Programación II

07. Conjuntos – implementación estática

Ricardo Wehbe

UADE

29 de enero de 2021

Programa

1. Repaso de tipos de datos
2. Tipos de datos abstractos
   - Abstracciones
3. Especificación e implementación
   - Especificación
   - Implementación estática

¿Por dónde vamos?

Tipos abstractos de datos (TDAs o ADTs)

Tipos abstractos de datos (TDAs o ADTs)

- ¿Qué son los *tipos abstractos de datos*?

Tipos abstractos de datos (TDAs o ADTs)

- ¿Qué son los *tipos abstractos de datos*?
- ¿Qué son los *tipos de datos*?

Tipos abstractos de datos (TDAs o ADTs)

- ¿Qué son los *tipos abstractos de datos*?
- ¿Qué son los *tipos de datos*?
- Algunos ejemplos en Java: `boolean` (1 bit), `int` (32 bits), `short` (16 bits), `char` (16 bits), `byte` (8 bits), `long` (32 bits), `float` (32 bits), `double` (64 bits).

Tipos abstractos de datos (TDAs o ADTs)

- ¿Qué son los *tipos abstractos de datos*?
- ¿Qué son los *tipos de datos*?
- Algunos ejemplos en Java: `boolean` (1 bit), `int` (32 bits), `short` (16 bits), `char` (16 bits), `byte` (8 bits), `long` (32 bits), `float` (32 bits), `double` (64 bits).
- El tipo de datos define un conjunto de valores admisibles (restringe el valor que una expresión puede tomar) y y define las operaciones que pueden efectuarse sobre datos de ese tipo.

Tipos abstractos de datos (TDAs o ADTs)

- ¿Qué son los *tipos abstractos de datos*?
- ¿Qué son los *tipos de datos*?
- Algunos ejemplos en Java: `boolean` (1 bit), `int` (32 bits), `short` (16 bits), `char` (16 bits), `byte` (8 bits), `long` (32 bits), `float` (32 bits), `double` (64 bits).
- El tipo de datos define un conjunto de valores admisibles (restringe el valor que una expresión puede tomar) y y define las operaciones que pueden efectuarse sobre datos de ese tipo.
- Un tipo abstracto de datos es un modelo matemático en el que un tipo de datos es definido por su comportamiento (semántica) desde el punto de vista del usuario: deben definirse los valores que puede tomar, las operaciones que pueden efectuarse sobre esos valores y el resultado de esas operaciones.

Tipos abstractos de datos (TDAs o ADTs)

- ¿Qué son los *tipos abstractos de datos*?
- ¿Qué son los *tipos de datos*?
- Algunos ejemplos en Java: `boolean` (1 bit), `int` (32 bits), `short` (16 bits), `char` (16 bits), `byte` (8 bits), `long` (32 bits), `float` (32 bits), `double` (64 bits).
- El tipo de datos define un conjunto de valores admisibles (restringe el valor que una expresión puede tomar) y y define las operaciones que pueden efectuarse sobre datos de ese tipo.
- Un tipo abstracto de datos es un modelo matemático en el que un tipo de datos es definido por su comportamiento (semántica) desde el punto de vista del usuario: deben definirse los valores que puede tomar, las operaciones que pueden efectuarse sobre esos valores y el resultado de esas operaciones.
- La *abstracción* es sobre los detalles de la implementación, que permanencen ocultos al usuario.

Tipos de datos primitivos y no primitivos

Tipos de datos primitivos y no primitivos

- Un *tipo de datos simple* es un tipo de datos atómico (que no puede ser subdividido.) Por ejemplo, `boolean`, `int`, `char`.

Tipos de datos primitivos y no primitivos

- Un *tipo de datos simple* es un tipo de datos atómico (que no puede ser subdividido.) Por ejemplo, `boolean`, `int`, `char`.
- Una *estructura de datos* es una organización de un conjunto de valores. Por ejemplo, un arreglo.

Tipos de datos primitivos y no primitivos

- Un *tipo de datos simple* es un tipo de datos atómico (que no puede ser subdividido.) Por ejemplo, `boolean`, `int`, `char`.
- Una *estructura de datos* es una organización de un conjunto de valores. Por ejemplo, un arreglo.
- Un *tipo de datos abstracto* (TDA) es una estructura de datos donde la abstracción se refiere a que nos concentramos en el comportamiento del TDA (qué hace) y no en cómo lo hace.

Tipos de datos primitivos y no primitivos

- Un *tipo de datos simple* es un tipo de datos atómico (que no puede ser subdividido.) Por ejemplo, `boolean`, `int`, `char`.
- Una *estructura de datos* es una organización de un conjunto de valores. Por ejemplo, un arreglo.
- Un *tipo de datos abstracto* (TDA) es una estructura de datos donde la abstracción se refiere a que nos concentramos en el comportamiento del TDA (qué hace) y no en cómo lo hace.
- Separamos entonces el comportamiento del TDA (la *especificación*) de cómo se consigue ese comportamiento (la *implementación*).

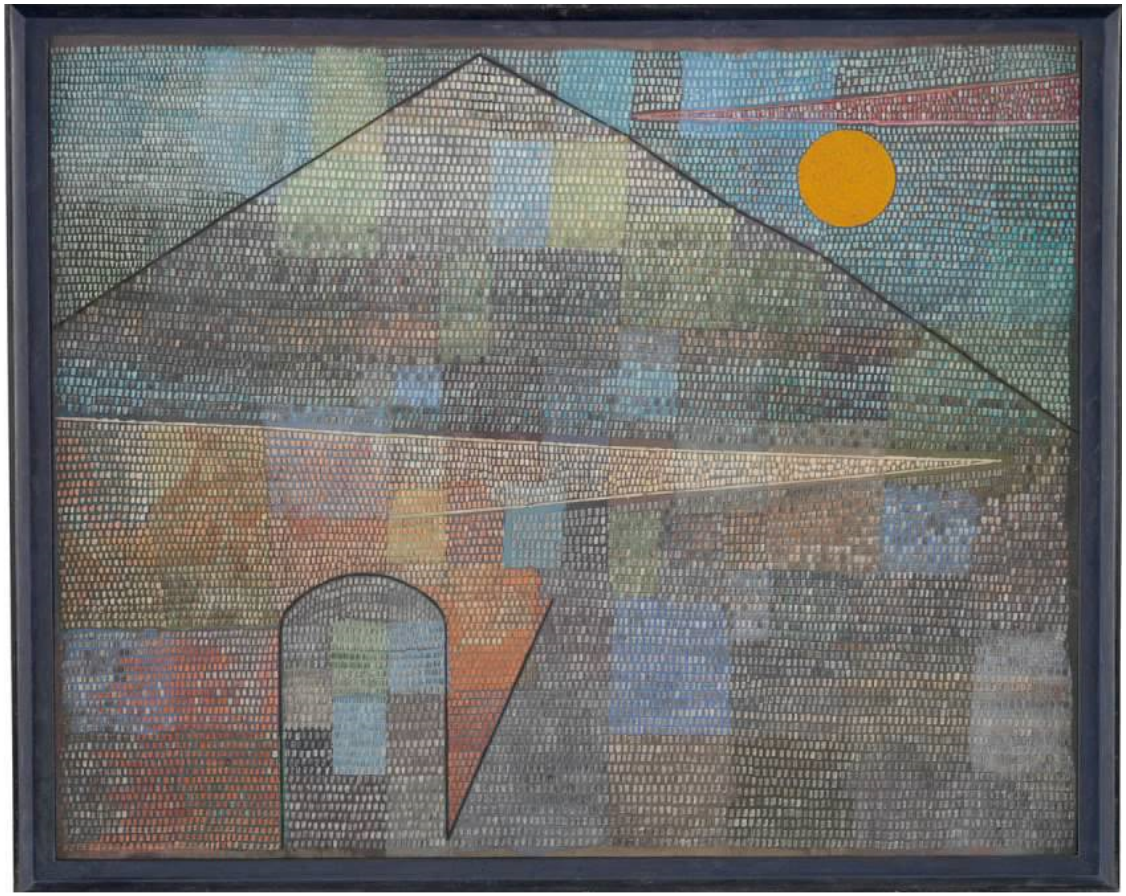
Tipos de datos primitivos y no primitivos

- Un *tipo de datos simple* es un tipo de datos atómico (que no puede ser subdividido.) Por ejemplo, `boolean`, `int`, `char`.
- Una *estructura de datos* es una organización de un conjunto de valores. Por ejemplo, un arreglo.
- Un *tipo de datos abstracto* (TDA) es una estructura de datos donde la abstracción se refiere a que nos concentramos en el comportamiento del TDA (qué hace) y no en cómo lo hace.
- Separamos entonces el comportamiento del TDA (la *especificación*) de cómo se consigue ese comportamiento (la *implementación*).
- Un mismo TDA puede entonces tener diferentes implementaciones.

¿Por dónde vamos?

¿Por dónde vamos?

Abstracciones



Paul Klee, *Ad Parnassum*, 1932

Abstracciones

Abstracciones

- Un objeto es una abstracción. Con esto queremos decir que los detalles de cómo funciona el objeto son irrelevantes para el usuario del objeto.

Abstracciones

- Un objeto es una abstracción. Con esto queremos decir que los detalles de cómo funciona el objeto son irrelevantes para el usuario del objeto.
- Al usuario le interesa *qué* hace el objeto, no *cómo* lo hace.

Abstracciones

- Un objeto es una abstracción. Con esto queremos decir que los detalles de cómo funciona el objeto son irrelevantes para el usuario del objeto.
- Al usuario le interesa *qué* hace el objeto, no *cómo* lo hace.
- Utilizamos abstracciones todo el tiempo. Son necesarias para la vida cotidiana. Cuando usamos un auto nos abstraemos de su funcionamiento concreto: nos limitamos a su *interfaz* (pedales, volante.)

Abstracciones

- Un objeto es una abstracción. Con esto queremos decir que los detalles de cómo funciona el objeto son irrelevantes para el usuario del objeto.
- Al usuario le interesa *qué* hace el objeto, no *cómo* lo hace.
- Utilizamos abstracciones todo el tiempo. Son necesarias para la vida cotidiana. Cuando usamos un auto nos abstraemos de su funcionamiento concreto: nos limitamos a su *interfaz* (pedales, volante.)
- De la misma manera, podemos organizar mentalmente una cantidad de información agrupándola según algún criterio e ignorando detalles irrelevantes. Es decir, *abstrayéndonos* de ellos.

Abstracciones

- Un objeto es una abstracción. Con esto queremos decir que los detalles de cómo funciona el objeto son irrelevantes para el usuario del objeto.
- Al usuario le interesa *qué* hace el objeto, no *cómo* lo hace.
- Utilizamos abstracciones todo el tiempo. Son necesarias para la vida cotidiana. Cuando usamos un auto nos abstraemos de su funcionamiento concreto: nos limitamos a su *interfaz* (pedales, volante.)
- De la misma manera, podemos organizar mentalmente una cantidad de información agrupándola según algún criterio e ignorando detalles irrelevantes. Es decir, *abstrayéndonos* de ellos.
- Desde este punto de vista un objeto es una abstracción muy útil, que nos permite agrupar la información en un bloque.

Niveles de abstracción

Niveles de abstracción

- Existen diferentes niveles de abstracción dependiendo de qué nivel de detalle nos interese.

Niveles de abstracción

- Existen diferentes niveles de abstracción dependiendo de qué nivel de detalle nos interese.
- En un nivel alto, nos interesa manejar el auto y no necesitamos los detalles del funcionamiento del motor. En un nivel bajo (el de un mecánico o un ingeniero) es necesario conocer perfectamente estos detalles.

Niveles de abstracción

- Existen diferentes niveles de abstracción dependiendo de qué nivel de detalle nos interese.
- En un nivel alto, nos interesa manejar el auto y no necesitamos los detalles del funcionamiento del motor. En un nivel bajo (el de un mecánico o un ingeniero) es necesario conocer perfectamente estos detalles.
- De una manera análoga, los detalles ocultos de un objeto deben estar implementados. El nivel de abstracción del implementador del objeto es por lo tanto más bajo que el nivel de abstracción del usuario.

Niveles de abstracción

- Existen diferentes niveles de abstracción dependiendo de qué nivel de detalle nos interese.
- En un nivel alto, nos interesa manejar el auto y no necesitamos los detalles del funcionamiento del motor. En un nivel bajo (el de un mecánico o un ingeniero) es necesario conocer perfectamente estos detalles.
- De una manera análoga, los detalles ocultos de un objeto deben estar implementados. El nivel de abstracción del implementador del objeto es por lo tanto más bajo que el nivel de abstracción del usuario.
- Trabajaremos con todos estos niveles de abstracción durante este curso.

Tipos de datos abstractos (TDAs)

Tipos de datos abstractos (TDAs)

- Dentro de la filosofía de los objetos, los TDAs son un elemento clave.

Tipos de datos abstractos (TDAs)

- Dentro de la filosofía de los objetos, los TDAs son un elemento clave.
- Como dijimos anteriormente, es más fácil tratar con una abstracción que tener en cuenta una multiplicidad de detalles que pueden ser en su mayoría superfluos para nuestro objetivos.

Tipos de datos abstractos (TDAs)

- Dentro de la filosofía de los objetos, los TDAs son un elemento clave.
- Como dijimos anteriormente, es más fácil tratar con una abstracción que tener en cuenta una multiplicidad de detalles que pueden ser en su mayoría superfluos para nuestro objetivos.
- Un TDA se define a través de su semántica (comportamiento.) Los detalles de la implementación quedan ocultos al usuario.

Tipos de datos abstractos (TDAs)

- Dentro de la filosofía de los objetos, los TDAs son un elemento clave.
- Como dijimos anteriormente, es más fácil tratar con una abstracción que tener en cuenta una multiplicidad de detalles que pueden ser en su mayoría superfluos para nuestro objetivos.
- Un TDA se define a través de su semántica (comportamiento.) Los detalles de la implementación quedan ocultos al usuario.
- El TDA se define a través de una *interfaz*, que es una clase abstracta que contiene la especificación de los métodos que ofrece el TDA.

Tipos de datos abstractos (TDAs)

- Dentro de la filosofía de los objetos, los TDAs son un elemento clave.
- Como dijimos anteriormente, es más fácil tratar con una abstracción que tener en cuenta una multiplicidad de detalles que pueden ser en su mayoría superfluos para nuestro objetivos.
- Un TDA se define a través de su semántica (comportamiento.) Los detalles de la implementación quedan ocultos al usuario.
- El TDA se define a través de una *interfaz*, que es una clase abstracta que contiene la especificación de los métodos que ofrece el TDA.
- Hay otra clase que hereda los métodos de la interfaz y los implementa. Esta clase podría suplantarse por otra implementación diferente de manera que el usuario no podría percibirlo.

Estado y comportamiento de un TDA

Estado y comportamiento de un TDA

- Un TDA tiene un *estado* que está dado por los valores de sus parámetros (por ejemplo, el monto del saldo en la cuenta bancaria.)

Estado y comportamiento de un TDA

- Un TDA tiene un *estado* que está dado por los valores de sus parámetros (por ejemplo, el monto del saldo en la cuenta bancaria.)
- Un TDA tiene un *comportamiento* que está definido por los métodos que ofrece (por ejemplo, la posibilidad de efectuar retiros de una cuenta o de deposita dinero en ella.)

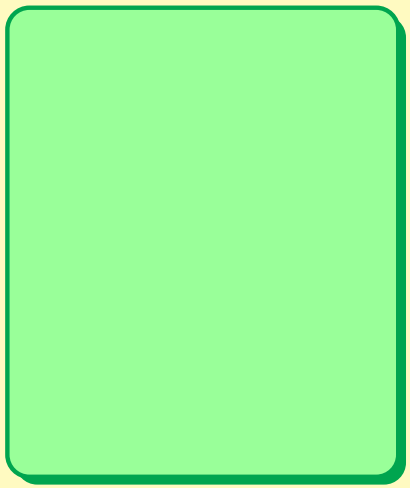
Estado y comportamiento de un TDA

- Un TDA tiene un *estado* que está dado por los valores de sus parámetros (por ejemplo, el monto del saldo en la cuenta bancaria.)
- Un TDA tiene un *comportamiento* que está definido por los métodos que ofrece (por ejemplo, la posibilidad de efectuar retiros de una cuenta o de deposita dinero en ella.)
- Muchas veces se refiere uno al comportamiento de un TDA como su *semántica*.

Estado y comportamiento de un TDA

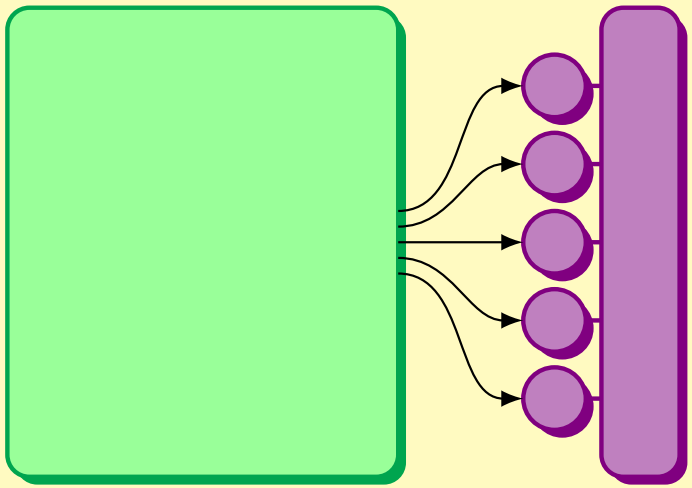
- Un TDA tiene un *estado* que está dado por los valores de sus parámetros (por ejemplo, el monto del saldo en la cuenta bancaria.)
- Un TDA tiene un *comportamiento* que está definido por los métodos que ofrece (por ejemplo, la posibilidad de efectuar retiros de una cuenta o de deposita dinero en ella.)
- Muchas veces se refiere uno al comportamiento de un TDA como su *semántica*.
- Una implementación es *correcta* si el comportamiento del TDA es el esperado para cualquier estado posible.

