El futuro digital es de todos

MinTIC

UNIVERSIDAD DEL NORTE

Vigilada Mineducación

CICLO II:

Programación Básica en Java

MisionTIC 2022

El futuro digital es de todos | MinTIC

UNIVERSIDAD DEL NORTE
Vigilada Mineducación

Sesión 7:
Introducción a Java

Programación Orientada a Objetos (POO)

MisiónTIC2022

Objetivos de la sesión

Al finalizar esta sesión estarás en capacidad de:

1. Aplicar adecuadamente el cast de un tipo de dato a otro y de igual forma usar la promoción de argumentos en un método.
2. Usar la referencia this y demostrar su uso en un programa de aplicación
3. Explicar y aplicar el encapsulamiento en Java de variables y métodos a través de la construcción de una aplicación
4. Definir y aplicar los métodos set y get con variables de instancias
5. Implementar los métodos set y get para modificar las variables privadas definidas dentro del encapsulamiento.

Cast

Es un procedimiento para transformar una variable primitiva de un tipo a otro, o transformar un objeto de una clase a otra clase siempre y cuando haya una relación de herencia entre ambas. Dentro del casting de variables primitivas se distinguen dos clases:

- Implícito: no se necesita escribir código para que se lleve a cabo. Ocurre cuando se realiza una conversión ancha (widening casting), es decir, cuando se coloca un valor pequeño en un contenedor grande.

- Explícito: sí es necesario escribir código. Ocurre cuando se realiza una conversión estrecha (narrowing casting), es decir, cuando se coloca un valor grande en un contenedor pequeño. Son susceptibles de pérdida de datos.

Cast

byte -> short -> int -> long -> float -> double

Conversión implícita .

double -> float -> long -> int -> short -> byte

Conversión explícita.

Cast

- Cambios de tipo automáticos
- De int a float, de float a int

```
int a;
float b;
a = (int) b;   // Se pierde información
b = (float) a;  // No es necesario
```

Cast de Tipos Primitivos

- Puede perderse precisión

```
double d = 20.5;
long l = (long) d;
System.out.println(l);
```

```
20
```

- Pueden perderse dígitos

```
long l = 1000000;
short s;
s = (short) l;
System.out.println(s);
```

```
1696
0
```

Cast

Convertir desde	Convertir a...							
	boolean	byte	short	char	int	long	float	double
boolean		no	no	no	no	no	no	no
byte	no		si	cast	si	si	si	si
short	no	cast		cast	si	si	si	si
char	no	cast	cast		si	si	si	si
int	no	cast	cast	cast		si	si*	si*
long	no	cast	cast	cast	cast		si*	si*
float	no	cast	cast	cast	cast	cast		si
double	no	cast	cast	cast	cast	cast	cast	

Donde:

- no: indica que no hay posibilidad de conversión.
- si: indica que el casting es implícito.
- si*: indica que el casting es implícito pero se puede producir pérdida de precisión.
- cast: indica que hay que hacer casting explícito.

This

- Dentro de un método, hace referencia al objeto al que se aplica el método.
- Sólo aparece en métodos de instancia
    - Si no hay instancia, no hay this
- Se puede usar para acceder a variables aunque estén ocultas.
- Si se intenta usar dentro de un método estático lanzará:
  “Cannot use This in a static context”

This

```
public class Prueba {
        private int a;  // declara la variable de instancia
        public void metodo() {
                int a = 0;   // declaración local. Oculta la variable de instancia
                a = 3;       // accede a la variable local
                this.a = 2;  // accede a la variable de instancia
        }
}
```

Final

La palabra clave “final” se usa en diferentes contextos. En primer lugar, final es un modificador de no acceso aplicable solo a una variable, un método o una clase.

La palabra clave reservada final hace que el compilador genere un error de compilación cuando a una variable final se le intenta asignar un nuevo valor y se utiliza en tres contextos:

- Variables con palabra clave final
- Clases finales (No se incluye aquí)
- Métodos finales(No se incluye aquí)

Final

Variables con palabra clave final

Algunas de las variables de instancia necesitan modificarse, mientras que otras no. Para esto se puede utilizar la palabra clave **final** para especificar que una variable no puede modificarse (es decir, que sea una constante) y que cualquier intento por modificarla sería
un error. Por ejemplo:

```
private final int INCREMENTO;
```

Encapsulamiento de variables.

