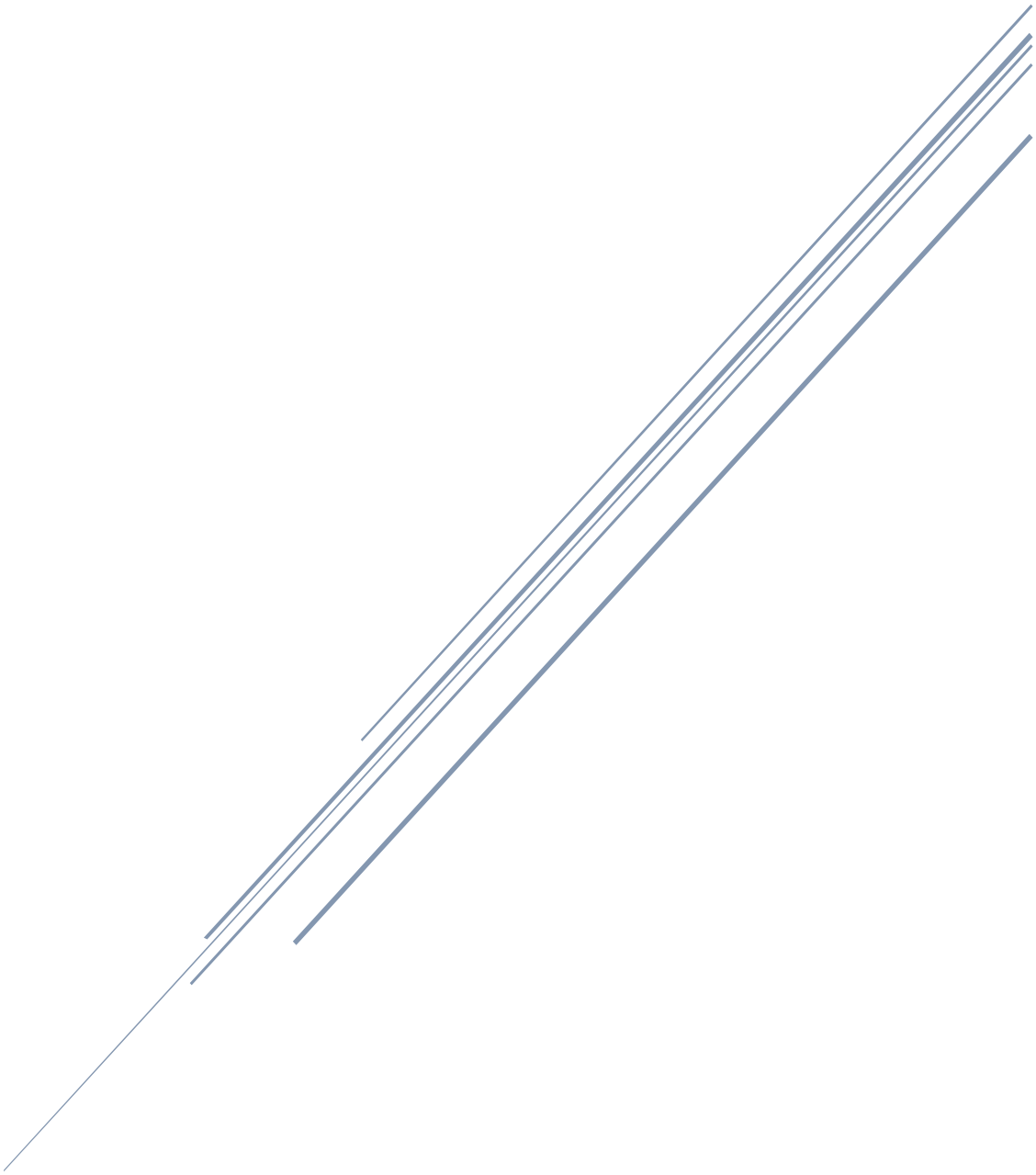
METHOD DISPATCHING AND METHOD OVERRIDING

MFC – Modelos Formales de Computación



MITSS
Sergi Sanz Carreres

Índice:

¿Qué es el Method Dispatching?

