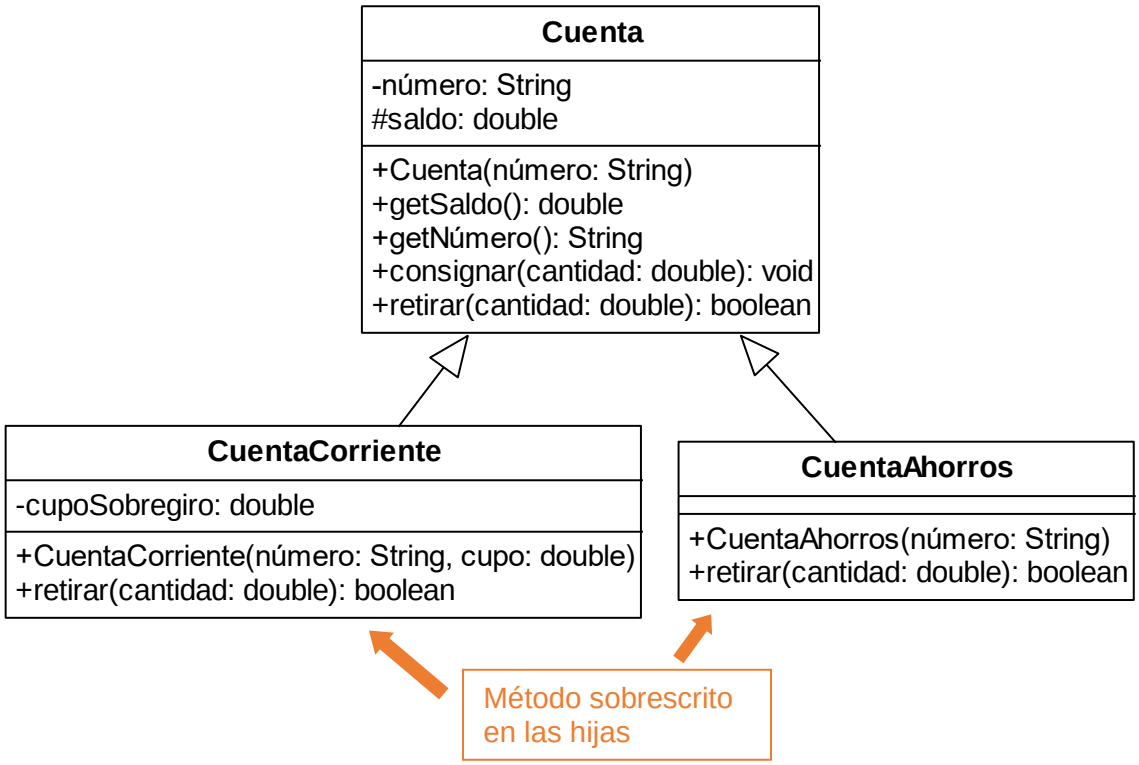
Clases y Métodos Abstractos

Material de clase elaborado por Sandra Victoria Hurtado Gil

El banco MuchoDinero ofrece cuentas de ahorro y cuentas corrientes. De todas las cuentas guarda el número y el saldo, y, además, en las cuentas corrientes guarda la cantidad autorizada de sobregiro. Ahora el banco tiene una nueva regla: en los retiros de una cuenta de ahorro debe verificar que el saldo final no puede ser inferior a $20.000.

El método “retirar” estaba definido en la clase padre y solo la clase CuentaCorriente lo sobrescribía, para tener en cuenta el cupo de sobregiro. Sin embargo, con esta nueva regla la clase CuentaAhorros también tiene que sobrescribir este método.

