CURSO:
PROGRAMACIÓN ORIENTADA A OBJETOS

Docente: Ing. CIP Loncán Salazar, Pierre Paul

SESIÓN Nº 11

CLASES ABASTRACTAS

Una clase abstracta…

es una clase que no se puede instanciar

se usa únicamente para definir subclases

¿Cuándo es una clase abstracta?

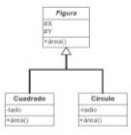
En cuanto uno de sus métodos no tiene implementación (en Java, el método abstracto se etiqueta con la palabra reservada `abstract`).

EJEMPLO DE CLASES ABSTRACTAS

- Clases Abstractas: ObjetoGrafico y Paralelogramo
- En el programa de dibujo sólo se van a crear objetos gráficos concretos de : puntos, elipses, círculos, cuadrados, rectángulos, rombos o romboides.

¿CUÁNDO SE UTILIZA UNA CLASE ABSTRACTA?

Cuando deseamos definir una abstracción que englobe objetos de distintos tipos y queremos hacer uso del polimorfismo.



`Figura` es una clase abstracta (nombre en cursiva en UML) porque no tiene sentido calcular su área, pero sí la de un cuadrado o un círculo. Si una subclase de Figura no redefine `area()`, deberá declararse también como clase abstracta.

¿CUÁNDO SE UTILIZA UNA CLASE ABSTRACTA?

- Para definir una clase como abstracta se coloca la palabra reservada ***abstract*** antes de ***class***:
  - `public abstract class ObjetoGrafico {...`
- Si en la clase abstracta se quiere obligar a que las subclases implementen un determinado método, basta declararlo como método abstracto. No tendrá cuerpo y terminará en punto y coma:

```
public abstract class ObjetoGrafico { // Abstracta
    public abstract String toString();
    public abstract void desplaza();
    public abstract boolean esCerrada();
    public abstract double area();
}
```

Métodos Abstractos

  - Para definir un **método como abstracto** se coloca la palabra reservada ***abstract*** antes de este
- Las subclases (no abstractas) no podrán compilarse si no implementan métodos con esos prototipos.

EJEMPLO…

```
public abstract class Figura
{
  protected double x;
  protected double y;
  public Figura (double x, double y)
  {
    this.x = x;
    this.y = y;
  }
  public abstract double area ();
}
```

```
public class Circulo extends Figura
{
  private double radio;
  public Circulo (double x, double y,
  {
    super(x,y);
    this.radio = radio;
  }
  public double area ()
  {
    return Math.PI*radio*radio;
  }
}
```

```
public class Cuadrado extends Figura
{
  private double lado;
  public Cuadrado (double x, double y, double lado)
  {
    super(x,y);
    this.lado = lado;
  }
  public double area ()
  {
    return lado*lado;
  }
}
```

Ing. CIP Loncán Salazar, Pierre Paul

7

POLIMORFISMO

- El concepto de Polimorfismo es uno de los fundamentos para cualquier lenguaje orientado a Objetos, las mismas raíces de la palabra pueden ser una fuerte pista de su significado: **Poli = Multiple, morfismo= Formas** , esto implica que un mismo Objeto puede tomar diversas formas.
- A través del concepto de Herencias ("Inheritance") es posible ilustrar este comportamiento:

Ing. CIP Loncán Salazar, Pierre Paul

8

POLIMORFISMO

- Hay varias formas de polimorfismo:
  - Cuando invocamos el mismo nombre de método sobre instancias de distinta clase
  - cuando creamos múltiples constructores
  - cuando vía subtipo asignamos una instancia de una subclase a una referencia a la clase base.
- Cuando creamos una clase derivada, gracias a la relación es-un podemos utilizar instancias de la clase derivada donde se esperaba una instancia de la clase base. También se conoce como principio de subtitución.

Ing. CIP Loncán Salazar, Pierre Paul

9

POLIMORFISMO

- Sea:

```
class Employee { ..... }
class Manager extends Employee { .... }
```

- Employee e; //declaración de un objeto de Employee

```
e=new Employee(...); // instancia
e=new Manager(..);   // OK. Sustitución
```

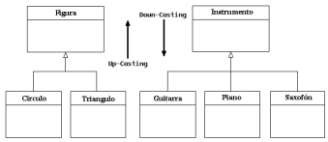
- En el primer caso a través de e tenemos acceso a todo lo correspondiente a un Employee.
- En el segundo caso tenemos acceso a todo lo correspondiente a Employee, pero con la implementación de Manager.
- Al revés no es válido porque toda referencia a Manager debe disponer de todos los campos.

