Apuntes semana 09/9

Herencia

> **Herencia, clases concretas, clases abstractas, generalización y especialización, this y super**

La **herencia** es un mecanismo en programación orientada a objetos que permite a una clase (subclase) adquirir las propiedades y comportamientos de otra clase (superclase). Esto facilita el **reúso de código** y de conceptos, ya que la subclase puede reutilizar o extender lo que ya está definido en la superclase. En Java, la herencia es **simple**, lo que significa que una clase solo puede heredar de una única superclase. Además, la herencia es **transitiva**, es decir, si una clase B hereda de A y una clase C hereda de B, C también hereda las propiedades de A.

```
public class CuentaCorriente extends CuentaBancaria {
```

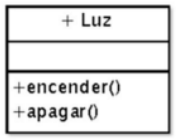
La prueba “Es un”

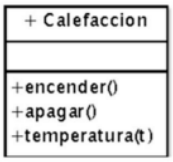
Preguntarse “es un” es la regla para identificar usos adecuados de herencia

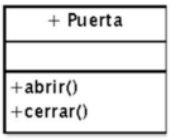
- Si suena bien en el lenguaje del dominio, es probable que sea un uso adecuado.
- Una cuenta corriente es una cuenta, una ventana de texto es una ventana, un array es una colección, un robot es un agente, etc...

Recordatorio Method Lookup

- Cuando un objeto recibe un mensaje, se busca en su clase un método cuya firma se corresponda con el mensaje.



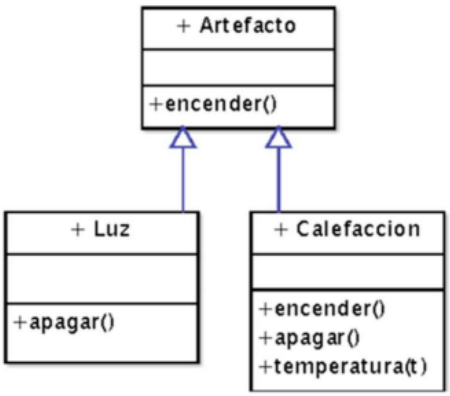





```
(new Luz()).encender();
(new Calefaccion()).encender();
(new Puerta()).encender();
```

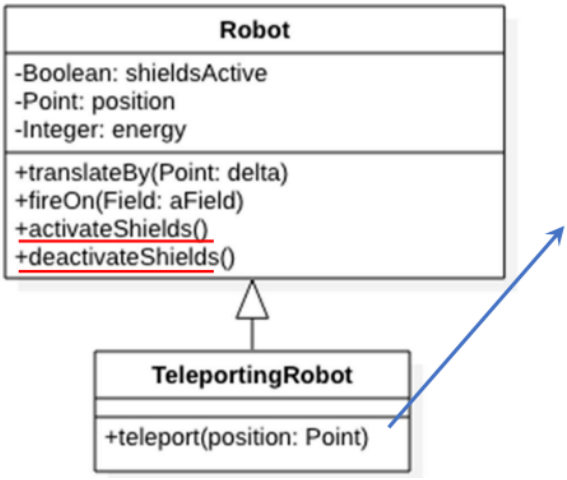
Method Lookup con herencia

- Cuando un objeto recibe un mensaje, se busca en su clase un método cuya firma se corresponda con el mensaje. Si no lo encuentra, sigue buscando en la superclase de su clase, y en la superclase de esta...




```
(new Luz()).encender();
(new Calefaccion()).encender();
```

Aprovechamiento de comportamiento heredado

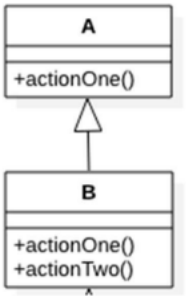



```
public void teleport(Point2D position) {
    this.activateShields();
    this.position = position;
    this.deactivateShields();
}
```

Sobre escribir métodos (overriding)

- La búsqueda en la cadena de superclases termina tan pronto encuentro un método cuya firma coincide con la que busco. Si heredaba un método con la firma, el mismo queda “oculto” (re definir-override).

```
(new B()).actionOne();
```




```
public void actionOne() {
    // Hacer algo como le gusta a A
}

public void actionOne() {
    // Hacer algo como le gusta a B
}
```

Super

- Super lo podemos pensar con una pseudo-variable (como this), ya que no puedo asignarle un valor y toma valor automáticamente cuando un objeto comienza a ejecutar un método.
- Es un método, “super y this” hacen referencia al objeto que lo ejecuta.
- Utilizando super en lugar de this solo cambia la forma en la que se hace el method lookup.
- Se utiliza para solamente extender el comportamiento heredado (re implementar un método e incluir el comportamiento que se heredaba de el).

Super y Method Lookup

Cuando super recibe mensaje, la búsqueda de los métodos comienza en la clase inmediata superior a aquella donde esta definido el método que envía el mensaje, sin importar la clase del receptor.

