

Ereditarietà

Ereditarietà

Ereditarietà: il rapporto tra una classe più generale (superclasse) e una classe più specializzata (sottoclasse).

La sottoclasse eredita i dati e il comportamento dalla superclasse.

Es: Le auto condividono i tratti comuni di tutti i veicoli. La capacità di trasportare persone da un luogo all'altro

Ereditarietà è Polimorfismo: Riusare il software

- A volte si incontrano classi con funzionalità simili
In quanto sottendono concetti semanticamente “vicini”
- È possibile creare classi disgiunte replicando le porzioni di stato/comportamento condivise
- L’approccio “Taglia&Incolla”, però, non è una strategia vincente Difficoltà di manutenzione correttiva e perfezionativa
- Meglio “specializzare” codice funzionante
 - Sostituendo il minimo necessario

Ereditarietà

Meccanismo per definire una nuova classe (classe derivata) come specializzazione di un'altra (classe base)

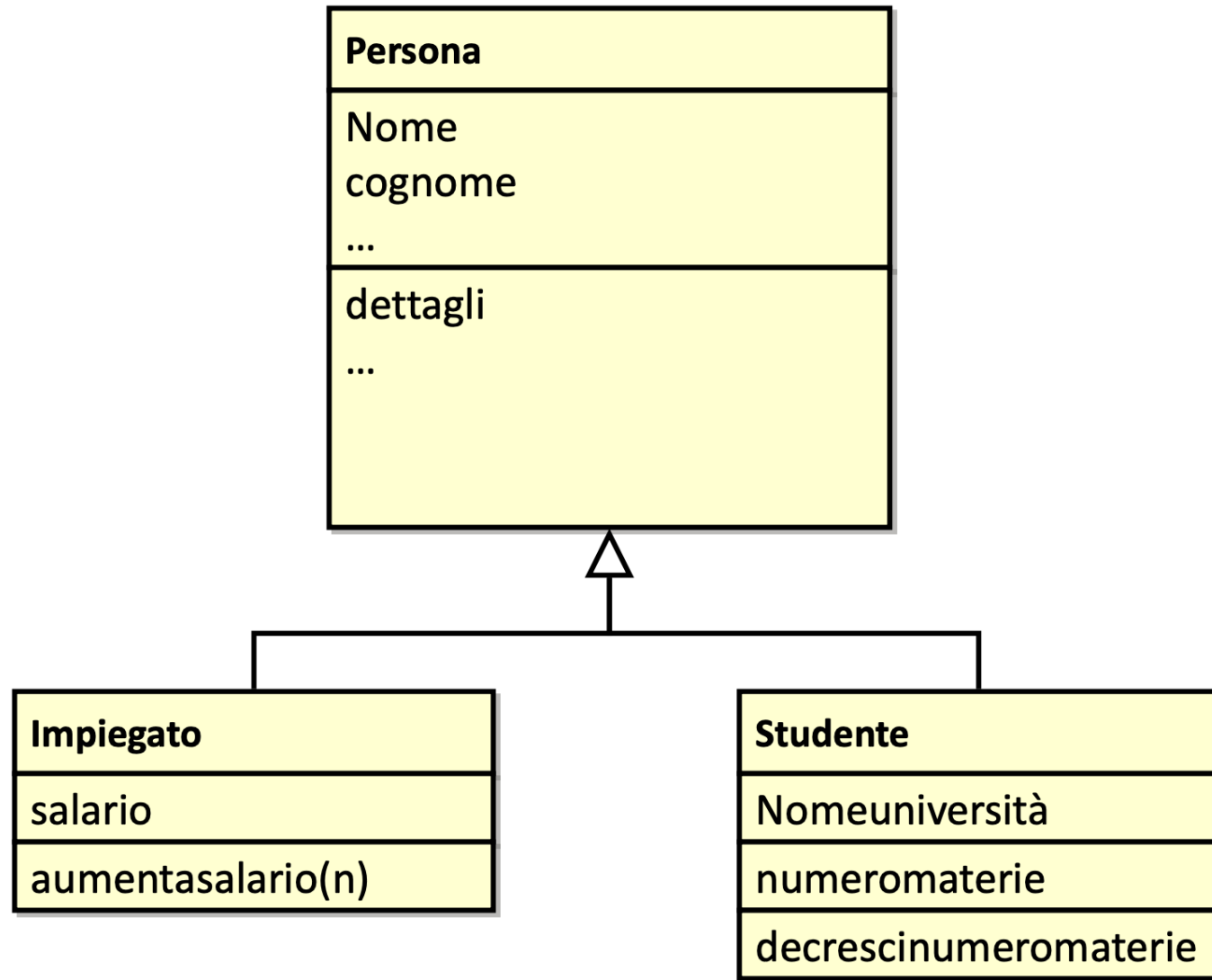
- La classe base modella un concetto generico
- La classe derivata modella un concetto più specifico
- La classe derivata:
 - Dispone di tutte le funzionalità (attributi e metodi) di quella base
 - Può aggiungere funzionalità proprie
 - Può ridefinirne il funzionamento di metodi esistenti (polimorfismo)