Esquema de las vistas de un TDA

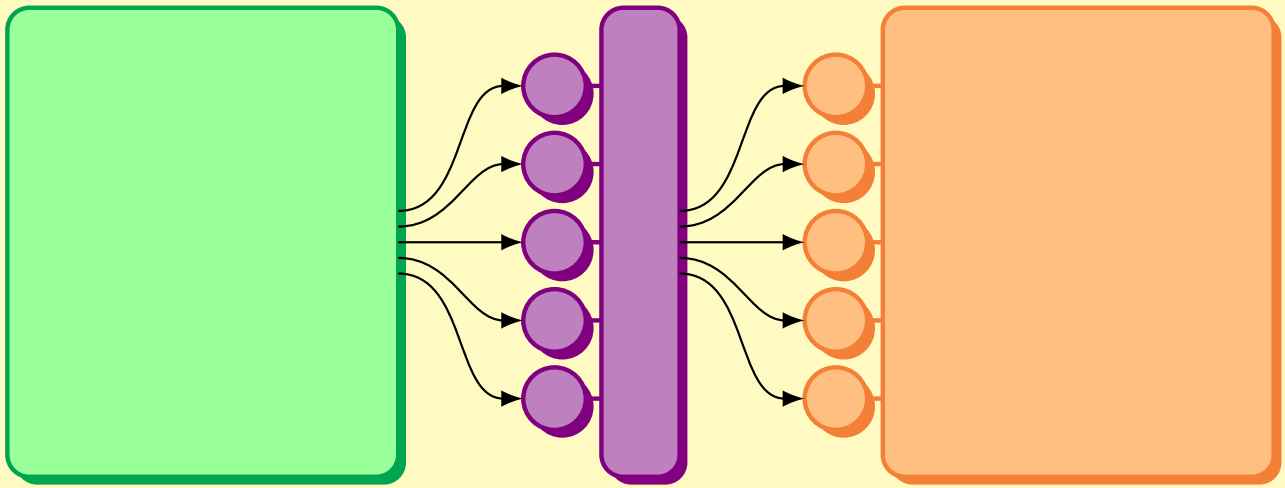


Clase que utiliza el TDA

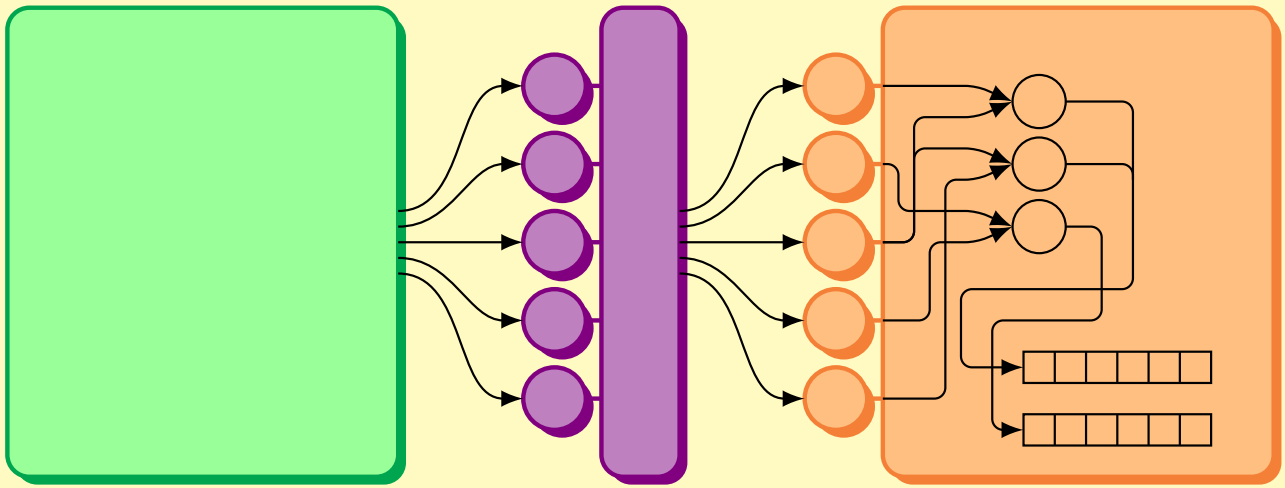
Esquema de las vistas de un TDA



Esquema de las vistas de un TDA



Esquema de las vistas de un TDA

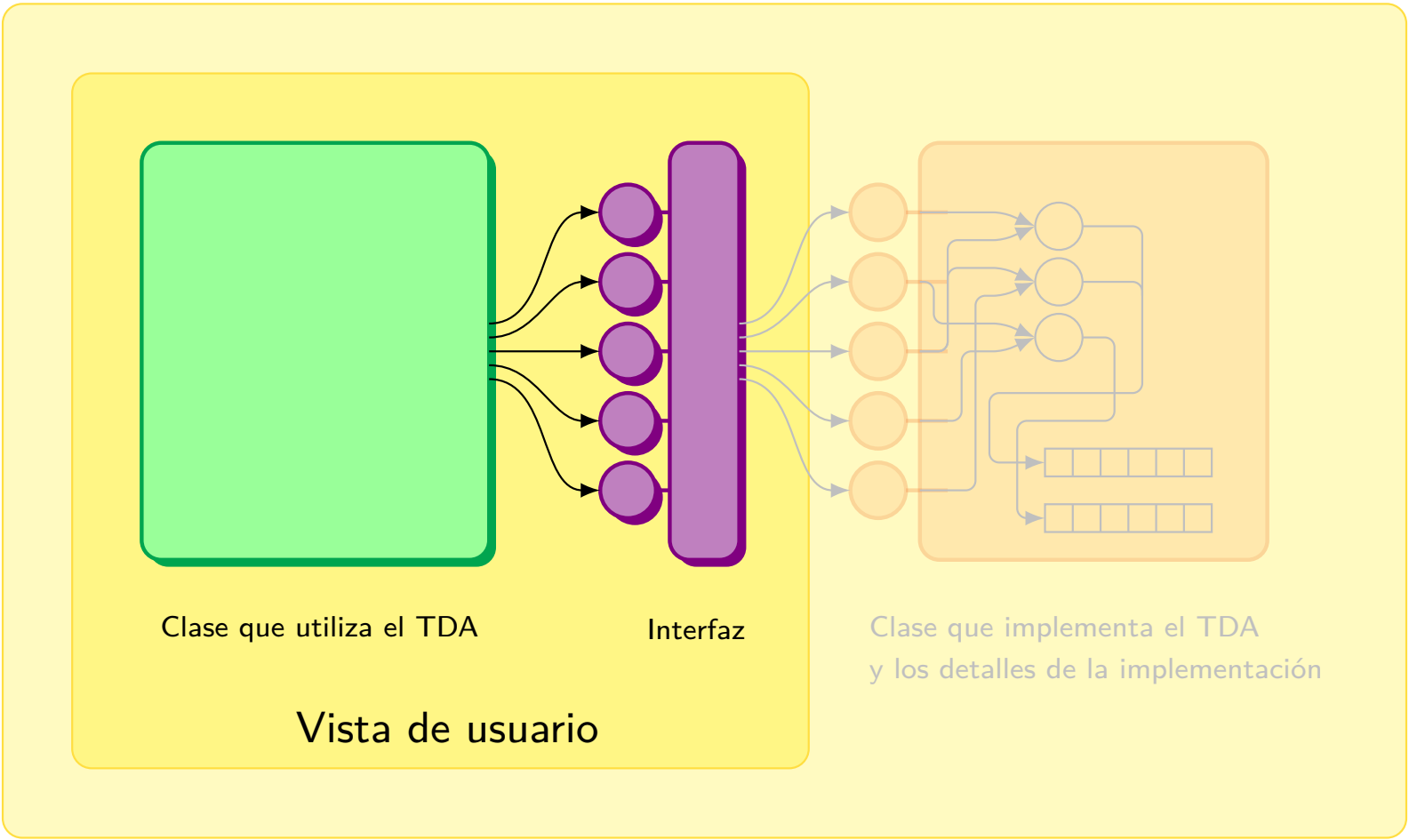


Clase que utiliza el TDA

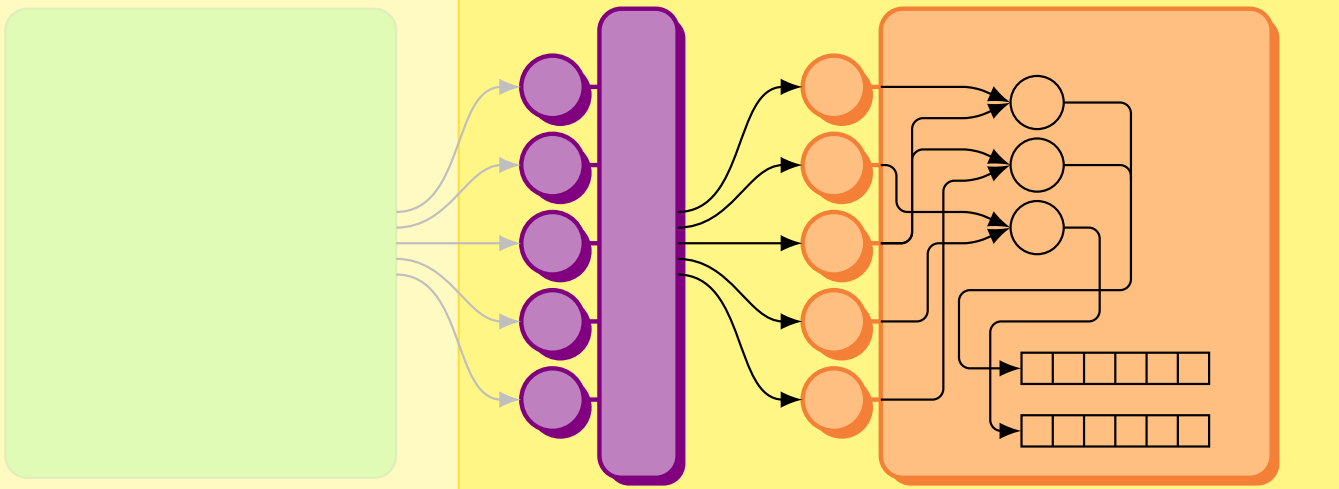
Interfaz

Clase que implementa el TDA
y los detalles de la implementación

Esquema de las vistas de un TDA



Esquema de las vistas de un TDA



Conjuntos

Conjuntos

- Un *conjunto* es una colección de elementos en la que no existen duplicados. No existe ninguna relación posicional concreta entre los elementos de un conjunto (contrariamente a una pila, en la que el elementos más reciente tiene prioridad o una cola, en la que el elemento más antiguo tiene prioridad.)

Conjuntos

- Un *conjunto* es una colección de elementos en la que no existen duplicados. No existe ninguna relación posicional concreta entre los elementos de un conjunto (contrariamente a una pila, en la que el elementos más reciente tiene prioridad o una cola, en la que el elemento más antiguo tiene prioridad.)
- Un conjunto puede verse entonces como una caja o una bolsa de elementos.

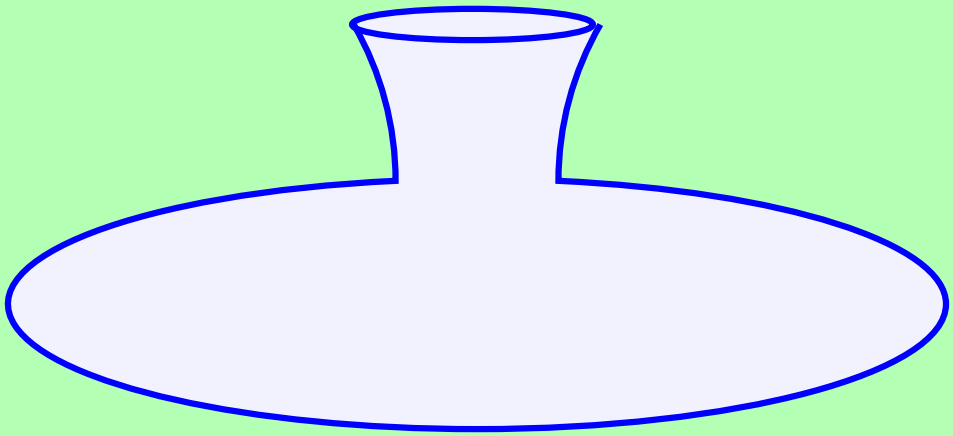
Conjuntos

- Un *conjunto* es una colección de elementos en la que no existen duplicados. No existe ninguna relación posicional concreta entre los elementos de un conjunto (contrariamente a una pila, en la que el elementos más reciente tiene prioridad o una cola, en la que el elemento más antiguo tiene prioridad.)
- Un conjunto puede verse entonces como una caja o una bolsa de elementos.
- Las operaciones que necesitaremos son: agregar y eliminar datos del conjunto (que llamaremos posteriormente *Agregar* y *Sacar*), elegir un elemento arbitrario del conjunto (que llamaremos *Elegir*), consultar si el conjunto está o no vacío (a esta consulta la llamaremos *ConjuntoVacio*) y si un elemento dado pertenece o no al conjunto (que llamaremos *Pertenece*.)

Conjuntos

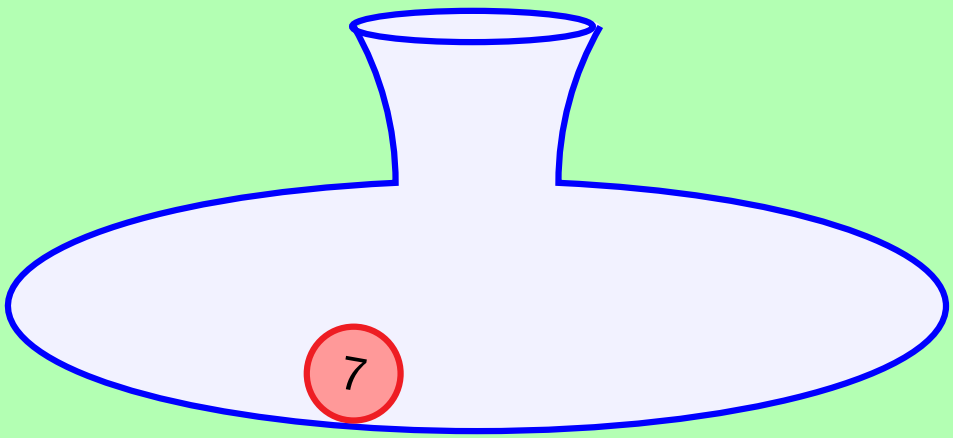
- Un *conjunto* es una colección de elementos en la que no existen duplicados. No existe ninguna relación posicional concreta entre los elementos de un conjunto (contrariamente a una pila, en la que el elementos más reciente tiene prioridad o una cola, en la que el elemento más antiguo tiene prioridad.)
- Un conjunto puede verse entonces como una caja o una bolsa de elementos.
- Las operaciones que necesitaremos son: agregar y eliminar datos del conjunto (que llamaremos posteriormente *Agregar* y *Sacar*), elegir un elemento arbitrario del conjunto (que llamaremos *Elegir*), consultar si el conjunto está o no vacío (a esta consulta la llamaremos *ConjuntoVacio*) y si un elemento dado pertenece o no al conjunto (que llamaremos *Pertenece*.)
- A estas operaciones agregaremos la inicialización de un conjunto, que llamaremos *InicializarConjunto*.

Semántica de *ConjuntoTDA*



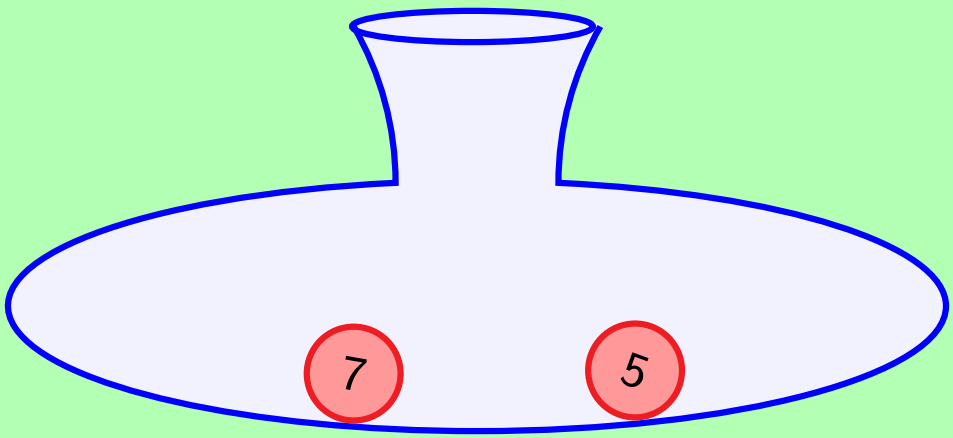
```
ConjuntoVacio() = true
Pertenece(4) = false
Pertenece(5) = false
```

Semántica de *ConjuntoTDA*



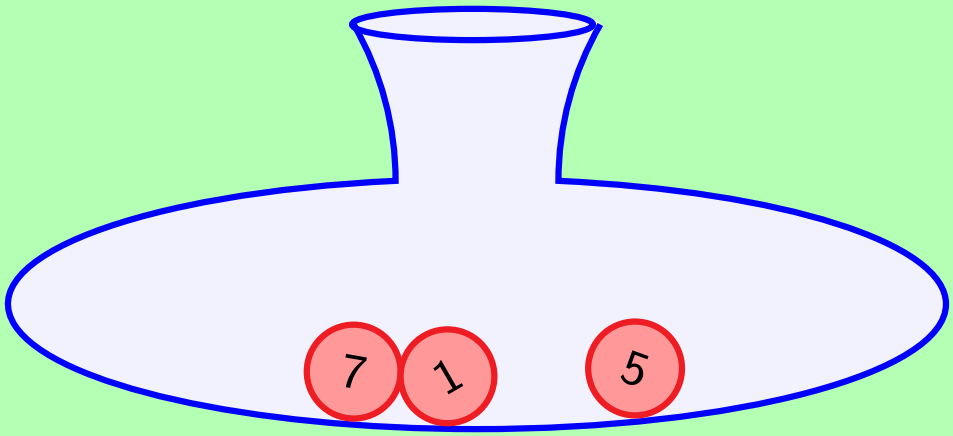

```
Agregar(7)
ConjuntoVacio() = false
Pertenece(4) = false
Pertenece(5) = false
```

Semántica de *ConjuntoTDA*



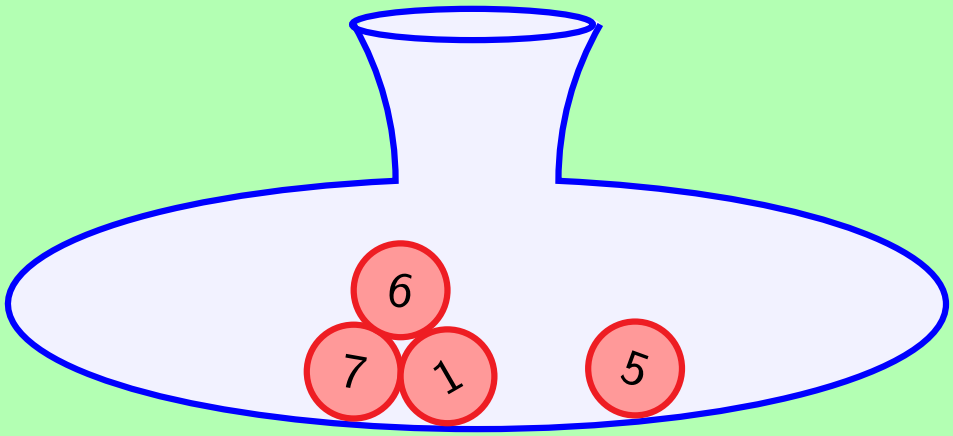

```
Agregar(5)
ConjuntoVacio() = false
Pertenece(4) = false
Pertenece(5) = true
```

Semántica de *ConjuntoTDA*



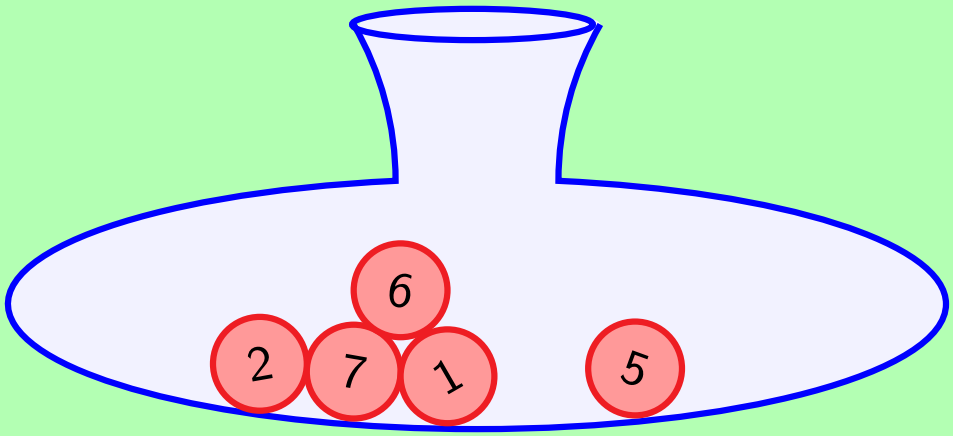

```
Agregar(1)
ConjuntoVacio() = false
Pertenece(4) = false
Pertenece(5) = true
```

