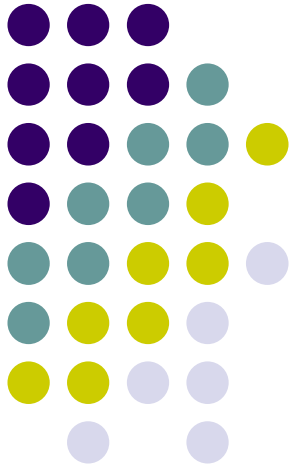
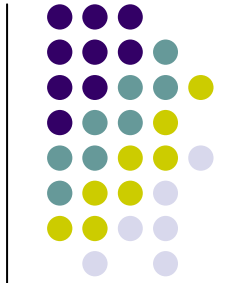
Java



Threads y Collection

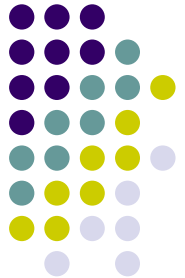
Lic. Claudio Zamoszczyk
claudio@honou.com.ar

Contenidos



- Threads
- Collections
- Otros temas.

Variables estáticas o de clase



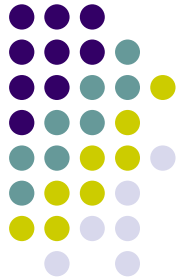
- **Una clase puede tener variables propias de la clase y no de cada objeto. A estas variables se las llama variables de clase o static.**

- **Las variables static se suelen utilizar para definir variables que solo tienen sentido para toda la clase, por ejemplo un contador de objetos creados.**

- **Las variables static son lo mas parecido que posee Java para definir variables globales similares a las de c/c++, vb, etc.**

- **No es necesario crear un objeto para utilizar una variable static.**

Métodos estáticos o de clase



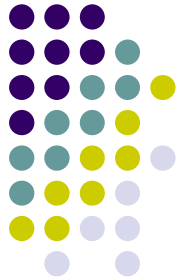
- **A veces se desea crear un método que se utiliza fuera del contexto de cualquier instancia (puede ser accedido sin necesidad de instanciar un objeto del tipo).**

- **Todo lo que se tiene que hacer es declarar estos métodos como static.**

- **Los métodos estáticos sólo pueden llamar a otros métodos static directamente, y no se pueden referir a this o super de ninguna manera.**

