



Esame di Java

Davide Marietti, Università di Torino

Andrea Balbo Mossetti, Università di Torino

Master IT Full Stack Design and Development, a.a. 2022/2023

Indice

1	Descrizione del lavoro	3
2	Classi implementate	3
2.1	Classe <code>Animal</code>	3
2.2	Classe <code>AnimalShelter</code>	4
2.3	Classe <code>User</code>	6
A	Codice Sorgente	8

Listings

1	<code>Animal.java</code> – Allestimento della classe <code>Animale</code>	3
2	<code>Aniaml.java</code> – Estensione della classe <code>Animal</code> per definire nuove specie.	3
3	<code>AnimalShelter.java</code>	4
4	<code>AnimalShelter.java</code>	6
5	<code>Animal.java</code>	8
6	<code>User.java</code>	10
7	<code>AnimalShelter.java</code>	11
8	<code>DemoAnimalShelter.java</code>	12

1 Descrizione del lavoro

Per la realizzazione della consegna siamo partiti dalla traccia fornita dalla Professoressa, con cui venivamo invitati ad implementare uno *Shelter* per animali. Questo progetto riguarda il corso di Java base. Pertanto, ci siamo concentrati sullo sviluppo dei concetti base della programmazione ad oggetti vista a lezione, piuttosto che sull'implementazione di funzionalità complesse che utilizzino librerie esterne. Tutte le funzionalità implementate utilizzano esclusivamente oggetti della libreria Standard di Java [1]. Abbiamo cercato di includere nel codice i concetti fondamentali della programmazione ad oggetti in Java, il che ci ha permesso di ragionare sui concetti di **binding dinamico**, **classi astratte**, **sottoclassi**, **upcasting**, **sovratipo**, **array di oggetti**, **overloading ereditarietà**, **overwriting**, **information hiding**, **security leaking** [2][3].

2 Classi implementate

Il codice seppur minimale, nasconde al suo interno spunti interessanti.

2.1 Classe Animal

Il primo passo è quindi consistito nell'allestimento di una classe **Animale**, che astrae il concetto di animale. La classe è definita **astratta** per due principali motivi:

- Non posso definire un oggetto con tipo esatto **Animal**, quindi ciascun animale deve essere definito attraverso una delle sottoclassi **concrete** di **Animal** [2].
- Avere una sovraclassa comune (e obbligata) per ciascuna specie di animale ci consente di usare l'**upcasting** per poter definire una array di oggetti di tipo **Animal** che ci permette di astrarre il concetto di shelter [3].
- La definizione del metodo **equals** ci ha permesso di discutere il concetto di *uguaglianza* tra oggetti.

Listing 1: Animal.java – Allestimento della classe Animale

```
1 abstract class Animal {
2     private String name, breed;
3     private int age, conditions; // condizioni di salute dell'animale adottato
4
5     // costruttore
6     public Animal(String breed, int age, int conditions) {
7         this.breed = breed;
8         this.age = age;
9         this.conditions = conditions;
10    }
11
12
13    // some methods here
14 }
```

Abbiamo definito animali appartenenti a specie specifiche, sfruttando la possibilità di estendere la suddetta classe. In questo modo, le specie di animali che stiamo andando ad implementare ereditano gli attributi che abbiamo definito poco sopra e ne specificano di nuovi per aumentare il dettaglio di astrazione dell'animale.

Notare che per ciascun animale è indicato anche il suo stato di salute al momento dell'ingresso nello shelter. Lo scopo dello shelter è anche quello di curare gli animali, oltre che ospitarli. Alleghiamo codice esemplificativo:

Listing 2: Aniaml.java – Estensione della classe **Animal** per definire nuove specie.

```
1 class Dog extends Animal {
```

```

2
3 // Subclasses add more feature alla classe che specializzano
4 private String size;
5
6 public Dog(String breed, int age, String size, int conditions) {
7     super(breed, age, conditions);
8     this.size = size;
9 }
10
11 public Dog(String name, String breed, int age, String size, int conditions) {
12     super(name, breed, age, conditions);
13     this.size = size; // NB: this esplicito
14 }
15
16 // OVERWRITING del metodo toString()
17 public String toString() {
18     String s = super.toString() + ", taglia: " + size + ", animale: cane"; // NB: this implicito
19
20     return s;
21 }
22 }
23
24
25 class Cat extends Animal {
26     private boolean longFur;
27
28     public Cat(String breed, int age, boolean longFur, int conditions) {
29         super(breed, age, conditions);
30         this.longFur = longFur;
31     }
32
33     public Cat(String name, String breed, int age, boolean longFur, int conditions) {
34         super(name, breed, age, conditions);
35         this.longFur = longFur;
36     }
37
38     public String toString() {
39         String s = super.toString() + ", pelo lungo: " + longFur + ", animale: gatto";
40
41         return s;
42     }
43 }

