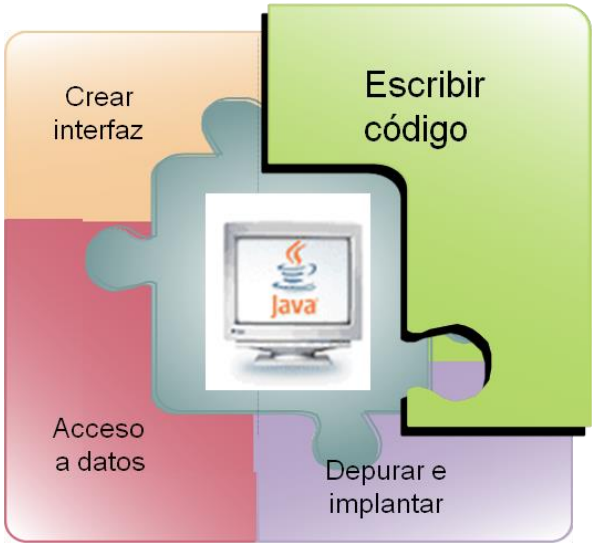
Clases Abstractas e Interfaces

- **Unidad 8 - Segunda Parte**

Introducción



- Métodos y Clases Abstractas
- Métodos y Clases Finales
- ¿Qué es una interface?
  - Definiendo interfaces
  - Implementando interfaces
  - ¿Por qué usar interfaces?
  - Interface vs. Clase Abstracta
  - Interface como Tipo
  - Interface y Clase: Similitudes y diferencias
  - Relación entre una interface y una clase
  - Herencia entre interfaces
  - Posibles errores implementando interfaces
- Clases Internas o Anidadas

Clases Abstractas

- **Clases abstractas:**
  - Clases de las que no se pueden crear Objetos, pero que agrupan características de clases que derivan de ellas
  - Ejemplo:

    Por la carretera no circulan `Vehículos` en abstracto, circulan `Coches`, `Camiones`, `Bicicletas`, etc...

    La finalidad de `Vehículos` es definir los aspectos comunes a todos los modelos de vehículos que se fabrican

Clases Abstractas

- En Java se puede indicar explícitamente que una clase es abstracta y no se pueden crear objetos de ella:

```
abstract class Vehiculo {
  //...
}
```

Métodos Abstractos

- **Son métodos que no tienen implementación**
  - Para crear un método abstracto sólo escribir la declaración del método sin el cuerpo y usar la palabra reservada abstract
  - No llevan las llaves { }

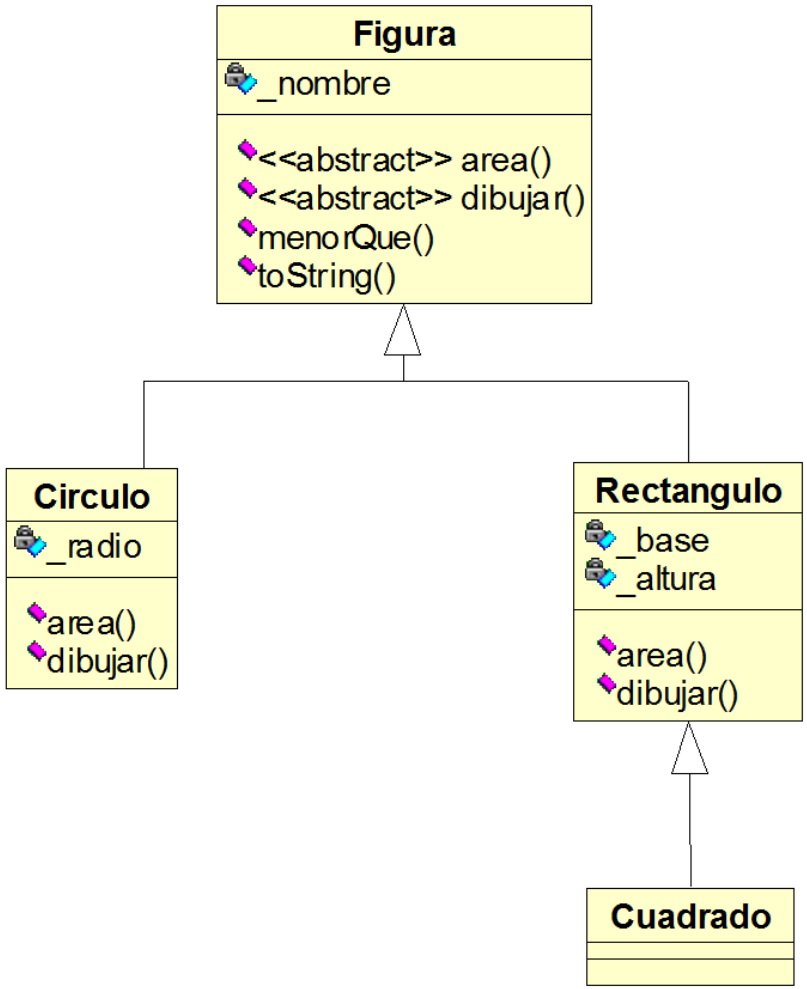
- Ejemplo:

```
public abstract void algunMetodo();  //no hay cuerpo
```

