Programa de Repaso: Herencia, Clases Abstractas e Interfaces

Herencia

La herencia permite definir una clase en base a otra ya existente. Se utiliza para reducir el desarrollo de código y agregar características o funcionalidades a una clase ya existente.

Por ejemplo:

Se tiene una clase Trabajador

Trabajador.java

```java
package uam.patrones.repaso.clases;

public class Trabajador {

	private String nombre;
	private String primerApellido;
	private String segundoApellido;
	private String RFC;
	private float salario;

	public String getNombre() {
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public String getPrimerApellido() {
		return primerApellido;
	}
	public void setPrimerApellido(String primerApellido) {
		this.primerApellido = primerApellido;
	}
	public String getSegundoApellido() {
		return segundoApellido;
	}
	public void setSegundoApellido(String segundoApellido) {
		this.segundoApellido = segundoApellido;
	}
	public String getRFC() {
		return RFC;
	}
	public void setRFC(String rFC) {
		RFC = rFC;
	}
	public float getSalario() {
		return salario;
	}
```

```
	public void setSalario(float salario) {
		this.salario = salario;
	}

	public String toString() {
		String mensaje ="";
		mensaje = nombre+" " + primerApellido+" "+
		segundoApellido+" " + RFC + " " + salario;
		return mensaje;
	}
}
```

Se tienen varios tipos de Trabajadores, en este caso: Recepcionista, Portero y Recamarero

Un Recepcionista es un Trabajador que además de los datos anteriores, tiene una categoría:

Recepcionista

```
package uam.patrones.repaso.clases;

public class Recepcionista extends Trabajador{

	private String categoria;

	public String getCategoria() {
		return categoria;
	}

	public void setCategoria(String categoria) {
		this.categoria = categoria;
	}

	public String toString() {
		String mensaje = "";
		mensaje = "El Recepcionista " + super.toString() + " " +
categoria;
		return mensaje;
	}

}
```

Un Portero es un Trabajador que además tiene la puerta que atiende

Portero

```
package uam.patrones.repaso.clases;

public class Portero extends Trabajador{


```

```
	private String puerta;

	public String getPuerta() {
		return puerta;
	}

	public void setPuerta(String puerta) {
		this.puerta = puerta;
	}

	/*Sobre escribe el método toString*/
	public String toString() {
		String mensaje = "";
		mensaje = "El Portero " + super.toString() + " " + puerta;
		return mensaje;
	}
}
```

Finalmente se tiene un **Recamarero**, el cual es un Trabajador que tiene adicionalmente el piso del que es responsable

Recamarero

```
package uam.patrones.repaso.clases;

public class Recamarero extends Trabajador{

	/*Además de los datos de un Trabajador, tiene un piso que atiende*/

	private int piso;

	public int getPiso() {
		return piso;
	}

	public void setPiso(int piso) {
		this.piso = piso;
	}

	/*Sobre escribe el método toString*/
	public String toString() {
		String mensaje = "";
		mensaje = "El Recamarero " + super.toString() + " " + piso;
		return mensaje;
	}

}
```

Se crea una clase **Principal** para manejar la lógica del programa y un clase OperacionesTrabajador para las funcionalidades

OperacionesTrabajador

```
package uam.patrones.repaso.operaciones;

import uam.patrones.repaso.clases.Trabajador;

public class OperacionesTrabajador {


	/*En este caso recibe a cualquier objeto de tipo trabajador
	 * e imprime sus datos llamando al método correspondiente
	 */
	public void imprimirDatosTrabajador(Trabajador trabajador) {
		System.out.println("Los datos del trabajador son: " );
		System.out.println(trabajador.toString());
	}

}
```

