



Utilizzare oggetti

▼ INDICE

[0 Introduzione](#)

[1 Sintassi java](#)

[1.1 Tipi di variabili](#)

[1.2 Cicli in Java](#)

[1.3 Metodi Java](#)

[1.4 Java OOP](#)

[Esercizi](#)

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.println()` 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();
        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 la 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());
    }
}
```



7.

```
/*Estendere l'esercizio precedente aggiungendo il metodo addInterest
il quale riceve in input una percentuale di interesse da applicare al
saldo al fine di incrementarlo.
Es: dato un saldo è pari a 100, la chiamata addInterest(20)
aumenterebbe il valore del saldo del 20%, portandolo a 120.
La classe BankAccountTester in questo caso crea un nuovo conto
con un saldo pari a 1000, applicando successivamente un interesse
del 10%. Il risultato finale vedrà un valore del saldo pari a 1100.*/
package Main;

import java.util.Scanner;

class BankAccount {
    private double balance;

    public BankAccount() {balance = 1000;}
    public BankAccount(double VALUE) {balance = VALUE;}
    public void Deposit(double MONEY) {balance += MONEY;}
    public void Withdraw(double MONEY) {balance -= MONEY;}
    public void AddInterest(double PERCENTAGE){
        balance = (PERCENTAGE/100 * balance) + balance;
    }
    public double GetBalance() { return balance;}
}

public class Main {

    public static void main(String[] args) {
        BankAccount OBJECT = new BankAccount();
        OBJECT.AddInterest(10);
        System.out.println("Saldo finale: " + OBJECT.GetBalance());
    }
}
```



8.

```
/*Creare una classe chiamata SavingsAccount. Essa avrà la stessa
struttura di BankAccount vista nell'esercizio precedente, differendo
solamente nella modalità con cui il tasso di interesse viene gestito.
Sarà infatti necessario aggiungere un attributo privato di tipo double
che rappresenti il tasso di interesse, estendendo il costruttore custom
in modo tale che ne gestisca la valorizzazione. Così facendo, il
metodo addInterest non necessiterà più nessun parametro di input,
lasciando invariata la semantica.*/
package Main;

import java.util.Scanner;

class BankAccount {
    private double balance;
    private double interest;
    public BankAccount() {
        balance = 0;
        interest = 0;
    }
    public BankAccount(double VALUE, double PERCENTAGE) {
        balance = VALUE;
        interest = PERCENTAGE;
    }
    public void Deposit(double MONEY) {balance += MONEY;}
    public void Withdraw(double MONEY) {balance -= MONEY;}
    public void AddInterest(){
        balance = (interest/100 * balance) + balance;
    }
    public double GetBalance() { return balance;}
}

public class Main {

    public static void main(String[] args) {
        BankAccount OBJECT = new BankAccount(1000,10);
        OBJECT.AddInterest();
        System.out.println("Saldo finale: " + OBJECT.GetBalance());
    }
}
```



9.

```
/*Creare una classe chiamata CashRegister che rappresenti il
funzionamento di un registratore di cassa.
Attributi:

