

Programmazione II

A.A. 2022-23

Prof. Maria Tortorella



Strumento per la modellazione

Unified Modelling Language

Introduzione al Class Diagram

UML

- Alcune discipline ingegneristiche dispongono di validi mezzi di rappresentazione (schemi, diagrammi di prestazioni e consumi, ...)
- Lo **Unified Modelling Language (UML)** fornisce uno strumento utile per la progettazione di un sistema software, indipendente dal linguaggio di programmazione che deve essere usato per la sua implementazione

Diagrammi UML

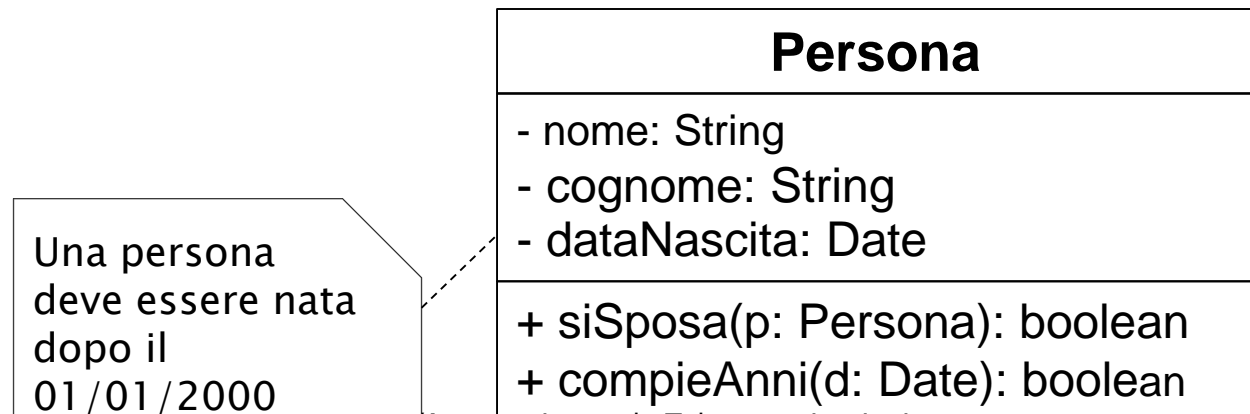
- Diagrammi di struttura
 - **diagrammi delle classi**, diagrammi degli oggetti, diagrammi dei componenti, diagrammi delle strutture composte, diagrammi dei package e i diagrammi di deployment
- Diagrammi di comportamento
 - diagrammi dei casi d'uso, diagrammi delle attività e diagrammi delle macchine a stati
- Diagrammi di interazione
 - diagrammi di sequenza, diagrammi di comunicazione, diagrammi di temporizzazione e diagrammi di interazione generale OCL (Object Constraint Language)

Diagramma delle classi

- Il diagramma delle classi consente di esprimere graficamente un sistema software a **livelli crescenti di dettaglio**
- Questi livelli crescenti di dettaglio permettono di utilizzare tale diagramma anche nella fase di specifica dei requisiti, anche se non sono stati definiti con tale scopo
- Diventano invece essenziali nella descrizione dell'**architettura della soluzione**, dove le classi corrispondono esattamente alle classi da implementare in un programma

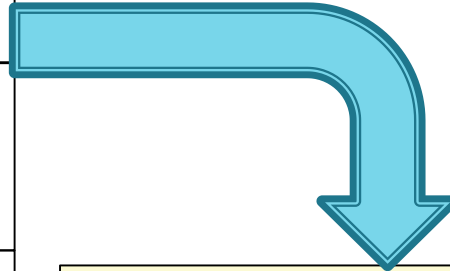
Composizione di una classe

- Nel diagramma delle classi, ciascuna classe è composta da tre parti
 - Nome
 - Attributi / Variabili d'istanza (lo stato)
 - Metodi (il comportamento)
- Metodo: **visibilità nome (lista parametri): tipo di ritorno**
 - Visibilità: + public, - private (esistono anche altri livelli di visibilità)
 - Parametro: **nome: tipo**
- Attributo: **visibilità nome: tipo [molteplicità] = default**



Traduzione

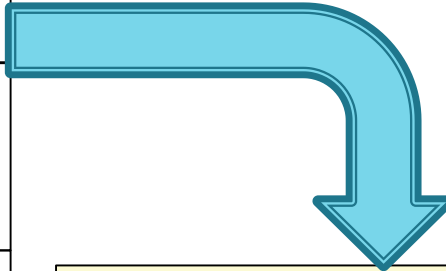
Persona
- nome: String - cognome: String - dataNascita: Date
+ siSposa(p: Persona): boolean + compieAnni(d: Date): boolean



```
public class Persona {  
    // methods  
    public boolean siSposa(Persona p) {  
        ...  
    }  
    public boolean compieAnni(Date d) {  
        ...  
    }  
    // instance variables  
    private String nome;  
    private String cognome;  
    private Date dataNascita;  
}
```