Ver ejemplo

Collections

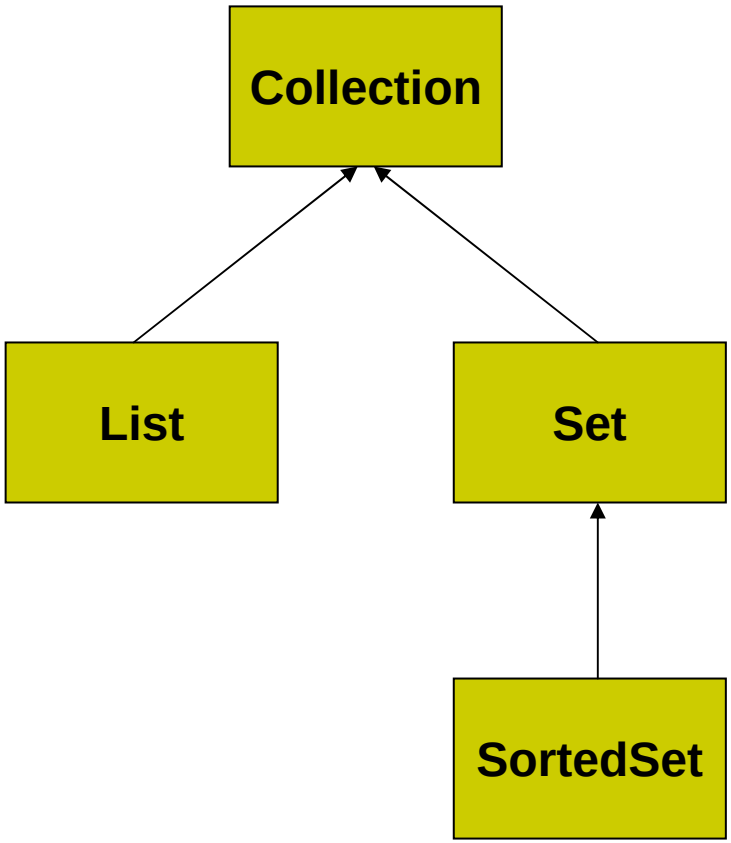


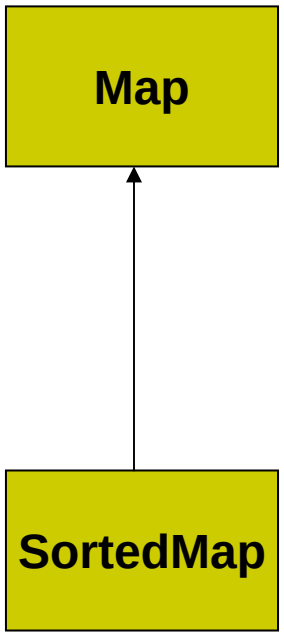
- El API Collections tiene como objetivo fundamental proporcionar una serie de clases e interfaces que nos permitan manejar diversos conjuntos de datos de una manera estandar.

- Dentro de esta API nos vamos a encontrar tanto con estructuras de datos como con algoritmos para utilizar dichas estructuras.

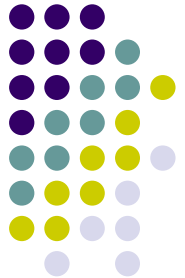
- La base del API Collections está formada por un conjunto de interfaces para trabajar con conjuntos de datos. Entender el funcionamiento de estas interfaces significará entender el funcionamiento de la totalidad de la API.

Collections





Interfaz Collection



Esta interfaz es una de las raíces de la jerarquía. Representa un conjunto de objetos de una manera generalizada. Esta interfaz define una serie de métodos que serán los que el resto de interfaces, o clases que nosotros realicemos, deban implementar

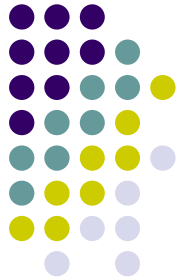
- **Métodos para agregar y elminar elementos**
  - **boolean add(Object element)**
  - **boolean remove(Object element)**

- **Métodos para realizar consultas**
  - **int size()**
  - **boolean isEmpty()**
  - **boolean contains(Object element)**

- **Métodos para recorrer todos los elementos**
  - **Iterator iterator()**

- **Métodos para realizar varias operaciones simultáneamente**
  - **boolean containsAll(Collection collection)**
  - **boolean addAll(Collection collection)**
  - **void clear()**
  - **void removeAll(Collection collection)**
  - **void retainAll(Collection collection)**

Interfaz List

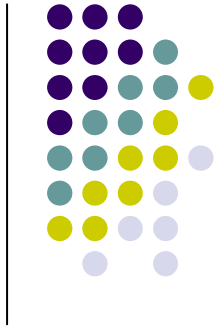


Esta interfaz define un conjunto de datos ordenados permitiendo elementos duplicados. Añade operaciones a nivel de índice de los elementos así como la posibilidad de trabajar con una parte de la totalidad de la lista.

por su posición:

void add(int index, Object element)
boolean addAll(int index, Collection collection)
Object get(int index)
int indexOf(Object object)
int lastIndexOf(Object object)
Object remove(int index)
Object set(int index, Object element)

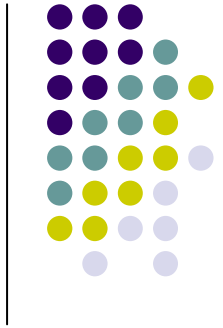
ArrayList



ArrayList es una implementacion concreta de la interface List y representa la estructura de datos utilizando un array de objetos. Este array comienza con un tamaño determinado que podemos especificar en el constructor y va creciendo dinámicamente a medida que vamos añadiendo elementos. Es decir, en el momento en que ya no entren más elementos en nuestro array, la clase ArrayList aumentará su tamaño en un 50% o 100%.