El diagrama de clases sería:



Incluso si más adelante se crean nuevos tipos de cuentas, cada una tendrá unas condiciones diferentes para retirar dinero. Esto lleva a pensar que el método “retirar” en la clase Cuenta ya no será usado directamente por ninguna de las clases hijas. Es posible pensar en dejar el código de este método solo con lo mínimo necesario para que compile, por ejemplo:

```
/**
 * Retira o saca una cantidad de dinero de la cuenta
 * @param  cantidad  la cantidad que se desea retirar, en pesos.
 * @return  si se pudo hacer el retiro o no
 */
public boolean retirar(double cantidad) {
      return false;     // Solo retorna un valor, no hace nada
                        // porque las hijas lo sobrescribirán
}
```

Pero esto lleva a otro problema: ¿Qué pasaría si por error se crea un objeto de la clase Cuenta, y no de las clases CuentaCorriente y CuentaAhorros? El programa no funcionaría apropiadamente, pues no se puede retirar de una cuenta que no es corriente ni de ahorros, porque no se sabe cuáles reglas aplicar.

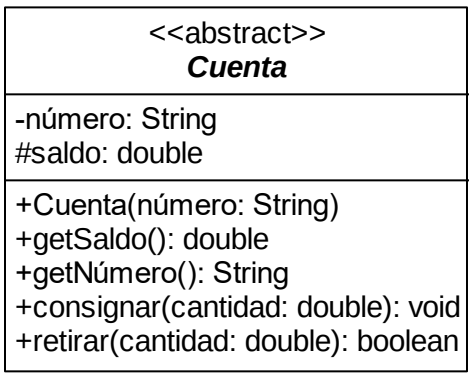
1. Clases abstractas y su representación

La solución es definir la clase Cuenta como una clase especial, que tiene atributos y métodos, pero de la cual no se desean crear instancias (objetos). Los atributos y métodos que define esta clase son usados solo en las clases hijas. Este tipo de clases se llaman clases abstractas.

Una **clase abstracta** es una clase de la cual no se pueden crear objetos o instancias; es decir, no se puede usar el operador *new* directamente para objetos de esa clase, pero sí se pueden crear objetos de sus clases hijas.

En el diagrama de clases, una clase abstracta se representa escribiendo antes del nombre de la clase la palabra *abstract*, encerrada en símbolos: << y >>.
En el caso de contar con alguna herramienta para elaborar los diagramas de clases, el nombre de las clases abstractas aparece en cursiva.

Por ejemplo, para el caso de la clase Cuenta:



Con la clase Cuenta definida como una clase abstracta, ya no puede crear por error un objeto de esta clase. De esta manera se garantiza que el programa solo puede crear objetos de las clases CuentaCorriente y CuentaAhorros, para así funcionar adecuadamente.

2. Métodos abstractos y su representación

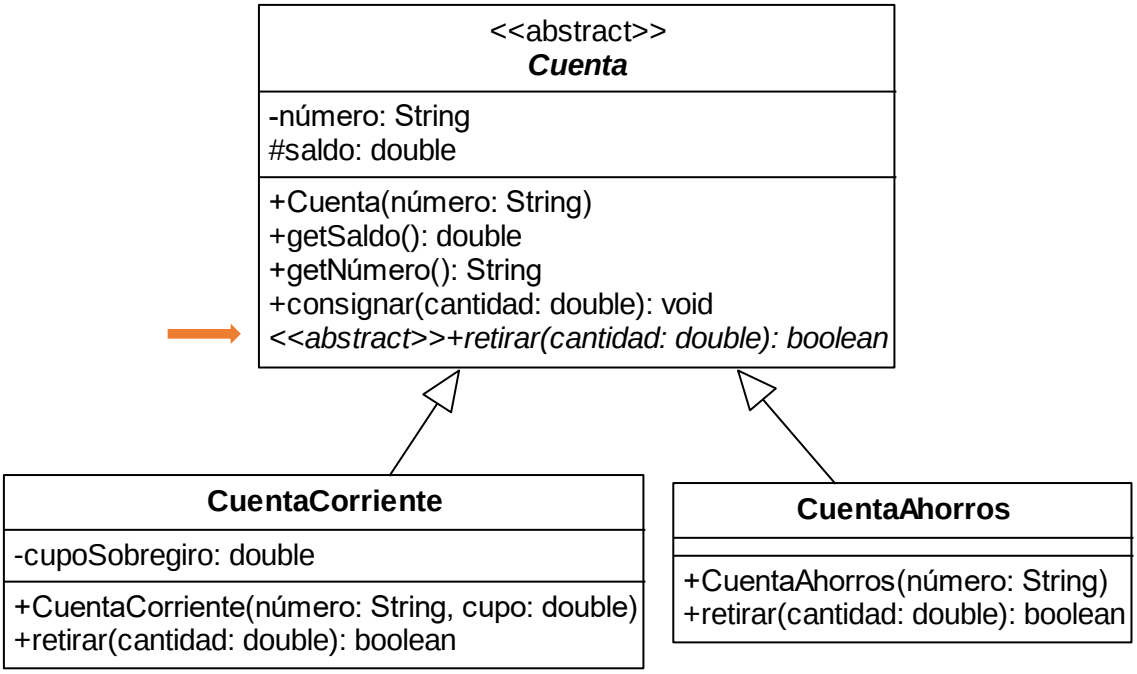
Una de las ventajas que tienen las clases abstractas es que pueden definir métodos que no tengan cuerpo, solo el encabezado, debido a que serán sobrescritos por las clases hijas.

