Ayudantía 6

Alexander Inostroza – alexander.inostroza@usm.cl

Modificadores de Acceso

Existen palabras clave en Java que permiten controlar el acceso a campos, métodos, e incluso clases completas al implementarlas.

MODIFICADOR	CLASE	PACKAGE	SUBCLASE	TODOS
public	Sí	Sí	Sí	Sí
protected	Sí	Sí	Sí	No
-	Sí	Sí	No	No
private	Sí	No	No	No

Herencia

Para entender una subclase, hay que saber que en Java una clase puede extender otra y heredar sus campos, métodos e incluso constructores.

Por ejemplo, si definimos una clase estudiante,

```
class Estudiante {
    protected String nombre;
    protected String rut;
    pprotected int edad;
    public void estudiar(){
        System.out.println("Estudiando matemáticas...");
    }
}
```

Y luego queremos ser más específicos para un estudiante universitario, podemos extender la clase **Estudiante** y reutilizar su código.

```
public class Universitario extends Estudiante {
    @Override
    public void estudiar(){
        System.out.println("Estudiando cálculo...");
    }
}
```

Encapsulamiento

La **encapsulación** es un principio fundamental de la programación orientada a objetos y consiste en ocultar el estado interno del objeto y obligar a que toda interacción se realice a través de los métodos del objeto.

Ejemplo

Para una clase **Ladrón**, que le roba a una clase **Víctima,**

```
public class Ladron {
    int dinero;
    public void robar(Victima victima){
        this.dinero = victima.dinero;
        victima.dinero = 0;
    }
}
public class Victima {
    int dinero;
}
```

Para aplicar encapsulamiento, cambiaríamos el código a:

```
public class Ladron {
    int dinero;
    public void robar(Victima victima){
        this.dinero = victima.entregarDinero();
    }
}
public class Victima {
    int dinero;
    public int entregarDinero(){
        int r = dinero;
        this.dinero = 0;
        return r;
    }
}
```

Clases Abstractas

Una clase abstracta un tipo especial de clase, que representa algún concepto abstracto que engloba a sus subclases. Una clase abstracta:

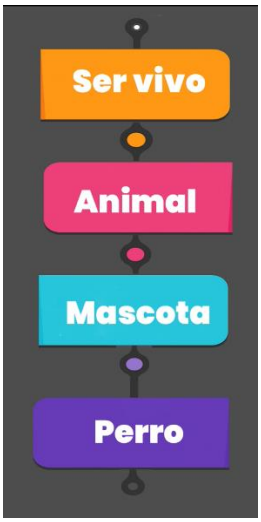
- No se puede instanciar.
- Puede tener atributos.
- Puede declarar métodos abstractos. Las subclases **deben** implementar estos métodos.
- Puede implementar métodos.
- En Java, se extiende igual que cualquier otra clase, usando **extends.**

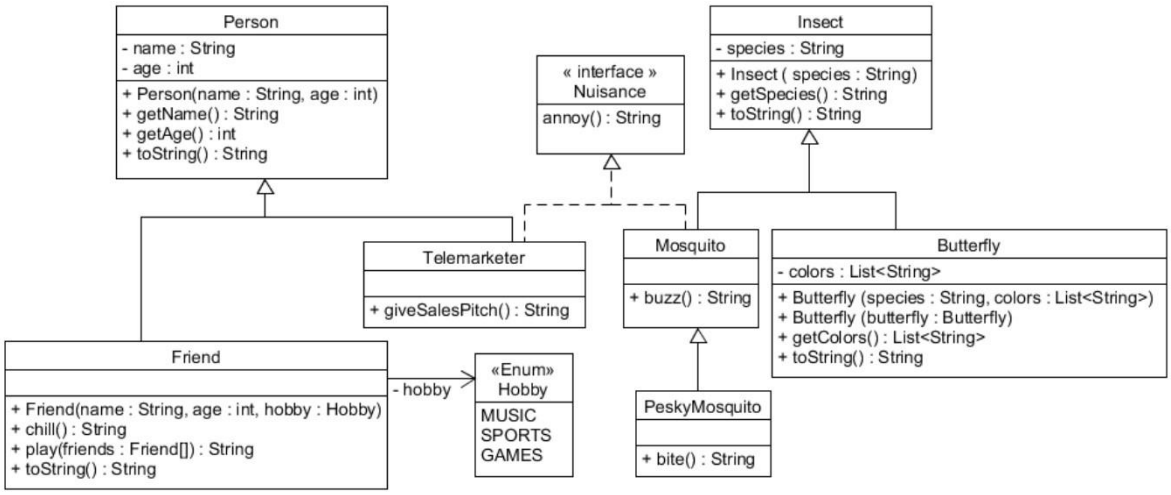
Interface

Una interface es una alternativa a la clase abstracta.