Ver ejemplo.

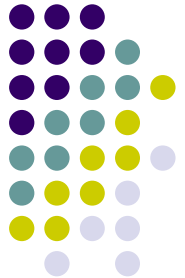
LinkedList



- **Esta lista está implementada utilizando una lista de nodos doblemente enlazada. En este caso para acceder a un elemento determinado deberemos recorrer todos los elementos previos de la lista, por consiguiente el rendimiento en acceso por índice a elementos es bastante peor que en un ArrayList. Por otra parte, en este caso el costo de eliminar o insertar un elemento en una determinada posición se reduce al costo de buscar dicho elemento**

  Con esta clase nos evitamos pues el costo de andar moviendo datos o de tener que aumentar el tamaño de nuestra estructura mientras perdemos rendimiento realizando la búsqueda de los elementos.

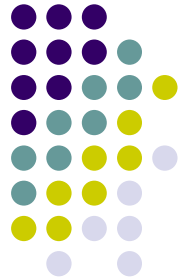
Interfaz Set



Esta interfaz extiende a la interfaz Collection y no añade ningún nuevo método. Representa un conjunto de datos en el más puro estilo matemático, es decir, no se admiten duplicados. Por lo tanto si añadimos un elemento a un conjunto y ese elemento ya se encontraba en su interior no se producirá ningún cambio en la estructura de datos.

Para determinar si un dato se encuentra o no en el conjunto, las implementaciones que vienen en el API de Collections se basan en los métodos equals() y hashCode() de la clase Object. El API de Collections proporciona dos implementaciones de esta interfaz: HashSet y TreeSet.

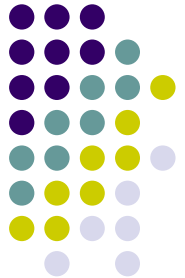
HashSet



Implementa el conjunto de datos utilizando un tabla hash. Los elementos no están ordenados y pueden variar su posición a lo largo del tiempo a medida que se vayan haciendo operaciones de balanceo en la estructura.

Ver Ejemplos

Interfaz Map

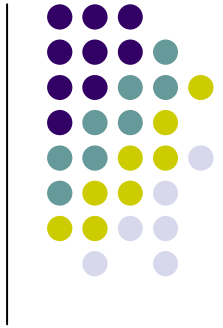


Esta interfaz es la raiz de una nueva jerarquía dentro del API de Collections. Por definición, la interfaz define una estructura de datos que mapeará claves con valores sin permitir claves duplicadas. La clase que implementa la interfaz Map es HashMap.

Modificación

Object put(Object clave, Object valor)
void putAll(Map mapa)
void clear()
Object remove(Object clave)

HashMap



Consulta

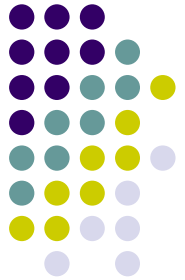
Object get(Object clave)
boolean containsKey(Object key)
boolean containsValue(Object value)
int size()
boolean isEmpty()

Vista

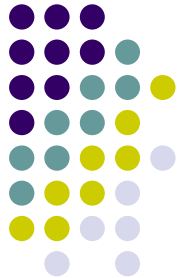
Set keySet()
Collection values()
Set entrySet()

Ver ejemplo

¿Preguntas?

