Lezione 3 – Esercitazione

prof. Marcello Sette
mailto://marcello.sette@gmail.com
http://sette.dnsalias.org

Esercizio 1

Si consideri il seguente programma Pascal:

```
program esercizio1(input, output);
  var a,b,c: integer;

  procedure p1;
    var a: integer;
  begin
    a:=1; b:=1; c:=1;
  end;

  procedure p2;
    var b: integer;

    procedure p3;
      var a,c: integer;
    begin
      a:=3; b:=3; c:=3;
      p1;
      writeln(a, b, c)
    end;

  begin
    a:=2; b:=2; c:=3;
    p3;
    writeln(a, b, c)
  end;

begin
  a:=0; b:=0; c:=0;
  p2;
  writeln(a, b, c)
end.
```

La sequenza delle procedure attivate è: (main, p2, p3, p1).

- Usando la regola di propagazione dei legami in ambito statico (standard Pascal):
  1. rappresentare graficamente lo stack di attivazione quando esso ha altezza massima;
  2. determinare, se possibile, quali nomi (variabili e procedure) sono accessibili in ciascuna procedura;
  3. determinare l'output del programma.
- Usando la regola di propagazione dei legami in ambito dinamico:
  1. rappresentare graficamente lo stack di attivazione quando esso ha altezza massima;
  2. determinare, se possibile, quali nomi (variabili e procedure) sono accessibili in ciascuna procedura;
  3. determinare l'output del programma.

Esercizio 2

Dato il seguente programma Pascal:

```
program esercizio2 (input, output);
  var a,b: integer;

  procedure p1;
  begin
    b:= 4
  end;

  procedure p2;
    var b: integer;
  begin
    a:= 50;
    p1
  end;

begin
  b:= 40;
  p2;
  a:= b;
  writeln(a)
end.
```

Determinare l'output,

a) nel caso in cui vi sia ambito statico di validità dei legami;

b) nel caso in cui vi sia ambito dinamico di validità dei legami.

Esercizio 3

Si consideri il codice seguente:

```
program esercizio3 (input, output);
  var a, b, c: integer;

  procedure p1;
    var b: integer;
  begin
    b:= 4;
    a:= a+b;
      {B}
  end;

  procedure p2;
    var b,c: integer;

    procedure p3;
      var a: integer;
    begin
      a:= b;
      c:= 14;
      p1
    end;

  begin
    b:= 10;
    c:= 20;
       {A}
    p3
  end;

  procedure p4;
  begin
    p2
  end;

begin
  a:=1; b:=2; c:=3;
  p4
end.
```

a) Determinare i nomi (variabili e procedure) accessibili da `p2` al punto {`A`} usando ambito statico di validità dei legami. Scrivere anche i valori delle variabili.

b) Determinare gli identificatori (variabili e procedure) accessibili da `p1` al punto {`B`} usando ambito dinamico di validità dei legami. Scrivere anche i valori delle variabili.

Lezione 4 – Esercitazione

prof. Marcello Sette
mailto://marcello.sette@gmail.com
http://sette.dnsalias.org

Esercizio 1

Si consideri il codice seguente:

```
program esercizio1 (input, output);
  var a,b,c,d: integer;

  procedure p1;
    var d,x: integer;
  begin
    {QUI}
  end;

  procedure p2;
    var b,c: integer;
    procedure p3;
      var b,x: integer;
    begin
      a:=1; p2; x:=a; p1
    end;
  begin
    if a=0 then p3 else p1
  end;

begin
a:=0; p2; p1
end.
```

Supponendo una propagazione di legami in ambito statico, rappresentare lo stack di attivazione nel momento in cui `p1` è chiamato la prima volta, nel punto marcato {`QUI`}.

Esercizio 2

Si consideri il codice seguente:

```
program esercizio2 (input, output);
  var a,b,c: integer;

  procedure p1 ([MODE] a,b: integer);
  begin
    a:= a*b;
    if (c/b)=a then a:=0 else a:=100
  end;

  procedure p2 ([MODE] a,b: integer);
  begin
    a:= a-b;
    if a=c then p1(b,a) else p1(a,b)
  end;

begin
  a:=1; b:=5; c:=10;
  p2(c,b);
  writeln(a, b, c)
end.
```

Supponendo una propagazione di legami in ambito statico, valutare l'uscita del programma quando, per entrambe le occorrenze di `[MODE]`, viene usato uno dei seguenti meccanismi di passaggio dei parametri:

1. IN realizzato per riferimento;
2. IN realizzato per copia;
3. OUT realizzato per riferimento;
4. OUT realizzato per copia;
5. IN OUT realizzato per riferimento;
6. IN OUT realizzato per copia.

