Interfícies

Característiques

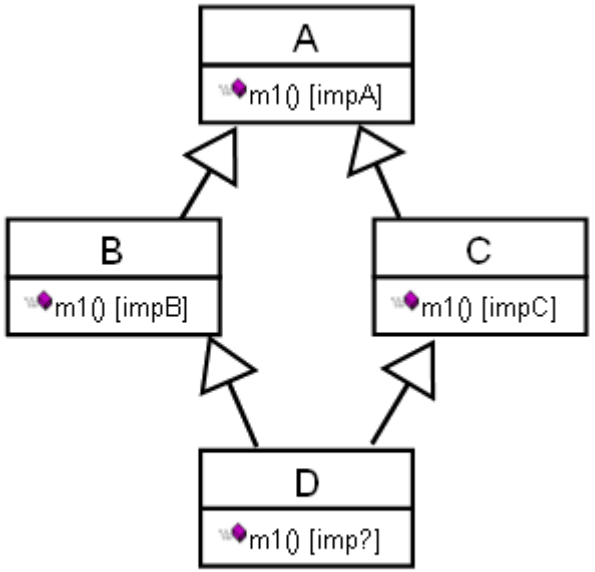
- Tots el mètodes d'una interfície són mètodes abstractes (signatures de mètodes sense implementació)

- Una classe concreta ha d'implementar (**implements**) la interfície, és a dir, implementar tots els mètodes

Característiques: Herència múltiple

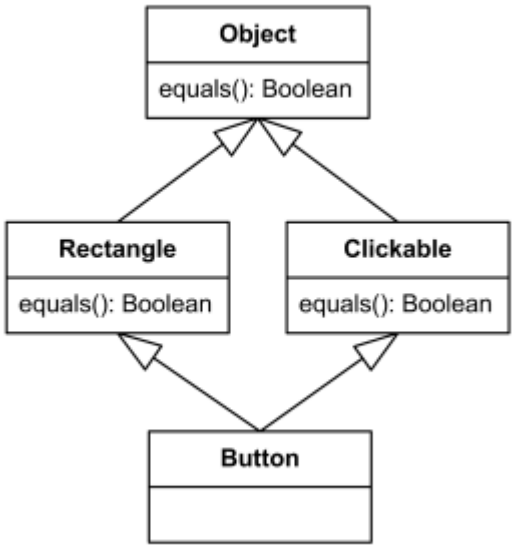
- Permet la implementació de classes amb comportaments comuns, sense importar la seva ubicació en la jerarquia de classes
- A Java només podem heredar d'una classe, però podem implementar tantes interfícies com vulguem.

Herència múltiple.El problema del diamant



Si tenim un mètode definit a A i sobreescrit a B i C, D quin hereta? El de B o el de C?

De fet, pot passar sempre:



- Com que els mètodes de les interfícies no estan implementats, mai es dóna el problema del diamant.

Exemple : Distància entre dos punts. La interfície

```
public interface Relation {
  public boolean isGreater(Object a, Object b);
  public boolean isLess(Object a, Object b);
  public boolean isEqual(Object a, Object b);
}
```

Exemple : Distància entre dos punts. La classe

```
/**
* Clase Line implements Relation interface
*/
public class Line implements Relation {
  private double x1;
  private double x2;
  private double y1;
  private double y2;
  public Line(double x1, double x2, double y1, double y2){
    this.x1 = x1;
    this.x2 = x2;
    this.y1 = y1;
    this.y2 = y2;
  }
```