La classe derivata:

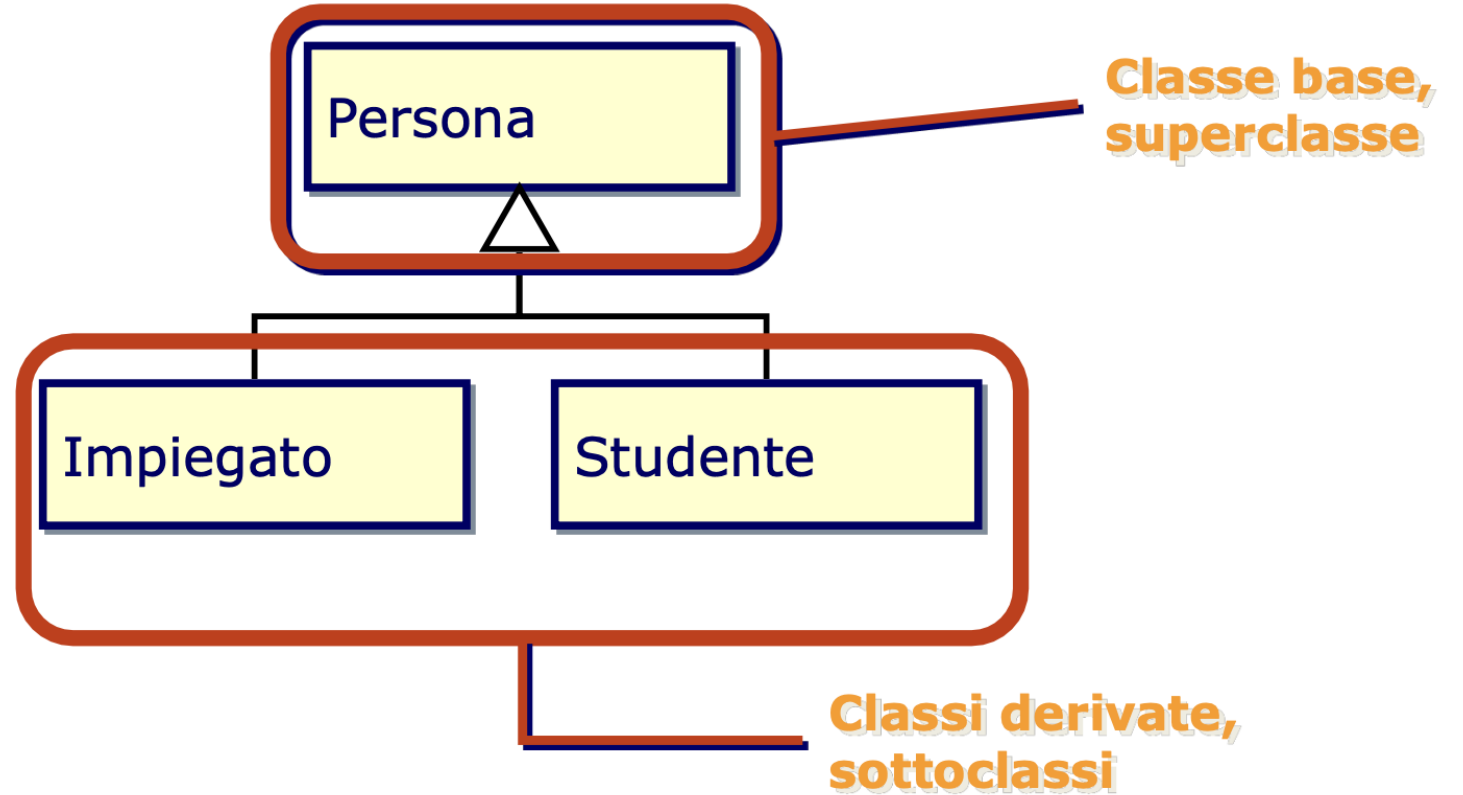
La classe derivata:

- Dispone di tutte le funzionalità (attributi e metodi) di quella base
- Può aggiungere funzionalità proprie
- Può ridefinirne il funzionamento di metodi esistenti (polimorfismo)