Esercizio 3

Si consideri il seguente codice Pascal:

```
program esercizio3 (input, output);
  var limit: integer;

  function sommatoria ([MODE] lim: integer): integer;
    var s: integer;
  begin
    s:= 0;
    while lim > 0 do
    begin
      s:= s + limit;
      limit:= limit - 1
    end;
    sommatoria:= s
  end;

begin
  limit:= 6;
  writeln('Sommatoria da 1 a 6: ', sommatoria(limit))
end.
```

Quale meccanismo di passaggio dei parametri provocherebbe un ciclo infinito? E quale invece farebbe funzionare il programma?

Esercizio 4

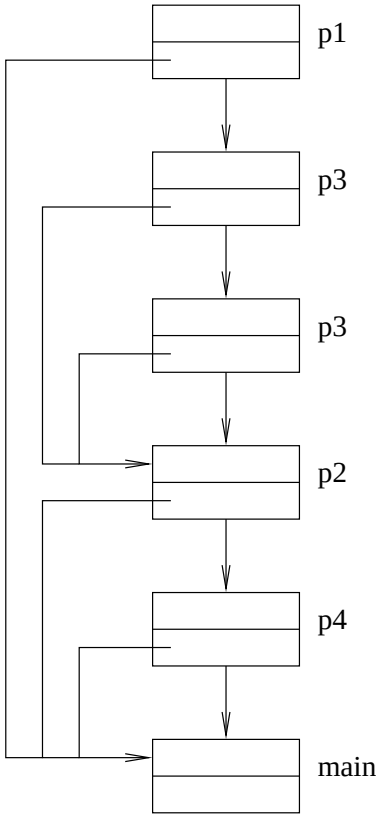
Si consideri il seguente programma:

```
program esercizio4 (input, output);
  var a,b,c: integer;
      d: boolean;

  procedure p1 (var q:boolean; var r, s: integer);
  begin
    if d then r:=100 else r:= 200;
    s:= s/a
  end;

  procedure p2 (var x, y: integer; var z: boolean);
  begin
    x:=15; y:= x+a; z:= (x<a);
    p1(z,y,x);
    z:= (x<a)
  end

begin
  a:= -1; b:= -1; c:= -1; d:= true;
  p2(a,b,d)
end.
```

Supponendo una propagazione di legami in ambito statico ed usando il meccanismo di passaggio dei parametri standard del Pascal (`var`: parametri IN OUT realizzati per riferimento; non `var`: parametri IN realizzati per copia), determinare il valore delle variabili locali al main alla terminazione del programma.

Esercizio 5

Determinare la struttura di annidamento di un programma, il cui stack di attivazione ad un certo istante della sua esecuzione è quello mostrata in figura:



Esercizio 6

Scrivere un programma che determini se, nel linguaggio usato, i parametri IN siano realizzati per riferimento o per copia.

Esercizio 7

Scrivere un programma che determini se, nel linguaggio usato, i parametri OUT siano realizzati per riferimento o per copia.

Esercizio 8

Scrivere un programma che determini se, nel linguaggio usato, i parametri IN OUT siano realizzati per riferimento o per copia.

Esercizio 9

Sapendo che il linguaggio in uso realizza i parametri OUT e IN OUT per copia, scrivere un programma che determini se tali parametri sono copiati da sinistra a destra oppure da destra a sinistra.

Esercizio 10

Scrivere un programma che determini, nel linguaggio in uso, se i parametri di ritorno VRP da una procedura sono parametri OUT oppure IN OUT.

Esercizio 11

Sia dato il programma:

```
program Exam (input, output);
  var a, b, c: integer;

  procedure p1([MODE1] a,c:integer);

    procedure p2([MODE2] a,b:integer);
      begin
        if a < b then
          p2(b, a)
        else
          b:= 1;
        c:= 1
      end;

    begin
      a:= 2; b:= 2;
      p2(b, c);
      writeln(a, b, c)
    end;

  procedure p3([MODE3] a,b:integer);
    begin
      a:= 3; c:= 3;
      p1(c, b);
      writeln(a, b, c)
    end;

  begin
    a:= 4; b:= 4; c:= 4;
    p3(b, c);
    writeln(a, b, c);
    readln
  end.
```

Versione semplice

Si supponga una propagazione di legami in ambito statico e che i possibili meccanismi di passaggio di parametri siano:

- IN realizzato per copia, oppure
- INOUT realizzato per riferimento;

Per ognuna delle possibili scelte di $\texttt{MODE}_i, i = 1, \ldots, 3$ tra quelle specificate in precedenza (8 scelte possibili, per un totale di 8 esercizi diversi):

1. rappresentare lo stack di esecuzione quendo esso ha altezza massima;
2. specificare le variabili locali di ogni procedura e quelle propagate da altre procedure (ambiente locale e non locale);
3. discutere il comportamento del programma ed i valori intermedi e finali di `a`, `b`, `c`.

