Working with inheritance

Herencia con clases

Para que una clase herede de una clase se utiliza la palabra `extends`

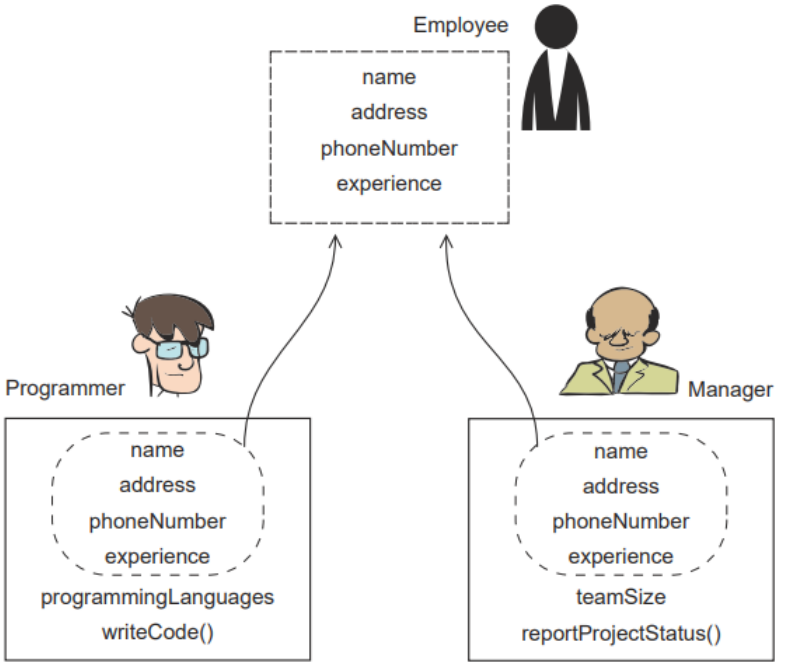


Figure 6.2 Identify common properties and behaviors of a Programmer and a Manager, pull them out into a new position, and name it Employee.

Todas las clases heredan de `java.lang.Object` .

Beneficios de la herencia:

- Subclases con una definición más corta
- Facilidad de modificación por economía de código
- Reutilización de código

Una clase derivada contiene dentro un objeto de la clase base

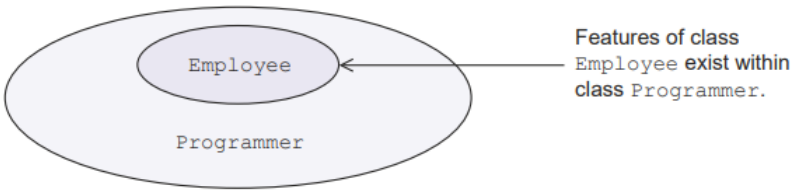


Figure 6.6 An object of a derived class can access features of its base class object.

Una clase derivada puede heredar lo que ve, es decir, no puede heredar miembros privados.

Miembros no heredados

- Miembros privados
- Miembro con acceso `default` en caso de que la clase base y la derivada estén en paquetes separados
- Constructores de la clase base. Se pueden llamar, pero no se heredan.

Definición de miembros propios

Las clase derivadas pueden definir sus propios miembros, constructores, métodos y variables. Esto puede provocar el `hide` y `override`

Cuando una clase derivada define un atributo con el mismo nombre que una definida en su clase base, únicamente estas nuevas variables y métodos son visibles al utilizar la clase derivada.

Abstract base clase vs concrete base class

Una clase abstracta no puede ser instanciada y obliga a sus clases derivadas a definir la implementación de sus comportamientos abstractos.

Si una clase define un método abstracto, la clase debe ser abstracta

Una clase abstracta no está obligada a definir métodos abstractos

Si una clase derivada no define los métodos abstractos de la clase base, entonces debe ser marcada como abstracta también.

