Programación Orientada a Objetos

Clases Genéricas

Ing. Aníbal Sardón Paniagua

C16290@utp.edu.pe

anibal.sardon@hotmail.com

Logros de Aprendizaje

Al finalizar la sesión el estudiante:

- Descubre las clases genéricas en Java mediante los conceptos de la programación orientada a objetos.
- Aprende a utilizar las clases genéricas mediante ejercicios en Java.

Universidad Tecnológica del Perú

Recordamos la sesión anterior

- ¿Qué son las colecciones?.
- ¿Cuáles son las principales colecciones?

- ¿Dudas de la sesión anterior?

Saberes previos

¿Qué vemos en las imágenes?

¿Cuál sería la interface?

¿Cuál es la importancia del tema?

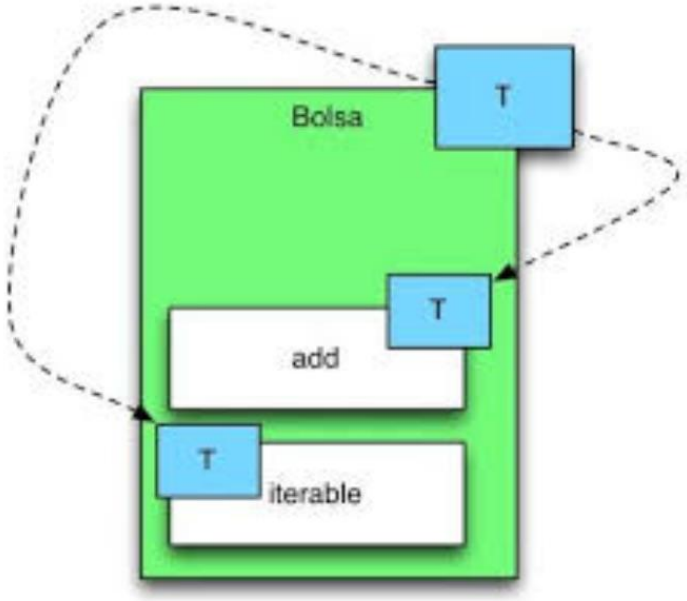
Universidad Tecnológica del Perú

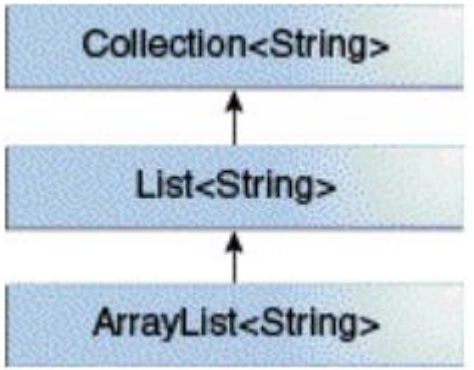
Temas a tratar

- Clases genéricas
- Restricciones de tipos genéricos

Clases Genéricas

- El término genéricos significa tipos parametrizados.
- Los tipos parametrizados son importantes porque le permiten crear clases, interfaces y métodos en los que el tipo de datos sobre los que operan se especifica como parámetro.
- Una clase, interfaz o método que funciona con un tipo de parámetro se denomina genérico, como una clase genérica o método genérico.





Clases Genéricas

- Los genéricos son clases, estructuras, interfaces y métodos que tienen marcadores de posición *(parámetros de tipo)* para uno o varios de los tipos que almacenan o utilizan.
- Un método genérico puede usar su **parámetro de tipo** como el **tipo de su valor devuelto** o como **el tipo de uno de sus parámetros** formales.

```java
class ClaseGenerica<T>
{
  T obj;

  public ClaseGenerica(T o)
  {
    obj = o;
  }

  public void classType()
  {
    System.out.println("El tipo de T es " + obj.getClass().getName());
  }
}
```

```java
public class MainClass
{
  public static void main(String args[])
  {
    // Creamos una instancia de ClaseGenerica para Integer.
    ClaseGenerica<Integer> intObj = new ClaseGenerica<Integer>(88);
    intObj.classType();

    // Creamos una instancia de ClaseGenerica para String.
    ClaseGenerica<String> strObj = new ClaseGenerica<String>("Test");
    strObj.classType();

  }
}
```

Clases Genéricas

- Este tipo genérico asumirá el tipo de dato que realmente le pasaremos a la clase.

```
public class ClaseGenerica<T> {
    private T atributo;

    public ClaseGenerica(T atributo) {
        this.atributo = atributo;
    }

    public void classType() {
        System.out.println("El tipo de T es "+ atributo.getClass().getName());
    }
}
```

```
public static void ejemploClaseGenerica() {
    //Creamos una instancia de ClaseGenerica para Integer
    ClaseGenerica<Integer> intObj = new ClaseGenerica<Integer>(88);
    intObj.classType();

    //Creamos una instancia de ClaseGenerica para String
    ClaseGenerica<String> strObj = new ClaseGenerica<String>("Test");
    strObj.classType();
}
```

Clases Genéricas

- **T** es el tipo genérico que será reemplazado por un tipo real.
- **T** es el nombre que damos al parámetro genérico.
- Este nombre se **sustituirá por el tipo real** que se le pasará a la clase.
- Hay que tener en cuenta que los **generics** de **java** solo funcionan con objetos.

```java
public class ClaseGenerica<T> {
    private T atributo;

    public ClaseGenerica(T atributo) {
        this.atributo = atributo;
    }

    public void classType() {
        System.out.println("El tipo de T es "+ atributo.getClass().getName());
    }
}
```

```java
public static void ejemploClaseGenerica() {
    //Creamos una instancia de ClaseGenerica para Integer
    ClaseGenerica<Integer> intObj = new ClaseGenerica<Integer>(88);
    intObj.classType();

    //Creamos una instancia de ClaseGenerica para String
    ClaseGenerica<String> strObj = new ClaseGenerica<String>("Test");
    strObj.classType();
}
```