Soluzioni degli 8 esercizi P_i

P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
2 2 1 2 3 3 4 2 3	2 1 1 3 4 3 4 1 3	2 1 1 1 3 3 4 1 3	2 2 1 3 1 2 4 2 2	1 2 1 2 1 1 4 2 1	2 1 1 3 1 2 4 1 2	1 2 1 2 1 1 4 2 1	2 2 1 3 4 3 4 2 3

Versione per masochisti

Si supponga una propagazione di legami in ambito statico e che i possibili meccanismi di passaggio di parametri siano:

- IN realizzato per copia, oppure
- INOUT realizzato per riferimento, oppure
- INOUT realizzato per copia.

Per ognuna delle possibili scelte di $\texttt{MODE}_i, i = 1, \ldots, 3$ tra quelle specificate in precedenza (27 scelte possibili, per un totale di 19 esercizi diversi, oltre quelli della sottosezione precedente):

1. rappresentare lo stack di esecuzione quando esso ha altezza massima;
2. specificare le variabili locali di ogni procedura e quelle propagate da altre procedure (ambiente locale e non locale);
3. discutere il comportamento del programma ed i valori intermedi e finali di `a`, `b`, `c`.

Lezione 7 – Esercitazione

prof. Marcello Sette
mailto://marcello.sette@gmail.com
http://sette.dnsalias.org

Esercizio 1

Determinare gli errori nei seguenti sorgenti Java.

1. **File:** `Test1.java`

```
public class Test1 {
  public static void main(String[] args) {
    System.out.println("Va tutto bene?");
  }
}
public class TestAnother1 {
  public static void main(String[] args) {
    System.out.println("Va tutto bene?");
  }
}
```

2. **File:** `Test2.java`

```
public class Testing2 {
  public static void main(String[] args) {
    System.out.println("Va tutto bene?");
  }
}
```

3. **File:** `Test3.java`

```
public class Test3 {
  public static void main(String args) {
    System.out.println("Va tutto bene?");
  }
}
```

4. **File:** `Test4.java`

```
public class Test4 {
  public void main(String[] args) {
    System.out.println("Va tutto bene?");
  }
}
```

Esercizio 2

In questo esercizio si vuole mostrare l'uso dell'incapsulazione. Si dovrà creare una classe `Veicolo` che permetta al programma di funzionare.

Sono proposte due versioni diverse dello stesso esercizio.

Versione 1

File: `TestVeicolo.java`

```
public class TestVeicolo {
  public static void main(String[] args) {

    // Crea un veicolo che possa caricare fino a 10000 kg
    System.out.println(
      "Creazione di un veicolo con 10000 kg di carico massimo.");
    Veicolo veicolo = new Veicolo(10000.0);

    // Aggiungi alcune scatole
    System.out.println("Aggiunta della scatola #1 (500kg)");
    veicolo.carico = veicolo.carico + 500.0;

    System.out.println("Aggiunta della scatola #2 (250kg)");
    veicolo.carico = veicolo.carico + 250.0;

    System.out.println("Aggiunta della scatola #3 (5000kg)");
    veicolo.carico = veicolo.carico + 5000.0;

    System.out.println("Aggiunta della scatola #4 (4000kg)");
    veicolo.carico = veicolo.carico + 4000.0;

    System.out.println("Aggiunta della scatola #5 (300kg)");
    veicolo.carico = veicolo.carico + 300.0;

    // Stampa il carico finale del veicolo
    System.out.println(
      "Carico finale del veicolo: " + veicolo.getCarico() + " kg");
  }
}
```

Versione 2

File: `TestVeicolo.java`

```
public class TestVeicolo {
  public static void main(String[] args) {

    // Crea un veicolo che possa caricare fino a 10000 kg
    System.out.println(
      "Creazione di un veicolo con 10000 kg di carico massimo.");
    Veicolo veicolo = new Veicolo(10000.0);

    // Aggiungi alcune scatole
    System.out.println(
      "Aggiunta della scatola #1 (500kg) : " + veicolo.addScatola(500.0));
    System.out.println(
      "Aggiunta della scatola #2 (250kg) : " + veicolo.addScatola(250.0));
    System.out.println(
      "Aggiunta della scatola #3 (5000kg) : " + veicolo.addScatola(5000.0));
    System.out.println(
      "Aggiunta della scatola #4 (4000kg) : " + veicolo.addScatola(4000.0));
    System.out.println(
      "Aggiunta della scatola #5 (300kg) : " + veicolo.addScatola(300.0));

    // Stampa il carico finale del veicolo
    System.out.println(
      "Carico finale del veicolo: " + veicolo.getCarico() + " kg");
  }
}
```

