Programmazione II

A.A. 2022-23
Prof. Maria Tortorella

Definire le classi

Il processo di definizione di una classe.

Struttura di una classe.

Metodi.

Costruttori.

Variabili d'istanza e variabili locali.

Commenti e documentazione.

Astrazione

- *Abstraction*: A view of an object that focuses on the information relevant to a particular purpose and ignores the remainder of the information
- *Information Hiding*: A software development technique in which each module's interfaces reveals as little as possible about the module's inner workings and other modules are prevented from using information about the module that is not in the module's interface specification
  - IEEE standard glossary of software engineering terminology

Scatole nere

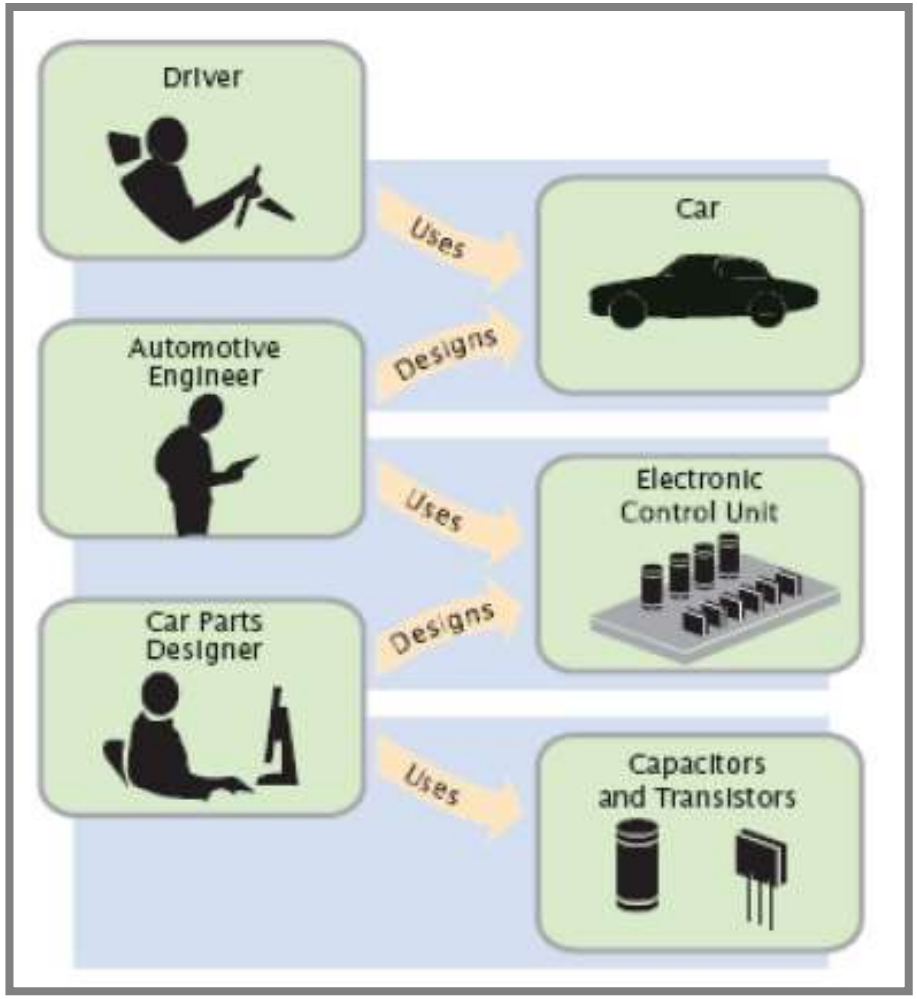
- Comportamento di una radio
  - Nasconde i dettagli interni
- Cosa mettere in una scatola ?