- Purchase: privato e di tipo double, contiene l'importo totale dei
prodotti che si vogliono acquistare
- Payment: privato e di tipo double, contiene l'importo ricevuto
dall'acquirente per il pagamento dei prodotti
- History: private e di tipo String, rappresenta lo scontrino con al
suo interno i prezzi dei prodotti acquistati
Metodi:
- recordPurchase: registra la vendita di un prodotto, ricevendo in
input un parametro che indichi il prezzo del prodotto che
acquistato. Tale funzione dovrà aggiornare l'importo totale dei
prodotti acquistati, aggiornando lo scontrino con l'aggiunta di
una nuova riga che contiene il prezzo del prodotto appena
scelto
- receivePayment: simula il passaggio di denaro tra acquirente e
venditore, aggiornando l'attributo relativo agli importi ricevuti
- giveChange: simula il calcolo del resto, ossia la differenza tra i
soldi forniti dall'acquirente per il pagamento e l'importo totale.
Una volta calcolato il resto, il contenuto dello scontrino e i due
totali dovranno essere azzerati. Il resto dovrà essere ritornato
dalla funzione come output.
- printReceipt: stampa il contenuto dello scontrino e,
successivamente, il totale dei prodotti acquistati.
La classe CashRegisterTest dovrà simulare un normale
acquisto, aggiungendo l'importo di diversi prodotti tramite la
recordPurchase e poi fornendo i soldi per il pagamento tramite
la receivePayment. Prima di chiamare la giveChange per dare il
resto, l'esecuzione del programma dovrà stampare il contenuto
dello scontrino tramite la printReceipt.*/
package Main;
class CashRegister {
    private double purchase;
    private double payment;
    private String history;
    public CashRegister(){
        purchase = 0;
        payment = 0;
        history = "";
    }
    public void RecordPurchase(double VALUE){
        purchase += VALUE;
        history += String.valueOf(VALUE) + "\n";
    }
    public void ReceivePayment(double MONEY){ payment += MONEY;}
    public double GiveChange(){
        double rest = payment-purchase;
        purchase = 0;
        payment = 0;
        history = "";
        return rest;
    }
    public void PrintReceipt(){
        System.out.println(history);
        System.out.println("Total: " + purchase);
    }
}

public class Main {

    public static void main(String[] args) {
        CashRegister OBJECT = new CashRegister();
        OBJECT.RecordPurchase(30);
        OBJECT.RecordPurchase(20);
        OBJECT.ReceivePayment(60);
        OBJECT.PrintReceipt();
        System.out.println("Il resto è di " + OBJECT.GiveChange() + " euro.");
    }
}
```



10.

```
/*Estendere la classe CashRegister dell'esercizio precedente
inserendo due nuovi attributi:
- salesTotal: privata e di tipo double, memorizza il totale di tutte
le vendite avvenute

- salesCount: privata e di tipo int, memorizza il numero totale di
vendite avvenute
Il costruttore e i metodi invece dovranno essere modificati nel
seguente modo:
- Costruttore: dovrà assegnare il valore 0 ai due nuovi attributi
- recordPurchase: l'importo dell'oggetto venduto dovrà essere aggiunto
al valore attuale di salesTotal
- giveChange: poiché questo metodo conclude un acquisto, l'attributo
salesCount dovrà essere incrementato di una unità
Dovranno essere aggiunti invece i seguenti metodi:
- getSalesTotal: ritornerà il valore attuale di salesTotal
- getSalesCount: ritornerà il valore attuale di salesCount
- Reset: reimposterà a 0 i valori di salesTotal e salesCount
La classe CashRegisterTester in questo caso dovrà accertarsi che, a
fronte vari acquisti, la getSalesCount e la getSalesTotal ritornino
rispettivamente il totale ed il numero delle vendite avvenute fino al
momento delle due chiamate.*/
package Main;
class CashRegister {
    private double purchase;
    private double payment;
    private String history;
    private double salesTotal;
    private int salesCount;
    public CashRegister(){
        purchase = 0;
        payment = 0;
        salesCount = 0;
        salesTotal = 0;
        history = "";
    }
    public void RecordPurchase(double VALUE){
        purchase += VALUE;
        salesTotal += VALUE;
        history += String.valueOf(VALUE) + "\n";
    }
    public void ReceivePayment(double MONEY){ payment += MONEY;}
    public double GiveChange(){
        double rest = payment-purchase;
        purchase = 0;
        payment = 0;
        history = "";
        salesCount += 1;
        return rest;
    }
    public void PrintReceipt(){
        System.out.println(history);
        System.out.println("Total: " + purchase);
    }
    public double GetSalesTotal(){return salesTotal;}
    public double GetSalesCount(){return salesCount;}
    public void Reset(){
        salesTotal = 0;
        salesCount = 0;
    }
}

public class Main {

    public static void main(String[] args) {
        CashRegister OBJECT = new CashRegister();
        // transaction #1
        OBJECT.RecordPurchase(30);
        OBJECT.RecordPurchase(10);
        OBJECT.ReceivePayment(50);
        System.out.println("Change for customer 1: " + OBJECT.GiveChange());
        System.out.println("Expected: 10.0");

