Introducción a Java

//Parte 3

IT BOARDING

BOOTCAMP

Índice

IT BOARDING BOOTCAMP

TEMA

// Una interface, múltiples implementaciones

IT BOARDING

BOOTCAMP

Si pudiésemos delegar una tarea sin preocuparnos por cómo funciona su implementación, eso es abstracción.

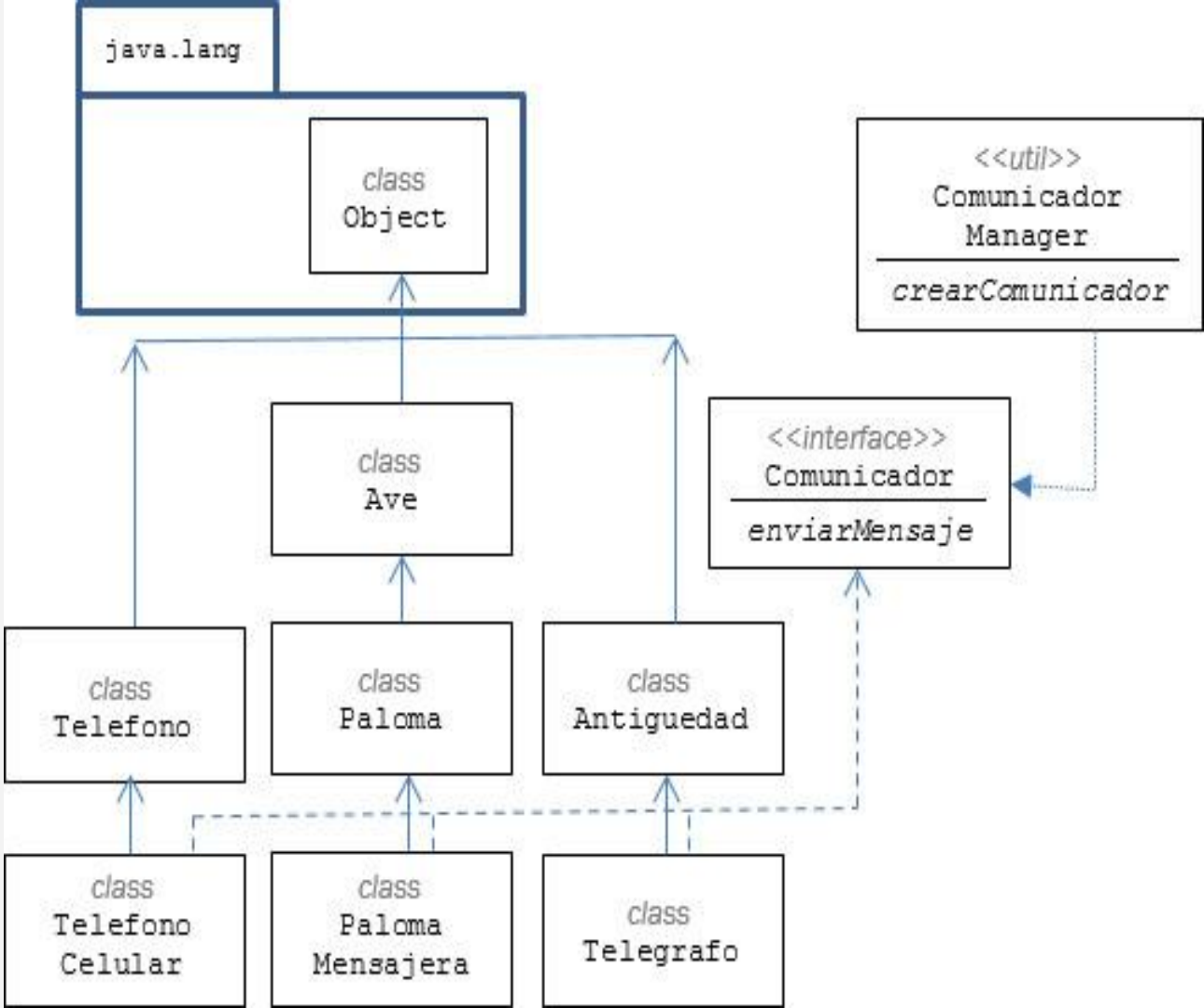
Interfaces 1

Una interface es una “cáscara”, una especie de skin que permite ocultar qué hay por debajo.