Semántica de *ConjuntoTDA*



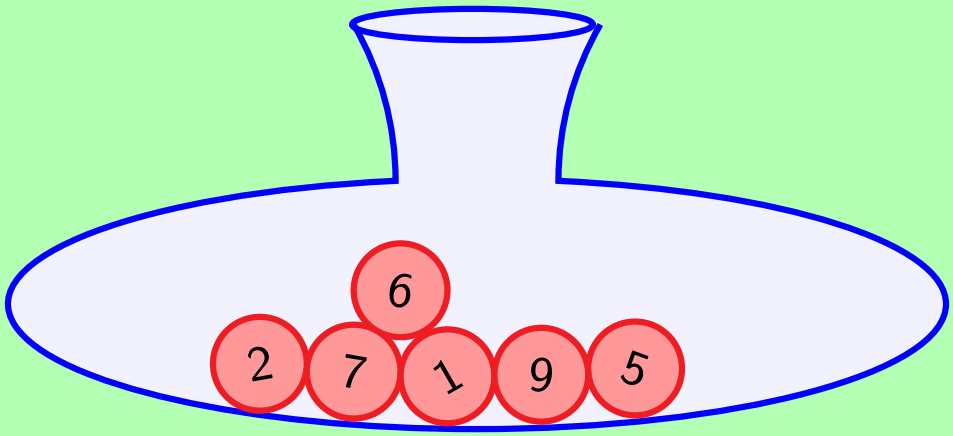

```
Agregar(6)
ConjuntoVacio() = false
Pertenece(4) = false
Pertenece(5) = true
```

Semántica de *ConjuntoTDA*



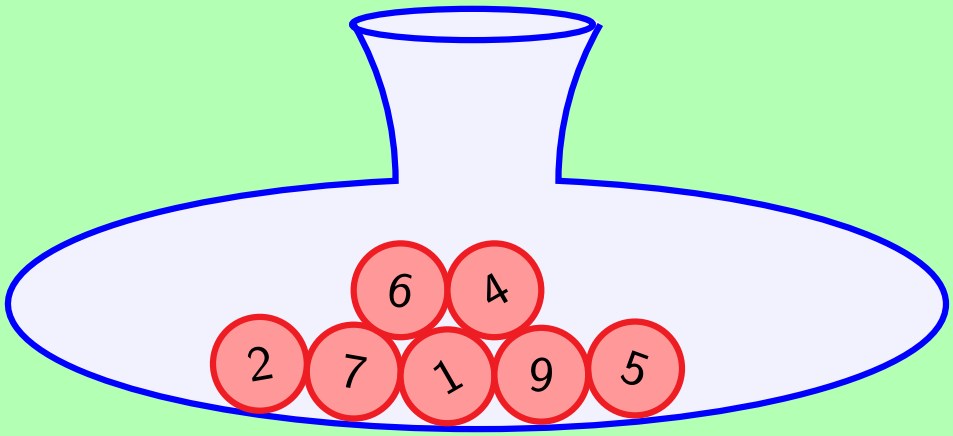

```
Agregar(2)
ConjuntoVacio() = false
Pertenece(4) = false
Pertenece(5) = true
```

Semántica de *ConjuntoTDA*



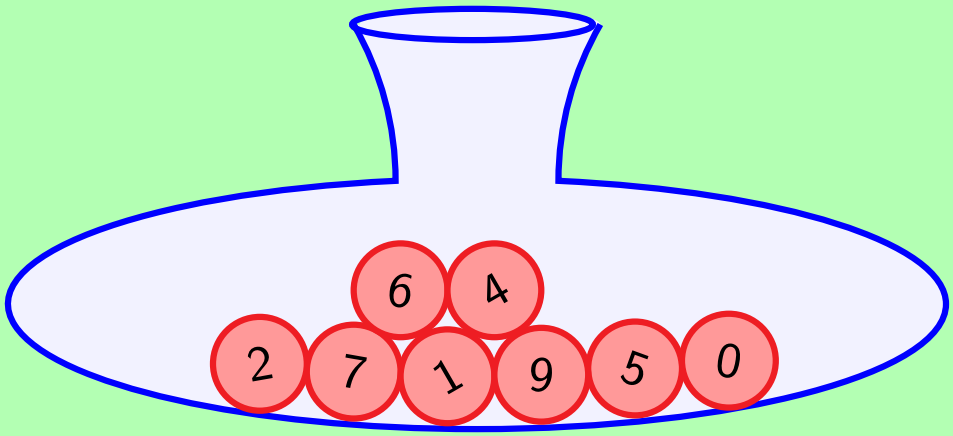

```
Agregar(9)
ConjuntoVacio() = false
Pertenece(4) = false
Pertenece(5) = true
```

Semántica de *ConjuntoTDA*



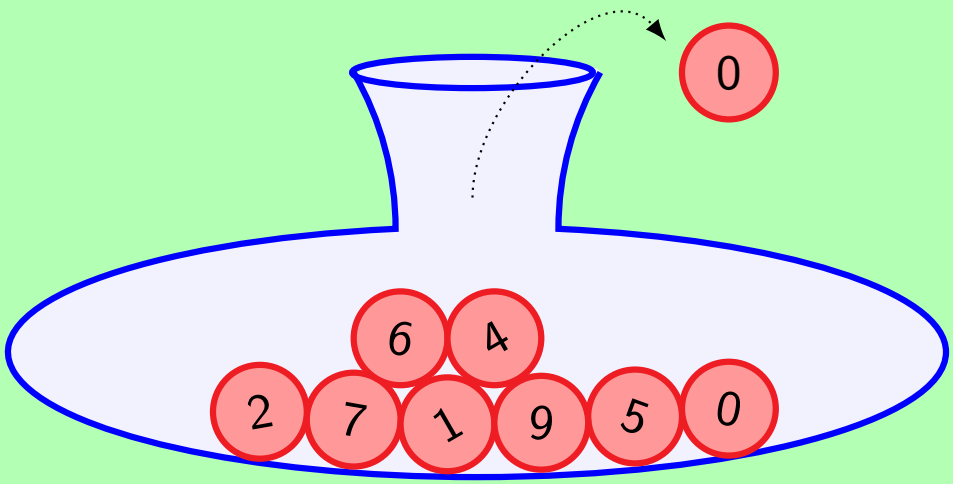

```
Agregar(4)
ConjuntoVacio() = false
Pertenece(4) = true
Pertenece(5) = true
```

Semántica de *ConjuntoTDA*



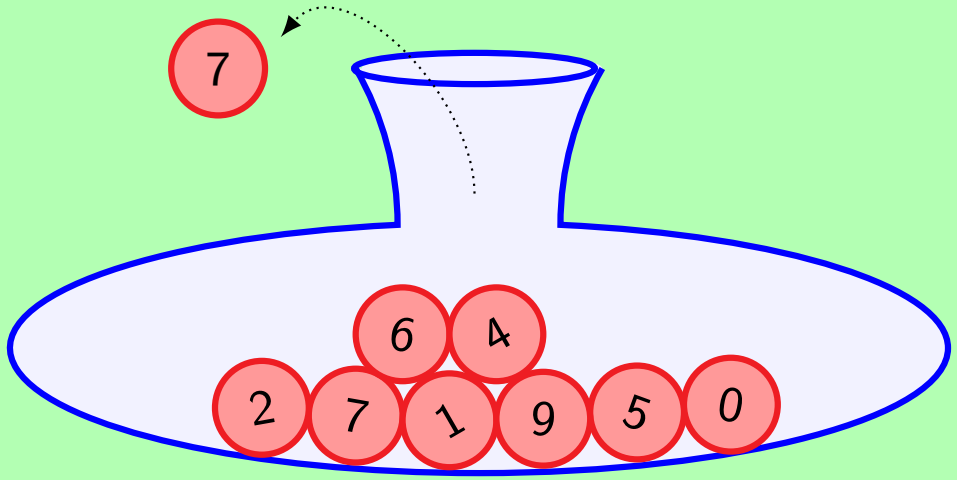

```
Agregar(0)
ConjuntoVacio() = false
Pertenece(4) = true
Pertenece(5) = true
```

Semántica de *ConjuntoTDA*



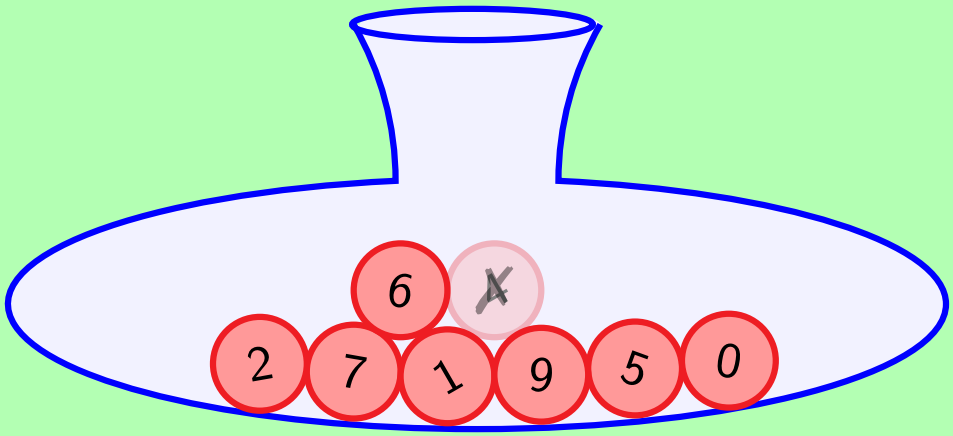

```
Elegir()
ConjuntoVacio() = false
Pertenece(4) = true
Pertenece(5) = true
```

Semántica de *ConjuntoTDA*



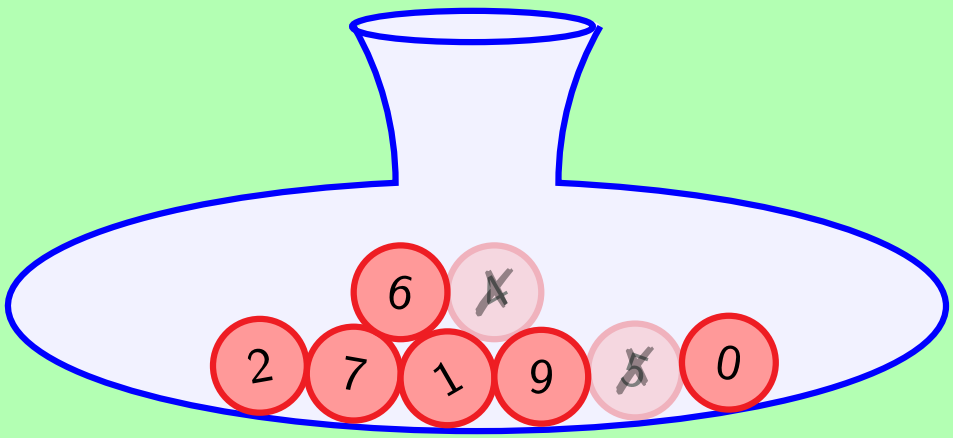

```
Elegir()
ConjuntoVacio() = false
Pertenece(4) = true
Pertenece(5) = true
```

Semántica de *ConjuntoTDA*




```
Eliminar(4)
ConjuntoVacio() = false
Pertenece(4) = false
Pertenece(5) = true
```

Semántica de *ConjuntoTDA*




```
Eliminar(5)
ConjuntoVacio() = false
Pertenece(4) = false
Pertenece(5) = false
```

¿Por dónde vamos?

1. Repaso de tipos de datos
2. Tipos de datos abstractos
   - Abstracciones
3. Especificación e implementación
   - Especificación
   - Implementación estática

Especificaciones e interfaces

Especificaciones e interfaces

- La semántica de un tipo abstracto de datos está definida por una *interfaz*, que es una clase abstracta.

Especificaciones e interfaces

- La semántica de un tipo abstracto de datos está definida por una *interfaz*, que es una clase abstracta.
- La interfaz es una colección de métodos abstractos, en los que sólo se indica su tipo (qué devuelven, si es que devuelven algo) y sus parámetros de entrada.

Especificaciones e interfaces

- La semántica de un tipo abstracto de datos está definida por una *interfaz*, que es una clase abstracta.
- La interfaz es una colección de métodos abstractos, en los que sólo se indica su tipo (qué devuelven, si es que devuelven algo) y sus parámetros de entrada.
- La interfaz está implementada por otra clase que hereda sus métodos.

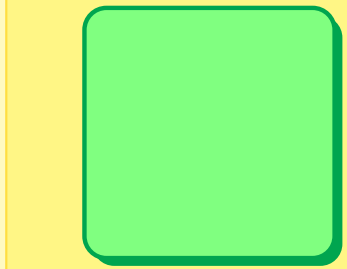
Especificaciones e interfaces

- La semántica de un tipo abstracto de datos está definida por una *interfaz*, que es una clase abstracta.
- La interfaz es una colección de métodos abstractos, en los que sólo se indica su tipo (qué devuelven, si es que devuelven algo) y sus parámetros de entrada.
- La interfaz está implementada por otra clase que hereda sus métodos.
- Para el usuario no es necesario conocer los detalles de la implementación. Basta con conocer la semántica de los métodos de la interfaz.

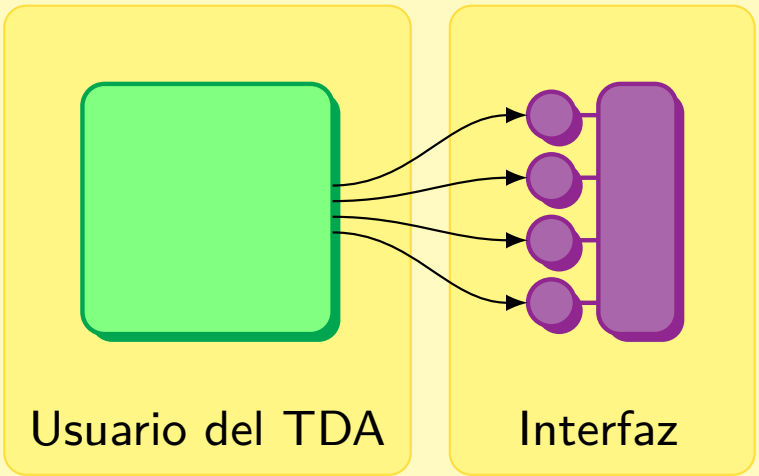
Especificaciones e interfaces

- La semántica de un tipo abstracto de datos está definida por una *interfaz*, que es una clase abstracta.
- La interfaz es una colección de métodos abstractos, en los que sólo se indica su tipo (qué devuelven, si es que devuelven algo) y sus parámetros de entrada.
- La interfaz está implementada por otra clase que hereda sus métodos.
- Para el usuario no es necesario conocer los detalles de la implementación. Basta con conocer la semántica de los métodos de la interfaz.
- Una implementación podría entonces intercambiarse por otra sin que el usuario pudiera percibir la diferencia desde el punto de vista de la semántica (aunque eventualmente sí desde el punto de vista de la eficiencia.)

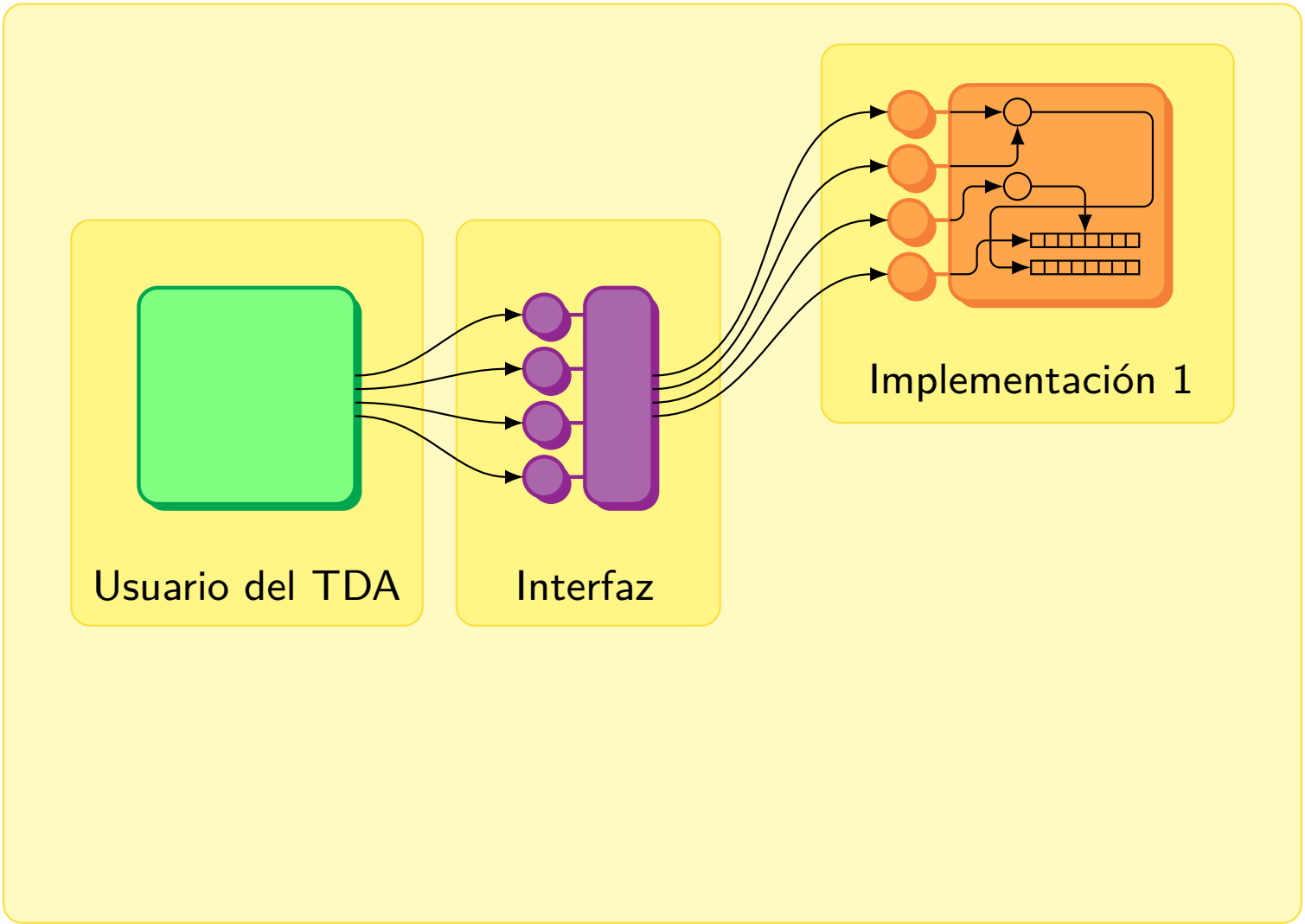
Especificaciones e implementaciones de un TDA