        // transaction #2
        OBJECT.RecordPurchase(20);
        OBJECT.ReceivePayment(10);
        OBJECT.ReceivePayment(10);
        System.out.println("Change for customer 2: " + OBJECT.GiveChange());
        System.out.println("Expected: 0.0");

        // test new functionality
        System.out.println("Total amt of sales: " + OBJECT.GetSalesTotal());
        System.out.println("Expected: 60.0");
        System.out.println("Count of sales: " + OBJECT.GetSalesCount());
        System.out.println("Expected: 2");

        OBJECT.Reset();
        System.out.println("Total amt of sales: " + OBJECT.GetSalesTotal());
        System.out.println("Expected: 0.0");
        System.out.println("Count of sales: " + OBJECT.GetSalesCount());
        System.out.println("Expected: 0");

    }
}
```




11.

```
/*Creare una class Employee che contenga le informazioni
relative ad un impiegato.
Attributi:
- Name: privata di tipo String, rappresenta il nome dell'impiegato
- Salary: privata di tipo double, rappresenta lo stipendio
dell'impiegato
Costruttori e metodi:
- Employee: dovrà essere ridefinito in modo tale da accettare il
nome e lo stipendio dell'impiegato al fine di valorizzarli
- getName: ritornerà il nome dell'impiegato
- getSalary: ritornerà lo stipendio dell'impiegato

- raiseSalary: aumenta lo stipendio di una percentuale fornita
come parametro alla funzione
La classe EmployeeTester creerà un dipendente con nome e
stipendio. Quest'ultimo sarà successivamente aumentato del
10%.*/
package Main;
class Employee {
    private String name;
    private double salary;
    public Employee(String name,double salary){
        this.name = name;
        this.salary = salary;
    }
    public String GetName(){return name;}
    public double GetSalary(){return salary;}
    public void RaiseSalary(double interest){salary = (interest/100 * salary) + salary;}
}

public class Main {

    public static void main(String[] args) {
        Employee OBJECT = new Employee("Niccolò",1000);
        OBJECT.RaiseSalary(10);

    }
}
```



12.

```
/*Creare la classe Car che rappresenti un'automobile e il suo
consumo di carburante.
Attributi:
- Gas: privata e di tipo double, rappresenta i litri di carburante
rimanente
- Efficiency: privata e di tipo double, rappresenta il numero di km
per litro che la macchina può fare
Costruttore e metodi:
- Car: il costruttore dovrà essere ridefinito accettando un
parametro che rappresenti l'efficienza dell'auto in modo tale da
essere assegnato al corrispondente attributo. L'attributo gas
invece dovrà essere valorizzato a 0.
- addGas: incrementa l'attributo gas di un valore pari a quello del
parametro passato al metodo
- Drive: simula un viaggio in auto di una distanza pari a quella del
parametro passato alla funzione. In virtù di ciò, il carburante
dovrà essere diminuito in base all'efficienza dell'automobile.
- getGasInTank: ritorna l'ammontare di carburante residuo
La classe CarTester creerà un'auto con un'efficienza di
50km/litro. Dopodiché, effettuerà un rifornimento di 20 litri di
carburante e guiderà per 100km. A fine viaggio, dovrà essere
stampato il carburante residuo, accertandosi che sia pari a 18
litri.*/
package Main;
class Car {
    private double gas;
    private double efficiency;
    public Car(double efficiency){
        this.efficiency = efficiency;
        gas = 0;
    }
    public void AddGas(double VALUE){gas += VALUE;}
    public void Drive(double DISTANCE){gas -= (DISTANCE/efficiency);}
    public double GetGasInTank(){return gas;}
}

public class Main {

    public static void main(String[] args) {
        Car OBJECT = new Car(50);
        OBJECT.AddGas(20);
        OBJECT.Drive(100);
        System.out.println("Sono rimasti "+ OBJECT.GetGasInTank() + " litri di gas.");
    }
}
```



13.