Scatole nere

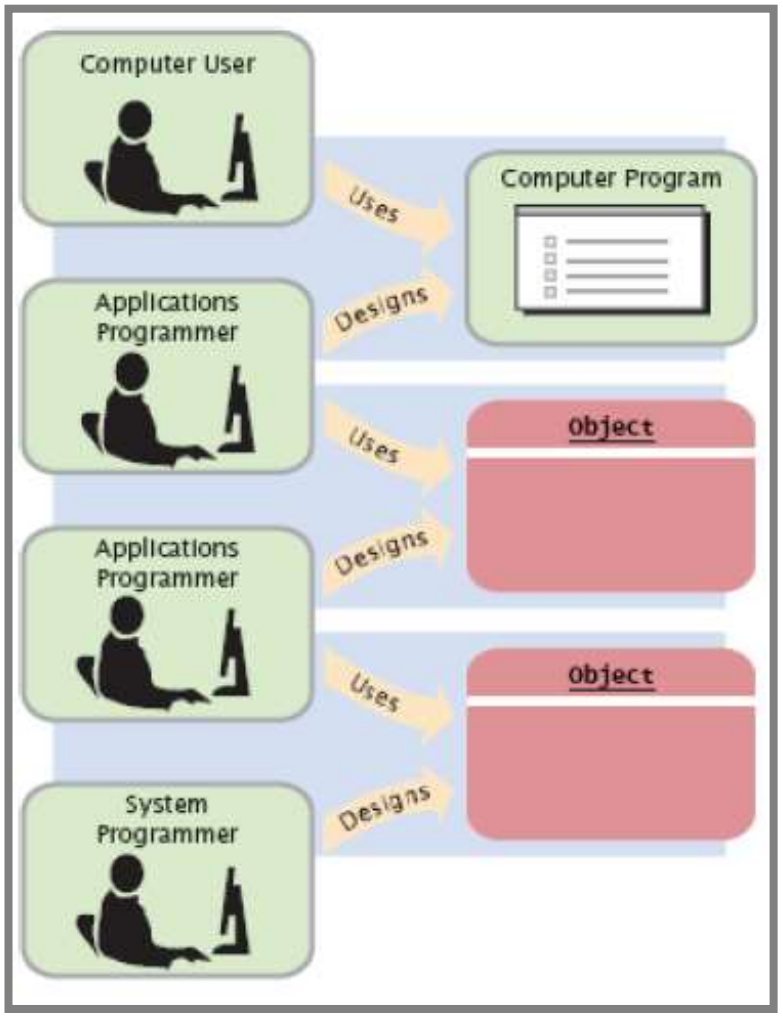
- Nella programmazione orientata agli oggetti le "scatole nere" sono gli oggetti
- Il concetto di astrazione guida nella identificazione della classe che può essere utilizzata per risolvere un determinato problema
- Il concetto di information hiding nella realizzazione della classe stessa

Un esempio di astrazione

- Automobile
  - trasmissione
  - centralina
  - ...



