

1.00Clase13

Herencia e interfaces

Clases abstractas

- **Las clases son muy generales en la cúspide de una jerarquía.**
 - **Por ejemplo, el MIT podría tener una clase Person (persona), de la que hereden Employees (empleados), Students (estudiantes), Visitors (visitantes), etc.**
 - **Persones una clase demasiado abstracta para que el MIT la utilice en un sistema informático, pero puede contener los atributos comunes a todas las subclases, como: nombre, dirección, estado...**
 - **Podemos hacer de Person una clase abstracta: no se pueden crear objetos Person, pero sí objetos de las subclases como Student.**
- **Las clases pueden ser concretas o abstractas**

Clases abstractas, 2ª parte

- Otro ejemplo (relacionado con gráficos de las siguientes clases):
 - La clase Shape (forma) en un sistema de gráficos.
 - La mayoría de las formas son demasiado generales para dibujarlas; sólo sabemos cómo dibujar formas específicas, como círculos o rectángulos.
 - La clase abstracta Shape puede definir un conjunto común de métodos que todas las formas deben implementar para que el sistema de gráficos pueda contar con determinados elementos disponibles en toda clase concreta.
 - La clase abstracta Shape puede implementar algunos métodos que todas las subclases deben usar. P. ej.: ID objeto, tipo de objeto, etc.

Clase Shape

```
abstract class Shape {
    public abstract void draw();
    // La función draw (dibujar) se debe implementar en cada clase
    // derivada, pero no es posible establecerla por defecto: abstracta

    public void error(String message);
    // La función error se debe implementar en cada clase derivada
    // y dispone de una forma por defecto: método no abstracto

    public final int objectID();
    // Función objectID: cada clase derivada debe tener una y
    // tiene que utilizar esta implementación: método final

    ...};

class Square extends Shape {...};
class Circle extends Shape {...};
```

Método abstracto

- Shape es una clase abstracta(abstract)
- Shape tiene un método abstractodraw()
 - No pueden crearse objetos de tipoShape
 - draw() ha de ser redeclarado por cualquier clase concreta (no abstracta) que lo herede
 - No existe ninguna definición de draw() enShape
- Esto significa que es posible dibujar todas las formas, pero que la claseShape (forma) no sabe cómo dibujar formas específicas

Método no abstracto

- Shape tiene un métodoerror() no abstracto
 - Toda clase derivada debe tener un métodoerror
 - Cada clase derivada puede manejar los errores como desee:
 - Puede definir su propia función de errorutilizando esta interfaz (métodoargumentos/valor de retorno)
 - Puede usar la implementación de la superclasecomo valor por defecto
 - Esto puede ser peligroso:si se añaden nuevas clases derivadas y los programadores no redefinen métodos no abstractos,el valor por defecto será invocado pero puede ser incorrecto
 - P.ej.canguros

Método Final

- **Shapetiene un método final llamado objectID**
 - El método final no varía en las clases derivadas
 - El comportamiento no debe cambiar, sin importar el grado de especialización de la clase derivada
- **Las superclases deben tener una combinación de métodos**
 - No convierta en abstractos todos los métodos de una superclase abstracta. Tome una posición

Cómo evitar la herencia

- **Para evitar que alguien herede de su clase, declárela como *final*:**

```
final class Grad extends Student { ...
```

- Esto no permitirá que se construya SpecGrad
- (Class puede ir acompañada de las palabras clave *abstract*, *final* o de ninguna)
- **También puede evitar que los métodos individuales se sobrescriban (redefinan) en las subclases declarándolos como finales (*final*)**

```
final void getData() { ...
```

- Esto evitará que una subclase redefina `getData()`

Definición de interfaz

- Es una especificación para un conjunto de métodos que una clase debe implementar:
 - Los interfaces especifican, pero no implementan métodos
 - Una clase que implemente la interfaz debe implementar todos sus métodos
 - Puede entonces invocar todos los métodos de la clase que dependen de la interfaz. Ejemplos:
 - Si su clase implementa la interfaz Comparable, puede colocar objetos de su clase en Arrays y utilizar el método Arrays.sort()
 - Las interfaces se utilizan con frecuencia en Swing (GUI)

