Tarea Independiente 04/08/2025

David Núñez Franco

August 6, 2025

Inventario de Conceptos Claves

- Rol de Arquitecto versus Ingeniero en el diagrama (realidad-persona-modelo-implementación)
- UML y rol en la historia del modelado
- En compiladores:
  - Syntax y semántica (estática) (de nuevo)
  - AST (Abstract Syntax Tree)
  - Tabla de símbolos (symbol table)
- En java
  - Semántica de package
  - Nombre simple
  - Nombre completo (calificado)
  - Mapeo de package a ruta en disco
  - Semántica de import
  - Semántica de import static
  - Semántica del * en el import
  - Semántica modificador de visibilidad (4 P's)
    - public
    - private
    - protected
    - default
- Principio Don't Repeat Yourself (DRY)
- var en JAVA

Punto 1

Verifique con un ejemplo que javac detecta una posible ambigüedad en un identificador no calificado al querer compilar un archivo Test.java.

Concepto

Antes de comenzar con la implementación, debemos definir el concepto de ambigüedad. En java, una ambigüedad en un identificador no calificado ocurre cuando una variable, clase o método puede referirse a más de una cosa, y el compilador no puede decidir cuál utilizar.

Solución

```java
package paquete1;

public class A {
    public static int value = 10;
}
```

Listing 1: A.java

```java
package paquete2;

public class B {
    public static int value = 10;
}
```

Listing 2: B.java

```java
import static paquete1.A.value;
import static paquete2.B.value;

public class Test {
    public static void main(String[] args) {
        System.out.println(value); // Aqui se genera la
            ambiguedad
    }
}
```

Listing 3: Test.java

Prueba y error

```
mkdir -p paquete1 paquete2

mv A.java paquete1/
mv B.java paquete2/
```

```
javac paquete1/A.java paquete2/B.java Test.java
```

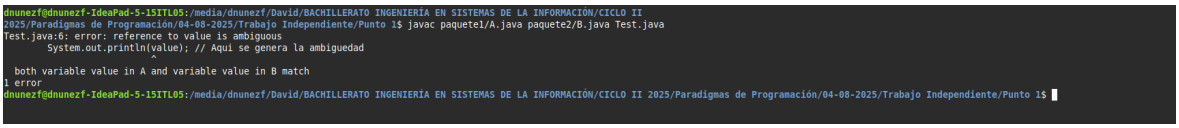

Figure 1: Mensaje error ambigüedad

Punto 2

¿Por qué javac no permite que dentro de un método que se declare un identificador público o privado?

```
1              class A{
2                  public int foo(){
3                      public int x = 666;
4                      return x;
5                  }
6              }
```

Solución

En Java, los modificadores de acceso (public, private, protected) se utilizan para controlar la visibilidad de clases, métodos y atributos a nivel de clase o instancia. Dentro de un método, solo se pueden declarar variables locales, y estas:

- Solo existen mientras se ejecuta el método.
- No forman parte de la clase ni del objeto.
- No pueden tener modificadores de acceso, porque no tiene sentido controlar su visibilidad fuera del método (nadie más puede acceder a ellas de todos modos).

Por eso, cuando tenemos

```
1          class A{
2              public int foo(){
3                  public int x = 666;
4                  return x;
5              }
6          }
```

javac lanza error, porque public no está permitido en la declaración de una variable local dentro del método foo.

Punto 3

Considere el código siguiente dentro de una clase Test.java. Note que hay unos ??? que indican un lugar donde Ud. debe escribir, según se explica. Solo puede cambiar esos ??? :

```
class A {
    private int x = 666;
    public class B {
        private int x = 999;
        public void foo() {
            System.out.println(???);
        }
    }
}
public class Test{
    public static void main(String... args){
        A a = new A();
        var b = ??? // Se necesita crear un b tal que
        b.foo();    // b.foo() imprima 666 no 999
    }
}
```

Solo se vale cambiar los ??? por algo correcto para que se logre imprimir el valor del x de A y no el del B

Solución

```
class A {
    private int x = 666;

    public class B {
        private int x = 999;

        public void foo() {
            System.out.println(A.this.x); // accedemos al x de A
        }
    }
}

public class Test {
    public static void main(String... args) {
        A a = new A();
        var b = a.new B(); // instanciamos la clase interna B con el objeto a
```

```
17                  b.foo();              // imprime 666
18              }
19          }
```

Ejecución y compilación

```
javac Test.java
java Test
```

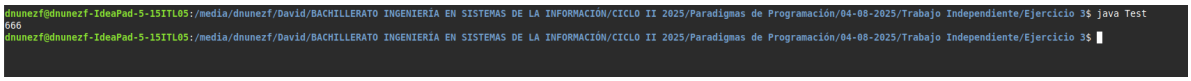

Figure 2: Salida