```

Notiamo alcuni aspetti interessanti:

- Innanzitutto l'**overloading** del costruttore di ciascuna sottoclasse.
- L'**overwriting** del metodo `toString()`, il quale appartiene alla superclasse `Object`, e viene specializzato per ciascuna animale.

L'overloading del costruttore permette di inizializza gli animale già con un nome, oppure, se “trovatelli”, di dargli un nome al momento del loro insediamento nello shelter.

2.2 Classe AnimalShelter

Listing 3: AnimalShelter.java

```

1 public class AnimalShelter {
2     private int nAnimals = 0; // inizialmente lo shelter non contiene animali
3     private Animal[] hostedAnimals;
4
5     // Definisco lo shelter come un array di oggetti, quindi avente una capienza limitata a
6     // maxNumAnimals
7     public AnimalShelter(int maxNumAnimals) {

```

```

7     hostedAnimals = new Animal[maxNumAnimals];
8 }
9
10 // Converto l'astrazione dell'AnimalShelter in una stringa. Notare la chiamata al metodo
11 // toString della classe Animal.
12 public String toString() {
13     String s = "Numero di animali ospitati = " + nAnimals;
14
15     int i = 0;
16     while (i < nAnimals) {
17         s = s + hostedAnimals[i].toString();
18         ++i;
19     }
20
21     return s;
22 }
23
24 public int getNumAnimals() {
25     return nAnimals;
26 }
27
28 public int findHostedAnimalIndex(String name) {
29     int i = 0;
30     while (i < nAnimals) {
31         if (hostedAnimals[i].getName().equalsIgnoreCase(name)) return i;
32         ++i;
33     }
34
35     return nAnimals;
36 }
37
38 public boolean checkIfPresent(String name) {
39     return (findHostedAnimalIndex(name) < nAnimals);
40 }
41
42 public boolean checkIfFull() {
43     return (nAnimals == hostedAnimals.length);
44 }
45
46 public boolean hostAnimal(Animal animal, String name) {
47     if (checkIfFull()){
48         return false; // Se lo shelter pieno il metodo fallisce
49     }
50
51     animal.setName(name);
52     hostedAnimals[nAnimals] = animal;
53     ++nAnimals; // aggiorno il numero degli animali presenti
54
55     return true;
56 }
57
58 public boolean animalAdopted(String name) {
59     int i = findHostedAnimalIndex(name);
60     if (i == nAnimals) {
61         return false; // se l'animale non esiste il metodo fallisce
62     }
63
64     --nAnimals;
65     hostedAnimals[i] = hostedAnimals[nAnimals];
66
67     return true;
68 }
69 }

```

L'astrazione dello shelter avviene tramite la definizione di un array di animali e dei relativi metodi `hostAnimal(...)` e `animalAdopted(...)`. Come accennato in precedenza, un array in Java deve contenere oggetti aventi lo stesso tipo apparente, per questo sfruttiamo il **binding dinamico** definendo ciascun animale con tipo apparente `Animal` e con tipo esatto quello specifico della specie. Notare:

- come l'**information hiding** sia stato opportunamente implementato per non dare accesso diretto all'array di animali, ma consentendo di interagire con esso tramite gli opportuni metodi pubblici [2],
- come il metodo `toString()` venga anche qui ridefinito tramite overwriting [2],
- come il concetto che lo shelter debba avere un numero di posti limitato venga preso in carico dalla variabile privata `nAnimals`, che definisce la lunghezza dell'array di animali [3]. Un array in Java ha infatti una lunghezza fissata al momento del suo "istanziamento" nella heap.

2.3 Classe User

La classe `User` è molto semplice e permette di astrarre la figura dell'utente che interagisce con lo shelter nell'adottare e nel curare gli animali.

