Esercizio 3

Scrivere un programma con i seguenti requisiti.
Utilizzare una **classe Persona** che dichiara le variabili nome, cognome, età. Si dichiari inoltre un metodo dettagli() che restituisce in una stringa le informazioni sulla persona in questione.

Utilizzare una classe Principale che, nel metodo main(), istanzia due oggetti chiamati persona1 e persona2 della classe Persona, inizializzando per ognuno di essi i relativi campi.
Dichiarare un terzo riferimento (persona3) che punti ad uno degli oggetti già istanziati. Controllare che effettivamente persona3 punti allo oggetto voluto, stampando i campi di persona3.

Oppure fare queste operazioni nel metodo main all'interno della classe Persona.

1 Soluzione Esercizio 3

```
public class Persona {

  private String nome;
  private String cognome;
  private int età;

  public Persona(String n, String c, int e){
  nome=n;
  cognome=c;
  età=e;
  }

  public String dettagli(){
  return nome + " " + cognome + " anni " + età;
  }
```

```
public static void main(String[] args) {
    Persona persona1 = new Persona("Laura","Giambruno",31);
    Persona persona2 = new Persona("Marco","Rossi",52);
    System.out.println("persona2 "+persona2.dettagli());
    Persona persona3 = persona1;
    System.out.println("persona3 "+persona3.dettagli());
}
}
```

2 Soluzione Esercizio 3

```
public class Persona1 {

public String nome;
public String cognome;
public int età;

public String dettagli() {
return nome + " " + cognome + "anni " + età;
}

public static void main (String args []) {
Persona1 persona1 = new Persona1();
Persona1 persona2 = new Persona1();
persona1.nome = "Mario";
persona1.cognome = "Rossi";
persona1.età = 30;
```

```
System.out.println("persona1 " +persona1.dettagli());
persona2.nome = "Giuseppe";
persona2.cognome = "Verdi";
persona2.età = 40;
System.out.println("persona2 "+persona2.dettagli());
Persona1 persona3 = persona1;
System.out.println("persona3 "+persona3.dettagli());
}
}
```

3 Soluzione Esercizio 3

```
public class Persona2{

private String nome;
private String cognome;
private int età;

public Persona2(String n,String c,int e){
nome=n;cognome=c;età=e;
}

public Persona2(String n,String c){
nome=n;cognome=c;età=20;
}

public String GetNome()
{return(nome);}
```

```
public String GetCognome()
{return(cognome);}

public int GetEta()
{return(età);}

public String Dettagli()
{return("Nome= "+nome+“ Cognome= "+cognome+“ Età= "+età);}

public static void main(String[] args) {
Persona2 persona1 = new Persona2("Mario","Rossi",27);
Persona2 persona2 = new Persona2("Marco","Gialli");
System.out.println(persona1.Dettagli());
System.out.println(persona2.Dettagli());
Persona2 persona3;
persona3 = persona1;
System.out.println(persona3.Dettagli());
}
}
```

Esercizio 4

Scrivere un programma con i seguenti requisiti.

Utilizzare una **classe Impiegato** che dichiara le variabili nome, cognome, salario. Si dichiari un metodo dettagli() che restituisce in una stringa le informazioni sulla persona in questione.

Si dichiari un metodo aumentasalario() che aumenti lo stipendio secondo una certa percentuale.

Utilizzare un metodo main nella classe Impiegato oppure una classe Principale oppure che, nel metodo main(), istanzia due oggetti chiamati impiegato1 e impiegato2 della classe Impiegato, inizializzando per ognuno di essi i relativi campi. Si aumenti il salario di impiegato1 del 10% e poi si verifichi se il salario di impiegato 1 è maggiore di quello di impegato2.

Soluzione Esercizio 4

```
public class Impiegato {
private String nome;
private String cognome;
private double salario;

  public Impiegato(String n, String c, double s){
  nome=n;
  cognome=c;
  salario=s;  }

  public void incrementasalario(int percentuale){
  salario = salario + ((salario * percentuale) / 100);  }

   public String dettagli()
{return("Nome "+nome+ " Cognome "+cognome+" Salario
  "+salario);}
```

```
/*      public void dettagli(){
      System.out.println("Nome = " + nome + " - Salario = " + salario);
       }
*/

public static void main(String[] args) {
Impiegato persona1 = new Impiegato(“Giuseppe",“Verdi",1000);
    Impiegato persona2 = new Impiegato("Marco","Rossi",1050);
    System.out.println(persona1.dettagli());
    //persona1.dettagli();
    persona1.incrementasalario(10);
    //persona1.dettagli();
    System.out.println(persona1.dettagli());
    if (persona1.salario > persona2.salario) {
    System.out.println(persona1.nome + " è più ricca di " + persona2.nome);
    }
    else {
    System.out.println(persona1.nome + " è meno ricca di " +
persona2.nome);} }}
```