Programmazione OO

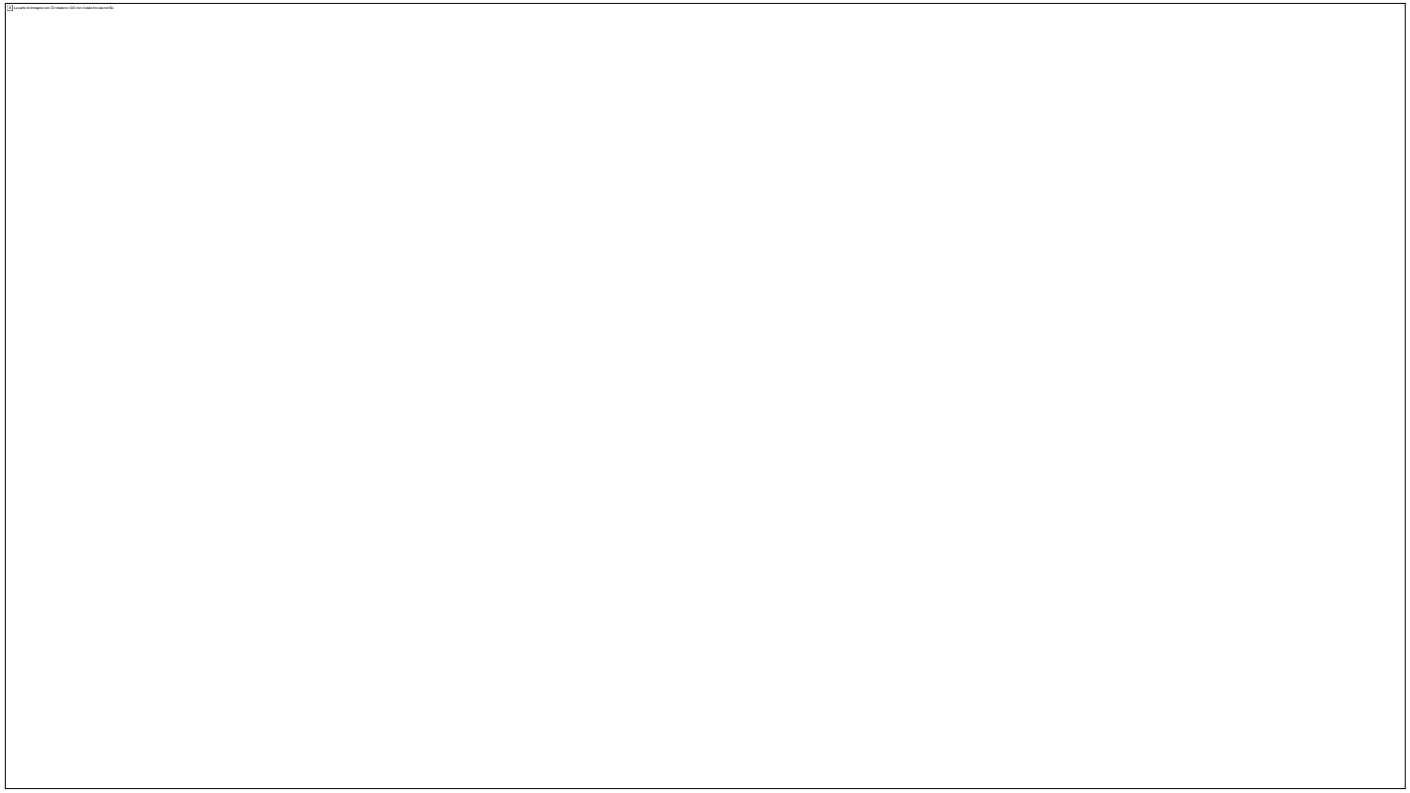


- Porta la stessa visione del software
  - Non solo tipi primitivi e routines
- Basta conoscere il comportamento esterno, non la struttura interna

Processo di progettazione

- Ovviamente lo strumento NON basta
  - È sicuramente possibile progettare cattivi programmi OO
- Per implementare delle buone classi:
  - Prima: definire il comportamento di una classe
  - Poi: definirne l'implementazione

Com’è fatta una classe



Interfaccia: insieme dei metodi pubblici

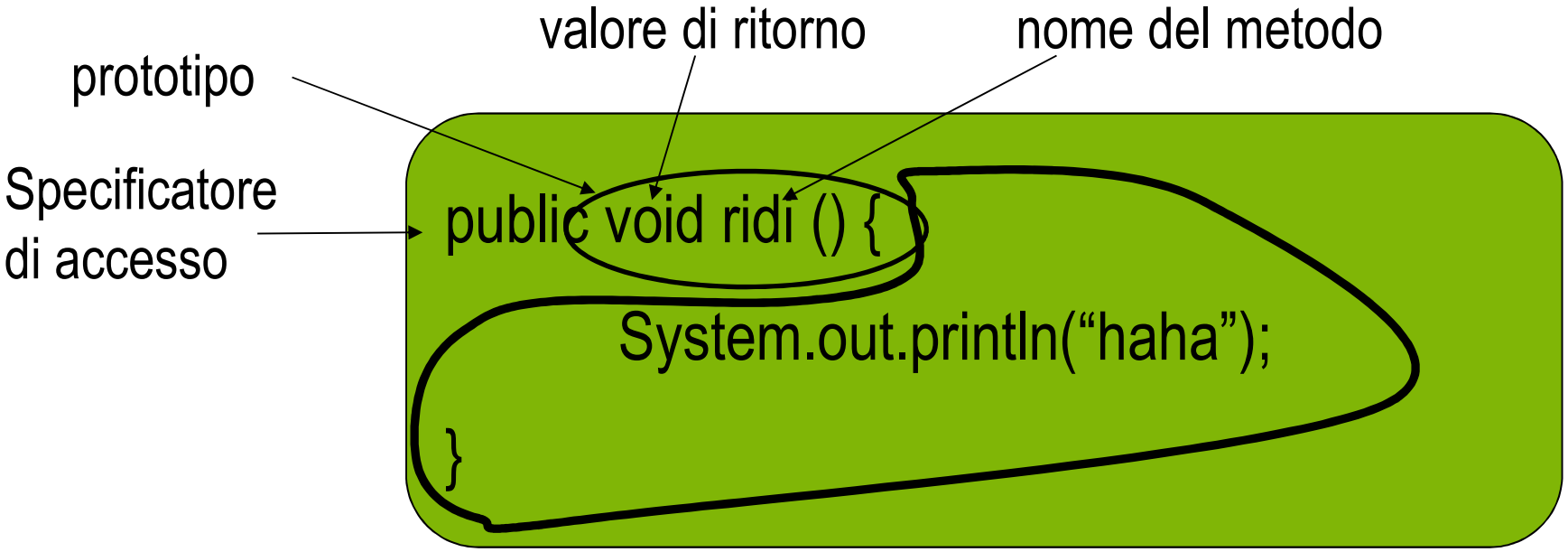
La definizione di una classe

```
class NomeDellaClasse {
        definizione di metodo
        definizione di metodo
        ...
}
```

