BankAccount

```
public class BankAccount {
  private double balance;
  public BankAccount(){
    this(0);
  }
  public BankAccount(double initialBalance)
  {
    balance = initialBalance;
  }
  public void deposit(double amount) {
       balance = balance + amount;
  }
  public void withdraw(double amount) {
       balance = balance - amount;
  }
  public double getBalance() {
       return balance;
  }
  public void transfer(BankAccount other,
                       double amount){
    withdraw(amount);
    other.deposit(amount);
  }
}
```


SavingsAccount

```
public class SavingsAccount
               extends BankAccount {
  private double interestRate;
  public SavingsAccount(double rate){
       interestRate = rate;
  }
  public void addInterest() {
    double interest = getBalance() *
                      interestRate / 100;
    deposit(interest);
  }
 public static void main(String[] args) {
    SavingsAccount mio =
                        new SavingsAccount(2.0);
    mio.deposit(100.0);
    System.out.println("saldo = " +
                        mio.getBalance());
    mio.addInterest();
    System.out.println("saldo = " +
                        mio.getBalance());
  }
}

// balance = 100.0
// balance = 102.0
```

CheckingAccount

```
public class CheckingAccount extends
                                    BankAccount {
  public CheckingAccount(int initialBalance)  {
    super(initialBalance);
    transactionCount = 0;
  }

  public void deposit(double amount) {
    transactionCount++;
    super.deposit(amount);
  }

  public void withdraw(double amount)    {
    transactionCount++;
    super.withdraw(amount);
  }

  public void deductFees() {
    if (transactionCount > FREE_TRANSACTIONS) {
      double fees = TRANSACTION_FEE *
          (transactionCount - FREE_TRANSACTIONS);
      super.withdraw(fees);
    }
    transactionCount = 0;
  }

  private int transactionCount;
  private static final int FREE_TRANSACTIONS =
                                                3;
  private static final double TRANSACTION_FEE =
                                              2.0;
}
```


TimeDepositAccount

```
public class TimeDepositAccount extends
                                    SavingsAccount{
  public TimeDepositAccount(double rate,
                                     int maturity){
    super(rate);
    periodsToMaturity = maturity;
  }

  public void addInterest() {
    periodsToMaturity--;
    super.addInterest();
  }

  public void withdraw(double amount) {
    if (periodsToMaturity > 0)
  super.withdraw(EARLY_WITHDRAWAL_PENALTY);
    super.withdraw(amount);
  }

  private int periodsToMaturity;
  private static double
              EARLY_WITHDRAWAL_PENALTY = 20;
}
```

Variabili in ombra

```
class Padre {
  public int unNumero = 1;
}


public class Figlio extends Padre {
  float unNumero = 2.0F;

  public Figlio() {
    System.out.println("unNumero = " +
                                    unNumero);
    System.out.println("unNumero = " +
                              super.unNumero);
  }

  public static void main(String[] args) {
    Figlio f = new Figlio();
  }
}

// unNumero = 2.0
// unNumero = 1
```


Costruttore (I)

```
class A {
  public A(int i) {
    System.out.println("A(int)");
  }
  public A() {
    System.out.println("A()");
  }
}

public class B extends A {
  public B(int i) {
    super(i);
    System.out.println("B(int)");
  }
  public B(double d) {
    System.out.println("B(double)");
  }

  public static void main(String[] args) {
    B b1 = new B(1);   // A(int), B(int)
    B b2 = new B(2.5); // A(), B(double)
  }
}
```

Costruttore (II)

```
class A {
  public A(int i) {
    System.out.println("A(int)");
  }
}

public class B extends A {
  public B(int i) {
    super(i);
    System.out.println("B(int)");
  }
  public B(double d) {
    System.out.println("B(double)");
  }

  public static void main(String[] args) {
    B b1 = new B(1);   // A(int), B(int)
    B b2 = new B(2.5); // Compile-time Err
  }
}

// B.java:15: cannot resolve symbol
// symbol  : constructor A ()
// location: class A
//    public B(double d) {
//                       ^
// 1 error
```


Inizializzazione e caricamento (I)

```
class F {
  static int x = print("F: Static:
                              inizializzazione");
  private static int print(String s) {
    System.out.println(s);
    return 1;
  }
  public F() {
    System.out.println("F: costruttore");
  }
  int get() {
    return x;
  }
}

public class D {
  public static void main(String args[]) {
    boolean b = true;
      if ( b ) {
        System.out.println("Then: inizio");
        F y = new F();
        int a = y.get();
       System.out.println("Then: fine");
      }
  }
}
// (b = true) =>
// Then: inizio
// F: Static: inizializzazione
// F: costruttore
// Then: fine
//
// (b = false) => nessun output
```