ESEMPIO



Terminologia



Astrazione

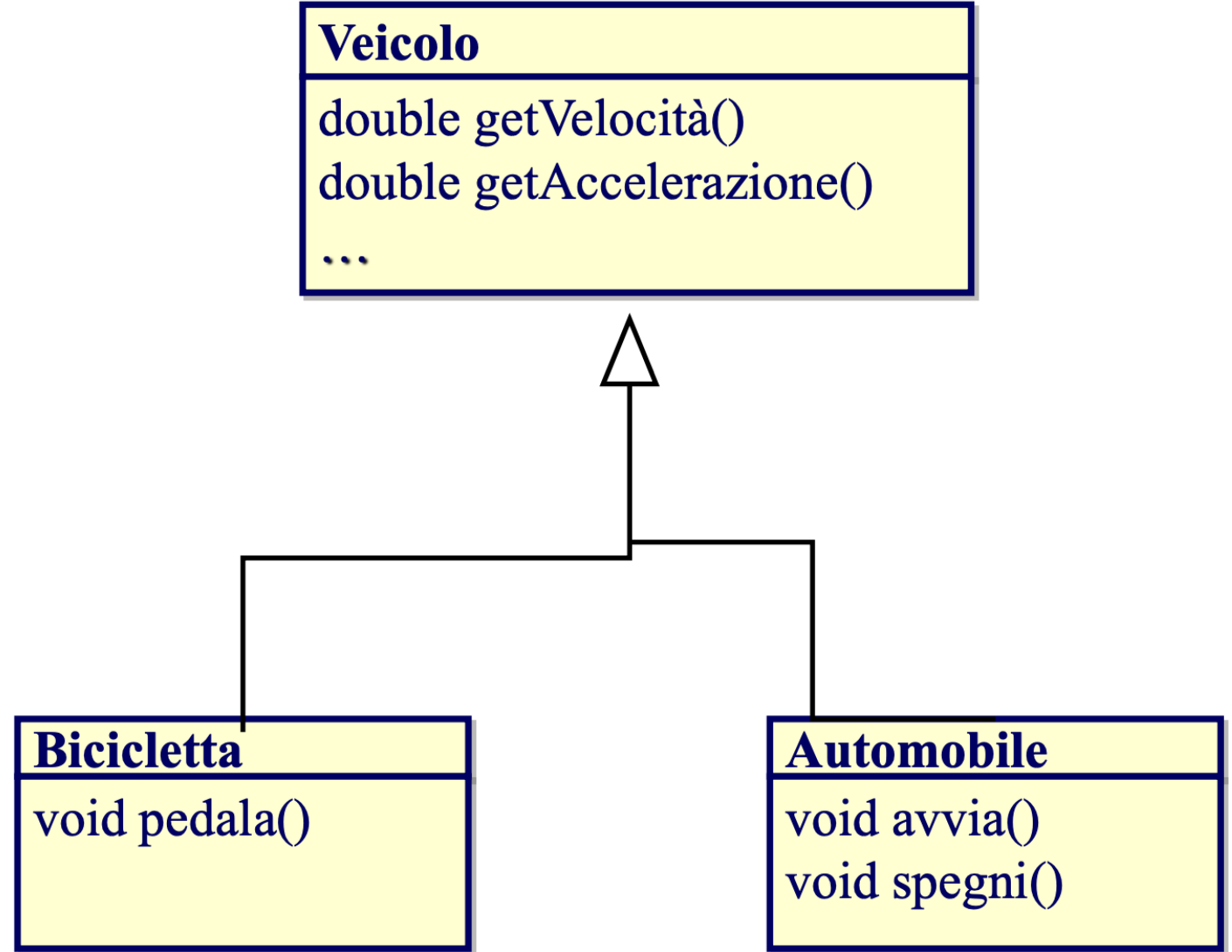
- Il processo di analisi e progettazione del software di solito procede per raffinamenti successivi
- Spesso capita che le similitudini tra classi non siano colte inizialmente
- In una fase successiva, si coglie l'esigenza/opportunità di introdurre un concetto più generico da cui derivare classi specifiche

Astrazione

Processo di astrazione:

- Si introduce la superclasse che “astraе” il concetto comune condiviso dalle diverse sottoclassi
- Le sottoclassi vengono “spogliate” delle funzionalità comuni che migrano nella superclasse

ESEMPIO



Tipi ed ereditarietà

Ogni classe definisce un tipo:

- Un oggetto, istanza di una sotto-classe, è formalmente compatibile con il tipo della classe base
- **Il contrario non è vero!**

Esempio

- Un'automobile è un veicolo
- Un veicolo non è (necessariamente) un'automobile

Tipi ed ereditarietà

La compatibilità diviene effettiva se

- I metodi ridefiniti nella sotto-classe rispettano la semantica della superclasse

L'ereditarietà gode delle proprietà transitiva

- Un tandem è un veicolo (poichè è una bicicletta, che a sua volta è un veicolo)

Vantaggi

- Evitare la duplicazione di codice
- Permettere il riuso di funzionalità
- Semplificare la costruzione di nuove classi
- Facilitare la manutenzione
- Garantire la consistenza delle interfacce

Ereditarietà in Java

Si definisce una classe derivata attraverso la parola chiave “extends”

- Seguita dal nome della classe base

Gli oggetti della classe derivata sono, a tutti gli effetti, estensioni della classe base

- Anche nella loro rappresentazione in memoria

```
public class Veicolo {  
    private double velocità;  
    private double accelerazione;  
    public double getVelocità() {...}  
    public double getAccelerazione() {...}  
}
```

Veicolo.java

```
public class Automobile  
    extends Veicolo {  
        private boolean avviata;  
        public void avvia() {...}  
}
```