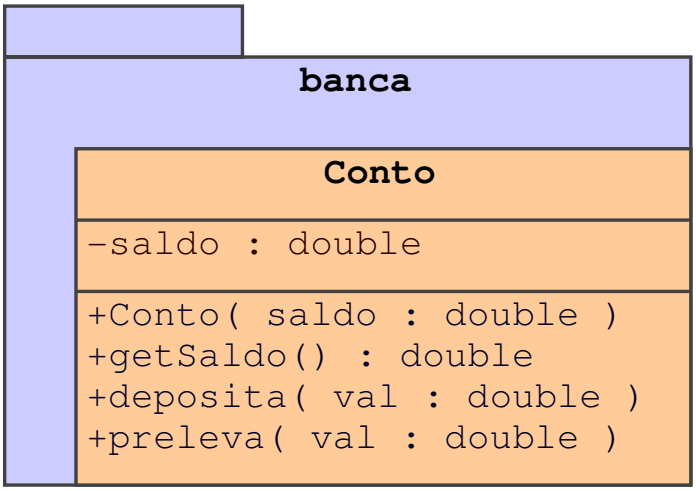
Esercizio 3

Obiettivo

Questo esercizio introdurrà la costruzione di un progetto di gestione bancaria che sarà riproposto e sviluppato ulteriormente nelle esercitazioni successive.

Il progetto consisterà (eventualmente) in una banca con alcuni clienti titolari di vari conti e destinatari di estratti conto.

In questo esercizio verrà chiesto di creare una semplice versione della classe `Conto` all'interno di un pacchetto `banca`. Nel pacchetto di default viene fornita una classe di test, `TestConto`, che crea un singolo conto, inizializza il saldo, esegue alcune semplici transazioni, mostra infine il saldo finale.



Traccia

1. Posizionarsi all'interno della cartella corrispondente al pacchetto di default. In essa è presente il file `TestBanca.java`.
2. Creare la cartella corrispondente al pacchetto (`banca`).
3. Nella cartella `banca` creare la classe `Conto` all'interno del file `Conto.java`. La classe deve realizzare il modello del diagramma UML precedente.
   - Dichiarare un attributo privato: `saldo`.
   - Dichiarare un costruttore pubblico con parametro: `saldo`.
   - Dichiarare un metodo pubblico, `getSaldo`, che restituisca il valore del saldo corrente.
   - Dichiarare un metodo pubblico, `deposita`, che aggiunga una somma al saldo corrente.
   - Dichiarare un metodo pubblico, `preleva`, che sottragga una somma dal saldo corrente.
4. Nella cartella di default, compilare il file `TestBanca.java`. Verranno compilate a cascata tutte le classi usate nel programma. Nella cartella `banca`, controllare che sia stato compilato il file `Conto.java` in `Conto.class`. Il comando è

   ```
   $ javac -d . TestBanca.java
   ```

5. Eseguire la classe `TestBanca`. Deve essere prodotto l'output seguente:

   ```
   $ java TestBanca
   Creazione di un conto con un saldo di 500.00
   Prelievo 150.00
   Deposito 22.50
   Prelievo 47.62
   Il conto ha attualmente un saldo di 324.88
   ```

Lezione 8 – Esercitazione

prof. Marcello Sette
mailto://marcello.sette@gmail.com
http://sette.dnsalias.org

Esercizio 1

Sulla creazione e uso degli oggetti.

Data la classe (file: `MioPunto.java`):

```
public class MioPunto {
  public int x;
  public int y;

  public String toString() {
    return ("[" + x + "," + y + "]);
  }
}
```

1. Costruire, nel file `TestMioPunto.java`, una classe `TestMioPunto` in cui sia presente il solo metodo `main` (attenzione alla segnatura corretta).
2. Nel metodo `main` si dichiarino e costruiscano due punti: `inizio` e `fine`.
3. Si assegnino ad `inizio` le coordinate $(10, 10)$ e a `fine` le coordinate $(20, 30)$.
4. Si stampino le rappresentazioni degli oggetti (il metodo `toString` non deve essere citato esplicitamente).
5. Si dichiari ed assegni un `altro` riferimento a `fine`.
6. Si stampino le rappresentazioni di `altro` e `fine`.
7. Si assegnino ad `altro` le coordinate $(12, 23)$.
8. Si stampino le rappresentazioni di `altro` e `fine`.
9. L'output complessivo del programma dovrà essere:

   ```
   $ java TestMioPunto
   Punto iniziale: [10,10]
   Punto finale:   [20,30]
   ```

```
Un altro punto: [20,30]
Punto finale:   [20,30]

Un altro punto: [12,23]
Punto finale:   [12,23]
```

Esercizio 2

In questo esercizio verrà ampliato il pacchetto `banca` precedentemente definito.

1. Posizionarsi nel pacchetto di default fornito dal docente. In esso è presente una nuova versione della classe `TestBanca`.
2. Ricopiare nel pacchetto di default la propria cartella `banca` nello stato in cui si trovava alla fine dell'esercizio precedente.
3. Aggiungere al pacchetto `banca` una classe `Cliente` (un cliente ha un solo conto in banca) che permetta a `TestBanca` di funzionare.