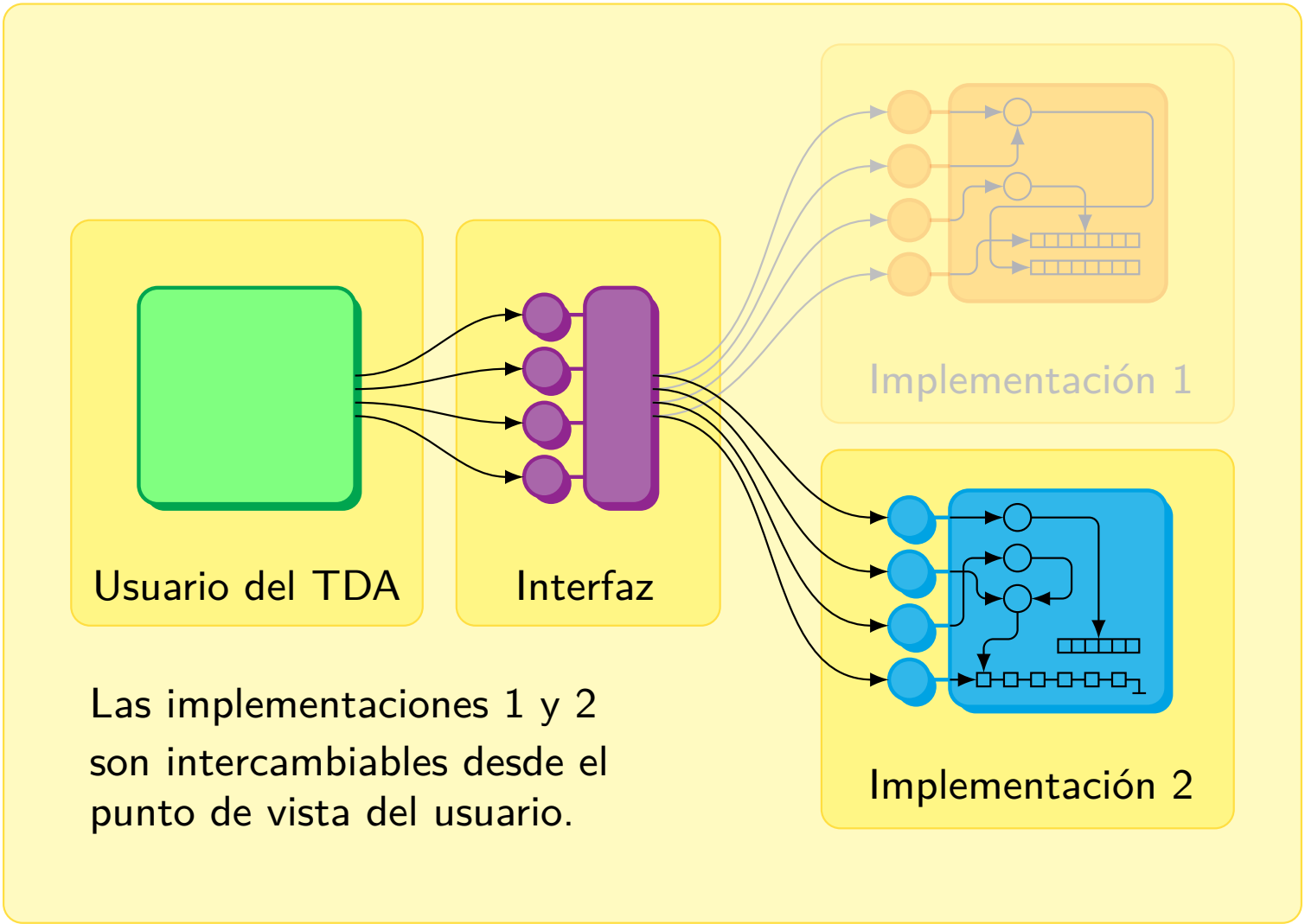
Especificaciones e implementaciones de un TDA



Especificaciones e implementaciones de un TDA

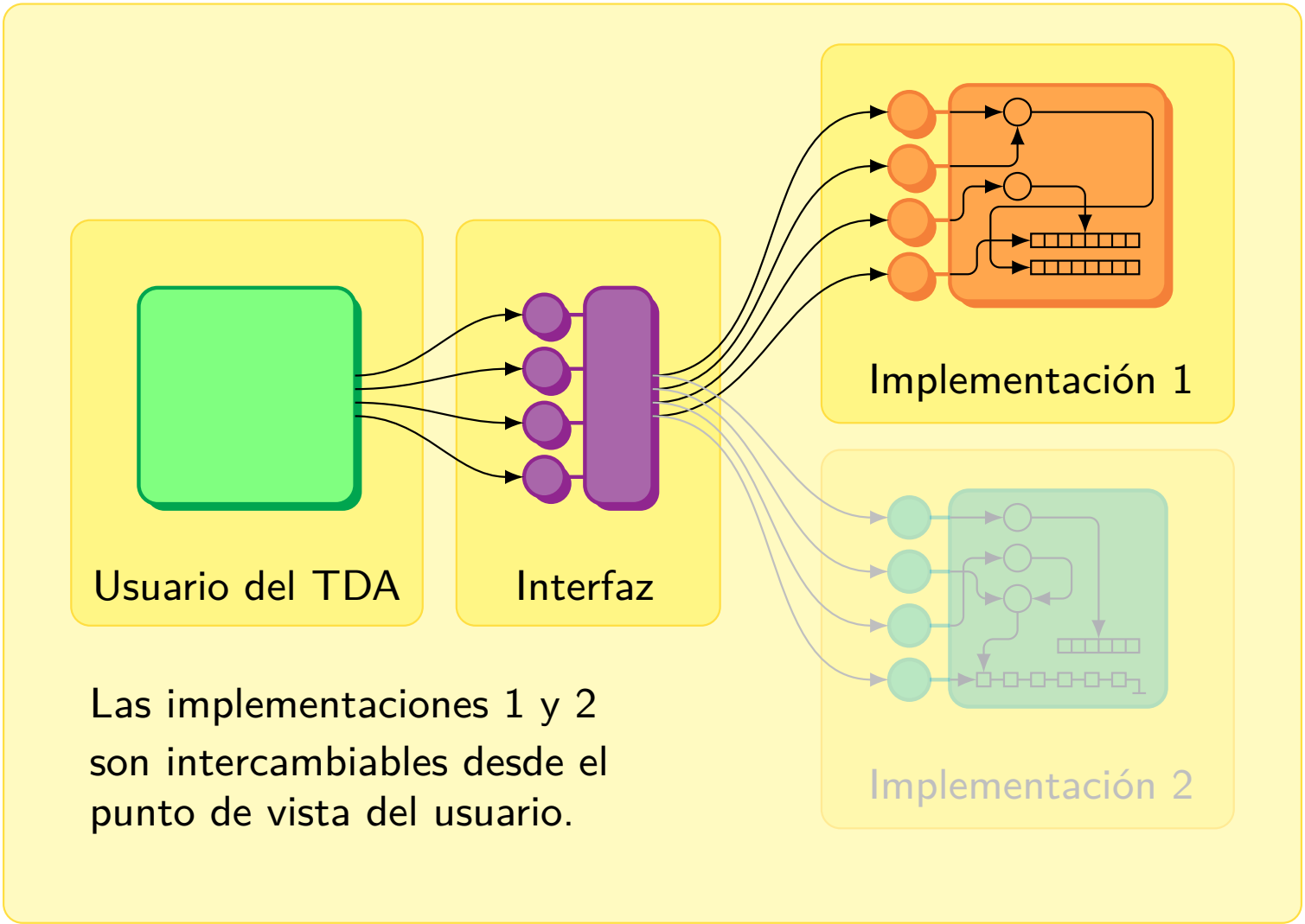


Especificaciones e implementaciones de un TDA



Las implementaciones 1 y 2 son intercambiables desde el punto de vista del usuario.

Especificaciones e implementaciones de un TDA



Las implementaciones 1 y 2 son intercambiables desde el punto de vista del usuario.

¿Por dónde vamos?

La especificación de *ConjuntoTDA*

La especificación de *ConjuntoTDA*

- Observación preliminar: puesto que la estructura no tiene un orden, cuando recuperamos un dato, la estructura nos devuelve uno cualquiera de los que pertenecen a ella. Por la misma razón, cuando queremos eliminar una valor debemos indicarle cuál.

La especificación de *ConjuntoTDA*

- Observación preliminar: puesto que la estructura no tiene un orden, cuando recuperamos un dato, la estructura nos devuelve uno cualquiera de los que pertenecen a ella. Por la misma razón, cuando queremos eliminar una valor debemos indicarle cuál.
- Los métodos son los siguientes:

La especificación de *ConjuntoTDA*

- Observación preliminar: puesto que la estructura no tiene un orden, cuando recuperamos un dato, la estructura nos devuelve uno cualquiera de los que pertenecen a ella. Por la misma razón, cuando queremos eliminar una valor debemos indicarle cuál.
- Los métodos son los siguientes:
- *Agregar*, que permite agregar un elemento a un conjunto.

La especificación de *ConjuntoTDA*

- Observación preliminar: puesto que la estructura no tiene un orden, cuando recuperamos un dato, la estructura nos devuelve uno cualquiera de los que pertenecen a ella. Por la misma razón, cuando queremos eliminar una valor debemos indicarle cuál.
- Los métodos son los siguientes:
- *Agregar*, que permite agregar un elemento a un conjunto.
- *Sacar*, que permite eliminar un elemento de un conjunto.

La especificación de *ConjuntoTDA*

- Observación preliminar: puesto que la estructura no tiene un orden, cuando recuperamos un dato, la estructura nos devuelve uno cualquiera de los que pertenecen a ella. Por la misma razón, cuando queremos eliminar una valor debemos indicarle cuál.
- Los métodos son los siguientes:
- *Agregar*, que permite agregar un elemento a un conjunto.
- *Sacar*, que permite eliminar un elemento de un conjunto.
- *Elegir*, que devuelve un elemento arbitrario de un conjunto no vacío.

La especificación de *ConjuntoTDA*

- Observación preliminar: puesto que la estructura no tiene un orden, cuando recuperamos un dato, la estructura nos devuelve uno cualquiera de los que pertenecen a ella. Por la misma razón, cuando queremos eliminar una valor debemos indicarle cuál.
- Los métodos son los siguientes:
- *Agregar*, que permite agregar un elemento a un conjunto.
- *Sacar*, que permite eliminar un elemento de un conjunto.
- *Elegir*, que devuelve un elemento arbitrario de un conjunto no vacío.
- *Pertenece*, que nos dice si un elemento está en un conjunto.

La especificación de *ConjuntoTDA*

- Observación preliminar: puesto que la estructura no tiene un orden, cuando recuperamos un dato, la estructura nos devuelve uno cualquiera de los que pertenecen a ella. Por la misma razón, cuando queremos eliminar una valor debemos indicarle cuál.
- Los métodos son los siguientes:
- *Agregar*, que permite agregar un elemento a un conjunto.
- *Sacar*, que permite eliminar un elemento de un conjunto.
- *Elegir*, que devuelve un elemento arbitrario de un conjunto no vacío.
- *Pertenece*, que nos dice si un elemento está en un conjunto.
- *ConjuntoVacio*, que indica si el conjunto tiene elementos o no.

La especificación de *ConjuntoTDA*

- Observación preliminar: puesto que la estructura no tiene un orden, cuando recuperamos un dato, la estructura nos devuelve uno cualquiera de los que pertenecen a ella. Por la misma razón, cuando queremos eliminar una valor debemos indicarle cuál.
- Los métodos son los siguientes:
- *Agregar*, que permite agregar un elemento a un conjunto.
- *Sacar*, que permite eliminar un elemento de un conjunto.
- *Elegir*, que devuelve un elemento arbitrario de un conjunto no vacío.
- *Pertenece*, que nos dice si un elemento está en un conjunto.
- *ConjuntoVacio*, que indica si el conjunto tiene elementos o no.
- *InicializarConjunto*, que inicializa el conjunto.

La interfaz (especificación) de *ConjuntoTDA*

La interfaz (especificación) de *ConjuntoTDA*

- La interfaz de `ConjuntoTDA` es la siguiente:

```
1.  public interface ConjuntoTDA; {
2.      void InicializarConjunto();      // sin precondiciones
3.      void Agregar(int x);             // conjunto inicializado
4.      int Elegir();                    // conjunto inicializado y no vacío
5.      boolean ConjuntoVacio();         // conjunto inicializado
6.      void Sacar();                    // conjunto inicializado
7.      boolean Pertenece();             // cola inicializado
8.  }
```

La interfaz (especificación) de *ConjuntoTDA*

- La interfaz de `ConjuntoTDA` es la siguiente:

```
1.  public interface ConjuntoTDA; {
2.      void InicializarConjunto();      // sin precondiciones
3.      void Agregar(int x);             // conjunto inicializado
4.      int Elegir();                    // conjunto inicializado y no vacío
5.      boolean ConjuntoVacio();         // conjunto inicializado
6.      void Sacar();                    // conjunto inicializado
7.      boolean Pertenece();             // cola inicializado
8.  }
```

- Ejemplo: determinación de si un conjunto *Conjunto1* incluye a otro conjunto *Conjunto2*.

```
1.  public boolean Incluye (ConjuntoTDA Conjunto1, ConjuntoTDA Conjunto2) {
2.      boolean incluye = true;
3.      while (!Conjunto2.ConjuntoVacio() && incluye) {
4.          int x = Conjunto2.Elegir());
5.          if (!Conjunto1.Pertenece(x)) {
6.              incluye = false;
7.          } else {
8.              Conjunto2.Sacar(x);
9.          }
10.     }
11.     return incluye
12. }
```

¿Por dónde vamos?

Estrategia para la implementación

Estrategia para la implementación

- Se definirá un arreglo *a* que contendrá los elementos del conjunto. Una variable entera indicará, como es habitual, la cantidad de posiciones utilizadas en el arreglo, que coincidirá con la cantidad de elementos del conjunto. La llamaremos *cant*.

Estrategia para la implementación

- Se definirá un arreglo *a* que contendrá los elementos del conjunto. Una variable entera indicará, como es habitual, la cantidad de posiciones utilizadas en el arreglo, que coincidirá con la cantidad de elementos del conjunto. La llamaremos *cant*.
- Como en un conjunto no puede haber repeticiones, antes de agregar un nuevo elemento debemos asegurarnos de que no esté ya en él. Como no importa el orden, lo colocaremos directamente en la primera posición disponible (el índice al que apunta *cant*.)

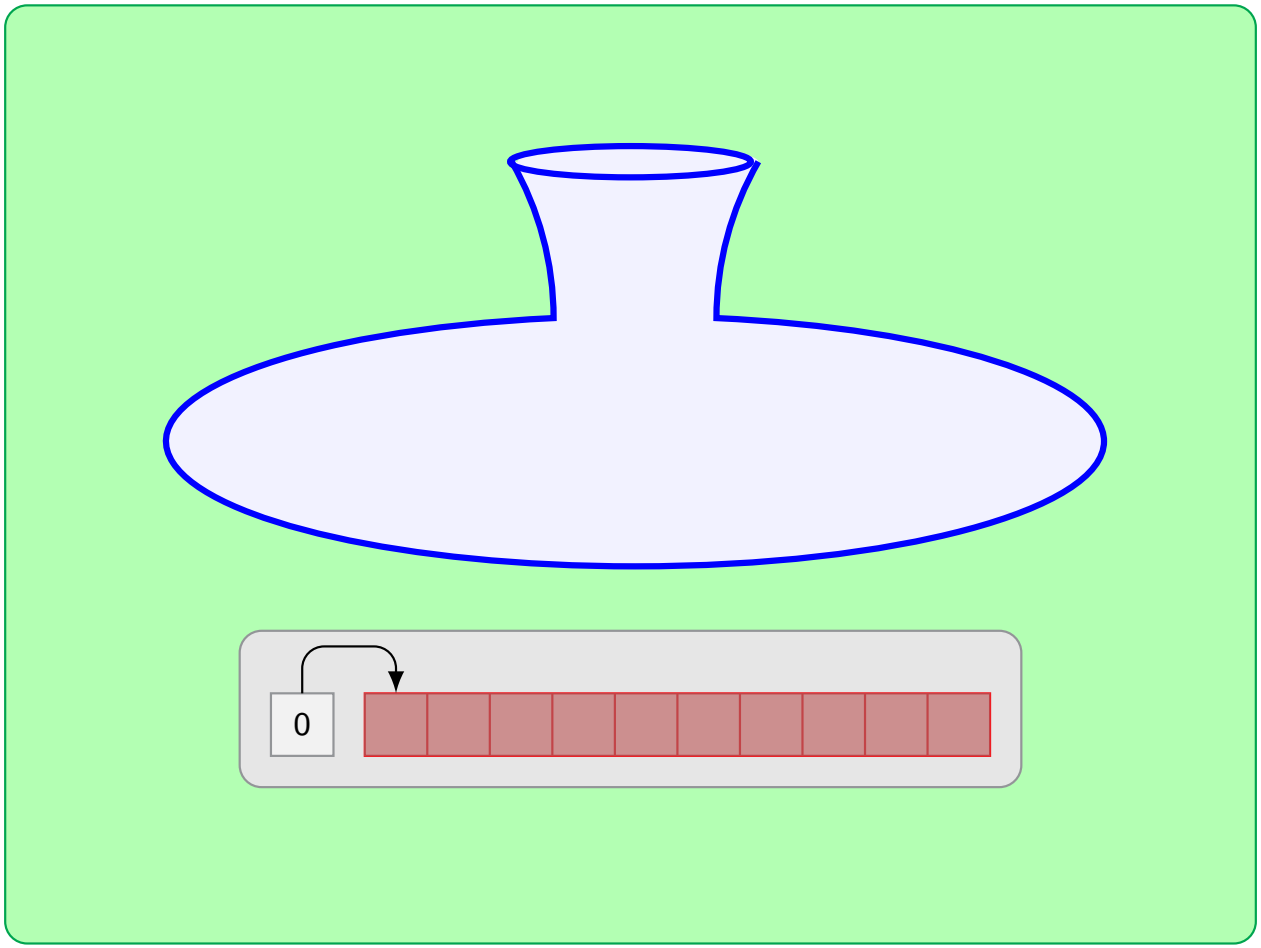
Estrategia para la implementación

- Se definirá un arreglo *a* que contendrá los elementos del conjunto. Una variable entera indicará, como es habitual, la cantidad de posiciones utilizadas en el arreglo, que coincidirá con la cantidad de elementos del conjunto. La llamaremos *cant*.
- Como en un conjunto no puede haber repeticiones, antes de agregar un nuevo elemento debemos asegurarnos de que no esté ya en él. Como no importa el orden, lo colocaremos directamente en la primera posición disponible (el índice al que apunta *cant*.)
- Para determinar pertenencia de un elemento dado al conjunto es necesario recorrerlo enteramente, pues el arreglo *a* está desordenado.

