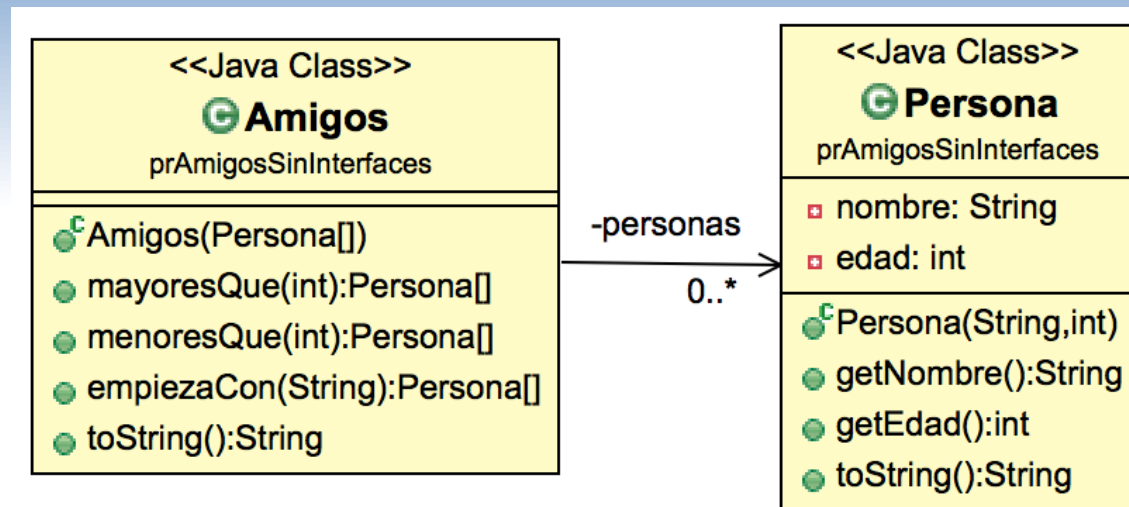


Interfaces como abstracción funcional

- Los interfaces proporcionan soluciones elegantes a problemas donde se desea aplicar diferentes funcionalidades a unos mismos datos (**abstracción funcional**).
- Para comprenderlo mejor, veamos un ejemplo solucionado sin interfaces y después con interfaces aplicando abstracción funcional.
 - Disponemos de dos clases
 - Clase **Persona** con información de una persona (nombre y edad)
 - Clase **Amigos** con información de muchas personas (un array de personas)
 - Queremos operar con la clase **Amigos** para almacenar los amigos de una persona y poder seleccionar subgrupos que cumplan algún criterio:
 - Los que son menores de una edad dada.
 - Los que son mayores de una edad dada.
 - Aquellos cuyo nombre empieza por una cadena dada.
 - ...

Interfaces como abstracción funcional



- Lo natural es crear un método en la clase **Amigos** que resuelva cada una de las funcionalidades requeridas
 - Para las personas menores de una edad dada
`Persona [] menoresQue(int n)`
 - Para las personas mayores de una edad dada.
`Persona [] mayoresQue(int n)`
 - Para las personas que empiezan por una cadena dada.
`Persona [] empiezaCon(String s)`
- El cuerpo de estos métodos **será muy parecido**, solo cambiarán la manera de **seleccionar** las personas.

Interfaces como abstracción funcional

```
public class Persona {  
    private String nombre;  
    private int edad;  
  
    public Persona(String n, int e) {  
        nombre = n;  
        edad = e;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public int getEdad() {  
        return edad;  
    }  
  
    @Override  
    public String toString() {  
        return "(" + nombre + "; " + edad + ")";  
    }  
}
```

Interfaces como abstracción funcional

```
public class Amigos {  
    private Persona[] personas;  
  
    public Amigos(Persona[] personas) {  
        this.personas = personas;  
    }  
    public Persona[] mayoresQue(int edad) {  
        Persona[] nueva = new Persona[personas.length];  
        int i = 0;  
        for (Persona p : personas) {  
            if (p.getEdad() >= edad) {  
                nueva[i] = p;  
                i++;  
            }  
        }  
        return Arrays.copyOf(nueva, i);  
    }  
    public Persona[] menoresQue(int edad) {  
        //...  
    }  
    public Persona[] empiezaCon(String s) {  
        //...  
    }  
    public String toString() {  
        return Arrays.toString(personas);  
    }  
}
```

Este método filtra el array de personas y devuelve un array con todas las que son mayores que una edad dada.

