LABORATORIO DE PROGRAMACIÓN ORIENTADA A OBJETOS

DRA. MARICELA BRAVO

DIVISION DE CIENCIAS BASICAS E INGENIERIA
UAM - Azcapotzalco

UNIVERSIDAD AUTONOMA METROPOLITANA
Casa abierta al tiempo
Azcapotzalco

MODIFICADORES DE ACCESO

- Se utilizan para definir la visibilidad de los miembros de una clase (atributos y métodos) y de la propia clase.
- En Java existen cuatro modificadores de acceso que ordenados de menor a mayor visibilidad, son:
  - *private*
  - *ninguno*
  - *protected*
  - *public*

2

MODIFICADORES DE ACCESO

- ***private***. Cuando un atributo o método es definido como ***private***, su uso está restringido al interior de la clase, solamente puede ser utilizado en el interior de su misma clase. Este modificador puede ser aplicado a métodos y atributos, pero no a la clase.
- ***ninguno***. A falta de la definición de modificador se utiliza el *acceso por defecto*. Si un elemento (clase, método o atributo) tiene acceso por defecto, únicamente las clases de su mismo paquete tendrán acceso al mismo.

3

MODIFICADORES DE ACCESO

- ***protected***. Un método o atributo definido como ***protected*** en una clase puede ser utilizado por cualquier otra clase de su mismo paquete y además por cualquier subclase de ella. Una clase no puede ser ***protected***, solo sus miembros.
- ***public***. El modificador public ofrece el máximo nivel de visibilidad. Un elemento (clase, método o atributo) ***public*** será visible desde cualquier clase, independientemente del paquete en que se encuentren.

4

MODIFICADORES DE ACCESO

	private	*default*	*protected*	*public*
Clase	No	Si	No	Si
Método	Si	Si	Si	Si
Atributo	Si	Si	Si	Si
Variable local	No	No	No	No

5

SOBRECARGA DE MÉTODOS

- Cuando en una misma clase se definen varios métodos con el mismo nombre, se llama sobrecarga de métodos.
- Para que un método pueda sobrecargarse es imprescindible que se cumpla la siguiente condición:
  - Cada versión del método debe distinguirse de las otras en el número o tipo de parámetros.
  - El tipo de devolución puede o no ser el mismo.

6

CONSTRUCTORES

- Un constructor es un método especial que es ejecutado en el momento en que se crea un objeto de la clase (cuando se llama al operador new).
- Podemos utilizar los constructores para añadir aquellas tareas que deban realizarse en el momento en que se crea un objeto de la clase, por ejemplo, la inicialización de los atributos.

7

CONSTRUCTORES

- Al crear un constructor hay que tener en cuenta las siguientes reglas:
  - El nombre del constructor debe ser el mismo que el de la clase.
  - El constructor no puede tener tipo de devolución.
  - Los constructores se pueden sobrecargar.
  - Toda clase debe tener al menos un constructor.

8

PRINCIPIOS DE LA POO

- Son las propiedades fundamentales que orientan el estilo de programación que permite crear código reutilizable, usable y de fácil mantenimiento.

1. Principio de abstracción
2. Principio de encapsulamiento
3. Principio de modularidad
4. Principio de paso de mensajes
5. Principio de herencia
6. Principio de polimorfismo

9

ABSTRACCIÓN Y ENCAPSULAMIENTO

10

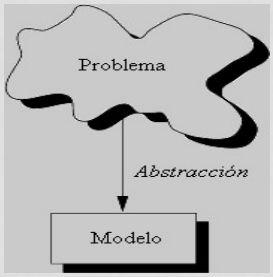
PRINCIPIO DE ABSTRACCIÓN

- Abstraer es suprimir u ocultar algunos detalles de un proceso o de un elemento, para resaltar algunos aspectos, detalles o estructuras.
- La abstracción de objetos se refiere a la vista externa del objeto (interfaces).
- La abstracción es la forma en que nuestra mente modela la realidad, formando los objetos.

11

ABSTRACCIÓN

- El modelo define una ***perspectiva abstracta*** del problema
  - Los ***datos*** que son afectados
  - Las ***operaciones*** que se aplican sobre los datos



12

ENCAPSULAMIENTO

- Proceso por el que se ocultan:
  - Las estructuras de datos
  - Los detalles de la implementación
- Permite considerar a los objetos como "cajas negras", evitando que otros objetos accedan a detalles que NO LES INTERESA
- Una vez creada la clase, las funciones usuarias no requieren conocer los detalles de su implementación

13

ENCAPSULAMIENTO

- Toda clase tiene un conjunto de ***atributos*** y ***métodos*** asociados a ella.
- Todos ellos están ***encapsulados*** o contenidos dentro de la misma clase, de manera que son ***miembros*** de dicha clase.
- Esos métodos y atributos pueden ser utilizados por ***otras*** clases ***sólo si*** la clase que los encapsula les brinda los ***permisos*** necesarios para ello.

