Programmazione a oggetti - Java

Esercitazione 6

Contatti:
Prof. Angelo Gargantini – angelo.gargantini@unibg.it
Dott.ssa Silvia Bonfanti – silvia.bonfanti@unibg.it

Eccezioni

Per ora (18/3/19) usa solo come eccezioni da creare o:

- RuntimeException(msg)
- -Exception

Esercizio 31

- Scrivi un programma che inserito un numero intero da tastiera stampi a video il risultato della divisione di tale numero per un altro numero generato casualmente, compreso fra 0 e 3, intercettando l'eventuale eccezione della divisione per 0.

Esercizio 32

- Scrivi un metodo che prenda in input un'unica stringa contenente un elenco di voti separati da punto e virgola, li memorizzi in una lista (Vector o ArrayList) di interi e ne calcoli media e massimo,
- Intercetta l'eventuale eccezione per i voti non validi perché sono stringhe e ignorale
  - esempio "a;18;20" → ignora a
- Lancia un'eccezione per voti non validi
  - negativi, minori di 18, o maggiori di 30
  - Usa eccezione non controllata RuntimeException

Esercizio 33

- A partire dall'Esercizio 28 (ordinazioni ai tavoli di un ristorante), definisci, lancia e cattura opportunamente delle eccezioni nei seguenti casi:
- Replicazioni di ID (nel caso di Tavolo e Piatto)
- Inserimento di una persona con lo stesso nome e cognome di una esistente
- Aggiunta di una persona a un tavolo già pieno
- Aggiunta e rimozione di un piatto il cui ID è inesistente.

- Scrivere una classe di prova in cui si crea almeno un tavolo, e si testano tutti i metodi che lanciano eccezioni.
- Come eccezioni, lancia delle RuntimeException

Esercizio 34

- A partire dall'esercizio 29 (gestione voli), aggiungere almeno tre eccezioni di tipo BoardingException:
- ExceedingPassengersCapacityException: quando si tenta di aggiungere più passeggeri rispetto ai posti disponibili nella classe richiesta (nella stampa dell'eccezione, inserire anche la classe richiesta)
- PassengerNotFoundException: quando si vuole rimuovere un passeggero dal volo, tramite in IDBiglietto inesistente fra i passeggeri registrati su quel volo.
- NoPassengersException: si vuol fare decollare un volo senza alcun passeggero registrato.

Esercizio 35

- Aggiungere eccezioni gestore di negozi dell'esercizio 30, ogniqualvolta:
- Viene inserito un ID uguale (SameIDException)
  - A livello di negozio (SameStoreIDException)
  - A livello di prodotto in un negozio (SameItemIDException)
- Un cliente acquista più della disponibilità (ExceedingAvailabilityException)
- Ricorda che un negozio può anche rifornire un prodotto (eventualmente lanciare eccezione NegativeRefillException se viene passata una quantità negativa) e rimuoverlo dalla lista dei suoi prodotti.
- Scrivi una classe di prova

UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Ereditarietà, Overriding, Overloading

Ripasso

OVERRIDING: si scrive in una sottoclasse un metodo della superclasse con la stessa segnatura

OVERLOADING: è possibile definire metodi con lo stesso nome ma con segnature differenti

Cos'è la SEGNATURA? È la «firma» del metodo costituita dal nome del metodo, dal numero dei suoi parametri e dal loro tipo

Lo stesso nome di un metodo può essere usato per operazioni diverse, con definizioni diverse che richiedono un diverso tipo o numero di parametri.

Esercizio 36

- Definire l'architettura a oggetti del dominio "Figure geometriche", definendo classe **Figura** (astratta), e le sottoclassi: **Rettangolo**, **Quadrato** (sottoclasse di Rettangolo), **Triangolo** e **Cerchio**.
  - è simile alle figure del libro ma non usare quelle, definisci delle tue figure
  - ogni figura ha due metodi per calcolare l'area e il perimetro
- Scrivi un main in cui Inserisci delle figure in una lista (ArrayList, Sequenza o Vector) (non fare input con l'utente, costruisci gli oggetti programmaticamente), e le ordini per perimetro (usa Comparable e Collections.sort o SequenzaOrdinata).
  - Stampare tale lista.
  - Stampare solo i triangoli sempre in ordine di perimetro
- [opzionale] Scrivi un metodo che stampa solo i rettangoli o quadrati con area > 20 in una lista passata come argomento, in ordine decrescente di area.
  - usa sort e un comparator diverso dal Comparable

Esercizio 37 (overriding)

Definire una classe **Dipendente** che ha i seguenti campi:

- String: Nome, Cognome
- int: oreLavorativeMensili, retribuzioneOraria

Scrivere un metodo che calcola lo stipendio del dipendente nel seguente modo: Stipendio=oreLavorativeMensili * retribuzioneOraria

Si definisca anche un particolare tipo (sottoclasse) di dipendente responsabile di progetto (**DipendenteResponsabile**), al cui stipendio del dipendente (vedi sopra) viene aggiunto un bonus:

Stipendio=oreLavorativeMensili * retribuzioneOraria + bonus

DipendenteResposabile fa OVERRIDING del metodo per il calcolo dello stipendio. Scrivere un main() che calcoli lo stipendio per una persona Dipendente e per una persona DipendenteResponsabile.