```
/*Creare una classe Product che rappresenti le informazioni relative ad un prodotto.
Attributi:
- Name: privato e di tipo String, rappresenta il nome del prodotto - Price: privato e di tipo double, rappresenta il prezzo del prodotto
Costruttore:
- Product: ridefinito per accettare come parametri il nome ed il prezzo dell'oggetto, valorizzando i corrispondenti attributi della classe
Metodi:
- getName: ritorna il nome del prodotto
- getPrice: ritorna il prezzo del prodotto
- reducePrice: riduce il prezzo del prodotto di un ammontare pari al parametro fornito
La classe ProductTester creerà un prodotto con un nome e un prezzo pari a 30, riducendolo di 5. Successivamente, stamperà il suo nome ed il suo prezzo.
*/
package Main;
class Product {
    private String name;
    private double price;
    public Product(String name, double price){
        this.name = name;
        this.price = price;
    }
    public String GetName(){return name;}
    public double GetPrice(){return price;}
    public void ReducePrice(double VALUE){price -= VALUE;}
}

public class Main {

    public static void main(String[] args) {
        Product OBJECT = new Product("Matita",30);
        OBJECT.ReducePrice(5);
        System.out.println("Name: " + OBJECT.GetName() + "\nPrice: " + OBJECT.GetPrice() );
    }
}
```



14.

```
/*Creare la classe Letter che rappresenta il contenuto di una lettera.
Attributi:
- Sender: privata e di tipo String, rappresenta il mittente della lettera
- Recipient: privata e di tipo String, rappresenta il destinatario della lettera
- Body: privata e di tipo String, rappresenta il corpo della lettera
Costruttore:
- Letter: ridefinito per accettare, come parametri, un mittente e un destinatario al fine di valorizzare i corrispettivi attributi. Il corpo sarà automaticamente valorizzato con una
Metodi:
- addLine: concatena il contenuto dell'attributo body e la stringa passata come parametro, andando a capo.
- getText: ritorna il contenuto della lettera, con il seguente formato:
Caro {recipient},
{body}
Tuo,
{mittente}
Consiglio: per il concatenamento di due stringhe, guardare il metodo concat() della classe String.
La classe LetterPrinter creerà una lettera con un mittente ed un destinatario, costruendo il corpo tramite l'inserimento di tre righe a piacere. Stampare il contenuto della lettera pe
*/
package Main;
class Letter {
    private String sender;
    private String recipient;
    private String body;
    public Letter(String sender, String recipient){
        this.sender = sender;
        this.recipient = recipient;
        body = "";
    }
    public void AddLine(String LETTER){body = body.concat("\n"+LETTER);}
    public String getText(){
        return "Caro " + recipient + "," + body + "\nTuo,\n" + sender;
    }
}

public class Main {

    public static void main(String[] args) {
        Letter OBJECT = new Letter("Andrea", "Marco");
        OBJECT.AddLine("Se ni' mondo esistesse un po' di bene e ognun si considerasse suo fratello, ");
        OBJECT.AddLine("ci sarebbe meno pensieri e meno pene e il mondo ne sarebbe assai più bello.");
        System.out.println(OBJECT.getText());
    }
}
```



15.

