Estructura de Datos y Algoritmos

ITBA 2024-Q1

Java es un lenguaje estáticamente tipado => hay que declarar el tipo de una variable antes de usarla.

Sin Generics, los casteos son una posibilidad de errores que se detectan en tiempo de ejecución.

Ej:

```
List v = new ArrayList();
v.add("test");
Integer i = (Integer)v.get(0); // Runtime Exception!
```

Ej: sin casteos, también podemos tener RuntimeException. Los arreglos en Java sin Generics son covariantes => puedo poner elementos de un subitpo.

```
Object[] elems = new String[2];
elems[0] = "hi";
elems[1] = 100;    // RuntimeException!
```

En Java hay que declarar el tipo de una variable antes de usarla.

Con la introducción de Java Generics ese “tipo” puede parametrizarse. Generics esta pensado para parametrizar y minimizar errores.

Técnicamente hablando, **Generics fue implementado usando la Técnica de Erasure.**

La técnica consiste en reemplazar todo tipo de parámetro con su “bound/restricción” y si no lo hay lo reemplaza por Object.

De ser necesario realiza casteos.

Ej:

```
public class P<T> {
public void method(T p) {
…
}
}
```

<T> is unbound => Object

Ej:

```
public class P<T extends Comparable<T>> {

   public void method(T   p)
  {
     …
   }
}
```

<T> is bound => Comparable

En Java los Generics son invariantes => no se puede asignar un subtipo generics a un supertipo generics

Ej: ni compila

```
ArrayList<Integer> ints = new ArrayList<Integer>();
List<Number> numbers = ints;
```

Ej: ni compila

```
public class P<T> {
..}

public static void main(String[] args) {
        P<Integer> myi = new P<Integer>();
        P<Number> myp = myi;
}
```

Hay muchas restricciones que se establecieron al diseñar en Java Generics y Erasure. Leer

https://docs.oracle.com/javase/tutorial/java/generics/restrictions.html

Ej: Probar

Ej: Probar

Solución opción 1: guardar un arreglo de Objects (no T). Castear cuando sea necesario.
Escribamos entre todos la clase P<E>

```
public class P<E> {

private Object[] arreglo;

public void initialize(int dim) {
      …
}

public void setElement(int pos, Eelement) {
      …
}

public E getElement(int pos)
{
   …
}
}
```

Caso de Uso:

```
P<Number> auxi = new P<>();
auxi.initialize(5);
auxi.setElement(3, 10);
auxi.setElement(2, 20.8);
for (int i= 0; i < 5; i++) {
System.out.println( auxi.getElement(i) );
}
```

```java
package test;

public class P1<E> {

    private Object[] arreglo;

    public void initialize(int dim) {
        arreglo= new Object[dim];
    }

    public void setElement(int pos, E element) {
        arreglo[pos]= element;
    }

    @SuppressWarnings("unchecked")
    public E getElement(int pos)
    {
        return (E) arreglo[pos];
    }

    public static void main(String[] args) {
        P1<Number> auxi = new P1<>();
        auxi.initialize(5);
        auxi.setElement(3, 10);
        auxi.setElement(2, 20.8);
        for (int i= 0; i < 5; i++) {
        System.out.println( auxi.getElement(i) );
        }
    }
}
```

```
null
null
20.8
10
null
```

Solución opción 2: Usar reflection
Escribamos entre todos la clase P<T>

```
public class P<T> {

private T[] arreglo;

public void initialize(int dim, Class<T> theClass) {
      …
}

public void setElement(int pos, T element) {
      …
}

public T getElement(int pos)
{
   …
}
}
```

Caso de Uso:

```
P<Number> auxi = new P<>();
auxi.initialize(5, Number.class);
auxi.setElement(3, 10);
auxi.setElement(2, 20.8);
for (int i= 0; i < 5; i++) {
System.out.println( auxi.getElement(i) );
}
```

```java
package test;

import java.lang.reflect.Array;

public class P2<E> {

    private E[] arreglo;

    @SuppressWarnings("unchecked")
    public void initialize(int dim, Class<E> theClass) {
        arreglo= (E[]) Array.newInstance(theClass, dim);
    }

    public void setElement(int pos, E element) {
        arreglo[pos]= element;
    }

    public E getElement(int pos)
    {
        return arreglo[pos];
    }

    public static void main(String[] args) {
        P2<Number> auxi = new P2<>();
        auxi.initialize(5, Number.class);
        auxi.setElement(3, 10);
        auxi.setElement(2, 20.8);
        for (int i= 0; i < 5; i++) {
            System.out.println( auxi.getElement(i) );
        }
    }

}
```

```
null
null
20.8
10
null
```