14

ENCAPSULAMIENTO

- A los métodos también se les puede aplicar distintos modificadores de acceso, por lo que un método puede ser también marcado como **private**.
- La recomendación general es inicialmente hacer todo privado e irlo haciendo público conforme se va necesitando.
- Entre menos métodos públicos tenga una clase es más fácil de entender.
- No se recomienda tener atributos públicos en lo absoluto.

15

EJEMPLO ENCAPSULAMIENTO

```
public class Animal
{
  // Definición de atributos privados
  private int numOjos;
  private String formaMoverse;
  private String tipoRespiracion;
  private String tipoReproduccion;
  private String haceRuido;
```

¿Cómo se almacenan estos datos?

16

EJEMPLO ENCAPSULAMIENTO

```
//Método constructor
public Animal(int numOjos, String formaMoverse,
              String tipoRespiracion,
              String tipoReproduccion,
              String haceRuido)
  {
      super();
      this.numOjos = numOjos;
      this.formaMoverse = formaMoverse;
      this.tipoRespiracion = tipoRespiracion;
      this.tipoReproduccion = tipoReproduccion;
      this.haceRuido = haceRuido;
  }
```

17

EJEMPLO ENCAPSULAMIENTO

```
public int getNumOjos() {
  return numOjos;
}
public String getFormaMoverse() {
  return formaMoverse;
}
public String getTipoRespiracion() {
  return tipoRespiracion;
}
public String getTipoReproduccion() {
  return tipoReproduccion;
}
public String getHaceRuido() {
  return haceRuido;
}
```

Métodos de acceso a los atributos

18

PRINCIPIO DE HERENCIA

ORGANIZACIÓN JERÁRQUICA

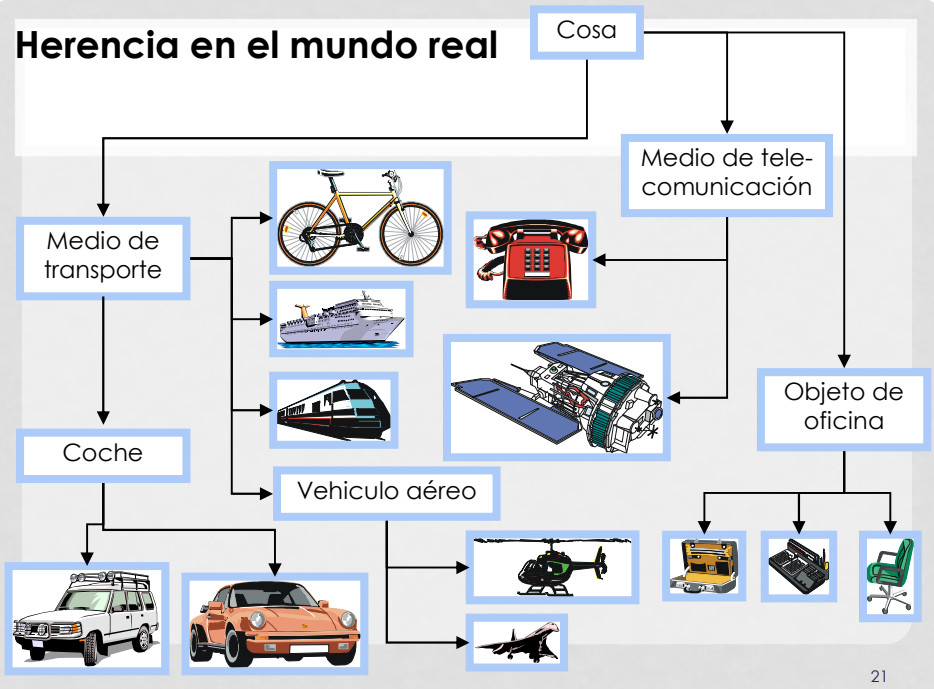
19

HERENCIA

- La herencia es un mecanismo que permite la definición de una clase a partir de la definición de otra ya existente.
- La herencia permite compartir automáticamente métodos y atributos entre clases, subclases y objetos.
- Permite reutilizar código creando nuevas clases a partir de las existentes (construidas y depuradas).
- Compromete una relación de jerarquía (es-un).

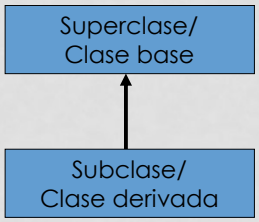
20

Herencia en el mundo real



21

HERENCIA



En POO, la clase de la que se va a heredar se llama superclase o clase base. Mientras que la que hereda se le conoce como subclase o clase derivada.

