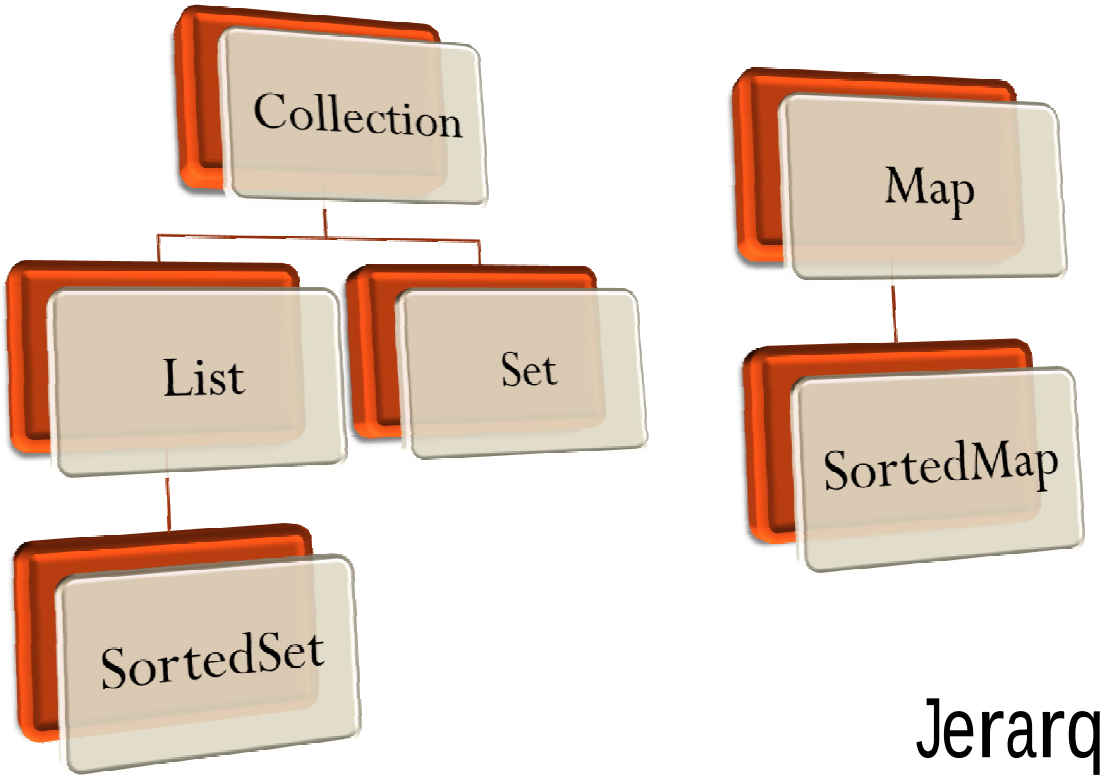
Módulo IV

Colecciones y Excepciones

Colecciones

- Java provee las clases necesarias para manipular múltiples objetos.
- El framework de colecciones provee una API para el manejo de diferentes tipos de colecciones:
  - Con elementos duplicados
  - Sin elementos duplicados
  - Indizados.
  - Ordenados.
  - Hashes.
  - Con y sin repeticiones

Colecciones



Jerarquía de interfaces

Collection

- Collection: Provee el protocolo básico para la manipulación de colecciones de elementos.
- La plataforma Java no provee una implementación directa para la interface Collection, pero si para sus
- subinterfaces Set, List y Queue

```
public interface Collection<E> extends Iterable<E> {
        int size();
        boolean isEmpty();
        boolean contains(Object o);
        Iterator<E> iterator();
        Object[] toArray();
        boolean add(E e);
        boolean remove(Object o);
        boolean containsAll(Collection<?> c);
        boolean addAll(Collection<? extends E> c);
        boolean removeAll(Collection<?> c);
...
}
```

List

- Los elementos poseen un Orden.
- Permite duplicados.
- Vector la implementa.

```
boolean add(index:int, el:Object);
boolean addAll(index:int, col:Collection);
Objetc get(index:int);
int  indexOf(element:Object);
Boolean remove(index:int);
List subList(from:int, to:int);
```

Set

- Representa a un conjunto de elementos.
- No permite duplicados.
- Los elementos no poseen orden.

```
public interface Set<E> extends Collection<E> {
   // Query Operations
int size();
boolean isEmpty();
boolean contains(Object o);
Object[] toArray();
<T> T[] toArray(T[] a);
   // Modification Operations
boolean add(E e);
boolean remove(Object o);
...
}
```

SortedSet

- Igual que Set pero los elementos estan ordenados.
- Los elementos pueden ser comparados
  - Comparator<? **super E> comparator();**