Automobile.java

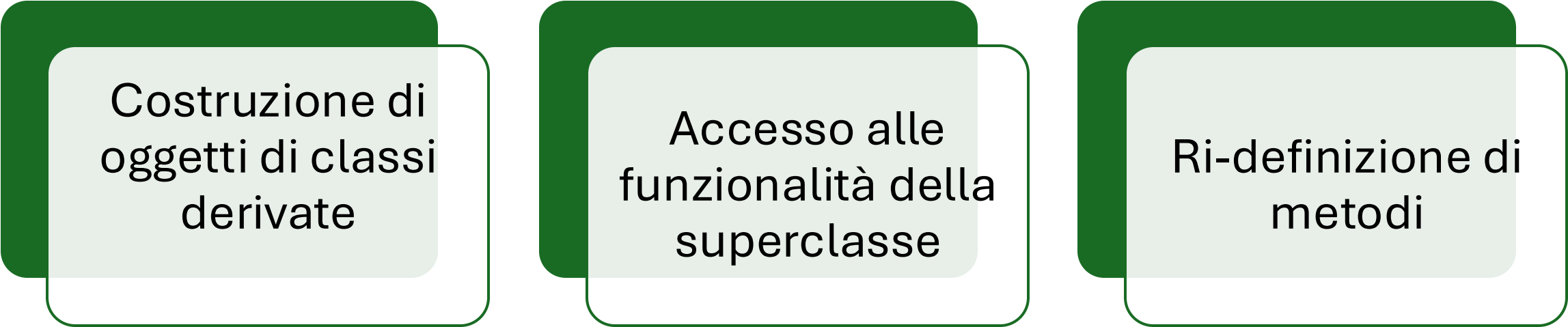
```
Automobile a=  
new Automobile();
```

a

Memoria

velocità: 0.0 accelerazione: 0.0
avviata: false

Meccanismi



Costruzione di
oggetti di classi
derivate

Accesso alle
funzionalità della
superclasse

Ri-definizione di
metodi

Costruttori

Per realizzare un'istanza di una classe derivata, occorre – innanzitutto – costruire l'oggetto base

- Di solito, provvede automaticamente il compilatore, invocando, come prima operazione di ogni costruttore della classe derivata, il costruttore anonimo della superclasse
- Si può effettuare in modo esplicito, attraverso il costrutto `super(...)`
- Eventuali ulteriori inizializzazioni possono essere effettuate solo successivamente

```
class Impiegato {  
    String nome;  
    double stipendio;
```

```
    Impiegato(String n) {  
        nome = n;  
        stipendio = 1500;  
    }  
}
```

```
class Funzionario  
    extends Impiegato {
```

```
    Funzionario(String n) {  
        super(n);  
        stipendio = 2000;  
    }  
}
```

Accedere alla superclasse

L'oggetto derivato contiene tutti i componenti (attributi e metodi) dell'oggetto da cui deriva

- Ma i suoi metodi non possono operare direttamente su quelli definiti privati
- La restrizione può essere allentata:
 - La super-classe può definire attributi e metodi con visibilità “protected”
 - Questi sono visibili alle sottoclassi

Ridefinire i metodi

Una sottoclasse può ridefinire metodi presenti nella superclasse

A condizione che abbiano

- Lo stesso nome
- Gli stessi parametri (tipo, numero, ordine)
- Lo stesso tipo di ritorno
- (La stessa semantica!)

Per le istanze della sottoclasse, il nuovo metodo nasconde l'originale

```
class Base {  
    int m() {  
        return 0;  
    }  
}
```

```
class Derivata  
    extends Base {  
        int m() {  
            return 1;  
        }  
}
```

```
Base b= new Base();  
System.out.println(b.m());  
Derivata d= new Derivata();  
System.out.println(d.m());
```

Metodi

- A volte, una sottoclasse vuole “perfezionare” un metodo ereditato, non sostituirlo in toto
- Per invocare l’implementazione presente nella super-classe, si usa il costrutto `super.<nomeMetodo> (...)`

```
class Base {  
    int m() {  
        return 0;  
    }  
}
```

```
class Derivata  
    extends Base {  
    int m() {  
        return super.m()+ 1;  
    }  
}
```

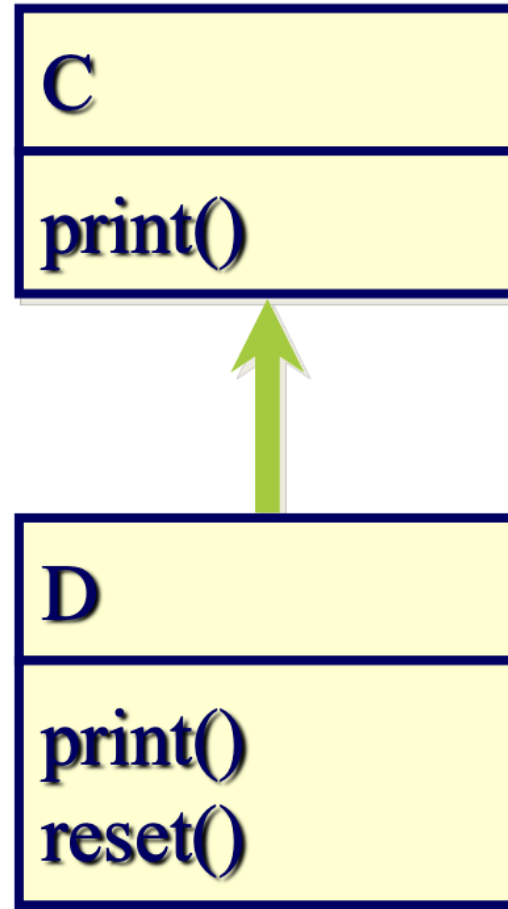
Compatibilità formale

Un'istanza di una classe derivata è formalmente compatibile con il tipo della super-classe