Los métodos `menoresQue(...)` y `empiezaCon(...)` tendrán el mismo código, a excepción de la condición del `if`, que dependerá del criterio de selección de personas que se quiera implementar.

Interfaces como abstracción funcional

```
public class Amigos {  
    private Persona[] personas;  
  
    public Amigos(Persona[] personas) {  
        this.personas = personas;  
    }  
    public Persona[] mayoresQue(int edad) {  
        //...  
    }  
    public Persona[] menoresQue(int edad) {  
        Persona[] nueva = new Persona[personas.length];  
        int i = 0;  
        for (Persona p : personas) {  
            if (p.getEdad() <= edad) {  
                nueva[i] = p;  
                i++;  
            }  
        }  
        return Arrays.copyOf(nueva, i);  
    }  
    public Persona[] empiezaCon(String s) {  
        //...  
    }  
    public String toString() {  
        return Arrays.toString(personas);  
    }  
}
```

Se crea un array de la máxima longitud posible, en previsión de que todas las personas del array cumpliesen la condición

Este método es igual que el anterior, sólo cambiamos \geq por \leq

Antes de salir ya sabemos que i personas han cumplido la condición, así que se ajusta el tamaño del array de salida al espacio que realmente se necesita.

Interfaces como abstracción funcional

```
public class Amigos {  
    private Persona[] personas;  
  
    public Amigos(Persona[] personas) {  
        this.personas = personas;  
    }  
    public Persona[] mayoresQue(int edad) {  
        //...  
    }  
    public Persona[] menoresQue(int edad) {  
        //...  
    }  
    public Persona[] empiezaCon(String s) {  
        Persona [] nueva = new Persona[personas.length];  
        int i = 0;  
        for(Persona p : personas) {  
            if(p.getNombre().startsWith(s)) {  
                nueva[i] = p;  
                i++;  
            }  
        }  
        return Arrays.copyOf(nueva, i);  
    }  
    public String toString() {  
        return Arrays.toString(personas);  
    }  
}
```

Cambiamos la condición de selección.

startsWith(...) es un método de la clase String que devuelve true si la cadena que recibe el mensaje empieza por la que se le pasa como parámetro

Interfaces como abstracción funcional

```
public class MainAmigos {  
    public static void main(String[] args) {  
        Persona[] personas = { new Persona("juan", 25), new Persona("maria", 32),  
                                new Persona("marta", 28), new Persona("julio", 33),  
                                new Persona("manuel", 29), new Persona("justino", 25) };
```

Clase distinguida de prueba

```
        Amigos amigos = new Amigos(personas);
```

Se crea un objeto de la clase amigos.

Extraemos tres grupos de personas:

1. Aquellos cuyos nombres empiezan por "ma"
2. Los mayores de 28 años.
3. Los menores de 27 años.


```
        Persona[] grupo1 = amigos.empiezaCon("ma");  
        System.out.println("Empiezan con \"ma\": " + Arrays.toString(grupo1));
```

```
        Persona[] grupo2 = amigos.mayoresQue(28);  
        System.out.println("Mayores de 28: " + Arrays.toString(grupo2));
```

```
        Persona[] grupo3 = amigos.menoresQue(27);  
        System.out.println("Menores de 27: " + Arrays.toString(grupo3));
```

```
    }
```

```
}
```



```
<terminated> MainAmigos [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_31.jdk/Conte  
Empiezan con "ma": [(maria; 32), (marta; 28), (manuel; 29)]  
Mayores de 28: [(maria; 32), (marta; 28), (julio; 33), (manuel; 29)]  
Menores de 27: [(juan; 25), (justino; 25)]
```

