiorgiaZanini  
6  
    public ADD() {} 3 usages ↗ GiorgiaZanini  
8  
    @Override ↗ GiorgiaZanini  
10     public void execute(List<Integer> stack) throws IllegalProgramException {  
11         int size = stack.size();  
12         if (size < 2)  
13             throw new IllegalProgramException("Errore: Operandi insufficienti per un'operazione binaria");  
14         // se l'operazione non va a buon fine, rimetto i numeri nello stack (in questo caso non li rimuovo nemmeno dalla lista)  
15  
16         /*...*/  
17         //new PUSH(stack.get(size - 2) + stack.get(size - 1)); // no prima di caricare il risultato devo togliere i1 e i2  
18  
19         // pop1 -> i2  
20         // pop2 -> i1  
21  
22         int result = stack.get(size - 2) + stack.get(size - 1);  
23         new POP().execute(stack);  
24         new POP().execute(stack);  
25         new PUSH(result).execute(stack);  
26     }  
27  
28     @Override ↗ GiorgiaZanini  
29     public String toString() { return "add"; }  
30 }
```

IllegalProgramException.java Instruction.java POP.java PUSH.java

```
1 package it.univr.instructions;
2
3 import java.util.List;
4
5 public class PUSH implements Instruction { 9 usages ± GiorgiaZanini
6     private Integer c; 3 usages
7
8     public PUSH(Integer c) { 9 usages ± GiorgiaZanini
9         this.c = c;
10    }
11
12    @Override ± GiorgiaZanini
13    public void execute(List<Integer> stack) throws IllegalProgramException {
14        int size = stack.size();
15        // in caso l'errore sarebbe throw new StackOverflowError(); se si supera il nuro di elementi massimo (non consciuto)
16        // ipotizzo che il valore massimo di inserimento è il numero massimo di int
17        if (size >= Integer.MAX_VALUE)
18            //throw new StackOverflowError("spazio nello stack finito");
19            throw new IllegalProgramException("StackOverflowError: spazio nello stack finito");
20        // se l'operazione non va a buon fine, non aggiungo il numero allo stack
21
22        stack.add(c); // aggiunge l'elemento in cima allo stack
23    }
24
25    @Override ± GiorgiaZanini
26    public String toString() {
27        return "push(" + c + ")";
28    }
29}
30
```

IllegalProgramException.java Instruction.java POP.java

```
1 package it.univr.instructions;
2
3 import java.util.List;
4
5 public class POP implements Instruction { 8 usages ± GiorgiaZanini
6
7     public POP() { 8 usages ± GiorgiaZanini
8     }
8
9
10    @Override ± GiorgiaZanini
11    public void execute(List<Integer> stack) throws IllegalProgramException {
12        int size = stack.size(); // n
13        if (size == 0)
14            // lo stack non deve essere vuoto, altrimenti non c'e' una cima da rimuovere
15            throw new IllegalProgramException("Pop da uno stack vuoto");
16
17        stack.remove( index: size - 1); // elimina l'elemento in cima allo stack // n - 2
18    }
19
20    @Override ± GiorgiaZanini
21    > public String toString() { return "pop"; }
22}
23
```

```
IllegalProgramException.java  Instruction.java  SUB.java  MUL.java  DIV.java  REPEAT.java
1 package it.univr.instructions;
2
3 import java.util.List;
4
5 public class SUB implements Instruction { no usages ▾ GiorgiaZanini
6
7     public SUB() {} no usages ▾ GiorgiaZanini
8
9     @Override ▾ GiorgiaZanini
10    public void execute(List<Integer> stack) throws IllegalProgramException {
11        int size = stack.size();
12        if (size < 2)
13            throw new IllegalProgramException("Errore: Operandi insufficienti per un'operazione binaria");
14        // se l'operazione non va a buon fine, rimetto i numeri nello stack (in questo caso non li rimuovo nemmeno dalla lista)
15
16        // pop1 -> i2
17        // pop2 -> i1
18
19        int result = stack.get(size - 2) - stack.get(size - 1);
20        new POP().execute(stack);
21        new POP().execute(stack);
22        new PUSH(result).execute(stack);
23    }
24
25    @Override ▾ GiorgiaZanini
26    public String toString() { return "sub"; }
27}
28
29
30
```

```
IllegalProgramException.java  Instruction.java  DIV.java  MUL.java  REPEAT.java
1 package it.univr.instructions;
2
3 import java.util.List;
4
5 public class MUL implements Instruction { no usages ▾ GiorgiaZanini
6
7     public MUL() {} no usages ▾ GiorgiaZanini
8
9     @Override ▾ GiorgiaZanini
10    public void execute(List<Integer> stack) throws IllegalProgramException {
11        int size = stack.size();
12        if (size < 2)
13            throw new IllegalProgramException("Errore: Operandi insufficienti per un'operazione binaria");
14        // se l'operazione non va a buon fine, rimetto i numeri nello stack (in questo caso non li rimuovo nemmeno dalla lista)
15
16        // pop1 -> i2
17        // pop2 -> i1
18
19        int result = stack.get(size - 2) * stack.get(size - 1);
20        new POP().execute(stack);
21        new POP().execute(stack);
22        new PUSH(result).execute(stack);
23    }
24
25    @Override ▾ GiorgiaZanini
26    public String toString() { return "mul"; }
27}
28
29
30
```

```
IllegalProgramException.java     Instruction.java     DIV.java     REPEAT.java
1 package it.univr.instructions;
2
3 import java.util.List;
4
5 public class DIV implements Instruction { no usages ± GiorgiaZanini
6
7     public DIV() {} no usages ± GiorgiaZanini
8
9     @Override ± GiorgiaZanini
10    public void execute(List<Integer> stack) throws IllegalProgramException {
11        int size = stack.size();
12        if (size < 2)
13            throw new IllegalProgramException("Errore: Operandi insufficienti per un'operazione binaria");
14        // se l'operazione non va a buon fine, rimetto i numeri nello stack (in questo caso non li rimoovo nemmeno dalla lista)
15
16        // pop1 -> i2
17        // pop2 -> i1
18
19        int i2 = stack.get(size - 1);
20        if (i2 == 0)
21            throw new IllegalProgramException("i2 = 0");
22
23        int result = stack.get(size - 2) - i2;
24        new POP().execute(stack);
25        new POP().execute(stack);
26        new PUSH(result).execute(stack);
27    }
28
29    @Override ± GiorgiaZanini
30    public String toString() { return "div"; }
31
32 }
```

```
IllegalProgramException.java     Instruction.java     REPEAT.java
1 package it.univr.instructions;
2
3 import java.util.List;
4
5 public class REPEAT implements Instruction { 4 usages ± GiorgiaZanini *
6     private final Integer c; // != da c delle istruzioni (non passare come parametro), 3 usages
7                                         // serve per sapere quante volte eseguire l'operazione passata
8     private final Instruction ins; 3 usages
9
10    public REPEAT(Integer c, Instruction ins) { 4 usages ± GiorgiaZanini
11        this.c = c;
12        this.ins = ins;
13        //System.out.println(this.ins.toString());
14    }
15
16    @Override ± GiorgiaZanini
17    public void execute(List<Integer> stack) throws IllegalProgramException {
18        for (int i = 0; i < c; i++)
19            ins.execute(stack);
20    }
21
22    @Override ± GiorgiaZanini
23    public String toString() { return "repeat(" + c + ", " + ins.toString() + ")"; }
24
25 }
```

```
① Machine.java ×
1 package it.univr.instructions;
2
3 ② public interface Machine { 1 usage 2 implementations ▾ GiorgiaZanini
4     int getResult(); 2 usages 1 implementation ▾ GiorgiaZanini
5 }
```

```
① Machine.java      © SimpleMachine.java ×
1 package it.univr.instructions;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 ③ public class SimpleMachine implements Machine { 2 usages 1 inheritor ▾ GiorgiaZanini
7
8     // inizialmente lo stack e' vuoto
9     private final List<Integer> stack = new ArrayList<>(); 3 usages
10
11    private final int result; 2 usages
12
13    Esegue il programma indicato.
14
15    Params: program – il programma da eseguire
16
17    Throws: IllegalProgramException – se l'esecuzione del programma fallisce per qualche motivo
18
19    public SimpleMachine(List<Instruction> program) throws IllegalProgramException { 2 usages ▾ GiorgiaZanini
20        result = execute(program);
21    }
22
23
24 ④ @Override 2 usages ▾ GiorgiaZanini
25    public final int getResult() { return result; }
26
27
28    Esegue le istruzioni del programma, dall'inizio alla fine, partendo da uno stack vuoto. Dopo avere
29    eseguite tutte le istruzioni, restituisce l'elemento in cima allo stack risultante alla fine.
30
31    Params: program – il programma da eseguire
32
33    Returns: l'elemento che c'e' in cima allo stack alla fine dell'esecuzione del programma
34
35    Throws: IllegalProgramException – se l'esecuzione del programma va in errore oppure se
36            alla fine dell'esecuzione lo stack e' vuoto (e quindi non c'e' nessun elemento in cima
37            da potere ritornare)
38
39    private int execute(List<Instruction> program) throws IllegalProgramException { // TODO 1 usage ▾ GiorgiaZanini
40        for (int i = 0; i < program.size(); i++) {
41            execute(program.get(i), stack);
42        }
43        return stack.get(stack.size() - 1);
44    }
45
46
47    Esegue un'istruzione sullo stack indicato.
48
49    Params: ins – l'istruzione da eseguire
50    Stack: stack – lo stack su cui eseguire l'istruzione
51    Returns: l'elemento in cima allo stack dopo aver eseguito l'istruzione
52
53    Throws: IllegalProgramException – se l'esecuzione dell'istruzione fallisce
54
55    protected void execute(Instruction ins, List<Integer> stack) throws IllegalProgramException { 1 override ▾ GiorgiaZanini
56        ins.execute(stack);
57    }
58 }
```

Machine.java SimpleMachine.java PrintingMachine.java

```
1 package it.univr.instructions;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class PrintingMachine extends SimpleMachine{ 1 usage  ↳ GiorgiaZanini
7     //private final List<Integer> stack = new ArrayList<>();
8
9     public PrintingMachine(List<Instruction> program) throws IllegalProgramException { 1 usage  ↳ GiorgiaZanini
10        super(program);
11    }
12
13    /*...*/
14
15    @Override  ↳ GiorgiaZanini
16    protected void execute(Instruction ins, List<Integer> stack) throws IllegalProgramException {
17        System.out.print(ins + ": ");
18        ins.execute(stack);
19        System.out.println(stack);
20    }
21
22}
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
```

Main.java