Exemple : Distància entre dos punts. La classe

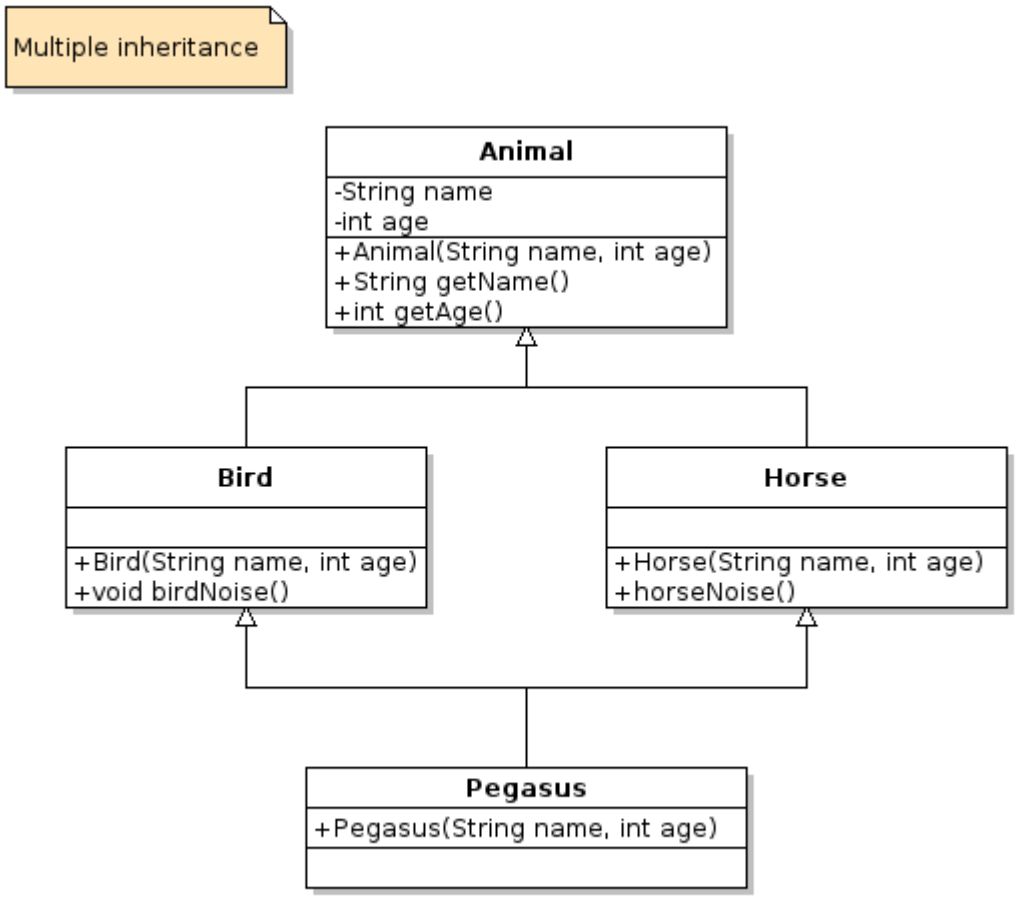
```
  public double getLength(){
    double length = Math.sqrt((x2-x1)*(x2-x1) +
                    (y2-y1)* (y2-y1));  return length;
  }
  public boolean isGreater( Object a, Object b){
    double aLen = ((Line)a).getLength();
    double bLen = ((Line)b).getLength(); return (aLen > bLen);
  }
  public boolean isLess( Object a, Object b){
    double aLen = ((Line)a).getLength();
    double bLen = ((Line)b).getLength(); return (aLen < bLen);
  }
  public boolean isEqual( Object a, Object b){
    double aLen = ((Line)a).getLength();
    double bLen = ((Line)b).getLength();
    return (aLen == bLen);
  }
}
```