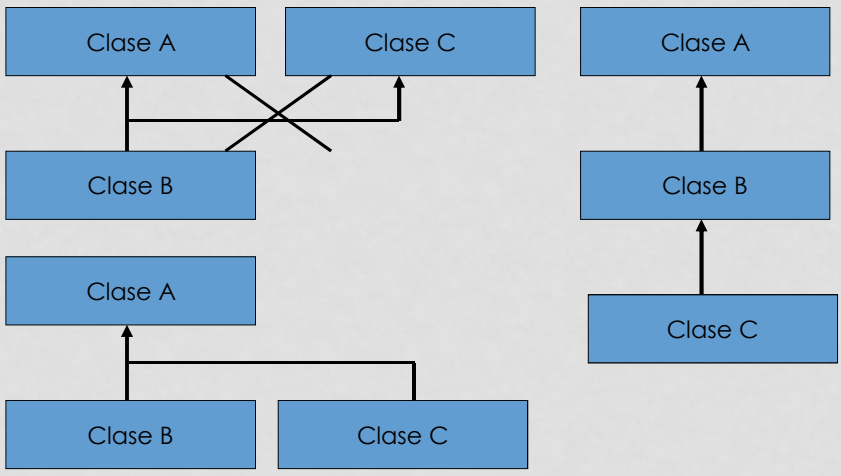
22

HERENCIA

- Reglas básicas para la herencia en Java
  - No está permitida la herencia múltiple
  - Si es posible la herencia multinivel, A puede ser heredada por B y C puede heredar de B
  - Una clase puede ser heredada por varias clases.

23

HERENCIA

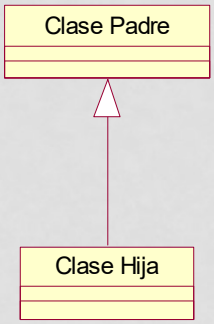


24

HERENCIA EN JAVA

- Java permite definir una clase como subclase de una clase padre.

class *clase_hija* ***extends*** *clase_padre*
{
.....
}



25

CONSTRUCTORES Y HERENCIA

- Cuando se declara un objeto de una clase derivada, se ejecutan los constructores siguiendo el orden de derivación, es decir, primero el de la clase base, y después los constructores de las clases derivadas de arriba a abajo.
- Para pasar parámetros al constructor de la clase padre:

super *(para1, para2, ..., paraN)*

26

EJEMPLO

- Se requiere el desarrollo de un sistema de registro de personal en un entorno académico en el cual participan profesores impartiendo materias a los alumnos, un jefe por cada departamento dirigiendo las actividades administrativas, coordinadores académicos atendiendo asuntos académicos de los alumnos, así como el apoyo de empleados administrativos (secretarias, conserjes, etc.), y visitantes que acuden por diversas razones.

27

28

EJEMPLO DE HERENCIA
SISTEMA DE INFORMACIÓN PARA UN ZOOLÓGICO

- Deberá proporcionar información básica sobre todos los animales:
  - Región donde habita
  - Tipo de alimentación
  - Forma de reproducción
  - Forma de moverse
  - Forma de comunicarse
- Además se deberá proporcionar información específica de cada animal de acuerdo a la clasificación de animales vertebrados a la que pertenece:
  - Peces
  - Anfibios
  - Reptiles
  - Aves
  - Mamíferos
  - *Medio de respiración (branquias o pulmones)*
  - *Tipo de piel (escamas, plumas, pelos)*
- Cada individuo del zoológico deberá identificarse por una clave, su nombre y edad.

29

30

CLASIFICACIÓN DE LOS ANIMALES

- Invertebrados:
  - Pólipos y medusas (Cnidarios)
  - Gusanos (Anélidos)
  - Equinodermos
  - Moluscos
  - Artrópodos:
    - Insectos
    - Arácnidos
    - Crustáceos
    - Miriápodos
- Vertebrados:
  - Peces
  - Anfibios
  - Reptiles
  - Aves
  - Mamíferos

EJEMPLO DE HERENCIA

- Simular el comportamiento que tendrían los diferentes integrantes de la selección mexicana de futbol; tanto los Futbolistas como el cuerpo técnico (Entrenadores, Masajistas, etc…).
- Para simular este comportamiento se requieren tres clases que van a representaran a objetos tipo Futbolista, Entrenador y Masajista.
- De cada unos de estos objetos se requiere definir los atributos y una serie de acciones que reflejaremos en sus métodos.

