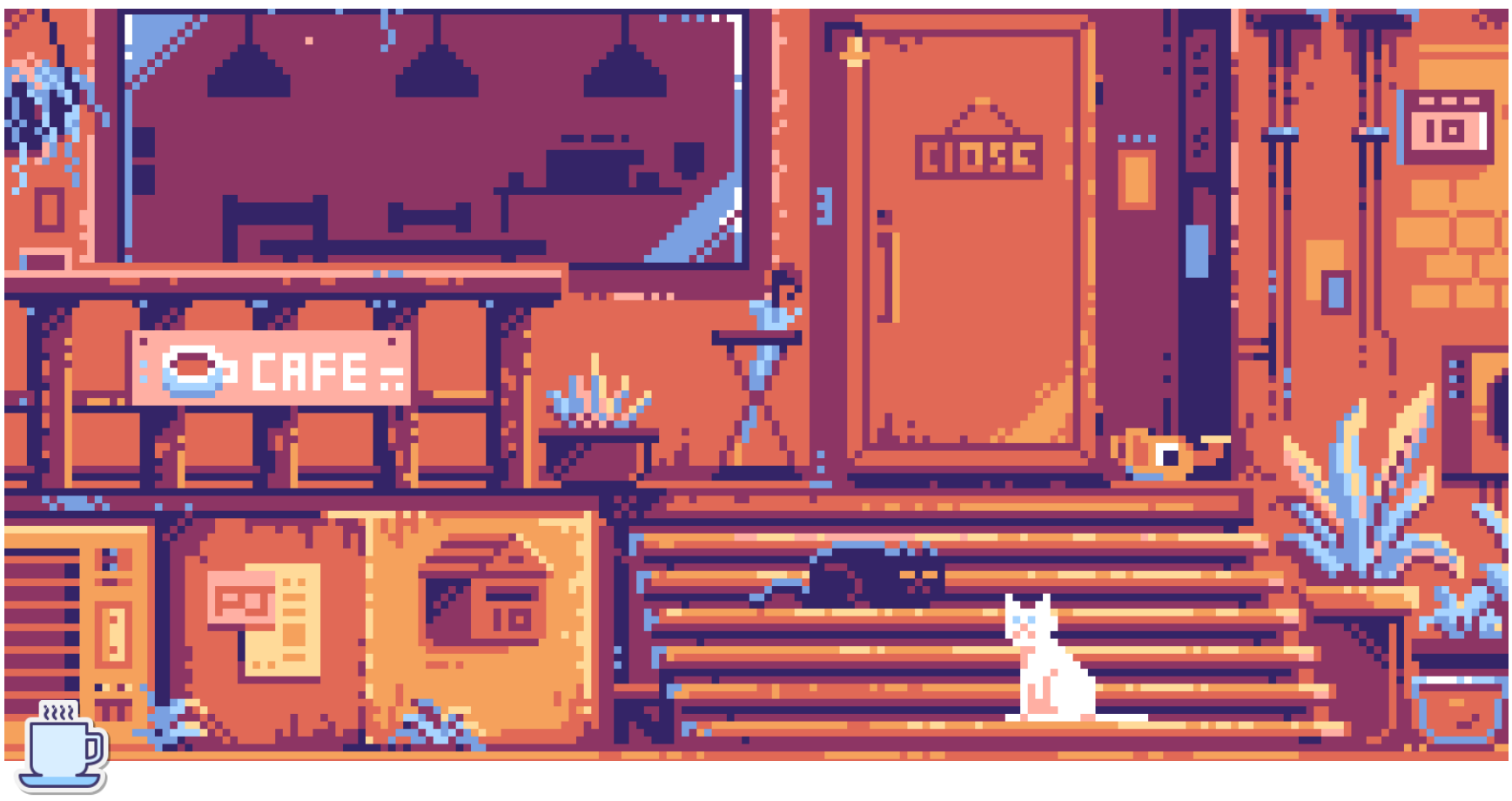


Utilizzare oggetti

0 Introduzione

Java è un popolare linguaggio di programmazione, creato nel 1995.

È di proprietà di Oracle e più di 3 miliardi di dispositivi eseguono Java.

