

Encapsulación

- Mantiene las variables de instancia protegidas
- Crea métodos de acceso público para modificar los valores de la variable de instancia.

Herencia

2 razones de uso más comunes para el uso de herencias son:

- Para promover el uso del código
- Para el uso de polimorfismos

Términos de Orientado a Objetos para la herencia son los siguientes:

Vehículo es la superclase de Carro.

Carro es la subclase de Vehículo.

Carro es la superclase de Subaru.

Subaru es la subclase de Vehículo.

Carro hereda de Vehículo

Subaru hereda de Vehículo y Carro.

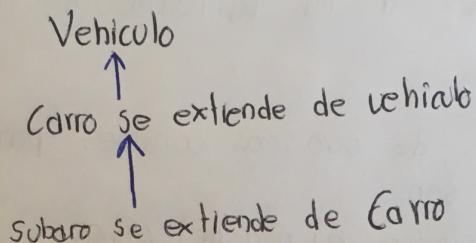
Subaru es derivado de Carro.

Carro es derivado de Vehículo

Subaru es derivado de Vehículo

Relación IS-A. Las siguientes afirmaciones son ciertas

Carro se extiende de Vehículo significa Carro IS-A Vehículo
Subaru se extiende de Carro significa Subaru IS-A Carro.



HAS-A

Las relaciones se basan en el uso, en lugar de herencia, se refiere a la instancia de la clase.

Polimorfismo

- Una variable de referencia puede ser de un solo tipo, y una vez declarada, ese tipo nunca puede cambiarse (aunque el objeto al que hace referencia puede cambiar).
- Una referencia es una variable, por lo que se puede reasignar a otros objetos (a menos que la referencia se declare definitiva).
- El tipo de una variable de referencia determina los métodos que se pueden invocar en el objeto al que hace referencia la variable.
- Una variable de referencia puede referirse a cualquier objeto del mismo tipo que la referencia declarada, o, este es el más grande, si puede referirse a cualquier subtipo del tipo declarado.

■ Una variable de referencia se puede declarar como un tipo de clase o un tipo de interfaz. Si la variable se declara como un tipo de interfaz, puede hacer referencia a cualquier clase que implemente la interfaz.

Java soporta únicamente de una sola herencia, esto quiere decir que una clase puede tener una sola inmediata superclase, no más de una.

En algunos lenguajes (como C++) permiten que una clase se extienda a más de una otra clase, esta capacidad se le conoce como "múltiples herencias". La razón por la que el creador de Java no permitió las "herencias múltiples" porque sería muy enmarañado.

Método polimórfico se aplica la invocación solamente para los métodos de instancia.

Sobreescritura / Sobre carga

Métodos sobreescritos

Reglas para Métodos sobreescritos.

- La lista de argumentos debe coincidir exactamente para el método sobreescrito. Si no coinciden, tu puedes terminarla con una sobrecarga del método que no pretendías.
- El tipo de retorno debe ser el mismo que, o un subtipo de, el tipo de retorno declarado en el método original anulado en la superclase.
- Los niveles de acceso no puede ser mas restrictivos que los métodos sobreescritos.
- Los niveles de acceso puede ser menos restrictivos que de los métodos sobreescritos.
- Los métodos de instancia pueden ser sobreescritos solo si son heredados por la subclase. Una subclase con el mismo paquete como la superclase de instancia puede sobreescribirse cualquier método de clase que no sea marcado private o final. Una subclase en un diferente paquete sobreescribirse solamente aquellos métodos públicos o protegidos no sean final.

- El método sobrescrito puede lanzar cualquier excepción sin marcar (en tiempo de ejecución), independientemente de si el método sobrescrito declara la excepción.
- El método sobrescrito no debe lanzar excepciones marcadas que sean nuevas o más amplias que las declaradas por el método anulado.
- El método sobrescrito puede lanzar más o menos excepciones. Justo porque un método sobrescrito toma riesgos no significa que la excepción de subclase sobrescrita tome los mismos riesgos.
- Tu no puedes sobreescibir un método con marca final.
- Tu no puedes sobreescibir un método con marca static.
- Si un método no puede ser heredado no puedes sobreescibirlo

Invocando una versión sobrecargada de un método sobreescrito.

El super funciona para invocar un método sobreescrito solo aplicando para la instancia del método.

Métodos sobrecargados

Sobrecarga: métodos que reusan el mismo nombre en una clase, pero con diferentes argumentos.

- Métodos sobrecargados deben cambiar la lista de argumentos
- Métodos sobrecargados pueden cambiar el tipo de retorno.
- Métodos sobrecargados pueden cambiar los modificadores de acceso.
- Métodos sobrecargados pueden declarar nuevas o excepciones.