En el caso de la clase Cuenta, se tenía el método “retirar”, el cual se dejó solo con una instrucción para que compilara y para poder ser usado con variables de tipo Cuenta, pero sin que fuera la operación real de retirar. Este es un ejemplo de un método que no necesita tener código en la clase padre.

Un método que no tiene cuerpo, solo el encabezado, y que debe ser sobrescrito por las clases hijas, es un **método abstracto.** Se debe tener en cuenta que los métodos abstractos solo pueden ser definidos en clases abstractas.

De manera similar a las clases abstractas, para mostrar que un método es abstracto en el diagrama de clases, se escribe la palabra *abstract* antes del nombre del método, entre los símbolos << y >>. También el nombre del método va en cursiva.

Con todos estos elementos, el diagrama de clases para las cuentas bancarias queda:



Implementación en Java

En Java se utiliza la palabra reservada *abstract* en las clases y métodos abstractos.
Además, los métodos abstractos solo tienen el encabezado, el cual termina en punto y coma. Estos métodos no tienen cuerpo.

Por ejemplo, el código de la clase Cuenta es:

```java
/**
 * Una cuenta bancaria, es decir, un registro en
 * una entidad bancaria que tiene un saldo (cantidad de dinero)
 * que se puede modificar mediante retiros o consignaciones.
 * @version 4.5
 */
public abstract class Cuenta {
	private String numero;
	protected double saldo;

	public Cuenta(String numero) {
		this.numero = numero;
		this.saldo = 0;
	}

	public double getSaldo() {
		return this.saldo;
	}

	public String getNumero() {
		return this.numero;
	}

	/**
	 * Consigna o adiciona una cantidad de dinero en la cuenta,
	 * lo cual incrementa el saldo.
	 * @param   cantidad    la cantidad de dinero que se desea
	 *                       consignar, en pesos
	 */
	public void consignar(double cantidad) {
		saldo = saldo + cantidad;
	}

	/**
	 * Retira o saca una cantidad de dinero de la cuenta
	 * @param  cantidad  la cantidad que se desea retirar, en pesos.
	 * @return  si se pudo hacer el retiro o no
	 */
	public abstract boolean retirar(double cantidad);

} // Fin clase Cuenta
```