Java viene usato per:

- Applicazioni mobili(specialmente Android)
- Applicazioni desktop
- Applicazioni web
- Server web e server applicativi
- Giochi
- Connessione alla banca dati
- ecc…

1 Sintassi java

Prima di tutto dobbiamo conoscere la sintassi di Java per poter iniziare a programmare:

```
public class main{
  public static void main(string[] args){
    System.out.println("Hello world");
  }
}
```

Come possiamo vedere abbiamo una prima classe chiamata `Main` in questo caso.

> 📖 ***NOTA:***
>
> Il nome del file deve sempre corrispondere al nome della classe.

Successivamente avremo il metodo `main()` , metodo fondamentale per poter eseguire il proprio codice, e successivamente `System.out.prinln()` che è la stampa.

1.1 Tipi di variabili

La variabili sono contenitori per la memorizzazione di valori di dati. In Java esistono diversi tipi di variabili, ad esempio:

- String
- Int
- Float
- Char
- Boolean

Per creare una variabile, è necessario specificare il tipo e assegnarle un valore, utilizzando l'operatore `=` che non significa "uguaglia" ma "diventa":

```
tipo nomevariabile = value;
```

Andiamo a vedere degli esempi:

```
String = "Marco"; //String
int numero = 5; //Integer
float floatNum = 5.99f //Floating
char lettera = "D" //Char
boolean bool = true; //Boolean
```

1.2 Cicli in Java

In Java esistono 2 tipi di cicli:

- While: che va a ciclare finché una condizione specificata è true:

  ```
  int i=0;
  while (i<5) {
    System.out.println(i);
    i++;
  }
  ```

- For: si usa quando si sa esattamente quante volte eseguire il ciclo:

  ```
  for(int i=0;i<5;i++){
    System.out.println(i);
  }
  ```

1.3 Metodi Java

Un metodo è un blocco di codice che viene eseguito solo quando viene chiamato.

È possibile passare dati, noti come parametri, in un metodo.

Possiamo dire che esistono 2 tipi di metodo, il quale si contraddistinguono tra:

- Void: ossia quei metodi che non restituiscono nessun valore;

  ```
  public class Main {
    static void myMethod(String fname, int age) {
  ```

```
    System.out.println(fname + " ha " + age + " anni");
  }

  public static void main(String[] args) {
    myMethod("Marco", 5);
    myMethod("Luca", 8);
    myMethod("Antonio", 31);
  }
}
```

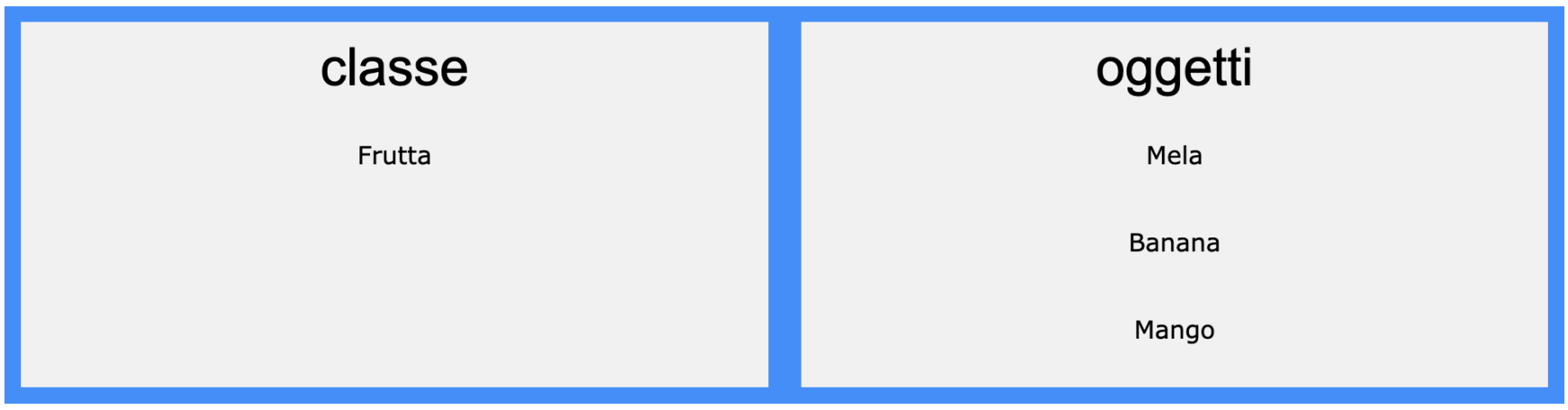
- Int, char ecc…: ossia quei metodi che restituiscono un valore.

```
public class Main {
    static int MyMethod(int value) {
        int sum = 0;
        for(int i=0;i<=10;i++){ sum += i;}
        return sum;
    }
    public static void main(String[] args) {
        int value = 0;
        System.out.println(MyMethod(value));
    }
}
```

