

Clase 12: Implementación de clases II

Todo lo fundamental de la implementación de clases ya fue visto, ahora veremos los detalles, que son menos importantes

1 Abreviaturas

Java permite omitir this

```
void sumaDias(int d) {  
    this.dia+=d;  
    ...  
}
```

puede ser escrito

```
void sumaDias(int d) {  
    dia+=d;  
}
```

2 Accesibilidad de un método

Los métodos en java se pueden clasificar en 4 tipos de acuerdo a su accesibilidad

public estos métodos pueden ser accedidos desde cualquier código

private los métodos private, solo se pueden llamar desde código definido en la misma clase del método. Por ejemplo, todos los métodos auxiliares deberían ser declarados private

protected cuando se vea herencia...

friendly esta es la accesibilidad por omisión. Cuando no se coloca nada, java asume que el método solo puede ser accesado por el código del mismo paquete. Una forma simplificada de entender eso es que el código definido en el mismo directorio de la clase puede acceder a sus métodos

Las variables de R.I, por lo general, siempre se declaran como privadas, para evitar que el usuario acceda directamente al método.

3 Inicialización de un objeto

Un objeto suele ser inicializado de forma coherente mediante el constructor, sin embargo también se permite inicializarlo de inmediato

```
class Punto {  
    double x=0;  
    double y=0;  
    public Punto(double x, double y) {  
        this.x=x;  
        this.y=y;  
    }  
}
```

No es una buena práctica y en general no se recomienda su uso.

4 Métodos estáticos

Hay ocasiones, en las que uno puede querer que **todos** los objetos de una misma clase compartan una variable o método. Estos miembros deben ser definidos como static. En la clase Fecha, podríamos declarar un método que devuelva el “nombre” del día de la semana:

```
class Fecha {  
    ...  
    public static String nombreDia(int d) {  
        if (d==1)  
            return “Lunes”;  
        if (d==2)  
            return “Martes”;  
        ...  
    }  
}
```

Este método no necesita la información de un objeto en particular. De hecho no hay ningún this dentro de él. Este método puede ser accedido directamente poniendo el nombre de la clase en su invocación:

```
String dia=Fecha.nombreDia(7);
```

En general, los métodos que no acceden a la representación interna, se deben declarar estáticos. Los ejemplos abundan: todos los métodos de la clase Math son estáticos.

4.1 Variables estáticas

Como un método estático no pertenece a ningún objeto en particular, no tiene sentido poner `this` dentro de él. En particular, no tiene sentido acceder a una variable de instancia de la clase... a menos que esta también sea estática y no pertenezca a ningún objeto.

Si quisieramos saber cuantos objetos de la clase `Fecha` se han creado, podríamos hacer

```
class Fecha {
    static int cuantos=0;
    public Fecha(...) {
        Fecha.cuantos++;
    }
    public static cuantasFechas() {
        return Fechas.cuantos;
    }
}
```

La inicialización automática de variables estáticas se hace solo una vez.

4.2 El método main

Los métodos estáticos, como se puede deducir de lo ya dicho, no pueden acceder a métodos no estáticos dentro de la misma clase. En particular el método `main`, no puede acceder a ningún método que no sea estático.

Cuando uno escribe

```
java MiPrograma
```

para ejecutar un programa, lo que hace Java internamente, es invocar

```
MiPrograma.main(...);
```

Como `main` es estático, la forma de un programa tiene que ser una de las dos siguientes:

```
class MiPrograma {
    static metodo1(...) {...}
    static metodo2(...) {...}
    static main(...) {
        metodo1(...);
        metodo2(...);
    }
}
```

```
class MiPrograma {
    metodo1(...) {...}
    metodo2(...) {...}
    static main(...) {
        new MiPrograma(...);
    }
    MiPrograma(...) {
        metodo1(...);
        metodo2(...);
    }
}
```

La ventaja del primer enfoque es la simplicidad. La ventaja del 2do, es que se pueden (si así se desea), tener dos instancias de `MiPrograma` al mismo tiempo. Visto de otra forma, el primer enfoque obliga a que todo sea estático y por lo tanto a que no pueda haber dos instancias distintas de la misma clase. En la práctica, en los programas verdaderos, se usa el 2do enfoque casi siempre.