Programación Orientada a Objetos y TADs

Algoritmos y Estructuras de Datos I

Pablo Castro

Departamento de Computación, FCEFQyN, UNRC, Argentina

Tipos de Datos

Un **tipo de datos** es una colección de elementos con operaciones sobre ellos.

Los `int` en Java:

- Elementos: …,-1,0,1,2,…
- Operaciones: *,+, etc

Los `bool` en Java:

- Elementos: true, false
- Operaciones: &&, ||, !, etc

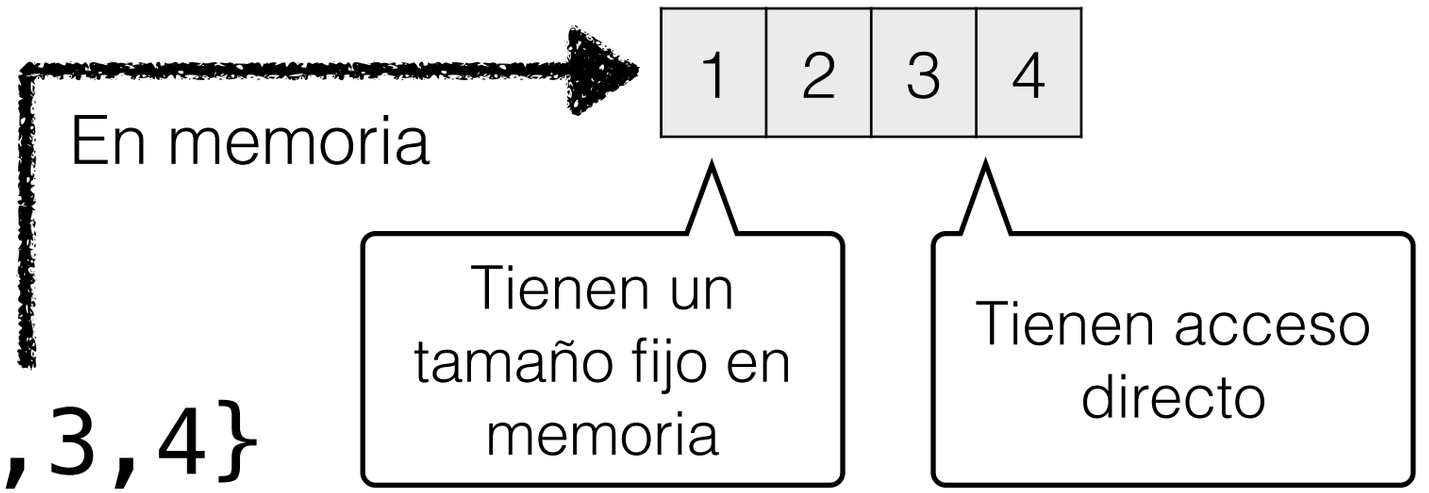
Estructuras de Datos

Las **estructuras de datos** son los mecanismos que proveen los lenguajes de programación para poder guardar y manipular datos

Ejemplos:

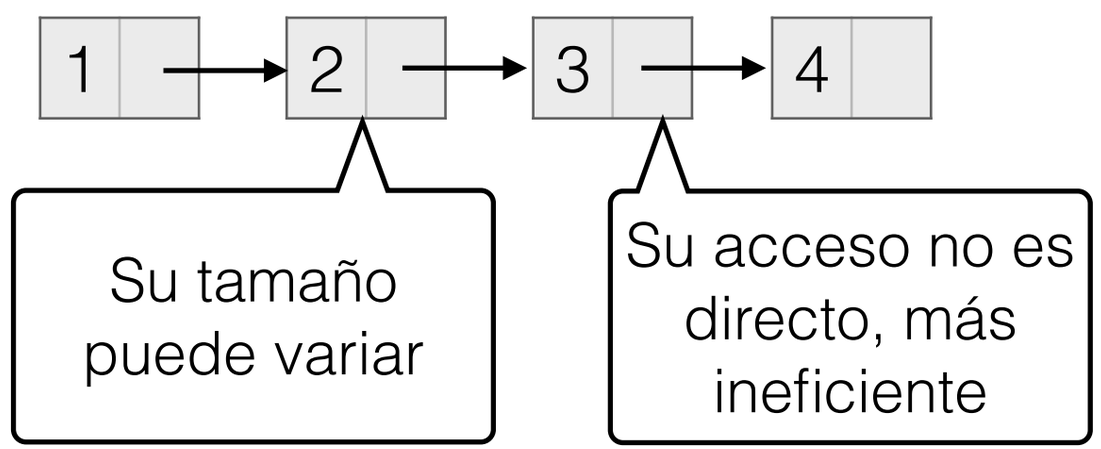
- Arreglos en Java:

```
int[] a = {1,2,3,4}
```