El Method Dispatching es el algoritmo utilizado para decidir qué método debe invocarse en respuesta a un mensaje. Estos algoritmos varían significativamente dependiendo del lenguaje de programación.

Existen dos tipos de method dispatching, el static dispatch y el dynamic dispatch, veámoslos con más detalle:

Static Dispatch

El static dispatch es una forma de polimorfismo en que los métodos se resuelven en tiempo de compilación de manera que en el momento en que se produce la ejecución el programa ya sabe que métodos se van a utilizar. La principal ventaja que presenta respecto al dinamic dispatch, es que debido a que no debe calcular que método utilizar en tiempo de ejecución presenta tiempos de ejecución menores.

Dynamic Dispatch

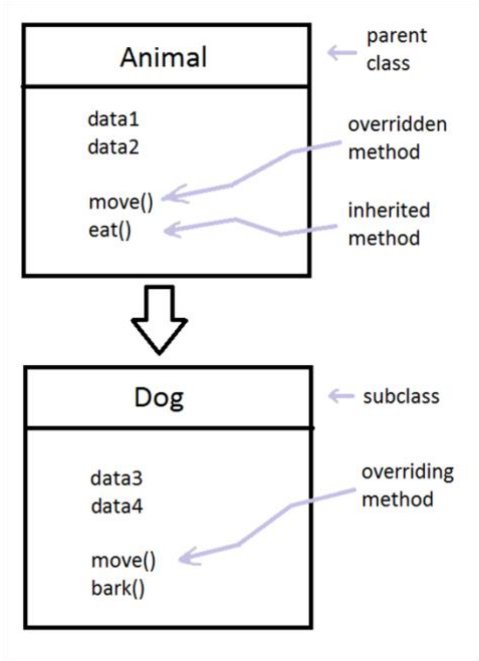
El dynamic dispatch es una forma de polimorfismo en que los métodos se resuelven en tiempo de ejecución, de manera que durante la ejecución se toma la decisión sobre que método es necesario utilizar.

Es increíblemente utilizado en lenguajes de programación orientados a objetos, debido a que cuando crea una instancia de un objeto de una clase derivada, el asignador de memoria asigna memoria para ese objeto en particular más su padre, y se devuelve una referencia a este fragmento de memoria. Por lo general, mantiene esta referencia en una variable.

Aunque no es tan rápido como el static dispatch ofrece una serie de ventajas importantes:

- No es necesario volver a compilar la parte del código que se basa en el código enlazado dinámicamente.
- Ofrece unos tiempos de compilación menores.
- Presenta la misma interfaz utilizando diferentes implementaciones.

¿Qué es el Method Overriding?



En la programación orientada a objetos, es una característica del lenguaje que permite que una subclase o clase secundaria proporcione una implementación específica de un método que ya proporciona una de sus superclases o clases principales.

La implementación en la subclase reemplaza la implementación en la superclase al proporcionar un método que tiene el mismo nombre, los mismos parámetros o firma y el mismo tipo de retorno que el método en la clase padre.

Usos en lenguajes de programación

Java

En Java todos los métodos de instancia son calculados de forma dinámica por defecto, esto implica que las clases en Java son mucho más flexibles ya que todo puede ser sobrescrito en las subclases. Por otra parte, Java nos permite utilizar static dispach haciendo uso de las palabras clave “final” o declarando un método como “private” o “static”.

Method overriding es una de las formas en que Java logra el Polimorfismo en tiempo de ejecución. La versión de un método que se ejecuta estará determinada por el objeto que se utiliza para invocarlo.

Si se usa un objeto de una clase primaria para invocar el método, se ejecutará la versión en la clase primaria, pero si se usa un objeto de la subclase para invocar el método, se ejecutará la versión en la clase secundaria. Como se puede observar en el siguiente ejemplo 1:

```
// method overriding in java

// Base Class
class Parent {
    void show()
    {
        System.out.println("Parent's show()");
    }
}
// Inherited class
class Child extends Parent {
    // This method overrides show() of Parent
    @Override
    void show(){
        System.out.println("Child's show()");
    }
}
// Driver class
class Main {
    public static void main(String[] args)
    {
        // If a Parent type reference refers
        // to a Parent object, then Parent's
        // show is called
        Parent obj1 = new Parent();
        obj1.show();

        // If a Parent type reference refers
        // to a Child object Child's show()
        // is called. This is called RUN TIME
        // POLYMORPHISM.
        Parent obj2 = new Child();
        obj2.show();
    }
```

Ejemplo 1

Output:

```
Parent's show()
Child's show()
```

C++

Una de las particularidades de C++ es que el method dispatching se realiza por defecto en tiempo de compilación, pero podemos solicitar que se haga de forma dinámica, para ello será necesario utilizar las palabras claves como “virtual” y “override”.