Listing 4: `AnimalShelter.java`

```
1 public class User {
2     private String name;
3     private String surname;
4     private String email;
5     private String cell;
6
7     public User(String name, String surname, String email, String cell){
8         this.name = name;
9         this.surname = surname;
10        this.email = email;
11        this.cell = cell;
12    }
13
14    public boolean adoptAnimal(AnimalShelter shelter, String name){
15        return shelter.animalAdopted(name);
16    }
17
18    public void cureAnimal(Animal animal){
19        animal.setConditions(100);
20    }
21 }
```

Notare come lo user non agisce direttamente sull'oggetto `AnimalShelter`, ma solo tramite i metodi pubblici di `AnimalShelter`, in questo modo si evitano possibili problemi di **security leaking**.

Un esempio di utilizzo delle classi sopra citate ci è fornito dal client `DemoAnimalShelter.java` riportato in appendice A. Sempre in appendice A, in figura 2 si può osservare il semplice e chiaro output del client.

Una rappresentazione della connessione delle classi brevemente descritte qui sopra è presentata sottoforma di diagramma UML in figura 1.

A Codice Sorgente

Listing 5: Animal.java

```
1 // Definendo Animal come astratta posso usare il Binding Dinamico
2 // per astrarre lo shelter come un array di tipo
3 // apparente Animal, ma di tipo esatto quello specifico della specie
4 // che estende la classe astratta Animal.
5 abstract class Animal {
6     private String name, breed;
7     private int age, conditions; // condizioni di salute dell'animale adottato
8
9     public Animal(String breed, int age, int conditions) {
10         this.breed = breed;
11         this.age = age;
12         this.conditions = conditions;
13     }
14
15     // OVERLOADING: ridefinisco il costruttore della classe Animal
16     // per accettare anche l'inizializzazione del campo "name"
17     public Animal(String name, String breed, int age, int conditions) {
18         this.name = name;
19         this.breed = breed;
20         this.age = age;
21         this.conditions = conditions;
22     }
23
24     // Converto l'astrazione dell'animale in una stringa. OVERWRITING: ridefinisco il metodo
25     // toString() della classe Object
26     public String toString() {
27         String s = "\n nome: " + name + ", razza: " + breed + ", eta': " + age + ", condizioni
28             generali: " + conditions + "%";
29         return s;
30     }
31
32     public String getName() {
33         return name;
34     }
35
36     public void setName(String name) {
37         this.name = name;
38     }
39
40     public String getBreed() {
41         return breed;
42     }
43
44     public int getAge() {
45         return age;
46     }
47
48     public String getConditions() {
49         return name;
50     }
51
52     public void setConditions(int conditions) {
53         this.conditions = conditions;
54     }
55
56     // Il metodo equals() fa parte dei metodi "standard" delle librerie Java, lo ridefinisco perch
57     // mi serve definire
58     // un concetto di uguaglianza tra gli animali.
59     public boolean equals(Animal otherAnimal) {
60         // Per definire l'uguaglianza uso i campi comuni a tutti gli animali. Tali campi rendono
61         // univoco il
```



```

58 // riconoscimento di un animale ospitato nello shelter
59 return
60     (this.name.equalsIgnoreCase(otherAnimal.name))
61         &&
62         (this.breed.equalsIgnoreCase(otherAnimal.breed))
63         &&
64         (this.age == otherAnimal.age);
65 }
66 }
67
68
69 class Dog extends Animal {
70
71 // Subclasses add more feature alla classe che specializzano
72 private String size;
73
74 public Dog(String breed, int age, String size, int conditions) {
75     super(breed, age, conditions);
76     this.size = size;
77 }
78
79 public Dog(String name, String breed, int age, String size, int conditions) {
80     super(name, breed, age, conditions);
81     this.size = size; // NB: this esplicito
82 }
83
84 // OVERWRITING del metodo toString()
85 public String toString() {
86     String s = super.toString() + ", taglia: " + size + ", animale: cane"; // NB: this implicito
87
88     return s;
89 }
90 }
91
92
93 class Cat extends Animal {
94 private boolean longFur;
95
96 public Cat(String breed, int age, boolean longFur, int conditions) {
97     super(breed, age, conditions);
98     this.longFur = longFur;
99 }
100
101 public Cat(String name, String breed, int age, boolean longFur, int conditions) {
102     super(name, breed, age, conditions);
103     this.longFur = longFur;
104 }
105
106 public String toString() {
107     String s = super.toString() + ", pelo lungo: " + longFur + ", animale: gatto";
108
109     return s;
110 }
111 }
112
113
114 class Sheep extends Animal {
115 private String sex;
116
117 public Sheep(String breed, int age, String sex, int conditions) {
118     super(breed, age, conditions);
119     this.sex = sex;
120 }
121
122 public Sheep(String name, String breed, int age, String sex, int conditions) {
123     super(name, breed, age, conditions);