Invocando Métodos Sobrecargados

Decidiendo cuales de los métodos repetidos para invocas es basandose en los argumentos de esta. Si tu pasas el método ~~cons~~ string, la versión sobre-cargada toma la string. y si invocas al mismo método pero con argumento float, llama la versión sobrecargada que toma float.

Polimorfismo en métodos sobrecargados y sobreescritos

Código de invocación	Método	Resultado
Animal a = new Animal(); a.eat();		Generic Animal Eating Generically
Horse h = new Horse(); h.eat();		Horse eating hay
Animal ah = new Horse(); ah.eat();		Horse eating hay El polimorfismo funciona: el tipo de objeto real (caballo) no el tipo de referencia (Animal) se utiliza para determinar a que se llama eat().
Horse he = new Horse(); he.eat("Apples");		Horse eating Apples El método sobre cargado eat(String) es invocado.
Animal a2 = new Animal(); a2.eat("treats");		Error de compilador, El compilador ve que la clase Animal no tiene un método eat que tome un String
Animal ah2 = new Horse(); ah2.eat("Carrots")		Error de compilador, El compilador todavía ve solo en la referencia, y ve que Animal no tiene un método eat que tome un String. El compilador no le importa que el objeto actual podría ser un caballo en tiempo de ejecución.

Diferencias entre métodos de sobrecarga y sobreescritura

	Métodos sobrecargados	Métodos sobreescritos
Argumentos	Debe cambiarse	No debe cambiarse
Tipo de retorno	Puede cambiarse	No puede cambiarse, excepción para retornos covariantes
Excepciones	Puede cambiarse	Puede reducirse o eliminarse
Accesos	Puede cambiarse	No debe ser más restrictivo (puede ser menos restrictivo)
Invocación	Tipo de referencia determina cual versión de sobrecarga (basado en los tipos de argumentos declarados) es seleccionado. El método actual que es invocado es todavía un método virtual de invocación que pasa en el tiempo de ejecución, pero el compilador ya sabrá la firma del método a invocar. Aunque esté en tiempo de ejecución, el argumento de coincidencia ya habrá sido clavado, simplemente no en la clase en la que vive el método	Tipo de objeto (en otras palabras, el tipo de la instancia actual en el montón) determina cual método es seleccionado en la ejecución

Casteando variables de referencia

Hemos visto cómo es posible y comúnmente utilizar tipos de variables de referencia genéricos para referirse a tipos de objetos más específicos. Esto es el corazón del polimorfismo.

No se puede castear objetos que no sean del mismo herencia.

Implementando una interface

Cuando implementa una interfaz, acepta adherirse al contrato definido en la interfaz. Eso significa que estás aceptando proporcionar implementaciones legales para cada método definido en la interfaz, y que cualquiera que sepa cómo se ven los métodos de la interfaz puede estar seguro de que puede invocar esos métodos en una instancia de su clase de implementación.

Por ejemplo, si crea una clase que implementa la interfaz Runnable (para que su código pueda ser ejecutado por un hilo específico), debe proporcionar el método public void run(). De lo contrario, podría dendirse al subproceso deficiente que execute el código del objeto Runnable y, sorprendentemente, el subproceso descubre que el objeto no tiene un método run().

Si la clase dice que está implementando una interfaz, sería mejor tener una implementación para cada método en la interfaz...

Clases de implementación debe adherir las mismas reglas para el método de implementación como una clase extendida en una clase abstracta.

Para ser una clase de implementación legal en orden, una clase de implementación no abstracta debe de seguir lo siguiente:

- Proporcionar implementaciones concretas (no abstractas) para todos los métodos desde la interfaz declarada.
- Siga todas las reglas para sobreescritura legales.
- No declarar excepciones comprobadas en los métodos de implementación distintos a los declarados por el método de interfaz, o subclases de los declarados por el método de interfaz.
- Mantener la firma del método de interfaz y mantener el mismo tipo de retorno (o un subtipo).

Dos reglas más que necesita saber y luego podemos poner este tema en reposo:

1. Una clase implementa mas que una interface. Esto es perfectamente legal decir, por ejemplo lo siguiente:
public class Ball implements Bounceable, Serializable, Runnable
{...}

Tu puedes extender una sola clase, pero implementas multiples interfaces

2. Una interface puede extenderse con otra interface, pero nunca implementar cualquiera. El siguiente código es perfectamente legal:

```
public interface Bounceable extends Moveable{ } //ok!
```