Ereditarietà e Polimorfismo 29-04

Ereditarietà

- **Riutilizzo del software**: Consente di creare nuove classi (classi derivate) come specializzazione di un'altra classe esistente (classe base).
- **Astrazione**: Introdurre una superclasse che astrae il concetto comune condiviso dalle sottoclassi.
- **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

Sintassi

- Utilizzo della parola chiave `extends` per definire una classe derivata:

```
public class ClasseDerivata extends ClasseBase {
  // ...
}
```

Meccanismi

- **Costruzione di oggetti di classi derivate**: Uso del costrutto `super( ... )` per invocare il costruttore della superclasse.
- **Accesso alle funzionalità della superclasse**: Le sottoclassi ereditano gli attributi e i metodi della superclasse, ma non possono accedere direttamente a quelli privati. La superclasse può definire membri `protected` per renderli accessibili alle sottoclassi.
- **Ridefinizione di metodi**: Una sottoclasse può ridefinire i metodi della superclasse, a condizione che abbiano la stessa firma. Per invocare l'implementazione della superclasse, utilizzare `super.<nomeMetodo>( ... )`.

Polimorfismo

- **Compatibilità formale**: Un'istanza di una classe derivata è formalmente compatibile con il tipo della superclasse.
- **Legame dinamico**: Java mantiene traccia della classe effettiva di un oggetto e seleziona sempre il metodo più specifico, anche se la variabile che lo contiene appartiene a una classe più generica.
- **Implementazione**: Definire metodi con implementazione generica nella superclasse e sostituirli con implementazioni specifiche nelle sottoclassi. Utilizzare variabili del tipo della superclasse per sfruttare il polimorfismo.

Esercizi da sapere

Veicoli in garage

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

```
import java.util.Scanner;

class Veicolo {
    protected int id;
```

```
    protected String marca;
    protected int anno;
    protected int cilindrata;

    public Veicolo(int id, String marca, int anno, int cilindrata) {
        this.id = id;
        this.marca = marca;
        this.anno = anno;
        this.cilindrata = cilindrata;
    }

    @Override
    public String toString() {
        return id + "] " + marca + " " + anno + " " + cilindrata;
    }
}

class Auto extends Veicolo {
    private int porte;
    private char alimentazione; // 'd' per diesel, 'b' per benzina

    public Auto(int id, String marca, int anno, int cilindrata, int porte,
char alimentazione) {
        super(id, marca, anno, cilindrata);
        this.porte = porte;
        this.alimentazione = alimentazione;
    }

    @Override
    public String toString() {
        return super.toString() + " " + porte + " " + alimentazione;
    }
}

class Moto extends Veicolo {
    private int tempi;

    public Moto(int id, String marca, int anno, int cilindrata, int tempi)
{
        super(id, marca, anno, cilindrata);
        this.tempi = tempi;
    }
```

```
    @Override
    public String toString() {
        return super.toString() + " " + tempi;
    }
}

class Furgone extends Veicolo {
    private int capacita;

    public Furgone(int id, String marca, int anno, int cilindrata, int
capacita) {
        super(id, marca, anno, cilindrata);
        this.capacita = capacita;
    }

    @Override
    public String toString() {
        return super.toString() + " " + capacita;
    }
}

class Garage {
    private static final int MAX_POSTI = 15;
    private Veicolo[] posti;
    private int numVeicoli;

    public Garage() {
        posti = new Veicolo[MAX_POSTI];
        numVeicoli = 0;
    }

    public void inserisciVeicolo(Veicolo veicolo) {
        if (numVeicoli < MAX_POSTI) {
            posti[numVeicoli] = veicolo;
            numVeicoli++;
        } else {
            System.out.println("Garage pieno!");
        }
    }

    public Veicolo estraiVeicolo(int posto) {
        if (posto ≥ 0 && posto < numVeicoli) {
            Veicolo veicolo = posti[posto];
```

```
            posti[posto] = null;
            ricompattiPosti(posto);
            numVeicoli--;
            return veicolo;
        } else {
            System.out.println("Posto non valido!");
            return null;
        }
    }

    private void ricompattiPosti(int startIndex) {
        for (int i = startIndex; i < numVeicoli; i++) {
            posti[i] = posti[i + 1];
        }
    }

    public void stampaSituazione() {
        for (int i = 0; i < numVeicoli; i++) {
            System.out.println(posti[i]);
        }
    }
}

public class GestioneGarage {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Garage garage = new Garage();

        char scelta;
        do {
            System.out.print("Inserire veicolo (a/m/f): ");
            char tipo = scanner.nextLine().charAt(0);
            System.out.print("Marca: ");
            String marca = scanner.nextLine();
            System.out.print("Anno: ");
            int anno = scanner.nextInt();
            scanner.nextLine(); // Consumare il carattere di newline
            System.out.print("Cilindrata: ");
            int cilindrata = scanner.nextInt();
            scanner.nextLine(); // Consumare il carattere di newline

            switch (tipo) {
                case 'a':
```

```
                    System.out.print("Numero porte: ");
                    int porte = scanner.nextInt();
                    scanner.nextLine(); // Consumare il carattere di
newline
                    System.out.print("Alimentazione (d/b): ");
                    char alimentazione = scanner.nextLine().charAt(0);
                    Auto auto = new Auto(garage.numVeicoli, marca, anno,
cilindrata, porte, alimentazione);
                    garage.inserisciVeicolo(auto);
                    break;
                case 'm':
                    System.out.print("Tempi: ");
                    int tempi = scanner.nextInt();
                    scanner.nextLine(); // Consumare il carattere di
newline
                    Moto moto = new Moto(garage.numVeicoli, marca, anno,
cilindrata, tempi);
                    garage.inserisciVeicolo(moto);
                    break;
                case 'f':
                    System.out.print("Capacità di carico: ");
                    int capacita = scanner.nextInt();
                    scanner.nextLine(); // Consumare il carattere di
newline
                    Furgone furgone = new Furgone(garage.numVeicoli,
marca, anno, cilindrata, capacita);
                    garage.inserisciVeicolo(furgone);
                    break;
                default:
                    System.out.println("Tipo di veicolo non valido!");
            }

            garage.stampaSituazione();

            System.out.print("Continuare? (s/n) ");
            scelta = scanner.nextLine().charAt(0);
        } while (scelta == 's');

        System.out.print("Inserire il posto del veicolo da estrarre: ");
        int posto = scanner.nextInt();
        Veicolo veicolo = garage.estraiVeicolo(posto);
        if (veicolo != null) {
            System.out.println("Veicolo estratto: " + veicolo);
```

```
        }

        scanner.close();
    }
}
```