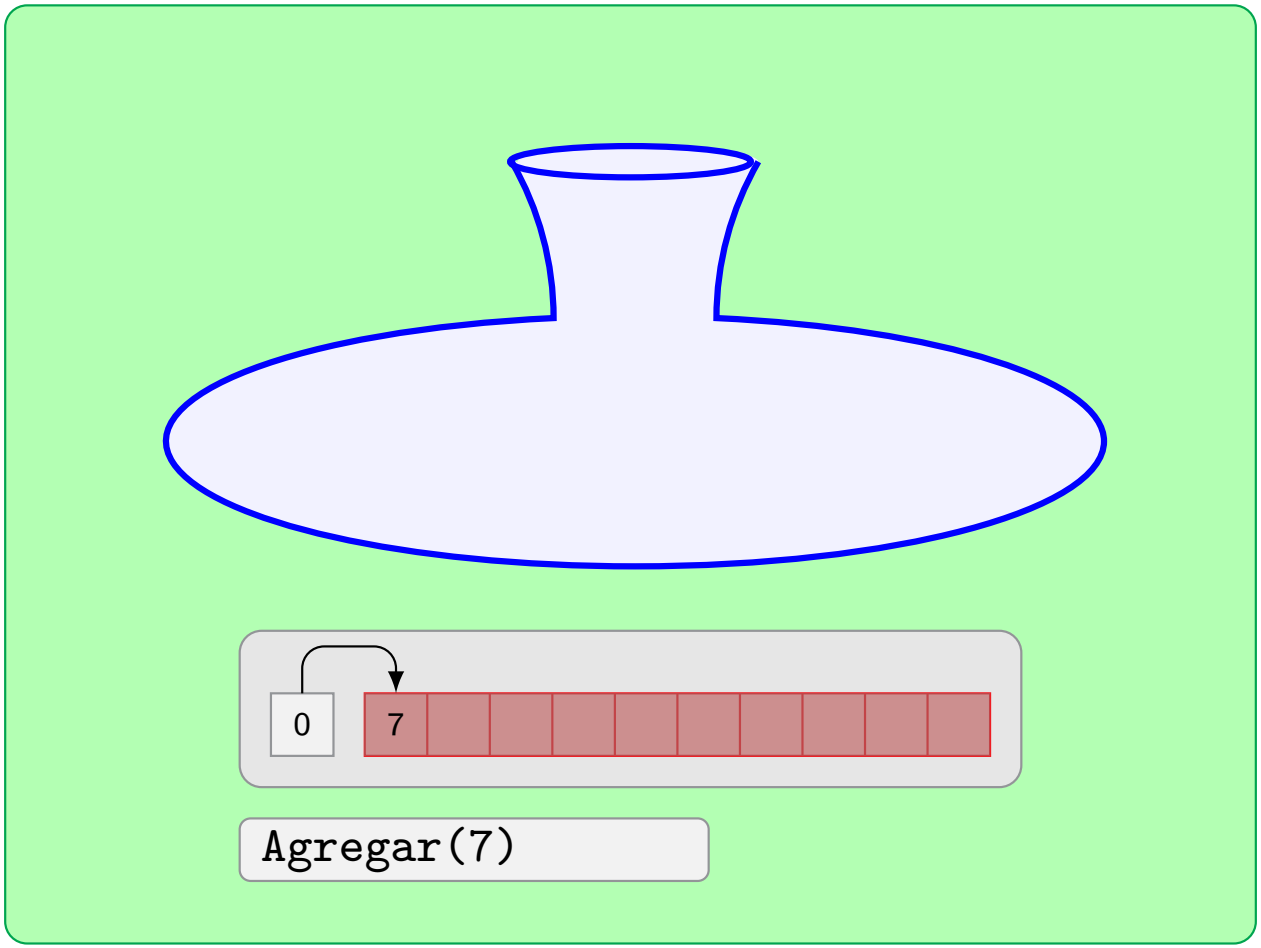
Estrategia para la implementación

- Se definirá un arreglo *a* que contendrá los elementos del conjunto. Una variable entera indicará, como es habitual, la cantidad de posiciones utilizadas en el arreglo, que coincidirá con la cantidad de elementos del conjunto. La llamaremos *cant*.
- Como en un conjunto no puede haber repeticiones, antes de agregar un nuevo elemento debemos asegurarnos de que no esté ya en él. Como no importa el orden, lo colocaremos directamente en la primera posición disponible (el índice al que apunta *cant*.)
- Para determinar pertenencia de un elemento dado al conjunto es necesario recorrerlo enteramente, pues el arreglo *a* está desordenado.
- Para eliminar un elemento, se lo sobreescribe con el último elemento el arreglo (posición *cant-1*) y se decrementa posteriormente *cant*.