```
1 package it.univr.instructions;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class Main {  ↳ GiorgiaZanini
7
8     public static void main(String[] args) {  ↳ GiorgiaZanini
9         List<Instruction> program = new ArrayList<>();
10        program.add(new PUSH(c: 5));
11        program.add(new PUSH(c: 13));
12        program.add(new PUSH(c: 17));
13        program.add(new ADD());
14
15        execute(program);
16
17        List<Instruction> program2 = new ArrayList<>();
18        program2.add(new REPEAT(c: 5, new PUSH(c: 13)));
19        program2.add(new REPEAT(c: 4, new ADD()));
20        execute(program2);
21
22        List<Instruction> program3 = new ArrayList<>();
23        program3.add(new REPEAT(c: 5, new PUSH(c: 13)));
24        program3.add(new REPEAT(c: 5, new ADD()));
25        execute(program3);
26    }
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
```

```
private static void execute(List<Instruction> program) { 3 usages  ↳ GiorgiaZanini
1    System.out.println(" * * * Esegua " + program + " * * *\n");
2    try {
3        System.out.println("Simple machine:");
4        int result = new SimpleMachine(program).getResult();
5        System.out.println("result = " + result + "\n");
6    }
7    catch (IllegalProgramException e) {
8        System.out.println("Errore: " + e.getMessage() + "\n");
9    }
10
11    try {
12        System.out.println("Printing machine:");
13        int result = new PrintingMachine(program).getResult();
14        System.out.println("result = " + result + "\n");
15    }
16    catch (IllegalProgramException e) {
17        System.out.println("Errore: " + e.getMessage() + "\n");
18    }
19}
```

```

© PunishableSet.java × © Main.java

1 package it.univr.sets;
2
3 import java.util.HashMap;
4 import java.util.Iterator;
5 import java.util.Map;
6
7 // l'unico metodo di questa interfaccia di libreria "consuma"
8 // un elemento, cioè è definito come:
9 // void accept(E element);
10 // in questo esercizio usiamo questa interfaccia per specificare cosa fare
11 // quando un elemento viene aggiunto o tolto da un insieme
12 import java.util.function.Consumer;
13
14 // l'unico metodo di questa interfaccia di libreria è una funzione da E a int,
15 // cioè è definito come:
16 // int applyAsInt(E element);
17 // in questo esercizio usiamo questa interfaccia per specificare come inizializzare
18 // il punteggio degli elementi aggiunti all'insieme
19 import java.util.function.ToIntFunction;
20

Un insieme i cui elementi hanno un punteggio positivo. E' possibile aggiungere e togliere elementi. E' possibile punire o perdonare un elemento per una certa quantità di punti, il che significa che il suo punteggio viene ridotto o aumentato di tale quantità (rispettivamente). Se il punteggio diventasse non positivo, l'elemento verrebbe rimosso dall'insieme. Iterando su un oggetto di questa classe si devono ottenere i suoi elementi.

💡 Type parameters: <E> – il tipo degli elementi dell'insieme

32 public class PunishableSet<E> implements Iterable<E> { 11 usages GiorgiaZanini
33
34     Gli elementi dell'insieme, con il loro punteggio attuale.
35
36     private final Map<E, Integer> container = new HashMap<E>(); 12 usages
37
38     La funzione che specifica come inizializzare il punteggio degli elementi, quando vengono aggiunti all'insieme.
39
40     private final ToIntFunction<E> init; 3 usages
41
42     Il codice che viene eseguito quando un elemento viene aggiunto all'insieme.
43
44     private final Consumer<E> onAdd; 2 usages
45
46     Il codice che viene eseguito quando un elemento viene rimosso dall'insieme.
47
48     private final Consumer<E> onRemove; 2 usages
49
50
51
52
53
54
55
56

```

```
© PunishableSet.java x © Main.java
32 public class PunishableSet<E> implements Iterable<E> { 11 usages ▾ GiorgiaZanini
33
34     /**
35      * Costruisce un insieme contenente inizialmente esattamente gli elementi indicati, con il punteggio
36      * dato dalla funzione init.
37
38      * Params: elements – gli elementi iniziali
39      * init – la funzione (da E a int) usata per inizializzare il punteggio degli elementi
40      * aggiunti all'insieme
41      * onAdd – la funzione che verrà chiamata ogni volta che si aggiunge un elemento
42      * onRemove – la funzione che verrà chiamata ogni volta che si toglie un elemento
43
44     @
45     public PunishableSet(E[] elements,ToIntFunction<E> init, Consumer<E> onAdd, Consumer<E> onRemove) {
46         this.init = init;
47         this.onAdd = onAdd;
48         this.onRemove = onRemove;
49
50         // COMPLETARE
51         for (E element : elements) {
52             add(element);
53         }
54     }
55
56
57     /**
58      * Costruisce un insieme contenente inizialmente gli elementi indicati, con il punteggio dato dalla
59      * funzione init. Le funzioni da chiamare in caso di aggiunta o rimozione non fanno nulla (il che
60      * equivale a dire che si usano onAdd e onRemove che dato un elemento qualsiasi eseguono il
61      * codice vuoto {}).
62
63      * Params: elements – gli elementi iniziali
64      * init – la funzione (da E a int) usata per inizializzare il punteggio degli elementi
65      * aggiunti all'insieme
66
67     public PunishableSet(E[] elements,ToIntFunction<E> init) { 4 usages ▾ GiorgiaZanini
68         // COMPLETARE
69         this(elements, init, E onAdd -> {}, E onRemove -> {});
70     }
71
72
73     /**
74      * Costruisce un insieme contenente inizialmente gli elementi indicati. Il punteggio di un elemento
75      * aggiunto all'insieme è inizialmente 1000 (il che equivale a dire che si usa una init che ritorna
76      * sempre 1000).
77
78      * Params: elements – gli elementi iniziali
79
80     public PunishableSet(E[] elements) { 1 usage ▾ GiorgiaZanini
81         // COMPLETARE
82         this(elements, E init -> 1000);
83     }
84
85
86
87
88
89
89
90
91
92
93
94
95
```

```

32     public class PunishableSet<E> implements Iterable<E> { 11 usages ▾ GiorgiaZanini
  ↗
  | Prova ad aggiungere element all'insieme, con un punteggio iniziale dato dalla funzione init. Il
  | valore di ritorno indica se l'elemento è stato aggiunto o meno. Questa funzione dovrebbe
  | ritornare false solo in due casi: 1) se element fosse già nell'insieme; oppure 2) se il valore iniziale
  | dato dalla funzione init fosse non positivo. Nel caso in cui ritorna true, questa funzione esegue il
  | codice di onAdd.

  | Params: element – l'elemento da aggiungere
  |
  | Returns: true se l'elemento è stato aggiunto; false se l'elemento non è stato aggiunto

119    public boolean add(E element) { 2 usages ▾ GiorgiaZanini
120        // COMPLETARE
121        if (init.applyAsInt(element) < 0)
122            return false;
123        if (container.containsKey(element))
124            return false;

125
126        container.put(element, init.applyAsInt(element));
127        onAdd.accept(element);
128        return true;
129    }
130

Prova ad eliminare element dall'insieme. Il valore di ritorno indica se l'elemento è stato eliminato o
meno. Questa funzione dovrebbe ritornare false solo se element non fosse già nell'insieme. Nel
caso in cui ritorna true, questa funzione esegue il codice di onRemove.

Params: element – l'elemento da rimuovere
Returns: true se l'elemento è stato rimosso; false se l'elemento non è stato rimosso

142    public boolean remove(E element) { 4 usages ▾ GiorgiaZanini
143        // COMPLETARE
144        if (!container.containsKey(element))
145            return false;

146
147        container.remove(element);
148        onRemove.accept(element);
149        return true;
150    }
151

Punisce element togliendogli points punti. Se, così facendo, i punti di element diventassero non
positivi, questa funzione rimuove element dall'insieme. Se element non fosse nell'insieme, questa
funzione non ha effetto (oltre all'eventuale eccezione).

Params: element – l'elemento da punire
      points – quanti punti devono venire tolti a element

Throws: IllegalArgumentException – se points fosse negativo

162    public void punish(E element, int points) { 4 usages ▾ GiorgiaZanini
163        // COMPLETARE
164        if (points < 0)
165            throw new IllegalArgumentException("points negativo");
166        if (!container.containsKey(element))
167            //throw new IllegalArgumentException(element + " non è contento nell'insieme");
168            return;

169
170        Integer value = container.get(element);
171        if (value != null)
172            points = value - points;
173
174    > /*...*/
175    > /*...*/
176    if (points > 0)
177        container.put(element, points);
178    else
179        remove(element);
180    }
181
182
183
184
185
186
187
188
189
190

```

```
© PunishableSet.java × © Main.java
32 public class PunishableSet<E> implements Iterable<E> { 11 usages ▾ GiorgiaZanini
33
34     /**
35      * Perdona element aggiungendogli points punti. Se element non fosse nell'insieme, questa
36      * funzione non ha effetto (oltre all'eventuale eccezione).
37
38      * Params: element – l'elemento da perdonare
39      *          points – quanti punti devono venire aggiunti al punteggio di element
40
41      * Throws: IllegalArgumentException – se points fosse negativo
42
43
44     public void pardon(E element, int points) {  no usages ▾ GiorgiaZanini
45         // COMPLETARE
46         if (points < 0)
47             throw new IllegalArgumentException("points negativo");
48         if (!container.containsKey(element))
49             //throw new IllegalArgumentException(element + " non è contento nell'insieme");
50             return;
51
52         Integer value = container.get(element);
53         if (value != null)
54             points = value + points;
55
56         remove(element);
57         container.put(element, points);
58     }
59
60
61     /**
62      * Ritorna una stringa che enumera su due colonne gli elementi dell'insieme, con a lato il loro
63      * punteggio (si veda l'esempio nel compito).
64
65     @Override ▾ GiorgiaZanini
66     public String toString() { // FATTO, NON MODIFICARE
67         String result = "";
68
69         for (E element: this)
70             result += element + ": " + container.get(element) + " points\n";
71
72         return result;
73     }
74
75     // QUI VA AGGIUNTO UN SOLO METODOO public
76
77     @Override ▾ GiorgiaZanini
78     public Iterator<E> iterator() { return container.keySet().iterator(); }
79 }
```