Característiques

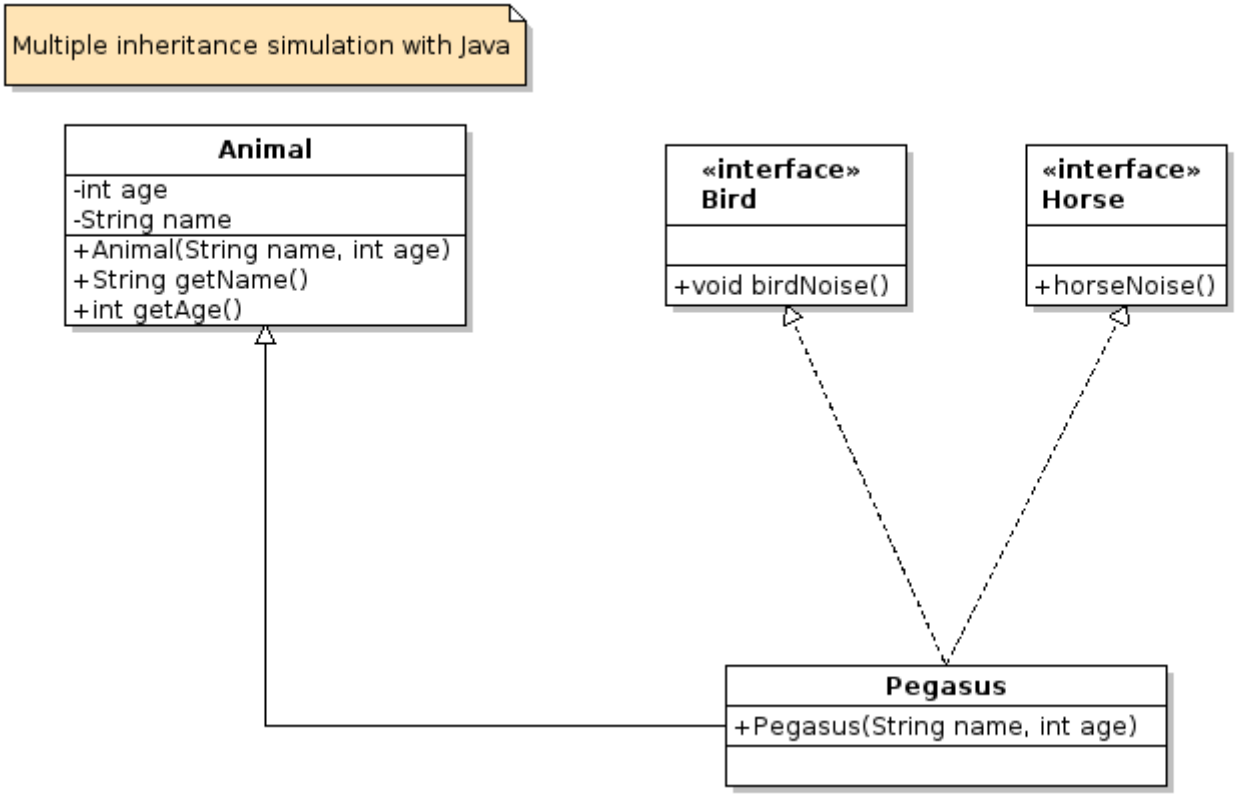
- Les interfícies no poden tenir variables d'instància, però és poden especificar constants
- Totes les variables en una interfície són automàticament **public static final** pel que es pot

```
public interface SwingConstants {
   int NORTH = 1;
   int NORTHEAST = 2;
   int EAST = 3;
   . . .
}
```

Exemple UML.Pegasus



Exemple UML.Pegasus



Exemple UML.Pegasus

```
public class Animal {

    protected int age;
    protected String name;

    public Animal()              {}

    public String getName()      {return name;}
    public int getAge()          {return age;}
}
```

```
public interface Bird {
    //all implementers have to overwrite methods
    void birdNoise();
}

public interface Horse {
    //all implementers have to overwrite methods
    void horseNoise();
}
```

Exemple UML.Pegasus

```
public class Pegasus extends Animal implements Bird, Horse {
    public Pegasus(String n, int a)
    {
        name = n;
        age = a;
    }

    //overriding interface methods
    public void horseNoise()
    {
        System.out.println("Horse Noise!");
    }

    public void birdNoise()
    {
        System.out.println("Bird Noise!");
    }


    public static void main(String[] args)
    {
        Pegasus peggi = new Pegasus("Pegasus", 5);
        System.out.println(peggi.getName() + "'s age is " + peggi.getAge());
        peggi.birdNoise();
        peggi.horseNoise();
    }
```

Exemple : Attendance