Métodos Abstractos

- **Si una clase posee un método abstracto es clase abstracta**
- **Si una clase deriva de una clase abstracta y no implementa alguno de los métodos abstractos de la clase base, entonces hereda el método abstracto y se convierte en abstracta:**
    - Para poder crear objetos hay que implementar todos los métodos abstractos heredados

Clases Abstractas: Ejemplo



Métodos finales

- **Expresa que un determinado método no se puede sobrescribir o redefinir (override)**

```
final public boolean menorQue (Figura f) {
        return ( this.area() < f.area() );
}
```

  - Las clases derivadas NO pueden sobrescribir **`menorQue`**

    Ojo: sobrescribir (override) implica mantener la signatura del método. Si podré sobrecargarlo (overload)

Clases finales

- **Se puede prohibir que una clase sea extendida declarándola como final**

```
final class Cuadrado extends Rectangulo
{
        public Cuadrado (double lado) {
                super (lado, lado);
        }

        public Cuadrado () {
                super (10.0, 10.0);
        }
}
```

  - No es posible definir una nueva clase que derive de **Cuadrado**

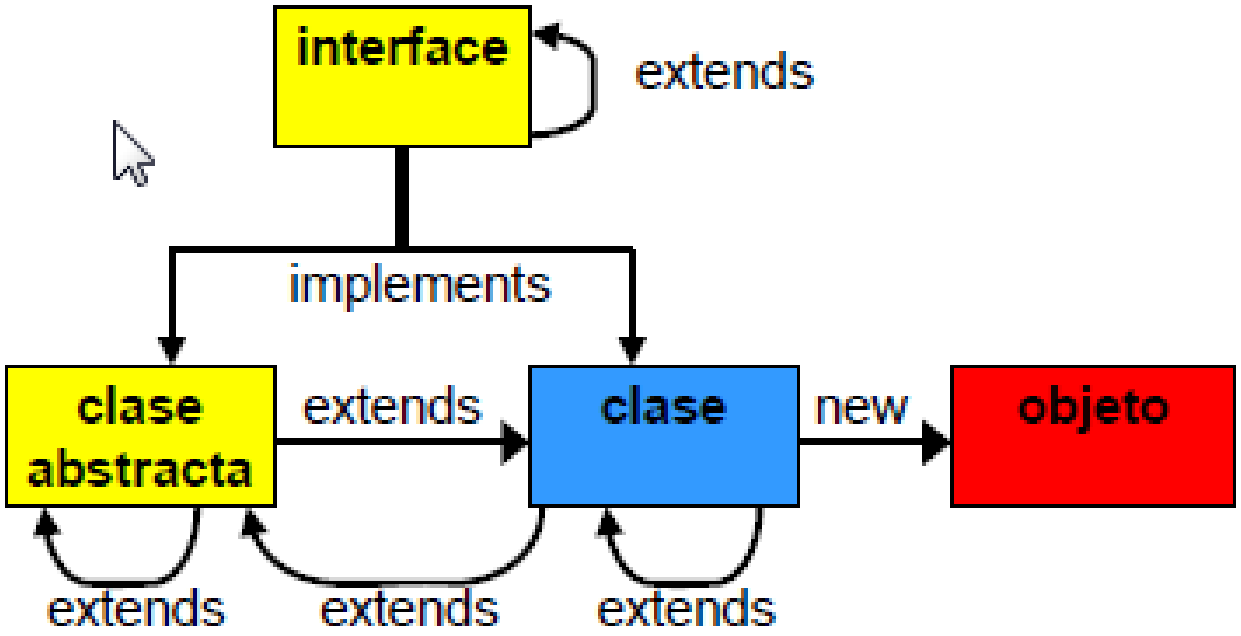
¿Qué es una interface?