```

```

124     this.sex = sex;
125 }
126
127 public String toString() {
128     String s = super.toString() + ", sesso: " + sex + ", animale: pecora";
129
130     return s;
131 }
132 }
133
134
135 class Donkey extends Animal {
136
137     public Donkey(String breed, int age, int conditions) {
138         super(breed, age, conditions);
139     }
140
141     public Donkey(String name, String breed, int age, int conditions) {
142         super(name, breed, age, conditions);
143     }
144
145     public String toString() {
146         String s = super.toString() + ", animale: asino";
147
148         return s;
149     }
150 }
151
152
153 class Piglet extends Animal {
154     public Piglet(String breed, int age, int conditions) {
155         super(breed, age, conditions);
156     }
157
158     public Piglet(String name, String breed, int age, int conditions) {
159         super(name, breed, age, conditions);
160     }
161
162     public String toString() {
163         String s = super.toString() + ", animale: maialino";
164
165         return s;
166     }
167 }

```

Listing 6: User.java

```

1 public class User {
2     private String name;
3     private String surname;
4     private String email;
5     private String cell;
6
7     public User(String name, String surname, String email, String cell){
8         this.name = name;
9         this.surname = surname;
10        this.email = email;
11        this.cell = cell;
12    }
13
14    public boolean adoptAnimal(AnimalShelter shelter, String name){
15        return shelter.animalAdopted(name);
16    }
17
18    public void cureAnimal(Animal animal){

```

```

19 animal.setConditions(100);
20 }
21 }

```

Listing 7: AnimalShelter.java

```

1 public class AnimalShelter {
2
3 private int nAnimals = 0; // inizialmente lo shelter non contiene animali
4 private Animal[] hostedAnimals;
5
6 // Definisco lo shelter come un array di oggetti, quindi avente una capienza limitata a
   maxNumAnimals
7 public AnimalShelter(int maxNumAnimals) {
8     hostedAnimals = new Animal[maxNumAnimals];
9 }
10
11 // Converto l'astrazione dell'AnimalShelter in una stringa. Notare la chiamata al metodo
   toString della classe Animal.
12 public String toString() {
13     String s = "Numero di animali ospitati = " + nAnimals;
14
15     int i = 0;
16     while (i < nAnimals) {
17         s = s + hostedAnimals[i].toString();
18         ++i;
19     }
20
21     return s;
22 }
23
24 public int getNumAnimals() {
25     return nAnimals;
26 }
27
28 public int findHostedAnimalIndex(String name) {
29     int i = 0;
30     while (i < nAnimals) {
31         if (hostedAnimals[i].getName().equalsIgnoreCase(name)) return i;
32         ++i;
33     }
34
35     return nAnimals;
36 }
37
38 public boolean checkIfPresent(String name) {
39     return (findHostedAnimalIndex(name) < nAnimals);
40 }
41
42 public boolean checkIfFull() {
43     return (nAnimals == hostedAnimals.length);
44 }
45
46 public boolean hostAnimal(Animal animal, String name) {
47
48     if (checkIfFull()){
49         return false; // Se lo shelter pieno il metodo fallisce
50     }
51
52     animal.setName(name);
53     hostedAnimals[nAnimals] = animal;
54     ++nAnimals; // aggiorno il numero degli animali presenti
55
56     return true;
57 }

```

```

58
59 public boolean animalAdopted(String name) {
60     int i = findHostedAnimalIndex(name);
61     if (i == nAnimals) {
62         return false; // se l'animale non esiste il metodo fallisce
63     }
64
65     --nAnimals;
66     hostedAnimals[i] = hostedAnimals[nAnimals];
67
68     return true;
69 }
70 }