```
/*Creare la classe Bug che rappresenti il movimento di un insetto lungo una linea.
Attributi:
- Position: privato di tipo intero, rappresenta la posizione dell'insetto
- Direction: privato di tipo intero, rappresenta la direzione dell'insetto, che può essere pari a 1 (insetto che si muove in avanti) o -1 (insetto che si muove all'indietro)
Costruttore:
- Bug: ridefinito per poter accettare il valore della posizione iniziale come parametro di input, assegnandolo al corrispondente attributo. La direzione invece sarà valorizzata a 1.
Metodi:
- Move: muove l'insetto, modificando la sua posizione in base al valore della posizione
- Turn: cambia direzione
- GetPosition: ritorna il valore corrente della posizione
La classe BugTester creerà un insetto in posizione 10. Successivamente, l'insetto effettuerà due movimenti, cambierà direzione e si muoverà un'ultima volta.
Stampare il valore della sua posizione finale, accertandosi che sia pari a 11.
*/
package Main;
class Bug {
    private int position;
    private int direction; //-1(indietro),1(avanti)
    public Bug(int position){
        this.position = position;
        direction = 1;
    }
    public void Move(){position += direction;}
    public void Turn(){direction = -direction;}
    public int GetPosition(){return position;}
}

public class Main {

    public static void main(String[] args) {
        Bug OBJECT = new Bug(10);
        OBJECT.Move();
        OBJECT.Move();
        OBJECT.Turn();
        OBJECT.Move();
        System.out.println("Position: " + OBJECT.GetPosition());
    }
}
```



16.

```
/*Creare una classe Moth che simuli il movimento di una libellula.
Attributi:
- Position: privata e di tipo double, rappresenta la posizione della libellula
Costruttore:
- Moth: ridefinito per accettare come parametro la posizione iniziale della libellula, assegnandola al corrispettiva parametro.
Metodi:
- moveToLight: muove la libellula verso la posizione della fonte di luce, fornita come parametro. La nuova posizione della libellula coinciderà con la metà della distanza tra la posizione attuale e quella della fonte di luce.
- getPosition: ritorna la posizione attuale della libellula

La classe MothTester creerà una libellula la cui posizione iniziale è pari a 10. Successivamente, effettuerà 3 movimenti:
- Il primo verso una fonte di luce in posizione 0
- Il secondo verso una fonte di luce in posizione 10
- Il terzo verso una fonte di luce in posizione 0
Ai fini del corretto funzionamento, la sequenza delle posizioni della libellula dovrà essere la seguente: 5, 7.5, 3.75
*/
package Main;
class Moth {
    private double position;
    public Moth(double position){this.position = position;}
    public void MovToLight(double LIGHT){position += (LIGHT-position)/2;}
    public double GetPosition(){return position;}
}

public class Main {

    public static void main(String[] args) {
        Moth OBJECT = new Moth(10);
        OBJECT.MovToLight(0);
        System.out.println("Position: " + OBJECT.GetPosition());
        OBJECT.MovToLight(10);
        System.out.println("Position: " + OBJECT.GetPosition());
        OBJECT.MovToLight(0);
        System.out.println("Position: " + OBJECT.GetPosition());
    }
}
```



17.

```
/*Scrivere una classe Rettangolo i cui oggetti rappresentano
rettangoli. Lo stato interno di un rettangolo è dato dai valori della
base e dell'altezza. Un rettangolo deve mettere a disposizione tre
operazioni: ridimensiona() che prende come parametri due nuovi
valori di base e altezza e aggiorna lo stato, perimetro() che restituisce
il perimetro e area() che restituisce l'area. Prevedere inoltre un
costruttore che inizializza base e altezza del rettangolo. Per testare la
classe, scrivere un programma TestRettangolo che crea tre rettangoli
(oggetti della classe Rettangolo) ne calcola la somma delle aree e la
somma dei perimetri (stampando i risultati), ridimensiona uno dei tre
rettangoli e ristampa le somme.
*/
package Main;
class Rettangolo{
    private double base;
    private double height;
    public Rettangolo(double base, double height){
        this.base = base;
        this.height = height;
    }
    public double Perimeter(){return (base+height)*2;}
    public double Area(){return base*height;}
    public void Resize(double base, double height){
        this.base = base;
        this.height = height;
    }
    public double GetBase(){return base;}
    public double GetHeight(){return height;}
}

public class Main {

    public static void main(String[] args) {
        Rettangolo REC_1 = new Rettangolo(6,12);
        Rettangolo REC_2 = new Rettangolo(7,14);
        Rettangolo REC_3 = new Rettangolo(8,16);
        System.out.println("Sum perimeters: " + (REC_1.Perimeter()+REC_2.Perimeter()+REC_3.Perimeter()));
        System.out.println("Sum areas: " + (REC_1.Area()+REC_2.Area()+REC_3.Area()));
        REC_1.Resize(5,10);
        System.out.println(REC_1.GetBase() + "\n" + REC_1.GetHeight());
    }
}
```



18.