1.4 Java OOP

OOP sta per programmazione orientata ad oggetti. Le classi e gli oggetti sono i due aspetti principali della programmazione orientata ad oggetti.

Proviamo a fare un esempio per distinguere classe e oggetti:



Quindi una classe è un modello per oggetti e un oggetto è un'istanza di una classe.

Proviamo a vedere ora un esempio:

```
class Second{
    int x = 5;
}
public class Main {

    public static void main(String[] args) {
        Second object = new Second();
        System.out.println(5+ object.x);
    }
}
```

Sapendo questo possiamo introdurre i costruttori in java.

Un costruttore in Java è un metodo speciale utilizzato per inizializzare gli oggetti. Il costruttore viene chiamato quando viene creato un oggetto di una classe. Può essere utilizzato per impostare i valori iniziali per gli attributi dell'oggetto.

> ⚠️ ***ATTENZIONE:***
>
> Il costruttore deve ***SEMPRE*** avere lo stesso nome della classe dove viene creato.

Ad esempio il codice di prima con l'uso di un costruttore diventa:

```
class Second{
    int x;
```

```
    public Second(){
        x = 5;
    }
}
public class Main {

    public static void main(String[] args) {
        Second object = new Second();
        System.out.println(5+ object.x);
    }
}
```

Esercizi

🧮 1.

```
/*Creare una classe Counter che rappresenti il funzionamento di un
contatore. Al suo interno sarà presente un campo privato, di tipo
intero, che memorizzi il suo valore attuale. Dal punto di vista delle
funzionalità, la classe conterrà:
  - Un metodo getValue che ritorni il valore attuale del contatore
  - Un metodo click che permetta l'incremento del campo di un'unità
  - Un metodo reset che reimposti il valore del campo a 0
  - Un metodo undo che annulli l'ultima esecuzione di click, evitando
che il valore del campo non scenda sotto allo zero.
La classe CounterTester dovrà:
  - Creare un oggetto di tipo Counter
  - Incrementare il contatore di tre unità
  - Stampare il suo valore
  - Decrementare il contatore di un'unità
  - Stampare il suo valore, controllando che sia pari a due
  - Decrementare il contatore di tre unità
  - Stampare il suo valore, controllando che sia pari a zero
Consiglio: la classe Math offre il metodo max() che può essere
utilizzato nella definizione di undo per evitare che il decremento porti
ad avere un numero negativo.*/
package Main;
class Counter {
    private int value;
    public int getValue(){ return value; }
    public void click(){ value += 1; }
    public void reset(){ value = 0; }
    public void undo(){ value = Math.max(value -1,0); }
}
public class Main {

    public static void main(String[] args) {
    Counter OBJECT = new Counter();
    OBJECT.click();
    OBJECT.click();
    OBJECT.click();
    OBJECT.getValue();
    OBJECT.undo();
    System.out.println("Il numero dev'essere uguale a 2: " + OBJECT.getValue());
    OBJECT.undo();
    OBJECT.undo();
    System.out.println("Il numero dev'essere uguale a 0: " + OBJECT.getValue());
    }
}
```