Base b = new Derivata();

- Il tipo della variabile “b” (Base) limita le operazioni che possono essere eseguite sull'oggetto contenuto
- Anche se questo ha una classe più specifica (Derivata), in grado di offrire un maggior numero di operazioni
- Altrimenti viene generato un errore di compilazione

Esempio



C v1= new **C**();

C v2= new **D**();

D v3= new **D**();

v1.print() ✓

v2.print() ✓

v2.reset() ✗

v3.reset() ✓

RECAP

L'**ereditarietà** in Java si riferisce alla possibilità di creare una nuova classe (chiamata **classe derivata** o **classe figlia**) a partire da una classe esistente (chiamata **classe base** o **classe genitore**). L'ereditarietà permette alla classe derivata di **riutilizzare** e **modificare** le proprietà e i comportamenti (metodi) definiti nella classe base. Questo facilita la creazione di una struttura di classi più organizzata e riduce la duplicazione del codice.

RECAP

Caratteristiche principali dell'ereditarietà in Java:

1. **Riutilizzo del codice:** La classe figlia eredita i membri (variabili e metodi) della classe genitore, evitando la duplicazione di codice.
2. **Estensione:** La classe figlia può aggiungere nuove proprietà e metodi che non sono presenti nella classe genitore, estendendo così il comportamento della classe base.
3. **Override:** La classe figlia può **modificare** i comportamenti ereditati dalla classe genitore attraverso la sovrascrittura dei metodi. Questo avviene mediante l'uso dell'annotazione `@Override`.
4. **Accesso ai membri della classe genitore:** I membri della classe genitore (variabili e metodi) possono essere accessibili nella classe figlia, ma con limitazioni di visibilità (ad esempio, i membri private non sono accessibili direttamente dalla classe figlia).
5. **Costruttori:** I costruttori della classe genitore non vengono ereditati dalla classe figlia, ma la classe figlia può invocare il costruttore della classe genitore usando `super()`.

Vantaggi e limiti dell'ereditarietà:

Vantaggi

1. **Modularità:** È possibile suddividere il codice in moduli più piccoli e più facili da gestire.
2. **Estensibilità:** Nuove funzionalità possono essere aggiunte senza modificare il codice esistente.
3. **Manutenibilità:** Cambiamenti nella classe base si riflettono automaticamente nelle classi derivate, facilitando la manutenzione del codice.

Limiti dell'ereditarietà:

1. **Rigida relazione:** L'ereditarietà crea una relazione rigida tra la classe genitore e quella derivata, che può rendere difficile apportare modifiche a una delle due senza influenzare l'altra.
2. **Possibili conflitti:** Sovrascrivere metodi o modificare comportamenti ereditati può portare a conflitti se non fatto con attenzione.

Polimorfismo

Java mantiene traccia della classe effettiva di un dato oggetto

- Seleziona sempre il metodo più specifico...
- ...anche se la variabile che lo contiene appartiene ad una classe più generica!

Una variabile generica può avere “molte forme”

- Contenere oggetti di sottoclassi differenti
- In caso di ridefinizione, il metodo chiamato dipende dal tipo effettivo dell'oggetto

Polimorfismo

Per sfruttare questa tecnica:

Si definiscono, nella super-classe, metodi con implementazione generica...

- ...sostituiti, nelle sottoclassi, da implementazioni specifiche
- Si utilizzano variabili aventi come tipo quello della super-classe

Meccanismo estremamente potente e versatile, alla base di molti “pattern” di programmazione

Polimorfismo

Il **polimorfismo** consente agli oggetti di essere trattati come istanze della loro classe base, pur comportandosi in modo diverso a seconda del contesto. In altre parole, il polimorfismo permette a un oggetto di assumere **molteplici forme**. Questo concetto si applica principalmente ai metodi e alle classi.

Il polimorfismo si manifesta in due principali forme:

- 1. Polimorfismo di inclusione (o sottotipizzazione):** Un oggetto di una classe derivata può essere trattato come se fosse un oggetto della sua classe base.
- 2. Polimorfismo di override (o sovrascrittura):** La classe derivata può fornire una versione specifica di un metodo che è già definito nella classe base.