```
② PunishableSet.java ③ Main.java ×

1 package it.univr.sets;
2
3 public class Main { ▲ GiorgiaZanini
4
5 >     public static void main(String[] args) { new Main(new String[] { "Fausto", "Samantha", "Giulio", "Giovanna" }); }
6
7
8     /**
9      * Crea alcuni insieme contenenti i nomi indicati e li stampa.
10     */
11
12     private Main(String[] names) { 1 usage ▲ GiorgiaZanini
13         // crea un PunishableSet contenente names inizializzati a 1000 punti
14         PunishableSet<String> set1 = new PunishableSet<>(names); // COMPLETARE
15         play( setName: "set1", set1);
16
17         // crea un PunishableSet contenente names inizializzati a una
18         // quantita' di punti pari alla propria lunghezza (cioè Aurora 6 punti,
19         // Samantha 7 punti ecc.)
20         PunishableSet<String> set2 = new PunishableSet<>(names, String init -> init.length()); // COMPLETARE
21         play( setName: "set2", set2);
22
23         // crea un PunishableSet contenente names inizializzati a una
24         // quantita' di punti pari al proprio numero di vocali (cioè Fausto 3 punti,
25         // Giulio 4 punti ecc.)
26         /*...*/
27         PunishableSet<String> set3 = new PunishableSet<>(names, String init -> countVowels(init)); // COMPLETARE
28         play( setName: "set3", set3);
29
30         // crea un PunishableSet contenente names inizializzati a 2000 punti
31         PunishableSet<String> set4 = new PunishableSet<>(names, String init -> 2000); // COMPLETARE
32         play( setName: "set4", set4);
33
34         // crea un PunishableSet contenente names inizializzati a 600 punti,
35         // in modo che quando si aggiunge un elemento "s" venga stampato "adding s"
36         // e che quando si rimuove un elemento "s" venga stampato "removing s"
37         PunishableSet<String> set5 = new PunishableSet<>(names, String init -> 600, String onAdd -> adding(onAdd), String onRemove -> removing(onRemove)); // COMPLETARE
38         play( setName: "set5", set5);
39
40         // questo dovrebbe generare una IllegalArgumentException perché points è negativo
41         set5.punish( element: "Fausto", points: -2);
42     }
43
44     // esegue alcune operazioni sull'insieme "set" e alla fine lo stampa
45     private void play(String setName, PunishableSet<String> set) { 5 usages ▲ GiorgiaZanini
46         set.add("Aurora");
47         set.punish( element: "Fausto", points: 500);
48         set.punish( element: "Giovanna", points: 100);
49         set.punish( element: "Giovanna", points: 900);
50         set.remove("Giulio");
51         set.remove("Giulia");
52         System.out.println(setName + ":" + "\n" + set + "\n");
53     }
54
55     // conta il numero di vocali contenute in "s"
56     private int countVowels(String s) { 1 usage ▲ GiorgiaZanini
57         int counter = 0;
58
59         for (int pos = 0; pos < s.length(); pos++)
60             if ("aeiouAEIOU".indexOf(s.charAt(pos)) != -1)
61                 counter++;
62
63         return counter;
64     }
65
66     // stampa "adding s"
67     private void adding(String s) { System.out.println("adding " + s); }
68
69     // stampa "removing s"
70     private void removing(String s) { System.out.println("removing " + s); }
71
72 }
```

Esame 7

```
© Tessera.java × © FattoriaDiTessere.java © FattoriaDiTessereAlfabetiche.java © FattoriaDiTessereNumeriche.java © Gioco.java
1 package it.univr.quindici;
2
3     // Una tessera contiene un valore di tipo T. Si richiede che gli oggetti di tipo T
4     // siano comparabili fra di loro, in modo da sapere se una tessera viene prima di un'altra
5     public final class Tessera<T extends Comparable<T>> implements Comparable<Tessera<T>> { 10 usages ▲ GiorgiaZanini
6
7         // il valore contenuto dentro la tessera
8         private final T s; 7 usages
9
10        // crea una tessera che contiene il valore indicato
11        > Tessera(T s) { this.s = s; }
12
13        public boolean equals(Object other) { ▲ GiorgiaZanini
14            // due tessere sono uguali se contengono valori uguali
15            return other instanceof Tessera && ((Tessera<?>) other).s.equals(s);
16        }
17
18        public int hashCode() { // modificate: non deve essere banale ▲ GiorgiaZanini
19            //return (Integer) s;
20            return s.hashCode();
21        }
22
23        public String toString() { return s.toString(); // intanto -> vedere se modificare }
24
25        public int compareTo(Tessera<T> other) { return this.s.compareTo(other.s); }
26    }
```

```
© FattoriaDiTessere.java × © FattoriaDiTessereAlfabetiche.java © FattoriaDiTessereNumeriche.java © Gioco.java
1 package it.univr.quindici;
2
3 import java.util.Random;
4 import java.util.function.Supplier;
5
6     // Una fattoria di tessere e' un oggetto capace di generare una nuova
7     // tessera ogni volta che si chiama il suo metodo get() (definito in Supplier).
8     // Al suo interno mettiamo un oggetto Random che puo essere usato nelle sottoclassi
9     // per fare scelte casuali
10
11     /! non modificate
12     public abstract class FattoriaDiTessere<T extends Comparable<T>> implements Supplier<Tessera<T>> {
13         protected final static Random random = new Random(); 3 usages
14     }
```

```
© FattoriaDiTessereAlfabetiche.java © FattoriaDiTessereNumeriche.java × © Gioco.java © Main.java
1 package it.univr.quindici;
2
3     public class FattoriaDiTessereNumeriche extends FattoriaDiTessere<Integer> { 1 usage ▲ GiorgiaZanini
4
5         private final int max; 2 usages
6
7         > public FattoriaDiTessereNumeriche (int max) { this.max = max; }
8
9         @Override ▲ GiorgiaZanini
10         > public Tessera<Integer> get() { return new Tessera<Integer> (random.nextInt(max - 1) + 1); }
11     }
```

```
© FattoriaDiTessereAlfabetiche.java × © FattoriaDiTessereNumeriche.java © Gioco.java © Main.java
1 package it.univr.quindici;
2
3 public class FattoriaDiTessereAlfabetiche extends FattoriaDiTessere<String> { 1 usage ▾ GiorgiaZanini
4
5     public FattoriaDiTessereAlfabetiche () { 1 usage ▾ GiorgiaZanini
6     }
7
8     private final static String lettere = "abcdefghijklmnopqrstuvwxyz"; // 26 1 usage
9
10    @Override ▾ GiorgiaZanini
11    public Tessera<String> get() {
12        String string = "";
13
14        int max;
15        do {
16            max = random.nextInt( bound: 6 );
17        } while (max == 0);
18
19        for (int i = 0; i < max; i++)
20            string += lettere.charAt(random.nextInt( bound: 26));
21        return new Tessera<String>(string);
22    }
23}
24
```

© Gioco.java × © Main.java

```
1 package it.univr.quindici;
2
3 import java.util.ArrayList;
4 import java.util.List;
5 import java.util.Random;
6
7 /\ un gioco con tessere che contengono valori di tipo T
8 public class Gioco<T extends Comparable<T>> { 3 usages ▲ GiorgiaZanini
9     private final int width, height; 6 usages
10
11    // le tessere del gioco: questa lista le contiene per righe
12    // (la prima riga seguita dalla seconda riga seguita dalla terza riga ecc.);
13    // questa lista contiene null nel punto vuoto del gioco
14    private final List<Tessera<T>> tessere; 8 usages
15
16    public Gioco(FattoriaDiTessere<T> fattoria, int width, int height) { 2 usages ▲ GiorgiaZanini
17        this.width = width;
18        this.height = height;
19
20        // costruite la lista "tessere": dovra' contenere
21        // tessere casuali tutte diverse fra di loro
22        // e un elemento casuale a null: in totale la lista "tessera"
23        // dovra' avere quindi width * height elementi di cui uno solo a null
24
25        // modificate
26
27        Random random = new Random();
28        int tesseraNull = random.nextInt( bound: width * height);
29        tessere = new ArrayList<T>(initialCapacity: width * height);
30        for (int i = 0; i < width * height; i++) {
31            if (i == tesseraNull)
32                tessere.add(null);
33                //tessere.set(i, null);
34            else
35                tessere.add(fattoria.get());
36        }
37    }
38
39    // restituisce il gioco come una stringa: non modificate
40    @Override
41    public String toString() { ▲ GiorgiaZanini
42        String result = "";
43
44        for (int y = 0; y < height; y++) {
45            for (int x = 0; x < width; x++)
46                if (tessere.get(x + y * width) != null)
47                    result += String.format("%5s ", tessere.get(x + y * width).toString());
48                else
49                    result += "      ";
50
51            result += "\n";
52        }
53
54        return result;
55    }
56
57    // determina se il gioco e' risolto:
58    // 1) la posizione vuota deve essere in basso a destra, e
59    // 2) le tessere devono essere in ordine crescente (per riga e colonna)
60    public boolean risolto() { // modificate 1 usage ▲ GiorgiaZanini
61        if (tessere.get((width * height) - 1) != null)
62            return false;
63
64        for (int i = 0; i < (width * height) - 2; i++) {
65            if (tessere.get(i).compareTo(tessere.get(i + 1)) >= 0)
66                return false;
67            }
68        return true;
69    }
70}
```

```
>Main.java ×  
1 package it.univr.quindici;  
2  
3 D public class Main { ↗ GiorgiaZanini  
4 D     public static void main(String[] args) { ↗ GiorgiaZanini  
5         // gioco 4x4 con tessere alfabetiche di lunghezza da 1 a 5 inclusi  
6         System.out.println(new Gioco<>(new FattoriaDiTessereAlfabetiche(), width: 4, height: 4));  
7  
8         FattoriaDiTessere<Integer> f = new FattoriaDiTessereNumeriche( max: 8);  
9         Gioco<Integer> gioco;  
10        do {  
11            // gioco 3x2 con tessere numeriche da 1 a 8 inclusi  
12            gioco = new Gioco<>(f, width: 3, height: 2);  
13            System.out.println(gioco);  
14        }  
15        while (!gioco.risolto());  
16    }  
17 }  
18
```

Esame 8

```
Person.java ×
1 package it.univr.doodle;
2
3 // si renda questa classe comparabile con un'altra Person:
4 // ordinando prima per priorita' crescente e poi alfabeticamente per nome
5 /* nota: dovrete aggiungere un metodo public: quale?
6 ⑥ public abstract class Person implements Comparable<Person> { 22 usages 4 inheritors ▲ GiorgiaZanini
7     private final String name; 6 usages
8
9     > protected Person(String name) { this.name = name; }
12
13 @Override ▲ GiorgiaZanini
14 ⑤ > public String toString() { return name; }
17
18 ④ public abstract int priority(); 7 usages 4 implementations ▲ GiorgiaZanini
19
20 @Override ▲ GiorgiaZanini
21 ⑤ > public boolean equals(Object other) { // devono avere stesso nome e stessa priorità
22     if (other == null)
23         return false;
24     if (!(other instanceof Person))
25         return false;
26
27     return (this.priority() == ((Person) other).priority()) && (this.name.equals(((Person) other).name));
28 }
29
30
31 ④@④
32 @Override ▲ GiorgiaZanini
33 public int compareTo(Person other) {
34     if (this.priority() == other.priority())
35         return this.name.compareTo(other.name);
36     else if (this.priority() > other.priority())
37         return 1;
38     return -1;
39 }
```

```
Person.java CEO.java ×
1 package it.univr.doodle;
2
3 public class CEO extends Person{ 1 usage ▲ GiorgiaZanini
4
5     > protected CEO(String name) { super(name); }
6
7
8
9     @Override 7 usages ▲ GiorgiaZanini
10 ④ > public int priority() { return 4; }
11
12 }
13
14
```