Principal.java

```
package uam.patrones.repaso.principal;

import uam.patrones.repaso.operaciones.OperacionesTrabajador;
import uam.patrones.repaso.clases.Portero;
import uam.patrones.repaso.clases.Recamarero;
import uam.patrones.repaso.clases.Recepcionista;
import uam.patrones.repaso.clases.Trabajador;

public class Principal {

	public static void main(String[] args) {

		manejarTrabajador();
		manejarRecamarero();
		manejarRecepcionista();
		manejarPortero();
	}

	private static void manejarTrabajador() {
		Trabajador trabajador = new Trabajador();

		trabajador.setNombre("Nombre");
		trabajador.setPrimerApellido("Primer Apellido");
		trabajador.setSegundoApellido("Segundo Apellido");
		trabajador.setRFC("RFC");
		trabajador.setSalario(1000F);
```

```
            OperacionesTrabajador operaciones = new
OperacionesTrabajador();
            operaciones.imprimirDatosTrabajador(trabajador);
      }

      private static void manejarRecamarero() {
            Recamarero recamarero = new Recamarero();

            recamarero.setNombre("Nombre");
            recamarero.setPrimerApellido("Primer Apellido");
            recamarero.setSegundoApellido("Segundo Apellido");
            recamarero.setRFC("RFC");
            recamarero.setSalario(3000F);
            recamarero.setPiso(2);

            OperacionesTrabajador operaciones = new
OperacionesTrabajador();
            operaciones.imprimirDatosTrabajador(recamarero);
      }

      private static void manejarRecepcionista() {
            Recepcionista recepcionista = new Recepcionista();

            recepcionista.setNombre("Nombre");
            recepcionista.setPrimerApellido("Primer Apellido");
            recepcionista.setSegundoApellido("Segundo Apellido");
            recepcionista.setRFC("RFC");
            recepcionista.setSalario(5000F);
            recepcionista.setCategoria("Negocios");

            OperacionesTrabajador operaciones = new
OperacionesTrabajador();
            operaciones.imprimirDatosTrabajador(recepcionista);
      }
      private static void manejarPortero() {
            Portero portero = new Portero();

            portero.setNombre("Nombre");
            portero.setPrimerApellido("Primer Apellido");
            portero.setSegundoApellido("Segundo Apellido");
            portero.setRFC("RFC");
            portero.setSalario(4000F);
            portero.setPuerta("Lateral Derecha");

            OperacionesTrabajador operaciones = new
OperacionesTrabajador();
            operaciones.imprimirDatosTrabajador(portero);
      }
}
```

En este caso se reutiliza código al no tener que crear un método que mande a imprimir los atributos de cada una de las clases, solo se realiza para la clase padre, en este caso, la clase Trabajador.

Sin embargo, es necesario considerar:

* ¿Es necesario utilizar (instanciar) la clase Trabajador?
* ¿Y si cada clase estuviera relacionada con funcionalidades diferentes?

Clases Abstractas

La respuesta a la primera pregunta es No, no es necesario instanciar la clase Trabajador, de hecho el objetivo de esta clase es que no se "utilice" más que para crear clases con sus atributos. Es aquí en donde aparece el concepto de clase **Abstracta**.

Una clase Abstracta no puede ser instanciada y se utiliza solamente para definir subclases. Se utiliza para crear una abstracción que agrupe objetos de distintos tipos y se quiere usar el concepto de **Polimorfismo**.

Trabajador

```
package uam.patrones.repaso.clases;

public abstract class Trabajador {

	private String nombre;
	private String primerApellido;
	private String segundoApellido;
	private String RFC;
	private float salario;

	public String getNombre() {
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public String getPrimerApellido() {
		return primerApellido;
	}
	public void setPrimerApellido(String primerApellido) {
		this.primerApellido = primerApellido;
	}
	public String getSegundoApellido() {
		return segundoApellido;
	}
	public void setSegundoApellido(String segundoApellido) {
		this.segundoApellido = segundoApellido;
	}
```

```
	public String getRFC() {
		return RFC;
	}
	public void setRFC(String rFC) {
		RFC = rFC;
	}
	public float getSalario() {
		return salario;
	}
	public void setSalario(float salario) {
		this.salario = salario;
	}

	public String toString() {
		String mensaje ="";
		mensaje = nombre+" " + primerApellido+" "+
		segundoApellido+" " + RFC + " " + salario;
		return mensaje;
	}
}
```

Como una clase abstracta no puede ser instanciada, es necesario eliminar esto en la clase principal.

