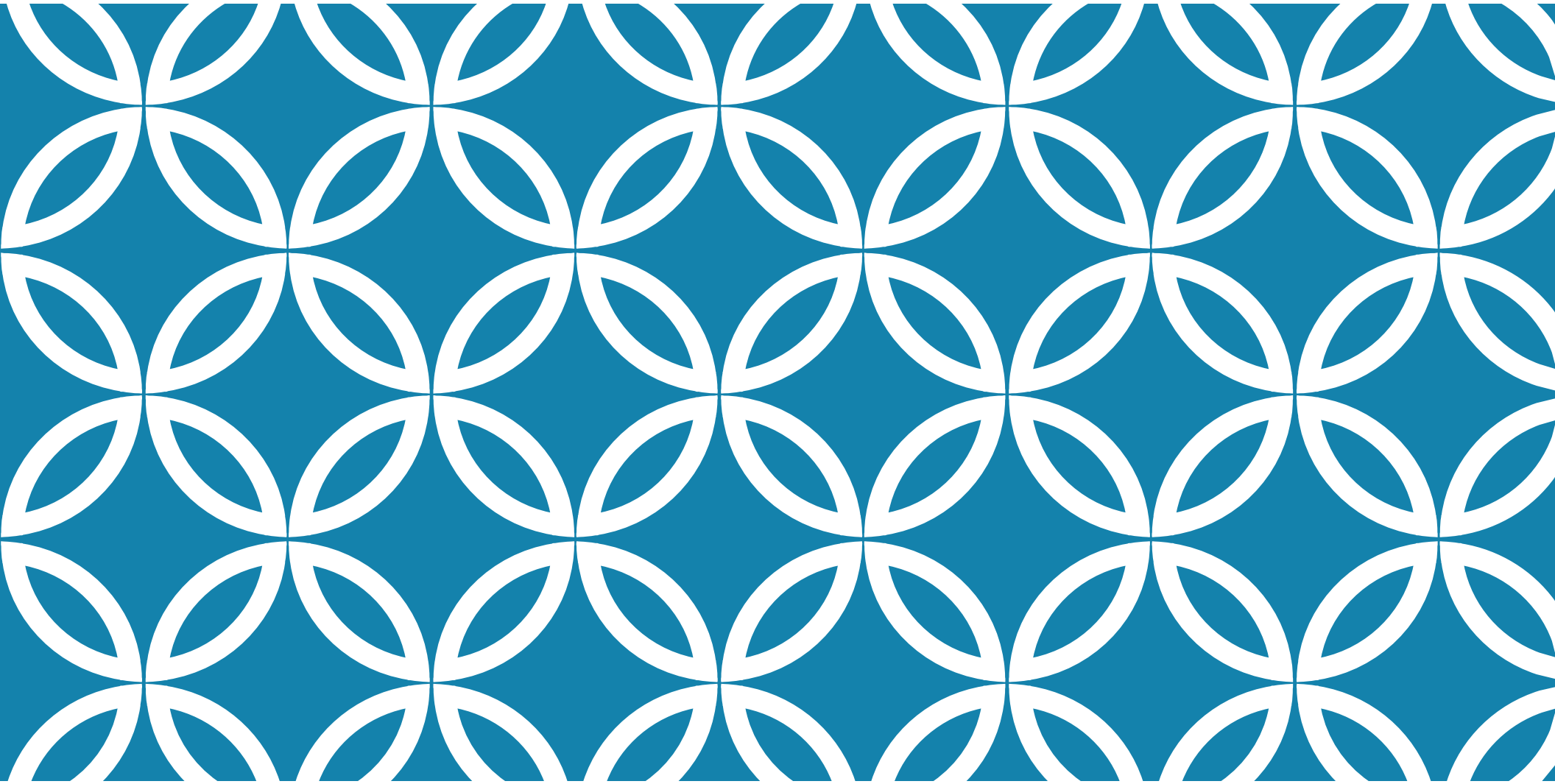

LABORATORIO DI PROGRAMMAZIONE

Corso di Laurea Ing. Gestionale 21/22

Ing. Antonio Luca Alfeo

luca.alfeo@ing.unipi.com

CLASSI E OGGETTI

- Un oggetto è una particolare entità informatica definita da:
  - attributi, che definiscono le caratteristiche dell'oggetto;
  - metodi, che definiscono le operazioni che si possono fare sull'oggetto.
- Una classe rappresenta una categoria di oggetti.
- Gli oggetti appartenenti ad una determinata classe prendono il nome di istanze della classe.
- Esempio: "Mario" e "Giovanni" sono due oggetti, entrambi istanze della classe "Persona"

