Polimorfismo

- Herencia y Upcasting
  - Métodos polimórficos
- *Bindings*
  - *Binding* tardío
  - *Binding* dinámico
- Repaso de clases abstractas con un ejemplo

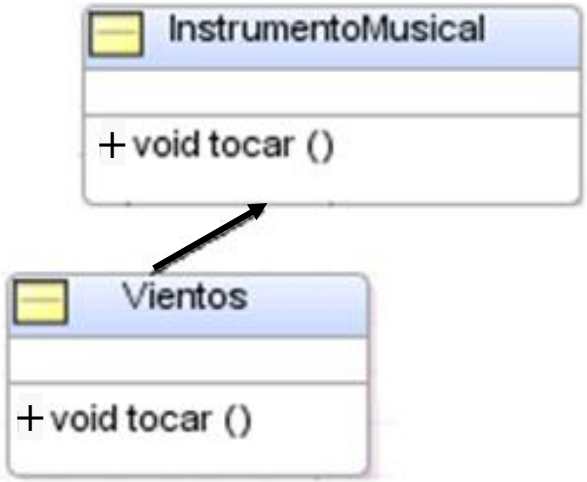
Polimorfismo
Herencia y Upcasting

Analicemos esta jerarquía de clases.

```
public abstract class InstrumentoMusical{
    public abstract void tocar(Nota n);
}
```

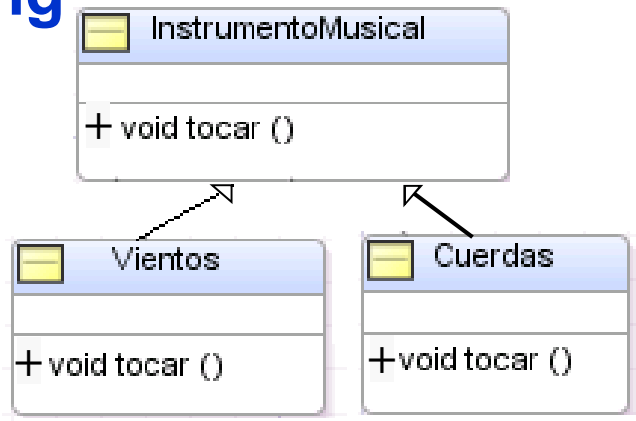
```
public class Vientos extends InstrumentoMusical{
   public void tocar(Nota n) {
     System.out.print("Vientos.tocar(): "+n);
   }
}
```

```
public class CancionSimple {
 private Nota[] pentagrama = new Nota[100];
 public void sonar(Vientos i){
  for (Nota n:pentagrama)
      i.tocar(n);
 }
 public static void main(String[] args) {
  CancionSimple cs = new CancionSimple();
  Vientos flauta = new Vientos();
  cs.sonar(flauta);
 }
}
```



¿Es extensible este programa? Qué sucede si crece la jerarquía de instrumentos musicales?

Polimorfismo
Herencia y Upcasting



¿Qué sucede si agregamos la subclase cuerdas?

```
public class CancionSimple {
 private Nota[] pentagrama =new Nota[100];

 public void sonar(Vientos i){
   for (Nota n:pentagrama)
      i.tocar(n);
 }

 public void sonar(Cuerdas i){
    for (Nota n:pentagrama)
       i.tocar(n);
 }

 public static void main(String[] args){
   CancionSimple cs = new CancionSimple();
   Vientos flauta = new Vientos();
   cs.sonar(flauta);

   Cuerdas piano = new Cuerdas();
   cs.sonar(piano);

 }
}
```

Solución apropiada

```
public class CancionSimple {
  Nota[] pentagrama = new Nota[100];
  public void sonar(InstrumentoMusical i){
  for (Nota n:pentagrama)
      i.tocar(n);
  }

 public static void main(String[] args) {
  CancionSimple cs = new CancionSimple();
  Vientos flauta=new Vientos();
  cs.sonar(flauta);
  Cuerdas guitarra=new Cuerdas();
  cs.sonar(guitarra);
 }
}
```

Acepta como parámetro una referencia a un Instrumento o a cualquier derivado de Instrumento

se hace un **upcasting** automático!