Principal

```
package uam.patrones.repaso.principal;

import uam.patrones.repaso.clases.OperacionesTrabajador;
import uam.patrones.repaso.clases.Portero;
import uam.patrones.repaso.clases.Recamarero;
import uam.patrones.repaso.clases.Recepcionista;

public class Principal {

	public static void main(String[] args) {

		manejarRecamarero();
		manejarRecepcionista();
		manejarPortero();

	}

	private static void manejarRecamarero() {
		Recamarero recamarero = new Recamarero();
		recamarero.setNombre("Nombre");
		recamarero.setPrimerApellido("Primer Apellido");
		recamarero.setSegundoApellido("Segundo Apellido");
		recamarero.setRFC("RFC");
		recamarero.setSalario(3000F);
		recamarero.setPiso(2);
```

```
            OperacionesTrabajador operaciones = new
OperacionesTrabajador();
            operaciones.imprimirDatosTrabajador(recamarero);
      }

      private static void manejarRecepcionista() {

            Recepcionista recepcionista = new Recepcionista();

            recepcionista.setNombre("Nombre");
            recepcionista.setPrimerApellido("Primer Apellido");
            recepcionista.setSegundoApellido("Segundo Apellido");
            recepcionista.setRFC("RFC");
            recepcionista.setSalario(5000F);
            recepcionista.setCategoria("Negocios");

            OperacionesTrabajador operaciones = new
OperacionesTrabajador();
            operaciones.imprimirDatosTrabajador(recepcionista);
      }

      private static void manejarPortero() {

            Portero portero = new Portero();

            portero.setNombre("Nombre");
            portero.setPrimerApellido("Primer Apellido");
            portero.setSegundoApellido("Segundo Apellido");
            portero.setRFC("RFC");
            portero.setSalario(4000F);
            portero.setPuerta("Lateral Derecha");

            OperacionesTrabajador operaciones = new
OperacionesTrabajador();
            operaciones.imprimirDatosTrabajador(portero);
      }
}
```

Al ejecutar nuevamente el programa, el resultado será el mismo, pero sin imprimir los datos del Trabajador “puro”.

El concepto de Herencia se sigue manejando, solo que al momento de indicar a la clase padre como abstracta, ya no es posible instanciarla.

Clases Abstractas y Funcionalidades
Se agregan dos operaciones adicionales, para imprimir la hora de entrada y la hora de salida, estas dos actividades son comunes para todos los trabajadores, por lo que se agregan en la clase **OperacionesTrabajador**.

OperacionesTrabajador

```
package uam.patrones.repaso.operaciones;

public class OperacionesTrabajador {


	/*En este caso recibe a cualquier objeto de tipo trabajador
	 * e imprime sus datos llamando al método correspondiente
	 */
	public void imprimirDatosTrabajador(Trabajador trabajador) {
		System.out.println("Los datos del trabajador son: " );
		System.out.println(trabajador.toString());
	}

	public void imprimirHoraDeEntrada(Trabajador trabajador, String
horaEntrada) {
		System.out.println("Los datos del trabajador son: " );
		System.out.println(trabajador.toString());
		System.out.println("Y entró a trabajar a las: " + horaEntrada);
	}

	public void imprimirHoraDeSalida(Trabajador trabajador, String
horaSalida) {
		System.out.println("Los datos del trabajador son: " );
		System.out.println(trabajador.toString());
		System.out.println("Y salió de trabajar a las: " +
horaSalida+"\n");
	}
}
```