CLASSI IN JAVA

```java
public class Persona {

    // Variabili (attributi)
    private String nome;
    private int eta;

    // Costruttore
    public Persona(String n, int e){
        nome = n;
        eta = e;
    }

    // Metodi
    public void presentati(){
        System.out.println("Ciao, sono "
+ nome + " ed ho " + eta + " anni");
    }

    public void invecchia(int nAnni){
        eta += nAnni;
    }

}
```

```java
public class TestPersona {

    public static void main(String[] args) {
        /* Creazione di un'istanza di Persona
         * tramite costruttore */
        Persona a = new Persona("Mario", 27);

        /* Invocazione di metodi */
        a.presentati();
        // Ciao, sono Mario ed ho 27 anni
        a.invecchia(3);
        a.presentati();
        // Ciao, sono Mario ed ho 30 anni

    }

}
```

MODIFICATORI PUBLIC E PRIVATE

- I modificatori *public* e *private* permettono di stabilire la visibilità di un membro (variabile o metodo) di una classe.
  - I membri *private* sono visibili solo alla classe di appartenenza.
  - I membri *public* sono visibili anche alle altre classi.

```
public class MyClass {
    public int varA;
    private int varB;

    public void metA() { … }
    private void metB() { … }
}

public class TestMyClass {
    public static void main(String[] args) {
        MyClass a = new MyClass();
        a.varA = 10;
        a.varB = 10;     // non consentito
        a.metA();
        a.metB();        // non consentito
    }
}
```

METODI GETTER E SETTER

- Generalmente gli attributi di una classe sono privati, per evitare che un utilizzatore esterno modifichi lo stato interno di un oggetto in maniera inconsistente.
- Tuttavia, in certi casi è necessario poter accedere al valore di questi attributi. Questo può essere realizzato tramite dei metodi che prendono il nome di getter (permettono di leggere un attributo) o setter (permettono di modificare un attributo).
- Possono implementare politiche di validazione dei dati.

```
public class MyClass {
    private int var;

    public int getVar(){          // getter per var
        return var;
    }

    public void setVar(int v){    // setter per var
        var = v;
    }

}
```

ESEMPIO "POPOLAZIONESCARAFAGGI"

- Creare una classe *PopolazioneScarafaggi* che simuli l'evoluzione di una popolazione di scarafaggi.
- Il costruttore deve avere come parametro il numero iniziale di scarafaggi della popolazione.
- Realizzare i metodi:
  - *raddoppia*, che simula il passaggio di un'era generazionale, in cui si assume che gli scarafaggi si riproducano raddoppiando la popolazione;
  - *spray*, che simula un'azione di disinfestazione che riduce del 10% la popolazione;
  - *getNumeroScarafaggi*, metodo getter che ritorna il numero di scarafaggi della popolazione.
- Per testare il corretto funzionamento della classe *PopolazioneScarafaggi*, realizzare la classe *TestScarafaggi* ed il relativo metodo main che esegue le seguenti operazioni:
  - Crea una popolazione di 100 scarafaggi.
  - Esegue due volte il raddoppio della popolazione.
  - Esegue una disinfestazione.
  - Esegue nuovamente un raddoppio.
- Ad ogni passo della simulazione, il programma deve stampare a video il numero di scarafaggi.

```java
public class PopolazioneScarafaggi {
    /* Rappresenta il numero di scarafaggi della popolazione */
    private int numeroScarafaggi;

    /* Costruttore */
    public PopolazioneScarafaggi(int n) {
        numeroScarafaggi = n;
    }


    /* Restituisce il numero di scarafaggi nella popolazione */
    public int getNumeroScarafaggi() {
        return numeroScarafaggi;
    }


    /* Raddoppia il numero di scarafaggi nella popolazione */
    public void raddoppia() {
        numeroScarafaggi *= 2;
    }


    /* Disinfesta la popolazione, uccidendo il 10% */
    public void spray() {
        numeroScarafaggi = numeroScarafaggi * 9 / 10;
    }
}
```