Detalles de interfaz

- Las interfaces son como una clase abstracta pero:
 - Si se implementasen como clase abstracta, una subclase sólo podría heredar de una superclase
 - Se pueden heredar múltiples interfaces (p.ej., Comparable y Cloneable) en su clase
 - Las interfaces no se pueden instanciar

```
Comparable list= new Comparable();           // Error
```
 - Puede declarar objetos para ser de tipo interfaz

```
Comparable list;                             // OK
```
 - Pueden ser nombres para objetos de una clase que implementa la interfaz:

```
Comparable list= new MySortClass();          // OK
```
 - Las interfaces pueden contener métodos y constantes

```
public interface Rotatable {
    void rotate(double theta);                // Lista de métodos requeridos
    double MAX_ROTATE= 360; }                 // Final implícito
// Métodos y campos definidos por defecto como public
```

Interfaces y herencia

- **Las interfaces se pueden heredar**

- **Habilitación para el programa de becas**

```
public interface Eligible {  
    boolean IsEligible(double age, double income); }  
}
```

- **Selección utilizada en el programa de becas real**

```
public interface Selected extends Eligible {  
    boolean IsSelected(double gpa, double terms); }  
}
```

- **Nuestro programa podría tener una clase de selección de becas que funcionase sobre objetos de las clases Student que implementan una o ambas de estas interfaces**

- Las clases estudiante de licenciatura, posgrado y posgrado especial necesitarían implementar estos métodos

Ejemplo de interfaz

```
import java.util.*;  
  
public static void main(String[] args) {  
    Student[] list= new Student[4];  
    list[0]= new Student("Mary", 6);  
    list[1]= new Student("Joan", 4);  
    list[2]= new Student("Anna", 8);  
    list[3]= new Student("John", 2);  
    Arrays.sort(list);  
    for (int i= 0; i < list.length; i++) {  
        Student s= list[i];  
        System.out.println(s.getName() + " "  
            + s.getTerms());  
    }  
}
```

// Busque o consulte la interfaz Comparable en Javadoc

Ejemplo de interfaz, 2ª parte

```
class Student implements Comparable { // Requerido por Arrays.sort
    public Student(String n, int t) {
        name= n; terms= t;    }
    public String getName() { return name; }
    public int getTerms() { return terms; }
    public int compareTo(Object b) { // Requerido por Comparable
        Student two= (Student) b;
        if (terms < two.terms)
            return -1;
        else if (terms > two.terms)
            return 1;
        else
            return 0;    }
    private String name;
    private int terms;
}
```

Herencia: puntos clave

- **La herencia permite a un programador ampliar objetos que no escribió :**
 - Las restricciones de acceso se mantienen para la superclase
 - Si la clase básica modifica datos o miembros privados, las subclases no se verán afectadas
 - Los miembros protegidos en una superclase permiten el acceso directo por subclases
 - No debe modificar la superclase; debe estar diseñada con la intención de que las subclases puedan utilizarla
 - La subclase tiene todos los datos (private, protected y public) de la superclase. Cada objeto tiene todos estos datos.
 - La subclase sólo puede utilizar los métodos y datos públicos y los protegidos de la superclase, no los métodos o datos privados
 - Todos los objetos Java heredan implícitamente de la clase Object
 - Las librerías y documentación de Java usan Object con frecuencia