Gestione universitaria

Si vuole realizzare un'applicazione per la gestione di un'università. L'università offre diversi corsi, e ogni corso può avere più classi. Ogni classe è composta da uno o più studenti.

Implementare le seguenti classi:

1. `Persona`: classe base che rappresenta una persona, con attributi come nome, cognome e data di nascita.
2. `Studente`: classe derivata da `Persona`, che rappresenta uno studente. Aggiunge attributi come il numero di matricola e il corso di studi.
3. `Corso`: classe che rappresenta un corso di studi, con attributi come il nome del corso e la durata in anni.
4. `Classe`: classe che rappresenta una classe di un corso, con attributi come l'anno accademico e una lista di studenti iscritti.
5. `Università`: classe principale che gestisce l'insieme di corsi offerti e le relative classi.

L'applicazione deve permettere di:

1. Aggiungere un nuovo corso all'università
2. Aggiungere una nuova classe a un corso esistente
3. Iscrivere un nuovo studente a una classe
4. Stampare la lista di studenti iscritti a una determinata classe
5. Stampare la lista di corsi offerti dall'università

```
import java.time.LocalDate;
import java.util.Scanner;

// Classe base per rappresentare una persona
class Persona {
    protected String nome;
    protected String cognome;
```

```
    protected LocalDate dataNascita;

    public Persona(String nome, String cognome, LocalDate dataNascita) {
        this.nome = nome;
        this.cognome = cognome;
        this.dataNascita = dataNascita;
    }

    public String getNomeCompleto() {
        return nome + " " + cognome;
    }
}

// Classe derivata per rappresentare uno studente
class Studente extends Persona {
    private int matricola;
    private String corso;

    public Studente(String nome, String cognome, LocalDate dataNascita,
int matricola, String corso) {
        super(nome, cognome, dataNascita);
        this.matricola = matricola;
        this.corso = corso;
    }

    public int getMatricola() {
        return matricola;
    }

    public String getCorso() {
        return corso;
    }
}

// Classe per rappresentare un corso di studi
class Corso {
    private String nome;
    private int durata;
    private Classe[] classi;
    private int numClassi;

    public Corso(String nome, int durata, int maxClassi) {
        this.nome = nome;
```

```
        this.durata = durata;
        this.classi = new Classe[maxClassi];
        this.numClassi = 0;
    }

    public String getNome() {
        return nome;
    }

    public int getDurata() {
        return durata;
    }

    public Classe[] getClassi() {
        return classi;
    }

    public void aggiungiClasse(Classe classe) {
        if (numClassi < classi.length) {
            classi[numClassi] = classe;
            numClassi++;
        } else {
            System.out.println("Impossibile aggiungere nuove classi.
Limite raggiunto.");
        }
    }
}

// Classe per rappresentare una classe di un corso
class Classe {
    private int annoAccademico;
    private Studente[] studenti;
    private int numStudenti;

    public Classe(int annoAccademico, int maxStudenti) {
        this.annoAccademico = annoAccademico;
        this.studenti = new Studente[maxStudenti];
        this.numStudenti = 0;
    }

    public int getAnnoAccademico() {
        return annoAccademico;
    }
```

```
    public Studente[] getStudenti() {
        return studenti;
    }

    public void iscriviStudente(Studente studente) {
        if (numStudenti < studenti.length) {
            studenti[numStudenti] = studente;
            numStudenti++;
        } else {
            System.out.println("Impossibile iscrivere nuovi studenti.
Limite raggiunto.");
        }
    }
}

// Classe principale per gestire l'università
public class GestioneUniversita {
    private static final int MAX_CORSI = 10;