Tipi di polimorfismo in Java:

- 1. Polimorfismo a tempo di compilazione (o statico) — Overloading:**
Questo tipo di polimorfismo si verifica quando più metodi con lo stesso nome vengono definiti nella stessa classe, ma con parametri diversi (tipo o numero). La scelta del metodo da chiamare avviene al momento della compilazione.
- 2. Polimorfismo a tempo di esecuzione (o dinamico) — Overriding:**
Questo tipo di polimorfismo si verifica quando una classe derivata fornisce una versione specifica di un metodo già definito nella classe base. La decisione su quale metodo invocare avviene al momento dell'esecuzione (runtime) in base al tipo effettivo dell'oggetto, non al tipo di riferimento.

Esercizio 1

Create il tipo di dato Impiegato come estensione del tipo di dato Persona.

- Dove una classe Persona ha le variabili nome, cognome, età. Ha i metodi GetNome, GetCognome, GetEtà e un metodo toString() che restituisce in una stringa le informazioni sulla persona in questione.
- Dove una classe Impiegato ha le variabili nome, cognome, età, salario. Ha un metodo toString() che restituisce in una stringa le informazioni sulla persona in questione.
Ha un metodo aumentasalario() che aumenti lo stipendio secondo una certa percentuale.

Esercizio 2

Si realizzi una applicazione java per la gestione di un garage secondo le specifiche:
il garage ha al max 15 posti ognuno dei quali è identificato da un num a partire da 0 e per motivi di capienza può ospitare solo auto moto e furgoni partendo dalla classe base veicolo a motore V; la si estenda, realizzando anche le classi che modellano le entità furgone (F) auto (A) e moto (M).

Ridefinire il metodo toString in modo che ogni entità possa esternalizzare in forma di stringa tutte le informazioni che la riguardano.

Si implementi una classe che modelli il garage sopradescritto offrendo le seguenti operazioni di gestione :

- immissione di un nuovo veicolo
- estrazione dal garage del veicolo che occupa un determinato posto (ritornare l'istanza del veicolo stesso)
- stampa della situazione corrente dei posti nel garage veicolo:
marca,anno,cilindrata;
auto:porte, alimentazione (diesel/benzina)
moto:tempi
furgone:capacità

Esercizio 2

Secondo le specifiche indicate, deve essere gestito un garage che ha al massimo 15 posti, ciascuno dotato di un identificatore numerico id, della marca del veicolo, dell'anno di fabbricazione e della cilindrata.

Questi, però, sono soltanto gli attributi comuni a i veicoli che possono essere ospitati.

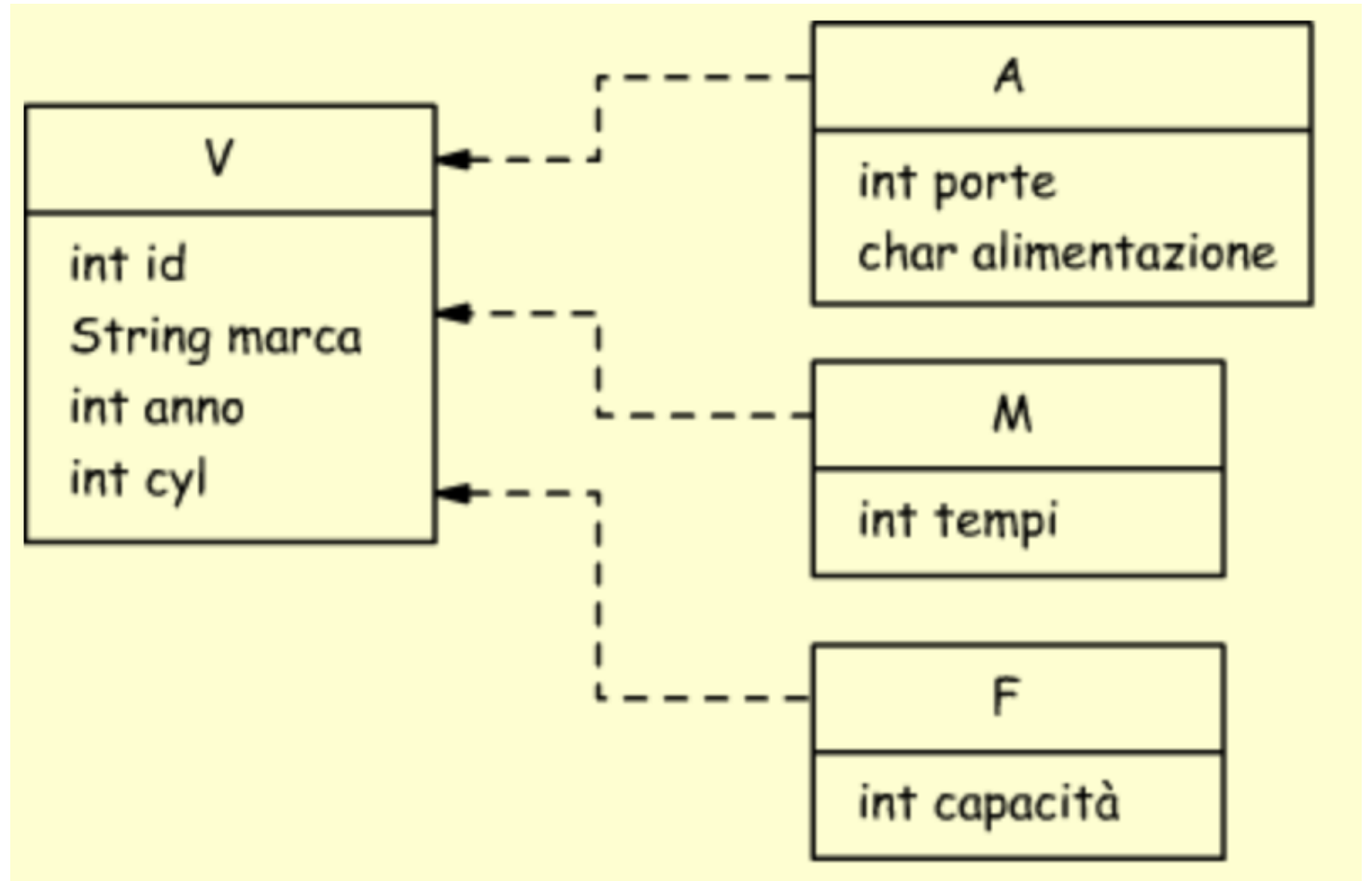
I veicoli possono essere di tre tipi: auto **[A]** furgone **[F]** e moto **[M]**.