Inizializzazione e caricamento (II)

```
class B {
  static int b = print("B: Static: iniz."); //2
   private static int print(String s) {
     System.out.println(s);
     return 1;
   }
   public B() {
     System.out.println("B: costruttore");//4
   }
}

class A extends B {
  static int a = print("A: Static: iniz.");//3
  private static int print(String s) {
     System.out.println(s);
     return 1;
   }
   public A() {
     System.out.println("A: costruttore"); //5
   }
   public static void main(String[] args) {
     System.out.println("Inizio main");
     A x = new A();
   }
}

public class C {
  public static void main(String[] args) {
     System.out.println("Main: inizio"); // 1
     A x = new A();
     System.out.println("Main: fine"); // 6
   }
}
```


Inizializzazione e caricamento (III)

```
class B {
  static int b = print("B: Static: iniz.");//1
  private static int print(String s) {
     System.out.println(s);
     return 1;
   }
  public B() {
     System.out.println("B: costruttore");//4
   }
}

public class A extends B {
  static int a = print("A: Static iniz.");//2
  private static int print(String s) {
     System.out.println(s);
     return 1;
   }
   public A() {
     System.out.println("A: costruttore");//5
   }
   public static void main(String[] args) {
     System.out.println("Inizio main");//3
     A x = new A();
   }
}
```

Dynamic binding

```
class B {

  void foo() {
    System.out.println("B.foo");
  }

}

public class A extends B {

  void foo() {
    System.out.println("A.foo");
  }

  public static void main(String[] args) {
    B b = new B();
    b.foo(); // B.foo
    b = new A();
    b.foo(); // A.foo
 }
}
```


Classe Employee

```
class Employee {
  public Employee(String n, double s, int year,
                  int month, int day) {
     name = n;
     salary = s;
     GregorianCalendar calendar
        = new GregorianCalendar(year, month -
                                          1, day);
      // GregorianCalendar uses 0 for January
     hireDay = calendar.getTime();
  }

  public String getName() {
     return name;
  }

  public double getSalary() {
     return salary;
  }

  public Date getHireDay() {
     return hireDay;
  }

  public void raiseSalary(double byPercent) {
     double raise = salary * byPercent / 100;
     salary += raise;
  }

  private String name;
  private double salary;
  private Date hireDay;
}
```

Classe Manager

```
class Manager extends Employee {
  public Manager(String n, double s,
            int year, int month, int day) {
     super(n, s, year, month, day);
     bonus = 0;
  }

  public double getSalary() {
     double baseSalary = super.getSalary();
     return baseSalary + bonus;
  }

  public void setBonus(double b) {
     bonus = b;
  }

  private double bonus;

}
```


equals

```
public class BankAccount {
  private double balance;

  public BankAccount(double aBalance) {
    balance = aBalance;
  }
  public BankAccount() {
    this(0);
  }
  public String toString() {
    return "BankAccount[balance=" + balance + "]";
  }
  void deposit(double amount) {
    balance += amount;
  }
  void withdraw(double amount) {
    balance -= amount;
  }
  double getBalance() {
    return balance;
  }
  public void transfer(BankAccount other, double amount){
    withdraw(amount);
    other.deposit(amount);
  }
  public static void main(String args[]) {
    BankAccount mio = new BankAccount(5.0);
    BankAccount tuo = new BankAccount(5.0);

    if (mio == tuo)
      System.out.println("mio e tuo sono lo stesso oggetto");
    else
      System.out.println("mio e tuo non sono lo stesso
                                                    oggetto");
    if (mio.equals(tuo))
      System.out.println("mio e tuo sono uguali");
    else
      System.out.println("mio a e tuo non sono uguali");
  }
}
// Output
// mio e tuo non sono lo stesso oggetto
// mio a e tuo non sono uguali
```

equals

```
public class DemoEquals {
  public static void main(String[] args) {
    Integer a = new Integer("123");
    Integer b = new Integer("123");

    if (a == b)
      System.out.println("a e b sono lo
                          stesso oggetto");
    else
      System.out.println("a e b non sono lo
                          stesso oggetto");
    if (a.equals(b))
      System.out.println("a e b sono
                                  uguali");
      else
      System.out.println("a e b non sono
                                  uguali");
  }
}

// Output
// a e b non sono lo stesso oggetto
// a e b sono uguali

// Integer sovrascrive il metodo equals
```


