Introducción

Cuando una clase hereda de una a otra, adquiere sus atributos y métodos visibles, así la clase hija pudiendo reutilizar estas y pudiendo extender en función de las necesidades

8.1 Subclase y Superclase

Una subclase es la clase hija, es decir , la clase que hereda de la superclase(clase padre).
La subclase se amplía con la superclase y con lo que se añada a esto.

Por ejemplo, empleado y persona;
Persona es la superclase y empleado es la subclase.
Llevado a la práctica , la clase empleado heredaría de persona todos los atributos/métodos que no tengan el modificador private, y además la clase empleado tendría sus atributos/métodos propios.

```
class Persona {
     String nombre;
     byte edad;

    double estatura;
}
class Empleado extends Persona {
     double salario;
     Empleado(String nombre, byte edad, double estatura, double salario) {
            ...
     }
}
```

Java solo permite herencia simple, es decir, una clase hija solo puede tener una clase padre

8.2 Modificador de acceso para herencia:

Se heredan todos los miembros de una clase **excepto los privates**, que no son accesibles en la subclase.(Aun así, se puede acceder a ellos indirectamente mediante un método no privado).

Para que un miembro sea accesible desde una subclase, podemos hacer uso de un nuevo modificador de acceso,protected.

Funciona parecido a la visibilidad por defecto a diferencia de que los miembros con protected serán siempre visibles para las clases que hereden.(Un miembro protected es visible en las clases vecinas, no para las externas, pero **siempre es visible independientemente al paquete al que pertenezca desde una clase hija**.)

Tabla 8.1. Alcance de la visibilidad (incluida la herencia) según el modificador de acceso

	Visible desde...			
	la propia clase	clases vecinas	subclases	clases externas
`private`	✓			
sin modificador	✓	✓		
`protected`	✓	✓	✓	
`public`	✓	✓	✓	✓

8.3 Redefinición de miembros heredados

Cuando una clase hereda de otra , en alguna ocasión puede ocurrir que interese modificar atributos, o redefinir algún método.
Este método se conoce como **ocultación para** los **atributos** y **Overriding para** los **métodos.**
Consiste en declarar un miembro con igual nombre a uno heredado, lo que hace que este se oculte y siendo sustituido por el nuevo.

Ejemplo:

SuperClase

```
class Persona {
     String nombre;
     byte edad;
     double estatura;
     void mostrarDatos() {
             System.out.println(nombre);
             System.out.println(edad);
             System.out.println(estatura);
     }
}
```

SubClase

```
class Empleado extends Persona {
     double salario;
     @Override //significa: sustituye un método de la superclase
     void mostrarDatos() {
             System.out.println(nombre);
             System.out.println(edad);
             System.out.println(estatura);
             System.out.println(salario);
     }
}
```

8.3.1 super y super():

Igual que la palabra this, indica a la propia clase, super hace referencia a la superclase(clase padre) donde se usa.

Desde un constructor de la clase hija, podemos usar super para hacer referencia a los atributos de la clase padre.

En caso de que el constructor de la clase padre esté sobrecargado, podemos variar los parámetros de entrada super() en el número o en el tipo para coincidir con la versión del constructor de la clase padre.

Obligatoriamente, si vamos a usar super(), debe de ser la primera instrucción.

Al declarar super(), los atributos privados también aparecerán, sin embargo debemos inicializarlos.

8.3.2 Selección dinámica de métodos

8.4 La clase Object

La clase Object del paquete java.lang es una clase de la que heredan todas las clases de Java. Es la superclase/clase padre por excelencia, ya que se sitúa en la cima de la estructura de las herencias.

Todas las clases descienden de Object.

¿Cuál es el objetivo de todas las clases hereden de Object?:

-Que todas las clases implementen un conjunto de métodos de uso universal en Java(realizar comparaciones entre objetos, clonarlos, representar un objeto como una cadena.)
La función de estos métodos se ser implementados a la medida de cada clase.

-Poder referenciar cualquier objeto, de cualquier tipo, mediante una variable tipo Object.

Si queremos ver los métodos de Object heredados por otra clase, escribiremos en netbeans una variable de tipo clase(Persona si la clase es Persona, por ejemplo.) seguida de un punto.

Se desplegarán todos los atributos y métodos disponibles más los heredados de Object.

Los métodos más importantes de Object son los siguientes puntos:

8.4.1 Método toString():

Este método se piensa para que devuelva una cadena que represente al objeto que lo invoca con toda la información que nos interese mostrar.