Se modifica la clase **Principal** para invocar estas funcionalidades

Principal

```
package uam.patrones.repaso.principal;

import uam.patrones.repaso.operaciones.OperacionesTrabajador;
import uam.patrones.repaso.clases.Portero;
import uam.patrones.repaso.clases.Recamarero;
import uam.patrones.repaso.clases.Recepcionista;


public class Principal {

	public static void main(String[] args) {
		manejarRecamarero();
		manejarRecepcionista();
		manejarPortero();
	}
```

```
	private static void manejarRecamarero() {

		Recamarero recamarero = new Recamarero();

		recamarero.setNombre("Nombre");
		recamarero.setPrimerApellido("Primer Apellido");
		recamarero.setSegundoApellido("Segundo Apellido");
		recamarero.setRFC("RFC");
		recamarero.setSalario(3000F);
		recamarero.setPiso(2);

		OperacionesTrabajador operaciones = new
OperacionesTrabajador();
		operaciones.imprimirDatosTrabajador(recamarero);
		operaciones.imprimirHoraDeEntrada(recamarero, "9:00:00");
		operaciones.imprimirHoraDeSalida(recamarero, "18:00:00");
	}
	private static void manejarRecepcionista() {
		Recepcionista recepcionista = new Recepcionista();

		recepcionista.setNombre("Nombre");
		recepcionista.setPrimerApellido("Primer Apellido");
		recepcionista.setSegundoApellido("Segundo Apellido");
		recepcionista.setRFC("RFC");
		recepcionista.setSalario(5000F);
		recepcionista.setCategoria("Negocios");

		OperacionesTrabajador operaciones = new
OperacionesTrabajador();
		operaciones.imprimirDatosTrabajador(recepcionista);
		operaciones.imprimirHoraDeEntrada(recepcionista, "10:00:00");
		operaciones.imprimirHoraDeSalida(recepcionista, "17:00:00");
	}
	private static void manejarPortero() {
		Portero portero = new Portero();

		portero.setNombre("Nombre");
		portero.setPrimerApellido("Primer Apellido");
		portero.setSegundoApellido("Segundo Apellido");
		portero.setRFC("RFC");
		portero.setSalario(4000F);
		portero.setPuerta("Lateral Derecha");

		OperacionesTrabajador operaciones = new
OperacionesTrabajador();
		operaciones.imprimirDatosTrabajador(portero);
		operaciones.imprimirHoraDeEntrada(portero, "11:00:00");
		operaciones.imprimirHoraDeSalida(portero, "19:00:00");
	}
}
```

¿Qué sucedería si se tuviera una funcionalidad diferente?, es decir, algo que solo hiciera un Recepcionista, un Portero y un Recamarero.

Considerar un método *imprimirActividades*, que solo imprimirá un mensaje, pero este sería distinto dependiendo el trabajador. En este caso, como se habla de funcionalidades distintas, se crearán clases que las contengan, sin embargo, se quiere seguir teniendo acceso a las funcionalidades comunes, por lo que será necesario utilizar Herencia. Nuevamente, se considera si será necesario instanciar la clase padre de las funcionalidades, como solo se desea utilizar la funcionalidad de las clases hijas, se declarará como abstracta.

OperacionesTrabajador

```
package uam.patrones.repaso.operaciones;

import uam.patrones.repaso.clases.Trabajador;

public abstract class OperacionesTrabajador {

	public void imprimirDatosTrabajador(Trabajador trabajador) {
		System.out.println("Los datos del trabajador son: " );
		System.out.println(trabajador.toString());
	}
	public void imprimirHoraDeEntrada(Trabajador trabajador, String
horaEntrada) {
		System.out.println("Los datos del trabajador son: " );
		System.out.println(trabajador.toString());
		System.out.println("Y entró a trabajar a las: " + horaEntrada);
	}
	public void imprimirHoraDeSalida(Trabajador trabajador, String
horaSalida) {
		System.out.println("Los datos del trabajador son: " );
		System.out.println(trabajador.toString());
		System.out.println("Y salió de trabajar a las: " + horaSalida);
	}
}
```

OperacionesPortero

```
package uam.patrones.repaso.operaciones;

import uam.patrones.repaso.clases.Portero;

public class OperacionesPortero extends OperacionesTrabajador{
	public void imprimirActividades(Portero portero) {
		System.out.println(portero.toString());
		System.out.println("Su función es atender a los clientes que
entran "
				+ "o salen del Hotel\n");
	}
}
```

OperacionesRecamarero

```
package uam.patrones.repaso.operaciones;

import uam.patrones.repaso.clases.Recamarero;

public class OperacionesRecamarero extends OperacionesTrabajador{
	public void queHago(Recamarero recamarero) {
		System.out.println(recamarero.toString());
		System.out.println("Su función es limpiar los cuartos del piso
que le "
				+ "fue asignado\n");
	}
}
```