    private static final int MAX_CLASSI_PER_CORSO = 5;
    private static final int MAX_STUDENTI_PER_CLASSE = 30;

    private static Corso[] corsi = new Corso[MAX_CORSI];
    private static int numCorsi = 0;
    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        int scelta;
        do {
            System.out.println("\nScegli un'opzione:");
            System.out.println("1. Aggiungi corso");
            System.out.println("2. Aggiungi classe");
            System.out.println("3. Iscrivi studente");
            System.out.println("4. Stampa studenti di una classe");
            System.out.println("5. Stampa corsi offerti");
            System.out.println("0. Esci");
            scelta = scanner.nextInt();
            scanner.nextLine(); // Consuma il carattere di newline

            switch (scelta) {
                case 1:
                    aggiungiCorso();
                    break;
```

```
                case 2:
                    aggiungiClasse();
                    break;
                case 3:
                    iscriviStudente();
                    break;
                case 4:
                    stampaStudentiClasse();
                    break;
                case 5:
                    stampaCorsi();
                    break;
                case 0:
                    System.out.println("Uscita ... ");
                    break;
                default:
                    System.out.println("Scelta non valida!");
            }
        } while (scelta != 0);
    }

    private static void aggiungiCorso() {
        if (numCorsi == MAX_CORSI) {
            System.out.println("Impossibile aggiungere nuovi corsi. Limite
raggiunto.");
            return;
        }

        System.out.print("Nome del corso: ");
        String nome = scanner.nextLine();
        System.out.print("Durata (anni): ");
        int durata = scanner.nextInt();
        scanner.nextLine(); // Consuma il carattere di newline

        Corso corso = new Corso(nome, durata, MAX_CLASSI_PER_CORSO);
        corsi[numCorsi] = corso;
        numCorsi++;
        System.out.println("Corso aggiunto con successo.");
    }

    private static void aggiungiClasse() {
        if (numCorsi == 0) {
            System.out.println("Nessun corso disponibile.");
```

```
            return;
        }

        System.out.println("Corsi disponibili:");
        for (int i = 0; i < numCorsi; i++) {
            System.out.println((i + 1) + ". " + corsi[i].getNome());
        }
        System.out.print("Seleziona il corso: ");
        int sceltaCorso = scanner.nextInt();
        scanner.nextLine(); // Consuma il carattere di newline

        if (sceltaCorso < 1 || sceltaCorso > numCorsi) {
            System.out.println("Scelta non valida.");
            return;
        }

        Corso corso = corsi[sceltaCorso - 1];
        System.out.print("Anno accademico: ");
        int annoAccademico = scanner.nextInt();
        scanner.nextLine(); // Consuma il carattere di newline

        Classe classe = new Classe(annoAccademico,
MAX_STUDENTI_PER_CLASSE);
        corso.aggiungiClasse(classe);
        System.out.println("Classe aggiunta con successo.");
    }

    private static void iscriviStudente() {
        if (numCorsi == 0) {
            System.out.println("Nessun corso disponibile.");
            return;
        }

        System.out.print("Nome studente: ");
        String nome = scanner.nextLine();
        System.out.print("Cognome studente: ");
        String cognome = scanner.nextLine();
        System.out.print("Data di nascita (aaaa-mm-gg): ");
        LocalDate dataNascita = LocalDate.parse(scanner.nextLine());
        System.out.print("Matricola: ");
        int matricola = scanner.nextInt();
        scanner.nextLine(); // Consuma il carattere di newline
```

```
        System.out.println("Corsi disponibili:");