Herencia de interfaces

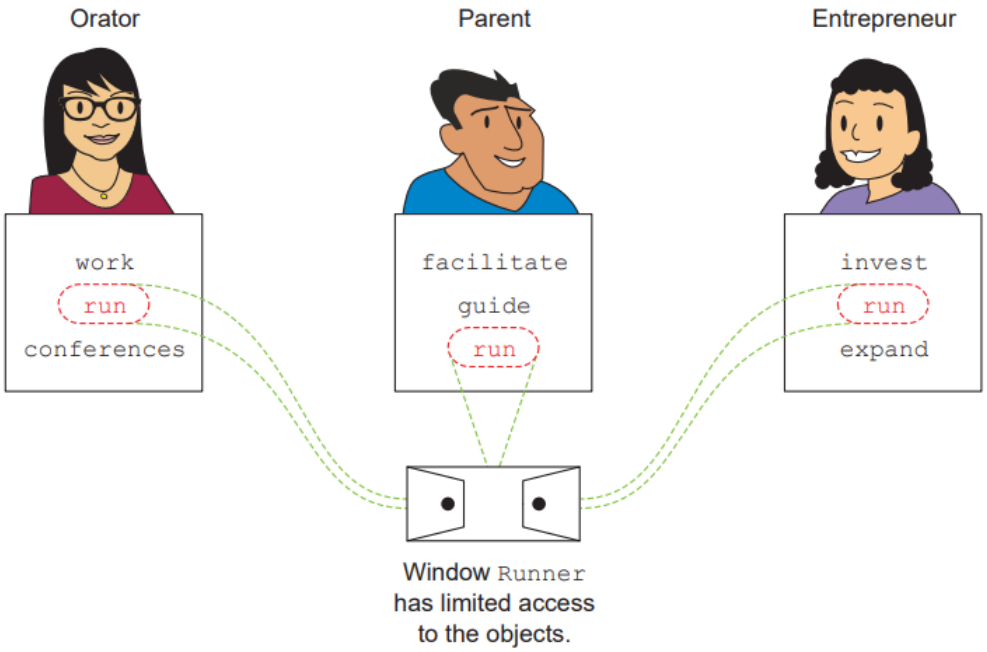


Figure 6.7 You can compare an interface with a window that can connect multiple objects but has limited access to them.

Una interfaz puede verse como un contrato que incluye un conjunto de reglas.

Los métodos definidos en la interfaz no incluyen implementación → son abstractos

En java 8 también se pueden definir `static` y `default` métodos en una interfaz. Este cambio se realizó principalmente para el manejo de Streams.

Java no permite herencia múltiple de clase, pero si de interfaces utilizando `implements`

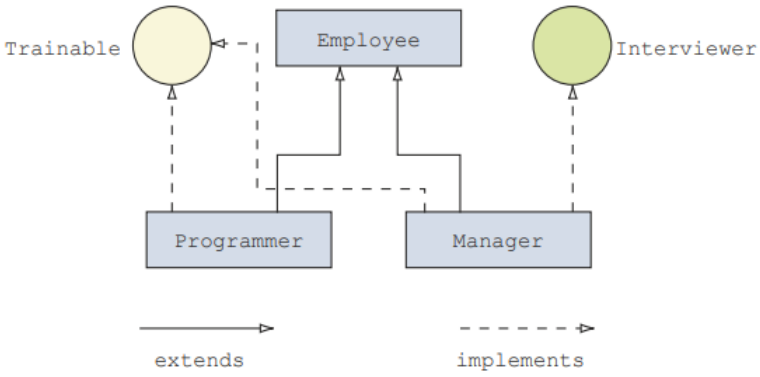


Figure 6.9 Relationships among the classes `Employee`, `Programmer`, and `Manager` and the interfaces `Trainable` and `Interviewer`, with interfaces represented by circles

Definiendo interfaces

- Todos los métodos definidos en una interfaz son públicos y abstractos por defecto

- Las variables definidas en una interfaz son publicas, estáticas y final.
- La interfaz debe ser pública o `default` → De lo contrario no compila.

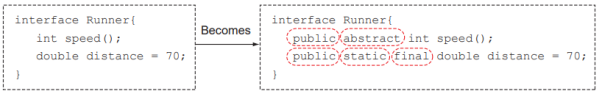


Figure 6.10 All the methods of an interface are implicitly public. Its variables are implicitly public, static, and final.

Tipos de métodos en una interfaz

A partir de Java 8 es posible definir métodos `default` y `static` .

Métodos default

Son métodos con implementación y utilizan la palabra `default` .

No es necesario sobreescribir estos métodos, y en caso de que se invoquen sin sobreescribir, se llama al método default.

Cuando se hace el override o se usa la palabra `default`

Métodos static

Método con implementación y que son invocados a través del nombre de la interfaz.