Esercizio 38 (1)

- Data la classe Animale:

```
class Animale {
  private String nome;
  public Animale(String s){
    nome = s;
  }
  public String comeTiChiami() {
     return nome;
  }
  public void parla(){}
  public void incontra(Animale  a) {
    System.out.println(nome + ":<Ciao, " + a.nome + ">");
    parla();
  }
}
```

Si costruiscano due classi Topo e Gatto che estendono la classe Animale. E (vedi slide successiva).... →

Esercizio 38 (2)

- In particolare le due classi (topo e gatto) devono ridefinire il metodo parla():
  - Il topo deve ridefinire il metodo in modo che esso stampi sullo schermo «Squit»
  - Il gatto deve ridefinire il metodo in modo che esso stampi sullo schermo «Miao»
- Si aggiunga ora nella classe **Topo** una nuova versione del metodo *incontra()* che prende come argomento un animale e se è un oggetto di tipo Gatto stampa:

System.out.println(nome + ":<Aiutoooooo!!!>");

altrimenti si comporta come la superclasse animale (usa super.)

- Analogamente per il **Gatto** si aggiunga una nuova versione del metodo *incontra()* che prende come argomento un animale e se è oggetto di tipo Topo stampa:

System.out.println(nome + ":<Ti prendo!!!>");

altrimenti si comporta come la superclasse animale (usa super.)

- Scrivere un main() per verificarne il funzionamento. Fai incontrare topi e gatti e anche un animale di nome "mucca"

Esercizio 39 (overloading)

- Definire una classe **Studente** che ha i seguenti campi:
  - nome String (Ereditato dalla classe Persona)
  - cognome String (Ereditato dalla classe Persona)
  - matricola Integer
  - ArrayList<Esame> Esame (nome, voto)
  - mediaVoti Double

- Scrivere un metodo che calcola la media dei voti degli studenti (memorizzarlo nel campo «mediaVoti»)

- Scrivere due metodi **getTasse** che, data una tassa massima, calcola la tassa finale applicando uno sconto nel seguente modo (vedi slide successiva →)

Esercizio 39 (2)

Se lo studente indica la fascia di appartenenza viene applicato uno sconto in base alla fascia inserita. Successivamente viene applicato anche uno sconto in base alla media dei voti (sulla tassa al netto dello sconto applicato precedentemente)

Se lo studente non indica la fascia di appartenenza viene applicato uno sconto in base alla media dei voti

Scrivere una classe di prova

- NB: Overloading dei metodi

Fascia	Sconto
1	50%
2	35%
3	25%
4	10%
5	0%

Voto	Sconto
30	10%
29	8%
28	6%
27	4%
altro	0%

Esercizio 40

Scrivere la classe **Motorino** che ha i seguenti attributi:

colore: una stringa indicante il colore del motorino, **velocità**: un numero con la virgola indicante la velocità in Km/h che possiede il motorino, **tipo**: una stringa indicante la marca e il modello del motorino es. «Piaggio scarabeo», l'attributo **antifurto** un boolean che indica se è stato inserito l'antifurto (ha un valore iniziale pari a false). Il costruttore ha come parametri il colore, il tipo, la velocità. Scrivere il metodo getVelocità che restituisce la velocità del motorino, scrivere inoltre il metodo accelera che ha come parametro un numero con la virgola indicante i Km/h che si vogliono aggiungere alla velocità, il metodo verifica il valore dell'attributo antifurto se è false aggiunge il valore del parametro all'attributo velocità, altrimenti non fa nulla. Scrivere il metodo inserisciAntifurto che assegna un valore true all'attributo antifurto.

•Scrivere la classe **MotorinoImmatricolato** sottoclasse della classe **Motorino** che ha in più 2 attributi, **maxVelocità**: un numero con la virgola (deve essere maggiore di velocità altrimenti sollevare un'eccezione creata da te e assegna a maxVelocità il valore di velocità) indicante la velocità massima in Km/h che il motorino può raggiungere, **targa**: una stringa indicante la targa del motorino. Definire un costruttore che tenga in considerazione anche maxVelocità e targa. Aggiungere il metodo getMax, il metodo stampa il valore dell'attributo maxVelocità. Ridefinire il metodo accelera in modo che prima di modificare la velocità effettui un controllo sulla velocità massima raggiunta. Il metodo definisce una variabile s (dello stesso tipo di velocità) ed assegna ad s la somma tra il valore del parametro del metodo ed il valore dell'attributo velcotià; se s è minore del valore dell'attributo maxVelocità assegna il valore di s all'attributo velocità altrimenti assegna all'attributo velocità il valore dell'attributo maxVelocità

•Scrivere una classe di prova in cui vengono istanziati un oggetto di tipo Motorino e un oggetto MotorinoImmatricolato. Testare i metodi definiti nella classi. Provare il caso in cui viene sollevata un'eccezione.