**2014 Big Java**

**Eligiendo una colección**

* Supongamos que necesitamos guardar objetos en una colección. Hay diferentes tipos de colecciones. A continuación, se mencionan unos puntos para elegir una colección apropiada para nuestra aplicación.

**Paso 1: Determinar cómo accederemos a los valores:** Almacenamos valores en una colección para que después podamos recuperarlos. ¿Cómo queremos acceder a los valores individuales? Tenemos varias opciones:

* + Los valores son accesados por una posición integer. Utilizar un **ArrayList**.
  + Los valores son accesados por una clave que no es parte de un objeto. Utilizar un **Map**.
  + Los valores son accesados solamente en uno de los dos extremos. Usar una **cola** (queue: first in, first out – primero en entrar, primero en salir) o una **pila** (last in, first out – ultimo en entrar, primero en salir)
  + No necesitas acceder a los valores individuales por posición. Refina tu elección en los pasos 3 y 4.

**Paso 2: Determinar los tipos de elementos o los tipos clave-valor**:

* Para una **List** o **Set**, debemos determinar el tipo de elementos que queremos guardar. Por ejemplo, si coleccionamos un conjunto de libros, entonces el tipo de elemento es Libro.
* Similarmente, para un **map**, determinar los tipos de claves y los valores asociados. Si queremos buscar libros por ID, podemos usar un Map<Integer, Libro> o Map<String, Book>, dependiendo del tipo del ID.

**Paso 3: Determinar si importa el orden del elemento o la clave:** cuando accedemos/visitamos los elementos de una colección o claves de un mapa ¿Te importa el orden en que son accesados/visitados? Tenemos varias opciones:

* Los elementos o claves deben ser ordenados. Usar un **TreeSet** o **TreeMap**. Ir al paso 6.
* Los elementos deben estar en el mismo orden en el que fueron insertados. Ahora, la elección se reduce a una **LinkedList** o **ArrayList**.
* No importa, siempre y cuando visites todos los elementos, no importa en qué orden.Si eligen un **map**  en el paso 1, usa un **HashMap** y ve al paso 5.

**Paso 4: Para una colección, determinar qué operaciones deben ser eficientes:** tenemos varias opciones.

* La búsqueda de elementos debe ser eficiente. Utilizar un **HashSet.**
* Debe ser eficiente agregar o eliminar elementos al principio u otra posición, siempre que ya esté inspeccionando un elemento allí. Usar una **LinkedList**.
* Solamente insertaremos o eliminaremos al final, o coleccionamos tan pocos elementos que no nos preocupamos por la velocidad. Usar un **ArrayList**.

**Paso 5: Para Hash Sets y Maps, decidir si necesitamos implementar los métodos hashCode y equals.**

* Si nuestro elemento o clave pertenece a una clase que alguien más implementa, verificar si la clase tiene sus propios métodos **hashCode** y **equals**. Si es así, ya está todo listo. Este es el caso para la mayoría de las clases de la librería estándar de Java como String, Integer, Rectangle y así.
* De lo contrario, decida si puede comparar los elementos por identidad. Esto es el caso si nunca construyes dos elementos distintos con los mismos contenidos. En ese caso, no necesitas hacer nada. Los métodos **hashCode** y **equals** de la clase Object son apropiados.
* De lo contrario, debemos implementar nuestros propios métodos **equals** y **hashCode**.

Paso 6: Si usas un Tree, decidir si proporcionar un comparador.

* Mirar la clase de conjunto de elementos o mapa de claves. ¿La clase implemente la interface **Comparable**? Si es así ¿el orden de clasificación dado por el método compareTo es el que desea? Si es así, entonces no necesitas hacer nada más. Este es el caso para muchas clases de la biblioteca estándar, en particular para String e Integer.
* Si no, entonces tu clase elemento debe implementar la interface **Comparable**, o debemos declarar una clase que implemente la interface **Comparator**.