- Listas en Python:

```
X = [1,2,3,4,5]
```

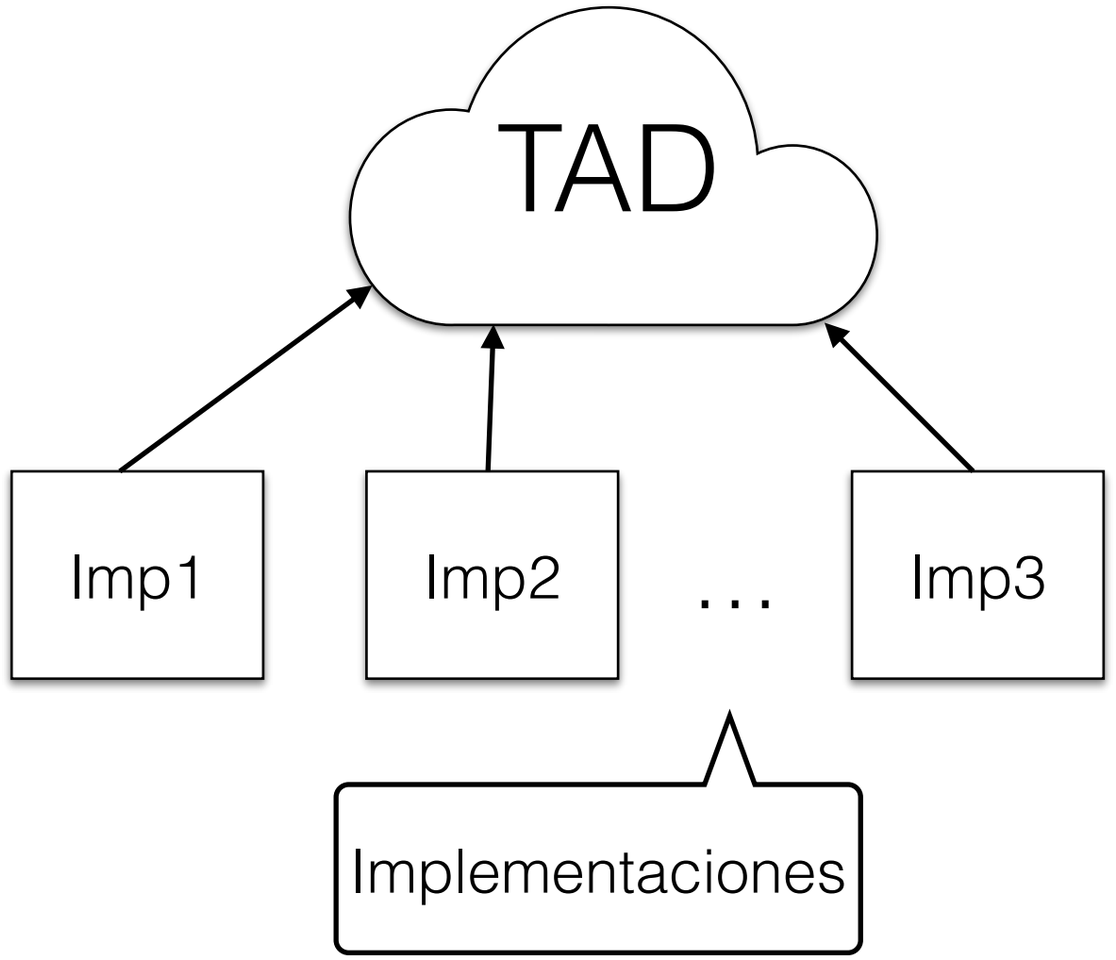


Tipos Abstractos de Datos

Un **Tipo Abstracto de Datos** es una descripción abstracta de un tipo de datos, mediante la especificación de sus operaciones

Observaciones:

- Cuando se define un TAD no se dice cómo se lo implementa.
- Un TAD se puede implementar de diversas formas.