```
Person.java Secretary.java Programmer.java ×
1 package it.univr.doodle;
2
3 public class Programmer extends Person{ 1 usage ▲ GiorgiaZanini
4
5     > protected Programmer(String name) { super(name); }
6
7
8
9     @Override 7 usages ▲ GiorgiaZanini
10 ④ > public int priority() { return 2; }
11
12 }
13
14
```

```
Person.java CEO.java CTO.java ×
1 package it.univr.doodle;
2
3 public class CTO extends Person{ 1 usage ▲ GiorgiaZanini
4
5     > protected CTO(String name) { super(name); }
6
7
8
9     @Override 7 usages ▲ GiorgiaZanini
10 ④ > public int priority() { return 3; }
11
12 }
13
14
```

```
Person.java Secretary.java ×
1 package it.univr.doodle;
2
3 public class Secretary extends Person { 1 usage ▲ GiorgiaZanini
4
5     > protected Secretary(String name) { super(name); }
6
7
8
9     @Override 7 usages ▲ GiorgiaZanini
10 ④ > public int priority() { return 1; }
11
12 }
13
14
```

```
© Doodle.java ×
1 package it.univr.doodle;
2
3 import java.util.*;
4
5 public class Doodle { 3 usages 1 inheritor ▲ GiorgiaZanini
6     // per ogni persona, dice in quali slot temporali è disponibile
7     private final Map<Person, Set<Slot>> availabilities = new HashMap<>(); 8 usages
8     //private Set<Slot> w = new TreeSet<>();
9
10    // aggiunge gli slot temporali "when" a quelli disponibili per person
11    public void available(Person person, Slot... when) { 8 usages ▲ GiorgiaZanini
12        //
13        Set<Slot> w = new TreeSet<>(); // classi concrete
14            // HashSet<>(); -> non ordinato, più veloce
15            // TreeSet<>(); -> ordinato -> prende l'equals e il compare to dell'oggetto passato
16            //
17            // Set<Slot> w = new TreeSet<>(List.of(when));
18        /*
19        for (int i = 0; i < when.length; i++) {
20            w.add(when[i]);
21        }
22        */
23        // w.addAll(Arrays.asList(when)); // addAll(Collection<? extends Slot> collection) boolean
24        Collections.addAll(w, when);
25        availabilities.put(person, w);
26    }
27
28    // aggiunge gli slot temporali "when" a quelli disponibili per person
29    public void available(Person person, Iterable<Slot> when) { no usages ▲ GiorgiaZanini
30        //
31        //Set<Slot> w = new TreeSet<>();
32
33        if (availabilities.get(person) == null) {
34            Set<Slot> w = new TreeSet<>();
35
36            while (when.iterator().hasNext())
37                w.add(when.iterator().next());
38
39            availabilities.put(person, w);
40        } else {
41            //w = availabilities.get(person);
42            while (when.iterator().hasNext())
43                //w.add(when.iterator().next());
44            availabilities.get(person).add(when.iterator().next()); // get -> restituisce puntatore al Set,
45            //                // aggiorno il Set che stà già nella mappa
46
47            //availabilities.put(person, w);
48        }
49    }
}
```

```
© Doodle.java ×
 5  public class Doodle { 3 usages 1 inheritor ▲ GiorgiaZanini
 6
 7      // costruisce una tabella come negli esempi del compito:
 8      // nella prima riga tutti gli slot temporali in "availabilities",
 9      // in ordine crescente;
10      // poi le disponibilità di ciascuna persona in "availabilities"
11      // poi la priorità di ciascuno slot temporale (somma delle priorità di
12      // chi può partecipare), con un asterisco a lato del primo slot
13      // con priorità massima
14
15  @  public String toString() { ▲ GiorgiaZanini
16      String result = "";
17      //
18      // Slot disponibili tra tutte le persone
19      Set<Slot> wAll = new TreeSet<>();
20      Set<Person> persone = new TreeSet<>(availabilities.keySet());    // insieme ordinato delle persone
21      for (Person person : persone) {
22          wAll.addAll(availabilities.get(person));
23      }
24      for (Slot slot : wAll)
25          result += slot.toString() + "\t";
26
27      // lista disponibilità della persona per ogni slot
28      for (Person person : persone) {
29          result += "\n";
30
31          for (Slot slot : wAll) {
32              result += "\t";
33              if (availabilities.get(person).contains(slot)) {
34                  result += "yes";
35              } else {
36                  result += "no";
37              }
38              result += "\t\t\t";
39          }
40
41          result = result.trim();
42          result += "\t" + person;
43      }
44
45      // conteggi disponibilità, con '*' sulla maggiore
46      ArrayList<Integer> disponibilita = new ArrayList<>();
47      Integer counterDisponibilita;
48      int max = 0;
49
50      for (Slot slot : wAll) {
51          counterDisponibilita = 0;
52
53          for (Person person : persone) {
54              if (availabilities.get(person).contains(slot))
55                  counterDisponibilita += priority(person);
56          }
57
58          disponibilita.add(counterDisponibilita);
59      }
60
61      for (Integer dispMax : disponibilita) {
62          if (dispMax > max)
63              max = dispMax;
64      }
65
66      result += "\n";
67      for (Integer disp : disponibilita) {
68          result += "\t" + disp;
69          if (disp.equals(max))
70              result += "*";
71          result += "\t\t\t";
72      }
73
74      // return
75      return result.trim() + "\n";
76  }
77
78  @  protected int priority(Person person) { 1 usage 1 override ▲ GiorgiaZanini
79      return 1; // non modificate: per default i doodle danno a tutti la stessa priorità (1)
80  }
81
82  }
```

```
© Doodle.java © WeightedDoodle.java ×  
1 package it.univr.doodle;  
2  
3 public class WeightedDoodle extends Doodle { 2 usages ▾ GiorgiaZanini  
4  
5     @Override 1 usage ▾ GiorgiaZanini  
6     protected int priority(Person person) { return person.priority(); }  
7  
8 }  
9  
10
```

```
© Doodle.java © WeightedDoodle.java © Main.java ×  
1 package it.univr.doodle;  
2  
3 public class Main { ▾ GiorgiaZanini  
4     public static void main(String[] args) { ▾ GiorgiaZanini  
5         Slot s1 = new Slot( year: 2017, month: 2, day: 4, Slot.Moment.MORNING);  
6         Slot s2 = new Slot( year: 2017, month: 2, day: 4, Slot.Moment.AFTERNOON);  
7         Slot s3 = new Slot( year: 2017, month: 2, day: 5, Slot.Moment.AFTERNOON);  
8         Slot s4 = new Slot( year: 2017, month: 2, day: 5, Slot.Moment.EVENING);  
9         Slot s5 = new Slot( year: 2017, month: 2, day: 5, Slot.Moment.AFTERNOON); // come s3  
10  
11         Doodle doodle1 = new Doodle();  
12         Person ceo = new CEO( name: "Fausto");  
13         Person cto = new CTO( name: "Giovanni");  
14         Person secretary = new Secretary( name: "Alessandro");  
15         Person programmer = new Programmer( name: "Alessandra");  
16         doodle1.available(ceo, s2, s4);  
17         doodle1.available(cto, s1, s3, s4, s5);  
18         doodle1.available(secretary, s1, s3, s5);  
19         doodle1.available(programmer, s3);  
20         System.out.println("doodle1:\n" + doodle1);  
21  
22         WeightedDoodle doodle2 = new WeightedDoodle();  
23         doodle2.available(ceo, s2, s4);  
24         doodle2.available(cto, s1, s3, s4, s5);  
25         doodle2.available(secretary, s1, s3, s5);  
26         doodle2.available(programmer, s3);  
27         System.out.println("doodle2:\n" + doodle2);  
28     }  
29 }  
30
```

```
Letters.java × LowerCase.java Vulcanian.java Main.java

1 package it.univr.letters;
2
3 import java.util.function.Consumer;
4
5 /**
6  * Una sequenza di lettere (cioe' caratteri alfabetici).
7 */
8 public interface Letters {
9
10    /**
11     * Restituisce la lunghezza della sequenza.
12     */
13    int length(); // no usages 1 implementation GiorgiaZanini
14
15    /**
16     * Restituisce la concatenazione delle lettere della sequenza.
17     */
18    String toString(); // 1 implementation GiorgiaZanini
19
20    /**
21     * Applica il comando indicato alle lettere della sequenza (dalla prima all'ultima).
22     */
23    void forEach(Consumer<Character> command); // 2 usages 1 implementation GiorgiaZanini
24
25 }
```

```

1 package it.univr.letters;
2
3 import java.util.Random;
4 import java.util.function.Consumer;
5
6
7 Una sequenza di caratteri "minuscola", cioe' fatta da lettere minuscole dell'alfabeto inglese. Sono ammesse lettere ripetute.
8
9 @public class LowerCase implements Letters { 3 usages 1 inheritor ▲ GiorgiaZanini
10    protected final static Random random = new Random(); 2 usages
11    // AGGIUNGERE QUI CAMPI PRIVATI SE SERVISERO
12    private String sequenza = ""; 7 usages
13    private final static String lettereLower = "abcdefghijklmnopqrstuvwxyz"; //26 2 usages
14
15
16
17     Crea una sequenza minuscola casuale.
18
19     Params: length – la lunghezza della sequenza da creare
20
21     Throws: IllegalArgumentException – se length e' negativo
22
23     public LowerCase(int length) { 1 usage ▲ GiorgiaZanini
24         // COMPLETARE
25         if (length < 0)
26             throw new IllegalArgumentException("length non può essere negativo, lenght passato: " + length);
27
28         //sequenza = "";
29         for (int i = 0; i < length; i++) {
30             sequenza += lettereLower.charAt(random.nextInt(bound: 26));
31         }
32     }
33
34
35     Crea una sequenza minuscola fatta dai caratteri di s, identici, nello stesso ordine.
36
37     Params: s – la stringa che contiene i caratteri da inserire nella sequenza
38
39     Throws: IllegalArgumentException – se i caratteri di s non sono una sequenza minuscola
40
41     public LowerCase(String s) { 3 usages ▲ GiorgiaZanini
42         // COMPLETARE
43         char[] seq = s.toCharArray();
44         for (Character character : seq)
45             if (!(lettereLower.contains(character.toString())))
46                 throw new IllegalArgumentException("la sequenza contiene caratteri non compatibili con una sequenza randomica di caratteri minuscoli");
47         sequenza = s;
48     }
49
50
51     @Override no usages ▲ GiorgiaZanini
52     public final int length() { return sequenza.length(); }
53
54
55     @Override ▲ GiorgiaZanini
56     public final String toString() { return sequenza; }
57
58
59     @Override 2 usages ▲ GiorgiaZanini
60     public final void forEach(Consumer<Character> what) {
61         // COMPLETARE
62         char[] seqChar = sequenza.toCharArray();
63         sequenza = "";
64         for (Character character : seqChar) {
65             what.accept(character);
66             sequenza += character; // mette nella stringa il character modificato
67         }
68     }
69 }
70

```