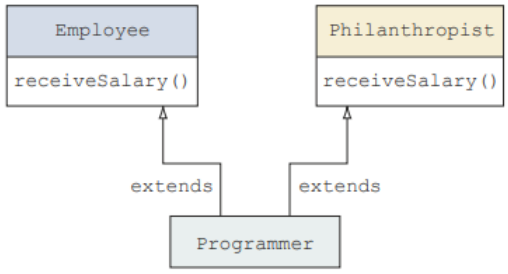
No se pueden llamar desde una variable de referencia.

Reglas para sobreescribir métodos

- Sucede a través de herencia
- El nombre del método debe ser el mismo
- La lista de argumentos debe ser la misma
- El tipo de retorno en la subclase puede ser el mismo, o una subclase del retorno. → Covariant return type.
- El método en la clase base puede o no ser abstracto.
- Solo se pueden sobreescribir métodos no finales.
- Los modificadores de acceso pueden ser el mismo o menos restrictivos.
- Un método estático en una clase no oculta o sobreescribe un método estático en una interfaz.

Una clase no puede heredar múltiples clases

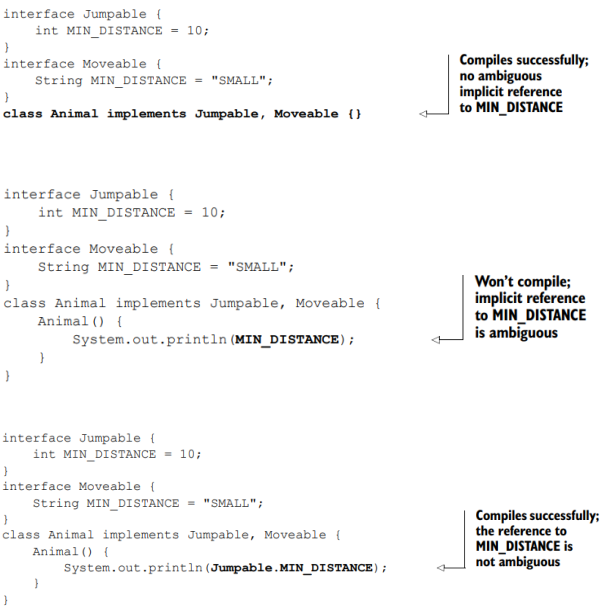
In Java, a class can’t extend multiple classes.



Una clase puede implementar múltiples interfaces

Implementando múltiples interfaces con constantes de mismo nombre

Una clase puede implementar múltiples interfaces con el mismo nombre de constante solo si la referencia a las constantes no son ambiguas.




```
interface Jumpable {
    int MIN_DISTANCE = 10;
}
interface Moveable {
    String MAX_DISTANCE = "SMALL";
}
class Animal implements Jumpable, Moveable {
    Animal() {
        System.out.println(MIN_DISTANCE);
    }
}
```

Compiles successfully; implicit reference to MIN_DISTANCE is not ambiguous

Implementando interfaces múltiples con el métodos abstractos con el mismo nombre

Estos métodos no definen su cuerpo, por lo que es aceptable, ya que se considera que la clase implementa ambos métodos

```
interface Jumpable {
    abstract String currentPosition();
}
interface Moveable {
    abstract String currentPosition();
}
class Animal implements Jumpable, Moveable {
    public String currentPosition() {
        return "Home";
    }
}
```

Pero no se puede si no se cumplen las reglas de la sobreescritura, por ejemplo, cambiar el tipo de retorno.

```
interface Jumpable {
    abstract String currentPosition();
}
interface Moveable {
    abstract void currentPosition();
}
class Animal implements Jumpable, Moveable {
    public String currentPosition() {
        return "Home";
    }
}
```

Won't compile

Implementando múltiples interfaces con métodos default con el mismo nombre

Una clase puede implementar interfaces con los mismos métodos default si se cumplen las reglas de sobreescritura y se sobreescribe la implementación default.

```
interface Jumpable {
    default void relax() {
        System.out.println("No jumping");
    }
}
interface Moveable {
    default void relax() {
        System.out.println("No moving");
    }
}
class Animal implements Jumpable, Moveable { }
```

Won't compile; inherits unrelated defaults for relax() from Jumpable and Moveable

```
class Animal implements Jumpable, Moveable {
    public void relax() {
        System.out.println("Watch movie");
    }
}
```

