Ereditarietà

Teoria generale

Attributi : campi privati
Detti anche campi o variabili d'esemplare
- Information hiding
- = solo accessibili all'esterno tramite get()

Struttura classe:

- attributi (di istanza)
  - setter/getter
  - costruttori
- metodi
- `toString`
  - stampa i campi classe

Overloading (sovraccarico):

- stessa funzione con stessa firma ma con parametri diversi

Overriding (ridefinizione):

- ridefinizione della funzione (a livello logico)
- Classi astratte vs interfacce

Astratto = Per contratto fa operazioni di altri (le implementa = `implements`)

- Ridefinizione da parte delle sottoclassi
- La classe astratta fornisce comportamento
- Sotto viene riutilizzato alla bisogna

Interfaccia = Fa operazioni e le fornisce

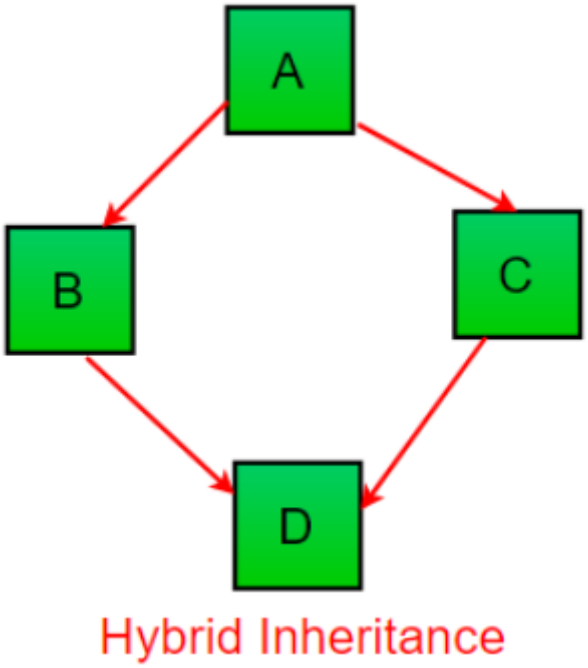
```
//Creating interface that has 4 methods
interface A{
void a();//bydefault, public and abstract
void b();
void c();
void d();
}

//Creating abstract class that provides the implementation
abstract class B implements A{
public void c(){System.out.println("I am C");}
}

//Creating subclass of abstract class, now we need to provide the implementatior
class M extends B{
public void a(){System.out.println("I am a");}
public void b(){System.out.println("I am b");}
public void d(){System.out.println("I am d");}
}

//Creating a test class that calls the methods of A interface
class Test5{
public static void main(String args[]){
A a=new M();
a.a();
a.b();
a.c();
a.d();
```

- Is-A
  - Macchina è un Veicolo
  - = è sottoclasse

- Has-A
  - Macchina ha un Motore
  - = usa funzionalità

Ereditarietà multipla = Diamond problem

- Non c'è in Java
- Tutti le classi sono virtuali
  - Permettono ridefinizione



- Polimorfismo
  - `Parent p = new Child();`
  - `Shape s = new Square();`
  - `Shape s = new Circle();`

UML

- Visibilità
  - private `-`
  - protected `#`
    - visibile solo a sottoclassi oppure classi del package
  - public `+`
- Astratta

    - Corsivo
- Array:
    - `- listOfRegistered : Employee[0..*]`

```
public class Class{
        int x;

        // setter
        void setX(int x1){
                x1 = x;
        }

        //costruttore = simile all'altro
        Class(){
                this.x = x;
        }

        int getX(){
                return x;
        }
}

public class Subclass extends Class{
        // overriding

        @Override
        int getX(){
                // super();
                x = 5;
        }

        // overloading
        int getX(int x2){
                return x + x2;
        }
}

public class Interface implements Class{

};

public class abstract Shape{
```

```
        private:
                int meters;
        public:
                void shape();
};

public class Triangle extends Shape{
        public:
                @Override
                void draw();
}

public interface Drawable{
        void draw();
}
```