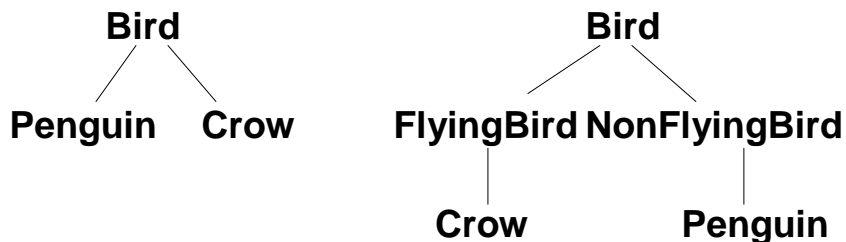
Diversión con animales

```
class Bird {
    public void fly();      // Los pájaros (birds) pueden volar
    ... };

class Penguin extends Bird {      // Los pingüinos (penguins) son pájaros
    ... };

// Problemas:
// Si el método fly() de la superclase no se declara como
// final, los pingüinos deben volar
// Si el método fly() de la superclase se declara como
// abstracto o no abstracto, el método fly() de la clase
// Penguin puede mostrar un mensaje de error.
// Con la herencia, cada subclase cuenta con todos los métodos
// y atributos de la superclase. Hay que estar muy atento.
// Esta es una de las dificultades del diseño en un sistema real.
```

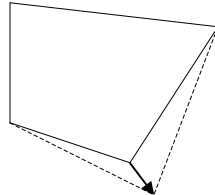
Soluciones posibles



La decisión depende del uso del sistema:
Si está estudiando picos, la diferencia entre volar y no volar tal vez no importe

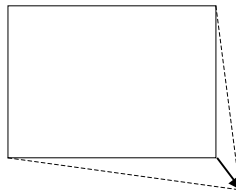
Más cuestiones

Quadrilateral



MoveCorner()

Rectangle



MoveCorner()

El método `MoveCorner()`, que permite mover múltiples esquinas, se debe anular en las subclases para preservar la forma correcta

Ejercicio1

A. Descargue `Vehicle.java` y `VehicleTest.java` :

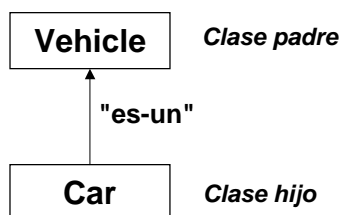
- Debe obtener la ruta del directorio montado: en la vista Explorer haga clic en la pestaña FileSystems. La ruta del directorio montado aparece como un enlace en su árbol de Filesystems (si tiene varios directorios montados, escoja el primero)
- En su navegador vaya a:
web.mit.edu/1.00/www/Lectures/Lecture13/Vehicle.java
Guarde el archivo descargado en el directorio montado. (Haga clic con el botón derecho, 'SaveLinkAs')
- Haga lo mismo con:
web.mit.edu/1.00/www/Lectures/Lecture13/VehicleTest.java
- Vuelva a FileSystems, en la vista Explorer. Haga clic con el botón derecho sobre el directorio montado y elija *refresh* (actualizar). Aparecerá el nuevo archivo guardado.

Práctica

- A. Cree un proyecto llamado `vehicle` y añada los archivos `Vehicle.java` y `VehicleTest.java` al proyecto (para añadir un archivo haga clic con el botón derecho sobre él y escoja *Tools->AddToProject*).
- B. He aquí un resumen de la clase `Vehicle` que usted ha descargado:
 - `String brand`: marca del vehículo (p. ej. `Honda`)
 - `String model`: modelo del vehículo (p. ej. `Accord`)
 - `String type`: tipo de vehículo (p. ej. `Sedan`)
 - `print()`: imprime las propiedades del vehículo.
- C. El `main()` de `VehicleTest.java` crea una bicicleta (que es un objeto de vehículo) e imprime sus propiedades. Compile y ejecute su programa para probar la clase vehículo.

Cree una clase `Car` (coche)

- La clase `Vehicle` contiene información básica, y estaría bien para bicicletas. Pero un coche, que es un vehículo necesita más información. Queremos crear una clase llamada `Car`, que además de las propiedades básicas contenidas en `Vehicle`, tenga dos campos nuevos: el número de cilindros y el de asientos. La relación entre `Vehicle` y `Car` se muestra a continuación:



Cree una clase Car(2)

- A. Cree la clase `Car` que amplíe `Vehicle`. La clase `Car` debería tener los siguientes miembros:
- `int cylinders`: número de cilindros del motor
 - `int seats`: número de asientos.
 - Un constructor que inicialice los miembros de datos de un objeto `Car`: `brand, model, type, cylinders y seats`.