- Es un principio del lenguaje Java cuya filosofía es hacer que los atributos de las clases se puedan editar sólo a través de métodos.
- Se hace teniendo las propiedades como privadas y métodos que la controlan públicos. Comúnmente, se crea un grupo de métodos llamados getters (se encargan de establecer el valor de la propiedad) y setters (se encargan de setearla).
- Mantener las clases con el principio de encapsulamiento nos permite controlar el cambio de valor que pueda producirse en ellas añadiendo validaciones.

Ejemplo:

```
public class Gato {
    public String nombre;
    public int patas;
}

//Mi método main...
Gato g = new Gato();
g.nombre = "Nemo";
g.patas = 4;
System.out.println("Mi gato se llama: " +
g.nombre);
System.out.println("El número de patas
de mi gato es: " + g.patas);
```

Sin encapsulamiento

//

```
public class Gato {
    private String nombre;
    private int patas;
    public getNombre(){ return nombre;}
    public getPatas(){ return patas;}
    public setNombre(String nuevoNombre){ nombre =
nuevoNombre;}
    public setPatas(int numeroPatas){ patas = numeroPatas;}
}

//Mi método main...
Gato g = new Gato();
g.setNombre("Nemo");
g.setPatas(3);
System.out.println("Mi gato se llama: " + g.getNombre());
System.out.println("El número de patas de mi gato es: " +
g.getPatas());
```

Con encapsulamiento

¿Por qué debemos aplicar el principio de encapsulamiento?

El uso de la encapsulación nos permite construir estructuras más complejas relacionándolas e incrementando la flexibilidad y la capacidad de reutilización de nuestro código. Encapsular nos permite:

- Mantenibilidad, estabilidad y seguridad principalmente: Tras aplicar el principio de encapsulamiento, ya no tenemos acceso a las propiedades directamente, por lo que tenemos que usar los métodos (que son públicos y sí tenemos acceso a ellos) para poder acceder a las variables, tanto para modificarlas como para obtenerlas.

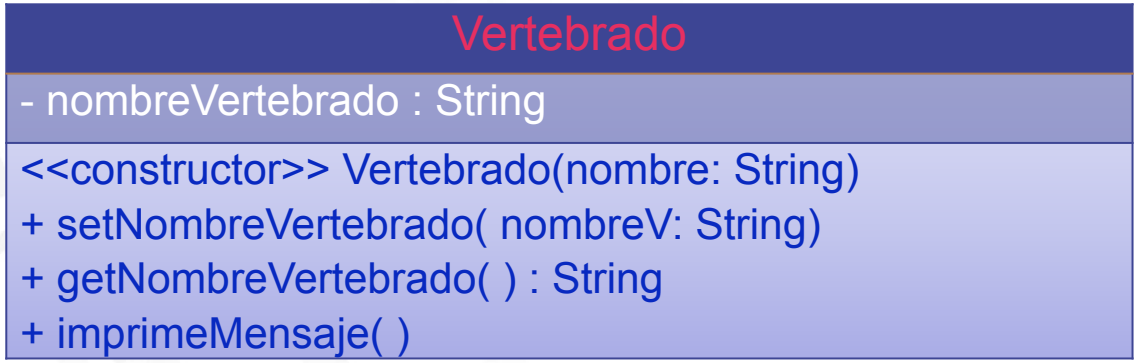
Estructura en Java usando get y set

```
public class Miclase {
        private TipoAtributo Atributo;

        //Constructor
        public Miclase(TipoAtributo AtributoT){
                Atributo=AtributoT;
        }
        //Método set
        public void setAtributo (TipoAtributo Variable) {
                Atributo=Variable;
        }
        //Método get
        public TipoAtributo getAtributo() {
                return Atributo;
        }
}
```

Constructor en Java usando get y set (clase Vertebrado)

```
public class Vertebrado {
        private String nombreVertebrado;

        public Vertebrado(String nombre){
                NombreVertebrado=Nombre;
        }
        public void setNombreVertebrado (String NombreV) {
                NombreVertebrado=NombreV
        }
        public String getNombreVertebrado() {
                return NombreVertebrado
        }
}
```

Diagrama de Clase Vertebrado con constructor usando set y get



El futuro digital es de todos

MinTIC

UNIVERSIDAD DEL NORTE

Vigilada Mineducación

¡GRACIAS

POR SER PARTE DE ESTA EXPERIENCIA DE APRENDIZAJE!

Mision TIC 2022