Esempio: la classe Name

Name
- fist: String - last: String - title: String
+ getInitials(): String + getFirstLast(): String + getTitleLastFirst(): String + setTitle(title: String): void



```
public class Name {  
    // constructors  
    // methods  
    public String getInitials() { ...}  
    public String getFirstLast() { ...}  
    public String getTitleLastFirst () { ...}  
    public void setTitle (String title) { ...}  
  
    // instance variables  
    private String first;  
    private String last;  
    private String title;  
}
```

Esempio:

Retribuzione dei dipendenti

➤ Presentazione del problema

Modellare un sistema per la gestione della retribuzione dei dipendenti che sono pagati con una tariffa oraria.

Per un dipendente, il sistema deve riuscire a calcolare la sua retribuzione sulla base della tariffa oraria e delle ore di lavoro effettuate e deve stampare il nome, le ore e la paga calcolata. Se il dipendente ha lavorato più di 40 ore, riceve una somma per gli straordinari, pagati una volta e mezzo la tariffa salariale normale. Se il dipendente ha 30 o più ore di straordinario nelle ultime due settimane viene emesso un messaggio d'avviso

Retribuzione

- Scenario d'esempio
 - Immaginiamo l'uso del sistema

Enter employee name: Gerald Weiss

Enter employee rate/hour: 20

Enter Gerald Weiss's hours for week 1: 30

Gerald Weiss earned \$600 for week 1

Enter Gerald Weiss's hours for week 2: 50

Gerald Weiss earned \$1100 for week 2

Enter Gerald Weiss's hours for week 3: 60

Gerald Weiss earned \$1400 for week 3

*** Gerald Weiss has worked 30 hours of overtime in the last two weeks.

Esempio:

Retribuzione dei dipendenti

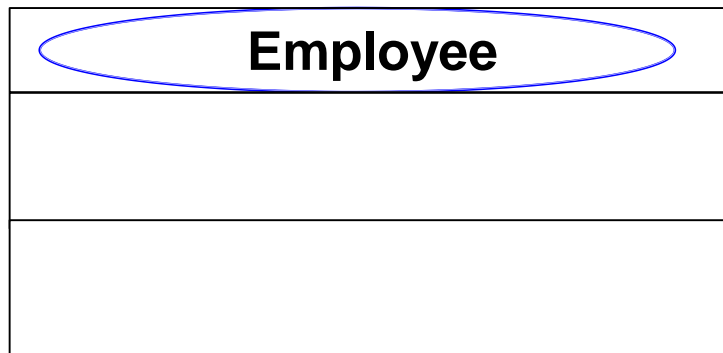
➤ Presentazione del problema

Modellare un sistema per la gestione della retribuzione dei dipendenti che sono pagati con una tariffa oraria.

Per un dipendente, il sistema deve riuscire a calcolare la sua retribuzione sulla base della tariffa oraria e delle ore di lavoro effettuate e deve stampare il nome, le ore e la paga calcolata. Se il dipendente ha lavorato più di 40 ore, riceve una somma per gli straordinari, pagati una volta e mezzo la tariffa salariale normale. Se il dipendente ha 30 o più ore di straordinario nelle ultime due settimane viene emesso un messaggio d'avviso

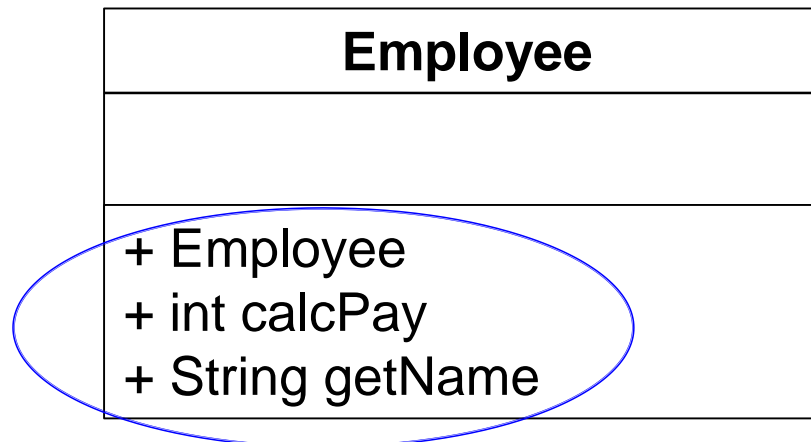
Retribuzione

- Oggetti primari
 - Termini chiave: dipendente, ore, nome, retribuzione oraria ...
 - DIPENDENTE, gli altri descrivono attributi del dipendente