2.

```
/*Estendere l’esercizio precedente aggiungendo un ulteriore campo
privato e di tipo intero che rappresenti il valore massimo assumibile
dal contatore. Tale valore è impostabile esclusivamente tramite il
metodo setLimit, il quale accetta un parametro di tipo intero.
L’introduzione di questo limite modificherà il comportamento del
metodo click. Infatti, non sarà possibile incrementare ulteriormente il
contatore oltre il valore massimo.
La classe CounterTester riprende il comportamento di quella vista
nell’esercizio precedente. La differenza risiede nel fatto che il primo
metodo chiamato sarà setLimit, il quale dovrà impostare il valore
massimo del contatore a tre. L’efficacia di questo limite dovrà essere
accertata chiamando quattro volte di fila il metodo click e accertarsi
che il valore attuale del contatore sia pari a tre..*/
package Main;
class Counter {
    private int value;
    private int max_value;
    public void set_limit(int i){max_value = i;}
    public int getValue(){ return value; }
    public void click(){ value = Math.max(value -+1,max_value); }
    public void reset(){ value = 0; }
    public void undo(){ value = Math.max(value -1,0); }
}
public class Main {

    public static void main(String[] args) {
    Counter OBJECT = new Counter();
    OBJECT.set_limit(3);
    OBJECT.click();
    OBJECT.click();
    OBJECT.click();
    OBJECT.click();
    System.out.println("Il numero dev'essere uguale a 3: " + OBJECT.getValue());
    OBJECT.getValue();
    OBJECT.undo();
    System.out.println("Il numero dev'essere uguale a 2: " + OBJECT.getValue());
    OBJECT.undo();
    OBJECT.undo();
    OBJECT.undo();
    System.out.println("Il numero dev'essere uguale a 0: " + OBJECT.getValue());
    }
}
```

3.

```
/*Creare una classe chiamata RangeInput che rappresenti un intervallo
numerico tra due numeri.
Attributi:
 - Min: di tipo intero, rappresenta il valore minimo dell’intervallo
 - Max: di tipo intero, rappresenta il valore massimo dell’intervallo
 - Current: di tipo intero, rappresenta il valore attuale all’interno
dell’intervallo.
Costruttore: ridefinito in modo tale che accetti due parametri che
rappresentano i due limiti dell’intervallo.
Metodi:
 - Up: incrementa di un’unità il valore attuale dell’intervallo,
controllando che non superi il limite massimo
 - Down: decrementa di un’unità il valore attuale dell’intervallo,
controllando che non superi il limite minimo
La classe RangeInputTester vedrà la creazione un oggetto di tipo
RangeInput valorizzando i due estremi dell’intervallo.
Successivamente, dovranno essere controllata la consistenza dei
suoi limiti chiamando più volte i metodi Up and Down a piacimento.*/
package Main;
import java.util.Scanner;
class RangeInput {
   //Dichiaro gli attributi
   int min;
   int max;
   int current;
   //Costruttore
    public RangeInput(int input_min, int input_max,int input_current)
    {
        min = input_min;
        max = input_max;
        current = input_current;
    }
    public int getCurrent() {return current;}
    public void Up() {current = Math.min(current +1,max);}
    public void Down() {current = Math.max(current -1,min);}
}
public class Main {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int input_min;
        int input_max;
        int cont;
        System.out.print("Inserire il valore minimo dell'intervallo: ");
        input_min = scan.nextInt();
        do {System.out.print("Inserire il valore massimo dell'intervallo: ");
            input_max = scan.nextInt(); } while (input_max <= input_min);
        RangeInput OBJECT = new RangeInput(input_min, input_max,((input_min+input_max)/2));
        do {
            System.out.println("Il valore attuale è di " + OBJECT.getCurrent());
            System.out.println("\t\t MENU' SCELTA:");
            System.out.println("-Inserisci 1 per incrementare di 1 il valore dell'intervallo,\n" +
                    "-Inserisci 2 per decrementare di 1 il valore dell'intervallo,\n" +
                    "-Inserisci 0 se vuoi uscire dal programma,\n" +
                    "Scelta:");
            cont = scan.nextInt();
            switch (cont) {
                case 0 -> System.out.println("Hai deciso di uscire dal programma");
                case 1 -> OBJECT.Up();
                case 2 -> OBJECT.Down();
            }
        }while(cont !=0);
    }
}
```