OperacionesRecepcionista

```
package uam.patrones.repaso.operaciones;

import uam.patrones.repaso.clases.Recepcionista;

public class OperacionesRecepcionista extends OperacionesTrabajador{
	public void imprimeMisTareas(Recepcionista recepcionista) {
		System.out.println(recepcionista.toString());
		System.out.println("Su función es atender a los clientes que "
				+ "llegan pidiendo información o van a dejar el
hotel\n");
	}
}
```

Se modifica la clase Principal para reflejar estos cambios

Principal

```
package uam.patrones.repaso.principal;

import uam.patrones.repaso.clases.Portero;
import uam.patrones.repaso.clases.Recamarero;
import uam.patrones.repaso.clases.Recepcionista;
import uam.patrones.repaso.operaciones.OperacionesPortero;
import uam.patrones.repaso.operaciones.OperacionesRecamarero;
import uam.patrones.repaso.operaciones.OperacionesRecepcionista;

public class Principal {
	public static void main(String[] args) {
		manejarRecamarero();
		manejarRecepcionista();
		manejarPortero();
	}
```

```
	private static void manejarRecamarero() {

		Recamarero recamarero = new Recamarero();

		recamarero.setNombre("Nombre");
		recamarero.setPrimerApellido("Primer Apellido");
		recamarero.setSegundoApellido("Segundo Apellido");
		recamarero.setRFC("RFC");
		recamarero.setSalario(3000F);
		recamarero.setPiso(2);

		OperacionesRecamarero operaciones = new
OperacionesRecamarero();

		operaciones.imprimirDatosTrabajador(recamarero);
		operaciones.imprimirHoraDeEntrada(recamarero, "9:00:00");
		operaciones.imprimirHoraDeSalida(recamarero, "18:00:00");
		operaciones.queHago(recamarero);
	}

	private static void manejarRecepcionista() {

		Recepcionista recepcionista = new Recepcionista();

		recepcionista.setNombre("Nombre");
		recepcionista.setPrimerApellido("Primer Apellido");
		recepcionista.setSegundoApellido("Segundo Apellido");
		recepcionista.setRFC("RFC");
		recepcionista.setSalario(5000F);
		recepcionista.setCategoria("Negocios");

		OperacionesRecepcionista operaciones = new
OperacionesRecepcionista();
		operaciones.imprimirDatosTrabajador(recepcionista);
		operaciones.imprimirHoraDeEntrada(recepcionista, "10:00:00");
		operaciones.imprimirHoraDeSalida(recepcionista, "17:00:00");
		operaciones.imprimeMisTareas(recepcionista);
	}

	private static void manejarPortero() {

		Portero portero = new Portero();

		portero.setNombre("Nombre");
		portero.setPrimerApellido("Primer Apellido");
		portero.setSegundoApellido("Segundo Apellido");
		portero.setRFC("RFC");
		portero.setSalario(4000F);
		portero.setPuerta("Lateral Derecha");
```

```
            OperacionesPortero operaciones = new OperacionesPortero();
            operaciones.imprimirDatosTrabajador(portero);
            operaciones.imprimirHoraDeEntrada(portero, "11:00:00");
            operaciones.imprimirHoraDeSalida(portero, "19:00:00");
            operaciones.imprimirActividades(portero);
      }
}
```

Al ejecutar el programa se imprimen las actividades de cada tipo de trabajador. Sin embargo hay un detalle, la funcionalidad agregada, la realizan todos los trabajadores, pero cada uno de manera diferente, por lo que es deseable que todas tengan el mismo nombre. Hay dos formas de lograr esto, una con métodos abstractos y otra a través del uso de Interfaces.

Un método abstracto se declara en una clase abstracta solo con su firma (tipo de retorno, nombre y parámetros), es decir sin cuerpo. Al declararlo como abstracto, las clases que hereden de ella, se verán obligadas a implementarlo.