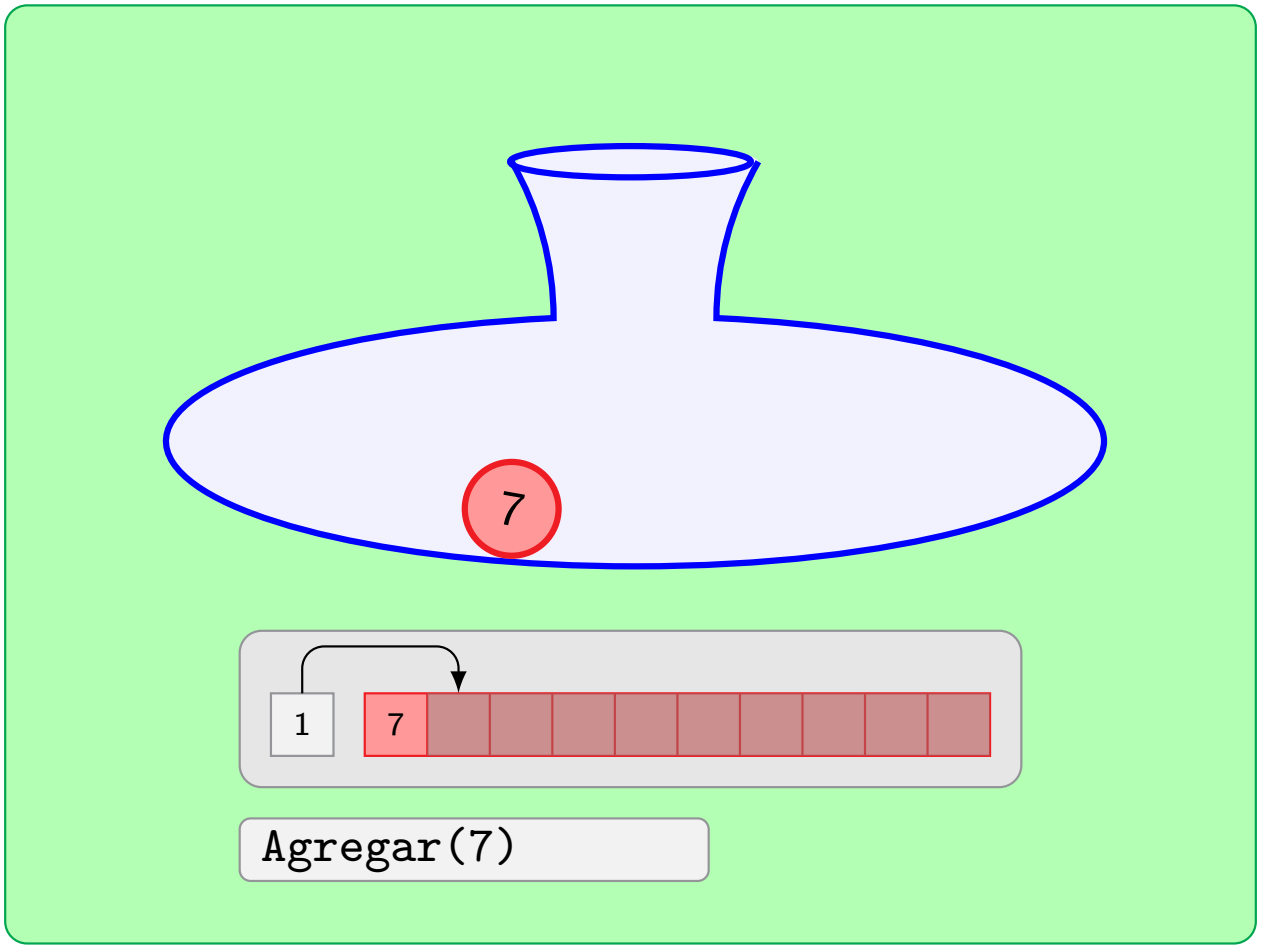
Visualización de la estrategia



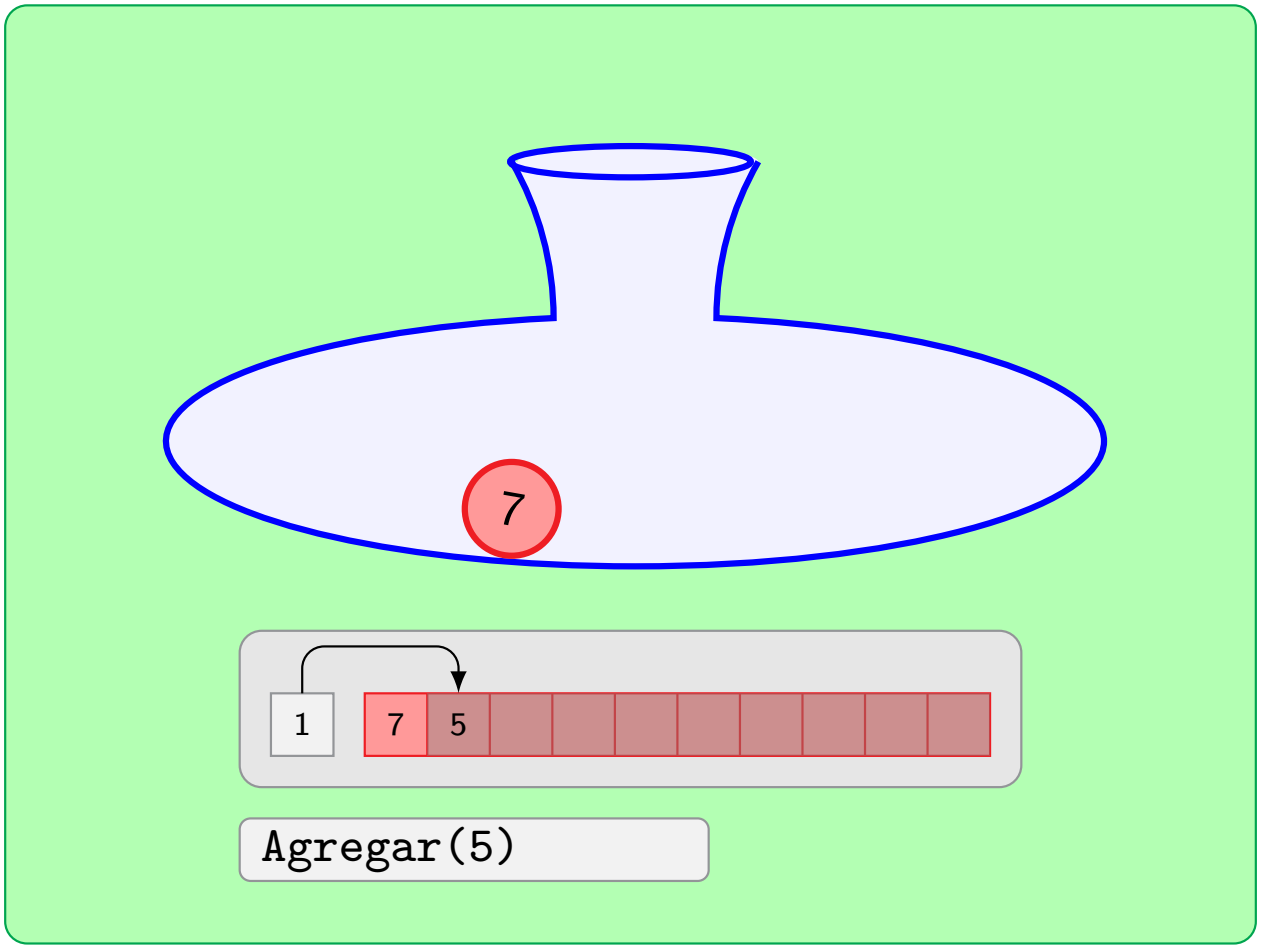
Visualización de la estrategia



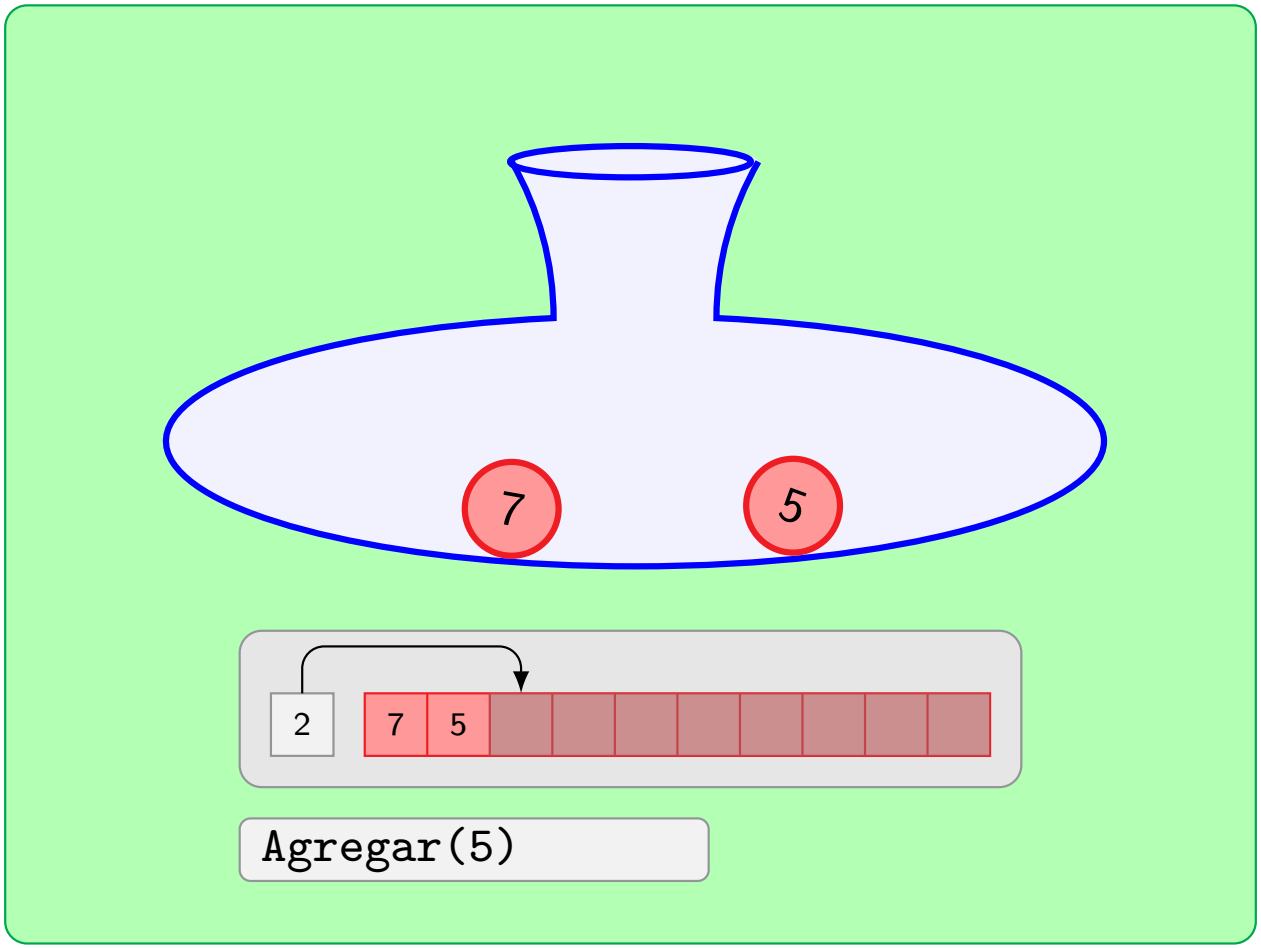
Visualización de la estrategia



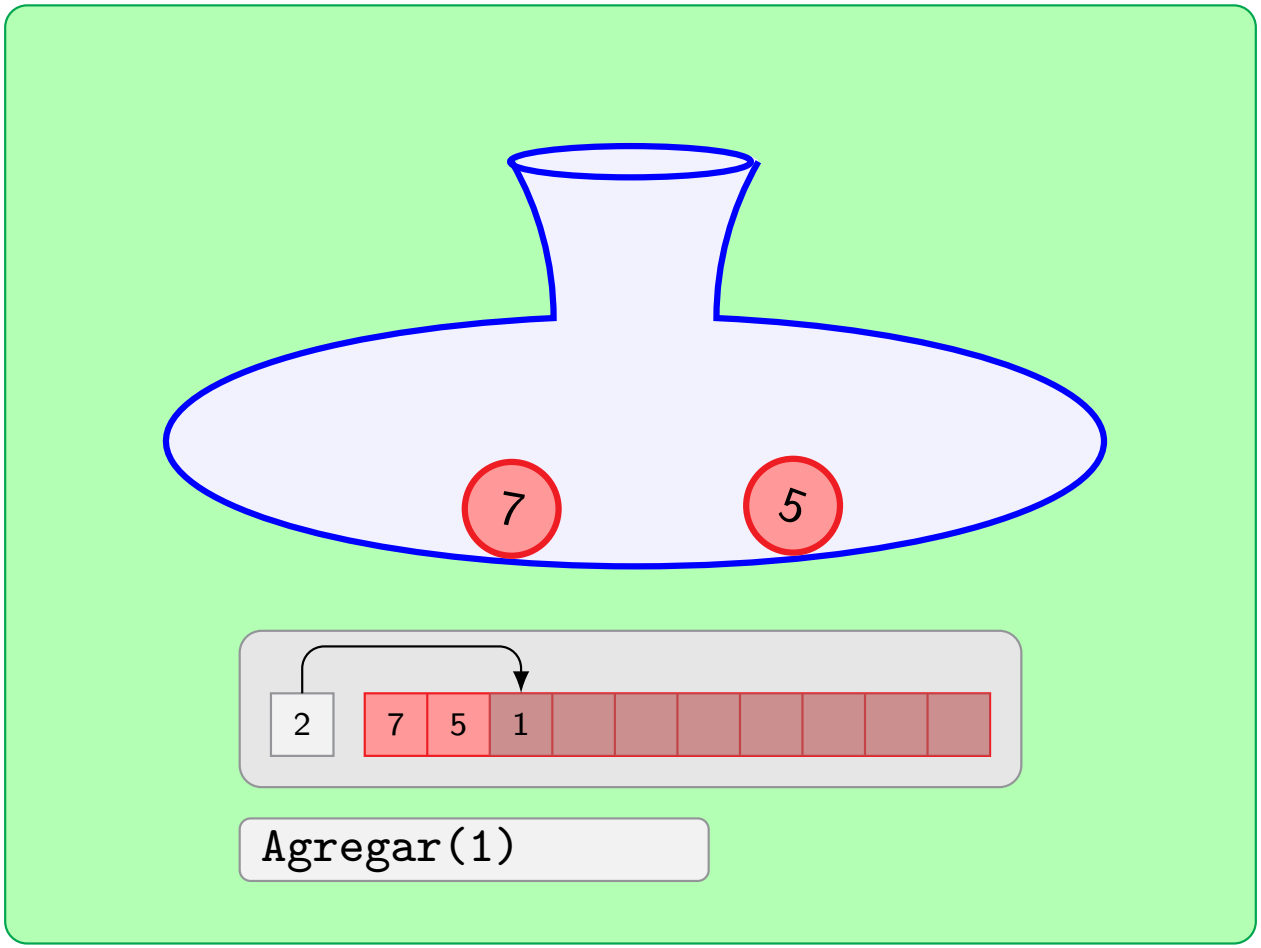
Visualización de la estrategia



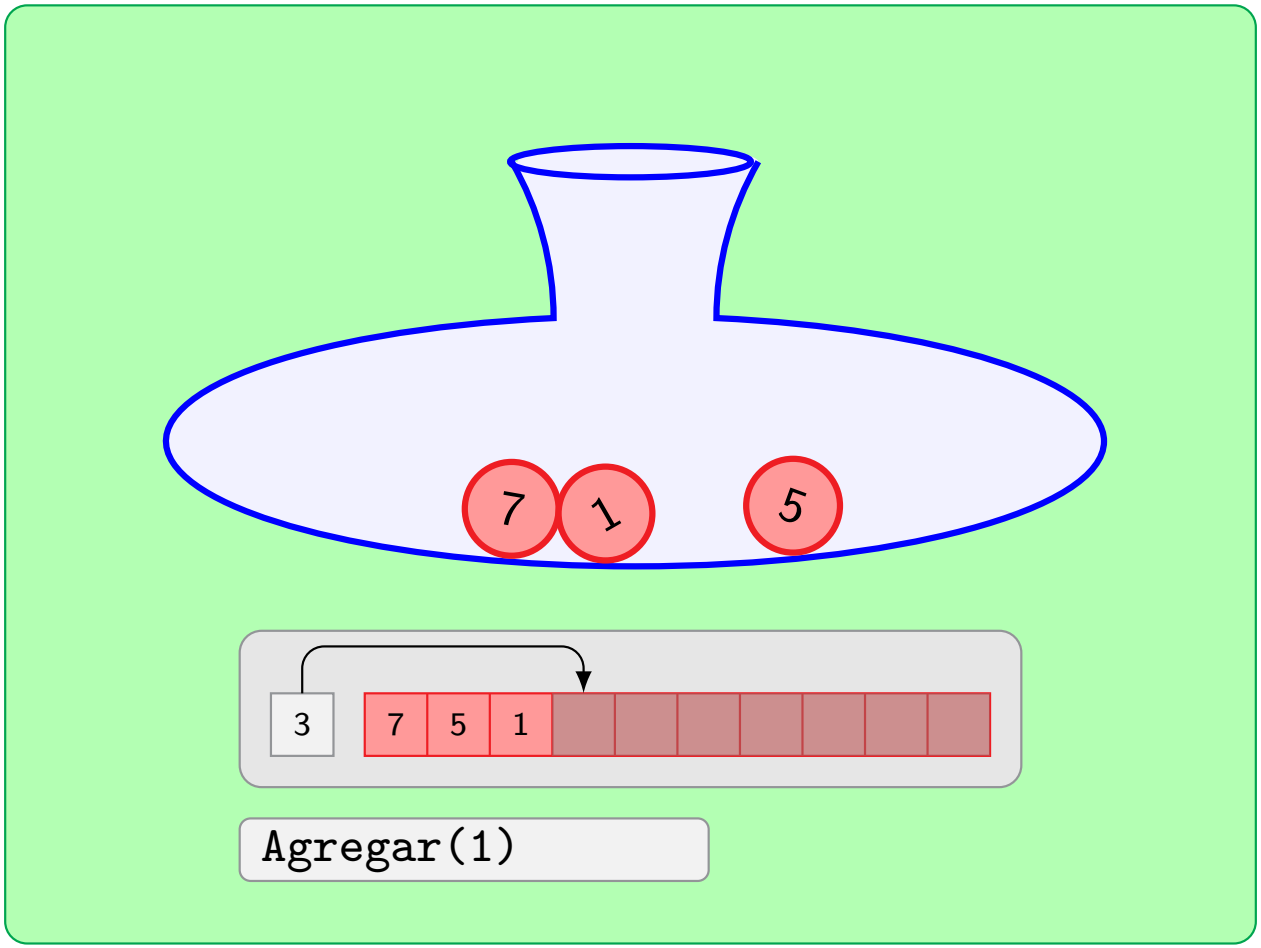
Visualización de la estrategia



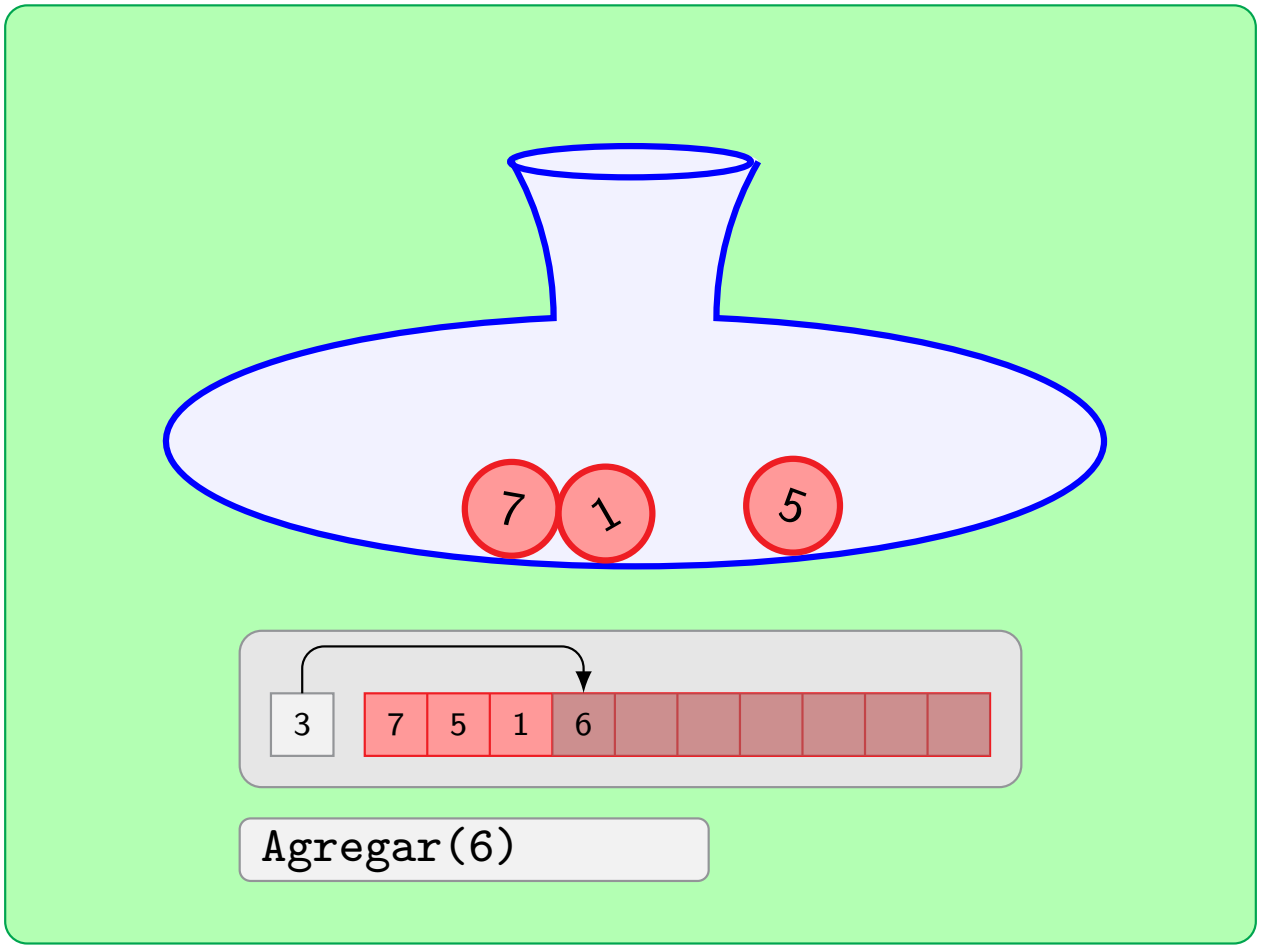
Visualización de la estrategia



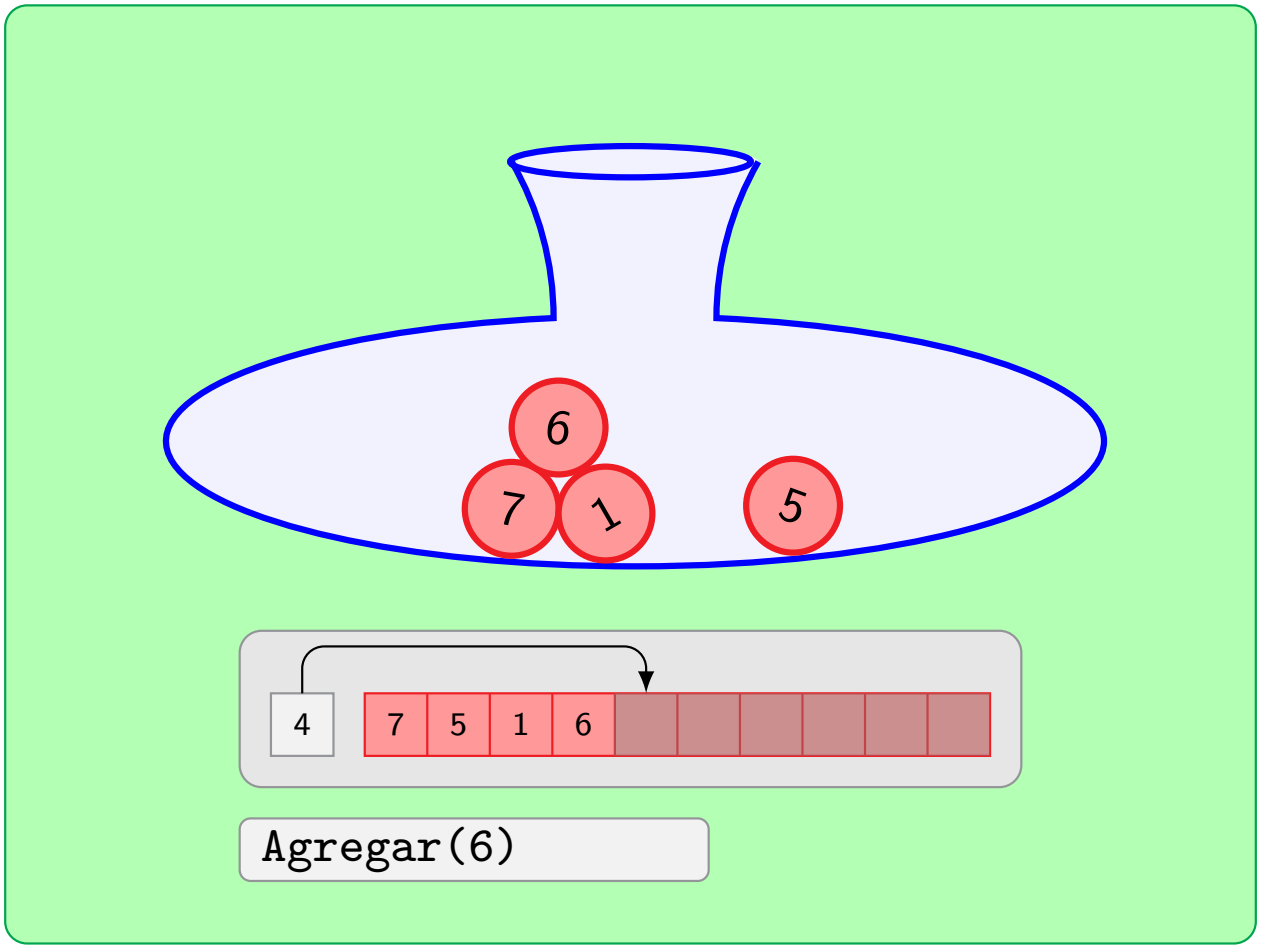
Visualización de la estrategia



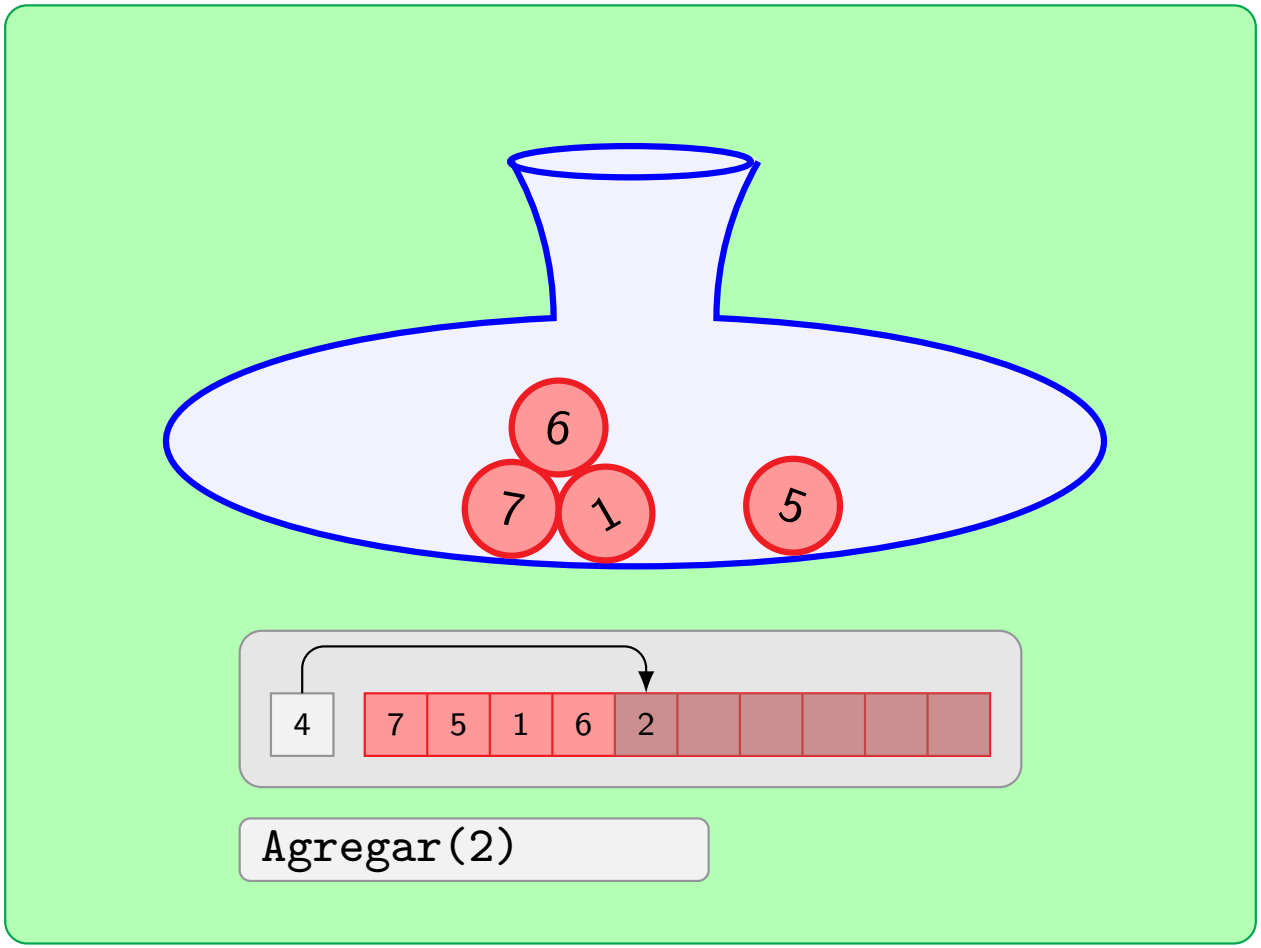
Visualización de la estrategia



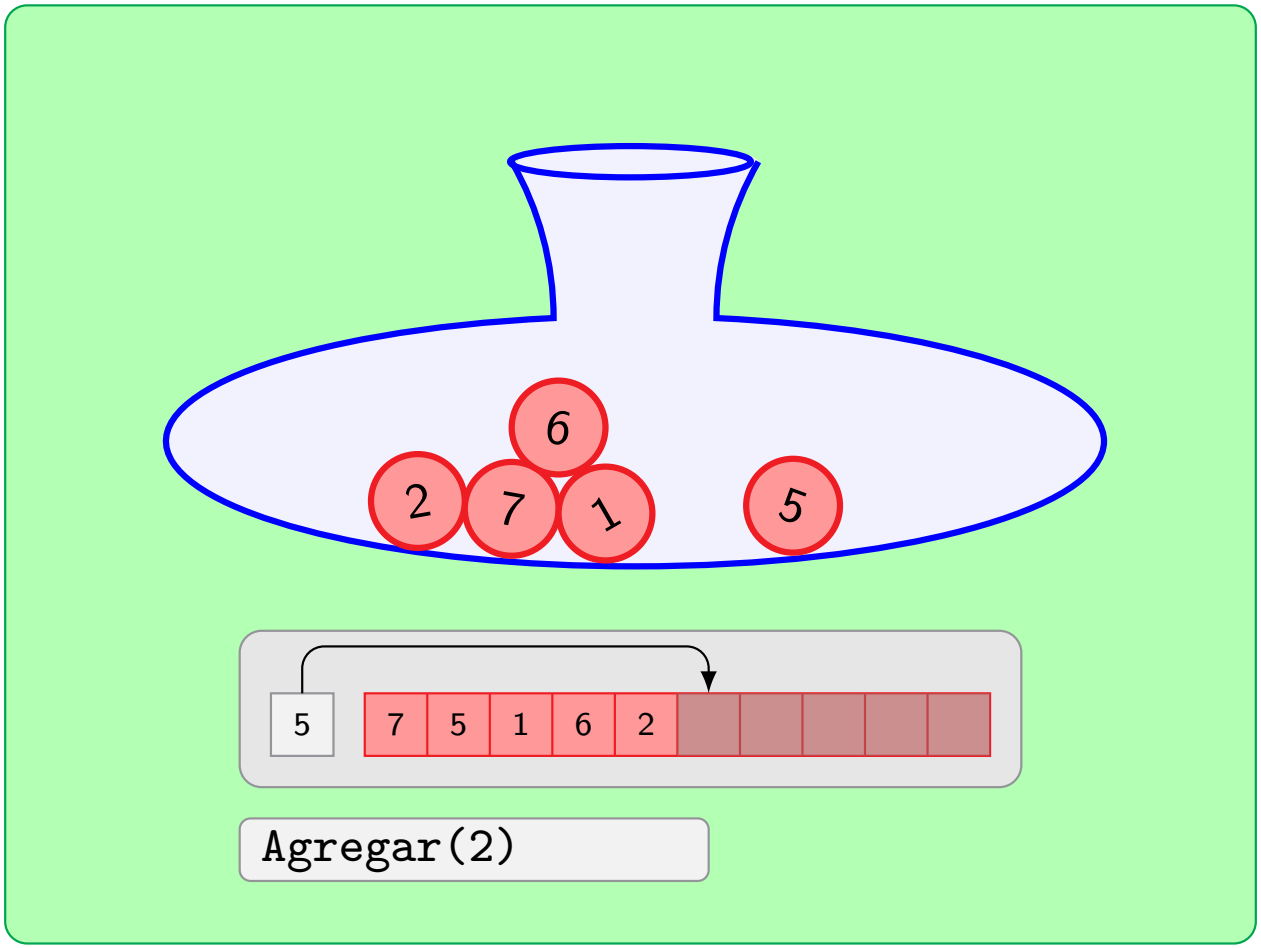
Visualización de la estrategia



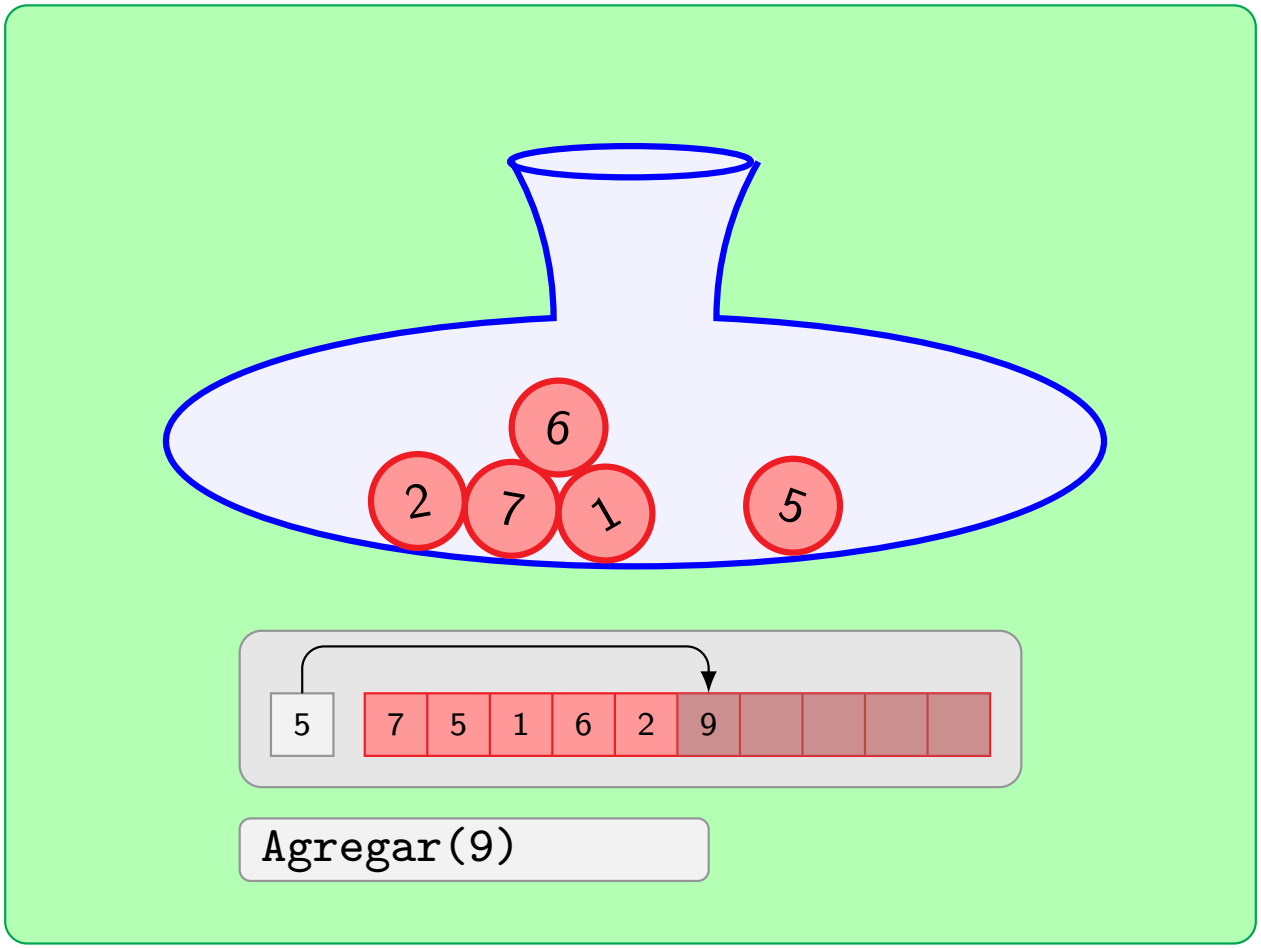
Visualización de la estrategia



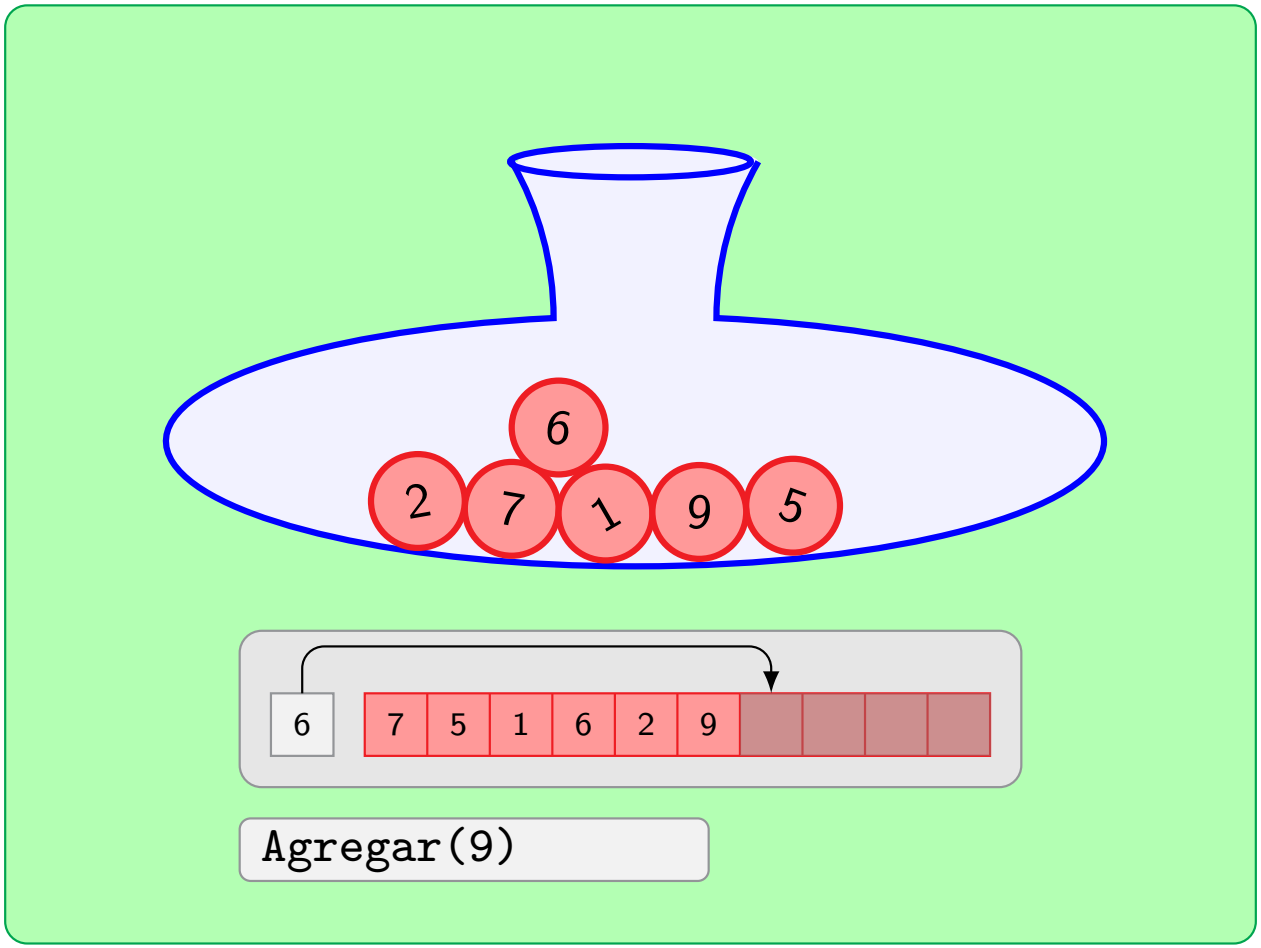
Visualización de la estrategia



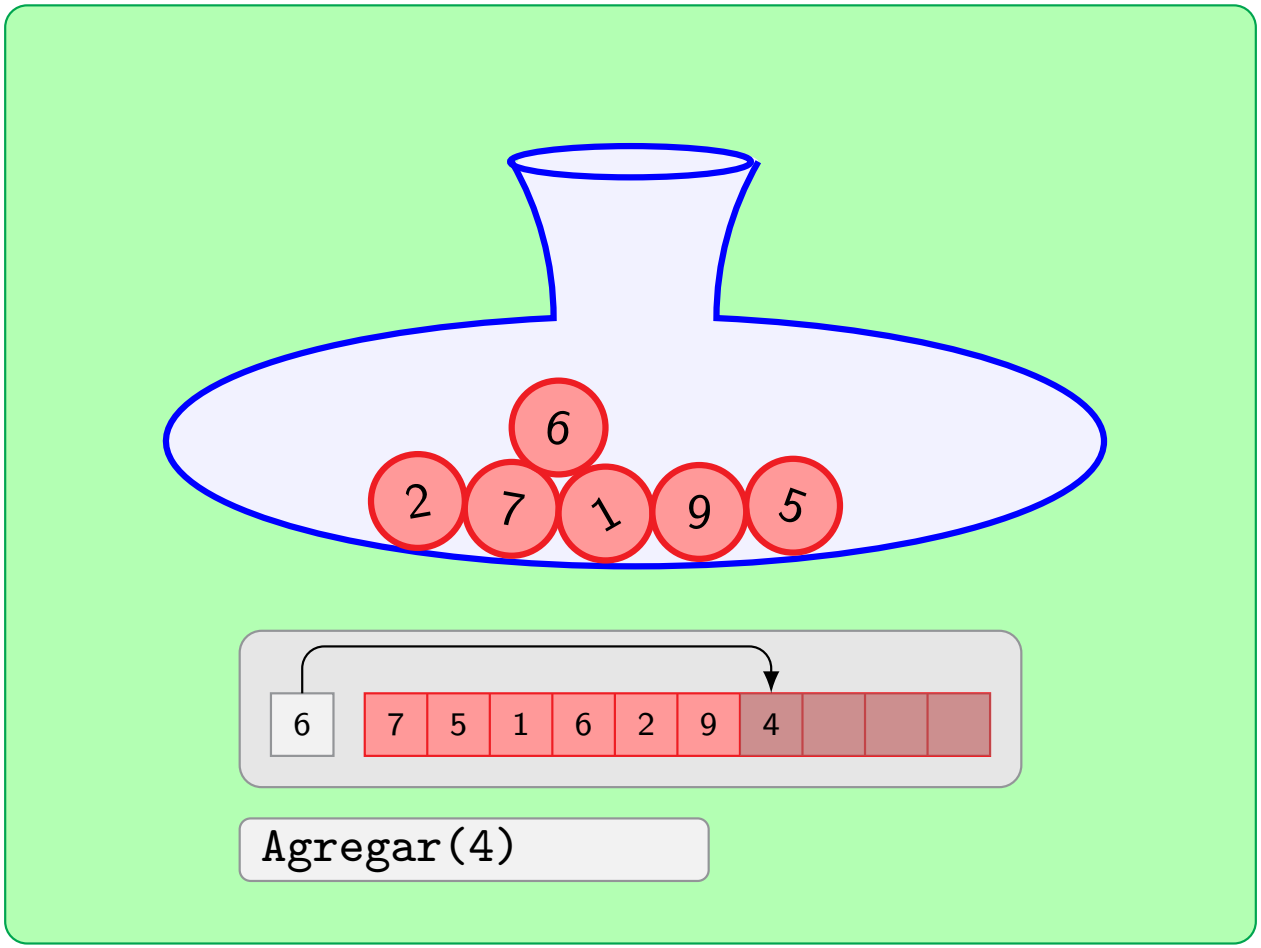
Visualización de la estrategia



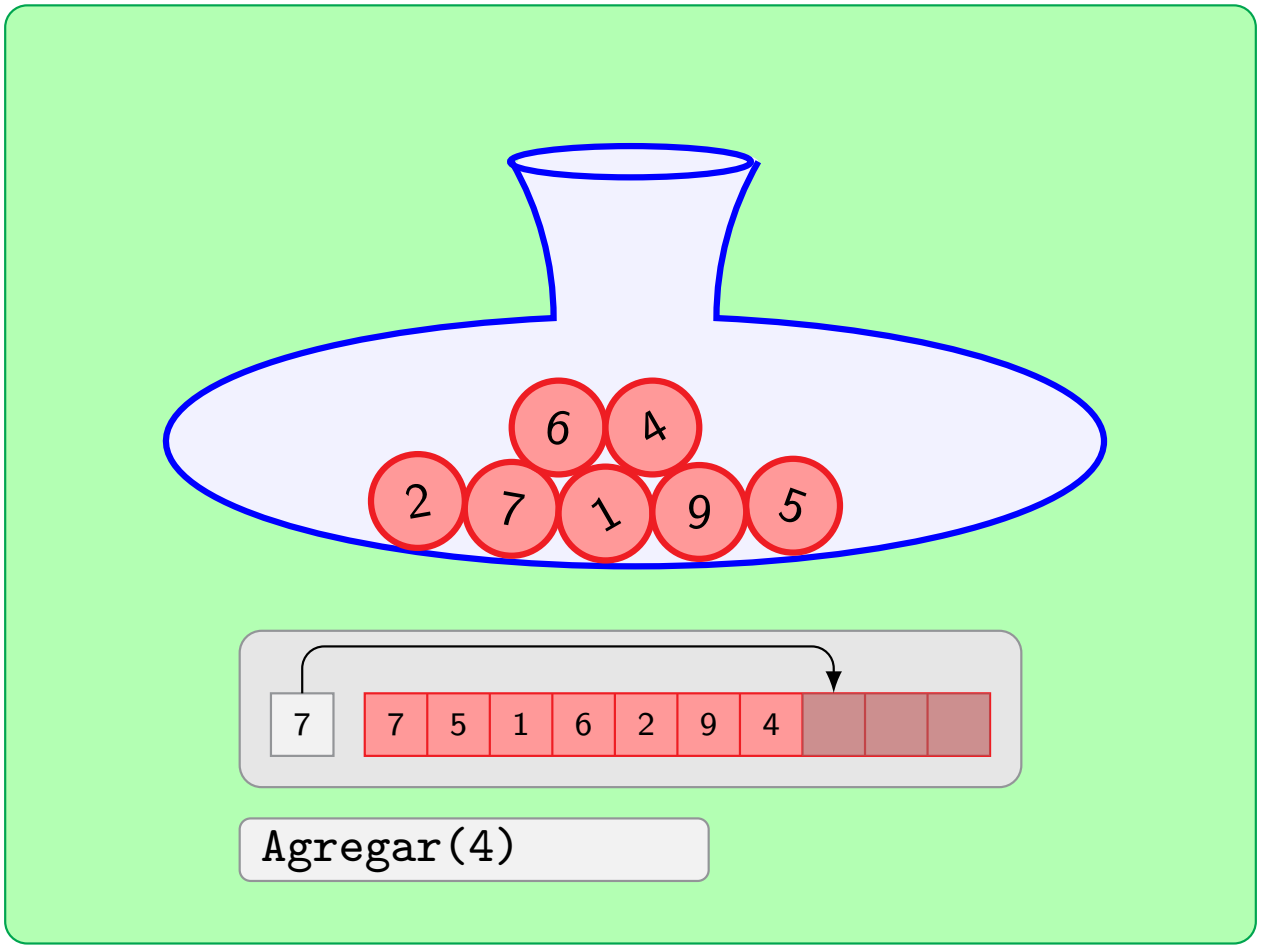
Visualización de la estrategia



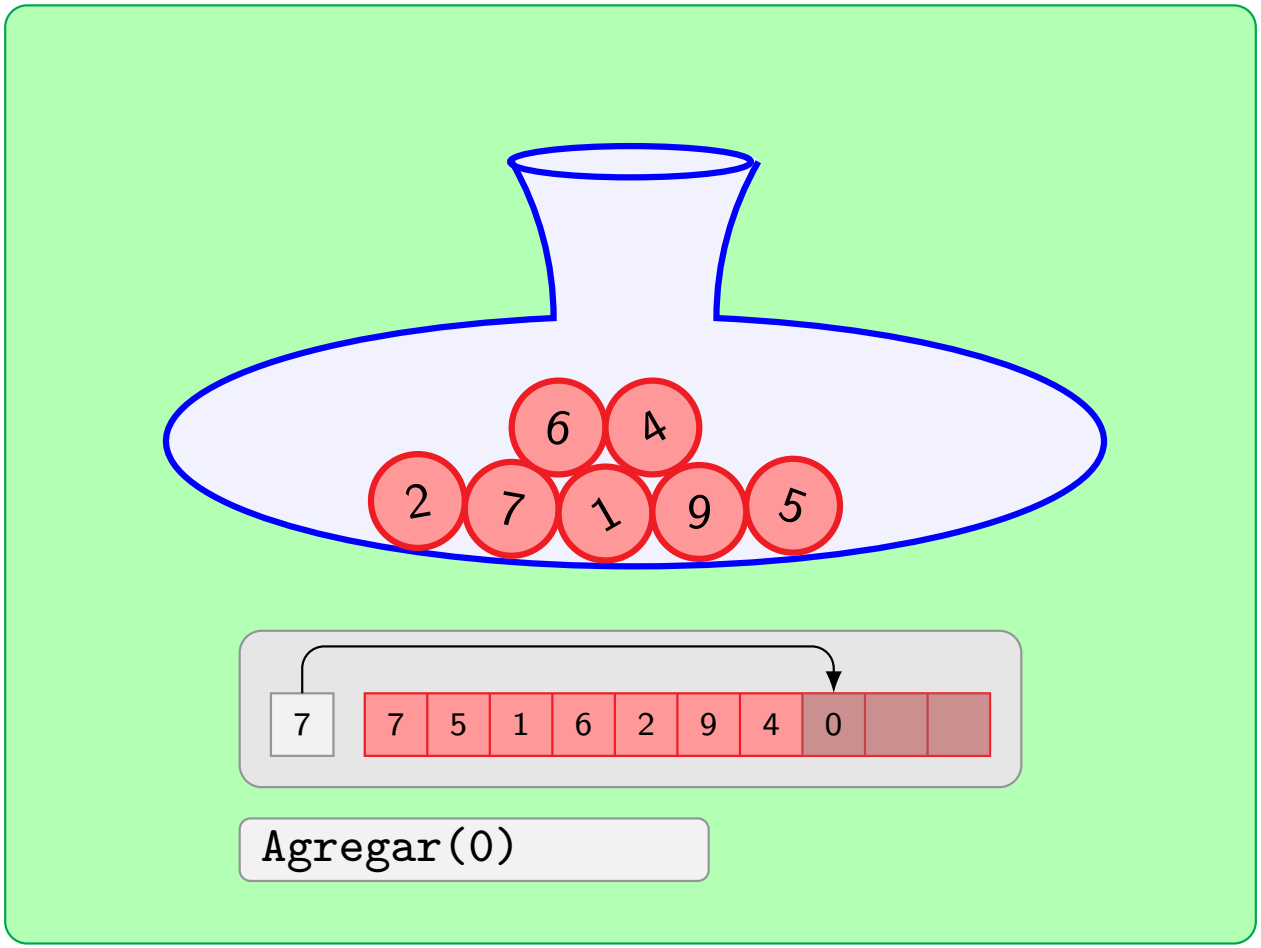
Visualización de la estrategia