Lezione 9 – Esercitazione

prof. Marcello Sette
mailto://marcello.sette@gmail.com
http://sette.dnsalias.org

Esercizio 1

In questo esercizio verranno modificati i metodi `deposita` e `preleva` nella classe `Conto` del pacchetto `banca` precedentemente definito, in modo che restituiscano un valore booleano che indichi se la transazione ha avuto successo.

1. Posizionarsi nel pacchetto di default fornito dal docente. In esso è presente una nuova versione della classe `TestBanca`.
2. Ricopiare nel pacchetto di default la propria cartella `banca` nello stato in cui si trovava alla fine dell'esercizio precedente.
3. Modificare i metodi `preleva` e `deposita` nella classe Conto, in modo che la classe `TestBanca` possa funzionare.

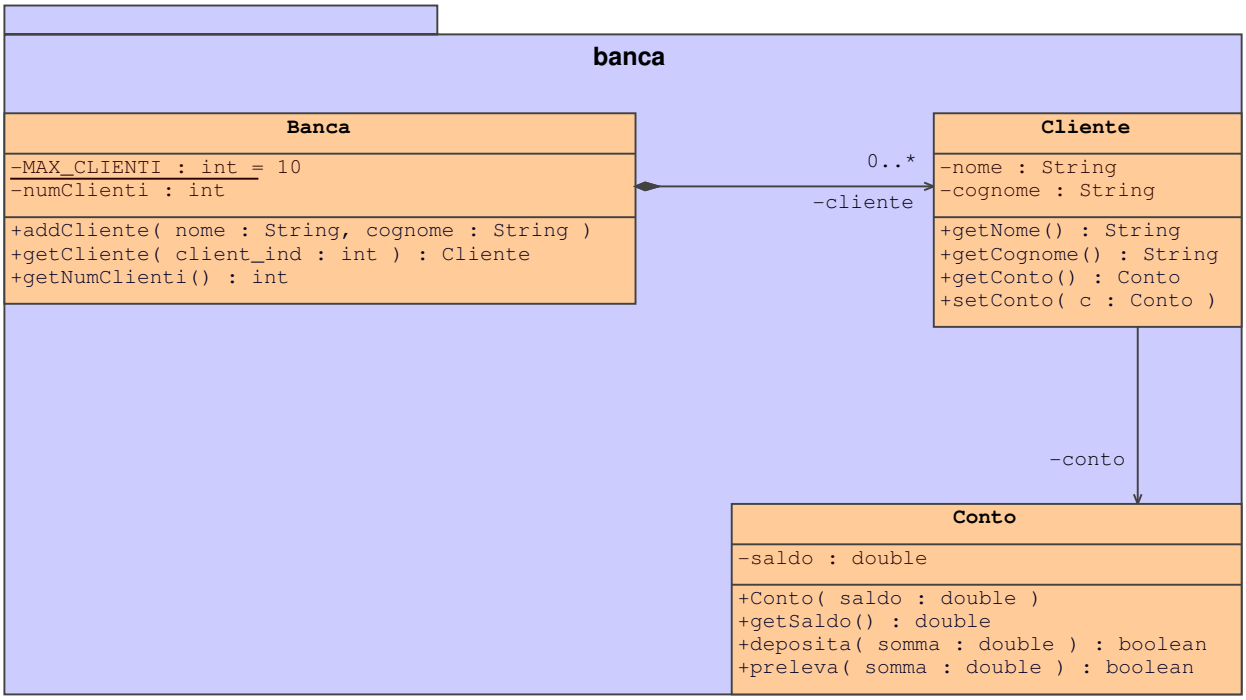
Lezione 10 – Esercitazione

prof. Marcello Sette
mailto://marcello.sette@gmail.com
http://sette.dnsalias.org

Esercizio 1

In questo esercizio verranno usati gli array per realizzare la molteplicità nella relazione (composizione) tra una banca e i propri clienti.

1. Posizionarsi nel pacchetto di default fornito dal docente. In esso è presente una nuova versione della classe `TestBanca`.
2. Ricopiare nel pacchetto di default la propria cartella `banca` nello stato in cui si trovava alla fine dell'esercizio precedente.
3. Fare in modo che la classe `TestBanca`, nel pacchetto di default, possa funzionare, aggiungendo una opportuna classe `Banca` che rispetti il diagramma seguente:



Lezione 11 – Esercitazione

prof. Marcello Sette
`mailto://marcello.sette@gmail.com`
`http://sette.dnsalias.org`