No añada todavía el método `print()` a la clase `Car`

- B. En la clase `VehicleTest`, cree un objeto `Car` con los siguientes atributos:
- Brand (marca): `BMW`
 - Model (modelo): `330i`
 - Type (tipo): `SportsCar`
 - Cylinders (cilindros): `6`
 - Seats (asientos): `5`

Cree una clase Car(3)

- C. Imprima la información de coche después de la de bicicleta.
- D. Compile y ejecute. Debe obtener el siguiente resultado:
Marca:GT;Modelo:01-IDrive 2.0;Tipo: Mountain Bike;
Marca:BMW;Modelo:330i;Tipo:SportsCar;

Hasta ahora, un objeto del tipo `Car` contiene los siguientes miembros:

Miembros	
<code>brand</code>	Heredado de <code>Vehicle</code>
<code>model</code>	Heredado de <code>Vehicle</code>
<code>type</code>	Heredado de <code>Vehicle</code>
<code>cylinders</code>	Definido en <code>Car</code>
<code>seats</code>	Definido en <code>Car</code>
<code>print()</code>	Heredado de <code>Vehicle</code>

Cómo anularPrint()

- A. Necesitamos un nuevo método `print()` para la clase `Car` que imprima todos sus miembros, incluidos `scylinders` y `seats`.
- B. En la clase `Car`, anule el método `print()` de la clase padre. Un hijo anula el método de su padre definiendo un método de reemplazo con la misma firma.
- C. Compile y ejecute el programa. Debería obtener el siguiente resultado:

Marca:GT;Modelo:01-IDrive2.0;Tipo: Moutain Bike;

Marca:BMW;Modelo:330i;Tipo:SportsCar;
Cilindros:6;Asientos:5;

Ejercicio2

- A. Descargue `Electronics.zip` :

- En su navegador acceda a:
web.mit.edu/1.00/www/Lectures/Lecture13/Electronics.zip
Guarde el archivo descargado en su directorio montado (haga clic con el botón derecho, 'SaveLinkAs')
- Descomprima `Electronics.zip` y extraiga todos los archivos al directorio montado. Deberá tener los siguientes archivos:
 - `Electronic.java`
 - `CellPhone.java`
 - `Computer.java`
 - `Laptop.java`
- Vuelva a `FileSystems`, en la vista Explorer. Haga clic con el botón derecho en el directorio y elija "refresh" (actualizar). Aparecerá el archivo recién guardado.

Ejercicio2

- B. Cree un proyecto llamado `Electronics` y añada todos los archivos descomprimidos a su proyecto (para ello, haga clic con el botón derecho sobre el archivo y elija `Tools->AddToProject`).
- C. La figura 1 es el diagrama de clases del proyecto `Electronics`. Muestra la relación entre las clases, así como todos sus campos de datos.
- `Electronic`, la clase padre de `Computer` y `Cellphone`, es una clase abstracta.
 - `Computer` es la clase padre de `Laptop`, pero no es abstracta. Esto significa que puede crear un objeto de tipo `Computer` (que, en nuestro caso, representará un ordenador portátil).

Observe el diagrama de clases y asegúrese de que es coherente con las clases de Java que ha descargado.