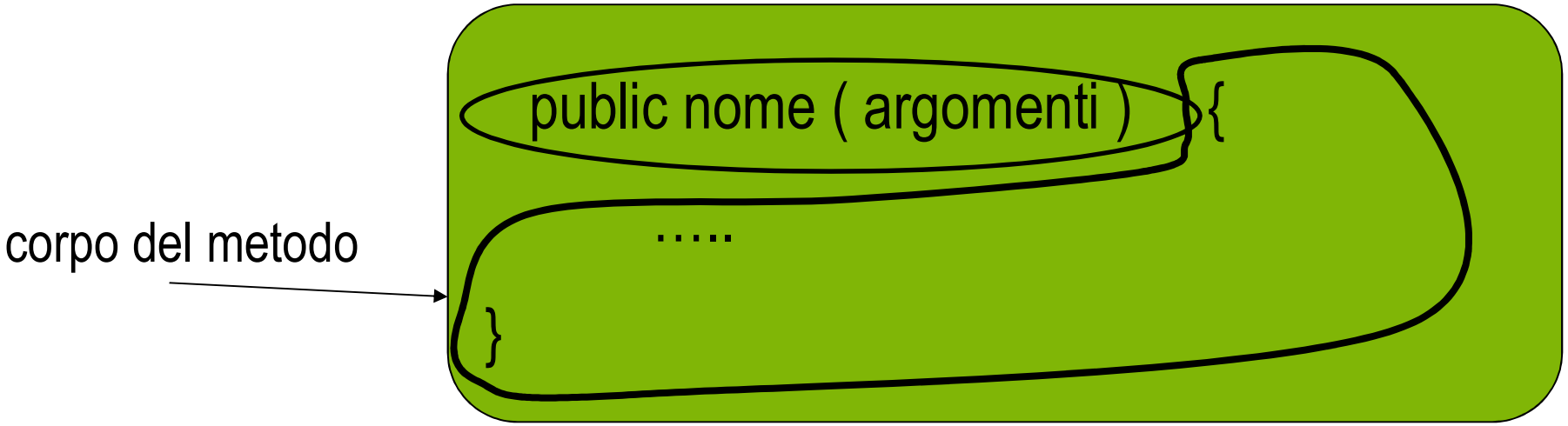
- Una classe contiene i suoi metodi
- Le parentesi graffe aperte e chiuse fungono da delimitatori del contenuto di una classe

La definizione di un metodo

- Prototipo e corpo (body)
- Le parentesi graffe aperte e chiuse fungono da delimitatori del corpo del metodo



La definizione di un costruttore



- Nessun valore di restituzione
- Il nome del metodo coincide con quello della classe

Un esempio

- La classe Allegro1 modella una persona allegra

```
class Allegro1 {
        public Allegro1() {
        }
        public void ridi() {
                System.out.println("haha");
        }
}
```

Utilizzare la classe **Allegro1**

1. Salvare la classe in un file (Allegro1.java)
2. Compilare la classe (javac Allegro1.java)
3. Scrivere un programma che utilizzi la classe Allegro1

```
class UsaAllegro1 {
        public static void main(String[] a) {
                Allegro1 x;
                x = new Allegro1();
                x.ridi();
                x.ridi();
        }
}
```