Super() en los constructores

- Los constructores en Java son subrutinas que se ejecutan en la creación de objetos, osea no se heredan.

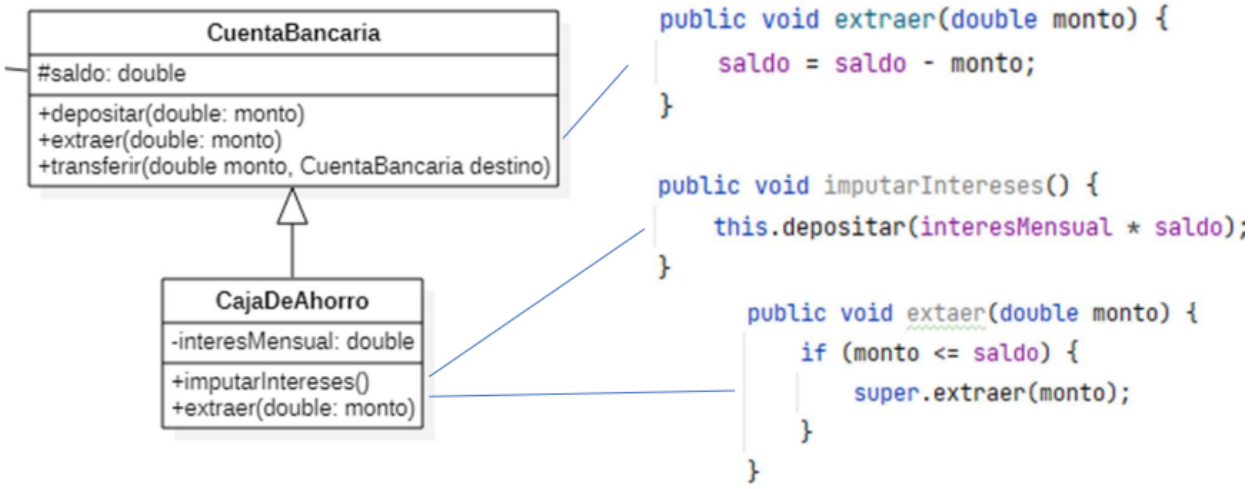
- Si quiero reutilizar comportamiento de otro constructor debo invocarlo de manera explicita usando super().

```java
public CuentaBancaria(Persona titular) {
    this.titular = titular;
}

public CuentaCorriente(Persona titular, double saldoInicial, double limiteDescubierto) {
    super(titular);
    saldo = saldoInicial;
    this.limiteDescubierto = limiteDescubierto;
}
```

Especializar

- Especializar es crear una subclase especializando una clase existente.



El pendejo puso “extaer” en vez de “extraer” xddd.

Clase abstracta

Clase abstracta

- Una clase abstracta captura comportamiento y estructura que será común a otras clases
- Una clase abstracta no tiene instancias (no modela algo completo)
- Seguramente será especializada
- Puede declarar comportamiento abstracto y utilizarlo para implementar comportamiento concreto

```
public abstract class CuentaBancaria {
    protected double saldo;

    public void depositar(double monto) {
        saldo = saldo + monto;
    }

    public void transferir(double monto, CuentaBancar
        destino.depositar(monto);
        this.extraer(monto);
    }

    public abstract void extraer(double monto);
}
```

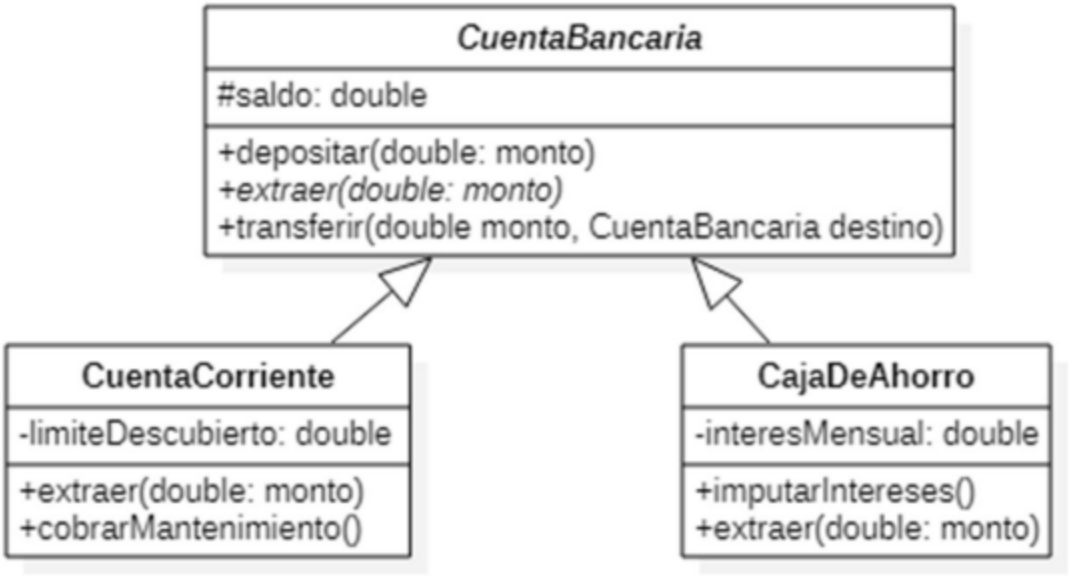
Gancho que deben implementar las subclases

Generalizar

- Generalizar: Introducir una superclase que abstrae aspectos comunes a otras - suele resultar en una clase abstracta.