        for (int i = 0; i < numCorsi; i++) {
            System.out.println((i + 1) + ". " + corsi[i].getNome());
        }
System.out.print("Seleziona il corso: ");
int sceltaCorso = scanner.nextInt();
scanner.nextLine(); // Consuma il carattere di newline

if (sceltaCorso < 1 || sceltaCorso > numCorsi) {
    System.out.println("Scelta non valida.");
    return;
}

Corso corso = corsi[sceltaCorso - 1];
Classe[] classiCorso = corso.getClassi();

if (classiCorso == null || classiCorso.length == 0) {
    System.out.println("Nessuna classe disponibile per il corso
selezionato.");
    return;
}

System.out.println("Classi disponibili per il corso " + corso.getNome() +
":");
for (int i = 0; i < classiCorso.length; i++) {
    if (classiCorso[i] ≠ null) {
        System.out.println((i + 1) + ". Anno accademico: " +
classiCorso[i].getAnnoAccademico());
    }
}
System.out.print("Seleziona la classe: ");
int sceltaClasse = scanner.nextInt();
scanner.nextLine(); // Consuma il carattere di newline

if (sceltaClasse < 1 || sceltaClasse > classiCorso.length ||
classiCorso[sceltaClasse - 1] == null) {
    System.out.println("Scelta non valida.");
    return;
}

Classe classe = classiCorso[sceltaClasse - 1];
Studente studente = new Studente(nome, cognome, dataNascita, matricola,
corso.getNome());
```

```
classe.iscriviStudente(studente);
System.out.println("Studente iscritto con successo.");
}

private static void stampaStudentiClasse() {
    if (numCorsi == 0) {
        System.out.println("Nessun corso disponibile.");
        return;
    }

    System.out.println("Corsi disponibili:");
    for (int i = 0; i < numCorsi; i++) {
        System.out.println((i + 1) + ". " + corsi[i].getNome());
    }
    System.out.print("Seleziona il corso: ");
    int sceltaCorso = scanner.nextInt();
    scanner.nextLine(); // Consuma il carattere di newline

    if (sceltaCorso < 1 || sceltaCorso > numCorsi) {
        System.out.println("Scelta non valida.");
        return;
    }

    Corso corso = corsi[sceltaCorso - 1];
    Classe[] classiCorso = corso.getClassi();

    if (classiCorso == null || classiCorso.length == 0) {
        System.out.println("Nessuna classe disponibile per il corso
selezionato.");
        return;
    }

    System.out.println("Classi disponibili per il corso " +
corso.getNome() + ":");
    for (int i = 0; i < classiCorso.length; i++) {
        if (classiCorso[i] != null) {
            System.out.println((i + 1) + ". Anno accademico: " +
classiCorso[i].getAnnoAccademico());
        }
    }
    System.out.print("Seleziona la classe: ");
    int sceltaClasse = scanner.nextInt();
    scanner.nextLine(); // Consuma il carattere di newline
```

```
    if (sceltaClasse < 1 || sceltaClasse > classiCorso.length ||
classiCorso[sceltaClasse - 1] == null) {
        System.out.println("Scelta non valida.");
        return;
    }

    Classe classe = classiCorso[sceltaClasse - 1];
    Studente[] studentiClasse = classe.getStudenti();

    if (studentiClasse == null || studentiClasse.length == 0) {
        System.out.println("Nessuno studente iscritto a questa classe.");
        return;
    }

    System.out.println("Studenti iscritti alla classe dell'anno accademico
" + classe.getAnnoAccademico() + ":");
    for (Studente studente : studentiClasse) {
        if (studente ≠ null) {
            System.out.println("- " + studente.getNomeCompleto() + "
(Matricola: " + studente.getMatricola() + ")");
        }
    }
}

private static void stampaCorsi() {
    if (numCorsi == 0) {
        System.out.println("Nessun corso disponibile.");
        return;
    }

    System.out.println("Corsi offerti dall'università:");
    for (int i = 0; i < numCorsi; i++) {
        Corso corso = corsi[i];
        System.out.println((i + 1) + ". " + corso.getNome() + " (Durata: "
+ corso.getDurata() + " anni)");
    }
}
```