```
© Vulcanian.java x © Main.java
1 package it.univr.letters;
2
3 import java.util.Arrays;
4
5 Una sequenza vulcaniana di lettere, cioe' un caso particolare di sequenza LowerCase che e' fatta da
6 due parti: la prima parte contiene vocali in ordine alfabetico; la seconda parte contiene consonanti in
7 ordine alfabetico.
8
9 public class Vulcanian extends LowerCase { 2 usages ▾ GiorgiaZanini
10
11     // AGGIUNGERE QUI CAMPI PRIVATI SE SERVISERO
12     private final static String lettereLower = "abcdefghijklmnopqrstuvwxyz";    //26 2 usages
13
14     /**
15      * Crea una sequenza vulcaniana di length lettere.
16
17      * Params: length – la lunghezza della sequenza da creare
18
19      * Throws: IllegalArgumentException – se length e' negativo
20
21     >     public Vulcanian(int length) { super(randomVulcanian(length)); }
22
23
24     /**
25      * Genera una stringa vulcaniana casuale lunga length.
26
27     @     private static String randomVulcanian(int length) { // MODIFICARE E COMPLETARE 1 usage ▾ GiorgiaZanini
28         String sequenza = "";
29
30         for (int i = 0; i < length; i++) {
31             sequenza += lettereLower.charAt(random.nextInt(bound: 26));
32         }
33
34         return ordinamentoVulcaniano(sequenza);
35     }
36
37     /**
38      * Crea una sequenza vulcaniana fatta dai caratteri di s, identici, nello stesso ordine.
39
40      * Params: s – la stringa che contiene i caratteri da inserire nella sequenza
41
42      * Throws: IllegalArgumentException – se i caratteri di s non formano una sequenza vulcaniana
43
44     >     public Vulcanian(String s) { // MODIFICARE E COMPLETARE 1 usage ▾ GiorgiaZanini
45         super(s);
46         if (!(s.equals(ordinamentoVulcaniano(s))))
47             throw new IllegalArgumentException("la sequenza non è in ordine vulcaniano");
48     }
49
50
51     /**
52      * private static String ordinamentoVulcaniano(String sequenza) { 2 usages ▾ GiorgiaZanini
53         char seq[] = sequenza.toCharArray();    // creo un array dalla sequenza passata
54         String vocali = "";
55         String consonanti = "";
56
57         Arrays.sort(seq);    // ordino i char, cosi sono già in ordine per poi separarli
58
59         for (Character character : seq) {
60             if (!(lettereLower.contains(character.toString())))
61                 throw new IllegalArgumentException("sequenza non convertibile a sequenza vulcaniana");
62
63             if (character == 'a' || character == 'e' || character == 'i' || character == 'o' || character == 'u')
64                 vocali += character;    // la prima parte di una sequenza vulcaniana sono le vocali
65             else
66                 consonanti += character;    // uso tmp per le vocali
67         }
68
69         return vocali + consonanti;
70     }
71 }
```

```
>Main.java ×
1 package it.univr.letters;
2
3 public class Main { ➜ GiorgiaZanini
4
5     public static void main(String[] args) { ➜ GiorgiaZanini
6         Letters l1 = new LowerCase( length: 5);
7         System.out.println("l1 = " + l1);
8         System.out.println("I suoi caratteri sono:");
9         l1.forEach(System.out::println);
10        Letters l2 = new Vulcanian( length: 20);
11        System.out.println("\nl2 = " + l2);
12        System.out.println("I suoi caratteri non italiani sono:");
13        l2.forEach( Character c -> {
14            if (c == 'j' || c == 'k' || c == 'w' || c == 'x' || c == 'y')
15                System.out.println(c);
16        });
17        System.out.println();
18        createLowerCase( s: "tmwdmgħbonqlwbwpigkv");
19        createLowerCase( s: "tmwd(ghbonqlwbwpigkv");
20        createLowerCase( s: "tmwDmghbonqlwbwpigkv");
21        createVulcanian( s: "iobcchjjknnnnnprsstx");
22        createVulcanian( s: "ioBcchjjknnnnnprsstx");
23        createVulcanian( s: "io(cchjjknnnnnprsstx");
24        createVulcanian( s: "ibocchjjknnnnnprsstx");
25        createVulcanian( s: "oibcchjjknnnnnprsstx");
26        createVulcanian( s: "iocbchjjknnnnnprsstx");
27    }
28
29     Prova a creare un oggetto LowerCase a partire dalla stringa s. Se ci riesce, stampa: new
30     LowerCase(s) => successo Se invece la creazione va in eccezione, stampa: new LowerCase(s)
31     => eccezione
32
33     private static void createLowerCase(String s) { 3 usages ➜ GiorgiaZanini
34         // COMPLETARE
35         try {
36             new LowerCase(s);
37             System.out.println("new LowerCase(" + s + ") => successo");
38         } catch (IllegalArgumentException e) {
39             System.out.println("new LowerCase(" + s + ") => eccezione");
40         }
41     }
42
43 }
```

Esame 10

```
© Partito.java x © Elezioni.java © ElezioniVincitore.java © VotiPerPartito.java © Main.java
1 package it.univr.elezioni;
2
3     Un partito ha un nome, passato al momento della costruzione.
4
5 public class Partito implements Comparable<Partito> { 20 usages ± GiorgiaZanini
6     private final String nome; 13 usages
7
8     public Partito(String nome) { 5 usages ± GiorgiaZanini
9         // completare
10        this.nome = nome;
11    }
12
13
14     Determina chi fra this e other viene prima in ordine alfabetico per nome.
15
16     @Override ± GiorgiaZanini
17     public int compareTo(Partito other) { // modificare
18         if ((this.nome == null) && (other.nome == null))
19             return 0;
20         else if (other.nome == null)
21             return -1;
22         else if (this.nome == null)
23             return 1;
24
25         return this.nome.compareTo(other.nome);
26     }
27
28
29     // due partiti sono uguali se e solo se hanno nome uguale
30     @Override ± GiorgiaZanini
31     public boolean equals(Object other) { // modificare
32         if (this.nome == null || !(other instanceof Partito) || ((Partito) other).nome == null) // (other == null) coperto da instanceof
33             return false;
34
35         return this.nome.equals(((Partito) other).nome);
36     }
37
38     @Override ± GiorgiaZanini
39     public int hashCode() { return nome.hashCode(); }
40
41
42     Restituisce il nome del partito.
43
44     @Override ± GiorgiaZanini
45     public String toString() { return nome; }
46
47     >
48 }
49
50 }
```

```
© Partito.java   © Elezioni.java ×  © ElezioniVincitore.java   © VotiPerPartito.java   © Main.java
1 package it.univr.elezioni;
2
3 import java.util.*;
4
5 /**
6  * Un oggetto di questa classe permette di registrare voti per dei partiti.
7  * Iterando su questo oggetto, si ottengono delle coppie partito/voti ottenuti,
8  * messe in ordine crescente per partito.
9 */
10 @ public class Elezioni implements Iterable<VotiPerPartito> { 2 usages 1 inheritor ± GiorgiaZanini
11
12     //private SortedSet<Partito> partiti = new TreeSet<>();
13     private Map<Partito, Integer> elezioni = new TreeMap<>(); 11 usages
14
15     // registra un voto per il partito indicato
16     public final void vota(Partito partito) { 1 usage ± GiorgiaZanini
17         // completare
18         if (elezioni.get(partito) == null)
19             elezioni.put(partito, 1);
20         else
21             elezioni.put(partito, elezioni.get(partito) + 1);
22         // elezioni.merge(partito, 1, Integer::sum);
23     }
24
25     /**
26      * Ritorna una stringa che descrive l'elezione, del tipo:
27
28      Bassotti: 4467 voti (28.11%)
29      Caotico: 4679 voti (29.45%)
30      Felice: 1591 voti (10.01%)
31      Floreale: 3950 voti (24.86%)
32      Pensionati: 1202 voti (7.56%)
33
34      I partiti sono riportati in ordine crescente, con a sinistra un indice
35      crescente del partito (da 1 in su). Dopo il nome del partito viene riportato
36      il numero dei voti ottenuti e la percentuale ottenuta fra tutti i voti espressi.
37 */
38 @Override 1 override ± GiorgiaZanini*
39 @ public String toString() { // modificare
40     float votiTot = 0;
41     for (Partito partito : elezioni.keySet()) {
42         votiTot += elezioni.get(partito);
43     }
44
45     String string = "";
46
47     int i = 1;
48     for (Partito partito : elezioni.keySet()) {
49         string += String.format("%d %15s: %5d voti (%.2f%%)\n", i, partito, elezioni.get(partito), ((100 * elezioni.get(partito)/votiTot)));    // %s -> stringa
50         i++;
51     }
52
53     return string;
54 }
55
56 Iterando su questo oggetto, si ottengono delle coppie partito/voti ottenuti, messe in ordine
crescente per partito.
57
58 @Override ± GiorgiaZanini
59 @ public final Iterator<VotiPerPartito> iterator() { // modificare
60     SortedSet<VotiPerPartito> set = new TreeSet<VotiPerPartito>(new Comparator<VotiPerPartito>() { ± GiorgiaZanini
61         @Override ± GiorgiaZanini
62         public int compare(VotiPerPartito thisVoti, VotiPerPartito otherVoti) {
63             return thisVoti.partito.compareTo(otherVoti.partito);
64         }
65     });
66
67     for (Partito partito : elezioni.keySet()) {
68         set.add(new VotiPerPartito(partito, elezioni.get(partito)));
69     }
70
71     return set.iterator();
72 }
73
74 }
75
76 }
```

© Partito.java © Elezioni.java © ElezioniVincitore.java © VotiPerPartito.java © Main.java

```
1 package it.univr.elezioni;
2
3 import java.util.Iterator;
4
5 /**
6  * Un tipo di elezione la cui stampa aggiunge l'indicazione di quale partito ha vinto le elezioni.
7 */
8
9 public class ElezioniVincitore extends Elezioni { 1 usage  ± GiorgiaZanini
10
11     /**
12      * Si comporta come il toString() della superclasse, ma in piu' aggiunge in fondo l'indicazione del
13      * partito che ha vinto le elezioni, del tipo "Vince X". In particolare, vince l'elezione il partito che ha
14      * ottenuto piu' voti. A parita' di voti, vince il partito che viene prima in ordine alfabetico. Se non ci
15      * fossero partiti (elezione vuota), aggiunge l'indicazione "Non ci sono vincitori".
16     */
17
18     @Override  ± GiorgiaZanini
19     public String toString() { // modificare
20         String vincitore = "";
21         int max = 0;
22
23         Iterator<VotiPerPartito> iterator = iterator();
24         if (iterator.hasNext()) {
25             if (iterator.next().voti > max) {
26                 vincitore = iterator.next().partito.toString();
27                 max = iterator.next().voti;
28             }
29             else if (iterator.next().voti == max)
30                 vincitore += " e " + iterator.next().partito;
31         }
32
33         return super.toString() + "\nVince " + vincitore;
34     }
35 }
36
37 }
```

© Partito.java © Elezioni.java © ElezioniVincitore.java © VotiPerPartito.java © Main.java