- **Es un elemento java** que especifica de forma estándar y pública el comportamiento de clases
  - Dice lo que se ofrece pero no dice cómo se hace (define un contrato)
- Todos lo métodos de una interface son métodos abstractos
- Una clase concreta debe implementar (implements) la interface, es decir, implementar todos los métodos
- Permite la implementación de clases con comportamientos comunes, sin importar su ubicación en la jerarquía de clases

¿Qué es una interface?

- **Una interface proporciona:**
  - valores constantes, que son variables “public static final”
  - métodos, que son “public”
  - NO incluye constructores
- **Las interfaces**
  - pueden extenderse con nuevas constantes y/o métodos
  - pueden implementarse totalmente dando lugar a una clase que debe proporcionar código para todos y cada uno de los métodos definidos
  - pueden implementarse parcialmente dando lugar a una clase abstracta

¿Qué es una interface?



Definiendo interfaces

- Sintaxis para definir una interface:

```
public   interface    <NombreInterface>  {
   // metodos sin cuerpo
}
```

- Ejemplo: interface que define relaciones entre dos objetos de acuerdo al “orden natural” de los objetos

```
public interface Relacion {
  public boolean esMayor(Object a, Object b);
  public boolean esMenor(Object a, Object b);
  public boolean esIgual(Object a, Object b);
}
```

Implementando interfaces

- **Se dice que una clase implementa una interfaz cuando proporciona código concreto para los métodos definidos en la interfaz**
- Para crear una clase concreta que implementa una interface, se usa la palabra reservada **implements**
  - De una misma interfaz pueden derivarse múltiples implementaciones
  - Una misma clase puede implementar varias interfaces (implementación múltiple)
  - Si una clase no implementa todos los métodos definidos en una interfaz sino sólo parte de ellos, el resultado es una clase abstracta (implementación parcial)

Implementando interfaces

- **Ejemplo:**

```
/*** Clase Linea implementa la interface  Relacion */
public class Linea implements Relacion {
  private double x1;
  private double x2;
  private double y1;
  private double y2;

  public Linea(double x1, double x2, double y1, double y2){
      this.x1 = x1;
      this.x2 = x2;
      this.y1 = y1;
      this.y2 = y2;
  }
  public double getLongitud(){
    double longitud = Math.sqrt((x2-x1)*(x2-x1) +(y2-y1)* (y2-y1));
    return longitud;
  }
```

Implementando interfaces. Ejemplo

```
    public boolean  esMayor( Object a, Object b){
      double aLen = ((Linea)a).getLongitud();
      double bLen = ((Linea)b).getLongitud();
      return (aLen > bLen);
    }
    public boolean esMenor( Object a, Object b){
      double aLen = ((Linea)a).getLongitud();
      double bLen = ((Linea)b).getLongitud();
      return (aLen < bLen);
    }
    public boolean esEqual( Object a, Object b){
      double aLen = ((Linea)a).getLongitud();
      double bLen = ((Linea)b).getLongitud();
      return (aLen == bLen);
    }
}
```

¿Por qué usar interfaces?