El polimorfismo permite escribir código que hable con la clase básica y pasar referencias de clases derivadas.

Polimorfismo y Binding Dinámico

¿Puede el compilador java saber que el objeto **InstrumentoMusical** pasado como parámetro en **sonar()**, es una referencia a un objeto Vientos o Cuerdas?

```
package musica;

public class CancionSimple {
  private Nota[] pentagrama = new Nota[100];
  public void sonar(InstrumentoMusical i){
     for (Nota n:pentagrama)
         i.tocar(n);
  }
}
```

NO!!

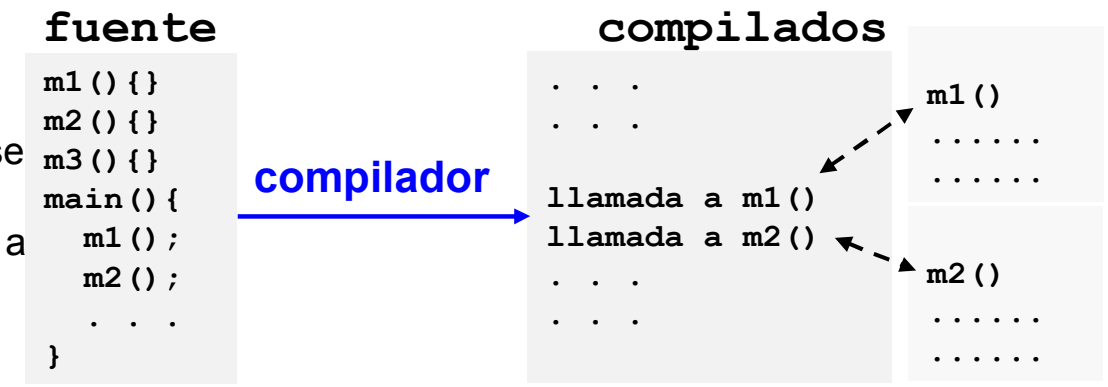
No hay manera de saber, mirando el código, que tipo de instrumento ejecutará el **tocar()**, eso depende **del valor que tome la variable i en ejecución**.

```
package musica;

public class CancionSimpleTest {
  public static void main(String[] args) {
     CancionSimple cs = new CancionSimple();
     if (args[0].equals("V"))
        Vientos i=new Vientos();
       else
        Cuerdas i=new Cuerdas();
     cs.sonar(i);
  }
}
```

Conectar la invocación de un método con el cuerpo del método, se llama *binding*. Si el *binding*, se hace en compilación, se llama *binding* temprano y si se hace en ejecución *binding* dinámico o tardío.