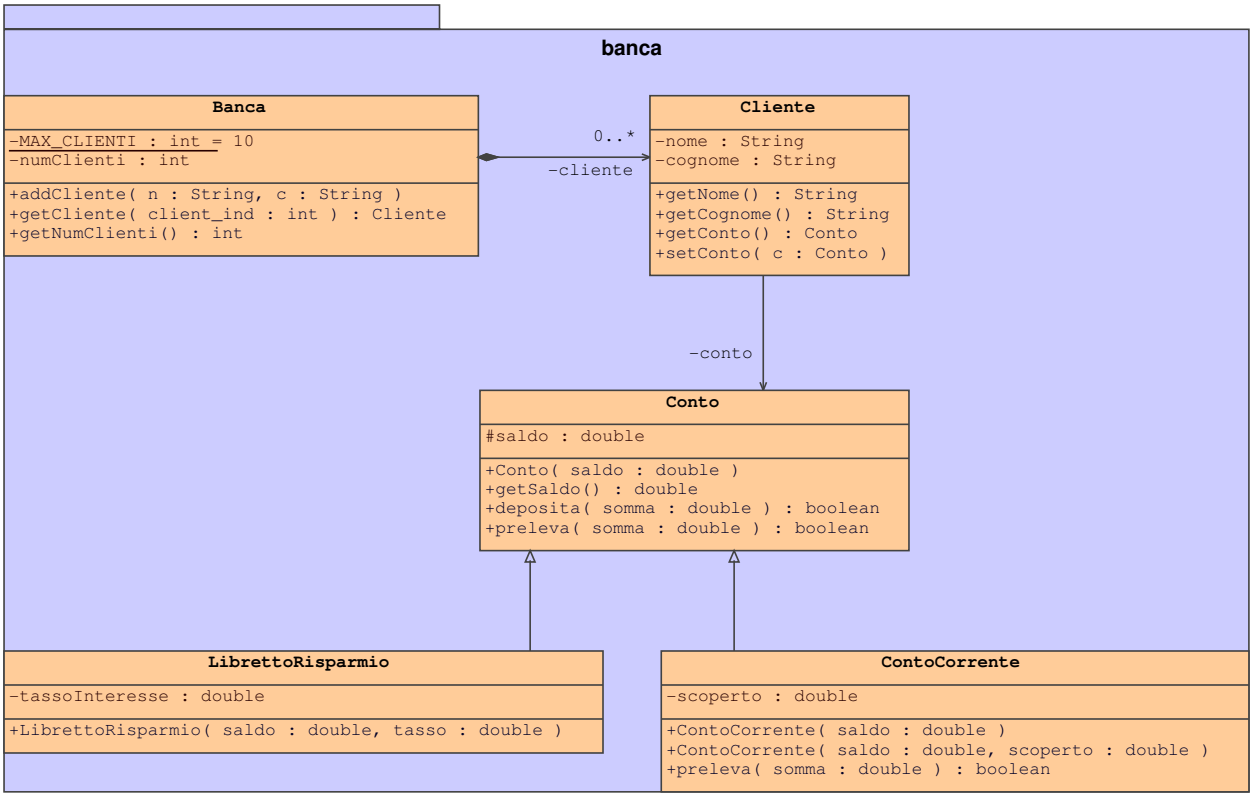
Esercizio 1

Obbiettivo

In questo esercizio verranno create due sottoclassi della classe `Conto` nel progetto `banca`: `ContoCorrente` e `LibrettoRisparmio`.

Si dovrà sovrapporre il metodo `preleva` in `ContoCorrente` ed usare `super` per invocare il costruttore della superclasse.

Traccia



1. Posizionarsi nel pacchetto di default fornito dal docente. In esso è presente una nuova versione della classe `TestBanca`.

2. Ricopiare nel pacchetto di default la propria cartella `banca` nello stato in cui si trovava alla fine dell'esercizio precedente.

3. Modificare la classe `Conto`: l'attributo `saldo` è ora `protected`.

4. Realizzare le due classi `ContoCorrente` e `LibrettoRisparmio` come rappresentate nel precedente diagramma UML.

5. Nella classe `ContoCorrente` sovrapporre il metodo `preleva` della superclasse con uno che:

   (a) Se il saldo corrente è adeguato a coprire il prelievo, allora procedi come al solito.

   (b) Altrimenti, se è autorizzato lo scoperto, allora modifica il saldo fino ad un minimo valore negativo di modulo uguale allo scoperto.

   (c) Se la somma è superiore al saldo più lo scoperto, allora la transazione fallisce e il saldo resta invariato.