- Para revelar la interface de la programación de un objeto (funcionalidad del objeto) sin revelar su implementación (encapsulado)
  - La implementación puede cambiar sin afectar el llamador de la interface, que no necesita la implementación en tiempo de compilación
- Para tener implementación de métodos similares (comportamientos) en clases sin relacionar
- Para modelar herencia múltiple, imponiendo conjuntos múltiples de comportamientos a la clase

Interface vs. Clase Abstracta

- Todos los métodos de una interface son métodos abstractos mientras algunos métodos de una clase abstracta son métodos abstractos
  - Los métodos abstractos de una clase abstracta tienen el modificador abstract
- Una interfaz solo puede definir constantes mientras que una clase abstracta puede tener atributos
- Las interfaces no tienen ninguna relación de herencia directa con una clase particular, se definen independientemente

Interface como Tipo

- La definición de una interface implica una definición de un nuevo tipo de referencia y por ello se puede usar el nombre de la interface como nombre de tipo
- Si se define una variable cuyo tipo es una interface, se le puede asignar un objeto que es una instancia de una clase que implementa la interface
  - Ejemplo: la clase Person implementa PersonInterface

```
Person p1 = new Person();
PersonInterface pi1 = p1;
PersonInterface pi2 = new Person();
```

Interface y Clases: características comunes

- Interfaces y clases son tipos
  - Significa que una interface se puede usar en lugares donde una clase se puede usar
  - Ejemplo: la clase Person implementa PersonInterface

```
PersonInterface pi = new Person();
```

- Tanto la interface como la clase definen métodos

Interface y Clases: características diferentes

- Todos los métodos de una interface son métodos abstractos
  - No tienen cuerpo
- No se puede crear una instancia de una interface
  - Ejemplo:

    PersonInterface pi = new PersonInterface();
- Una interface sólo puede ser implementado por clases o extendido por otras interfaces

Relación entre una interface y una clase

- Una clase concreta sólo puede extender una super-clase, pero puede implementar múltiples interfaces
  - El lenguaje Java no permite herencia múltiple, pero las interfaces proporcionan una alternativa
- Todos los métodos abstractos de todas las interfaces tiene que ser implementados por la clase concreta

```
public class IndustrialStudent extends Student
    implements PersonInterface, OtraInterface, MasInterface
{
  // todos los métodos abstractos de todas las
  // interfaces deben ser implementados
}
```

Herencia entre interfaces

- Las interfaces no son parte de la jerarquía de clases
- Sin embargo, las interfaces pueden tener relación de herencia entre ellas

```
public interface PersonInterface {
  void doSomething();
}
public interface StudentInterface extends PersonInterface {
  void doExtraSomething();
}
```

Posibles errores implementando interfaces

- Cuando una clase implementa una interface siempre debe implementar todas los métodos de la interface
- En la implementación debe declararse todos los métodos public
- Las interfaces no pueden tener variables de instancia, pero es legal especificar constantes
  - Todas las variables en una interface son automáticamente public static final por lo que se puede omitir en la declaración

```
public interface SwingConstants {
  int NORTH = 1;
  int NORTHEAST = 2;
  int EAST = 3;
  . . .
}
```

Clases Internas o Anidadas

- **Java nos permite definir Clases Internas (o Anidadas o Embebidas) o Inner Class:**
  - De manera general es una clase que se declara dentro de otra clase

```
class Externa {
   . . .
      class Interna {
            . . .
      }
}
```

  - La clase interna puede acceder directamente a los miembros de la clase contenedora

Clases Internas Anónimas

- **Una clase anónima no tiene nombre**
- **En vez de definirse como entidad diferenciada con una cabecera class, se define en el momento en que se instancia**
- **Nos aparecerán al implementar las clases Oyentes de eventos dentro de la Programación Gràfica**
- **La definición y la creación del objeto se hacen en el mismo paso:**
  - new seguida de la definición de la clase entre llaves
  - new seguida del nombre de la clase de la que hereda (sin extends) y la definición de la clase entre llaves
  - new seguida del nombre de la interface que implementa (sin implements) y la definición de la clase entre llaves

```
boton1.addActionListener( new ActionListener() {
   public void actionPerformed( Event evt ) { …}
  } //cierre de la clase anónima
); //cierre del argumento del método
```