Per le auto devono essere qualificati oltre agli attributi suddetti, il numero di porte(3/5) e l'alimentazione (diesel/benzina). Per le moto i tempi (2/4) e per i furgoni il carico.

Le entità presenti: furgone **[F]** auto **[A]** e moto (**[M]**) vengono derivate da una unica superclasse veicolo **[V]** che raccoglie gli attributi comuni cioè:

Esercizio 2

int id (identificatore del numero di posto)
String marca (azienda produttrice)
int anno (anno di fabbricazione)
int cyl (cilindrata)



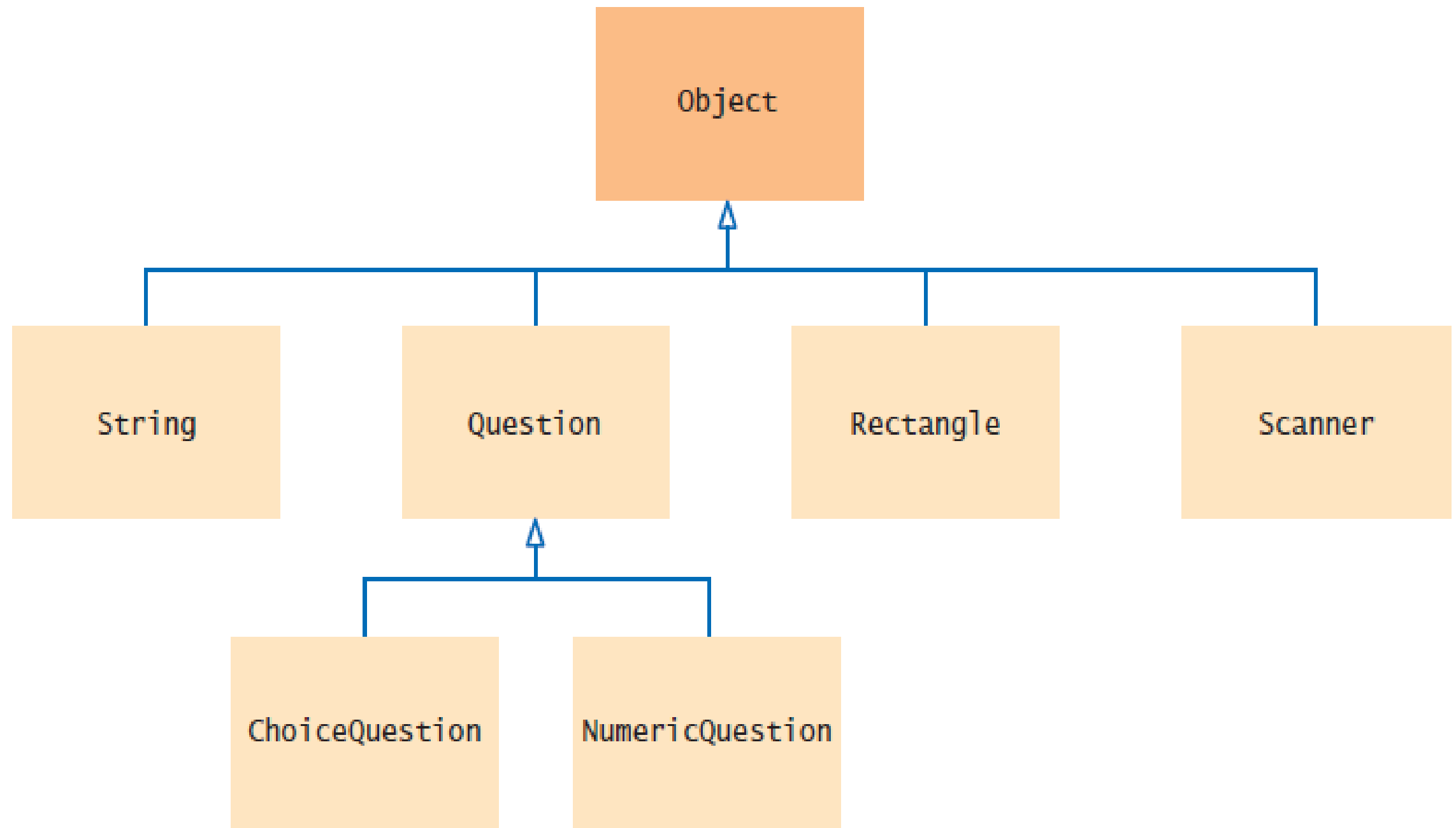
Classe Object

Object-Ereditarietà

- Ogni classe definita senza una clausola extends esplicita si estende automaticamente Object:
- La classe Object è la superclasse diretta o indiretta di ogni classe in Java.

Alcuni metodi definiti in Object:

- toString - che restituisce una stringa che descrive l'oggetto
- equals - che confronta gli oggetti tra loro
- hashCode - che fornisce un codice numerico per memorizzare l'oggetto in un set



Overriding toString()

- Restituisce una stringa che rappresenta l'oggetto
- Utile per il debug
- toString viene chiamato ogni volta che si concatena una stringa con un oggetto.
- Il compilatore può invocare il metodo toString, perché sa che ogni oggetto ha un metodo toString: Ogni classe estende la classe Object che dichiara toString

Overriding toString()

`Object.toString` stampa il nome della classe e il codice hash dell'oggetto

```
BankAccount momsSavings = new BankAccount(5000);
```

```
String s = momsSavings.toString();
```

```
// imposta s come tipo: "BankAccount@d24606bf"
```

Sovrascrivi il metodo `toString` nelle classi per produrre una stringa che descriva lo stato dell'oggetto.

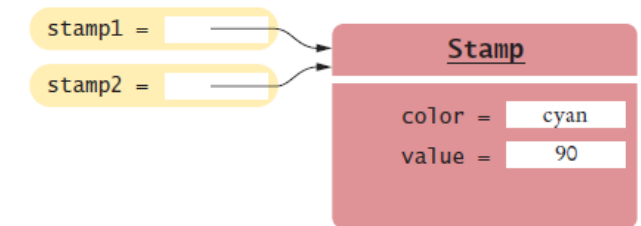
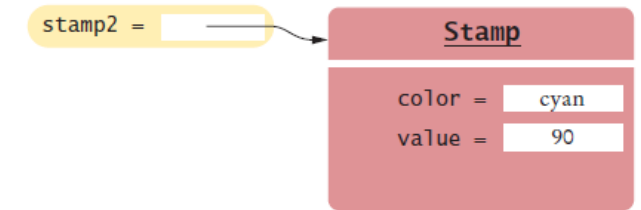
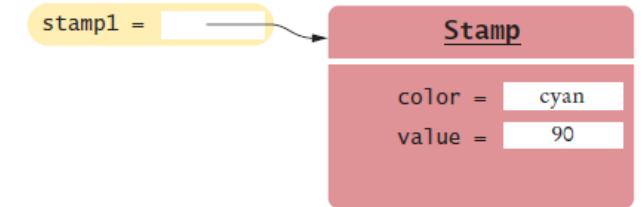
```