Esercizi utili

Esercizio 1: Creare una classe base `Figura` con un metodo `area()` che restituisce l'area della figura. Creare poi due classi derivate `Rettangolo` e `Cerchio` che

ridefiniscono il metodo `area()` per calcolare rispettivamente l'area di un rettangolo e di un cerchio. Scrivere un programma che crei un array di oggetti `Figura` contenente istanze di `Rettangolo` e `Cerchio`, e che stampi l'area di ciascuna figura utilizzando il polimorfismo.

```
class Figura {
    public double area() {
        return 0;
    }
}

class Rettangolo extends Figura {
    private double base;
    private double altezza;

    public Rettangolo(double base, double altezza) {
        this.base = base;
        this.altezza = altezza;
    }

    @Override
    public double area() {
        return base * altezza;
    }
}

class Cerchio extends Figura {
    private double raggio;

    public Cerchio(double raggio) {
        this.raggio = raggio;
    }

    @Override
    public double area() {
        return Math.PI * raggio * raggio;
    }
}

public class EsercizioFigure {
    public static void main(String[] args) {
        Figura[] figure = new Figura[4];
        figure[0] = new Rettangolo(4, 5);
```

```
        figure[1] = new Cerchio(3);
        figure[2] = new Rettangolo(6, 7);
        figure[3] = new Cerchio(2);

        for (Figura f : figure) {
            System.out.println("Area: " + f.area());
        }
    }
}
```

Esercizio 2: Creare una classe base `Veicolo` con attributi come marca, modello e anno di produzione. Creare poi due classi derivate `AutoVeicolo` e `Motoveicolo` che estendono `Veicolo` e aggiungono attributi specifici come il numero di porte per `AutoVeicolo` e il numero di cilindri per `Motoveicolo`. Implementare dei metodi per accedere e modificare gli attributi di ciascuna classe. Scrivere un programma che crei un array di oggetti `Veicolo` contenente istanze di `AutoVeicolo` e `Motoveicolo`, e che stampi le informazioni di ciascun veicolo utilizzando il polimorfismo.

```
class Veicolo {
    protected String marca;
    protected String modello;
    protected int annoProduzione;

    public Veicolo(String marca, String modello, int annoProduzione) {
        this.marca = marca;
        this.modello = modello;
        this.annoProduzione = annoProduzione;
    }

    public String getMarca() {
        return marca;
    }

    public String getModello() {
        return modello;
    }

    public int getAnnoProduzione() {
        return annoProduzione;
    }
}
```

```java
class AutoVeicolo extends Veicolo {
    private int numeroPorte;

    public AutoVeicolo(String marca, String modello, int annoProduzione,
int numeroPorte) {
        super(marca, modello, annoProduzione);
        this.numeroPorte = numeroPorte;
    }

    public int getNumeroPorte() {
        return numeroPorte;
    }

    public void setNumeroPorte(int numeroPorte) {
        this.numeroPorte = numeroPorte;
    }
}

class Motoveicolo extends Veicolo {
    private int numeroCilindri;

    public Motoveicolo(String marca, String modello, int annoProduzione,
int numeroCilindri) {
        super(marca, modello, annoProduzione);
        this.numeroCilindri = numeroCilindri;
    }

    public int getNumeroCilindri() {
        return numeroCilindri;
    }

    public void setNumeroCilindri(int numeroCilindri) {
        this.numeroCilindri = numeroCilindri;
    }
}

public class EsercizioVeicoli {
    public static void main(String[] args) {
        Veicolo[] veicoli = new Veicolo[4];
        veicoli[0] = new AutoVeicolo("Ford", "Mustang", 2020, 2);
        veicoli[1] = new Motoveicolo("Honda", "CBR600RR", 2018, 4);
        veicoli[2] = new AutoVeicolo("Toyota", "Corolla", 2022, 4);
        veicoli[3] = new Motoveicolo("Ducati", "Panigale V4", 2021, 4);
```

```
        for (Veicolo v : veicoli) {
            System.out.println("Marca: " + v.getMarca());
            System.out.println("Modello: " + v.getModello());
            System.out.println("Anno di produzione: " +
v.getAnnoProduzione());

            if (v instanceof AutoVeicolo) {
                AutoVeicolo auto = (AutoVeicolo) v;
                System.out.println("Numero di porte: " +
auto.getNumeroPorte());
            } else if (v instanceof Motoveicolo) {
                Motoveicolo moto = (Motoveicolo) v;
                System.out.println("Numero di cilindri: " +
moto.getNumeroCilindri());
            }

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