Retribuzione

- Comportamento desiderato
 - Creare oggetti di tipo Employee (costruttore)
 - Employee
 - Calcolare la retribuzione
 - calcPay
 - Interrogare un oggetto Employee per conoscere il nome
 - getName



Retribuzione

➤ Comportamento desiderato

- Creare oggetti di tipo Employee (costruttore)

```
Employee officeEmpl = new Employee("Mario Rossi", 20);
```

- Calcolare la retribuzione

- Ha bisogno di conoscere il numero di ore

```
double salary = officeEmpl.calcPay(50)
```

- Interrogare un oggetto Employee per conoscerne il nome

```
System.out.print("The salary of " + officeEmpl.getName());  
System.out.println(" is " + salary);
```

Retribuzione

- **Costruttore**
 - È necessario specificare il nome e la paga oraria
 - Le ore lavorate cambiano di settimana in settimana – `new Employee(name, rate)`
- Il calcolo della paga richiede la conoscenza del numero di ore lavorate – `calcPay(hours)`
- Il ritrovamento del nome restituisce una stringa – `getName()`

Employee
+ Employee(name:String, rate: int) + calcPay(hours: int): int + getName(): String

Retribuzione

➤ Variabili d'istanza

- **getName** ha bisogno di accedere al nome dell'oggetto Employee – è necessario **conservare il nome** di un dipendente
- **calcPay** ha bisogno di accedere alla tariffa orario per calcolare la paga settimanale – è necessario conservare la **paga oraria** del dipendente
- È anche necessario memorizzare le **ore di straordinario** dell'ultima settimana

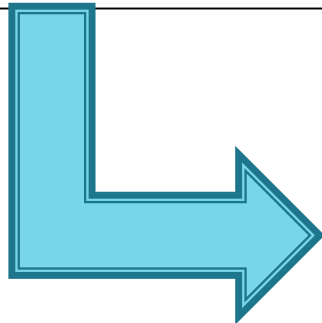
Employee
- name: String - rate:int - lastWeeksOvertime: int
+ Employee(name:String, rate: int) + calcPay(hours: int): int + getName(): String

Traduzione

Employee

- name: String
- rate: int
- lastWeeksOvertime: int

- + Employee(name: String, rate: int)
- + calcPay(hours: int): int
- + getName(): String



```
public class Employee{  
    // methods  
    public Employee(String name, int rate) {  
        ...  
    }  
    public int calcPay(int hours) {  
        ...  
    }  
    public String getName() {  
        ...  
    }  
    // instance variables  
    private String name;  
    private int rate;  
    private int lastWeeksOvertime;  
}
```


Retribuzione

- Costruttore

- Inizializza le variabili di stato con i valori degli argomenti

```
public Employee(String name, int rate) {  
    this.name = name;  
    this.rate = rate;  
    this.lastWeeksOvertime = 0;  
}
```

- getName

- Restituisce il nome del dipendente in una String

```
public String getName() {  
    return this.name;  
}
```

Retribuzione

➤ calcPay

- Prima si verifica se si è superato il limite delle 40 ore per verificare se c'è straordinario e poi calcola la paga
- Il metodo calcola inoltre il numero di ore di straordinario

Struttura di controllo if else

```
int pay, currentOvertime;
if (hours <= 40) {
    pay = hours * rate;
    currentOvertime = 0;
} else {
    pay = 40*rate + (hours-40)*(rate+rate/2);
    currentOvertime = hours - 40;
}
```

Retribuzione

- calcPay
 - Per la gestione del messaggio d'allarme si addiziona il numero di ore di straordinario della settimana corrente a quello della settimana passata e lo si confronta con 30

```
if (currentOvertime + lastWeeksOvertime >= 30)
    System.out.print(name + " has worked 30" +
        " or more hours overtime");
```

Struttura di
controllo if then

Retribuzione

- calcPay
 - L'ultimo atto è la memorizzazione del numero di ore di straordinario e la restituzione della paga

```
lastWeeksOvertime = currentOvertime;  
return pay;
```

Retribuzione

➤ calcPay

```
public int calcPay(int hours) {  
    int pay, currentOvertime;  
    if (hours <= 40) {  
        pay = hours * rate;  
        currentOvertime = 0;  
    } else {  
        pay = 40*rate+(hours-40)*(rate+rate/2);  
        currentOvertime = hours - 40;  
    }  
    if (currentOvertime + lastWeeksOvertime >= 30)  
        System.out.print(name + " has worked " +  
            " 30 or more hours overtime");  
    lastWeeksOvertime = currentOvertime;  
    return pay;  
}
```

Retribuzione

➤ Programma d'esempio

```
class Payroll {  
    public static void main(String a[]) {  
        Employee      e;  
        e = new Employee("Rudy Crew", 10);  
        int pay;  
        pay = e.calcPay(30);  
        System.out.print(e.getName());  
        System.out.print(" earned ");  
        System.out.println(pay);  
        pay = e.calcPay(60);  
    }  
}
```