Solución 3. Crear un arreglo de un tipo conocido (que soporte todos) y castear

```
public class PObjectToT<E> {

   private E[] arreglo;

  @SuppressWarnings("unchecked")
  public void initialize(int dim) {

  …
}

public void setElement(int pos, E element) {
…
}

 public E getElement(int pos) {
…
}
}
```

Caso de Uso:

```
P<Number> auxi = new P<>();
auxi.initialize(5);
auxi.setElement(3, 10);
auxi.setElement(2, 20.8);
for (int i= 0; i < 5; i++) {
System.out.println( auxi.getElement(i) );
}
```

```java
package test;

public class P3<E> {

private E[] arreglo;

    @SuppressWarnings("unchecked")
    public void initialize(int dim) {
        arreglo= (E[]) new Object[dim];
    }

    public void setElement(int pos, E element) {
        arreglo[pos]= element;
    }

    public E getElement(int pos)
    {
        return arreglo[pos];
    }

    public static void main(String[] args) {
        P3<Number> auxi = new P3<>();
        auxi.initialize(5);
        auxi.setElement(3, 10);
        auxi.setElement(2, 20.8);
        for (int i= 0; i < 5; i++) {
        System.out.println( auxi.getElement(i) );
        }
    }

}
```

```
null
null
20.8
10
null
```

Qué pasa si en esas 3 opciones cambiamos

public class PCasoN<E> {

Por

public class PCasoN<E extends Comparable<E> {

Explicar…

TP 3A- Ejer 4

Implementarlo el Indice con Generics

Cambiar int por generics. Implementar el índice con esta versión de interface:

```
package eda;

public interface IndexParametricService <T extends Comparable<? super T>>{

        // elements serán los valores del índice, los anteriores se descartan
    // lanza excepction si elements is null o si alguno de los elementos del
    // arreglo proporcionado son null
    void initialize(T [] elements);

   // busca una key en el índice, O(log2 N)
   boolean search(T key);

        // inserta el key en pos correcta. Crece automáticamente de a chunks.
    // si el valor proporcionado es null, ignora el pedido.
    void insert(T key);

        // borra el key si lo hay, sino lo ignora.
    // decrece automáticamente de a chunks
    void delete(T key);
```

```
 // devuelve la cantidad de apariciones de la clave especificada.
int occurrences(T key);

 // devuelve un nuevo arreglo ordenado con los elementos que pertenecen
 // al intervalo dado por leftkey y rightkey.  Si el mismo es abierto/cerrado depende
 // de las variables leftIncluded  y rightIncluded. True indica que es cerrado. El valor
// devuelto será un arrego de length 0 si no hay elementos que satisfagan al condicion
  T[] range(T leftKey, T rightKey, boolean leftIncluded, boolean rightIncluded);


 // imprime el contenido del índice ordenado por su key
 void sortedPrint();


 // devuelve el máximo elemento del índice o null si no hay elementos
 T getMax();

 // devuelve el mínimo elemento del índice o null si no hay elementos
 T getMin();

}
```

Caso de Uso:

```
IndexParametricService<Integer>  myIndex= new IndexWithDuplicates<>(Integer.class);
Integer[] rta = myIndex.range(10, 50, true, true);

myIndex.initialize( new Integer[] {100, 50, 30, 50, 80});
rta = myIndex.range(10, 50, true, true);


IndexParametricService<String>  anIndex=  new IndexWithDuplicates<>(String.class);
String[] rta2 = anIndex.range("hola", "tal", true, true);

anIndex.initialize( new String[] {"hola", "ha", "sii" });
rta2 = anIndex.range("a", "b", true, true);
rta2 = anIndex.range("a", "quizas", true, true);
```