Ejemplo: Pilas

Las pilas son uno de los TADs más usados en computación:

Descripción informal:

- Una **pila** lleva una colección lineal de elementos: `x1,x2,x3,…,xN` el último es llamando **tope**
- El TAD soporta las siguientes operaciones:
  - **Apilar** (push): Agregar un nuevo elemento en el tope, preservando el orden del resto
  - **Desapilar** (pop): Saca el elemento en el tope, preservando el resto.
  - **Tope** (top): Devuelve el elemento en el tope.

Ejemplo: Colas

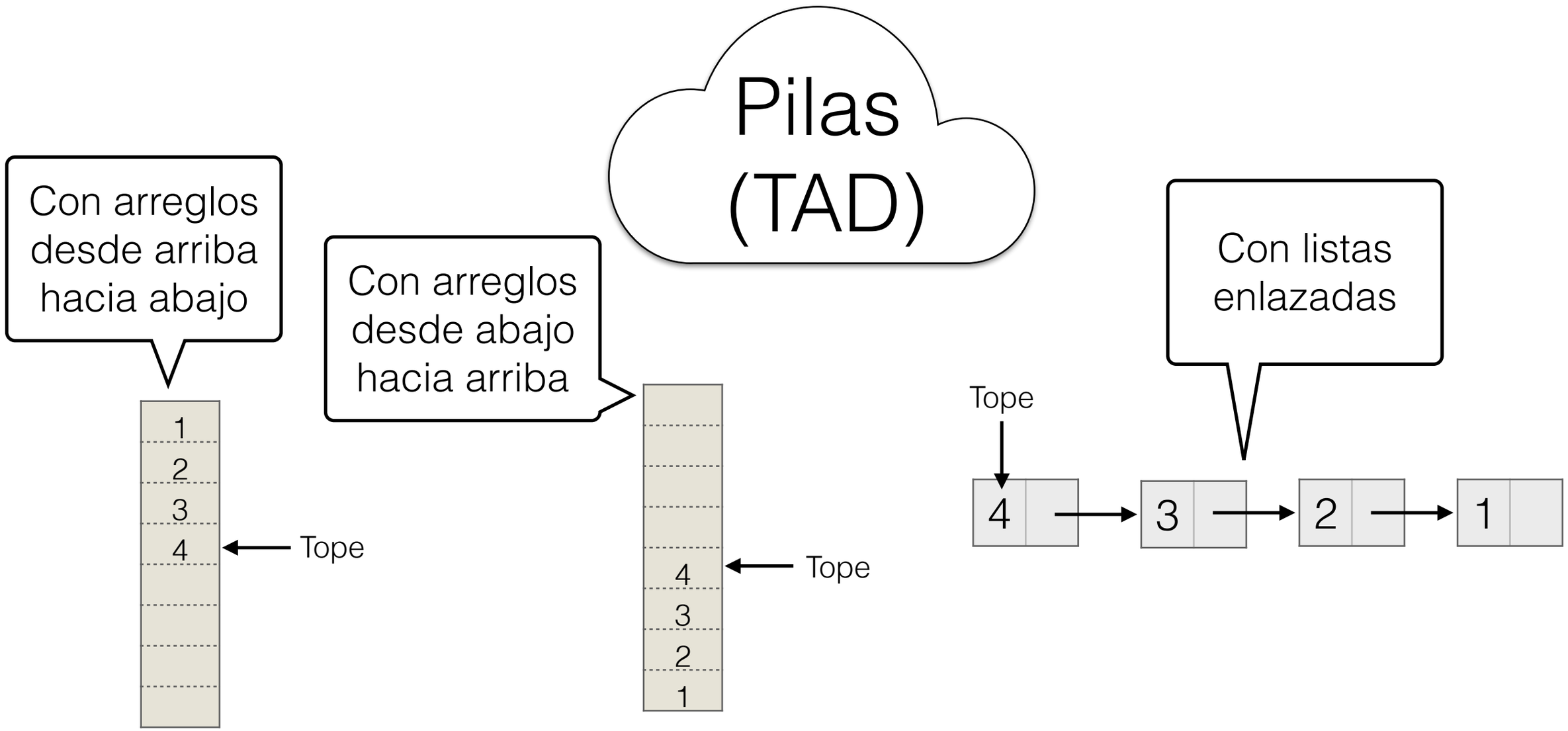
Las colas son otro de los TADs más usados en computación:

Descripción informal:

- Una **cola** lleva una colección lineal de elementos: `x1,x2,x3,…,xN`, un extremo es llamado inicio y otro final.
- El TAD soporta las siguientes operaciones:
  - **Encolar** (enqueue): Agregar un nuevo elemento en el final, preservando el orden del resto
  - **Desencolar** (dequeue): Saca el elemento en el inicio, preservando el resto.

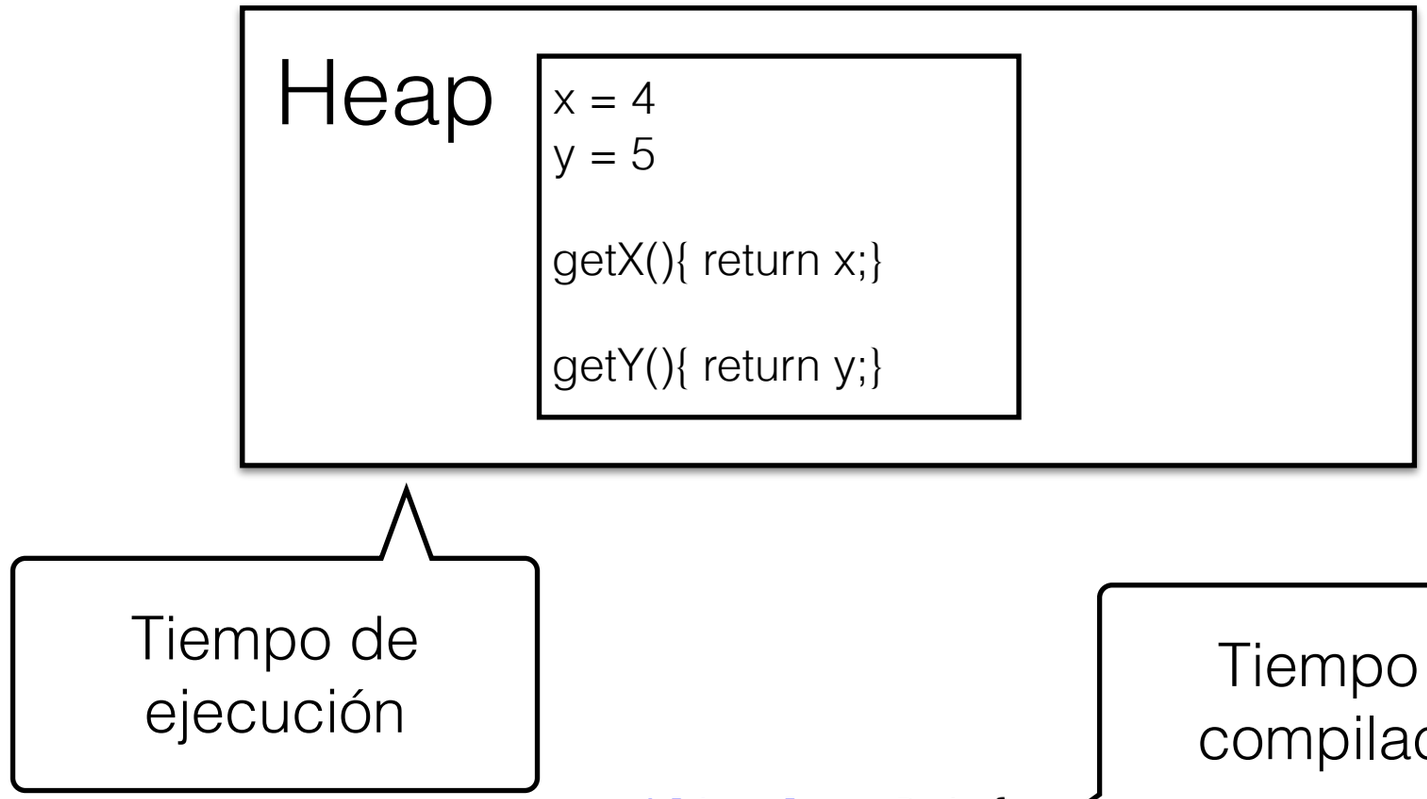
Implementación de TADs

Podemos implementar los TADs con diferentes estructuras de datos.



Clases y Objetos (Java)

Un **objeto** es un colección de datos y operaciones que reside en memoria en tiempo de ejecución



Una **clase** es una definición de una colección de objetos, que comparten la misma estructura

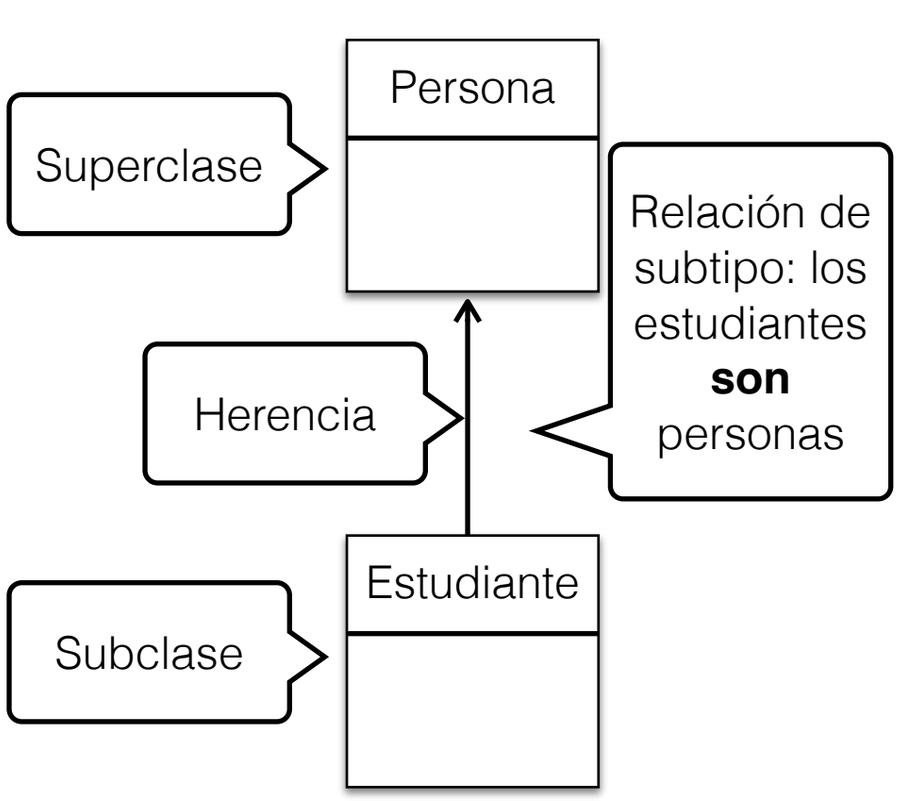
Tiempo de compilación

```
public class Pair{
    private int fst;
    private int snd;

    /**
    * Constructor of the class
    **/
    public Pair(){
        fst = 0;
        snd = 0;
    }

    // sigue…
```

Herencia

La orientación a objetos ofrece diversas herramientas para el reuso y modularización de código

La herencia introduce la noción de subclase:



En Java:

```java
public class Persona {
    private int dni;
    private String nombre;
    private String apellido;

    // sigue
}
```

Archivo: Persona.java

La clase estudiante hereda todos los atributos y métodos de Persona

```java
public class Estudiante extends Persona{
    private boolean esRegular;
    private float promedio;

    // sigue
}
```