```

Listing 8: DemoAnimalShelter.java

```

1 public class DemoAnimalShelter {
2     public static void main(String arg[]) {
3
4         // Inizializzo un utente
5         User Mila = new User("Mila", "Racca", "Mila.Racca@me.it", "+39 377 24583");
6         User Piero = new User("Piero", "Gillono", "Piero.Gillono@me.it", "+39 347 58232");
7
8         // Inizializzo lo shelter definendo il nume di posti che dispone per ospitare gli animali
9         AnimalShelter shelter = new AnimalShelter(100);
10
11         Animal dog1 = new Dog("Labrador", 6, "mid", 90);
12         Animal dog2 = new Dog("Bassotto", 6, "small", 80);
13         Animal cat1 = new Cat("European", 5, false, 85);
14         Animal cat2 = new Cat("Siamese", 2, true, 33);
15         Animal sheep1 = new Sheep("Irlandese", 4, "female", 95);
16         Animal sheep2 = new Sheep("Yorkshire", 4, "female", 60);
17         Animal sheep3 = new Sheep("British Columbia", 4, "male", 70);
18         Animal donkey1 = new Donkey("Sarda", 10, 90);
19
20         // Ospitando un animale, gli assegno un nome :)
21         shelter.hostAnimal(dog1, "Buddy");
22         shelter.hostAnimal(dog2, "Otto von Bismarck");
23         shelter.hostAnimal(cat1, "Tom");
24         shelter.hostAnimal(cat2, "Dorotea");
25         shelter.hostAnimal(sheep1, "Margarita");
26         shelter.hostAnimal(sheep2, "Jimmy");
27         shelter.hostAnimal(sheep3, "Ed");
28         shelter.hostAnimal(donkey1, "Arturo");
29
30         System.out.println("\nSituazione dello shelter a un mese dall'apertura:\n" + shelter);
31
32         System.out.println("\nOtto von Bismarck ospitato nello shelter: " + shelter.checkIfPresent("
            Otto von Bismarck"));
33
34         Mila.adoptAnimal(shelter, "Otto von Bismarck");
35
36         System.out.println("Otto von Bismarck viene adottato. \nOtto von Bismarck ospitato nello
            shelter: " + shelter.checkIfPresent("Otto von Bismarck"));
37
38         System.out.println("\nIl veterinario riesce a curare i due gatti presenti in struttura.");
39         Piero.cureAnimal(cat1);
40         Piero.cureAnimal(cat2);
41
42         System.out.println("\nSituazione dello shelter:\n" + shelter);
43     }
44 }

```

```

PS C:\Users\david\Desktop\Master_Full_Stack\Design Patterns\ITFS---Design-Pattern-Exam\java\basic java exam> java DemoAnimalShelter

Situazione dello shelter a un mese dall'apertura:
Numero di animali ospitati = 8
nome: Buddy, razza: Labrador, eta': 6, condizioni generali: 90%, taglia: mid, animale: cane
nome: Otto von Bismarck, razza: Bassotto, eta': 6, condizioni generali: 80%, taglia: small, animale: cane
nome: Tom, razza: European, eta': 5, condizioni generali: 85%, pelo lungo: false, animale: gatto
nome: Dorotea, razza: Siamese, eta': 2, condizioni generali: 33%, pelo lungo: true, animale: gatto
nome: Margarita, razza: Irlandese, eta': 4, condizioni generali: 95%, sesso: female, animale: pecora
nome: Jimmy, razza: Yorkshire, eta': 4, condizioni generali: 60%, sesso: female, animale: pecora
nome: Ed, razza: British Columbia, eta': 4, condizioni generali: 70%, sesso: male, animale: pecora
nome: Arturo, razza: Sarda, eta': 10, condizioni generali: 90%, animale: asino

Otto von Bismarck è ospitato nello shelter: true
Otto von Bismarck viene adottato.
Otto von Bismarck è ospitato nello shelter: false

Il veterinario riesce a curare i due gatti presenti in struttura.

Situazione dello shelter:
Numero di animali ospitati = 7
nome: Buddy, razza: Labrador, eta': 6, condizioni generali: 90%, taglia: mid, animale: cane
nome: Arturo, razza: Sarda, eta': 10, condizioni generali: 90%, animale: asino
nome: Tom, razza: European, eta': 5, condizioni generali: 100%, pelo lungo: false, animale: gatto
nome: Dorotea, razza: Siamese, eta': 2, condizioni generali: 100%, pelo lungo: true, animale: gatto
nome: Margarita, razza: Irlandese, eta': 4, condizioni generali: 95%, sesso: female, animale: pecora
nome: Jimmy, razza: Yorkshire, eta': 4, condizioni generali: 60%, sesso: female, animale: pecora
nome: Ed, razza: British Columbia, eta': 4, condizioni generali: 70%, sesso: male, animale: pecora

```

Figura 2: L'output del client.