Su uso en la clase Object es devolver el nombre de la clase a la que pertenece el objeto seguido de una arroba y la referencia del objeto.

```
"paquete.Persona@2a139a55"
```

Para implementar el método que hemos creado nosotros tenemos que sobreescribirlo con @Override

8.4.2 Método equals():

Compara dos objetos y decide si son iguales, devolviendo true en caso positivo y false en caso negativo. Su prototipo en la clase Object es:

public boolean equals(Object otro)

Se debe de usar .equals obligatoriamente sino no seria con Override, sería sobrecarga.

8.4.3 Método getClass():

Es común usar una variable Object para referenciar un objeto de cualquier clase.

El método getClass devuelve su clase , que a su vez es un objeto de la clase Class

8.5 Clases abstractas:

Hay métodos que no podemos implementar en una clase determinada por falta de datos, pero si podemos hacerlo en sus subclases, donde tienen los atributos necesarios.

La idea es implementarlos vacíos, solo con el prototipo en la clase padre y en las clases hijas hacer overriding.

Un método definido en una clase, pero cuya implementación se delega en las clases hijas, se conoce como abstracto.

Para declara un método abstracto se le antepone el modificador abstract y se declara el nombre de la función y nada más(No tiene cuerpo)

Las clases hijas deberán implementar el método abstracto, cada una con las especificaciones de la clase, que no están en la clase padre.

Toda clase que tiene un método abstracto debe ser declarada, a su vez, abstract.

No se pueden crear objetos en las clases abstractas.

Las clases abstractas existen para ser heredadas por otras, y no para ser instanciadas.

Si una clase hereda de una abstracta, pero deja alguno de sus métodos abstractos sin implementar ,también será abstracta.

Una clase abstracta puede tener algún método implementado y algunos atributos definidos que heredarán las clases hijas, pudiendo sustituir o ocultar estos.

Preguntas finales tipo test(Lo implementa Ja1ro)

1.Sobre una subclase es correcto afirmar que:
a)Tiene menos atributos que su superclase
b)Tiene menos miembros que su superclase
c)Hereda los miembros no privados de su superclase
d)Hereda los miembros de su superclase

Explicación:
Una subclase hereda los miembros no privados (atributos y métodos públicos y protegidos) de su superclase, pero no necesariamente tiene menos atributos o miembros que su superclase. El número de atributos y miembros puede variar en función del diseño de la jerarquía de clases y de las necesidades específicas de la subclase.

2. En relación con las clases abstractas es correcto señalar que:
a) Implementan todos sus métodos
b) No implementan ningún método
c) No tienen atributos
d) Tienen algún método abstracto

Explicación:
Las clases abstractas en la programación orientada a objetos son aquellas que no se pueden instanciar directamente y que suelen contener al menos un método abstracto. Los métodos abstractos son aquellos que no tienen una implementación concreta en la clase abstracta, sino que se definen solo como una firma o prototipo, dejando la implementación real a las subclases que heredan de la clase abstracta. Las clases abstractas pueden tener atributos y métodos concretos, es decir, métodos que tienen una implementación definida en la clase abstracta. Sin embargo, es común que al menos uno de sus métodos sea abstracto.

3 ¿En qué consiste la sustitución o overriding?:

a) En sustituir un método heredado por otro implementado en la propia clase

b) En sustituir un atributo por otro del mismo nombre

c) En sustituir una clase por un subclase

d) En sustituir un valor de una variable por otro

Explicación:

La respuesta correcta es la opción A: "En sustituir un método heredado por otro implementado en la propia clase". La sustitución o "overriding" es un concepto de programación orientada a objetos que permite a una clase hija proporcionar su propia implementación de un método heredado de su clase padre. Al hacerlo, la clase hija puede personalizar o modificar el comportamiento del método para adaptarse a sus propias necesidades sin cambiar la funcionalidad del método original en la clase padre.

4 Sobre la clase object es cierto indicar

a) Es abstracta

b) Hereda de todas las demás

c) Tiene todos sus métodos abstractos

d) Es superclase de todas las demás

Explicación:

La respuesta correcta es la opción D: "Es la superclase de todas las demás". La clase Object es una clase especial en Java que se considera la raíz de la jerarquía de clases de Java. Todas las clases de Java, directa o indirectamente, heredan de la clase Object. Esta clase proporciona algunos métodos comunes a todas las clases de Java, como `toString()`, `equals()` y `hashCode()`.