Riferimenti

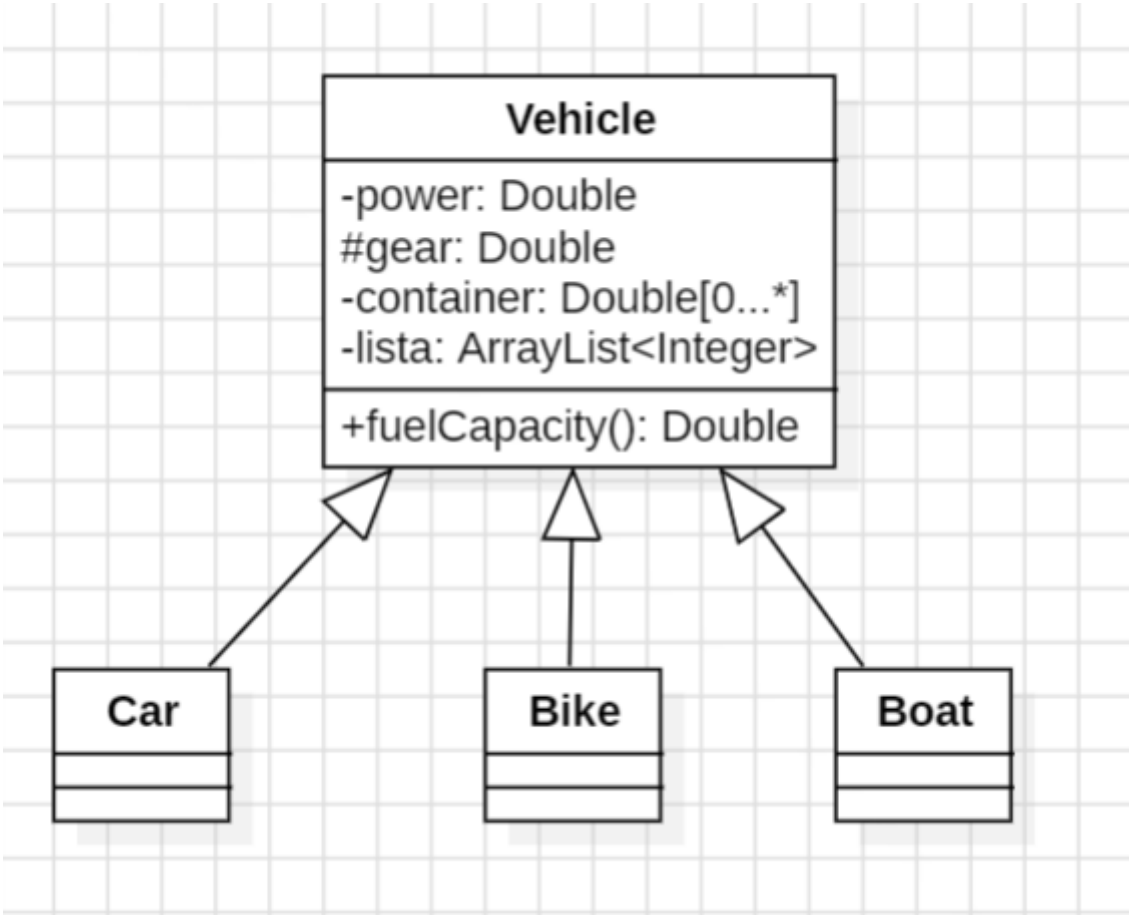
Inheritance (The Java™ Tutorials > Learning the Java Language > Interfaces and Inheritance) (oracle.com)

Array

- Creazione
  - `num = new int[10];`
  - `int x[] = new int[10];`
  - `String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
- Vengono creati dinamicamente con `new`
- Vengono distrutti automaticamente
- Operazioni *con cicli per tutti gli elementi*
  - Inserimento
  - Lettura dei valori
  - Modifica
  - Cancellazione

```
for(int i: vettore){
```

```
}

for(int i = 0; i < vettore.length; i++){

}
```



Generalizzazione = Ereditarietà = Frecce bianche