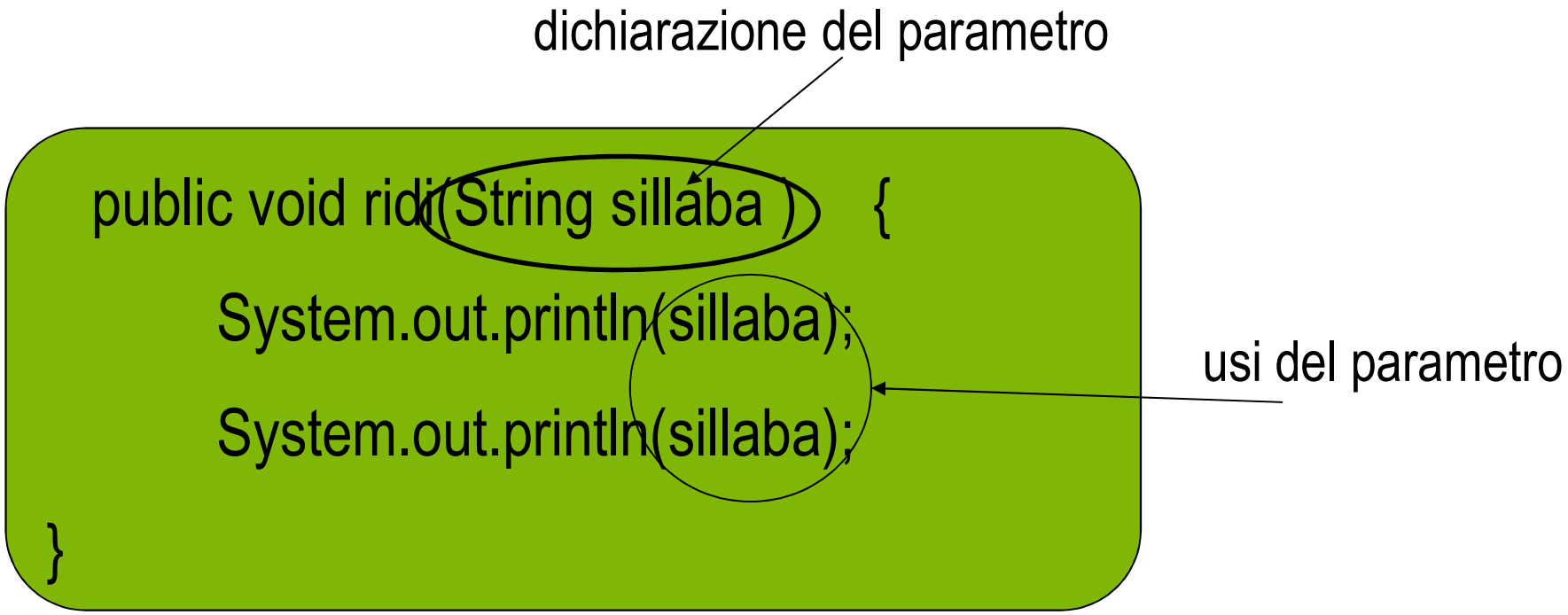
```
haha
haha
```

Metodi con argomenti

- Supponiamo che il mittente del messaggio ridi debba poter specificare la sillaba della risata
  - ha, ho, hee …

```
class UsaAllegro2 {
        public static void main(String[] a) {
                Allegro2 x;
                x = new Allegro2();
                x.ridi(”ho");
                x.ridi("hee");
        }
}
```

Metodi con argomenti



- Il numero e tipo degli argomenti nel messaggio devono coincidere con quelli nel prototipo

Esempio

- Notare che il metodo ridi è overloaded

```
class Allegro2 {
        public Allegro2() {
        }
        public void ridi() {
                System.out.println("haha");
        }
        public void ridi(String sillaba) {
                System.out.print(sillaba);
                System.out.println(sillaba);
        }
}
```

Utilizzare la classe **Allegro2**

1. Salvare la classe in un file (Allegro2.java)
2. Compilare la classe (javac Allegro2.java)
3. Scrivere un programma che utilizzi la classe Allegro2

```
class UsaAllegro2 {
        public static void main(String[] a) {
                Allegro2 x;
                x = new Allegro2();
                x.ridi();
                x.ridi("ho");
                x.ridi("hee");
        }
}
```

haha
hoho
heehee

Oggetti con stato

- Supponiamo di voler fornire al metodo costruttore un argomento String che rimpiazzi "ha" come sillaba di default per la risata

```
class UsaAllegro3 {
        public static void main(String[] a) {
                Allegro3 x, y;
                x = new Allegro3("ho");
                x.ridi();               // hoho
                x.ridi("hee");          // heehee
                x.ridi();               // hoho
                y=new Allegro3("hee");
                y.ridi();               //?
        }
}
```

hoho
heehee
hoho
heehee

Oggetti con stato

- Costruttore: public Allegro3 (String sillaba) …
- Problemi:
  - Gli argomenti di un metodo esistono solo durante l'esecuzione del metodo
  - Gli argomenti di un metodo sono visibili solo al metodo
- Come fare in modo che il metodo ridi possa conoscere il valore della sillaba ?
- Fornire agli oggetti la capacità di memorizzazione

Variabili di istanza (instance variables)

Variabili di istanza

- Una variabile dichiarata all'interno di una classe ma al di fuori di qualsiasi metodo
  - Accessibile a tutti i metodi dell'oggetto
  - Lo scopo è di memorizzare le informazioni necessarie ai metodi che devono essere preservate tra diverse invocazioni
  - Ciascun oggetto ha le proprie variabili di istanza le quali hanno i propri valori
  - Stato di un oggetto: i valori delle variabili di stato
  - Tipicamente è inizializzato dal costruttore

Esempio

```
class Allegro3 {
        public Allegro3(String sillaba) {
                defaultSillaba = sillaba;
        }
        public void ridi() {
                System.out.print (defaultSillaba );
                System.out.println (defaultSillaba );
        }
        public void ridi(String sillaba) {
                System.out.print(sillaba);
                System.out.println(sillaba);
        }
        private String defaultSillaba ;
}
```

Le variabili di istanza possono essere usate da tutti i metodi

Dichiarazione di una variabile di istanza

Utilizzare la classe **Allegro3**

1. Salvare la classe in un file (Allegro3.java)
2. Compilare la classe (javac Allegro3.java)
3. Scrivere un programma che utilizzi la classe Allegro3

```
class UsaAllegro3{
        public static void main(String[] a) {
                Allegro3 x;
                x = new Allegro3("hi");
                x.ridi();
                x.ridi("ho");
                x.ridi("hee");
        }
}
```

```
hihi
hoho
heehee
```

