Java SE

Curso Avanzado de Java Standard Edition

Arturo Rivas Rojas

26 de diciembre de 2018

Índice general

CAPÍTULO 1

Polimorfismo

Posibilidad de construir un método, con comportamiento diferente según la clase a la que pertenece, cuando las clases están relacionadas ya sea por herencia, interfaces o clases abstractas.

1.1. Herencia

En la herencia el polimorfismo se lleva a cabo mediante la sobreescritura de métodos. Por ejemplo en el constructor normalmnete una subclase tiene los mismos atributos que la clase padre y algunos otros, por lo que su constructor debe ser diferente entonces usando la palabra super llamamos a la clase padre y podemos lalmar a su cosntructor reciclandolo y agregar lo que falte.

1.2. Interfaces

Solo pueden contener costantes y definición de métodos, es decir solo la firma del método si implementación. Siempre que se implementa una interfaz se esta realizando polimorfismo.

1.3. Clases Abstractas

En una clase abstracta no se imlementan todos los métodos y no se pueden crear instancias de estas clases.

```
public abstract class Figura{
    abstract void dibujate();
}
```

La palabra **abstract** es la clave para las clases abstractas y los métodos abstractos son los métodos que son obligatorios de implementar.

```
class Triangulo extends Figura{
    abstract void dibujate();
}
```

Los métodos abstractos se heredan y se puede decidir si implementar el método o conservar sus abstracción.

CAPÍTULO 2

JavaDoc

Genera documentación en HTML desde el código Java.

2.1. Funcionamiento

Java genera la documentación a traves de los comentarios.

```
// un comentario común de una sola linea.
/* comentario común
de multiples lineas */
/** comentario de documentación
* <h1> Titulo </h1>
* <p> saltos de linea
* */
```

solo los comentarios dentro del tercer tipo son tomados en cuenta para la generación del javadoc.

Dentro de los comentarios javadoc se pueden utilizar etiquetas de HTML para dar formato a nuestra salida.

2.2. Etiquetas

- **@author** : autor.
- **@version** : versión del código.
- **@since** : año, fecha.
- **@see** : sugerencia de visualización, como más información.
- **@param** : parámetros.
- **@return** : retorno del método.
- **{@link clase/interfaz}** : genera un link a la clase o interfaz.
- **{inheritDoc}** : historia del método, herencia o implementación de clases.

- **{code código}** : da formato al texto código.
- **@exception** or **@throws** : exceptiones que puede lanzar.
- **{@value #constant}** : muestra el valor de la constante.

CAPÍTULO 3

Clases Anidadas

Las clases anidadas, o nested classes. Nacen por la necesidad de tener agrupadas clases que no pueden o deben existir por separado.

Una clase anidada tiene acceso a los miembros (atributos y métodos) de su clase contenedora, incluidos los privados. Lo inverso no es verdadero.

3.1. Clasificación

- Static Nested Classes

  Se declaran utilizando la palabra reservada **static**. Y se diferencian por tener la limitación de no poder acceder a los miembros no staticos de su clase contenedora. Ademas de que no es necesario instancia a la clase contenedora para instanciar a una **static nested class**.

```
// clase contenedora
public class Outer{
    // clase anidada estática
    public static StaticInner{
        // método de la clase anidada
        public run(){
        }
    }
    // método main de la clase contenedora
    public static void main (String[] args){
        // instanciando a la clase anidada
        Outer.StaticInner nested =  new     Outer.StaticInner();
        // llamando a un método de la clase anidada.
        nested.run():
        // el método run() no es estático por lo que    fue necesario instaciar a la clase para
    }
}
```

- Inner classes

  Son clases que se declaran de la manera común, y se diferencia por que para instanciarlas es necesario intanciar a la clase contenedora. Son mucho mas versátiles que las estáticas ya que pueden llamar a todos los miembros de la clase que los contiene.

```
// clase contenedpra
public class Outer {
    //declaración de la clase interna
    public class Inner {
        //atributos y métodos de la clase anidada
    }
    // método main de la clase contenedora
    public static void main(String[] args) {
        // instanciando de la clase contenedora
        Outer outer = new Outer():
        // instanciando de la clase anidada
        outer.Inner inner = outer.new Inner();
    }
}
```

  - Method Local Inner Classes

    Es un subconjunto de las Inner Classes pues su unica diferencia es que solo existen dentro de uno de los métodos de la clase contenedora.

```
// clase contenedora
public class Outer {
    // método run() de la clase contenedora
    void run(){
        // declaración de la clase local de     método anidada
        public class Local {
            // método run de la clase anidada
            void run() {
                //implementación
            }
        }
        // instancia de la clase anidada
        Local local = new Local()
        // llamada al método de la clase    anidada
        local.run();
    }
    // método main de la clase contenedora
    public static void main(String[] args)
    {
        // instacia de la clase contenedora
        Outer outer = new Outer();
        // llamada al método run() de la    clase contenedora
        outer.run();
    }
}
```

  - Anonymous Inner Classes

    Son clases que nos permiten instanciar clases abstracta implementando los métodos abstractos de la clase abstracta al mismo tiempo en el que instanciamos el objeto.

```
// Abstracta.java
//clase abstracta
public abstract Abstracta {
 public abstract void run();
}
// Anonima.java
public class Outer {
    public void ejecutar() {
    // instanciando clase anónima
    Abstracta abstracta = new Abstracta() {
        void run() {
            // implementación
        }
    }
    // llamando método de la clase anónima
    abstracta.run();
    }
}
```