```
1 package it.univr.elezioni;
2
3 /**
4  * Una coppia partito/voti ottenuti.
5 */
6
7 public class VotiPerPartito { 9 usages  ± GiorgiaZanini
8     public final Partito partito; 6 usages
9     public final int voti; 5 usages
10
11     public VotiPerPartito(Partito partito, int voti) { 1 usage  ± GiorgiaZanini
12         this.partito = partito;
13         this.voti = voti;
14     }
15
16     @Override  ± GiorgiaZanini
17     public String toString() { return partito + ": " + voti; }
18 }
19 }
```

© Partito.java

© Elezioni.java

© ElezioniVincitore.java

© VotiPerPartito.java

© Main.java X

```
1 package it.univr.elezioni;
2
3 import java.util.Random;
4
5 public class Main { ± GiorgiaZanini
6
7     public static void main(String[] args) { ± GiorgiaZanini
8         // creo cinque partiti
9         Partito[] partiti = new Partito[] {
10             new Partito(nome: "Pensionati"),
11             new Partito(nome: "Felice"),
12             new Partito(nome: "Floreale"),
13             new Partito(nome: "Caotico"),
14             new Partito(nome: "Bassotti")
15         };
16
17         // elezioni inizialmente vuote
18         Elezioni elezioni = new ElezioniVincitore();
19
20         // aggiungo dei voti casuali per i partiti
21         Random random = new Random();
22         for (Partito partito: partiti) {
23             int voti = random.nextInt(bound: 10000);
24             for (int i = 0; i < voti; i++)
25                 elezioni.vota(partito);
26         }
27
28         // stampo le elezioni con indicazione di vincitore
29         System.out.println(elezioni);
30     }
31 }
```

Esame 11

```
© Dado.java ×
1 package it.univr.dadi;
2
3 import java.util.Random;
4
5
6     Un dado ha un numero prefissato di facce e può essere lanciato.
7
8 @l public abstract class Dado { 7 usages 3 inheritors ▲ GiorgiaZanini
9
10
11     Il numero di facce del dado. Si noti che e' pubblico.
12
13     public final int facce; 3 usages
14     private static final Random random = new Random(); 1 usage
15
16
17     Costruisce un dado con un numero prefissato di facce. Lancia IllegalArgumentException se il numero
18     di facce non e' positivo.
19
20     protected Dado(int facce) { 3 usages ▲ GiorgiaZanini
21         // completare
22         /*
23         if (facce < 0)
24             throw new IllegalArgumentException("il numero di facce di un dado non può essere negativo");
25         */
26
27         if (facce < 1)
28             throw new IllegalArgumentException("il numero di facce di un dado non può essere negativo");
29
30         /*
31         // ???
32         if (facce < 2) // in aggiunta, non può esistere un dado a 0 o 1 faccia
33             // anche se esistesse, in lancio si parte da 1
34             throw new IllegalArgumentException("non esiste un dado di " + facce + " facce");
35         */
36
37         this.facce = facce;
38     }
39
40
41     Restituisce un numero casuale fra 1 (incluso) e il numero di facce del dado (incluso).
42
43     public final int lancio() { 1 usage ▲ GiorgiaZanini
44         // completare
45         return random.nextInt(facce) + 1;
46     }
47 }
48
```

© Dado.java	© D6.java ×	© Dado.java	© D8.java ×	© Dado.java	© D10.java ×
1 package it.univr.dadi; 2 3 public class D6 extends Dado { 9 usages ▲ GiorgiaZanini 4 5 > protected D6() { super(facce: 6); } 6 7 } 8		1 package it.univr.dadi; 2 3 public class D8 extends Dado { 1 usage ▲ GiorgiaZanini 4 5 > protected D8() { super(facce: 8); } 6 7 } 8 9		1 package it.univr.dadi; 2 3 public class D10 extends Dado { 1 usage ▲ GiorgiaZanini 4 5 > protected D10() { super(facce: 10); } 6 7 } 8 9	

```
© Dado.java © Lanci.java ×

1 package it.univr.dadi;
2
3 import java.util.Arrays;
4
5 Una classe che rappresenta l'esecuzione di piu' lanci con dei dadi. Permette di vedere i risultati
6 ottenuti e la frequenza dei numeri ottenuti.
7
8 @public class Lanci { 6 usages 1 inheritor ▲ GiorgiaZanini
9
10     private final Dado[] dadi; 3 usages
11     private final int[] risultati; 9 usages
12     private int[] frequenza; 2 usages
13
14
15     Costruisce un'esecuzione di quanti lanci con i dadi indicati. Questo costruttore eseguire i lanci
16     richiesti con i dadi forniti e si salvera' le informazioni necessarie a implementare i metodi della
17     classe.
18
19     Params: quanti – il numero di lanci da eseguire
20             dadi – i dadi da lanciare. Per ogni lancio, il risultato e' la somma dei risultati di ciascun
21             dado
22
23     Throws: IllegalArgumentException – se quanti non e' positivo oppure se non vengono forniti
24             dadi da lanciare
25
26
27     @public Lanci(int quanti, Dado... dadi) { 5 usages ▲ GiorgiaZanini
28         // completare
29
30         // numero numero dadi da lanciare
31         if (dadi.length == 0)
32             throw new IllegalArgumentException("non sono stati forniti dadi da lanciare");
33         this.dadi = dadi;
34
35         // dadi da lanciare
36         if (quanti < 0) // numro di vote in cui lanciare i dadi
37             throw new IllegalArgumentException("il numero di lanci non può essere negativo");
38         risultati = new int[quanti];
39
40
41         // lancio dadi
42         int risultato;
43         for (int i = 0; i < risultati.length; i++) {
44             risultato = 0;
45             for (int j = 0; j < dadi.length; j++) {
46                 risultato += dadi[j].lancio();
47             }
48             risultati[i] = risultato;
49         }
50     }
51
52 }
```

© Dado.java © Lanci.java ×

```
9     public class Lanci { 6 usages 1 inheritor ▲ GiorgiaZanini
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57     @Override ▲ GiorgiaZanini
58     public final String toString() { return Arrays.toString(risultati); }
59
60
61
62
63
64
65
66
67
68
69
70     public final String frequenze() { 5 usages ▲ GiorgiaZanini
71         // completare
72
73         // inizializzazione array della frequenza dei risultati
74         int risultatoMax = 0;
75         for (Dado dado : dadi)
76             risultatoMax += dado.facce;
77
78         frequenza = new int[risultatoMax]; // per default i valori sono 0
79
80         // frequenze risultati
81         for (int i = 0; i < risultati.length; i++) {
82             frequenza[risultati[i]-1] += 1;
83         }
84
85         // toString
86         String string = "";
87         float percentuale;
88         for (int i = dadi.length-1; i < risultatoMax; i++) {
89             percentuale = (100 * (float) occorrenze(risultato: i+1) / risultati.length);
90             //System.out.println(String.format("%f = (100 * %d / %d)", percentuale, occorrenze(i+1), risultati.length));
91             string += String.format("%3s: ", i+1) + barra(i, percentuale) + String.format(" (%.1f%%)\n", percentuale);
92         }
93         return string;
94     }
95
96
97
98
99
100
101
102
103     protected String barra(int i, double frequenza) { 1 usage 1 override ▲ GiorgiaZanini
104         // completare
105         return "*".repeat((int) frequenza);
106     }
107
108
109
110
111
112
113
114
115
116
117     private int occorrenze(int risultato) { 1 usage ▲ GiorgiaZanini
118         int counter = 0;
119         for (int i = 0; i < risultati.length; i++) {
120             if (risultati[i] == risultato)
121                 counter++;
122         }
123         return counter;
124     }
125 }
```

Restituisce una rappresentazione a istogramma delle frequenze dei numeri ottenuti dal costruttore lanciando i dadi. Per ogni numero ottenuto, questi istogrammi rappresentano quante volte quel numero e' stato ottenuto. Gli istogrammi sono fatti da una sequenza di asterischi lunga in modo proporzionale alla frequenza, seguita dalla frequenza tra parentesi tonde. Si vedano gli esempi nel testo del compito.

Restituisce una barra di lunghezza proporzionale alla frequenza indicata con cui si e' ottenuto il risultato "i" lanciando i dadi. Si trattera' di una sequenza di asterischi. Per esempio, se il numero 4 fosse uscito nel 15.3% dei casi, allora questo metodo verra' chiamato con i = 4 e frequenza = 15.3 e dovrà ritornare 15 asterischi.

Restituisce una barra di lunghezza proporzionale alla frequenza indicata con cui si e' ottenuto il risultato "i" lanciando i dadi. Si trattera' di una sequenza di asterischi. Per esempio, se il numero 4 fosse uscito nel 15.3% dei casi, allora questo metodo verra' chiamato con i = 4 e frequenza = 15.3 e dovrà ritornare 15 asterischi.

© Dado.java © Lanci.java © LanciBarreDiverse.java ×

```
1 package it.univr.dadi;
2
3     Un tipo speciale di classe Lanci dove le barre degli istogrammi sono stampate usando alternativamente
4     |    caratteri *, @ e + (la prima barra usa *, la second @, la terza + e cosi' via ciclicamente).
5
6
7
8
9 public class LanciBarreDiverse extends Lanci { 1 usage ▲ GiorgiaZanini
10
11     private String[] simboli = {"*", "@", "+"}; 1 usage
12
13     public LanciBarreDiverse(int quanti, Dado... dadi) { 1 usage ▲ GiorgiaZanini
14         // completare
15         super(quanti, dadi);
16     }
17
18     protected String barra(int i, double frequenza) { 1 usage ▲ GiorgiaZanini
19         i = i % 3;
20         return simboli[i].repeat((int) frequenza);
21     }
22
23 }
```

© Dado.java © Lanci.java © LanciBarreDiverse.java © Main.java ×