Compiles successfully

Implementando interfaces múltiples con métodos estáticos con el mismo nombre

Estos no son heredados a la clase, pertenecen a la interfaz, entonces no hay problema alguno.

```
interface Jumpable {
    static int maxDistance() {
        return 100;
    }
}
interface Moveable {
    static String maxDistance() {
        return "forest";
    }
}
class Animal implements Jumpable, Moveable { }
```

Compiles successfully

Extendiendo interfaces

Una interface puede extender múltiples interfaces

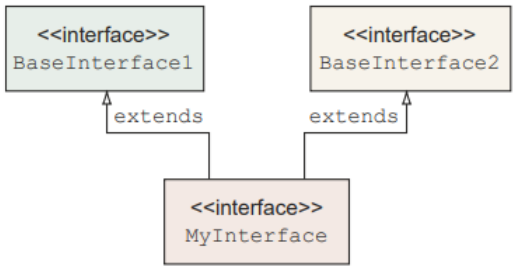


Figure 6.13 The interface `MyInterface` extends the interfaces `BaseInterface1` and `BaseInterface2`.

Extendiendo múltiples interfaces con métodos abstractos con el mismo nombre

```
interface BaseInterface1 {
    String getName();
}
interface BaseInterface2 {
    String getName();
}
interface MyInterface extends BaseInterface1, BaseInterface2 {}
```

```
class Employee implements MyInterface {
    String name;
    public String getName() {
        return name;
    }
}
```

Employee defines a body for the method getName, inherited from the interface MyInterface

Extendiendo múltiples interfaces con métodos default con el mismo nombre

```
interface BaseInterface1 {
    default void getName() {
        System.out.println("Base 1");
    }
}
interface BaseInterface2 {
    default void getName() {
        System.out.println("Base 2");
    }
}
interface MyInterface extends BaseInterface1, BaseInterface2 {}
```

Won't compile

```
interface BaseInterface1 {
    default void getName() { System.out.println("Base 1"); }
}
interface BaseInterface2 {
    default void getName() { System.out.println("Base 2"); }
}
interface MyInterface extends BaseInterface1, BaseInterface2 {
    default void getName() { System.out.println("Just me"); }
}
```

Compiles successfully

Extendiendo múltiples interfaces con métodos static con el mismo nombre

```
interface BaseInterface1 {
    static void status() {
        System.out.println("Base 1");
    }
}
interface BaseInterface2 {
    static String status() {
        System.out.println("Base 2");
        return null;
    }
}
interface MyInterface extends BaseInterface1, BaseInterface2 {
}
```

Compiles successfully

No existen problemas, los métodos estáticos no se heredan

Propiedades de los miembro de una interfaz

Constantes

Las variables en una interfaz son implícitamente `public` `final` `static`

Las variables deben estar inicializadas o no compilará

```
interface MyInterface {
    public static final int AGE = 10;
}
```

public, static, and final modifiers are implicitly added to variables defined in an interface

Métodos

Implícitamente son `public` y `abstract`

Constructores

Una interfaz no puede definir constructores.

Referencias a miembros

```
class Employee {
    String name;
    String address;
    String phoneNumber;
    float experience;
}
```

```
interface Interviewer {
    public void conductInterview();
}
class HRExecutive extends Employee implements Interviewer {
    String[] specialization;
    public void conductInterview() {
        System.out.println("HRExecutive - conducting interview");
    }
}
```

Class HRExecutive inherits class Employee and implements interface Interview

Usando una variable de la subclase para acceder a su objeto B b

Cuando se realiza esto, se puede tener acceso a los miembros definidos propios, en la base e interfaz.

```
class Office {
    public static void main(String args[]) {
        HRExecutive hr = new HRExecutive();
    }
}
```

A variable of type HRExecutive can be used to refer to its object

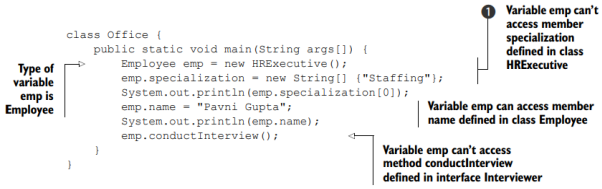
```
class Office {
    public static void main(String args[]) {
        HRExecutive hr = new HRExecutive();
        hr.specialization = new String[] {"Staffing"};
        System.out.println(hr.specialization[0]);
        hr.name = "Pavni Gupta";
        System.out.println(hr.name);
        hr.conductInterview();
    }
}
```