CuentaCorriente
-limiteDescubierto: double -saldo: double
+depositar(double: monto) +transferir(double monto, CuentaCorriente destino) +extraer(double: monto) +cobrarMantenimiento()

CajaDeAhorro
-interesMensual: double -saldo: double
+depositar(double: monto) +transferir(double monto, CajaDeAhorro destino) +extraer(double: monto) +imputarIntereses()



Situaciones de uso de herencia

- Suclasificar para especializar:
  - La subclase extiende métodos para especializarlos, y ambas clases concretas.
- Herencia para especificar:
  - La superclase es una combinación de métodos concretos y abstractos (clase abstracta). Dicha subclase implementa los métodos abstractos.
- Subclasificar para extender:
  - La subclase agrega nuevos métodos.

Situaciones de uso de herencia (feas)

- Heredar para construir:
  - Heredo comportamiento y estructuro, pero cambio el tipo (no-es-un)
  - Se debe evitar, aunque nos vamos a cruzar con ejemplos.
- Subclasificar para generalizar (No confundir con generalización)
  - La subclase re-implementa métodos para hacerlos mas generales.

    - Solo tiene sentidos si no puedo re ordenar la jerarquía (para especializar).
- Subclasificar para limitar:
    - La subclase re implementa un método para que deje de funcionar / limitarlo
    - Solo tendría sentido si no puedo reodenar la jerarquía (para especializar)
- Herencia indecisa (subclasificacion por varianza)
    - Tengo dos clases con un mismo tipo, y algunos métodos compartidos.
    - no puedo decidir cual es la subclase y cual la superclase.
    - Resolverlo buscando una superclase común (generalización).

Tipos en lenguajes de OO (recordatorio)

- Tipo: Conjunto de firmas de operaciones/métodos (nombre, tipos de argumentos)
- Decimos que un objeto “es de tipo” si ofrece el conjunto de operaciones definido por el tipo.
- Con esto en mente:
    - Cada clase en Java define “explicitamente” un tipo.
    - Cada instancia de una clase A “es del tipo” definido por esa clase.
- Donde espero un objeto de una clase A, acepto un objeto de cualquier subclase de A (lo opuesto no es cierto)

Modificadores de visibilidad con herencia

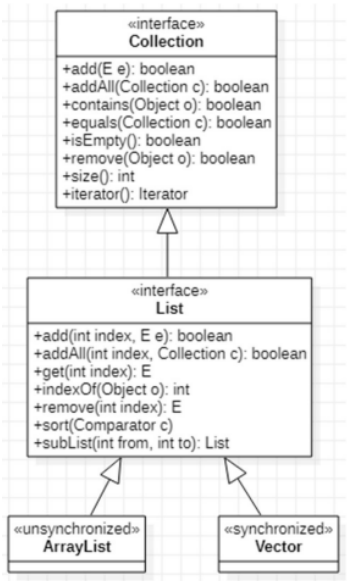
- Si yo declaro una variable de instancia como private en una clase A:
    - Las instancias de las subclases de A tendrán esa variable de instancia.
    - En los métodos de las subclases de A, no puedo hacer referencia a ella.

- Si yo declaro un método “m()” como private en una clase A:
  - Las instancias de sus subclases entenderán el mensaje m().
  - En los métodos de las subclases de A, no puedo enviar m() a “this”.

Clases abstractas e interfaces

- **Una clase abstracta “es una clase”**
  - Puedo usarla como el tipo
  - Puede o no tener métodos abstractos
  - Ofrece implementacion a algunos métodos
  - Sus metodos concretos pueden depender de sus metodos abstractos
- **Una interfaz define un tipo**
  - Sirve como contrato
  - Puede extender a otras
- Se pueden implementar muchas interfaces pero solo se puede heredar de una clase.

Listas




```java
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class Ticket {

    private List<Item> items;
    private Date fecha;
    private Cliente cliente;

    public Ticket(Cliente cliente) {
        this.cliente = cliente;
        fecha = new Date();
        items = new ArrayList<Item>();
    }

    public void agregarItem(Producto producto, int cantidad) {
        items.add(new Item(producto, cantidad));
    }

    public List<Item> getItems() {
        return items;
    }
```