- No se puede instanciar.
- No puede tener atributos.
- No puede implementar métodos, sólo declarar. Las clases que implementen una interface, **deben** implementar los métodos de la interface, **a menos que** sea una clase abstracta.
- En java, se utilizan mediante **implements.**

Una clase puede extender sólo una clase, y esto también aplica para clases abstractas. Por el contrario, una clase puede implementar muchas interfaces.





En un diagrama UML, el uso de una interface se representa con una línea punteada y una flecha blanca.

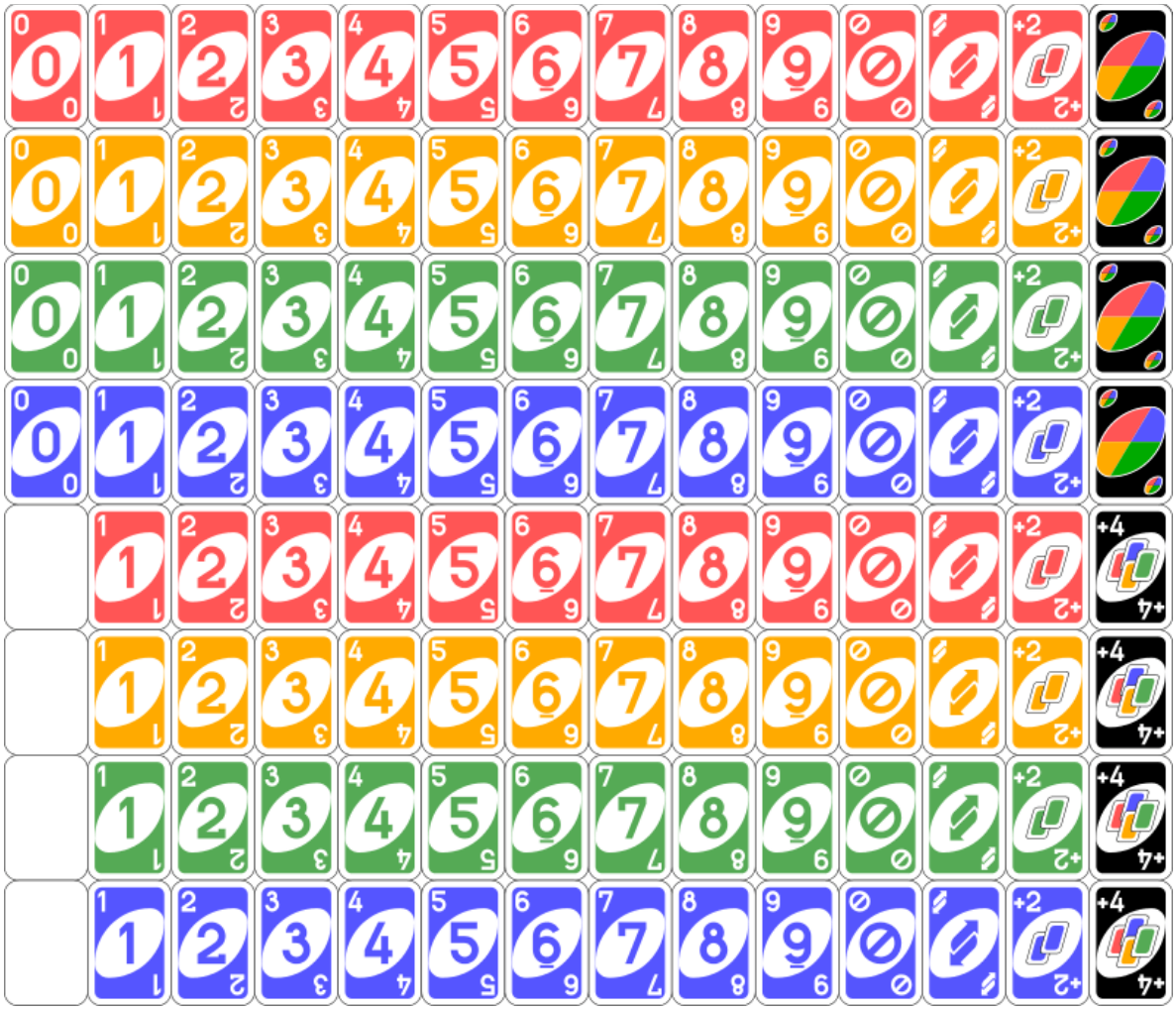
1. **Uno.**

   En el juego de cartas Uno, existen cartas de distintos colores, algunas son cartas especiales y otras son números sin ningúna habilidad particular.

   a) Represente las cartas usando clases e interfaces.

   b) Implemente el método isPlayable(Carta), que permita comparar la carta que se desea jugar con la última carta jugada y determine si la carta se puede jugar.

   c) Implemente el método play(), que simule el jugar la carta. Para aplicar habilidades de cartas basta con mostrar por pantalla lo que hace la habilidad.



2. **Catálogo de productos.**

   Se desea implementar un catálogo para una tienda que vende todo tipo de productos. Todo producto debe ser un artículo.

```java
public interface Articulo {
    public int getPrecio();
    public int getStock();
    public String getCategoria();
    public String getTipo();
    public String getNombre();
}
```

   Los productos estarán separados en categorías (muebles, computación, línea blanca, etc). Cada categoría se separará en tipos de productos (computación: notebooks, mouse, teclados, monitores, etc).

   Por ejemplo, para dos audífonos

Audifono1	**Audifono 2**
Precio: $ 239.990 Stock: 34 Categoría: Sonido Tipo: Audífonos Nombre: Apple AirPods Pro 2da Generación	Precio: $ 39.990 Stock: 75 Categoría: Sonido Tipo: Audífonos Nombre: Audífonos Bluetooth On-Ear 520BT Black

   Implemente dos categorías y dos tipos para cada categoría. Luego cree algunos artículos de los tipos creados en un programa main para comprobar la visualización de los datos.