L'istruzione if

- Le strutture di controllo selettive if ed if – else sono già note.
- Bisogna fare attenzione agli if innestati
 - Un'istruzione if annidata nel ramo true

```
if (hours < 60) {  
    if (hours >= 40)  
        System.out.println("Overtime");  
}  
else  
    System.out.println("Double-overtime");
```

Uso della struttura if

- Nella condizione dell'if possono essere confrontati il valore di variabili di tipo primitivo o di variabili di riferimento
- Sappiamo già confrontare valori di tipi primitivi,

```
int x, y;  
...  
if (x==y)  
...
```

- ma come si confrontano gli oggetti della classe String?

Confronto fra String

- Non è possibile usare `==`, che effettua il test sul valore della referenza
- Esiste un apposito metodo della classe String

```
if (input.equals("Y"))
```

```
if (input.equalsIgnoreCase("Y"))
```

- Questo vale per gli oggetti in generale
 - È opportuno che siano forniti nelle classi i metodi necessari per confrontare gli oggetti

Confronto fra oggetti

- Il confronto tra oggetti avviene attraverso un metodo **equals** che è implementato nella classe di appartenenza degli oggetti
- Es. nella classe Name:

```
class Name {  
    ...  
    public boolean equals(Name n) {  
        if (this.first.equals(n.first))  
            if (this.last.equals(n.last))  
                if (this.title.equals(n.title))  
                    return true;  
        return false;  
    }  
    ...  
    private String first, last, title;  
}
```

Confronto fra oggetti

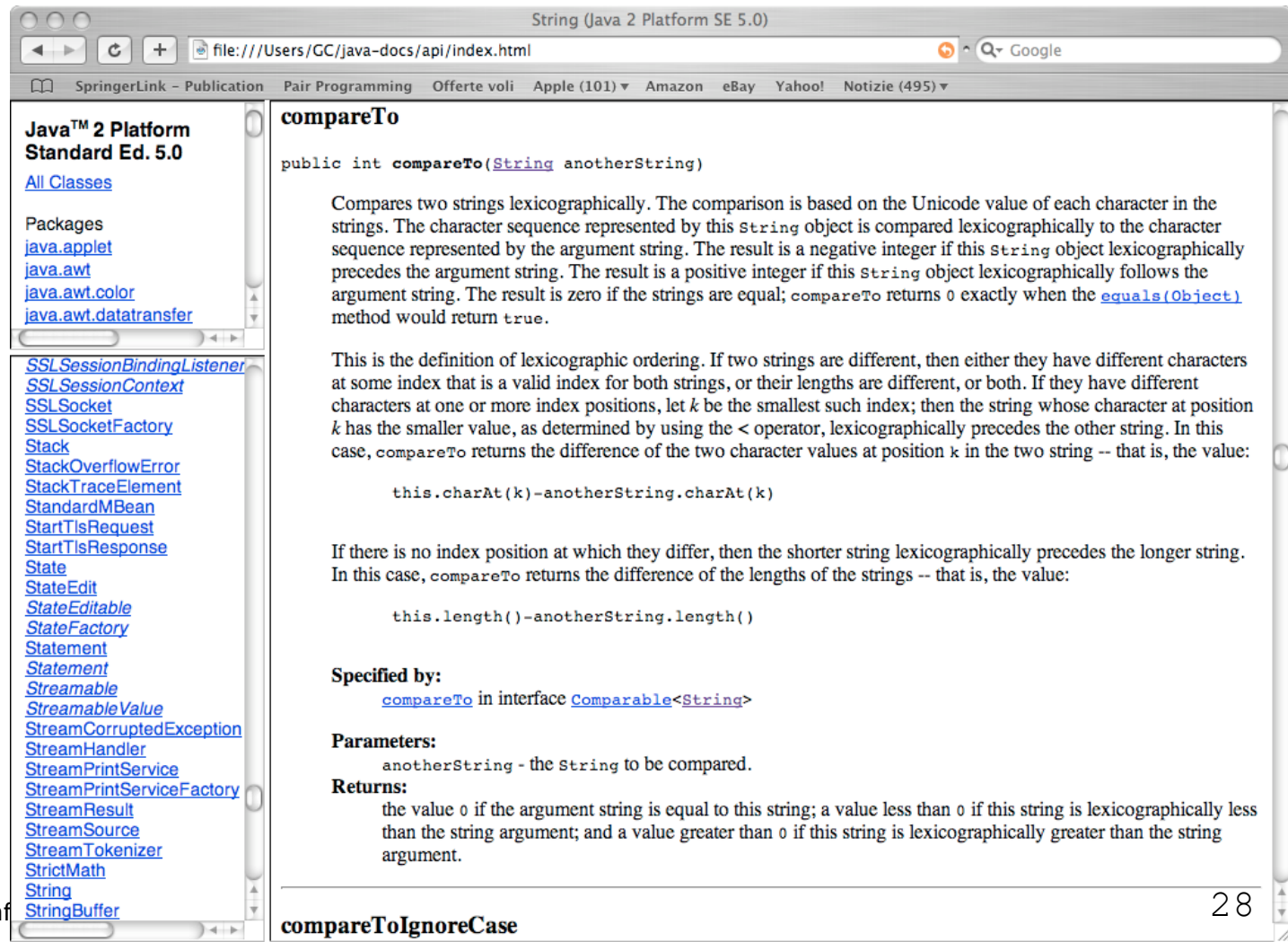
➤ Es. classe Employee:

```
class Employee{  
    ...  
    public boolean equals(Employee e){  
        if (this.name.equals(e.name))  
            if (this.rate == e.rate)  
                return true;  
        return false;  
    }  
    ...  
    private String name, rate, lastWeeksOvertime;  
}
```