OperacionesTrabajador

```
package uam.patrones.repaso.operaciones;

import uam.patrones.repaso.clases.Trabajador;


public abstract class OperacionesTrabajador {


      /*En este caso recibe a cualquier objeto de tipo trabajador
       * e imprime sus datos llamando al método correspondiente
       */
      public void imprimirDatosTrabajador(Trabajador trabajador) {
            System.out.println("Los datos del trabajador son: " );
            System.out.println(trabajador.toString());
      }

      public void imprimirHoraDeEntrada(Trabajador trabajador, String
horaEntrada) {
            System.out.println("Los datos del trabajador son: " );
            System.out.println(trabajador.toString());
            System.out.println("Y entró a trabajar a las: " + horaEntrada);
      }

      public void imprimirHoraDeSalida(Trabajador trabajador, String
horaSalida) {
            System.out.println("Los datos del trabajador son: " );
            System.out.println(trabajador.toString());
            System.out.println("Y salió de trabajar a las: " + horaSalida);
      }
      public abstract void imprimirActividades(Trabajador trabajador);
}
```

Esto hará obligatorio que todas las clases que hereden de ella, deban implementar el método *imprimirActividades()* y como se reciben distintos tipos de trabajadores, se indica que se recibe el general, es decir la clase padre Trabajador.

Es necesario realizar modificaciones a las clases “hijas” de OperacionesTrabajador

OperacionesPortero

```
package uam.patrones.repaso.operaciones;

import uam.patrones.repaso.clases.Trabajador;

public class OperacionesPortero extends OperacionesTrabajador{
/*En caso de necesitar algún dato en particular del portero*/
            Portero po = (Portero)portero;
      public void imprimirActividades(Trabajador portero) {

            System.out.println(portero.toString());
            System.out.println("Su función es atender a los clientes que
entran "
                        + "o salen del Hotel\n");
      }
}
```

OperacionesRecamarero

```
package uam.patrones.repaso.operaciones;

import uam.patrones.repaso.clases.Trabajador;

public class OperacionesRecamarero extends OperacionesTrabajador{
/*En caso de necesitar un dato en especial del recamarero*/
            Recamarero re = (Recamarero)recamarero;
      public void imprimirActividades(Trabajador recamarero) {
            System.out.println(recamarero.toString());
            System.out.println("Su función es limpiar los cuartos del piso
que le "
                        + "fue asignado\n");
      }
}
```

OperacionesRecepcionista

```
package uam.patrones.repaso.operaciones;

import uam.patrones.repaso.clases.Trabajador;

public class OperacionesRecepcionista extends OperacionesTrabajador{
      public void imprimirActividades(Trabajador recepcionista) {
/*En caso de necesitar un dato en particular del recepcionista*/
```

```
            Recepcionista re = (Recepcionista)recepcionista;
            System.out.println(recepcionista.toString());
            System.out.println("Su función es atender a los clientes que "
                        + "llegan pidiendo información o van a dejar el
hotel\n");
      }
}
```