```
/*Scrivere la classe Lampadina i cui oggetti rappresentano delle lampadine elettriche. Una lampadina può essere accesa, spenta o rotta, e mette a disposizione
due sole operazioni: stato() che restituisce una stringa che indica lo stato corrente della lampadina e click() che ne cambia lo stato da accesa a spenta o da spenta a accesa o la ro
- Una o più variabili d'istanza che descrivano opportunamente lo stato della lampadina
- Un opportuno costruttore
- I metodi previsti
Per testare la classe, scrivere un programma TestLampadina che crea un oggetto della classe Lampadina che ammetta
un numero massimo di click deciso dall'utente e poi iterativamente offre all'utente
la possibilità di invocare una delle due funzionalità (visualizzando l'esito dell'operazione, nel caso di stato())
o di terminare l'esecuzione.

*/
package Main;
import java.util.Scanner;

class Lampadina{
    private boolean lamp;
    private int clickMaximum;
    private int clickCurrent;
    public Lampadina(int clickMaximum){
        lamp = false;
        this.clickMaximum = clickMaximum;
        clickCurrent = 0;
    }
    public int Click(){
        lamp = !lamp;
        if (lamp) {System.out.println("La lampadina è accesa.");} else {System.out.println("La lampadina è spenta.");}
        clickCurrent++;
        if (clickCurrent == clickMaximum){return 1;} else {return 0;}
    }
}

public class Main {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.print("Inserisci click massimi:");
        Lampadina OBJECT = new Lampadina(scan.nextInt());
        int choice;
        int STATUS;
        do{
            System.out.println("\t\t MENU' SCELTA:");
            System.out.println("Vuoi cambiare lo stato della lampadina?\n" +
                "-Inserisci 1 cambiare lo stato della lampadina,\n" +
                "-Inserisci 0 se vuoi uscire dal programma,\n" +
                "Scelta:");
            choice= scan.nextInt();
            switch (choice) {
                case 0 -> System.out.println("Hai deciso di uscire dal programma");
                case 1 -> {
                    STATUS= OBJECT.Click();
                    if (STATUS == 1) {
                        System.out.println("Ops, la lampadina si è rotta!");
                        choice = 0;
                    }
                }
                default -> throw new IllegalStateException("Unexpected value: " + choice);
            }
        }while(choice!=0);
    }
}
```



19.

```
/*Scrivere la classe Interruttore i cui oggetti rappresentano degli interruttori. Ogni interruttore è collegato
a una lampadina (oggetto della classe Lampadina) e ne regola l'accensione e spegnimento. La classe deve contenere
tutte le variabili d'istanza i costruttori e i metodi ritenuti opportuni. Per testare la classe, scrivere un
programma TestInterruttore che crea due interruttori (oggetti della classe Interruttore) entrambi collegati
alla stessa lampadina (uno stesso oggetto della classe Lampadina) e poi, in maniera iterativa, offre all'utente
la possibilità di agire su uno dei due interruttori (visualizzando l'esito dell'operazione) o di terminare l'esecuzione.*/

package Main;
import java.util.Scanner;

class Lampadina{
    private boolean lamp;
    private int clickMaximum;
    private int clickCurrent;
    public Lampadina(int clickMaximum){
        lamp = false;
        this.clickMaximum = clickMaximum;
        clickCurrent = 0;
    }
    public int Click(){
        lamp = !lamp;
        if (lamp) {System.out.println("La lampadina è accesa.");} else {System.out.println("La lampadina è spenta.");}
        clickCurrent++;
        if (clickCurrent == clickMaximum){return 1;} else {return 0;}
    }
}
class Interruttore{
    Lampadina lampadina;
    private boolean button;
    public Interruttore(Lampadina lampadina){
        button = false;
        this.lampadina = lampadina;
    }
    public int Click(){
        button = !button;
        return lampadina.Click();
    }
}