- Non devono essere necessariamente considerate tutte le variabili d'istanza per stabilire se due oggetti sono uguali

Confronto fra String

- In alternativa si può utilizzare `compareTo`



The screenshot shows the Java 2 Platform Standard Ed. 5.0 API documentation for the `String` class. The left sidebar lists various classes and packages, including `String` and `StringBuffer`. The main content area displays the `compareTo` method, which compares two strings lexicographically. The method signature is `public int compareTo(String anotherString)`. The description explains that the comparison is based on the Unicode value of each character, and the result is a negative integer, zero, or a positive integer depending on whether the first string is lexicographically less than, equal to, or greater than the second string. The method returns the difference of the character values at the first index where they differ, or the difference of the lengths if no such index exists. The code snippet shows `this.charAt(k) - anotherString.charAt(k)` and `this.length() - anotherString.length()`. The documentation also specifies that the method is defined in the `Comparable<String>` interface and lists the parameters and return values.

String (Java 2 Platform SE 5.0)

file:///Users/GC/java-docs/api/index.html

SpringerLink - Publication Pair Programming Offerte voli Apple (101) Amazon eBay Yahoo! Notizie (495)

Java™ 2 Platform Standard Ed. 5.0

[All Classes](#)

Packages

- [java.applet](#)
- [java.awt](#)
- [java.awt.color](#)
- [java.awt.datatransfer](#)

[SSLSessionBindingListener](#)

[SSLSessionContext](#)

[SSLSocket](#)

[SSLSocketFactory](#)

[Stack](#)

[StackOverflowError](#)

[StackTraceElement](#)

[StandardMBean](#)

[StartTlsRequest](#)

[StartTlsResponse](#)

[State](#)

[StateEdit](#)

[StateEditable](#)

[StateFactory](#)

[Statement](#)

[Statement](#)

[Streamable](#)

[StreamableValue](#)

[StreamCorruptedException](#)

[StreamHandler](#)

[StreamPrintService](#)

[StreamPrintServiceFactory](#)

[StreamResult](#)

[StreamSource](#)

[StreamTokenizer](#)

[StrictMath](#)

[String](#)

[StringBuffer](#)

compareTo

```
public int compareTo(String anotherString)
```

Compares two strings lexicographically. The comparison is based on the Unicode value of each character in the strings. The character sequence represented by this `String` object is compared lexicographically to the character sequence represented by the argument string. The result is a negative integer if this `String` object lexicographically precedes the argument string. The result is a positive integer if this `String` object lexicographically follows the argument string. The result is zero if the strings are equal; `compareTo` returns 0 exactly when the `equals(Object)` method would return `true`.

This is the definition of lexicographic ordering. If two strings are different, then either they have different characters at some index that is a valid index for both strings, or their lengths are different, or both. If they have different characters at one or more index positions, let k be the smallest such index; then the string whose character at position k has the smaller value, as determined by using the `<` operator, lexicographically precedes the other string. In this case, `compareTo` returns the difference of the two character values at position k in the two string -- that is, the value:

```
this.charAt(k) - anotherString.charAt(k)
```

If there is no index position at which they differ, then the shorter string lexicographically precedes the longer string. In this case, `compareTo` returns the difference of the lengths of the strings -- that is, the value:

```
this.length() - anotherString.length()
```

Specified by:

`compareTo` in interface `Comparable<String>`

Parameters:

`anotherString` - the `String` to be compared.