***Binding* temprano**
En compilación se resuelven todas las invocaciones a métodos

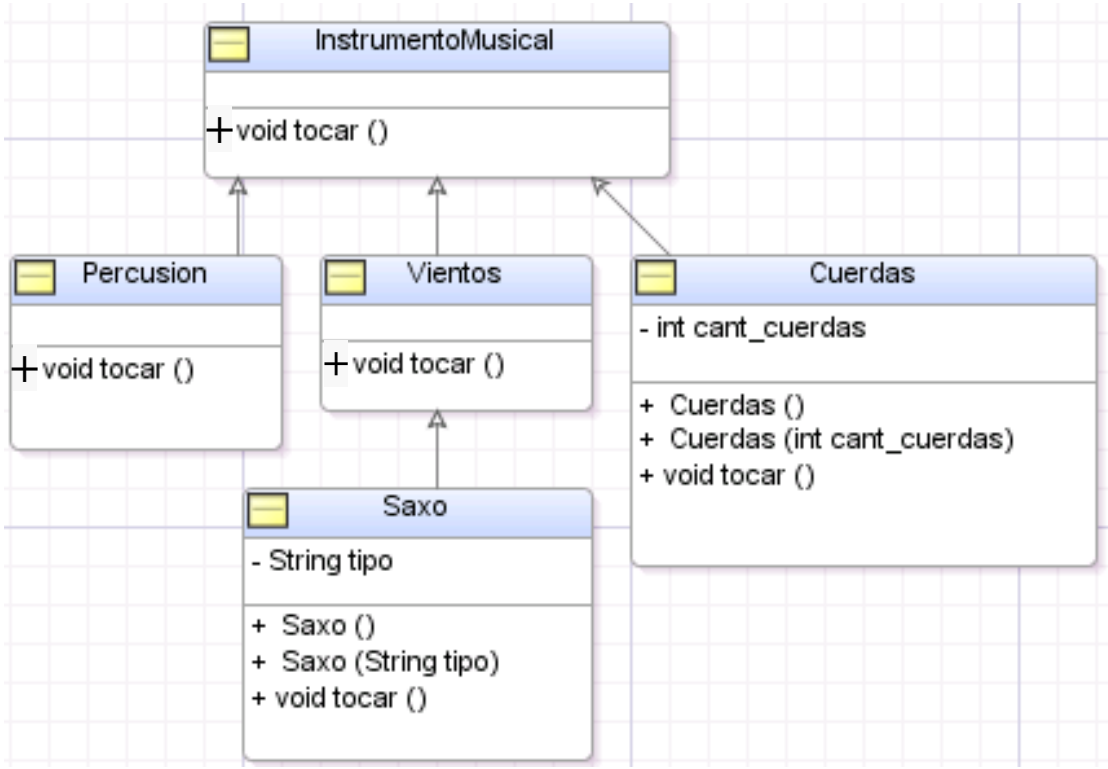


***Binding* Dinámico**
El compilador no conoce qué método invocará cuando recibe la referencia a un objeto InstrumentoMusical. El *binding* se hace en ejecución.

Polimorfismo y Binding Dinámico

- Java, normalmente usa **binding dinámico**: la resolución de las invocaciones a métodos, se hace en ejecución (run-time), basándose en el tipo del objeto receptor.
- Java provee un mecanismo para **determinar en ejecución el tipo del objeto receptor** e invocar al método apropiado.
- Aprovechando que java usa el binding dinámico, el programador puede escribir **código que hable con la clase base**, conociendo que las clases derivadas funcionarán correctamente usando el mismo código.
- El punto anterior hace que el programa sea **extensible**: un programador podría agregar subclases de InstrumentoMusical en la jerarquía de instrumentos y el método `sonar(InstrumentoMusical i)` de la clase `CancionSimple` seguirá funcionando correctamente, porque esta clase habla con la clase base `InstrumentoMusical`.

  Los programas orientados a objetos bien diseñados usan este modelo.



Polimorfismo y Binding Dinámico

```
package polimorfismo;

public class CancionSimple {
 private Nota[] pentagrama = new Nota[100];
 public void sonar(InstrumentoMusical i) {
    for (Nota n: pentagrama)
         i.tocar(n);
 }
 public static void main(String[] args) {
  CancionSimple cs = new CancionSimple();
  InstrumentoMusical[] m={new Viento(),new Cuerdas(),
                          new Percusion(),new Saxo()};
  for (int i=0;i<m.length;i++)
     cs.sonar(m[i]);
 }
}
```

Cuando ponemos objetos de subclases de InstrumentoMusical, automáticamante se hace un **upcasting** a InstrumentoMusical.

El *binding dinámico* resuelve a que método invocar, cuando más de una clase en una jerarquía de herencia implementó un método (en este caso `tocar()`).

La JVM busca la implementación de los métodos invocados de acuerdo al tipo de objeto referenciado en tiempo de ejecución, de esta manera se da lugar al *binding dinámico*.

La declaración del tipo de la variable es utilizada por el compilador de java para chequeo de errores durante la fase de compilación, sin determinarse cual será el cuerpo del método que se ejecutará. Si se agregan nuevas clases a la jerarquía de herencia el método `sonar(InstrumentoMucical i)`, no se ve afectado ➔ esto es lo que provee el polimorfismo.

El polimorfismo significa "diferentes formas". En POO se tiene una interface común en la clase base y diferentes versiones de métodos ligados dinámicamente: *binding dinámico*.