Archivo: Estudiante.java

Ejemplo: Pares

Definamos una clase par que guarda dos enteros:

```java
public class Pair{
    private int fst;
    private int snd;

    /**
    * Constructor of the class
    **/
    public Pair(){
        fst = 0;
        snd = 0;
    }

    /**
     * Constructor of the class
     * @param x the first component
     * @param y the second component
     * @precondition true
     * @postcondition creates the new object
     **/
    public Pair(int fst, int snd){
        this.fst = fst;
        this.snd = snd;
    }

    /**
     * Get the first coordinate
     * @return the first value of the pair
     * @precondition true
     * @postcondition return the first item
     **/
    public int getFst(){
        return this.fst;
    }

    /**
     * Get the second coordinate
     * @return the second value of the pair
     * @precondition true
     * @postcondition return the second value.
     **/
    public int getSnd(){
        return this.snd;
    }

    /**
     * Change the first coordinate
     * @param x change the first value
     * @precondition true
     * @postcondition change x
     **/
    public void setFst(int fst){
        this.fst = fst;
    }

    /**
     * Change the second coordinate
     * @param y change the second value
     * @precondition true
     * @postcondition change the second value
     **/
    public void setSnd(int snd){
        this.snd = snd;
    }

}
```

Atributos

Métodos

Constructor

Getters

Setters

Una subclase de la clase Par

Podemos pensar una variante de pares que tengan memoria (permiten hacer undo)

```
public class MemoryPair extends Pair{

    private int previousFst; // the previous value of x
    private int previousSnd; // the previous value of y

    /**
    * A basic constructor
    */
    public MemoryPair(){
        super(); // it calls to the super constructor
        previousFst = 0;
        previousSnd = 0;
    }

    /**
    * Another constructor that takes two parameters
    */
    public MemoryPair(int fst, int snd){
        super(fst, snd);
        previousFst = 0;
        previousSnd = 0;
    }

    /**
    * It changes the x
    * @param v  the new value
    */
    @Override
    public void setFst(int v){
        previousFst = getFst();
        super.setFst(v);
    }

    /**
       * It changes the y
       * @param v  the new value
       */
       @Override
       public void setSnd(int v){
           previousSnd = getSnd();
           super.setSnd(v);
       }

       /**
       *   It recovers the old values of x and y
       */
       public void recover(){
           setFst(previousX);
           setSnd(previousY);
       }

}// end of class
```

Agrega nuevos atributos

Llama al constructor de la super clase

Sobre escribe (overrides) el método de la super clase

Ejemplo de Uso

Veamos un ejemplo de uso de ambos:

```
public class Main {
    public static void main(String[] args) {
        Pair p = new Pair(5,6);
        System.out.println("p Fst:"+p.getFst());
        System.out.println("p Snd:"+p.getSnd());
        MemoryPair mp = new MemoryPair(3,4);
        mp.setFst(10);
        mp.setSnd(10);
        System.out.println("mp Fst:"+mp.getFst());
        System.out.println("mp Snd:"+mp.getSnd());
        mp.recover();
        System.out.println("mp Fst:"+mp.getFst());
        System.out.println("mp Snd:"+mp.getSnd());
        p = mp; // se puede hacer porque mp es una subclase
        // mp = p no se puede hacer!
    }
}
```

Clases Genéricas

Con genericidad podemos hacer clases polimorficas. Es decir, que puedan soportar diferentes tipos de datos.

```
public class Pair<T,G>{
    private T fst; // Type T is a parameter
    private G snd; // Type G is a parameter


    /**
     * Constructor of the class
     * @param x the first component
     * @param y the second component
     * @precondition true
     * @postcondition creates the new object
     **/
    public Pair(int fst, int snd){
        this.fst = fst;
        this.snd = snd;
    }

    /**
     * Get the first coordinate
     * @return the first value of the pair
     * @precondition true
     * @postcondition return the first item
     **/
    public T getFst(){
        return this.fst;
    }

    /**
     * Get the second coordinate
     * @return the second value of the pair
     * @precondition true
     * @postcondition return the second value.
     **/
    public G getSnd(){
        return this.snd;
    }

    /**
     * Change the first coordinate
     * @param x change the first value
     * @precondition true
     * @postcondition change x
     **/
    public void setFst(T fst){
        this.fst = fst;
    }

    /**
     * Change the second coordinate
     * @param y change the second value
     * @precondition true
     * @postcondition change the second value
     **/
    public void setSnd(G snd){
        this.snd = snd;
    }

}
```