Interfaces como abstracción funcional

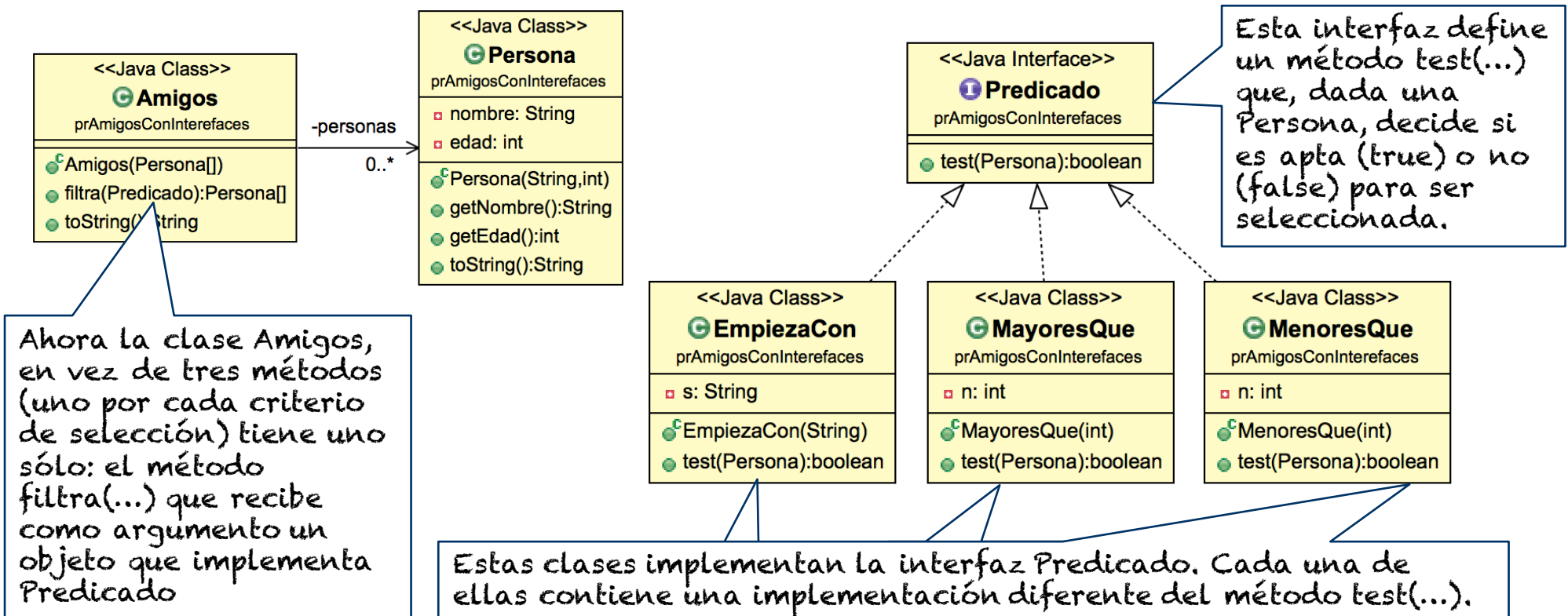
- ¿Y si queremos añadir una nueva funcionalidad?
 - Por ejemplo: ahora queremos filtrar las personas que contienen un determinado texto en su nombre.
 - Habrá que modificar la clase **Amigos**, añadir un nuevo método y volver a compilar la clase:

```
public class Amigos {  
    private Persona[] personas;  
    public Amigos(Persona[] personas) { //... }  
    public Persona[] mayoresQue(int edad) { //... }  
    public Persona[] menoresQue(int edad) { //... }  
    public Persona[] empiezaCon(String s) { //... }  
  
    public Persona [] contiene(String s) {  
        Persona [] nueva = new Persona[personas.length];  
        int i = 0;  
        for(Persona p : personas) {  
            if(p.getNombre().contains(s)) {  
                nueva[i] =p;  
                i++;  
            }  
        }  
        return Arrays.copyOf(nueva, i);  
    }  
    public String toString() { //... }  
}
```

contains(...) es un método de la clase String que devuelve true si la cadena que recibe el mensaje contiene a la que se le pasa como parámetro

Interfaces como abstracción funcional

- A continuación vamos a plantear una solución a este problema usando interfaces (abstracción funcional).
 - Se crea una interfaz que defina un método que decide qué personas seleccionar.
 - Cada criterio de selección se define en una clase que implementa la interfaz.
 - La clase contenedora dispondrá de un método que toma como argumento un objeto que implementa la interfaz y realiza la selección.