4.

```
/*Creare una classe Circuit che gestisca l’illuminazione di un corridoio.
Attributi: tutti e tre private

 - firstSwitch: può assumere due valori: 1 o 0, che rappresentano
  rispettivamente il selezionamento o meno della prima lampada
 - secondSwitch: come firstSwitch, ma riferita alla seconda
  lampada
 - lampState: rappresenta lo presenza o meno di corrente,
valorizzata con i valori 0 o 1 analogamente agli attributi
precedenti
Metodi:
 - getFirstSwitchState: ritorna il valore di firstSwitch
 - getSecondSwitchState: ritorna il valore di secondSwitch
 - getLampState: ritorna il valore di lampState
 - toggleFirstSwitch: cambia lo stato di firstSwitch e lampState da
   0 a 1 e viceversa
 - toggleSecondSwitch: cambia lo stato di secondSwitch e
   lampState da 0 a 1 e viceversa
La classe CircuitTester dovrà essere costruita accertandosi che
l’esecuzione consecutiva dello stesso toggleSwitch riporti lo stato
dello switch corrispondente e della lampada a zero. Per controllare il
contenuto dei diversi attributi durante i test, stampare il valore
ritornato dai metodi get..*/
package Main;
import java.util.Scanner;
class Circuit {
    private boolean firstSwitch = false; //true=1, false=0
    private boolean secondSwitch = false; //true=1, false=0
    private boolean lampState = false; //true=1, false=0
    public boolean GetFirstSwitchState(){return firstSwitch;}
    public boolean GetSecondSwitchState(){return secondSwitch;}
    public boolean GetLampState(){return lampState;}
    public void ToggleFirstSwitch(){
        firstSwitch = !firstSwitch;
        lampState = !lampState;
    }
    public void ToggleSecondSwitch(){
        secondSwitch = !secondSwitch;
        lampState = !lampState;
    }
}
public class Main {

    public static void main(String[] args) {
        Circuit OBJECT = new Circuit();

        OBJECT.ToggleFirstSwitch();
        OBJECT.ToggleFirstSwitch();
        OBJECT.ToggleSecondSwitch();
        System.out.println("Switch 1: " + OBJECT.GetFirstSwitchState());
        System.out.println("Switch 2: " + OBJECT.GetSecondSwitchState());
        System.out.println("Lamp: " + OBJECT.GetLampState());

        OBJECT.ToggleSecondSwitch();
        System.out.println("Switch 1: " + OBJECT.GetFirstSwitchState());
        System.out.println("Switch 2: " + OBJECT.GetSecondSwitchState());
        System.out.println("Lamp: " + OBJECT.GetLampState());

    }
}
```

6.

```
/*Realizzare un gestore di un conto bancario.
Attributi:
 - Balance: di tipo double, privata. Rappresenta il saldo del conto
Costruttore: ridefinire il costruttore di default in modo tale che:
 - Un oggetto istanziato senza parametri contenga un saldo pari a
  zero
 - Un oggetto può essere istanziato con un saldo fornito da input,
  valorizzando il corrispettivo attributo
Metodi:

 - Deposit: effettua un deposito, incrementando il saldo attuale di
   un ammontare pari al valore del parametro passato
 - Withdraw: effettua un prelievo, decrementando saldo attuale di
   un ammontare pari al valore del parametro passato
 - getBalance: restituisce il valore attuale del saldo

La classe BankAccountTester accerterà il corretto funzionamento
della classe depositando inizialmente un valore pari a mille.
Successivamente, dovranno essere fatti due prelievi: uno da 500 e
uno da 400. Stampare quindi il valore del saldo, accertandosi che il
saldo risultante sia pari a 100.*/
package Main;

import java.util.Scanner;

class BankAccount {
    private double balance;

    public BankAccount() {balance = 0;}
    public BankAccount(double VALUE) {balance = VALUE;}
    public void Deposit(double MONEY) {balance += MONEY;}
    public void WithDraw(double MONEY) {balance -= MONEY;}
    public double GetBalance() { return balance;}
}

public class Main {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        double VALUE;
        int choice;
        BankAccount OBJECT;

        System.out.println("\t\t MENU' SCELTA:");
        System.out.println("-Inserisci 1 se vuoi inserire un saldo,\n" +
                "-Inserisci 0 se non vuoi inserire un saldo,\n" +
                "Scelta:");
        choice = scan.nextInt();
        switch (choice) {
            case 0 -> OBJECT = new BankAccount();
            case 1 -> {
                System.out.println("Inserire il proprio saldo:");
                VALUE = scan.nextFloat();
                OBJECT = new BankAccount(VALUE);
            }
            default -> throw new IllegalStateException("Unexpected value: " + choice);
        }

        OBJECT.Deposit(1000);
        OBJECT.WithDraw(100);
        OBJECT.WithDraw(300);

        System.out.println("Saldo finale: " + OBJECT.GetBalance());
    }
}
```