Parametrizamos la clase con tipos

Clases Genéricas II

Ahora podemos tener pares de cualquier clase definida:

```
public class Main {
    public static void main(String[] args) {
        Pair<Integer,String> p1 = new Pair(5,"Hola");
        Pair<Integer,Integer> p2 = new Pair(5,6);
        Pair<String,String> p3 = new Pair("Hola","Mundo");
        System.out.println("P1 Fst:"+p1.getFst());
        System.out.println("P1 Snd:"+p1.getSnd());
    }
}
```

En el momento que declaramos los objetos le damos los tipos (parámetros actuales) a las clases

Las clases genéricas permiten introducir polimorfismo, reuso de código.

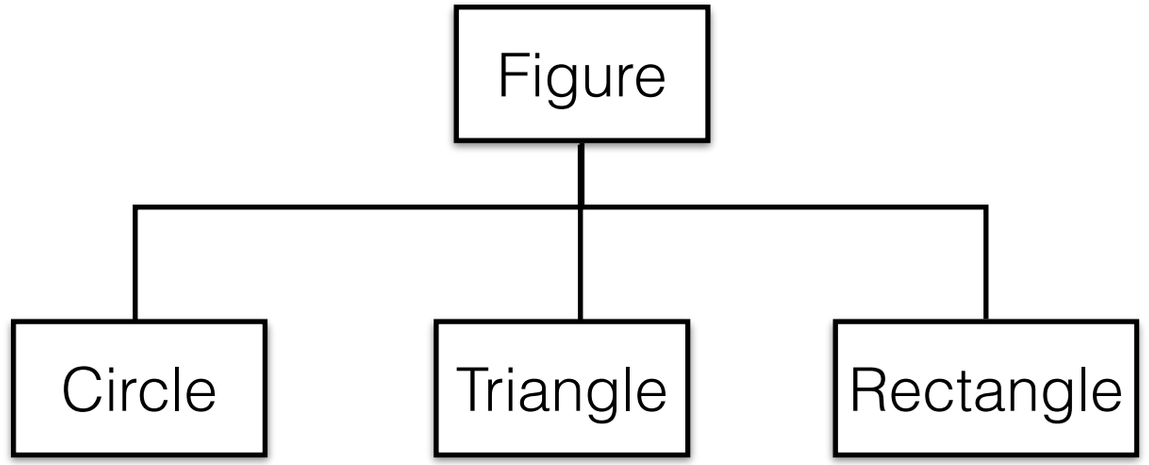
Además introducen seguridad de tipos en el código.

Clases Abstractas

Las clases abstractas son clases que tienen métodos sin implementación

```
abstract class Figure {
    int x, y;
    ...
    void moveTo(int newX, int newY) {
        ...
    }
    abstract void draw();
    abstract void resize();
}
```

Tienen que ser implementadas por las subclases



Interfaces

Las interfaces son parecidas a los TADs

- Solo proveen métodos abstractos, no hay implementaciones
- Los métodos tienen que ser implementados por las subclases
- Pueden ser genéricas
- Pueden pensarse como una especificación de un tipo de datos.