3. **Equipamiento en un RPG.**
   Bugisoft está planeando lanzar un nuevo RPG, están en una fase muy temprana del desarrollo y le encargan a usted desarrollar la lógica para una primera versión del equipamiento y un personaje genérico que se pueda equiparse este equipamiento. Para ello le entregan a usted varios códigos sin terminar.

```
public class Character {
    private int healthPoints;
    private int defensePoints;
    private int evasionPoints;
    private int attackPoints;
    private int accuracyPoints;
    private Helm helm;

    public Character(){
        healthPoints=defensePoints=evasionPoints=attackPoints=accuracyPoints=0;
        helm = null;
    }
    public void setHelm(Helm helm){}
    public void getHelm(){}
    public void equip(Equipment equpipment){}
    public void unequip(Equipment equipment){}
}
```

```
public interface Equipment {
    public void equip(Character character);
    public void unequip(Character character);
    public int getDefensePoints();
    public int getEvasionPoints();
    public int getAttackPoints();
    public int getAccuracyPoints();
}
```

```
public abstract class Weapon implements Equipment{
    public int getAttackPoints(){return 0;}
    public int getAccuracyPoints(){return 0;}
    public abstract boolean isTwoHanded();
}
```

```
public abstract class DefensivePiece implements Equipment {
    public int getDefensePoints(){return 0;};
    public int getEvasionPoints(){return 0;};
}
```

El personaje debe equiparse: un yelmo, una armadura, guantes, botas, arma en mano principal y arma en mano secundaria. Para ellos implemente las clases **Helm, Armor, Gloves, Boots, GreatSword** (de dos manos), **Bow** (de dos manos), **Sword** (mano principal), **Shield** (mano secundaria), **Dagger** (mano secundaria); utilizando herencia, polimorfismo y encapsulamiento.

Nota: Si el personaje intenta equiparse un yelmo, y ya tiene uno equipado, debe desequiparlo y descontar las estadísticas del anterior, antes de equipar el nuevo.