Map

- Define colecciones donde los elementos se agrupan por pares (clave, valor).

```
public interfaceMap <K,V> {
   V put(K clave, V valor);
   V get(K clave);
   V remove(K clave);
   boolean containsKey(K clave);
   boolean containsValue(V valor);
   int size();
   boolean isEmpty();
   void putAll(Map<? extends K, ? extends V> t);
   void clear();
   Set<K> keySet();
   Collection<V> values();
   Set<Map.Entry<K, V>> entrySet(); … }
```

Recorrer una colección

- **For each**

```
for (String iterador : coleccion) {
    System.out.println(iterador)
}
```

- Recorre toda la colección.
- Iterador es una variable que toma los valores de los objetos contenidos en la colección en cada iteración.

Recorrer una colección

- Usando la interfaz Iterator
  - Un objeto Iterator, permite recorrer una colección y remover elementos selectivamente si se desea. Es posible obtener un iterador para una colección, invocando al método **iterator().**

```
Iterator it= lista.iterator();
while (it.hasNext()){
   System.out.println(it.next());
                }
```

Clases Concretas de Colecciones:

Interface	**Implementación**				**Históricas**
	Con hashing	Tamaño var.	Arbol bal.	Lista enlaz.	
Set	HashSet		TreeSet		
List		ArrayList		LinkedList	Vector Stack
Map	HashMap		TreeMap		Hashtable Properties

Conclusiones

- Agrupar y relacionar multiples objetos en una estructura de longitud variable.

- Acepta cualquier objeto dependiendo del tipo.
- No acepta tipos primitivos sin embargo Java realiza Autoboxing.

- Las implementaciones más usadas:
  - Vector, funciona como un arreglo pero sin longitud fija
  - Hashtable, guarda pares de objetos (clave-valor), permite acceso directo por clave

Excepciones

- Representan situaciones anómalas que pueden ocurrir durante la ejecución de un programa y que es importante tratarlas.
- Se basa en la utilización de la clase Exception que extiende a Throwable.
- Throwable provee:
  - Un slot para un mensaje
  - El registro de la pila de ejecución.

Cuando se dispara una excepción

1. Se crea un objeto excepción en la heap, con el operador **new**, como cualquier objeto Java.
2. Se interrumpe la ejecución y el objeto excepción es expulsado del contexto actual mediante la palabra reservada **throw**.
3. En este punto, comienza a funcionar el mecanismo de manejo de errores: buscar un lugar apropiado donde continuar la ejecución del programa.
4. El lugar apropiado es el manejador de excepciones, cuya función es recuperar el problema.

Excepciones

- RuntimeException vs Exception
  - Las RuntimeExceptions no necesita ser declaradas ni manejadas explícitamente. No se tiene control sobre ellas.
    - Ej: ArrayIndexOutOfBoundsException, NullPointerException, IllegalArgumentException
  - Las Exceptions tiene que ver con el dominio de la aplicación. Deben ser declaradas y manejadas explícitamente.
    - IOException, MalformedURLException

Excepciones - Declaración

- Crear una sublcase de Exception.
- La convención es utilizar un nombre descriptivo delante de la palabra Exception.

```
package nombrePaquete;

{importaciones}

[modificadores] class nombreException extends Exception {

}
```

¿Cómo se generan excepciones?

- Implícitamente: el programa hace algo ilegal
- Explícitamente: usando la sentencia **`throw`**.

```
class OutOfMoneyException extends Exception { }
class BankAccount {
      ...
  if ( credit < amount ) throw new OutOfMoneyException();
      ...
  }
```

Excepciones - Manejo

`try` `block` debe haber por lo menos una de las dos opciones siguientes:

[**`catch`** `(arg) block]` puede haber 0 ó más de esta sentencias

[**`finally`** `block]` puede haber 0 ó 1 de esta sentencia

- block es una sentencia simple o un grupo de sentencias entre llaves

Excepciones

- Ejemplo:

```
public Object pop() throws SinElementosException {
  if (this.getElementos().isEmpty()) {
    throw new SinElementoException();
  } else {
    return this.getElementos().removeLast();
  }
}

...
try {
  elemento = pila.pop();
} catch (SinElementosException e) {
  e.printStackTrace();
}
```

Excepciones - Conclusiones

- Sobre el uso de excepciones:
  - Una condición de error debe ser tratada solamente donde tenga sentido.
  - Al comienzo el código puede ser escrito como si todas las operaciones funcionaran correctamente y luego agregar la lógica de excepciones.
  - En las sentencias catch deben declararse de más específica a más general.
  - Los bloques catch **NO** se deben dejar vacíos ni sólo imprimir el *stack trace (pila de ejecución)*.