- El código siguiente nos mostrará un error:

```java
ClaseGenerica<int> myOb = new ClaseGenerica<int>(53); // Error, can't use
primitive type
```

Clases Genéricas

Convención de Nombrado

- Por convenio, los nombres de los parámetros son letras mayúsculas
- Los nombres de los tipos de parámetros usados habitualmente son:
  - **E** - **Element** (usado en Java Collections Framework)
  - **K** – **Key** (Llave, usado en mapas)
  - **N** – **Number** (para números)
  - **T** – **Type** (Representa un tipo, es decir, una clase)
  - **V** – **Value** (representa el valor, también se usa en mapas)
  - **S,U,V** etc. – usado para representar otros tipos.

Restricción de Tipos Genéricos

Universidad Tecnológica del Perú

- Es posible restringir un tipo genérico para trabajar con un tipo específico.

```
package GenericoRestriccion;

public class CajaNumeros<N extends Number> {
    private N atributoNumerico;

    public N getAtributoNumerico() {
        return atributoNumerico;
    }
    public void setAtributoNumerico(N atributoNumerico) {
        this.atributoNumerico = atributoNumerico;
    }
}
```

```
public static void main(String[] args) {
    CajaNumeros<Double> cajaDoble = new CajaNumeros<>();
    cajaDoble.setAtributoNumerico(new Double(80.0));
    CajaNumeros<Integer> cajaInteger = new CajaNumeros<>();
    cajaInteger.setAtributoNumerico(999);
    CajaNumeros<int> cajaInt = new CajaNumeros<>();       //Incorrecto por no ser clase
    CajaNumeros<String> cajaString = new CajaNumeros<>(); //Incorrecto por restricción
}
```

Tipos Genéricos - Ejemplo

- Se escribe la interfaz Operable, con un **tipo genérico N.** Todos los métodos de esta interfaz reciben un objeto de **tipo N** y devuelven un objeto de **tipo N.**

```
package InterfacesClasesGenericas;

public interface Operable<N extends Number> {
    public N suma(N operando1,N operando2);
    public N resta(N operando1,N operando2);
    public N producto(N operando1,N operando2);
    public N division(N operando1,N operando2);
}
```

Tipos Genéricos - Ejemplo

```java
package InterfacesClasesGenericas;

public class OperacionesMatBInteger implements Operable<Integer>{
    @Override
    public Integer suma(Integer operando1, Integer operando2) {
        return operando1 + operando2;
    }
    @Override
    public Integer resta(Integer operando1, Integer operando2) {
        return operando1 - operando2;
    }
    @Override
    public Integer producto(Integer operando1, Integer operando2) {
        return operando1 * operando2;
    }
    @Override
    public Integer division(Integer operando1, Integer operando2) {
        return operando1 / operando2;
    }
}
```

Tipos Genéricos - Ejemplo

```java
package InterfacesClasesGenericas;

public class OperacionesMatBDouble implements Operable<Double>{

    @Override
    public Double suma(Double operando1, Double operando2) {
        return operando1 + operando2;
    }

    @Override
    public Double resta(Double operando1, Double operando2) {
        return operando1 - operando2;
    }

    @Override
    public Double producto(Double operando1, Double operando2) {
        return operando1 * operando2;
    }

    @Override
    public Double division(Double operando1, Double operando2) {
        return operando1 / operando2;
    }

}
```

Tipos Genéricos - Ejemplo

```
package InterfacesClasesGenericas;

public class Prueba {
    public static void main(String[] args) {
        OperacionesMatBInteger operacionesMatBInteger = new OperacionesMatBInteger();
        System.out.println(operacionesMatBInteger.suma(1, 100));
        OperacionesMatB<Integer> operacionesMatB = new OperacionesMatB<>();
        System.out.println(operacionesMatB.suma(1, 100));

    }
}
```

Preguntas

Ejercicio

- Completar la clase **“OperacionesMatB”** utilizando la interfaz genérica

```java
package InterfacesClasesGenericas;

public class OperacionesMatB<N extends Number> implements Operable<N>{

    @Override
    public N suma(N operando1, N operando2) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public N resta(N operando1, N operando2) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public N producto(N operando1, N operando2) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public N division(N operando1, N operando2) {
        // TODO Auto-generated method stub
        return null;
    }
}
```

Preguntas

Manos a la Obra, a programar ….

Conclusiones

- ¿Qué aprendiste en esta sesión?
- Te invitamos a compartir tus conclusiones en clase.

Resumen

1. Los genéricos son clases, estructuras, interfaces y métodos que tienen marcadores de posición (parámetros de tipo) para uno o varios de los tipos que almacenan o utilizan.
2. Un método genérico puede usar su parámetro de tipo como el tipo de su valor devuelto o como el tipo de uno de sus parámetros formales.
3. Por convenio, los nombres de los parámetros son letras mayúsculas Los nombres de los tipos de parámetros usados habitualmente son:
    1. E - Element (usado en Java Collections Framework)
    2. K – Key (Llave, usado en mapas)
    3. N – Number (para números)
    4. T – Type (Representa un tipo, es decir, una clase)
    5. V – Value (representa el valor, también se usa en mapas)
    6. S,U,V etc. – usado para representar otros tipos.


UTP
Universidad
Tecnológica
del Perú