... miglioriamo ancora ...

```
class Allegro4 {
        public Allegro4 () {
        defaultSyl = new String("ha");
        }
        public Allegro4 (String syl) {
                defaultSyl = syl;
        }
        public void ridi() {
                System.out.print (defaultSyl);
                System.out.println(defaultSyl);
        }
        public void ridi(String syl) {
                System.out.print(syl);
                System.out.println(syl);
        }
        public void ridi(String s1, String s2) {
                System.out.print(s1);
                System.out.println(s2);
        }
        private String defaultSyl;
}
```

… utilizzatore

```
import java.io.*;
class RidiamoUnPo {
        public static void main(String[] a) {
                System.out.println(“Vivi allegramente!");
                Allegro4 x,y,z;
                x = new Allegro4 ("yuk");
                y = new Allegro4 ("harr");
                z = new Allegro4 ();
                x.ridi();
                x.ridi("hee");
                y.ridi();
                z.ridi();
                y.ridi(“ha”,”harr”);
        }
}
```

Cosa sarà visualizzato?

yukyuk
hehe
harrharr
haha
haharr

Un approccio metodologico

FASE1: Progettazione dell'Interfaccia Pubblica

- ✓ Decidere il comportamento che la classe dovrà fornire
  - Identificare i metodi da fornire
- ✓ Stabilire in che modo la classe verrà usata
  - Definire l'interfaccia della classe, i prototipi dei metodi
- ✓ Scrivere un programma di esempio che utilizza la classe
- ✓ Scrivere lo scheletro della classe
  - Usare solo i prototipi e lasciare i corpi vuoti

FASE2: Implementazione di una classe

Un conto corrente bancario

- ✓ Decidere il comportamento che la classe dovrà fornire
  - Identificare i metodi da fornire

  - Depositare denaro
  - Prelevare denaro
  - Chiedere il saldo

  - Creare un nuovo conto

Un conto corrente bancario

- ✓ Decidere il comportamento che la classe dovrà fornire
  - Identificare i metodi da fornire

```
deposit
withdraw
getBalance
```

Progettazione dell'interfaccia

- ✓ Stabilire in che modo la classe verrà usata
  - ✓ Esempi di invocazione

```
harrysChecking.deposit(2000);
harrysChecking.withdraw(500);
System.out.println(harrysChecking.getBalance());
```

  - Definire l'interfaccia della classe, i prototipi dei metodi
    - Specificatori di accesso : public, private
    - Il tipo del valore che viene restituito
    - Nome di metodo: deposit
    - Lista Parametri: double amount

```
public void deposit(double amount) { . . . }
public void withdraw(double amount) { . . . }
public double getBalance() { . . . }
```

Nota sintattica

```
accessSpecifier returnType methodName(parameterType
                                       parameterName, . . .)
{
    method body
}
```

Esempio:

```
public void deposit(double amount)
{
    . . .
}
```

Definizione di un metodo

Costruttore

- Tipicamente serve ad inizializzare le variabili d'istanza

```
public BankAccount()
{
    // da definire . . .
}
```

- E' eseguito all'atto della creazione di un nuovo oggetto

Nota sintattica

```
accessSpecifier ClassName(parameterType parameterName,...)
{
    constructor body
}
```

Esempio:

```
public BankAccount(double initialBalance)
{
      . . .
}
```

Definizione di un costruttore

Interfaccia pubblica

```
public class BankAccount{
    // Constructors
    public BankAccount(){
        // body--filled in later
    }
    public BankAccount(double initialBalance) {
        // body--filled in later
    }
        // Methods
    public void deposit(double amount) {
        // body--filled in later
    }
    public void withdraw(double amount) {
        // body--filled in later
    }
    public double getBalance(){
        // body--filled in later
    }
    // private fields--filled in later
}
```

Nota sintattica

```
accessSpecifier class ClassName
 {
    constructors
    methods
    fields
 }
```

Esempio:

```
 public class BankAccount
 {
    public BankAccount(double initialBalance) { . . . }
    public void deposit(double amount) { . . . }
   . . .
 }
```

Definizione di una classe

Documentare l'interfaccia

```
/**
 * Withdraws money from the bank account.
 * @param the amount to withdraw
*/
public void withdraw(double amount)
{
   // implementation filled in later
}
```

```
/**
 * Gets the current balance of the bank account.
 * @return the current balance
*/
public double getBalance()
{
   // implementation filled in later
}
```

