**1) Scrivi una classe studente e persona dove Studente è sottoclasse di Persona e scrivere i metodi equals e hashCode senza ripetizione di codice.**
**Dopo scrivi il metodo toString che stampa il nome della classe + il nome + ID progressivo (+Matricola)**

```java
public class Persona {

    private int id;
    private static int progId = 0;
    private String nome;

    public Persona(String nome) {
        this.nome = nome;
        this.id = progId++;
    }
    public Persona(String nome, int id) {
        this.nome = nome;
    }
    public int getId() {
        return this.id;
    }
    public String getNome() {
        return this.nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    @Override
    public String toString() {
        return this.getClass().getSimpleName() + " "+ this.getNome() + " "+ this.id ;
    }
    @Override
    public boolean equals(Object o) {
        if(o==null || this.getClass()!=o.getClass()) return false;
        Persona p = (Persona)o;
        return this.nome.equals(p.getNome());
    }
    @Override
    public int hashCode() {
        return this.getClass().hashCode()+this.getNome().hashCode();
    }
}
```

```java
public class Studente extends Persona {

    private String matricola;
    private int id;
    private static int progId = 0;

    public Studente(String nome, String matricola) {
        super(nome, 0);
        this.matricola = matricola;
        this.id = progId++;
    }

    @Override
    public int getId() {
        return this.id;
    }
    public String getMatricola() {
        return matricola;
    }
    public void setMatricola(String matricola) {
        this.matricola = matricola;
    }
    @Override
    public String toString() {
        return super.toString() + " " + this.getMatricola();
    }
    @Override
    public boolean equals(Object o) {
        return super.equals(o) && this.getMatricola().equals( ((Studente)o).getMatricola());
    }
    @Override
    public int hashCode() {
        return super.hashCode() + this.getMatricola().hashCode();
    }
}
```

**2) Scrivi una classe Cerchio caratterizzata da un centro e un raggio.**
**Scrivi un metodo equals su Punto e Cerchio + test per verificare l'equivalenza**

**Scrivi un'altra classe Ordinatore con due metodi che ritornano una lista di cerchi ordinati:**
**- ordinamento per raggio**
**- Ordinamento per posizione del centro**

```java
public class Cerchio implements Comparable<Cerchio>{

    private int raggio;
    private Coordinate centro;

    public Cerchio(int raggio, Coordinate centro) {
        this.raggio=raggio;
        this.centro=centro;
    }

    @Override
    public boolean equals(Object o) {
        Cerchio c = (Cerchio)o;
        return this.getCentro().equals(c.getCentro()) && this.getRaggio()==c.getRaggio();
    }
    @Override
    public int hashCode() {
        return this.getRaggio() + this.getCentro().hashCode();
    }

    @Override
    public int compareTo(Cerchio o) {
        return this.getRaggio()-o.getRaggio();
    }
```

```java
public class Coordinate implements Comparable<Coordinate>{

    private int x;
    private int y;

    public Coordinate(int x, int y) {
        this.x=x;
        this.y=y;
    }

    public int getX() {
        return x;
    }
    public int getY() {
        return y;
    }

    @Override
    public boolean equals(Object o) {
        Coordinate c = (Coordinate)o;
        return this.getX()==c.getX() && this.getY()==c.getY();
    }

    @Override
    public int hashCode() {
        return this.getX() + this.getY();
    }

    @Override
    public int compareTo(Coordinate c) {
        if(this.getX()-this.getX()==0)
            return this.getY()-c.getY();
        return this.getX()-c.getX();
    }
```

```java
public class Ordinatore {

    public static List<Cerchio> ordinaRaggi(List<Cerchio> lista){
        Collections.sort(lista);
        return lista;
    }

    public static List<Cerchio> ordinaCoordinate(List<Cerchio> lista){
        Collections.sort(lista, new Comparator<Cerchio>() {

            @Override
            public int compare(Cerchio c1, Cerchio c2) {
                return c1.getCentro().compareTo(c2.getCentro());
            }
        });
        return lista;
    }
}
```

**3) Scrivi una classe Cerchio e una classe Rettangolo che sono sottotipo comune di Forma (interface)**
**Scrivere un criterio di equivalenza tra due forme: quando due formi sono degneri collassano in un punto e Cerchio e Rettangolo possono divenire equivalenti. Scrivi un metodo isDegenere (booleano).**

```java
public class Cerchio implements Forma{

    private int raggio;
    private Coordinate centro;

    public Cerchio(int raggio, Coordinate centro) {
        this.raggio=raggio;
        this.centro=centro;
    }

    @Override
    public Coordinate getCoordinate() {
        return this.centro;
    }

    @Override
    public boolean isDegenere() {
        if(this.getRaggio()==0) return true;
        return false;
    }
    @Override
    public boolean equals(Object o) {
        Forma f = (Forma)o;
        if(isDegenere() && f.isDegenere())
            return this.getCoordinate().equals(f.getCoordinate());

        if(o==null || o.getClass()!=this.getClass()) return false;
        Cerchio that = (Cerchio)o;
        return this.getRaggio()==that.getRaggio() && this.getCoordinate().equals(that.getCoordinate());
    }

    @Override
    public int hashCode() {
        return this.getRaggio()+this.getCoordinate().hashCode();
    }

    public int getRaggio() {
```

```java
public interface Forma {

    public boolean isDegenere();

    public Coordinate getCoordinate();
}
```

```java
public class Rettangolo implements Forma {

    private int base;
    private int altezza;
    private Coordinate vertice;

    public Rettangolo(int base, int altezza, Coordinate vertice) {
        this.base=base;
        this.altezza=altezza;
        this.vertice=vertice;
    }

    @Override
    public Coordinate getCoordinate() {
        return vertice;
    }

    @Override
    public boolean isDegenere() {
        if(this.getAltezza()==0 && this.getBase()==0) return true;
        return false;
    }

    @Override
    public boolean equals(Object o) {
        Forma f = (Forma)o;
        if(isDegenere() && f.isDegenere())
            return this.getCoordinate().equals(f.getCoordinate());

        if(o==null || o.getClass()!=this.getClass()) return false;
        Rettangolo that = (Rettangolo)o;
        return this.getAltezza()==that.getAltezza() && this.getBase()==that.getBase()
                    && this.getCoordinate().equals(that.getCoordinate());
    }
    @Override
    public int hashCode() {
        return this.getAltezza()+this.getBase()+this.getCoordinate().hashCode();
```