Test del codice

Compilare ed eseguire `TestBanca`. L'output dovrebbe essere:

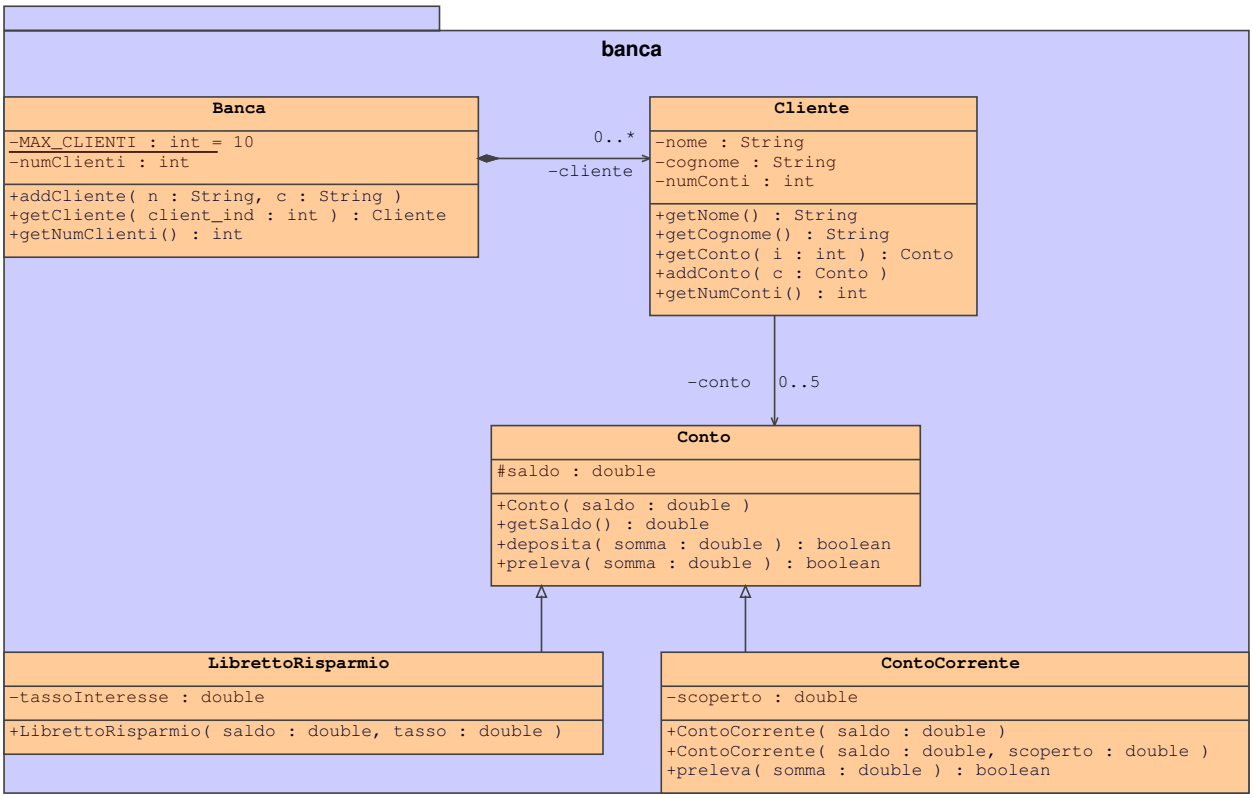
```
Creazione del cliente Carla Rossi.
Creazione del suo Libretto di Risparmio con saldo iniziale 500.00 e interesse 3%.
Creazione del cliente Anna Bruni.
Creazione del suo Conto Corrente con saldo iniziale 500.00 e senza tolleranza di scoperto.
Creazione del cliente Raul Falchi.
Creazione del suo Conto Corrente con saldo iniziale 500.00 e massimo scoperto 500.00.
Creazione del cliente Vale Bova.
Vale condivide il conto di suo marito Raul.

Test del Libretto Risparmio di Carla Rossi.
Prelievo di 150.00: true
Deposito di  22.50: true
Prelievo di  47.62: true
Prelievo di 400.00: false
Cliente Carla Rossi ha un saldo di 324.88

Test del Conto Corrente di Anna Bruni.
Prelievo di 150.00: true
Deposito di  22.50: true
Prelievo di  47.62: true
Prelievo di 400.00: false
Cliente Anna Bruni ha un saldo di 324.88

Test del Conto Corrente di Raul Falchi.
Prelievo di 150.00: true
Deposito di  22.50: true
Prelievo di  47.62: true
Prelievo di 400.00: true
Cliente Raul Falchi ha un saldo di -75.12

Test del ContoCorrente di Vale Bova.
Deposito di 150.00: true
Prelievo di 750.00: false
Cliente Vale Bova ha un saldo di 74.88
```

Esercizio 2

Obbiettivo

Come ulteriore continuazione del progetto di gestione bancaria, per ciascun cliente della banca in questo esercizio verrà creata una collezione eterogenea di conti (max. 5), cioè lo stesso cliente può essere titolare di conti di diverso tipo.

Traccia



1. Posizionarsi nel pacchetto di default fornito dal docente. In esso è presente una nuova, ma incompleta, versione della classe `TestBanca`.
2. Ricopiare nel pacchetto di default la propria cartella `banca` nello stato in cui si trovava alla fine dell'esercizio precedente.
3. Modificare la classe `Cliente` così come specificato nel precedente diagramma UML.
4. Completare la classe `TestBanca` fornita dal docente.