Diagrama de clases

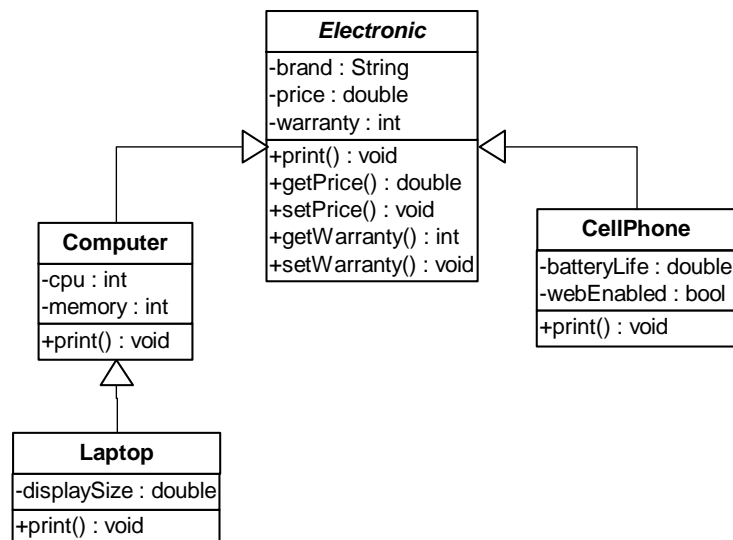


Figura1

Cómo crear unArray

- A. En el proyecto `Electronics`, cree una clase `Main` llamada `ElectronicMain` con un método `main()` vacío.
- B. En el método `main()`, cree un array capaz de almacenar tres objetos de tipo `Electronic`.
- C. Añada a este array 3 objetos con las siguientes especificaciones:
 - Cell phone: Brand: Nokia TS200; Price: \$300; Warranty: 18 months; Battery Life: 3.5Hr; Web Enabled: true;
 - Desktop (Computer): Brand: Dell D2100; Price: 1000; Warranty: 24 months; CPU speed: 1500 MHz; Memory: 512 MB;
 - Laptop: Brand: HP N5170; Price: 1500; Warranty: 24 months; CPU speed: 900 MHz; Memory: 256 MB; Display Size: 15"
- D. Imprima las especificaciones de todos los elementos contenidos en el array.

Cómo añadir funcionalidad

- A. Necesitamos un método para incrementar el precio de un aparato electrónico en un porcentaje dado. El método debería tener la siguiente firma:

```
public void increasePrice(int percentage)
```

- Este método no se comporta igual en todos los aparatos electrónicos:
 - Cuando el precio de un ordenador aumenta en cualquier porcentaje, su garantía aumenta en 12 meses.
 - Cuando el precio de un teléfono móvil aumenta en cualquier porcentaje, su garantía aumenta en 6 meses.

Cómo añadir funcionalidad(2)

B. Añada esta funcionalidad a las clases:

- Nota: la clase `Electronic` debería declarar como abstracto el método `increasePrice(int percentage)` ,y por lo tanto, la función debe implementarse en cada una de las clases derivadas.

C. En el método `main()` aument e el precio de todos los elementos del array en un 5%. Imprima de nuevo todos los elementos. El resultado debería ser el siguiente:

- Marca:NokiaTS200;Precio:\$315.0;Garantía:24meses
Batería:3.5h;Acceso a internet:true
- Marca:DellD2100;Precio:\$1050.0;Garantía:36meses
Velocidad de la CPU:1500MHz;Tamaño de memoria:512MB
- Marca:HPN5170;Precio:\$1575.0;Garantía:36meses
Velocidad de la CPU:900MHz;Tamaño de memoria:256MB
Tamaño de la pantalla:15.0"

Recuento de aparatos electrónicos (Opcional)

A. Necesitamos saber el número de aparatos electrónicos que hemos creado. Debería añadir los siguientes campos de datos en las clases apropiadas:

- `numberElectronics`
- `numberCellPhones`
- `numberComputers`
- `numberLaptops`

Estos campos de datos deben ser originalmente inicializados a 0.

Recuento de aparatos electrónicos (Opcional)

- B. Cada vez que se crea un nuevo objeto, deberían incrementarse en 1 los contadores correspondientes. Según nuestro array de ejemplo usted debería tener:
- `numberElectronics = 3`
 - `numberCellPhones = 1`
 - `numberComputers = 2` (recuerde que un `Laptop` es un `Computer`)
 - `numberLaptops = 1`
- C. En cada clase, cree un método `getCount()` que devuelva el contador de la clase. Debería poder llamar a `getCount()` sin crear ningún objeto.
- D. Desde el método `main()`, llame a la función `getCount()` de todas las clases (`Electronic`, `CellPhone`, `Computer` y `Laptop`). Compile y ejecute.