Classi
Attributi
Associazioni
Responsabilita'

Esame di Object Orientation (Gr. 2) – 16 giugno 2023

- *Scrivere immediatamente su ogni foglio che vi è stato consegnato Cognome, Nome, N° Matricola.*
- *Non è consentito consultare appunti, libri, o colleghi, né qualunque dispositivo elettronico, PENA IMMEDIATO ANNULLAMENTO DELLA PROVA*
- *Tempo a disposizione: 3 ore*

Esercizio 1

Si vuole realizzare un software per la gestione di polisportive, ogni struttura polisportiva è caratterizzata da un nome, un indirizzo, un numero di telefono ed un proprietario, caratterizzato dalle proprie generalità (nome, cognome, data di nascita, codice fiscale, email). Le strutture possono prevedere campi per diversi sport: calcio, tennis, basket e pallavolo. A loro volta i campi possono distinguersi in base al numero di giocatori per squadra ad esempio: i campi da calcio possono essere fatti per squadre da 5 ,8 o 11. Il proprietario può inserire nel sistema nuovi campi sportivi, identificati da nome, sport, tipologia, e prezzo. Gli utenti del sistema per registrarsi devono specificare nome utente, password, indirizzo e-mail e carta di credito. Gli utenti registrati possono richiedere una lista dei campi disponibili lo sport da praticare, la data e l'orario per la prenotazione (valida per una singola ora di gioco). Se non ci sono campi disponibili il sistema chiede all'utente di specificare data e orario differenti; altrimenti, l'utente seleziona un campo disponibile e la spesa viene detratta dalla carta di credito dell'utente.

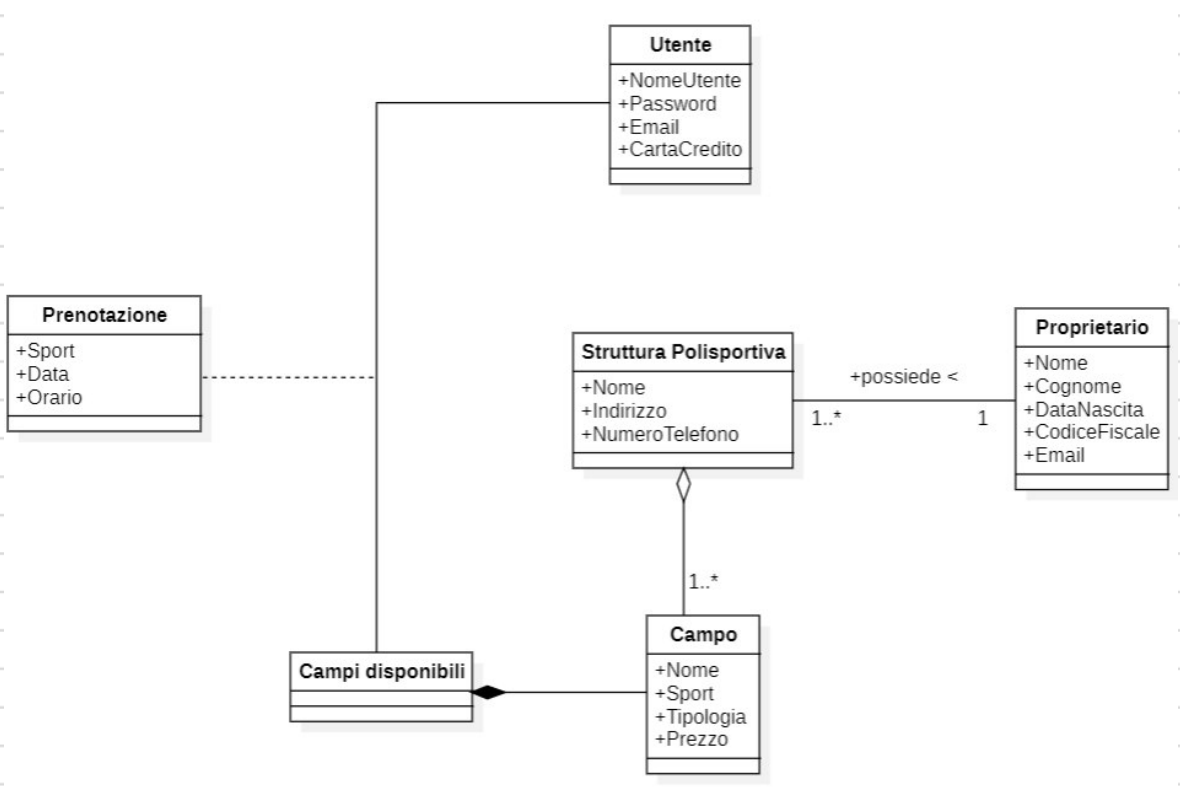
Si richiede di definire (a) Class Diagram e (b) Sequence Diagram per modellare la funzionalità di **richiesta di una nuova prenotazione (comprende la visione dei campi disponibili)**. Il Class Diagram e il Sequence Diagram, intesi come modello di dominio, devono rifarsi all'euristica Entity-Boundary-Control (EBC).