Returns:

the value 0 if the argument string is equal to this string; a value less than 0 if this string is lexicographically less than the string argument; and a value greater than 0 if this string is lexicographically greater than the string argument.

compareToIgnoreCase

Confronto fra oggetti

- Per confrontare due oggetti potrebbe essere necessario che nella classe di appartenenza degli oggetti sia implementato anche il metodo `compareTo`
- Es. nella classe `Name`:

```
class Name {  
    ...  
    public int compareTo(Name n) {  
        if (!this.last.equals(n.last))  
            return this.last.compareTo(n.last);  
        return this.first.compareTo(n.first);  
    }  
    ...  
    private String first, last, title;  
}
```

Confronto fra oggetti

- Es. classe Employee:

```
class Employee{
    ...
    public int compareTo(Employee e){
        if (!this.name.equals(e.name))
            return (this.name.compareTo(e.name));
        return this.rate-e.rate;
    }
    ...
    private String name, rate, lastWeeksOvertime;
}
```

- Non tutte le variabili d'istanza devono essere considerate per confrontare due oggetti

Il riferimento null

- Molti metodi restituiscono un riferimento ad un oggetto
- Occasionalmente vorremmo restituire un'indicazione che specifichi che non è possibile restituire alcun oggetto
 - Per esempio, se in un metodo si deve costruire un oggetto, ma non sono disponibili i dati necessari, questa operazione fallisce perché non si può costruire alcun oggetto
- In tale situazione si può usare il valore *null*
 - *null* è un riferimento che non fa riferimento ad alcun oggetto
 - *null* può essere assegnato a variabili di riferimento di ogni classe
 - È anche possibile verificare che una variabile contenga *null*

```
String s;  
s = null;  
if (s == null) ...
```

Un esempio:

La raccolta dei pedaggi

Descrizione del problema

- Modellare un sistema di raccolta dei pedaggi dei camion per le autostrade
- I camion pagano \$5 per asse più \$10 per ogni mezza tonnellata di peso totale
- Un display nel casello visualizza i pagamenti ed il numero di camion che hanno pagato dall'ultima raccolta

Un esempio: La raccolta dei pedaggi

Scenario d'esempio

- Nel casello ci sono lo schermo di un calcolatore ed un lettore di codici a barre
- Per leggere il numero di assi il casellante legge un codice a barre sul parabrezza del camion
- Il peso è letto da un codice a barre su un biglietto consegnato dal conducente del camion
- Le informazioni sul camion e sull'ammontare del pedaggio sono visualizzate sullo schermo del casello
- Il casellante può richiedere la visualizzazione sullo schermo dei dati registrati dall'ultimo prelievo della cassa
- All'atto del prelievo della cassa compare sullo schermo un opportuno messaggio con l'indicazione dei dati di riepilogo

Un esempio: La raccolta dei pedaggi

- Scenario d'esempio - Messaggi
 - Truck arrival – axles: 5 total weight: 12500
Toll due: \$145
 - Totals since last collection – Receipts: \$205
Trucks: 2
 - *** Collecting receipts ***
Totals since last collection – Receipts: \$523
Trucks: 5

Un esempio:

La raccolta dei pedaggi

- Individuazione degli oggetti primari
- Un buon punto di partenza: le parole chiave contenute nella presentazione del problema
 - Camion, casello, peso, ricevuta ...
- Nel nostro caso: camion (Truck) e casello (TollBooth)

Truck

TollBooth

Un esempio:

La raccolta dei pedaggi

- Comportamento desiderato – La classe Truck
- Un costruttore per costruire oggetti della classe Truck
 - Truck
- Metodi per ottenere le informazioni necessarie al calcolo del pedaggio, numero assi e peso
 - `getAxles`
 - `getTotalWeight`

Truck
+ Truck + getAxles + getTotalWeight

Un esempio:

La raccolta dei pedaggi

- Interfaccia – La classe Truck
- La creazione di un camion richiede l'indicazione del numero di assi e del peso
 - `Truck(int axles, int weight)`
- La richiesta di numero di assi e peso avviene senza parametri
 - `getAxles()`
 - `getTotalWeight()`

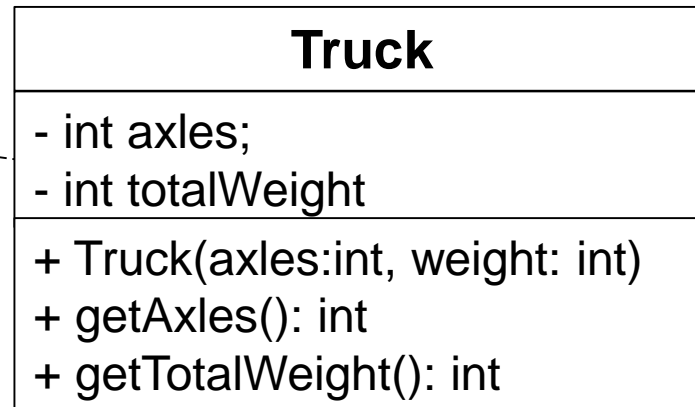
Truck
+ Truck(axles:int, weight: int) + getAxles(): int + getTotalWeight(): int