Se hacen las modificaciones necesarias en la clase Principal

Principal

```
package uam.patrones.repaso.principal;

import uam.patrones.repaso.clases.Portero;
import uam.patrones.repaso.clases.Recamarero;
import uam.patrones.repaso.clases.Recepcionista;
import uam.patrones.repaso.operaciones.OperacionesPortero;
import uam.patrones.repaso.operaciones.OperacionesRecamarero;
import uam.patrones.repaso.operaciones.OperacionesRecepcionista;

public class Principal {

      public static void main(String[] args) {

            manejarRecamarero();
            manejarRecepcionista();
            manejarPortero();

      }

      private static void manejarRecamarero() {

            Recamarero recamarero = new Recamarero();

            recamarero.setNombre("Nombre");
            recamarero.setPrimerApellido("Primer Apellido");
            recamarero.setSegundoApellido("Segundo Apellido");
            recamarero.setRFC("RFC");
            recamarero.setSalario(3000F);
            recamarero.setPiso(2);

            OperacionesRecamarero operaciones = new
OperacionesRecamarero();

            operaciones.imprimirDatosTrabajador(recamarero);
            operaciones.imprimirHoraDeEntrada(recamarero, "9:00:00");
            operaciones.imprimirHoraDeSalida(recamarero, "18:00:00");
            operaciones.imprimirActividades(recamarero);

      }
```

```
	private static void manejarRecepcionista() {

		Recepcionista recepcionista = new Recepcionista();

		recepcionista.setNombre("Nombre");
		recepcionista.setPrimerApellido("Primer Apellido");
		recepcionista.setSegundoApellido("Segundo Apellido");
		recepcionista.setRFC("RFC");
		recepcionista.setSalario(5000F);
		recepcionista.setCategoria("Negocios");

		OperacionesRecepcionista operaciones = new
OperacionesRecepcionista();
		operaciones.imprimirDatosTrabajador(recepcionista);
		operaciones.imprimirHoraDeEntrada(recepcionista, "10:00:00");
		operaciones.imprimirHoraDeSalida(recepcionista, "17:00:00");
		operaciones.imprimirActividades(recepcionista);

	}

	private static void manejarPortero() {

		Portero portero = new Portero();

		portero.setNombre("Nombre");
		portero.setPrimerApellido("Primer Apellido");
		portero.setSegundoApellido("Segundo Apellido");
		portero.setRFC("RFC");
		portero.setSalario(4000F);
		portero.setPuerta("Lateral Derecha");

		OperacionesPortero operaciones = new OperacionesPortero();
		operaciones.imprimirDatosTrabajador(portero);
		operaciones.imprimirHoraDeEntrada(portero, "11:00:00");
		operaciones.imprimirHoraDeSalida(portero, "19:00:00");
		operaciones.imprimirActividades(portero);
	}
}
```

Uso de Interfaces

Una interfaz se define como una clase abstracta que no encapsula datos, solamente operaciones (firmas) que las clases deben implementar.

Una interfaz se distingue de una clase abstracta en que no contiene atributos, ni funciones implementadas, solamente las firmas de los métodos que serán implementados por las clases que implementen la interfaz.

En este caso, no se podría crear una interfaz ya que se tienen clases implementadas, por lo que sería necesario crear una interfaz con la funcionalidad en común. Desde este punto de vista, la clase que la implemente debería “heredar” de dos clases, la que tiene clases implementadas y la que tiene las clases que ésta debería implementar.

Java no permite el uso de la herencia múltiple, pero esto se puede resolver con el uso de interfaces.

OperacionesCompartidas

```
package uam.patrones.repaso.operaciones;

import uam.patrones.repaso.clases.Trabajador;

public interface OperacionesCompartidas {
	public void imprimirActividades(Trabajador trabajador);
}
```

Se elimina el método abstracto de la clase **OperacionesTrabajador**

Se implementa la interfaz en las clase necesarias

OperacionesPortero

```
package uam.patrones.repaso.operaciones;

import uam.patrones.repaso.clases.Trabajador;

public class OperacionesPortero extends OperacionesTrabajador implements
OperacionesCompartidas{

	public void imprimirActividades(Trabajador portero) {
		System.out.println(portero.toString());
		System.out.println("Su función es atender a los clientes que
entran " + "o salen del Hotel\n");
	}
}
```

OperacionesRecamarero

```
package uam.patrones.repaso.operaciones;

import uam.patrones.repaso.clases.Trabajador;

public class OperacionesRecamarero extends OperacionesTrabajador
implements OperacionesCompartidas{
	public void imprimirActividades(Trabajador recamarero) {
		System.out.println(recamarero.toString());
		System.out.println("Su función es limpiar los cuartos del piso
que le "   + "fue asignado\n");
	}
}
```

OperacionesRecepcionista

```
package uam.patrones.repaso.operaciones;

import uam.patrones.repaso.clases.Trabajador;

public class OperacionesRecepcionista extends OperacionesTrabajador
implements OperacionesCompartidas{

	public void imprimirActividades(Trabajador recepcionista) {
		System.out.println(recepcionista.toString());
		System.out.println("Su función es atender a los clientes que "
				+ "llegan pidiendo información o van a dejar el
hotel\n");
	}

}
```

Finalmente se ejecuta el programa