TESTING DI "POPOLAZIONESCARAFAGGI"

```
public class TestScarafaggi {
     public static void main(String[] args) {
          PopolazioneScarafaggi pop;
          pop = new PopolazioneScarafaggi(100);

          System.out.println("N=" + pop.getNumeroScarafaggi());
          pop.raddoppia();
          System.out.println("N=" + pop.getNumeroScarafaggi());
          pop.raddoppia();
          System.out.println("N=" + pop.getNumeroScarafaggi());
          pop.spray();
          System.out.println("N=" + pop.getNumeroScarafaggi());
          pop.raddoppia();
          System.out.println("N=" + pop.getNumeroScarafaggi());
     }
}
```

VARIABILI STATICHE

- Se una variabile di una classe ha il modificatore *static*, non ne esiste una copia per ogni oggetto, ma è unica per tutti le istanze della classe.

```
public class MyClass {
    public static int i = 10;
    public int j = 10;

    public void incrementa(){
        i++;
        j++;
    }
}

public class TestMyClass {
    public static void main(String[] args) {
        MyClass a = new MyClass();
        MyClass b = new MyClass();

        a.incrementa(); // a.i == 11 e a.j == 11
        b.incrementa(); // b.i == 12 e b.j == 11
        // a.i e b.i sono la stessa variabile: MyClass.i
    }
}
```

METODI STATICI

- Se un metodo ha il modificatore *static* può essere riferito senza specificare un'istanza
- Non può accedere a variabili non statiche!

```
public class MyClass {
    public static int i = 10;
    public int j = 10;

    public static void incrementa(){
        i++;
        j++;    // non consentito
    }
}

public class TestMyClass {
    public static void main(String[] args) {
        MyClass.incrementa();
    }
}
```

RIFERIMENTO THIS

- Ogni metodo non statico di una classe, oltre ai parametri dichiarati formalmente ha un parametro implicito: il riferimento *this*.
- Il riferimento *this* è un riferimento all'oggetto sul quale viene invocato il metodo.
- Grazie al riferimento *this*, quindi, è possibile accedere ai membri dell'oggetto.
- I metodi statici non hanno il riferimento *this*.

```
public class MyClass {
    public int a;
    public int b;

    public void metodo(int v1, int v2) {
        this.a = v1;    // accedo al membro "a" della
                        // istanza riferita da this
        b = v2; // in questo caso il riferimento
                // this è sottinteso
    }
}
```

IL METODO TOSTRING() (1/2)

- In Java ciascuna classe che definisce oggetti personalizzati "dovrebbe" implementare il metodo *toString()*, che restituisce la rappresentazione testuale dell'oggetto su cui è invocato.
- Tipicamente la rappresentazione testuale è caratterizzata da una intestazione e dagli attributi della classe.
- Il prototipo del metodo è sempre il seguente:

```
public String toString()
```

- Se viene implementato il metodo *toString()*, è possibile utilizzare una variabile di tipo classe in un contesto in cui ci dovrebbe essere una stringa (ad esempio nella stampa a video: *System.out.print()*) ed ottenere un risultato significativo.
- Ciò è possibile perché l'oggetto viene automaticamente convertito in stringa invocando su di esso il metodo *toString()*.

IL METODO TOSTRING() (2/2)

Per stampare un oggetto di tipo *Complex* viene sfruttata la conversione automatica poiché è stata definita la *toString()*. Per stampare il valore di un solo attributo devo utilizzare il corrispondente metodo "getter":

```
public class NewTestComplex {
    public static void main(String[] args) {
        Complex c = new Complex(1,-1);
        System.out.println(c); // stampa (1.0, -1.0i)
    }
}
```

```
public class NewTestComplex {
    public static void main(String[] args) {
        Complex c = new Complex(1,-1);
        System.out.println(c.getIm()); // stampa -1.0
    }
}
```

ESERCIZIO "DISTANZAPUNTO"

- Creare una classe *Punto* che rappresenta un punto sul piano e realizzare:
  - Costruttore che ha come parametri due double indicanti le coordinate x e y del punto utilizzando il riferimento this.
  - Metodi getter *getX* e *getY* che ritornano le coordinate del punto.
  - Metodo statico *distanzaStatic* che prende due punti e ne calcola la distanza.

- Creare una classe *DistanzaPunto* e realizzare un metono main che:
  - Crei un vettore di 4 punti rappresentati i vertici di un quadrato con coordinate
    (0, 0), (0, 10), (10, 10), (10, 0).
  - Chieda in input via tastiera le coordinate di un punto.
  - Stampi a video le coordinate del vertice più vicino al punto inserito e la distanza che li separa.