public class Main {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.print("Inserisci click massimi:");
        Lampadina OBJECT = new Lampadina(scan.nextInt());
        Interruttore BUTTON_1 = new Interruttore(OBJECT);
        int choice;
        int STATUS;
        do{
            System.out.println("\t\t MENU' SCELTA:");
            System.out.println("Vuoi cambiare lo stato della lampadina?\n" +
                "-Inserisci 1 cambiare lo stato della lampadina,\n" +
                "-Inserisci 0 se vuoi uscire dal programma,\n" +
                "Scelta:");
            choice= scan.nextInt();
            switch (choice) {
                case 0 -> System.out.println("Hai deciso di uscire dal programma");
                case 1 -> {
                    STATUS= BUTTON_1.Click();
                    if (STATUS == 1) {
                        System.out.println("Ops, la lampadina si è rotta!");
                        choice = 0;
                    }
                }
                default -> throw new IllegalStateException("Unexpected value: " + choice);
            }
        }while(choice!=0);
    }
}
```



20.

```
/*Modificare la classe Lampadina dell'esercizio 20 in modo che tutte le lampadine condividano l'informazione sulla
presenza di corrente elettrica nell'impianto. Lo stato di una lampadina quando non c'e' corrente può essere solo
spento o rotto. Per testare la classe, estendere il programma TestLampadina dell'esercizio 20 in modo che offra
all'utente anche la possibilità di staccare o riattivare le corrente.*/

package Main;
import java.util.Scanner;

class Lampadina{
    private boolean lamp;
    private int clickMaximum;
    private int clickCurrent;
    private boolean state;
    public Lampadina(int clickMaximum){
        lamp = false;
        this.clickMaximum = clickMaximum;
        clickCurrent = 0;
        state = true;
    }
    public int Click(int state){
        if (state == 1){
            this.state = false;
            lamp = !lamp;
            if (lamp) {System.out.println("La lampadina è accesa.");} else {System.out.println("La lampadina è spenta.");}
            clickCurrent++;
            if (clickCurrent == clickMaximum){return 1;} else {return 0;}
        } else {
            this.state = true;
            System.out.println("Per mancanza di corrente la lampadina si è spenta");
            return 0;
        }
    }
}

class Interruttore{
    Lampadina lampadina;
    private boolean button;
    public Interruttore(Lampadina lampadina){
        button = false;
        this.lampadina = lampadina;
    }
    public int Click(int state){
        button = !button;
        return lampadina.Click(state);
    }
}

public class Main {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.print("Inserisci click massimi:");
        Lampadina OBJECT = new Lampadina(scan.nextInt());
        Interruttore BUTTON_1 = new Interruttore(OBJECT);
        int choice;
        int STATUS;
        do{
            System.out.println("\t\t MENU' SCELTA:");
            System.out.println("Vuoi cambiare lo stato della lampadina?\n" +
                "-Inserisci 1 cambiare lo stato della lampadina,\n" +
                "-Inserisci 0 se vuoi uscire dal programma,\n" +
                "Scelta:");
            choice= scan.nextInt();
            switch (choice) {
                case 0 -> System.out.println("Hai deciso di uscire dal programma");
                case 1 -> {
                    System.out.println("La corrente è accesa o spenta?(Inserisci 1 per accesa o 0 per spenta)");
                    STATUS= BUTTON_1.Click(scan.nextInt());
                    if (STATUS == 1) {
                        System.out.println("Ops, la lampadina si è rotta!");
                        choice = 0;
                    }
                }
                default -> throw new IllegalStateException("Unexpected value: " + choice);
            }
        }while(choice!=0);
    }
}
```