Un esempio:

La raccolta dei pedaggi

- Variabili di istanza – La classe camion
- Ogni oggetto di tipo camion deve memorizzare il proprio numero di assi ed il proprio peso

Tutti i truck
devono avere
numero di assi e
peso positivi



Un esempio:

La raccolta dei pedaggi

- Passiamo alla classe TollBooth
- Comportamento desiderato – La classe Tollbooth
 - Creazione di un casello
TollBooth
 - Calcolo del pedaggio
calculateToll
 - Visualizzazione dei dati dall'ultima raccolta
displayData
 - Prelievo dalla cassa
onReceiptCollection

TollBooth
+ TollBooth + calculateToll + displayData + onReceiptCollection

Un esempio:

La raccolta dei pedaggi

- Interfaccia – La classe Tollbooth
- Creazione di un casello
`TollBooth booth = new TollBooth();`
- Calcolo del pedaggio; richiede la conoscenza delle caratteristiche del camion
`double toll = calculateToll(truck);`
- Visualizzazione dei totali
`displayData();`
- Prelievo della cassa
`onReceiptCollection();`

TollBooth
+ TollBooth() + calculateToll(truck:Truck): double + displayData() + onReceiptCollection()

Un esempio:

La raccolta dei pedaggi

- Variabili di istanza – La classe TollBooth
- Un oggetto di tipo casello deve memorizzare le somme ricevute ed il numero di camion passati dall'ultimo prelievo di cassa

TollBooth
- receiptsSinceCollection: int - trucksSinceCollection: int
+ TollBooth() + calculateToll(truck:Truck): int + displayData() + onReceiptCollection()

Un esempio: La raccolta dei pedaggi

- Il diagramma delle classi completo

Truck
- int axles; - int totalWeight
+ Truck(axles:int, weight: int) + getAxles(): int + getTotalWeight(): int

TollBooth
- receiptsSinceCollection: int - trucksSinceCollection: int
+ TollBooth() + calculateToll(truck:Truck): int + displayData() + onReceiptCollection()

Un esempio: La raccolta dei pedaggi

Implementazione delle classe Truck

- La classe Truck: traduzione nello scheletro della classe Truck

Truck

Truck
- int axles; - int totalWeight
+ Truck(axles:int, weight: int) + getAxles(): int + getTotalWeight(): int

```
class Truck {  
    // Methods  
    public Truck(int axles, int weight) { ... }  
    public int getAxles() { ... }  
    public int getTotalWeight() { ... }  
  
    // Instance Variables  
    private int axles;  
    private int totalWeight;  
}
```

Un esempio:

La raccolta dei pedaggi

- Metodo costruttore – La classe Truck
- Deve semplicemente inizializzare le variabili di istanza

```
public Truck(int axles, int totalWeight) {  
    this.axles = axles;  
    this.totalWeight = totalWeight;  
}
```

Un esempio:

La raccolta dei pedaggi

- Altri metodi – La classe Truck
- Devono semplicemente ritornare il valore delle variabili di istanza

```
public int getAxles() {return axles;}
```

```
public int getTotalWeight() {return totalWeight;}
```

Un esempio:

La raccolta dei pedaggi

La classe Truck

```
class Truck {  
    // Methods  
    public Truck(int axles, int weight) {  
        this.axles = axles;  
        this.totalWeight = totalWeight;  
    }  
    public int getAxles() {  
        return axles;  
    }  
    public int getTotalWeight() {  
        return totalWeight;  
    }  
  
    // Instance Variables  
    private int axles;  
    private int totalWeight;  
}
```

Aggiungere i metodi affinché la classe sia di input e di output

Un esempio:

La raccolta dei pedaggi

- La classe TollBooth: traduzione nello scheletro della classe TollBooth

TollBooth
- receiptsSinceCollection: int - trucksSinceCollection: int
+ TollBooth() + calculateToll(truck:Truck): void + displayData() + onReceiptCollection()

```
class TollBooth {  
    // Methods  
    public TollBooth() {...}  
    public void calculateToll(Truck truck) {...}  
    public void onReceiptCollection() {...}  
    public void displayData() {...}  
    // Instance Variables  
    private double receiptsSinceCollection;  
    private int trucksSinceCollection;  
}
```

Un esempio:

La raccolta dei pedaggi

- Metodo costruttore – La classe TollBooth
- Deve inizializzare a zero le variabili di istanza

```
public TollBooth() {  
    trucksSinceCollection = 0; // Clear out totals  
    receiptsSinceCollection = 0;  
}
```


Un esempio:

La raccolta dei pedaggi

- Metodo DisplayData – La classe TollBooth
- Visualizza i totali sullo schermo (System.out)

```
public void displayData() {  
    System.out.print("Totals since last collection - " +  
        "Receipts: " + receiptsSinceCollection +  
        " Trucks: " + trucksSinceCollection);  
}
```

Un esempio:

La raccolta dei pedaggi

- Metodo onReceiptCollection – La classe TollBooth
- Visualizza i totali all'atto del prelievo della cassa e reinizializza le variabili di istanza
- Usa il metodo displayData

```
public void onReceiptCollection() {  
    System.out.println("*** Collecting receipts ***");  
    displayData();  
    trucksSinceCollection = 0; // Clear out totals  
    receiptsSinceCollection = 0;  
}
```

Un esempio:

La raccolta dei pedaggi

- Metodo calculateToll – La classe TollBooth
- Calcola e visualizza il dovuto

```
public void calculateToll(Truck truck) {  
    int axles = truck.getAxles();  
    int totalWeight = truck.getTotalWeight();  
    int tollDue = 5*axles+10*(totalWeight/1000)*2;  
    System.out.print("Truck arrival - axles: ");  
    System.out.print(axles);  
    System.out.print(" total weight: ");  
    System.out.print(totalWeight);  
    System.out.print(" Toll due: ");  
    System.out.println(tollDue);  
    trucksSinceCollection = trucksSinceCollection + 1;  
    receiptsSinceCollection =  
        receiptsSinceCollection + tollDue;  
}
```

La raccolta dei pedaggi

Un programma di prova:

```
class TestTollBooth {  
    public static void main(String [] args) {  
        // Create the tollbooth  
        TollBooth booth = new TollBooth();  
        // Now for some trucks  
        Truck truck1 = new Truck(5, 12500);  
        // Let's start collecting tolls!  
        booth.calculateToll(truck1);  
        booth.displayData();  
        Truck truck2 = new Truck(2, 5000);  
        booth.calculateToll(truck2);  
        // Time to collect the receipts  
        booth.onReceiptCollection();  
        // Here comes another truck  
        Truck truck3 = new Truck(6, 17000);  
        booth.calculateToll(truck3);  
        booth.displayData();  
    }  
}
```

Esercizio

- Gestire regole diverse di calcolo per diversi caselli

```
TollBooth booth1 = new TollBooth(5,7);  
TollBooth booth2 = new TollBooth(8,12);  
Truck tr = new Truck(5,12780);  
booth1.calculateToll(tr);  
booth2.calculateToll(tr);
```

Esercizio

- Implementare un registratore di cassa
 - Memorizza il numero di acquisti
 - Memorizza il pagamento
 - Calcola il resto

Costanti con nome

```
static final int  
    DuePerAxle = 5,  
    DuePerHalfTon = 10,  
    TonInPounds = 2000;
```

ES: modificare
il costo per
asse da \$5 a\$7

Costanti static final

- È possibile raggruppare costanti da utilizzare in più metodi di classi diverse in una classe apposita
 - Vanno dichiarati public, static e final

```
public class Math
{
    . . .
    public static final double E = 2.7182818284590452354;
    public static final double PI = 3.14159265358979323846;
}

double circumference = Math.PI * diameter;
```


Nota sintattica

In un metodo:

```
final typeName variableName = expression ;
```

In una classe:

```
accessSpecifier static final typeName variableName = expression;
```

Esempio:

```
final double NICKEL_VALUE = 0.05;  
public static final double LITERS_PER_GALLON = 3.785;
```

Definisce costanti con nome

Metodi statici

- Non operano su una istanza
 - non sono associati ad un oggetto ma alla classe stessa
- Possono essere invocati senza alcun riferimento ad oggetto
- Sono definiti come tutti gli altri metodi con l'aggiunta del prefisso static
- Sono invocati utilizzando il nome della classe
 - Il riferimento this NON ha senso
 - NON possono accedere variabili di istanza
 - NON possono accedere metodi non statici
 - POSSONO invocare il costruttore

Nota sintattica

```
ClassName.methodName(parameters)
```

Esempio:

```
Math.sqrt(4)
```

Invocazione di un metodo statico

... torneremo sui metodi statici

Ancora sulla classe String

- Modella sequenze di caratteri . . .
- Concatenazione "+"

```
String name = "Dave";  
String message = "Hello, " + name;  
    // message is "Hello, Dave"
```

- Basta che uno solo degli argomenti sia una stringa

```
String a = "Agent";  
int n = 7;  
String bond = a + n; // bond is Agent7
```

Conversioni

➤ Da String a numero

```
int n = Integer.parseInt(str);  
double x = Double.parseDouble(str);
```

➤ Da numero a String

```
String str = "" + n;  
str = Integer.toString(n);
```