SOLUZIONE (1/3)

```
public class Punto {
    private double x;
    private double y;

    public Punto(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public static double distanzaStatic(Punto p1, Punto p2) {
        double dx = p1.x - p2.x;
        double dy = p1.y - p2.y;
        return Math.sqrt(dx * dx + dy * dy);
    }
}
```

SOLUZIONE (2/3)

```
import fiji.io.Lettore;

 public class DistanzaPunto {


     public static void main(String[] args) {


         Punto[] vertici;
         vertici = new Punto[] {
             new Punto(0, 0),
             new Punto(10, 0),
             new Punto(10, 10),
             new Punto(0, 10)
         };

         double x, y;
         System.out.println("Inserisci le coordinate di un punto:");

        x = Lettore.in.leggiDouble();

        y = Lettore.in.leggiDouble();

        Punto p = new Punto(x, y);
```

SOLUZIONE (3/3)

```
        double minDist = Punto.distanzaStatic(p, vertici[0]);

        int minIdx = 0;
        for (int i = 1; i < vertici.length; i++) {

            double curDist = Punto.distanzaStatic(p, vertici[i]);
            if (curDist < minDist) {
                minDist = curDist;
                minIdx = i;
            }

        }

        System.out.println("Il punto più vicino è ("
                + vertici[minIdx].getX() + ", " + vertici[minIdx].getY()
                + ") e la sua distanza è " + minDist);
    }

}
```

ESERCIZIO "COMPLEX" (1/2)

- Creare una classe *Complex* per rappresentare numeri complessi.
- Realizzare un costruttore che prende come parametri due double rappresentanti parte reale e immaginaria del numero complesso utilizzando il riferimento this ed un costruttore di default che li inizializza a 0.
- Le variabili di classe devono essere private e bisogna realizzare i metodi setter e getter per modificarle.
- Realizzare i metodi:
  - *getConjugate*, che ritorna il complesso coniugato
  - *getNorm*, che ritorna il modulo
  - *add (statico)*, che ha come parametri 2 numeri complessi e ritorna un numero complesso rappresentante la somma tra i numeri passati
  - *toString()*, che restituisce la stringa contenente la parte reale e la parte immaginaria del numero complesso

ESERCIZIO "COMPLEX" (2/2)

- Creare una classe *TestComplex* con un metodo main che chiede in input da tastiera la parte reale e la parte immaginaria di un numero complesso e stampa a video il numero inserito, il suo complesso coniugato ed il suo modulo.
- Creare una classe *SommaComplex* con un metodo main che prende in input da tastiera un vettore di numeri complessi (chiedendo prima la dimensione e poi chiedendo l'inserimento della parte reale ed immaginaria di ciascun elemento) e stampa a video un numero complesso rappresentante la somma di tutti gli elementi del vettore.

SOLUZIONE (1/6)

```
public class Complex {
    private double re;
    private double im;

    public Complex() {
        re = 0;
        im = 0;
    }

    public Complex(double re, double im) {
        this.re = re;
        this.im = im;
    }

    public void setRe(double real) {
        re = real;
    }

    public void setIm(double imag) {
        im = imag;
    }

    public double getRe() {
        return re;
    }

    public double getIm() {
        return im;
    }
```

SOLUZIONE (2/6)

```
public Complex getConjugate() {
    return new Complex(re, -1 * im);
}

public double getNorm() {
    return Math.sqrt(re * re + im * im);
}

public String toString() {

    String s;
     if (re == 0) {
         if (im == 0) {
             s = "0";
         } else {
             s = im + "i";
         }
     } else {
         if (im == 0) {
             s = "" + re;
         } else if (im < 0) {
             s = re + " " + im + "i";
         } else {
             s = re + " +" + im + "i";
         }
     }

    return s;
}
```

SOLUZIONE (3/6)

```
    public static Complex add(Complex op1, Complex op2) {
        return new Complex(op1.re + op2.re, op1.im + op1.im);
    }

    public static Complex add2(Complex op1, Complex op2) {
        Complex r = new Complex();
        r.setRe(op1.getRe() + op2.getRe());
        r.setIm(op1.getIm() + op2.getIm());
        return r;
    }
}
```

SOLUZIONE (4/6)

```
import fiji.io.Lettore;

public class TestComplex {
    public static void main(String[] args) {
        Complex c = new Complex();
        System.out.println("Scrivi la parte reale:");
        c.setRe(Lettore.in.leggiDouble());
        System.out.println("Scrivi la parte immaginaria:");
        c.setIm(Lettore.in.leggiDouble());


       System.out.print("Numero inserito: " + c);

        System.out.println();
        System.out.println("Complesso coniugato: " +
                                        c.getConjugate());

        System.out.println();
        System.out.println("Modulo: " + c.getNorm());
    }
}
```

