Java Generics

UTN – Facultad Regional Paraná

Titular: Ernesto Zapata Icart

Aux: Mariano Carpio

Generics – Propósito de su implementación

- **Generics** en Java es la implementación de **Programación Genérica**.
- La **programación genérica** es un tipo de programación centrada más en los algoritmos que en los datos. La idea principal es generalizar las funciones utilizadas para que puedan usarse en más de una ocasión.
- Esto se consigue parametrizando lo máximo posible el desarrollo del programa.
- La biblioteca de funciones conseguida con esta manera de programar permite que esas funciones puedan servir para más programas; y también aplicando pocos cambios, conseguir que realice diferentes acciones.

Generics en Java

- **Generics** permite al programador abstraerse de los tipos de datos.
- **Generics** es aplicable a **Collections**. Dado que las colecciones pueden contener objetos de cualquier tipo, con este mecanismo se logra restringir el tipo de objetos que una colección pude almacenar.
- **Generics** permite identificar errores “en tiempo de ejecución” (runtime errors) cuando estamos “en tiempo de compilación”.

Generics en Java

```
List v = new ArrayList();
v.add(new String("test"));
Integer i = (Integer) v.get(0);  // Runtime error . Cannot cast from String to Integer
```

- Este error solo es detectable en tiempo de ejecución.

```
List<String> v = new ArrayList<String>();
v.add(new String("test"));
Integer i = v.get(0);  // Compile time error. Converting String to Integer
```

- El error en tiempo de compilación sucede cuando intentamos asignar un tipo String a una variable del tipo Integer.
- Un ventaja notable es que no necesitamos hacer el casting explicito cuando usamos el método get.
- Generics también soporta el uso de Interfaces.

Generics y clases parametrizables

- La mayoría de las clases que pertenecen a Collections son clases parametrizables.
- Ej: la clase **ArrayList** es una clase parametrizable.
- Esta clase tiene un parámetro que puede ser reemplazado por una referencia a cualquier tipo de clase. Esto permite tener un arreglo de un tipo específico.
- El tipo del parámetro se indica entre los signos <base_type>
- **ArrayList**<String> stringlist = new **ArrayList**<String>();

Generics y clases parametrizables

```
public class ParameterizedClass<P> { // P es un parámetro de tipo genérico.
        private P algo;
        public P getAlgo() {
                return algo;
        }
        public void setAlgo(P algo) {
                this.algo = algo;
        }
}
```

- Una clase que es definida con un parámetro de un tipo específico, se dice que es una clase parametrizable o una clase genérica.
- El tipo de parámetro puede ser usado como cualquier otro tipo en la definición de una clase.

Generics y clases parametrizables

```
public class Box<T> {
   private T t;
   public void add(T t) {
      this.t = t;
   }
   public T get() {
      return t;
   }
   public static void main(String[] args) {
      Box<Integer> integerBox = new Box<Integer>();
      Box<String> stringBox = new Box<String>();
      integerBox.add(new Integer(10));
      stringBox.add(new String("Hello World"));
      System.out.printf("Integer Value :%d\n\n", integerBox.get());
      System.out.printf("String Value :%s\n", stringBox.get());
   }
}
```

Integer Value :10
String Value :Hello World

Generics y clases parametrizables

- Aunque una clase parametrizable define el tipo de parámetro entre **<…>**, los constructores no llevan esta definición.
- El constructor puede contener parámetros del tipo parametrizable

```
public Box(T obj);
```

- Cuando se instancia una clase parametrizable, es necesario poner el tipo de clase parámetro entre **<…>**.

```
Box<String> myBox = new Box<String>(“Hola”);
Box<Integer> numBox = new Box<Integer>(5);
```

- Cuando defino un tipo de parámetro T, el mismo no puede ser usado para instanciar objetos. El parámetro es una referencia a un tipo de clase.

```
T object = new T();
T[] a = new T[10];
```

Wildcards – comodines

- Wildcards permite almacenar en una colección más de un tipo de objetos.
- Se utiliza la técnica de Upperbound y Lowerbound para definir el tipo de parámetro que debe permitirse en la colección.
- La vinculación del tipo de parámetro se realiza mediante el operador ? que significa “un tipo desconocido”.

Upperbound

Considere la siguiente definición:

List<? extends Number>

- La siguiente lista contiene objetos de tipos desconocidos, pero todos ellos heredan de la clase Number.

```
List<Integer> ints = new ArrayList<Integer>();
ints.add(2);
List<? extends Number> nums = ints;  // Allowed because of wildcards
nums.add(3.14);  // This is not allowed now after setting an upperbound
Integer x=ints.get(1);
```

Lowerbound

Considere la siguiente definición:

List<? super Integer>

- La siguiente lista contiene objetos de tipos desconocidos, pero todos ellos son ancestros (superclass) de la clase Integer.

```
List<Number> ints = new ArrayList<Number>();
ints.add(2);
List<? super Integer> nums = ints;  // Allowed because of wildcards
```

Métodos genéricos

- El concepto de tipos genéricos o Parametrización también es aplicado a los métodos.
- Un tipo genérico se puede utilizar para definir argumentos y/o para definir el tipo de retorno.
- No hay restricciones sobre el uso de tipos genéricos en los métodos.

```
protected abstract< T, R > R performAction( final T action );

static< T, R > R performActionOn( final Collection< T > action ) {
    final R result = ...;
    // Implementation here
    return result;
}
```