Lezione 12 – Esercitazione

prof. Marcello Sette
mailto://marcello.sette@gmail.com
http://sette.dnsalias.org

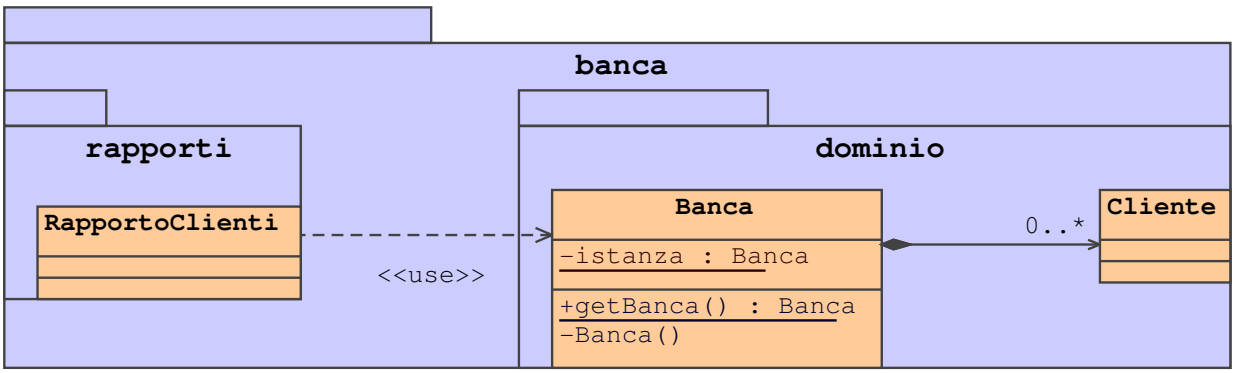
Esercizio 1

Obbiettivo

In questo esercizio verrà modificata la classe `Banca` in modo che realizzi un Singoletto.

Nota: a questo punto il progetto ha bisogno di una gerarchia di pacchetti più articolata, poiché dovremo creare una classe `RapportoClienti` che appartiene al pacchetto `banca.rapporti`; inoltre, le classi del dominio precedente dovranno essere poste nel pacchetto `banca.dominio` e si dovranno modificare le dichiarazioni di pacchetto in ognuno dei file precedenti.

Traccia



1. Nei precedenti esercizi, i rapporti di stampa erano immersi nel metodo `main` della classe `TestBanca`. In questo esercizio il codice è stato estratto in una classe `RapportoClienti` posta nel pacchetto `banca.rapporti`.

   Posizionarsi nel pacchetto di default fornito dal docente. In esso è presente una nuova versione della classe `TestBanca` ed una cartella `banca`.

   Nella cartella `banca` è presente una sottocartella `rapporti`, nella quale è data una nuova classe `RapportoClienti`.

2. Rinominare la propria cartella `banca` (così come si trovava alla fine dell'esercizio precedente) in `dominio` e ricopiarla nella cartella `banca` fornita dal docente.

3. Correggere in tutte le classi della cartella `dominio` il nome del pacchetto a cui esse appartengono (ora è `banca.dominio`).

4. Modificare la classe `Banca` in modo da realizzare un Singoletto come rappresentato nel precedente diagramma UML.

Test del codice

Compilare ed eseguire `TestBanca`. L'output dovrebbe essere:

```
            RAPPORTO CLIENTI
            ================

Cliente: Carla Rossi
    Saldo del libretto di risparmio: 500.0
    Saldo del conto corrente: 200.0

Cliente: Anna Bruni
    Saldo del conto corrente: 200.0

Cliente: Raul Falchi
    Saldo del conto corrente: 200.0
    Saldo del libretto di risparmio: 1500.0

Cliente: Vale Bova
    Saldo del conto corrente: 200.0
    Saldo del libretto di risparmio: 150.0
```

Esercizio 2

È possibile modificare la visibilità `protected` di `saldo` nella classe `Conto` in modo che sia nuovamente `private` e ripristinare così l'incapsulazione perfetta?

La risposta è si. Ma questo ha conseguenze nel codice della classe `ContoCorrente`.

In particolare:

1. poiché in un `ContoCorrente` il saldo può essere negativo, mentre nella superclasse esso è imposto non-negativo, occorre aggiungere in `ContoCorrente` un attributo che tenga conto della parte di saldo in rosso;

2. non è sufficiente sovrapporre (e riscrivere) solo il metodo `preleva`, ma è necessario sovrapporre anche `getSaldo` e `deposita` in modo da tenere conto sia della parte positiva (quella invisibile della superclasse), sia della parte negativa (propria della classe).

Esibire le necessarie (ed eleganti) modifiche nella classe `ContoCorrente` che rendano nuovamente il programma funzionante.