32

EJEMPLO DE SUPER

```
class Persona {
   private String nombre;
   private int edad;
   public Persona() {}
   public Persona (String n, int e)
   { nombre = n; edad = e; }
}
class Alumno extends Persona {
   private int curso;
   private String nivelAcademico;
   public Alumno (String n, int e, int c, String nivel) {
         super(n, e);
         curso = c; nivel_academico = nivel;
   }
   public static void main(String[] args) {
         Alumno a = new Alumno("Pepe", 1, 2, "bueno");
   }
}
```

33

REDEFINIR F. MIEMBROS DE LA CLASE PADRE

```
class Persona {
   private String nombre;
   private int edad;
   .....
   public String toString() { return nombre + edad; }
   public void setEdad(int e) { edad = e; }
}
class Alumno extends Persona {
   private int curso;
   private String nivelAcademico;
   .....
   public String toString() {
         return super.toString() + curso + nivelAcademico;
   }
   public void setCurso(int c) { curso = c; }
}
```

34

REFERENCIAS A OBJETOS DE CLASES HIJAS

- Si tenemos *ClaseHijo hijo = new ClaseHijo(...);*
- entonces es posible *padre = hijo* donde *padre* es una variable de tipo *ClasePadre*.
  - pero no es posible *hijo = padre*
  - es posible con casting *hijo = (ClaseHijo) padre*
- Ahora bien:
  - Con *padre* sólo podemos acceder a atributos y métodos def. en la clase padre.

35

REFERENCIAS A OBJETOS DE CLASES HIJAS

```
public static void main(String[] args) {
  Persona p;
  Alumno a = new
  Alumno("pepe",23,1,"universitario");
  p=a; //ref padre señala al objeto hijo
  // acceso al objeto hijo mediante la ref padre
  p.setEdad(24);
  /* no es posible acceder a un miembro de la clase
  hija usando una ref a la clase padre*/
  p.setCurso(88); // ERROR
  }
```

36

EJEMPLO

```
class Persona { ..... }
class Alumno extends Persona {
   ......................
   public String toString() {
         return super.toString() + curso + nivelAcademico;
   }
}
class Profesor extends Persona {
   private String asignatura;
   public Profesor (String n, int e, String asign) {
         super(n, e);
         asignatura = asign;
   }
   public String toString() {
         return super.toString() + asignatura;
   }
}
```

37

PRINCIPIO DE POLIMORFISMO

38

POLIMORFISMO

- Capacidad que permite a dos ***clases diferentes*** responder de forma distinta a un ***mismo mensaje***
- Esto significa que dos clases que tengan un método con el mismo nombre y que respondan al mismo tipo de mensaje (es decir, que reciban los mismo parámetros), ejecutarán acciones distintas

39

POLIMORFISMO

Ejemplo 1:

Al presionar el acelerador esperamos que aumente la velocidad del auto, independiente de si se tiene un:

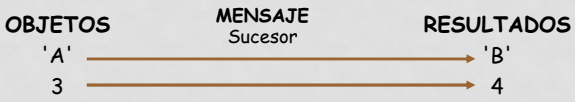
- Motor con carburador
- Motor con inyección electrónica

40

POLIMORFISMO

Ejemplo 2:

Si se tienen las clases **Entero** y **Char**, ambas responderán de manera distinta al mensaje **"Sucesor"**



41

PRINCIPIO DE MODULARIDAD

42

PRINCIPIO DE MODULARIDAD

- La modularidad permite dividir un problema complejo en varios módulos o partes diferentes, donde cada módulo resolverá una parte de un problema grande, y después interactuarán todos los módulos.
- Cada módulo debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes.
- Se debe tener en cuenta los conceptos asociados de dependencia, acoplamiento, cohesión, interfaz, encapsulación y abstracción

43

PRINCIPIO DE PASO DE MENSAJES

44

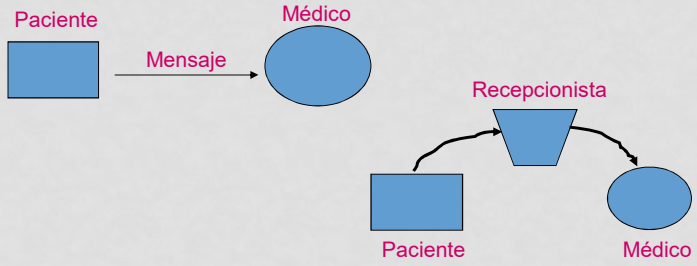
MENSAJE

- Mecanismo por el cual ***se solicita*** una acción sobre el objeto
- Un programa en ejecución es una colección de objetos que se crean, ***interactúan*** y se destruyen
- La ***interacción*** se basa en ***mensajes*** que son enviados de un objeto a otro, de modo que el emisor le pide al receptor la ejecución de un método

45

MENSAJES

- Un objeto invoca un método como una reacción al recibir un mensaje
- La interpretación del mensaje dependerá del receptor



46

- Con el paso de mensajes los objetos pueden solicitar a otros objetos que realicen alguna acción o que modifiquen sus atributos.
- Junto con el paso de mensajes se implementan llamadas a los métodos de un objeto, lo que nos permite hacer que ese objeto realice determinada acción que está dentro de su comportamiento.

47

DUDAS O COMENTARIOS???