Precisazioni sui metodi

Invocazione di un metodo e passaggio dei parametri

Ad ogni invocazione, un nuovo spazio di memoria viene allocato per ospitare i valori che il chiamante vuole passare al metodo

Quando invocate il metodo, il valore passato dal chiamante viene copiato sul parametro corrispondente

Il parametro è locale al metodo e i cambiamenti effettuati sul parametro non hanno effetto sugli argomenti passati dal chiamante

Però, se passate un riferimento ad un oggetto, copiate il suo indirizzo e le modifiche apportate dal codice presente nel metodo saranno visibili al chiamante

Ricordate sempre la distinzione tra tipi primitivi e oggetti:

Tipi primitivi: copia del valore
Oggetti: copia del riferimento, accesso allo stesso oggetto

Precisazioni sui metodi

```
public class TestPassaggioParametri {
    public static void incrementaContatore(Counter counter){
        counter.inc();
    }

    public static void main(String args[]){
        Counter counter = new Counter();
        incrementaContatore(counter);
        System.out.println("Value = " + counter.getValue());
    }
}
```

In questo caso, counter è un oggetto di tipo Counter

Quando invocate il metodo incrementaContatore passate una copia del valore attuale di counter (che però è un indirizzo)

Il metodo incrementaContatore richiama inc sull'oggetto

Il chiamante stampa il valore della variabile e trova un valore pari a 1
Questo perché è stata passata una copia del riferimento/indirizzo

Precisazioni sui metodi

```
public class TestPassaggioParametri {
    public static void incrementaContatore(int counter){
        counter++;
    }

    public static void main(String args[]){
        int counter = 0;
        incrementaContatore(counter);
        System.out.println("Value = " + counter);
    }
}
```

In questo caso, counter è una variabile di tipo primitivo

Quando invocate il metodo incrementaContatore, passate una copia del valore attuale (quindi 0)

Il metodo incrementaContatore modifica il parametro portandolo in 1

Il chiamante stampa il valore della variabile, che però vale ancora 0
Questo perché è stata passata una copia del valore

Precisazioni sui metodi

Vediamo … e questo cosa stampa?

```
public class TestPassaggioParametri {
    public static void incrementaContatore(Counter counter){
        Counter newCounter = new Counter();
        newCounter.inc();
        counter = newCounter;
        System.out.println("Counter = " + counter.getValue());
    }


    public static void main(String args[]){
        Counter counter = new Counter();
        incrementaContatore(counter);
        System.out.println("Counter = " + counter.getValue());
    }
}
```

Precisazioni sui metodi

Simili ragionamenti si applicano ai valori restituiti dal metodo

I metodi utilizzati fino ad oggi restituivano valori di tipi di dato primitivi
Ovviamente, un metodo può restituire anche un oggetto: basta dichiarare la firma del metodo in modo corretto

Ad esempio, se volessimo aggiungere all'oggetto Counter un metodo che clona il contatore su cui è invocato, scriviamo:

```
public Counter cloneCounter(){
    return new Counter(this.val);
}
```

Attenzione però: in questo caso, viene creato un nuovo oggetto ad ogni invocazione e viene restituito il corrispettivo indirizzo

Precisazioni sui metodi

Cosa succede se ritorniamo un riferimento che appartiene all'istanza corrente?

Se l'oggetto attuale contiene un riferimento e restituite quest'ultimo, non viene creato un nuovo oggetto uguale ma semplicemente ritornate l'indirizzo

Rendete visibile all'esterno l'oggetto appartenente all'istanza

Il chiamante può quindi modificare arbitrariamente l'oggetto contenuto (rompete l'incapsulamento)

Sovraccarico dei metodi

All'interno di una classe, possono esistere più metodi con lo stesso nome se e solo se o hanno un numero di parametri differente o hanno lo stesso numero di parametri ma di tipi differenti

Il tipo del valore di ritorno non influisce come discriminante. Di conseguenza, il codice riportato a destra non compila
All'invocazione, viene richiamato il metodo che fa match con gli argomenti forniti dal chiamante

```
public class Counter{
        public void inc(){
                ...
        }
        public void inc(int x){
                ...
        }
        ...
}
```

```
public class Counter{
        public int inc(){
                ...
        }
        public void inc(){
                ...
        }
        ...
}
```

Sovraccarico dei costruttori

Similmente, potete definire differenti costruttori purché rispettiate le stesse regole

```
public   class   Counter{
                 public   Counter(){
                                  this.val=0;
                 }
                 public   Counter(int   val){
                                  this.val=val;
                 }
                 ...
}
```