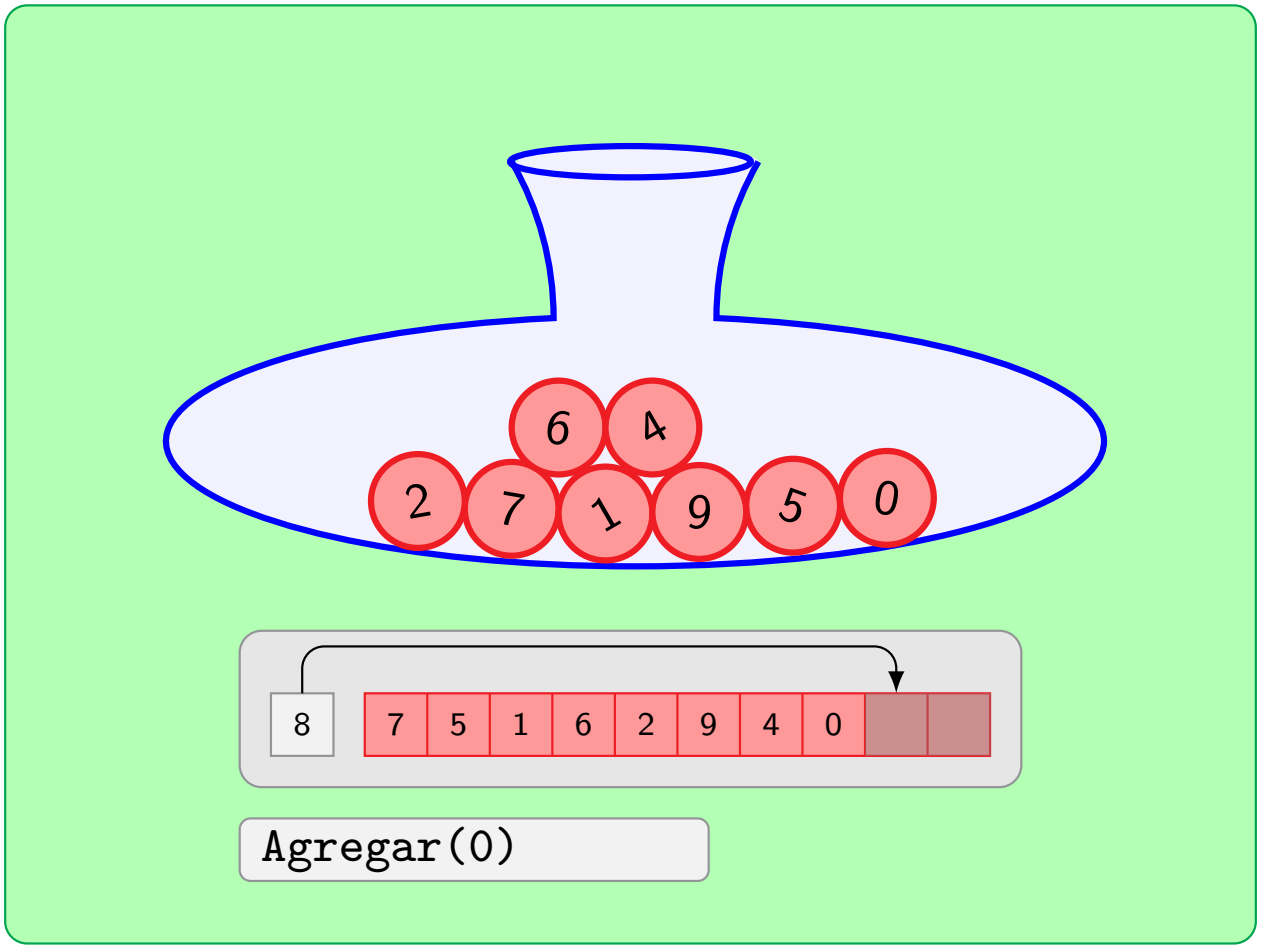
Visualización de la estrategia



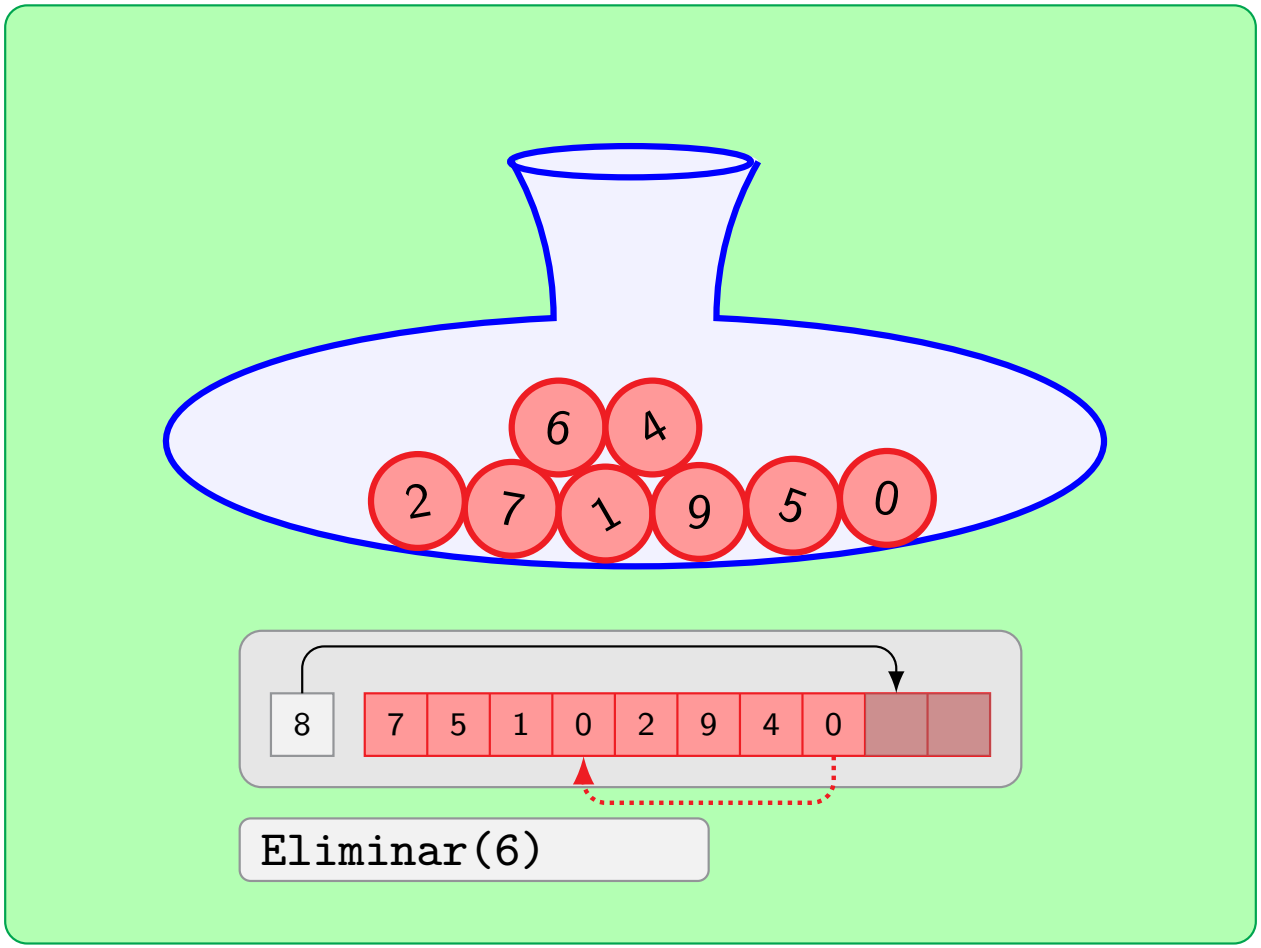
Visualización de la estrategia



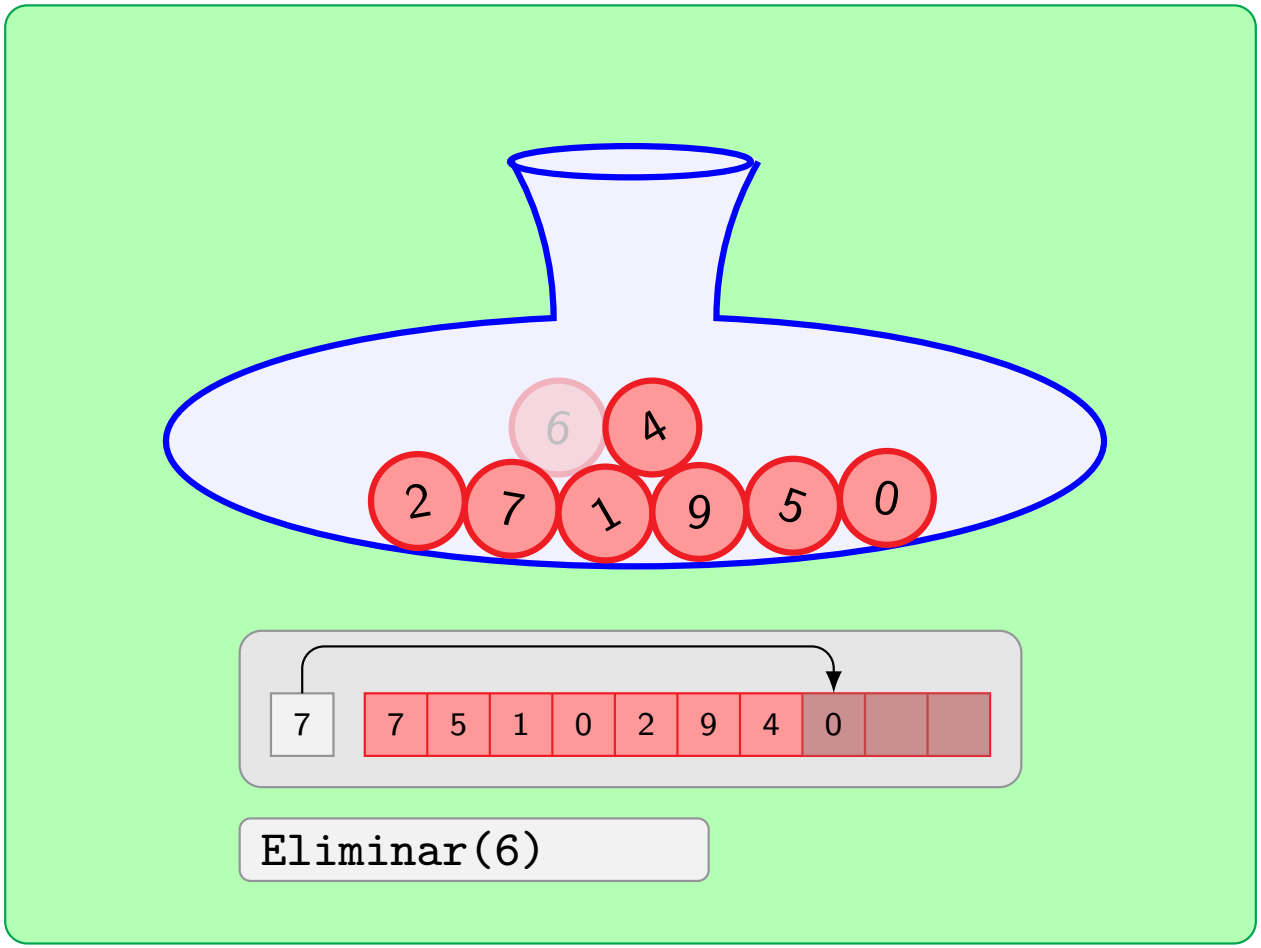
Visualización de la estrategia



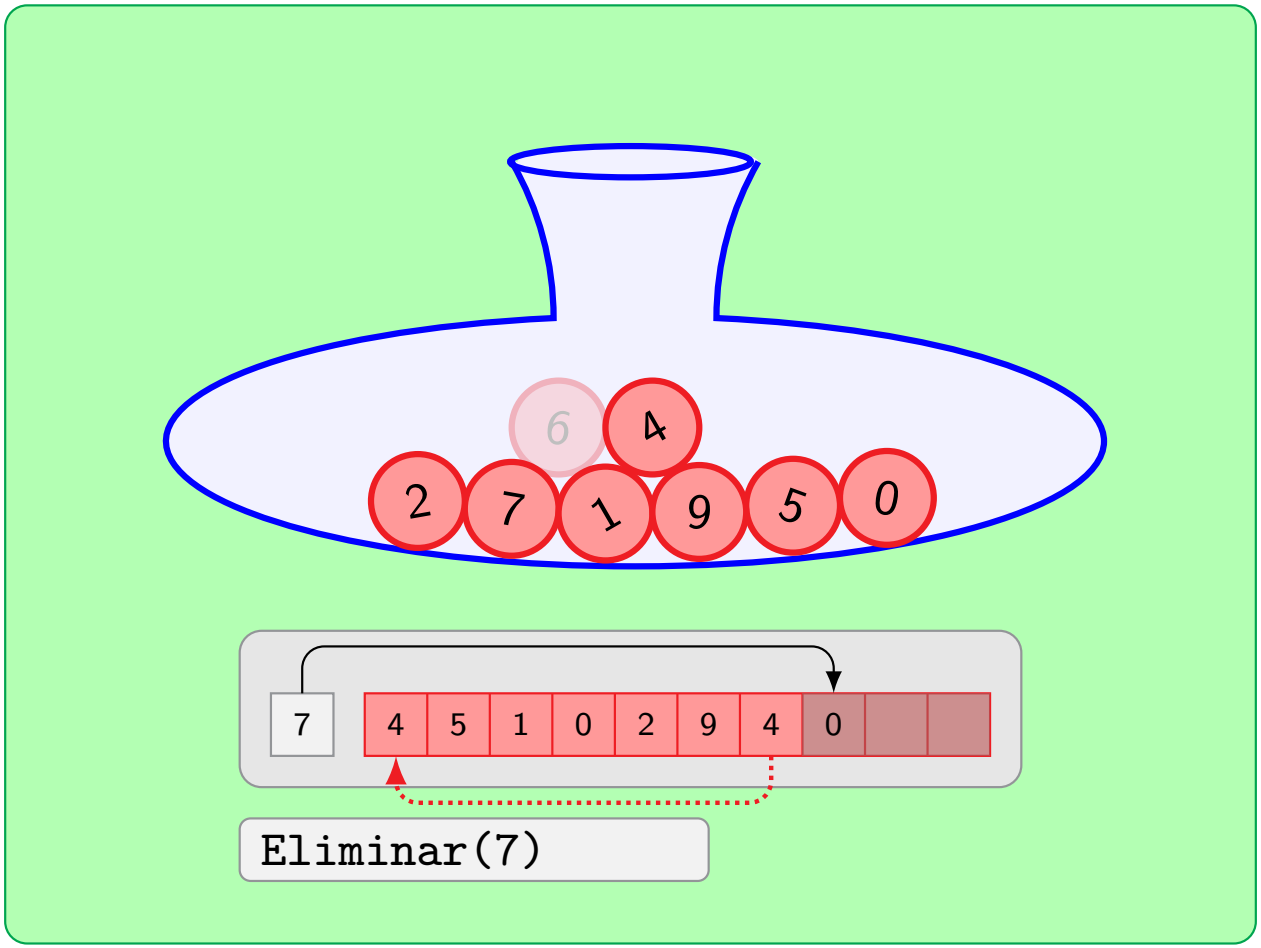
Visualización de la estrategia



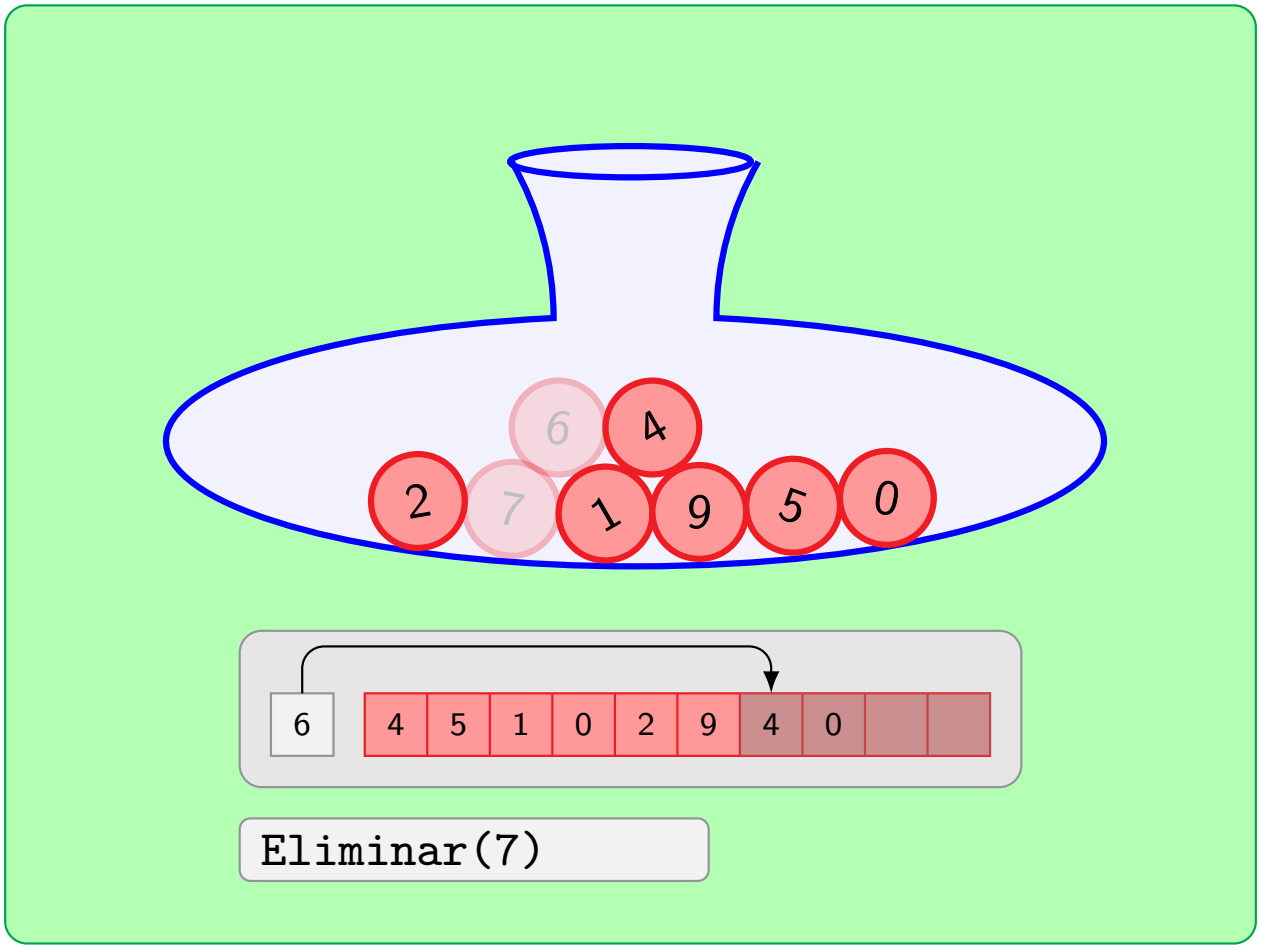
Visualización de la estrategia



Visualización de la estrategia



Visualización de la estrategia



Algunas explicaciones

Algunas explicaciones

- Adoptamos la convención de colores habitual:
  - Los métodos públicos, que pueden ser invocados desde fuera de la implementación, están en azul.
  - Los comentarios están en verde.
  - Los métodos y estructuras de datos no accesibles desde fuera de la implementación están en rojo.

Implementación de *ConjuntoTDA* 1

```
1.  public class ConjuntoA implements ConjuntoTDA {
2.      int[] a;                                  // El contenido del conjunto
3.      int cant;                                 // La cantidad de elementos
4.      public void InicializarConjunto() {
5.          a = new int[100];
6.          cant = 0;
7.      }
8.      public void Agregar (int x) {
9.          if (!this.Pertenece(x) {              // Verificación de no pertenencia
10.             a[cant] = x;
11.             cant++;                           // Nuevo elemento
12.         }
13.     }
14.     public boolean ConjuntoVacio() {
15.         return (cant == 0);
16.     }
17.     public int Elegir() {
18.         return a[cant-1];                     // Esto es arbitrario; podría ser cualquiera
19.     }
```

Implementación de *ConjuntoTDA* 2

```
20.         public boolean Pertenece (int x) {
21.             int i = 0;
22.             while (i < cant && a[i] != x)
23.                 i++;
24.             return (i < cant);
25.         }
26.         public void Sacar (int x) {
27.             int i = 0;
28.             while (i < cant && a[i] != x)
29.                 i++;
30.             if (i < cant) {                    // elemento encontrado
31.                 a[i] = a[cant-1];
32.                 cant--;
33.             }
34.         }
35.     }
```