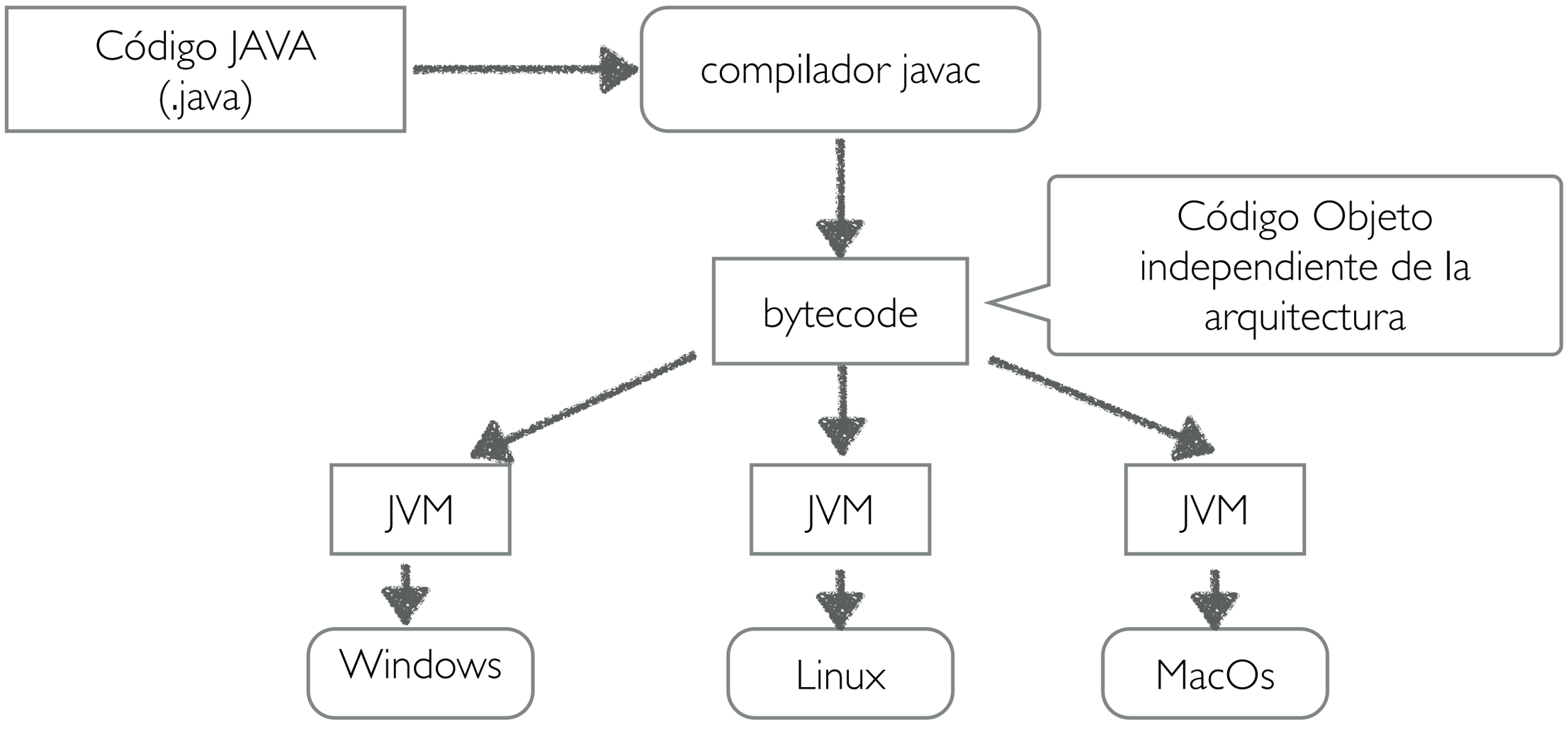
Implementando TADs en OO

```
public interface Pila<T>{
    /** apila un item al tope de la pila
    *   @pre true
    *   @param item el elemento ha ser apilado
    *   @throws OverflowException si la pila está llena
    */
    public void apilar(T item) throws OverflowException;

    /** desapila el elemento al tope de la lista.
    *   @pre: la pila no es vacia
    *   @throws UnderflowException si la pila es vacia
    */
    public void desapilar() throws UnderflowException;

    /** retorna el elemento al tope de la pila
    * @pre: la pila no es vacia
    * @return el tope de la pila
    * @throws UnderflowException si la pila es vaciaRe
    */
    public T tope() throws UnderflowException;

    /** Dice si la pila es vacia o no
    * Pre: true
    */
    public boolean esVacia();

    /** Vacia la pila de elementos
    * @pre: true
    */
    public void vaciar();
}
```

Para implementar TADs, podemos definir el TAD como una interface

Las implementaciones de la interface son las posibles implementaciones del TAD

El Compilador de Java y la Máquina Virtual de Java



La Máquina Virtual de Java