Esercizio 2

Rappresentare con un Sequence Diagram un'invocazione del metodo *getGoldCoins* della classe Gold riportato sotto.

```
public Coin[] getGoldCoins(Configuration conf){
    GoldVendingMachine g = new GoldVendingMachine();
    float gold_actual_price = g.get_gold_price(conf.gold_coins_type);
    float total_amount = gold_actual_price * conf.get_num_coins_required();
    int transaction_successful = 0;
    if(conf.get_payment_mode() == "card"){
        if(Bank.authorize_card(conf.get_iban())){
            Bank.charge_money(conf.get_iban(),total_amount);
            transaction_successful = 1;
        }else{
            throw new RuntimeException ("card not accepted!");
        }
    }else if(conf.get_payment_mode() == "cash") {
        g.open_money_desk_and_wait();
        if(g.money_inserted()){
            transaction_successful = 1;
        }
        throw  new RuntimeException ("money not inserted!");
    }
    if(transaction_successful == 1){
        return new Coin[conf.get_num_coins_required()];
    }
    return null;
}
```

ESERCIZIO 1


Utente
+NomeUtente
+Password
+Email
+CartaCredito
Prenotazione
+Sport
+Data
+Orario
Struttura Polisportiva
+Nome
+Indirizzo
+NumeroTelefono
+possiede <
1..*
1
Proprietario
+Nome
+Cognome
+DataNascita
+CodiceFiscale
+Email
1..*
Campo
+Nome
+Sport
+Tipologia
+Prezzo
Campi disponibili

Esercizio 3

```
public static void main(String[] args){
    try {
        Animal animal = new Dog();
        animal.bark();
        for (int i = 0; i <= 5; i++) {
            animal.eat();
            animal.increase_dog_weight();
        }
    }
    catch (Exception exception ){
        System.out.println("OH NO your animal is died, "+exception.getMessage());
    }
    catch (RuntimeException runtimeException){
        System.out.println(runtimeException.getMessage());
    }
}

public class Dog extends Animal{
    public int weight = 15;
    public void bark(){
        System.out.println("your dog is barking");
    }
    public void increase_dog_weight() {
        weight++;
        if(this.weight >=20){
            throw new RuntimeException("your dog is too fat!");
        }
    }
    public void eat(){
        System.out.println("your dog is eating");
    }
}

public class Animal {
    public void eat(){
        System.out.println("your animal is eating");
    }
}
```

Si considerino le classi java riportate sopra (si assuma che ogni classe sia definita in un file apposito).

a) Il codice compila correttamente? Motivare la risposta e fornire possibili correzioni in caso di risposta negativa.
b) Al netto delle eventuali correzioni apportate al punto precedente, qual è l'output di una esecuzione del metodo *main* della classe *Main*?
c) Nel codice è presente un esempio di *overriding*? Se sì indicare dove. Se no, indicare come è possibile modificare il codice per introdurne un esempio.
d) Nel codice è presente un esempio di *overloading*? Se sì indicare dove. Se no, indicare come è possibile modificare il codice per introdurne un esempio.

Esercizio 4

Qual' è la differenza tra *extends* e *implements*? riportare un esempio di ciascuno nel dominio della polisportiva presentata nell'esercizio 1 (eventualmente introducendo nuove classi ulteriori, rispetto al problema proposto).

ESERCIZIO 3

a) No, il programma non compila in quanto i metodi barK() e increase_dog_weight(), sono definiti solo in Dog. Per risolvere il problema dovrebbero essere implementate anche nella classe Animal, oppure si usa un oggetto di tipo Dog. In più manca la classe Main.

b) L'output sarà:
your dog is barKing
your dog is eating
your dog is eating
your dog is eating
your dog is eating
your dog is eating
you dog is too fat!

c) Sì, nel codice è presente un overriding per la presenza del metodo barK() sia in Animal sia in Dog (che estende la precedente).

d) No, non è presente un overloading. Potremmo averlo aggiungendo, ad esempio, alla classe Dog un altro metodo increase_dog_weight() con diverso tipo di ritorno o tipo o numero di parametri.

4) Exstends: viene utilizzata per creare una relazione di ereditarietà tra due classi. Quando una classe ne estende un'altra, eredita da questa tutti gli attributi e metodi, a patto che non siano private.

Implements: viene usato per far implementare una o più interfacce da una classe. Un'interfaccia è simile a una classe astratta in cui tutti i metodi sono astratti (non hanno implementazione). La classe che implementa l'interfaccia ha il compito di implementare i metodi definiti nell'interfaccia.