SOLUZIONE (5/6)

```
import fiji.io.Lettore;

public class SommaComplex {
    public static void main(String[] args) {
        int n;
        Complex[] v;
        System.out.println("Scrivi la dimensione dell'array:");
        n = Lettore.in.leggiInt();
        v = new Complex[n];


        for (int i = 0; i < n; i++) {
            double re, im;
            System.out.println("Scrivi la parte reale del "
                    + (i + 1) + "o numero :");
            re = Lettore.in.leggiDouble();
            System.out.println("Scrivi la parte immaginaria del "
                    + (i + 1) + "o numero :");
            im = Lettore.in.leggiDouble();
            v[i] = new Complex(re, im);
        }
```

SOLUZIONE (6/6)

```
        Complex somma = new Complex();
         for (int i = 0; i < n; i++) {
             somma = Complex.add(somma, v[i]);
         }


         System.out.print("La somma è: " + somma);
         System.out.println();

    }
}
```

ESERCIZIO "CONTOCORRENTE"

- Creare una classe *ContoCorrente* che permetta di gestire un conto corrente bancario caratterizzato dai seguenti attributi: *nome dell'intestatario* (tipo String), *cognome dell'intestatario* (tipo String), *numero di conto corrente* (tipo long), *saldo residuo* (tipo double).
- Il costruttore deve avere come parametri il nome, il cognome ed il saldo iniziale. Il numero di conto corrente deve essere calcolato automaticamente in maniera incrementale (si può realizzare usando una variabile statica).
- Realizzare i metodi:
  - *deposita*, che permette di depositare denaro sul conto corrente
  - *preleva*, che permette di prelevare denaro dal conto corrente
  - *toString*, che ritorna le informazioni sul conto corrente (numero, intestatario, saldo residuo)
- Per testare il corretto funzionamento della classe *ContoCorrente*, realizzare la classe *TestConto* ed il relativo metodo main che esegue le seguenti operazioni:
  - Crea due conti correnti intestati a due persone diverse
  - Effettua un'operazione di bonifico dal primo al secondo conto (utilizzano i metodi preleva e deposita)
  - Stampa a video le informazioni dei conti prima e dopo il bonifico

SOLUZIONE (1/3)

```
public class ContoCorrente {
     /* Dati personali intestatario */
     private String nome;
     private String cognome;
     /* Numero di conto corrente */
     private long numeroConto;
     /* Saldo attuale */
     private double saldoAttuale;

     /* Contatore interno per l'assegnazione del
      * numero di c/c per ogni nuovo conto */
     private static long prossimoNumeroConto = 1;

     /* Costruttore */
     public ContoCorrente(String n, String c,  double saldoIniz) {
          nome = n;
          cognome = c;
          numeroConto = prossimoNumeroConto;
          saldoAttuale = saldoIniz;
          /* Incremento il contatore statico: la prossima
           * invocazione del costruttore assegnerà
           * il valore successivo */
          prossimoNumeroConto++;
     }
```

SOLUZIONE (2/3)

```
    public boolean deposita(double ammontare) {
        boolean opValida = (ammontare > 0);
        if (opValida) {
            saldoAttuale += ammontare;
        }
        return opValida;
    }
    public boolean preleva(double ammontare) {
        boolean opValida = ((ammontare > 0 ) &&
                                (saldoAttuale >= ammontare ));
        if (opValida) {
            saldoAttuale -= ammontare;
        }
        return opValida;
    }
    public String toString() {
        return "Il signor " + cognome + " " + nome
                + " è titolare del c/c n. " + numeroConto
                + " con saldo pari a " + saldoAttuale;
    }
}
```

SOLUZIONE (3/3)

```
public class TestConto {
     public static void main(String[] args) {
          ContoCorrente cc1;
          cc1 = new ContoCorrente("Mario", "Rossi", 5000);

          ContoCorrente cc2;
          cc2 = new ContoCorrente("Fabio", "Pagani", 15000);

          System.out.println(cc1);
          System.out.println(cc2);

          if (cc1.preleva(500)) {
              cc2.deposita(500);

 System.out.println("Bonifico effettuato " + "con successo!");

          } else {

    System.out.println("Non è stato possibile " + "effettuare il bonifico!");

          }

          System.out.println(cc1);
           System.out.println(cc2);
     }
}
```