public String toString()  
{  
    return "BankAccount[balance=" + balance + " ]";  
}
```

Overriding equals



```
if (stamp1.equals(stamp2)) . .  
    // il contenuto è lo stesso
```

L'operatore == verifica se due riferimenti sono identici, facendo riferimento allo stesso oggetto.



Overriding equals

Per implementare il metodo equals per una classe Stamp

Sovrascrivi il metodo equals della classe Object:

```
public class Stamp{
    private String color;

    private int value;

    public boolean equals(Object otherObject) {
        . . .
    }
}
```

- Impossibile modificare il tipo di parametro del metodo equals - deve invece essere Object
Cast la variabile parametro alla classe Stamp: `Stamp other = (Stamp) otherObject;`

Overriding equals

```
public boolean equals(Object otherObject)

{

    Stamp other = (Stamp) otherObject;

    return color.equals(other.color) &&
        value==other.value;

}
```

Il metodo equals può accedere alle variabili di istanza di qualsiasi oggetto Stamp.
L'accesso a other.color è legale

Operatore instanceof

- È legale memorizzare un riferimento di sottoclasse in una variabile di superclasse:

```
ChoiceQuestion cq = new  
ChoiceQuestion();  
Question q = cq; // OK  
Object obj = cq; // OK
```

- A volte è necessario convertire da un riferimento di superclasse a un riferimento di sottoclasse. Se sai che una variabile di tipo Object contiene effettivamente un riferimento a Question, puoi eseguire il cast

```
Question q = (Question) obj;
```

Instanceof

Se obj fa riferimento a un oggetto di un tipo non correlato, viene generata un'eccezione "class cast". L'operatore instanceof verifica se un oggetto appartiene a un tipo particolare

```
obj instanceof Question
```

```
if (obj instanceof Question) {
```

```
    Question q = (Question) obj;
```

```
}
```