Esto es debido a que no siempre es necesario hacerlo de forma dinámica y en general supone una perdida de eficiencia en las llamadas a métodos que calculan de esta forma. La complejidad de las clases aumenta pues los métodos virtuales dependen de todas sus subclases haciendo el código más frágil.

C ++ no tiene la palabra clave super que una subclase puede usar en Java para invocar una versión de superclase de un método que desea anular. En su lugar, se utiliza el nombre de la clase principal o base seguido del operador de resolución de alcance. Como se observa en el siguiente ejemplo:

```cpp
#include <iostream>

//------------------------------------------------------------------
class Rectangle {
 public:
  Rectangle(double l, double w) : length_(l), width_(w) {}
  virtual void Print() const;

 private:
  double length_;
  double width_;
};

//------------------------------------------------------------------
void Rectangle::Print() const {
  // Print method of base class.
  std::cout << "Length = " << length_ << "; Width = " << width_;
}
```

Ejemplo 2: Código C++ - 1

```cpp
int main(int argc, char** argv) {
  Rectangle rectangle(5.0, 3.0);

  // Outputs: Length = 5.0; Width = 3.0
  rectangle.Print();

  Box box(6.0, 5.0, 4.0);

  box.Print();

  // This call illustrates overriding.
  // outputs: Length = 6.0; Width = 5.0; Height= 4.0
  static_cast<Rectangle&>(box).Print();
}
```

Ejemplo 3: Output

```cpp
class Box : public Rectangle {
 public:
  Box(double l, double w, double h) : Rectangle(l, w), height_(h) {}
  void Print() const override;

 private:
  double height_;
};

//------------------------------------------------------------------
// Print method of derived class.
void Box::Print() const {
  // Invoke parent Print method.
  Rectangle::Print();
  std::cout << "; Height = " << height_;
}
```

Ejemplo 4: Código C++ - 2

JavaScript

Debido a que en JavaScript no existe el static dispatch, todo se calcula en tiempo de ejecución.

La forma en la que el motor de JavaScript calcula que función (o propiedad) tiene que instanciar es la siguiente. Primero mira si el objeto tiene esa propiedad declarada, de manera que, si dispone de ella, simplemente se hace uso de esta. Si no la tiene el motor comprueba en el prototipo asociado al objeto si dispone de dicha propiedad declarada, de manera que si dispone de ella la utiliza y, si no la tiene vuelve a realizar la misma operación hasta que se queda sin más prototipos que comprobar.

En el siguiente ejemplo se observa un ejemplo de Method Overriding en JavaScript:

```
function Animal() { }

function Dog() { }

Animal.prototype.sleep = function () {
    alert("animal sleeping");
};

Animal.prototype.eat = function () {
    alert("animal eating");
};

Dog.prototype = new Animal;

Dog.prototype.eat = function () {
    alert("Dog eating");
};

var dog = new Dog;

dog.eat();
```

Ejemplo 5: JavaScript

Bibliografía

https://medium.com/ingeniouslysimple/static-and-dynamic-dispatch-324d3dc890a3

https://www.rightpoint.com/rplabs/switch-method-dispatch-table

https://en.wikipedia.org/wiki/Method_overriding

https://stackoverflow.com/questions/1805510/what-is-method-dispatch

https://en.wikipedia.org/wiki/Static_dispatch

https://en.wikipedia.org/wiki/Polymorphism_(computer_science)

https://en.wikipedia.org/wiki/Dynamic_dispatch

https://www.geeksforgeeks.org/overriding-in-java/

https://stackoverflow.com/questions/15497259/overriding-methods-in-javascript