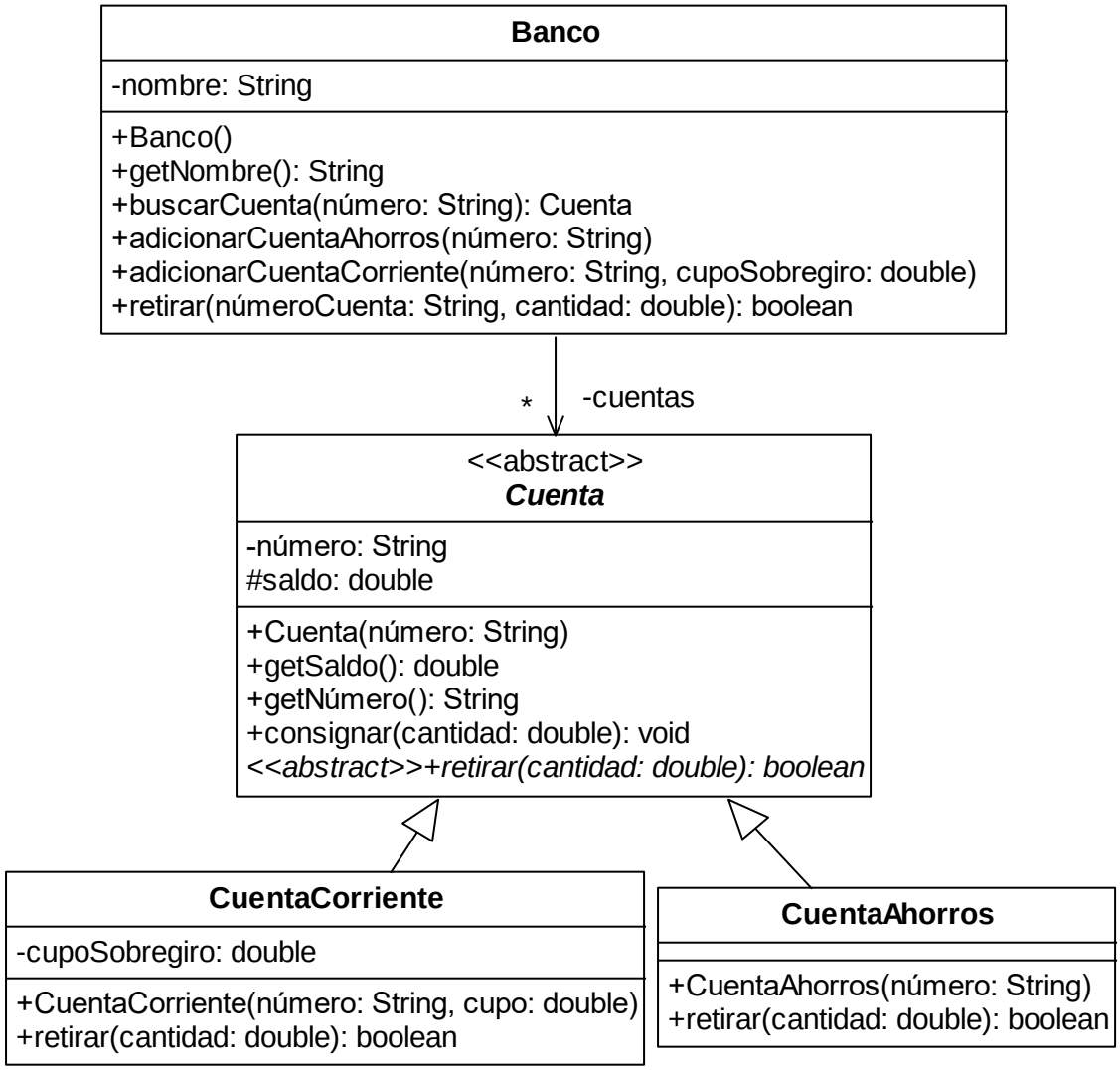
Clase abstracta

método abstracto

Las clases hijas están obligadas a sobrescribir los métodos abstractos que heredan. En caso de que no tengan el método, sale un error de compilación.

Las clases abstractas permiten que las clases hijas realicen de forma diferente las operaciones, pero sin que esto afecte el esquema general de trabajo con ellas. Por ejemplo, el banco MuchoDinero no necesita tener diferentes colecciones para las cuentas corrientes y las cuentas de ahorro, sino que con solo una colección de cuentas puede manejarlas todas.
El único momento donde se deben diferenciar las cuentas es cuando se crean, por ese motivo se tendrá un método para adicionar cuentas corrientes y otro para cuentas de ahorro, pero los demás métodos manejan las cuentas de manera uniforme.