Diremos que se trata de una “clase abstracta” (entre comillas), pero con todos sus métodos abstractos.

Las interfaces no se extienden, se implementan.

Interfaces 2



IT BOARDING

BOOTCAMP

Interfaces 3

Veamos el código de la interface Comunicador

```
public interface Comunicador
{
   public void enviarMensaje(String mssg);
}
```

Se declara como interface (en vez de class). Sus métodos no tienen cuerpo, simplemente finalizan con ; (punto y coma)

Interfaces 4

La clase PalomaMensajera implementa Comunicador

```
public class PalomaMensajera extends Paloma implements Comunicador
{
   @Override
   public void enviarMensaje(String mssg)
   {
      demorar(10000); // demora 10 segundo simulando que es lendo
      System.out.println("Soy una paloma, enviando mensaje:"+mssg);
   }
}
```

Interfaces 5

La clase Telegrafo también implementa Comunicador

```
public class Telegrafo extends Antiguedad implements Comunicador
{
   @Override
   public void enviarMensaje(String mssg)
   {
      demorar(5000); // demora 5 segundo, es mejor que la paloma mensajera
      System.out.println("Soy un telegrado, enviando mensaje:"+mssg);
   }
}
```

Interfaces 6

La clase TelefonoCelular es otra implementación de Comunicador

```
public class TelefonoCelular extends Telefono implements Comunicador
{
   @Override
   public void enviarMensaje(String mssg)
   {
      demorar(1000); // demora 1 segundo, casi imperceptible
      System.out.println("Soy el cel, enviando mensaje:"+mssg);
   }
}
```

Interfaces 6

Usar interfaces nos permite ser independientes de la implementación

```
PablomaMensajera c = new PalomaMensajera();
c.enviarMensaje(“Hola”);
```

```
Comunicador c = new PalomaMensajera();
c.enviarMensaje(“Hola”);
```

Factoría de objetos

El patrón de diseño **Factory Method** permite solucionar el problema

```
public class ComunicadorFactory
{
   public static Comunicador crearComunicador()
   {
      return new PalomaMensajera();
   }
}
```

```
Comunicador c = ComunicadorFactory.crearComunicador();
c.enviarMensaje("Hola");
```

Tipos de dato genéricos (Generics)

¿Cómo **parametrizar los tipos de dato** de las variables y funciones?

```
public class Gen<T>
{
   // variable de instancia tipo T
   private T v;

   // metodos de acceso
   public void setV(T v){ this.v=v; }
   public T getV(){ return this.v; }

   //:
}
```

```
Gen<Integer> gI = new Gen<>();
gI.setV(10);

Gen<String> gS = new Gen<>();
gS.setV("Hola");
```

Tipos de dato genéricos (Generics) 2

Principales usos:

- **Colecciones**
- Interfaces

```
ArrayList<String> aS = new ArrayList<>();
aS.add("Pablo");
aS.add("Juan");
for(String s:aS) System.out.println(s);

ArrayList<Integer> aI = new ArrayList<>();
aI.add(1);
aI.add(2);
for(int i:aI) System.out.println(i);
```

Tipos de dato genéricos (Generics) 2

Principales usos:

- Colecciones
- **Interfaces**

```
public class ComparaEnteros implements Comparator<Integer>
{
   public int compare(int a,int b)
     {
          return a-b;
     }
}

public class ComparaCadenas implements Comparator<String>
{
   public int compare(String s1,String s2)
   {
      return s1.compareTo(s2);
   }
}
```

¿Dudas? ¿Preguntas?

IT BOARDING

BOOTCAMP

Gracias.

IT BOARDING

BOOTCAMP