Access variable defined in class HRExecutive

Access variable defined in class Employee

Access method defined in interface Interviewer

Usando una variable de una superclase para acceder al objeto de la clase derivada. A b



- No puede acceder a los atributos de la subclase
- Accede a los atributos definidos en su propia clase
- No puede acceder al método de la interfaz, no conoce la implementación

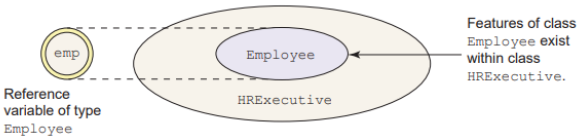
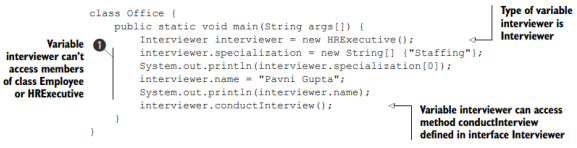
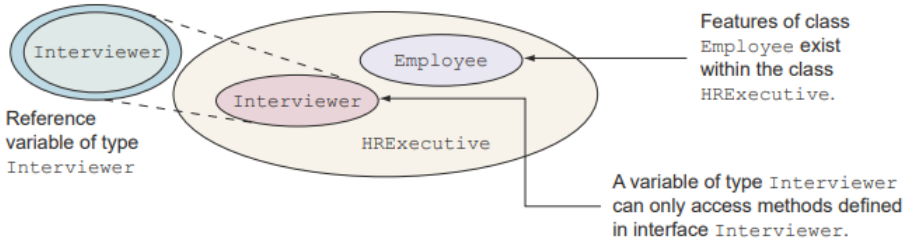


Figure 6.18 A variable of type `Employee` can see only the members defined in the class `Employee`.

Usando una variable de una interfaz implementada para acceder a una subclase I b



- No se puede acceder a miembros ni de la superclase ni de la subclase → Error de compilación
- Se puede acceder al método que tiene la interfaz.



Casting

El casting es el proceso se hacer que una variable se comporte como una variable de otro tipo.

```
Interviewer interviewer = new HRExecutive();
```

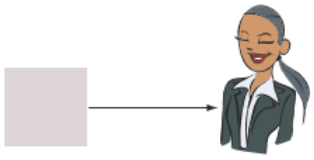


Figure 6.21 A reference variable of the interface `Interviewer` referring to an object of the class `HRExecutive`

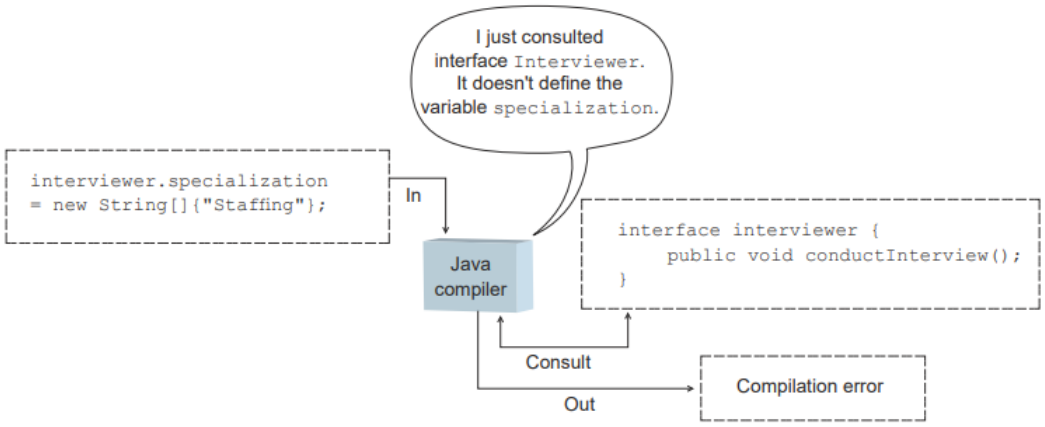


Figure 6.22 The Java compiler doesn't compile code if you try to access the variable `specialization`, defined in the class `HRExecutive`, by using a variable of the interface `Interviewer`.