Commentare l'interfaccia pubblica

- Prima di una classe:
  - Indicare cosa modella una classe

```
/**  A bank account has a balance that can
 * be changed by deposits and withdrawals
 */
public class BankAccount
```

- Prima di ogni metodo
- Indicare quale comportamento è modellato da ciascun metodo
  - Indicare i parametri facendoli precedere da @param
  - Indicare il valore restituito facendolo precedere da @return

Stato

- I dati che definiscono lo stato di un oggetto sono memorizzati in variabili d'istanza
  - Campi
- Ogni oggetto ha la sua copia delle variabili d'istanza
- La classe definisce le variabili d'istanza

```
public class BankAccount
{
   . . .
   private double balance;
}
```

- Specificatore d'accesso
- Tipo delle variabili
- Nome delle variabili

Nota sintattica

```
 accessSpecifier class ClassName
{
    . . .
    accessSpecifier fieldType fieldName;
    . . .
}
```

Esempio:

```
public class BankAccount
{ . . .
    private double balance;
}
```

Definire variabili d'istanza

Accedere alle variabili d'istanza

- Il metodo deposit, come tutti gli altri metodi, può accedere ai campi d'istanza privati

```
public void deposit(double amount)
{
   balance = balance + amount;
}
```

Accedere alle variabili d'istanza

- I metodi delle altre classi NON possono !

```
public class BankRobber
{
   public static void main(String[] args)
   {
     BankAccount momsSavings = new BankAccount(1000);
     . . .
     momsSavings.balance = -1000; // ERROR
   }
}
```

Esempio di chiamata di un costruttore

```
BankAccount harrysChecking = new BankAccount(1000);
```

- Crea un nuovo oggetto di tipo `BankAccount`
- Chiama il costruttore con argomenti
- Assegna al parametro del costruttore `initialBalance` il valore 1000
- Il costruttore assegna alla variabile d'istanza `balance` dell'oggetto appena creato il valore di `initialBalance`
- Restituisce un riferimento all'oggetto creato, ovvero la locazione di memoria dell'oggetto stesso, come valore dell'operatore new
- Memorizza il riferimento all'oggetto nella variabile `harrysChecking`

Costruttori

- Inizializzazione delle variabili di stato

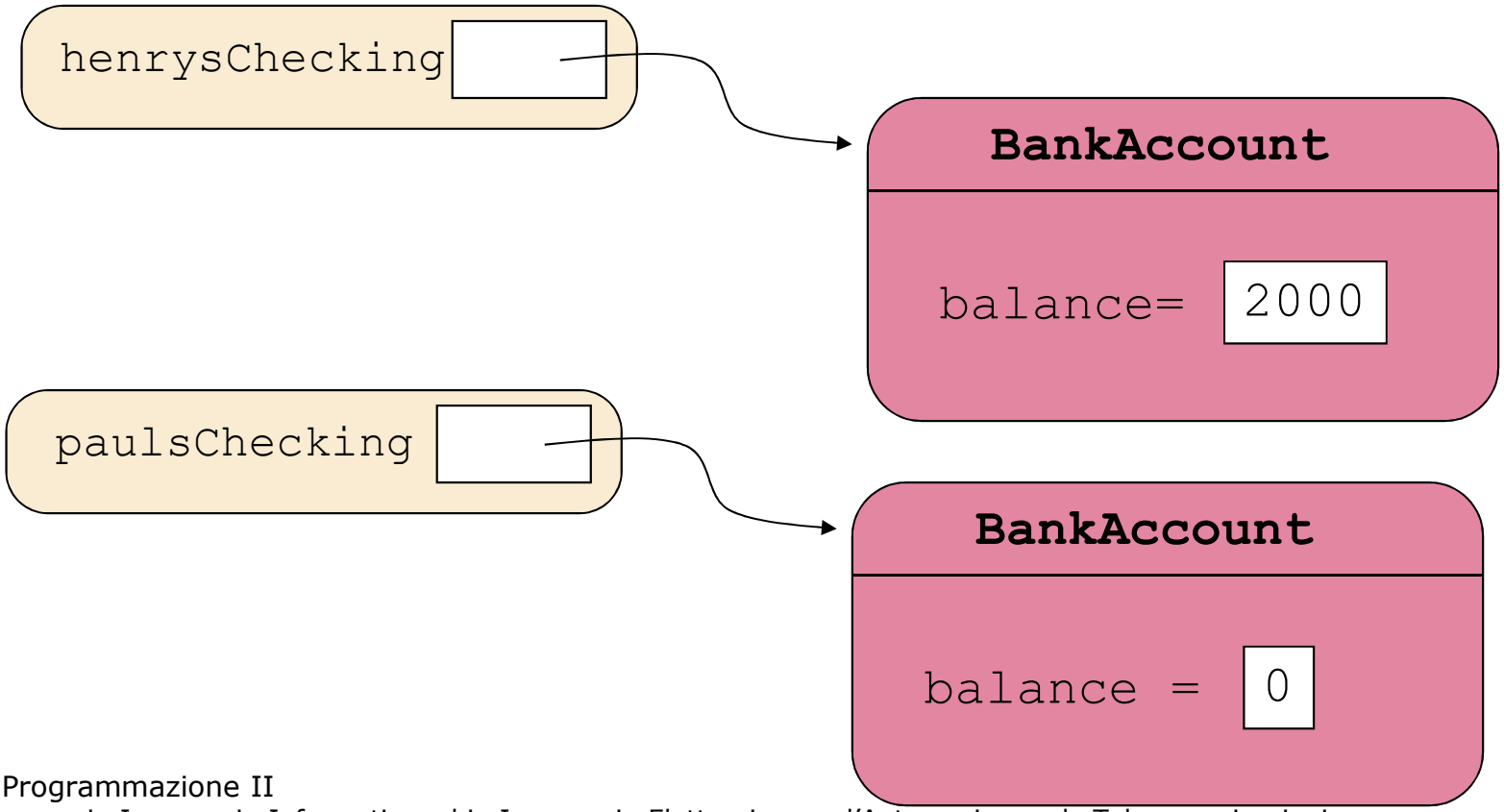
```
public BankAccount()
{
   balance = 0;
}

public BankAccount(double initialBalance)
{
   balance = initialBalance;
}
```