equals

```
public class BankAccount {
  private double balance;

  // costruttori, toString, deposit,
  // withdraw, getBalance, transfer

  public boolean equals(Object obj) {
    if (!(obj instanceof BankAccount))
      return false;
    return ((BankAccount)obj).balance ==
                                    balance;
  }

  public static void main(String args[]) {
    BankAccount mio = new BankAccount(5.0);
    BankAccount tuo = new BankAccount(5.0);

      if (mio == tuo)
        System.out.println("mio e tuo sono
                        lo stesso oggetto");
      else
        System.out.println("mio e tuo non
                  sono lo stesso  oggetto");
      if (mio.equals(tuo))
        System.out.println("mio e tuo sono
                                    uguali");
      else
        System.out.println("mio a e tuo non
                                sono uguali");
  }
}
// Output
// mio e tuo non sono lo stesso oggetto
// mio a e tuo sono uguali
```

equals

```
public class SavingsAccount extends BankAccount
{
  private double interestRate;
  public  SavingsAccount(double rate) {
    interestRate = rate;
  }
  public  SavingsAccount() {this(0);}
  public void addInterest() {
    double interest = getBalance() *
                              interestRate / 100;
    deposit(interest);
  }
  public static void main(String[] args) {
    SavingsAccount mio = new SavingsAccount(5);
    BankAccount tuo = new BankAccount(2.0);
    mio.deposit(2.0); // mio e tuo, stesso saldo

    if (mio == tuo)
      System.out.println("mio e tuo sono lo
                                stesso oggetto");
    else
      System.out.println("mio e tuo non sono lo
                                stesso oggetto");
    if (tuo.equals(mio))
      System.out.println("mio e tuo sono
                                        uguali");
    else
      System.out.println("mio e tuo non sono
                                        uguali");
  }
}
// Output
// mio e tuo non sono lo stesso oggetto
// mio e tuo sono uguali
```


clone

```
public class CloneDemo implements Cloneable {
    private int i;
    private double d;
    private String s;
    public CloneDemo(int i, double d, String s) {
        this.i = i;
        this.d = d;
        this.s = s;
    }
    public static void main(String[] args)
                    throws CloneNotSupportedException {
        CloneDemo a = new CloneDemo(1, 2.5, "ciao!");
        CloneDemo b;

        b = (CloneDemo)a.clone();
        System.out.println(a);
        System.out.println(b);
        System.out.println("a e b sono " +
                    ((a == b)? "uguali" : "diversi"));
    }
    protected Object clone()
                    throws CloneNotSupportedException {
        try {
            CloneDemo app = (CloneDemo)super.clone();
            return app;
        } catch(CloneNotSupportedException e) {
            throw e;
        }
    }
    public String toString() {
        return "CloneDemo[i="+i+"; d="+d+"; s="+s+"]";
    }
}
// Output
// CloneDemo[i=1; d=2.5; s=ciao!]
// CloneDemo[i=1; d=2.5; s=ciao!]
// a e b sono diversi
```

clone

```
class UnaClasse implements Cloneable {
    private int x;
    public UnaClasse(int x) {
        this.x = x;
    }
    public Object clone() throws CloneNotSupportedException {
        try {
            UnaClasse app = (UnaClasse)super.clone();
            return app;
        } catch(CloneNotSupportedException e) {
            throw e;
        }
    }
    public String toString() {
        return "UnaClasse[x="+x+"]";
    }
}

public class DeepCopy implements Cloneable {
    private int i;
    private double d;
    private UnaClasse s;

    public DeepCopy(int i, double d) {
        this.i = i;
        this.d = d;
        s = new UnaClasse(1);
    }
    public static void main(String[] args)
                            throws CloneNotSupportedException {
        DeepCopy a = new DeepCopy(1, 0.5);
        System.out.println(a);
        DeepCopy b = (DeepCopy)a.clone();
        System.out.println(b);
    }
    public String toString() {
        return "CloneDemo[i="+i+"; d="+d+"; s="+s+"]";
    }
    protected Object clone()
                            throws CloneNotSupportedException {
        try {
            DeepCopy app = (DeepCopy)super.clone();
            app.s = (UnaClasse)s.clone();
            return app;
        } catch(CloneNotSupportedException e) {
            throw e;
        }
    }
}
```