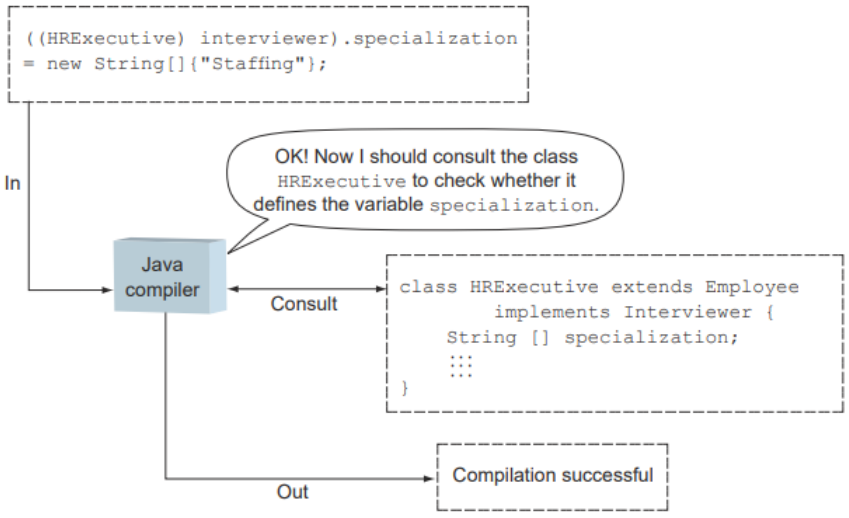
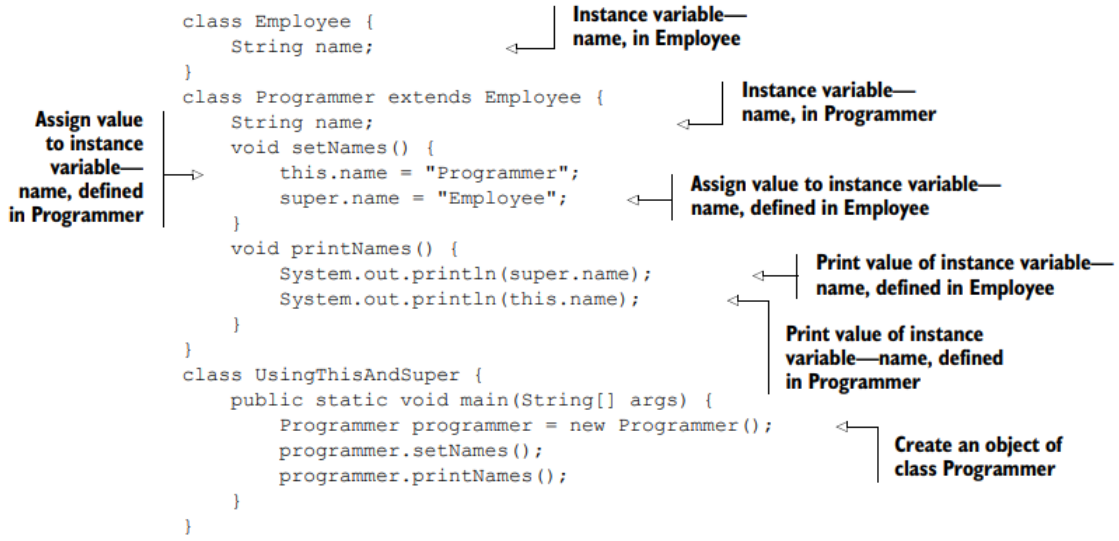


Figure 6.23 Casting can be used to access the variable `specialization`, defined in the class `HRExecutive`, by using a variable of the interface `Interviewer`.

Usando `this` y `super` para acceder a miembros

Se puede usar `this` para referirse a lo miembros accesible de la clase.

Se usa `super` para acceder a miembros de la clase base.



No se puede utilizar `this` en métodos `static`

Polimorfismo

En tiempo de ejecución se define que comportamiento tienen los objetos.

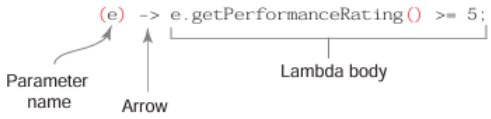
Métodos polimórficos → Métodos sobreescritos

El polimorfismo trabaja únicamente con métodos sobreescritos.

Remember that variables bind at compile time,
whereas methods bind at runtime.

Los métodos sobrecargados no participan en el polimorfismo.

Expresiones lambda



- Tipo de parámetro - Opcional
- Nombre del parámetro
- Flecha
- Llaves - Opcional
- `return` - Opcional y obligatorio si se usan llaves
- Cuerpo