Creare oggetto BankAccount

```
BankAccount henrysChecking = new BankAccount(2000);

BankAccount paulsChecking = new BankAccount();
```



Metodi

- Senza valore di restituzione

```
/**
 * Withdraws money from the bank account.
 * @param the amount to withdraw
/
public void withdraw(double amount){
      balance = balance - amount;
}
```

- Senza parametri di ingresso

```
/**
 * Gets the current balance of the bank account.
 * @return the current balance
*/

public double getBalance(){
   return balance;
}
```

Esempio di chiamata a metodo

```
henrysChecking.deposit(500);
```

- Pone il valore del parametro `amount` a 500
- Considera la variabile d'istanza dell'oggetto assegnato a `harrysChecking`
- Assegna a tale variabile d'istanza il valore della somma di `amount` e `balance`

Nota sintattica

```
return expression;
```

or

```
return;
```

Esempio:

```
return balance;
```

Definisce il valore di restituzione di un metodo

Esempio completo

```
01: /**
02:  * A bank account has a balance that can be changed by
03:  * deposits and withdrawals.
04: */
05: public class BankAccount
06: {
07:    /**
08:     * Constructs a bank account with a zero balance.
09:    */
10:    public BankAccount()
11:    {
12:       balance = 0;
13:    }
14:
15:    /**
16:     * Constructs a bank account with a given balance.
17:     * @param initialBalance the initial balance
18:    */
```

Esempio completo

```
19:    public BankAccount(double initialBalance)
20:    {
21:       balance = initialBalance;
22:    }
23:
24:    /**
25:     * Deposits money into the bank account.
26:     * @param amount the amount to deposit
27:    */
28:    public void deposit(double amount)
29:    {
30:       balance = balance + amount;
31:    }
32:
33:
34:    /**
35:     * Withdraws money from the bank account.
36:     * @param amount the amount to withdraw
```

Esempio completo

```
37:     */
38:     public void withdraw(double amount)
39:     {
40:        balance = balance - amount;
41:     }
42:
43:
44:     /**
45:      * Gets the current balance of the bank account.
46:      * @return the current balance
47:     */
48:     public double getBalance()
49:     {
50:        return balance;
51:     }
52:
53:     private double balance;
54: }
```

Testare una classe

- Test Driver
  - Una classe con un metodo main

1. Costruire una o più istanze della classe da testare
2. Invocarne i metodi
3. Stampare i risultati

Test Driver

```
01: /**
02:  * A class to test the BankAccount class.
03: */
04: public class BankAccountTester
05: {
06:    /**
07:     * Tests the methods of the BankAccount class.
08:     * @param args not used
09:    */
10:    public static void main(String[] args)
11:    {
12:       BankAccount harrysChecking = new BankAccount();
13:       harrysChecking.deposit(2000);
14:       harrysChecking.withdraw(500);
15:       System.out.println(harrysChecking.getBalance());
16:    }
17: }
```

Domande

- Quale potrebbe essere l'implementazione del metodo translate:

  public void translate(int dx, int dy)
  ??

Esercizio

- Completare la classe conto corrente con il nome dell'intestatario e la gestione del massimo scoperto