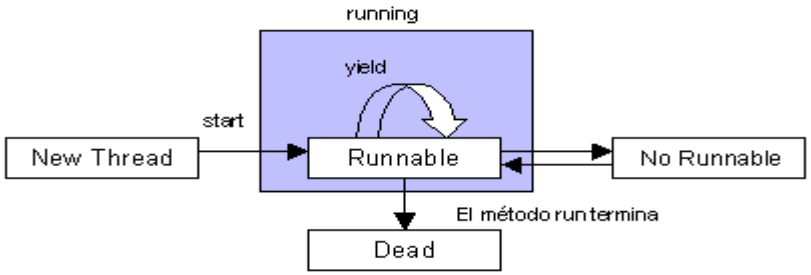


Threads



¿Qué es un thread? Un hilo es un flujo de control secuencial dentro de un programa. Un *único* hilo es similar a un programa secuencial; es decir, tiene un comienzo, una secuencia y un final, además en cualquier momento durante la ejecución existe un sólo punto de ejecución. Sin embargo, un hilo no es un programa; no puede correr por sí mismo, corre *dentro* de un programa. Un hilo por si mismo no nos ofrece nada nuevo. Es la habilidad de ejecutar varios hilos (secuencias de programación) dentro de un programa. Lo que ofrece algo nuevo y útil; es que estos hilos puede ejecutar tareas distintas en paralelo.

El ciclo de vida de un Thread

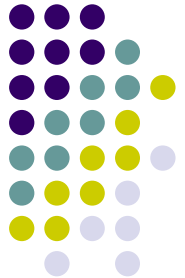


Cuando se instancia la clase Thread se crea un nuevo hilo que está en en su estado inicial ('New Thread'). En este estado es simplemente un objeto más. No existe todavía el thread en ejecución. El único método que puede invocarse sobre él es el método start.

Cuando se invoca el método start sobre el thread el sistem crea los recursos necesarios, lo planifica (le asigna prioridad) y llama al método run. En este momento el thread está corriendo.

Si el método run invoca internamente el método **sleep** o el thread tiene que esperar por una operación de entrada/salida, entonces el thread pasa al estado 'no runnable' hasta que la condición de espera finalice. Durante este tiempo el sistema puede ceder control a otros threads activos.
Por último cuando el método run finaliza el thread termina y pasa a la situación 'Dead'

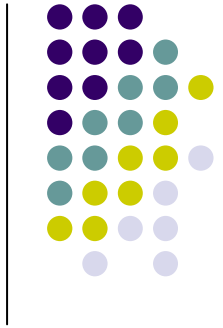
La Interface Runnable



La interfase Runnable proporciona un método alternativo a la utilización de la clase Thread, para los casos en los que no es posible hacer que nuestra clase extienda la clase Thread.

Esto ocurre cuando nuestra clase, que deseamos correr en un thread independiente deba extender alguna otra clase. Dado que no existe herencia múltiple, nuestra clase no puede extender a la vez la clase Thread y otra más. En este caso nuestra clase debe implantar la interface Runnable, variando ligeramente la forma en que se crean e inician los nuevos threads. Ver ejemplo

Prioridades

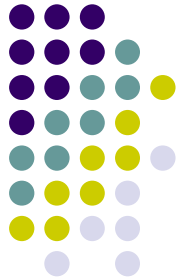


Aunque un programa utilice varios threads y aparentemente estos se ejecuten simultaneamente, el sistema ejecuta una sola instrucción cada vez (esto es particularmente cierto en sistemas con una sola CPU), aunque realizado a velocidad suficiente para proporcionar la ilusión de simultaneidad. En ciertas ocasiones es necesario darle mayor prioridad a la ejecución de un thread con relación al resto.

La prioridad de un thread es un valor entero (cuanto mayor es el número, mayor es la prioridad), que puede asignarse con el método setPriority. Por defecto la prioridad de un thread es igual a la del thread que lo creó. Cuando hay varios threads en condiciones de ser ejecutados, la máquina virtual elige el thread que tiene una prioridad más alta.

Si dos o más threads están listos para ejecutarse y tienen la misma prioridad, la máquina virtual va cediendo control de forma cíclica (round-robin).

Sincronización



Existen situaciones donde varios hilos de ejecución acceden al mismo recurso o un mismo objeto. La utilización de un recurso por más de un hilo puede traer problemas o comportamiento inesperados por el acceso concurrente. Para evitar estos posibles problemas es necesario contar con un sistema que permita que solo un hilo trabaje con un recurso al mismo tiempo. Esto se logra mediante la sincronización de métodos o de secciones de código.

public synchronized void metodoXX()…

```
synchronized(this) {
        contador++;
}
```

Ver Ejemplo

¿Preguntas?