Programmazione II

Nicolò Fornari

December 24, 2015

Concetti:

- Ereditarietà
- Collections
- Classi Abstract e Polimorfismo
- Interface
- Override
- Overload
- Casting
- Generics
- Eventi
- Static
- Final

1 Ereditarietà

Così come per scope considero la variabile più prossima, per le classi cerco il metodo più prossimo.

In realtà ogni classe che sembra non ereditare niente eredità da Object.
La classe Object è la root di tutta la gerarchia di classi.

Se voglio usare un metodo della superclasse: super.metodo()
Si può iterare: super.super.metodo()

Esempio

```
public class MySlider extends javafx.scene.control.Slider {

    public MySlider() {
        super(0,255,0);

        setShowTickLabels(true);
        setShowTickMarks(true);
        setMajorTickUnit(50);
        setMinorTickCount(5);
        setBlockIncrement(10);
    }
}
```

2 Classi abstract

Non possono essere istanziate,devono essere subclassate.

```
import java.util.*;
public abstract class Stack extends LinkedList {
    public void inserisci(int x) {
    Number n = new Number(x);
    this.add(n);
    }

    abstract public int estrai();
}
```

```
import java.util.*;

class Pila extends Stack {
    @Override
    public int estrai() {
        Number x = null;
        Iterator iter = this.iterator();
        while (iter.hasNext()) {
            x = (Number) iter.next();
        }
        if (x == null) {
            System.out.println("Pila vuota");
            System.exit(1);
        }
        iter.remove();
        return x.getInt();
    }
}
```

3 Polimorfismo

Voglio creare a runtime (Dynamic Binding) oggetti diversi a seconda delle scelte dell'utente

```
public class Main {
    public static void main(String[] args) {
        Stack s;
        if (args[0] != "")
            s=new Coda();
        else
            s = new Pila();
        s.inserisci(1);
        s.inserisci(2);
        s.inserisci(3);
        for (int k=0;k<=4;k++){
            int v=s.estrai();
            System.out.println(v);
        }
    }
```

4 Interfaces

Interface	Abstract class
Solo metodi non implementati Non può contenere variabili Una classe può implementare più interfacce alla volta.	Può avere metodi implementati e variabili. - Non vale l'ereditarietà multipla

Non è possibile che una classe erediti da due classi astratte contemporaneamente, perché se viene chiamato un metodo non sovrascritto dalla sottoclasse ed implementato in entrambe le superclassi astratte, ci sarebbe l'abiguità di andare a pescare da una superclasse piuttosto che dall'altra.

```
public class drawCircle implements EventHandler {

    Color color;
    Canvas canv;
    int r = 50;

    public drawCircle(Color color,Canvas canv) {
        this.color = color;
        this.canv = canv;
    }

    @Override
    public void handle(Event t) {
        double w = canv.getWidth();
        double h = canv.getHeight();

        int x = (int) (Math.random()*w);
        int y = (int) (Math.random()*h);

        canv.getGraphicsContext2D().setFill(color);
        canv.getGraphicsContext2D().fillOval(x,y,r,r);
    }
}
```

5 Collections

Una collection è un oggetto che raggruppa elementi multipli in un'unica entità, all' interno di cui i dati possono essere trattati o trasferiti da un metodo ad un altro.

- -	Presenza di duplicati (come identificatore)	Accessibile con un indice	Ordine di inserimento mantenuto
Set	No	No	No
List	Si	Si	Si

Poichè una Collection prende in input degli Oggetti, se voglio creare una Collection di variabili primitive (eg interi) ho bisogno di un wrapper per la variabile (eg. Integer)

```
public class Mazzo {

    List<Carta> Deck;
    int[] valori = {1,2,3,4,5,6,7,8,9,10};
    String[] semi = {"Denari","Spade","Bastoni","Coppe"};

    public Mazzo() {
        Deck = new ArrayList<>();
        for (int i=0;i<4;i++) {
                for (int j=0;j<10;j++) {
                    Carta newCard = new Carta(semi[i],valori[j]);
                    Deck.add(newCard);

                }
        }
        Collections.shuffle(Deck);
    }
}
```

The Collections class provides three constants, representing the empty Set, the empty List, and the empty Map

- $Collections.EMPTY\_SET$
- $Collections.EMPTY\_LIST$
- $Collections.EMPTY\_MAP$

The main use of these constants is as input to methods that take a Collection of values, when you don't want to provide any values at all.

6 Object ordering - comparable

A List l may be sorted as follows:
Collections.sort(l);

If the list consists of String elements, it will be sorted into lexicographic (alphabetical) order.
If it consists of Date elements, it will be sorted into chronological order.
How does Java know how to do this?
String and Date both implement the Comparable interface. The Comparable interfaces provides a natural ordering for a class, which allows objects of that class to be sorted automatically.

int compareTo(Object o)

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

```
public class Carta implements Comparable {
    String seme;
    String t;
    int valore;

    public Carta(String seme, int valore) {
        this.seme = seme;
        this.valore = valore;
    }

    @Override
    public int compareTo(Object o) {
        Carta c = (Carta) o;
        return this.valore-c.valore;
    }
```

7 Object ordering - comparator

```
public class Giocatore {
    List<Carta> Mano;

    public Giocatore() {
        Mano = new ArrayList();
    }

    public void sortHandByValue() {
        Comparator cmp1 = new ComparatorByValue();
        Collections.sort(Mano,cmp1);
    }

    public void sortHandBySeed() {
        Comparator cmp2 = new ComparatorBySeed();
        Collections.sort(Mano,cmp2);
    }
}
```

```
public class ComparatorBySeed implements Comparator {

    @Override
    public int compare(Object o1,Object o2) {
        Carta c1 = (Carta) o1;
        Carta c2 = (Carta) o2;
        return c1.seme.compareTo(c2.seme);
    }
}
```

```
public class ComparatorByValue implements Comparator {

    @Override
    public int compare(Object o1,Object o2) {
        Carta c1 = (Carta) o1;
        Carta c2 = (Carta) o2;
        return c1.valore - c2.valore;
    }
```

8 Listener

Listener esterno

```
Listener lsn = new Listener(this);
sp.addEventHandler(MouseEvent.MOUSE_CLICKED, lsn);
```

```
public class Listener implements EventHandler {
    SimEsame w = null;

    Listener(SimEsame w) {
        this.w = w;
    }

    @Override
    public void handle(Event t) {
        w.addCross();
        int id = Integer.parseInt(((Circle)t.getTarget()).getId());
    }
}
```

Listener interno

```
Button btn = new Button();
btn.setText("Reset");
Reset reset  = new Reset();
btn.addEventHandler(ActionEvent.ACTION, reset);



class Reset implements EventHandler {
        @Override
        public void handle(Event t) {
            updateScore();

            }
        }
    }
```

Listener anonimo

```
Circle c = new Circle(r*i);
c.setOnMouseClicked(new EventHandler<MouseEvent>() {
                @Override public void handle(MouseEvent event) {
                x = event.getX();
                y = event.getY();
        }
    });
```

Self listener

```
public class SelfListener extends Application implements EventHandler {

    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.addEventHandler(ActionEvent.ACTION, this);

        StackPane root = new StackPane();
        root.getChildren().add(btn);

        Scene scene = new Scene(root, 300, 250);

        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void handle(Event event) {
        System.out.println("Hi");
    }
```