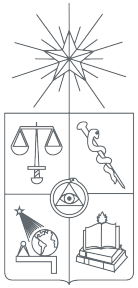

dcc
CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Auxiliar 2: Visibilidad, Accesibilidad, Interfaces y Clase Abstracta

Tomas Vallejos
tomas.vallejos@ing.uchile.cl

Agenda

- Repaso
- Visibilidad
- Accesibilidad
- Interfaces y Clases Abstractas
- Liskov

Repaso: this y super

- ¿ Qué son this y super ?

Repaso: this y super

- ¿ Qué son this y super ?
- ¿ Qué es una pseudo-variable ?

Repaso: this y super

- ¿ Qué son this y super ?
- ¿ Qué es una pseudo-variable ?
- ¿ Como hacen el method lookup ?

Repaso: Interfaces y Clase Abstracta

- ¿ Qué son las interfaces ?

Repaso: Interfaces y Clase Abstracta

- ¿ Qué son las interfaces ?
- ¿ Para qué se usan las interfaces ?

Repaso: Interfaces y Clase Abstracta

- ¿ Qué son las interfaces ?
- ¿ Para qué se usan las interfaces ?
- ¿ Qué es una Clase Abstracta ?

Repaso: Interfaces y Clase Abstracta

- ¿ Qué son las interfaces ?
- ¿ Para qué se usan las interfaces ?
- ¿ Qué es una Clase Abstracta ?
- ¿ Para qué se usan las Clases Abstractas ?

Repaso: Interfaces y Clase Abstracta

- ¿ Qué son las interfaces ?
- ¿ Para qué se usan las interfaces ?
- ¿ Qué es una Clase Abstracta ?
- ¿ Para qué se usan las Clases Abstractas ?
- ¿ Cuándo uso una o la otra ?

Repaso: Liskov

- ¿ Quién es Liskov ?

Repaso: Liskov

- ¿ Quién es Liskov ?
- ¿ Qué es el principio de liskov ?

Repaso: Liskov

- ¿ Quién es Liskov ?
- ¿ Qué es el principio de liskov ?
- ¿ Cuándo se dice que estamos en presencia de una clase frágil ?

Visibilidad

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Visibilidad 1

```
public class A {
    private String method1() {return "A.method1()";}

    public String method2() {return "A.method2() > " + this.method1();}
}

public class B extends A {
    public String method1() {return "B.method1()";}

    public static void main(String[] args) {
        System.out.println(new B().method2());
    }
}
```

Visibilidad 2

```
public class C {
    protected String method1() {return "C.method1()";}

    public String method2() {return "C.method2() > " + this.method1();}
}

public class D extends C {
    public String method1() {return "D.method1()";}

    public static void main(String[] args) {
        System.out.println(new D().method2());
    }
}
```

Visibilidad 3

```
public class A {
	private String method1() {
	return "A.method1()";}

	public String method2() {
	return "A.method2() > " + this.method1();}
}
public class B extends A {
	public String method1() {
		return "B.method1()";
	}
}
public class C {
	protected String method1() {
		return "C.method1()";
	}
	public String method2() {
		return "C.method2() > " + this.method1();
	}
}
public class D extends C {
	public String method1() {
		return "D.method1()";
	}
}
```

```
public class E {
	public String method1() {
		return "E.method1()";
	}
	public String method2() {
		return "E.method2() > "+this.method1();
	}
}
public class F extends E {
	public String method1() {
		return "F.method1()";
	}
}

public class G {

	public static void main(String[] args) {
		System.out.println(new A().method2());
		System.out.println(new B().method2());
		System.out.println(new C().method2());
		System.out.println(new D().method2());
		System.out.println(new E().method2());
		System.out.println(new F().method2());
	}
}
```

Visibilidad 3

```
public class A {
	private String method1() {
	return "A.method1()";}

	public String method2() {
	return "A.method2() > " + this.method1();}
}
public class B extends A {
	public String method1() {
		return "B.method1()";
	}
}
public class C {
	protected String method1() {
		return "C.method1()";
	}
	public String method2() {
		return "C.method2() > " + this.method1();
	}
}
public class D extends C {
	public String method1() {
		return "D.method1()";
	}
}
```

```
public class E {
	public String method1() {
		return "E.method1()";
	}
	public String method2() {
		return "E.method2() > "+this.method1();
	}
}
public class F extends E {
	public String method1() {
		return "F.method1()";
	}
}

public class G {

	public static void main(String[] args) {
		System.out.println(new A().method2());
		System.out.println(new B().method2());
		System.out.println(new C().method2());
		System.out.println(new D().method2());
		System.out.println(new E().method2());
		System.out.println(new F().method2());
	}
}
```

Accesibilidad

```
public class Animal {
     private String name;
     public Animal(String name) {
          this.name = name;
     }

     private String getName() {
          return name;
     }

     public String getPair(Animal paired) {
          return this.getName() + " with " + paired.getName();
     }

     public static void main(String[] args) {
          System.out.println(new Animal("Jirafa").getPair(new Animal("Antilope")));
          System.out.println(new Animal("Tigre").getName());
     }
}
```

Overloading

```
public class A {
     String m(A o1, B o2) {
          return "A.m(A,B)";
     }
}
```

```
public class B extends A {
     String m(A o1, A o2) {
          return "B.m(A,A)";
     }
public static void main(String[] args) {
     System.out.println("1. " + new B().m(new A(), new A()));
     System.out.println("2. " + new B().m(new A(), new B()));
     A object1 = new B();
     A object2 = new B();
     System.out.println("3. " + new B().m(object1, object2));
     System.out.println("4. " + new B().m((B) object1, object2));
     System.out.println("5. " + new B().m((B) object1, (B) object2));
     }
}
```

Interfaces y Clases Abstractas

Su amigo le dice que quiere abrir un bar, tiene un bartender que prepara Piscolitas, Tom Collins y Vino, el bartender necesita varios tipos de vasos y todos los ingredientes necesarios. Le pide a usted que diseñe un bar orientado a objetos. ¿ Cómo lo harías ?

Interfaces y Clases Abstractas

Su amigo le dice que quiere abrir un bar, tiene un bartender que prepara Piscolitas, Tom Collins y Vino, el bartender necesita varios tipos de vasos y todos los ingredientes necesarios. Le pide a usted que diseñe un bar orientado a objetos. ¿ Cómo lo harías ?