```
1 package it.univr.dadi;
2
3 public class Main { ▲ GiorgiaZanini
4
5     public static void main(String[] args) { ▲ GiorgiaZanini
6         System.out.println("Lanciamo 20 volte due dadi a sei facce");
7         Lanci l = new Lanci( quanti: 20, new D6(), new D6());
8         System.out.println("Lanci ottenuti: " + l);
9         System.out.println(l.frequenze());
10
11         System.out.println("Lanciamo 10000 volte un dado a sei facce e uno a dieci facce");
12         l = new Lanci( quanti: 10000, new D6(), new D10());
13         System.out.println(l.frequenze());
14
15         System.out.println("Lanciamo 10000 volte un dado a otto facce");
16         l = new Lanci( quanti: 10000, new D8());
17         System.out.println(l.frequenze());
18
19         System.out.println("Lanciamo 10000 volte tre dadi a sei facce");
20         l = new Lanci( quanti: 10000, new D6(), new D6(), new D6());
21         System.out.println(l.frequenze());
22
23         System.out.println("Lanciamo 10000 volte tre dadi a sei facce, usando barre diverse");
24         l = new LanciBarreDiverse( quanti: 10000, new D6(), new D6(), new D6());
25         System.out.println(l.frequenze());
26
27 }
```

Esame 12

```
Product.java x
1 package it.univr.supermarket;
2
3     Un prodotto del supermercato.
4 @  public abstract class Product implements Comparable<Product> { 23 usages 2 inheritors ▾ GiorgiaZanini
5
6         // nome del prodotto.
7
8             private final String name; 9 usages
9
10            Il prezzo in euro del prodotto.
11
12             private final double price; 8 usages
13
14            Il numero di millisecondi che formano 24 ore (un giorno).
15
16             public final static long _24_HOURS = 1000L * 60 * 60 * 24; 3 usages
17
18
19             Costruisce un prodotto.
20
21                 Params: name – il nome del prodotto
22                     price – il prezzo in euro del prodotto
23
24                 Throws: IllegalArgumentException – se name e' null oppure vuoto oppure price e' negativo
25
26             protected Product(String name, double price) { // TODO 2 usages ▾ GiorgiaZanini
27
28                 if (name == null || name.isEmpty())
29                     throw new IllegalArgumentException("nessun nome passato");
30
31                 this.name = name;
32
33                 if (price < 0)
34                     throw new IllegalArgumentException("il prezzo non può essere negativo");
35
36                 this.price = price;
37
38
39             @Override ▾ GiorgiaZanini
40             public final String toString() { return name; }
41
42
43             Confronta il momento di scadenza di questo prodotto con quello di un altro prodotto.
44
45                 Returns: un numero negativo se questo prodotto scade prima di other; un numero positivo se
46                     questo prodotto scade dopo other; 0 se questo prodotto e other scadono nello
47                     stesso momento (o entrambi non scadono mai)
48
49             protected abstract int compareExpiration(Product other); 3 usages 2 implementations ▾ GiorgiaZanini
50
51
52             Due prodotti sono equals se hanno stesso nome, stesso prezzo e stesso momento di scadenza.
53
54             @Override ▾ GiorgiaZanini
55             public final boolean equals(Object other) { // TODO
56
57                 if (!(other instanceof Product))
58                     return false;
59
60
61                 Product otherProduct = (Product) other;
62                 return (this.name.equals(otherProduct.name)) && (this.price == otherProduct.price) && (this.compareExpiration(otherProduct) == 0);
63
64
65             Non deve essere banale.
66
67             @Override ▾ GiorgiaZanini
68             public final int hashCode() { return name.hashCode() ^ (int) price; }
69
70
71             Determina se, al momento indicato, questo prodotto e' scaduto.
72
73                 Params: whenChecked – il momento indicato
74
75             public abstract boolean hasExpired(long whenChecked); 4 usages 2 implementations ▾ GiorgiaZanini
76
77
78             Determina se, al momento indicato, questo prodotto non e' ancora scaduto ma scadrà' nelle 24
79                 ore successive.
80
81                 Params: whenChecked – il momento indicato (in millisecondi da 1/1/1970)
82
83
84             public final boolean expiresInSubsequent24Hours(long whenChecked) { // TODO 2 usages ▾ GiorgiaZanini
85
86                 if (!hasExpired(whenChecked))
87                     return hasExpired(whenChecked: whenChecked + _24_HOURS);
88
89                 return false;
90 }
```

© Product.java ×

```
6     public abstract class Product implements Comparable<Product> { 23 usages 2 inheritors ▲ GiorgiaZanini
90    }
91
92        Restituisce il prezzo di questo prodotto, come e' stato fornito al costruttore.
93    >     public final double getPrice() { return price; }
94
95        Si comporta identicamente al getPrice() che c'e' sopra, ma restituisce un prezzo scontato del 40%
96        se, al momento indicato, questo prodotto non e' ancora scaduto ma scadrà nelle 24 ore
97        successive.
98
99            Params: whenChecked – il momento in cui si chiede di controllare il prezzo (in millisecondi da 1/
100            1/1970)
101
102            Returns: il prezzo del prodotto, possibilmente scontato
103
104        public final double getPrice(long whenChecked) {      // TODO 1 usage ▲ GiorgiaZanini
105            if (hasExpired(whenChecked))
106                return -1;
107            if (expiresInSubsequent24Hours(whenChecked))
108                return (60 * price) / 100;
109            return getPrice();
110            //return price;
111        }
112
113
114
115
116        Ordina i prodotti per momento di scadenza (si usi compareExpiration), poi per nome e infine per
117        prezzo (si usi il metodo statico della libreria Double.compare(double1, double2)).
118
119        @Override ▲ GiorgiaZanini
120    >     public final int compareTo(Product other) {
121            if (this.compareExpiration(other) == 0) {
122                if (this.name.equals(other.name))
123                    Double.compare(this.price, other.price);
124                else
125                    this.name.compareTo(other.name);
126            }
127            return this.compareExpiration(other);
128        }
129    }
130
131 }
```

© Product.java © ProductWithExpiration.java © ProductNotExpiring.java ×

```
1 package it.univr.supermarket;
2
3        ⚡ Un prodotto che non scade mai.
4
5    public class ProductNotExpiring extends Product { 2 usages ▲ GiorgiaZanini
6
7        // TODO: campi?
8
9
10        Costruisce un prodotto senza scadenza.
11
12            Params: name – il nome del prodotto
13            price – il prezzo in euro del prodotto
14
15            Throws: IllegalArgumentException – se name e' null oppure vuota oppure se price e' negativo
16
17    >     public ProductNotExpiring(String name, double price) { super(name, price); }
18
19        // TODO
20
21
22        // non hanno scadenza -> tutto a 0 o false
23
24
25        @Override 3 usages ▲ GiorgiaZanini
26    >     protected int compareExpiration(Product other) { return 0; }
27
28
29        @Override 4 usages ▲ GiorgiaZanini
30    >     public boolean hasExpired(Long whenChecked) { return false; }
31
32
33
34 }
```

Product.java ProductWithExpiration.java ×

```
1 package it.univr.supermarket;
2
3     Un prodotto con un momento di scadenza.
4
5 public class ProductWithExpiration extends Product { 11 usages ▾ GiorgiaZanini
6
7     // TODO: campi?
8     private long scadenza; 7 usages
9
10
11     Costruisce un prodotto con una scadenza.
12
13     Params: name – il nome del prodotto
14         price – il prezzo in euro del prodotto
15         whenProduced – il momento di produzione (millisecondi da 1/1/1970)
16         duration – il numero di giorni, successivi all'produzione, dopo i quali il prodotto scade
17
18     Throws: IllegalArgumentException – se name è null oppure vuota oppure se price è
19             negativo oppure se whenProduced è negativo oppure su duration è negativo
20
21     public ProductWithExpiration(String name, double price, long whenProduced, int duration) { // TODO 8 usages ▾ GiorgiaZanini
22         super(name, price);
23         if (whenProduced < 0 || duration < 0)
24             throw new IllegalArgumentException("la scadenza, data da whenProduced e duration, non può essere negativa");
25         scadenza = whenProduced + (duration * _24_HOURS);
26     }
27
28     // TODO
29
30
31     @Override 3 usages ▾ GiorgiaZanini
32     protected int compareExpiration(Product other) {
33         if (!(other instanceof ProductWithExpiration))
34             return 0;
35
36         if (this.scadenza > ((ProductWithExpiration) other).scadenza)
37             return 1;
38         else if (this.scadenza < ((ProductWithExpiration) other).scadenza)
39             return -1;
40         return 0;
41     }
42
43     @Override 4 usages ▾ GiorgiaZanini
44     public boolean hasExpired(long whenChecked) {
45         return ((whenChecked >= scadenza) || (whenChecked + _24_HOURS) >= scadenza);
46     }
47 }
```

© Product.java © ProductWithExpiration.java © Supermarket.java

```
1 package it.univr.supermarket;
2
3 import java.util.ArrayList;
4 import java.util.List;
5 import java.util.SortedSet;
6 import java.util.TreeSet;
7
8
9     In supermercato, cioe' un contenitore di prodotti.
10
11 public class Supermarket {  3 usages  ▲ GiorgiaZanini
12     // TODO: campi?
13     private SortedSet<Product> products = new TreeSet<>();  2 usages
14
15         Aggiunge i prodotti indicati a questo supermercato. Se ci fossero ripetizioni (cioe' due prodotti
16         equals), li aggiunge una volta sola.
17
18     > public void add(Product... products) { this.products.addAll(List.of(products)); }
19
20
21     Restituisce una stringa che descrive questo supermercato a questo istante (System.
22     currentTimeMillis). Si comporta quindi come il toString(when) che c'e' subito sotto, ma usa il
23     tempo corrente come momento in cui creare la descrizione del supermercato.
24
25     @Override  ▲ GiorgiaZanini
26     public String toString() { return toString(System.currentTimeMillis()); }
27
28
29     Restituisce una stringa che descrive questo supermercato al momento indicato. Sarà' una
30     sequenza di prodotti, in ordine crescente rispetto al loro compareTo(), ognuno descritto come
31     negli esempi che seguono: se il prodotto e' già scaduto al tempo when: "pane: expired"
32     altrimenti, se il prodotto scade nelle 24 ore successive a when: "mozzarella: 2.70 euros (special
33     offer)" altrimenti "uova: 2.50 euros" Si noti che nel secondo caso il prezzo riportato e' quello al
34     momento when, quindi scontato.
35
36     Params: when – Il momento in cui si vuole creare la descrizione del supermercato
37
38     Returns: la stringa
39
40     public String toString(Long when) {  2 usages  ▲ GiorgiaZanini
41         // TODO
42         String string = "";
43
44         for (Product product : products) {
45             string += product.toString() + ": ";
46
47                 if (product.hasExpired(when))
48                     string += "expired";
49                 else {
50                     string += String.format("%.2f euros", product.getPrice(when));
51                     if (product.expiresInSubsequent24Hours(when))
52                         string += " (special offer)";
53                 }
54
55                     string += "\n";
56
57         }
58
59         return string;
60     }
61
62 }
```