Ing. CIP Loncán Salazar, Pierre Paul

10

USO DE CASTING

- El termino "Casting" viene de la palabra "Cast" que significa Molde, por lo que el termino literal es Hacer un Molde, en Polimorfismo se lleva acabo este proceso de "Casting" implícitamente, una Guitarra se coloca en el molde de un Instrumento, un Triangulo en el molde de una Figura.

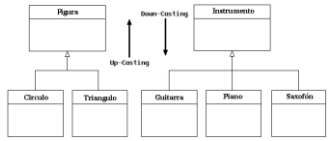


Ing. CIP Loncán Salazar, Pierre Paul

11

USO DE CASTING

- Anteriormente se mencionó que el "Casting" llevado acabo con Polimorfismo es implícito, esto se debe a que no se requiere de sintaxis especial, simplemente se convierte una Guitarra a un Instrumento, sin embargo, para llevar una transformación en sentido opuesto se requiere de sintaxis adicional para mantener la seguridad de transformación; analicemos: mientras se puede asegurar que un Triangulo es una Figura ("Up-Casting"), pero una Figura no necesariamente es un Triangulo, claro esta que lo puede ser, pero en Java se requiere definir explícitamente esta operación ("Down-Casting").



Ing. CIP Loncán Salazar, Pierre Paul

12

Polimorfismo con Late Binding

- El poder manipular un Objeto como si éste fuera de un tipo genérico otorga mayor flexibilidad al momento de programar con Objetos, el término Polimorfismo también es asociado con un concepto llamado Late-Binding (Ligamiento Tardío), observe el siguiente fragmento de código:
  - Figura a = new Circulo();
  - Figura b = new Triangulo();
- Inicialmente se puede pensar que este código generaría un error debido a que el tipo de referencia es distinta a la instancia del objeto, sin embargo, el fragmento anterior es correcto y demuestra el concepto de Polimorfismo; para asentar este tema se describe un ejemplo más completo:


```
public class Animal {
    public String hacerRuido(){
        return "No definido";
    }
}
```

```
public class Mamifero extends Animal{
    public String mensaje(){
        return "Soy Mamifero";
    }
}
```

```
public class Perro extends Mamifero{
    public String mensaje(){
        return "Soy Perro";
    }
    public String hacerRuido(){
        return "guau guau";
    }
}
```

Up-Casting

```
Mamifero m=new Perro(); //Upcasting
jtxaDatos.append(m.mensaje()+"\n");
jtxaDatos.append(m.hacerRuido()+"\n");
```

Down-Casting

```
//generando un objeto
 Animal a=new Perro();
//convirtiendo el objeto
Perro nuevoP=(Perro)m; //DOWN-CASTING

jtxaDatos.append(nuevoP.mensaje()+"\n");
jtxaDatos.append(nuevoP.hacerRuido()+"\n");
```


Tecnicas de Casting

- Consiste en realizar las conversiones de tipo, no modifican al objeto, solo su tipo.
- UPCASTING: permite interpretar un objeto de una clase derivada como del mismo tipo de la clase base. No hace falta especificarlo.
- DOWNCASTING: Permite interpretar un objeto de una clase base como del mismo tipo que su clase derivada. Se especifica precediendo al objeto a convertir con el nuevo tipo entre paréntesis.

Operador InstanceOf

- El operador instanceof sirve para consultar si un objeto es una instancia de una clase determinada, o de su padre. Se utiliza para evitar hacer casting de objetos a la hora de tratar un objeto de una forma y otra, llamando a un método de una clase o de otra dependiendo de qué tipo de objeto sea. Ejemplo

```
public class Empleado{..}
public class Jefe extends Empleadp{..}
public class Constructor extends Empleado{..}
```

```
public void metodo(Empleado e)
{
   if ( e instanceof  Jefe)
          //Obtener beneficios por su salario
   else if (e instanceof  Constructor)
          //obtener tarifa por hora
   else //empleados temporales
}
```