5 Cuál de las siguientes afirmaciones sobre el método equals() es correcta?

a) Hay que implementarlo, ya que es abstracto

b) Sirve para comparar solo objetos de la clase object

c) Se hereda de object pero debemos implementarlo al definirlo en una clase

d) No hay que implementarlo, ya que se hereda de object

Explicación:

La respuesta correcta es la opción C: "Se hereda de object pero debemos implementarlo al definirlo en una clase". El método `equals()` se hereda de la clase Object, pero a menudo es necesario que lo implementemos en nuestras propias clases para proporcionar una comparación personalizada entre objetos de la misma clase. La implementación predeterminada de `equals()` en la clase Object compara

simplemente si dos objetos son iguales mediante referencia, es decir, si ambos objetos apuntan a la misma ubicación en la memoria. Por lo tanto, si queremos comparar el contenido o los valores de dos objetos de nuestra propia clase, debemos proporcionar una implementación personalizada de `equals()` que haga esa comparación

6 Cuál de las siguientes afirmaciones sobre el método toString() es correcta?
a) Sirve para mostrar la información que nos interesa de un objeto
b) Convierte automáticamente un objeto en una cadena
c) Encadena varios objetos
d) Es un método abstracto de object que tenemos que implementar
Explicación:

La respuesta correcta es la opción A: "Sirve para mostrar la información que nos interesa de un objeto". El método `toString()` en Java se utiliza para devolver una representación de cadena del objeto. En otras palabras, convierte un objeto en una cadena de caracteres. Este método está definido en la clase Object y es heredado por todas las demás clases de Java. La implementación predeterminada de `toString()` en la clase Object devuelve una cadena que contiene el nombre de la clase del objeto, seguido de su dirección de memoria en formato hexadecimal. Por lo tanto, al redefinir el método `toString()` en una clase personalizada, podemos proporcionar una representación en cadena más significativa y útil del objeto, que muestre la información relevante para nuestro programa.

7 Cuál de las siguientes afirmaciones sobre el método getClass() es correcta?
a) Convierte objetos en clases
b) Obtiene la clase a la que pertenece un objeto
c) Obtiene la superclase de una clase
d) Obtiene una clase a partir de su nombre
Explicación:

La respuesta correcta es la opción B: "Obtiene la clase a la que pertenece un objeto". El método `getClass()` es un método de la clase Object que se utiliza para obtener la clase a la que pertenece un objeto. Devuelve un objeto de tipo `Class`, que proporciona información sobre la clase, como su nombre, los métodos y campos que contiene, y otras propiedades. Por lo tanto, podemos utilizar el método `getClass()` para obtener información sobre un objeto en tiempo de ejecución, lo que puede ser útil en situaciones en las que necesitamos realizar operaciones dinámicas o reflexivas en los objetos.

8 Una clase puede heredar

a) De una clase

b) De dos clases

c) De todas las clases que queramos

d) Solo de la clase object

Explicación:

La respuesta correcta es la opción A: "De una clase". En Java, una clase puede heredar de una única clase padre directa. Este mecanismo se conoce como herencia de una sola clase y es una característica fundamental de la programación orientada a objetos. La clase padre proporciona los campos y métodos que se pueden heredar por la clase hija, lo que permite a la clase hija reutilizar y extender el código de la clase padre. La herencia múltiple, en la que una clase hija hereda de varias clases padres, no es compatible en Java, aunque se pueden simular algunos de sus efectos utilizando interfaces.

9.La selección dinámica de métodos:

a) Se produce cuando una variable cambia de valor durante la ejecución de un programa.

b) Es el cambio de tipo de una variable en tiempo de ejecución.

c) Es la asignación de un mismo objeto a más de una variable en tiempo de ejecución.

d) Es la ejecución de distintas implementaciones de un mismo método, asignando objetos de distintas clases a una misma variable, en tiempo de ejecución.

Explicación:

La respuesta correcta es la opción D: Es la ejecución de distintas implementaciones de un mismo método, asignando objetos de distintas clases a una misma variable, en tiempo de ejecución". La selección dinámica de métodos es un concepto de la programación orientada a objetos que permite la ejecución de distintas implementaciones de un mismo método, asignando objetos de distintas clases a una misma variable, en tiempo de ejecución. Esto permite que el comportamiento de un programa pueda ser modificado durante su ejecución, sin necesidad de modificar su código fuente.

8.10.¿Cuál de las siguientes afirmaciones sobre el método super ( ) es correcta?

a) Sirve para llamar al constructor de la superclase.

b) Sirve para invocar un método escrito más arriba en el código.

c) Sirve para llamar a cualquier método de la superclase.

d) Sirve para hacer referencia a un atributo de la superclase.

Explicación:

La respuesta correcta es la opción c)

"Sirve para llamar a cualquier método de la superclase". En la programación orientada a objetos, la palabra clave "super" se utiliza para hacer referencia a la superclase de una clase y para acceder a sus métodos y atributos. El método "super()" se utiliza específicamente para llamar a cualquier método de la superclase desde la subclase. Cuando se llama a un método utilizando "super", se invoca la versión del método que está definida en la superclase, en lugar de la versión que puede estar definida en la subclase. Esto permite que la subclase pueda extender o modificar el comportamiento de los métodos heredados de la superclase, pero aún así utilizar la versión original del método de la superclase.