```
© Supermarket.java   © Main.java ×

1 package it.univr.supermarket;
2
3 public class Main { ▲ GiorgiaZanini
4     public static void main(String[] args) { ▲ GiorgiaZanini
5         long now = System.currentTimeMillis();
6
7         // la mozzarella costa 4 euro e 50 centesimi,
8         // e' prodotta adesso e scade 6 giorni dopo la produzione
9         Product p1 = new ProductWithExpiration( name: "mozzarella", price: 4.5, now, duration: 6);
10
11        Product p2 = new ProductWithExpiration( name: "pane", price: 1, now, duration: 3);
12        Product p3 = new ProductWithExpiration( name: "acqua", price: 2.2, now, duration: 500);
13        Product p4 = new ProductWithExpiration( name: "carote", price: 2, now, duration: 5);
14        Product p5 = new ProductWithExpiration( name: "uova", price: 2.5, now, duration: 10);
15        Product p6 = new ProductWithExpiration( name: "pane", price: 1.2, now, duration: 2);
16        Product p7 = new ProductWithExpiration( name: "cipolle", price: 1.7, now, duration: 5);
17
18        // uno strano prodotto, gia' scaduto al momento della produzione
19        Product p8 = new ProductWithExpiration( name: "melone marcio", price: 3, now, duration: 0);
20
21        // prodotti che non scadono
22        Product p9 = new ProductNotExpiring( name: "dentifricio", price: 2.98);
23        Product p10 = new ProductNotExpiring( name: "bagnoschiuma", price: 4.9);
24
25        // crea un supermercato con 10 prodotti
26        Supermarket sm = new Supermarket();
27        sm.add(p1, p2, p3, p4, p5, p6, p7, p8, p9, p10);
28
29        // stampa il supermercato nella settimana da now in avanti
30        printOneWeek(sm, now);
31    }
32
33 /**
34 * Stampa sette volte il supermercato indicato, all'inizio di ogni giorno di una
35 * settimana che comincia al tempo start. Questo vuol dire che questo metodo
36 * stamperà il supermercato:
37 *
38 * al tempo start
39 * al tempo start + 1 giorno
40 * al tempo start + 2 giorni
41 * al tempo start + 3 giorni
42 * al tempo start + 4 giorni
43 * al tempo start + 5 giorni
44 * al tempo start + 6 giorni
45 *
46 * @param sm il supermercato da stampare
47 * @param start il tempo a cui si inizia a stampare il supermercato
48 */
49 private static void printOneWeek(Supermarket sm, long start) { 1 usage ▲ GiorgiaZanini
50     // TODO
51     for (int i = 0; i < 7; i++) {
52         System.out.println(sm.toString(when: start + i));
53     }
54 }
55 }
```

Esame 13

```
① Identifier.java ×
1 package it.univr.identifiers;
2
3 ①↓ public interface Identifier { 1 usage 3 implementations ✎ GiorgiaZanini
4
5      // Restituisce la stringa che si ottiene stampando l'identificatore.
6 ①↑①↓     public String toString(); 1 implementation ✎ GiorgiaZanini
7 }
```

```
Identifier.java MultiWordIdentifier.java ×
1 package it.univr.identifiers;
2
3 import java.util.ArrayList;
4 import java.util.Iterator;
5 import java.util.LinkedList;
6 import java.util.List;
7
8 // un identificatore fatto da piu' parole in sequenza
9 @l public abstract class MultiWordIdentifier implements Identifier { 6 usages 2 inheritors ✎ GiorgiaZanini
10
11     // TODO
12     private final ArrayList<String> words; 6 usages
13     private final static String lettere = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"; 1 usage
14
15     // fallisce con una IllegalArgumentException se non c'e' nessuna parola
16     // o se c'e' una parola null oppure vuota
17     // o se una parola contiene un carattere non alfabetico
18     @protected MultiWordIdentifier(String... words) throws IllegalArgumentException {    // se non vengono passati parametri va in errore da solo
19         // TODO
20         if (words.length == 0)
21             throw new IllegalArgumentException();
22         for (String word : words) {
23             if (word == null || word.isEmpty())
24                 throw new IllegalArgumentException();
25             for (int i = 0; i < word.length(); i++) {
26                 if (!lettere.contains(((Character) word.charAt(i)).toString()))
27                     throw new IllegalArgumentException();
28             }
29         }
30         this.words = new ArrayList<>(List.of(words));
31     }
32
33     // fallisce con un'eccezione nelle stesse condizioni viste sopra
34     @protected MultiWordIdentifier(Iterable<String> words) throws IllegalArgumentException { 4 usages ✎ GiorgiaZanini
35         // TODO
36         /*...*/
37
38         this.words = new ArrayList<>();
39         Iterator<String> iterator = words.iterator();
40         while (iterator.hasNext()) {
41             String next = iterator.next();
42             if (next == null || next.isEmpty())
43                 throw new IllegalArgumentException();
44             this.words.add(next);
45         }
46         /*...*/
47     }
48 }
```

```
Identifier.java MultiWordIdentifier.java
9  public abstract class MultiWordIdentifier implements Identifier { 6 usages 2 inheritors ▾ GiorgiaZanini
58
59  @Override ▾ GiorgiaZanini
60  public final String toString() {
61      // TODO: si richiami il metodo ausiliario toString(pos, word)
62      // e si concatensi il risultato in un'unica stringa
63      String string = "";
64
65      /*
66       * for (String word : words)
67       *     string += toString(words.indexOf(word), word);
68       */
69      for (int i = 0; i < words.size(); i++) {
70          string += toString(i, words.get(i));
71      }
72
73      return string;
74  }
75
76  // restituisce la stringa con cui si stampa la componente pos-esima dell'identificatore
77  protected abstract String toString(int pos, String word); 1 usage 2 implementations ▾ GiorgiaZanini
78
79  // restituisce la concatenazione delle parole degli identificatori indicati
80  @protected static List<String> concat(MultiWordIdentifier... ids) { 2 usages ▾ GiorgiaZanini
81      // TODO
82      List<String> list = new ArrayList<>();
83      for (MultiWordIdentifier identifier : ids) {
84          list.addAll(identifier.words);
85      }
86      return list;
87  }
88
89  // POTETE AGGIUNGERE METODI PRIVATE
90 }
```

```
Identifier.java MultiWordIdentifier.java CamelStyleIdentifier.java SnakeStyleIdentifier.java
1  package it.univr.identifiers;
2
3  //TODO: fate compilare questa classe
4
5  public class SnakeStyleIdentifier extends MultiWordIdentifier { 4 usages ▾ GiorgiaZanini
6
7      // costruisce un identificatore snake-style le cui parole sono quelle indicate;
8      // fallisce nelle stesse condizioni del costruttore della superclasse
9      public SnakeStyleIdentifier(String... words) { no usages ▾ GiorgiaZanini
10          // TODO
11          super(words);
12      }
13
14      // come sopra
15      public SnakeStyleIdentifier(Iterable<String> words) { 1 usage ▾ GiorgiaZanini
16          // TODO
17          super(words);
18      }
19
20      @Override 1 usage ▾ GiorgiaZanini
21      protected String toString(int pos, String word) {
22          if (pos == 0)
23              return word;
24          return "_" + word;
25      }
26
27      // costruisce un identificatore snake-style le cui parole componenti
28      // sono la concatenazione delle parole degli ids
29      public SnakeStyleIdentifier(MultiWordIdentifier... ids) { 1 usage ▾ GiorgiaZanini
30          // TODO
31          super(concat(ids));
32          //MultiWordIdentifier.concat(ids);
33      }
34
35      // restituisce un identificatore camel-style con le stesse parole di this
36      public CamelStyleIdentifier toCamelStyle() { no usages ▾ GiorgiaZanini
37          // TODO
38          return new CamelStyleIdentifier( ...ids: this);
39      }
40  }
```

```
① Identifier.java      ② MultiWordIdentifier.java    ③ CamelStyleIdentifier.java ×
1 package it.univr.identifiers;
2
3 // TODO: fate compilare questa classe
4
5 public class CamelStyleIdentifier extends MultiWordIdentifier { 6 usages ▾ GiorgiaZanini
6
7     // costruisce un identificatore camel-style le cui parole sono quelle indicate;
8     // fallisce nelle stesse condizioni del costruttore della superclasse
9     public CamelStyleIdentifier(String... words) { no usages ▾ GiorgiaZanini
10         // TODO
11         super(words);
12     }
13
14     // come sopra
15     public CamelStyleIdentifier(Iterable<String> words) { 1 usage ▾ GiorgiaZanini
16         // TODO
17         super(words);
18     }
19
20     @Override 1 usage ▾ GiorgiaZanini
21     protected String toString(int pos, String word) {
22         if (pos == 0)
23             return word.toLowerCase();
24         String string = "";
25         Character character;
26         for (int i = 0; i < word.length(); i++) {
27             character = word.charAt(i);
28             if (i == 0)
29                 string += character.toString().toUpperCase();
30             else
31                 string += character.toString().toLowerCase();
32         }
33         return string;
34     }
35
36     // costruisce un identificatore camel-style le cui parole componenti
37     // sono la concatenazione delle parole degli ids
38     public CamelStyleIdentifier(MultiWordIdentifier... ids) { 2 usages ▾ GiorgiaZanini
39         // TODO
40         super(concat(ids));
41         //super(MultiWordIdentifier.concat(ids));
42     }
43
44     // restituisce un identificatore snake-style con le stesse parole di this
45     public SnakeStyleIdentifier toSnakeStyle() { 1 usage ▾ GiorgiaZanini
46         // TODO
47         return new SnakeStyleIdentifier( ...ids: this);
48     }
49 }
```

① Identifier.java

© MultiWordIdentifier.java

© CamelStyleIdentifier.java

>Main.java ×

```
1 package it.univr.identifiers;
2
3 import java.util.ArrayList;
4 import java.util.List;
5 import java.util.Scanner;
6
7 public class Main { ± GiorgiaZanini
8
9     public static void main(String[] args) { ± GiorgiaZanini
10        Iterable<String> words = readWords();
11
12        CamelStyleIdentifier id1 = new CamelStyleIdentifier(words);
13        System.out.println("id1 = " + id1);
14        SnakeStyleIdentifier id2 = new SnakeStyleIdentifier(words);
15        System.out.println("id2 = " + id2);
16        CamelStyleIdentifier id3 = new CamelStyleIdentifier(id1, id2);
17        System.out.println("id3 = " + id3);
18        System.out.println("id3 snake style = " + id3.toSnakeStyle());
19    }
20
21    // legge da tastiera una sequenza di parole
22    private static Iterable<String> readWords() { 1 usage ± GiorgiaZanini
23        System.out.println("Inserisci una parola alla volta e termina con END");
24        List<String> words = new ArrayList<>();
25
26        try (Scanner keyboard = new Scanner(System.in)) {
27            while (true) {
28                String word = keyboard.nextLine();
29                if (word.equals("END"))
30                    return words; // "END" non fa parte della sequenza ritornata
31
32                words.add(word);
33            }
34        }
35    }
36}
```