El diagrama de clases queda:



El código de la clase Banco es:

```
import java.util.ArrayList;

/**
 * Entidad bancaria que maneja varias cuentas con dinero
 * @version 2.5
 */
public class Banco {
	private String nombre;
	private ArrayList<Cuenta> cuentas;

	public Banco () {
		this.nombre = "MuchoDinero";
		this.cuentas = new ArrayList<>();
	}

	public String getNombre() {
		return nombre;
	}

	/**
	 * Buscar una cuenta en la lista, a partir del número.
	 * @param numero  el número que identifica la cuenta que se buscará
	 * @return  el objeto Cuenta que corresponde al número dado,
	 *          o null si no se encuentra.
	 */
	public Cuenta buscarCuenta(String numero) {
		for (Cuenta cuentaComparar : cuentas) {
			if (cuentaComparar.getNumero().equals(numero)) {
				return cuentaComparar;
			}
		}
		return null;
	}

	/**
	 * Se crea una nueva cuenta de ahorros y se adiciona a la lista.
	 * @param numero  el número de la nueva cuenta – debe ser único
	 * @return  indicación de si se pudo crear y adicionar la cuenta,
	 *          y false en caso contrario
	 */
	public boolean adicionarCuentaAhorros(String numero)
	{
```

//continúa

```
            Cuenta cuentaExistente = buscarCuenta(numero);
            if (cuentaExistente == null) {
                  // Se crea la cuenta de ahorros
                  Cuenta nuevaCuenta = new CuentaAhorros(numero);
                  return cuentas.add(nuevaCuenta);
            }
            return false;
      }

      /**
       * Se crea una nueva cuenta corriente y se adiciona a la lista.
       * @param numero  el número de la nueva cuenta – debe ser único
       * @param cupoSobregiro el cupo de retiro más allá del saldo
       * @return  true si se pudo crear y adicionar la cuenta,
       *          y false en caso contrario
       */
      public boolean adicionarCuentaCorriente(String numero,
                  double cupoSobregiro)
      {
            Cuenta cuentaExistente = buscarCuenta(numero);
            if (cuentaExistente == null) {
                  // Se crea la cuenta corriente
                  Cuenta nuevaCuenta =
                              new CuentaCorriente(numero,cupoSobregiro);
                  return cuentas.add(nuevaCuenta);
            }
            return false;
      }

      /**
       * Busca una cuenta en el banco y retira dinero de ella
       * @param numeroCuenta  el número que identifica la cuenta
       * @param cantidad      cantidad, en pesos, que desea retirar
       * @return true si pudo retirar, y false si no pudo
       *           retirar o si no encontró la cuenta.
       */
      public boolean retirar(String numeroCuenta, double cantidad) {
            Cuenta cuenta = buscarCuenta(numeroCuenta);
            if (cuenta != null) {
                  return cuenta.retirar(cantidad);
            }
            return false;
      }
} // fin clase Banco
```

Se llama al método “retirar”, sin importar si es cuenta corriente o de ahorros

Métodos y clases final

Las clases abstractas son usadas para comenzar una jerarquía de clases, es decir, se espera que tengan clases hijas. Lo opuesto de esto es definir una clase que NO pueda tener clases hijas. Sin embargo, es muy raramente usado, porque limita la flexibilidad que deben tener los programas orientados a objetos.

En Java, para indicar que una clase no puede tener hijas, se usa la palabra reservada *final*.
Por ejemplo:

```
 /**
 * Herramienta usada para cortar
 * @version 1.0
 */
public final class Tijera
{
      /**
       *  Cortar o sacar trozos de un papel con la tijera
       */
      public void cortarPapel()
      {
            System.out.println("Cortando un papel");
      }
}
```

No se pueden tener clases hijas de Tijera

También es posible tener métodos que NO pueden ser sobrescritos, lo cual se define en Java con la misma palabra reservada *final*. Por ejemplo:

```
/**
 * Método que muestra un mensaje por consola
 * @param mensaje la cadena de texto que se mostrará
 */
public final void mostrarMensaje(String mensaje)
{
      System.out.println(mensaje);
}
```

No se puede sobrescribir este método