Interfaces como abstracción funcional

```
public class Amigos {  
    private Persona[] personas;  
  
    public Amigos(Persona[] personas) {  
        this.personas = personas;  
    }  
    public Persona [] filtra(Predicado pred) {  
        Persona [] nueva = new Persona[personas.length];  
        int i = 0;  
        for(Persona p : personas) {  
            if(pred.test(p)) {  
                nueva[i] = p;  
                i++;  
            }  
        }  
        return Arrays.copyOf(nueva, i);  
    }  
  
    public String toString() {  
        return Arrays.toString(personas);  
    }  
}
```

El método filtra presenta dos diferencias respecto a los métodos (menorQue(...), etc.) del ejemplo anterior:

1. Tiene un argumento que no es ni la edad ni una cadena sino un objeto que implementa la interfaz Predicado.

2. La condición del if cambia, ahora se llama al método test de Predicado.

Interfaces como abstracción funcional

```
public interface Predicado {  
    public boolean test (Persona p);  
}
```

```
public class MayoresQue implements Predicado {  
    private int n;  
    public MayoresQue(int n) {  
        this.n = n;  
    }  
    public boolean test(Persona p) {  
        return p.getEdad() >= n;  
    }  
}
```

```
public class EmpiezaCon implements Predicado {  
    private String s;  
    public EmpiezaCon(String s) {  
        this.s = s;  
    }  
    public boolean test(Persona p) {  
        return p.getNombre().startsWith(s);  
    }  
}
```

```
public class MenoresQue implements Predicado {  
    private int n;  
    public MenoresQue(int n) {  
        this.n = n;  
    }  
    public boolean test(Persona p) {  
        return p.getEdad() <= n;  
    }  
}
```

Cada una de las clases tiene como atributo el dato que necesitan para determinar si una persona es apta o no: la edad en el caso de MayoresQue y MenoresQue y una cadena de texto en el caso de EmpiezaCon

El método test:

1. Recibe una persona.
2. Comprueba si cumple o no una condición

Interfaces como abstracción funcional

```
public class MainAmigos {  
    public static void main(String[] args) {  
        Persona[] personas = {  
            new Persona("juan", 25), new Persona("maria", 32),  
            new Persona("marta", 28), new Persona("julio", 33),  
            new Persona("manuel", 29), new Persona("justino", 25) };
```

Clase distinguida de prueba

```
        Amigos amigos = new Amigos(personas);
```

Se crea un objeto de la clase amigos.

Extraemos tres grupos de personas:
1. Aquellos cuyos nombres empiezan por "ma"
2. Los mayores de 28 años.
3. Los menores de 27 años.

```
        Persona[] grupo1 = amigos.filtra( new EmpiezaCon("ma") );  
        System.out.println("Empiezan con \"ma\": " + Arrays.toString(grupo1));
```

```
        Persona[] grupo2 = amigos.filtra( new MayoresQue(28) );  
        System.out.println("Mayores de 28: " + Arrays.toString(grupo2));
```

```
        Persona[] grupo3 = amigos.filtra( new MenoresQue(27) );  
        System.out.println("Menores de 27: " + Arrays.toString(grupo3));
```

```
    }
```

```
}
```

Console Progress

<terminated> MainAmigos [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_31.jdk/Conte
Empiezan con "ma": [(maria; 32), (marta; 28), (manuel; 29)]
Mayores de 28: [(maria; 32), (marta; 28), (julio; 33), (manuel; 29)]
Menores de 27: [(juan; 25), (justino; 25)]

Interfaces como abstracción funcional

```
public class MainAmigos {  
    public static void main(String[] args) {  
        Persona[] personas = {  
            new Persona("juan", 25), new Persona("maria", 32),  
            new Persona("marta", 28), new Persona("julio", 33),  
            new Persona("manuel", 29), new Persona("justino", 25) };  
    }
```

```
    Amigos amigos = new Amigos(personas);
```

La diferencia respecto al ejemplo anterior, es que ahora siempre se llama al método filtra(...) pero pasándole, según nos convenga, una implementación diferente de Predicado

```
        Persona[] grupo1 = amigos.filtra( new EmpiezaCon("ma") );  
        System.out.println("Empiezan con \"ma\": " + Arrays.toString(grupo1));
```

```
        Persona[] grupo2 = amigos.filtra( new MayoresQue(28) );  
        System.out.println("Mayores de 28: " + Arrays.toString(grupo2));
```

```
        Persona[] grupo3 = amigos.filtra( new MenoresQue(27) );  
        System.out.println("Menores de 27: " + Arrays.toString(grupo3));
```

```
    }
```

```
}
```

Interfaces como abstracción funcional

- ¿Y si queremos añadir una nueva funcionalidad?
 - Hay que añadir otra clase, **Contiene**, que implemente **Predicado**
 - No hay que tocar el código de la clase **Amigos** ni volver a compilarla.

