Tema 3: Genericidad en Java

Germán Moltó
Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

Tema 3: Genericidad en Java

Índice general:

Objetivos y Bibliografía

- Entender la Programación Genérica como mecanismo para la reutilización de software.
- Estudio de la sintaxis y la terminología necesaria para utilizar la genericidad en Java.

- Bibliografía:
  - Tutorial "Learning the Java Lenguage. Generics" disponible en Inglés en la siguiente dirección: http://java.sun.com/docs/books/tutorial/java/generics/

Modelos de Datos Genéricos

- Objetivo: Independizar el proceso del tipo de datos sobre los que se aplica.

```
public class Caja {
    private Integer dato;
    public Caja() { super();}
    public Integer quita() {return dato;}
    public void pon(Integer d){dato = d;}
}
```

```
public class TestCaja {
 public static void main(String args[]){
    Caja c = new Caja();
    c.pon(new Integer(46));
    Integer x = c.quita();
 }}
```

Modelos de Datos Genéricos (II)

- Es posible generalizar la Caja para que permita trabajar con cualquier tipo de datos.

```
public class Caja {
    private Object dato;
    public Caja() { super();}
    public Object quita() {return dato;}
    public void pon(Object d){dato = d;}
}
```

```
public class TestCaja {
 public static void main(String args[]){
    Caja c = new Caja();
    c.pon(new Manzana());
    Manzana x = (Manzana) caja.quita();
 }}
```

- La utilización de diversos tipos de datos únicamente puede detectarse en tiempo de ejecución.
- Requiere un Casting tras obtener el valor de la caja.

Genericidad en Java (I)

- A partir de la versión 1.5, Java introduce mecanismos explícitos para gestionar la genericidad.

```
public class Caja <T> {
    private T dato;
    public Caja() { super();}
    public T quita() {return dato;}
    public void pon(T d){dato = d;}
}
```

- Permite la comprobación de tipos correctos en tiempo de compilación.

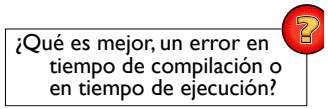
```
public class TestCaja {
   public static void main(String args[]){
      Caja<Integer> caja = new Caja<Integer>();
      caja.pon(new Integer(46));
      Integer x = caja.quita(); }
}
```

- Declaración explícita del tipo de datos a utilizar.
- No es necesario el casting tras obtener el valor.

Genericidad en Java (II)

- El uso de los mecanismos de Genericidad de Java 1.5:
  - Permite la detección de errores de tipos en tiempo de compilación y evita algunos errores de ejecución.
  - Evita realizar castings tras obtener el dato genérico.

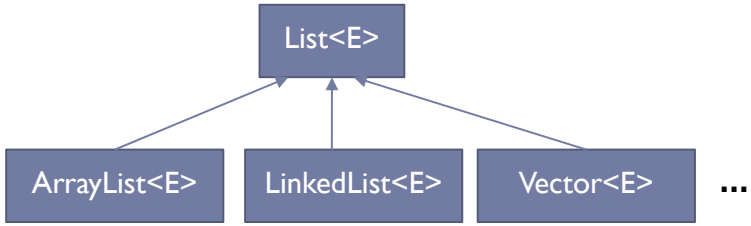
```
public class TestCaja
{
   public static void main(String args[]){
      Caja <Integer> caja = new Caja<Integer>();
      caja.pon("46");
      Integer x = caja.quita();
}}
```



- Mensaje del compilador "pon(java.lang.Integer) in Caja<java.lang.Integer> cannot be applied to (java.lang.String).

La Interfaz Genérica Estándar: List<E>

- La interfaz List<E> permite definir una colección de elementos:
  - El usuario controla en qué punto de la lista se realizan las inserciones (por defecto se añaden al final).
  - Puede contener elementos duplicados.



La Clase Genérica Estándar ArrayList

- La Clase *ArrayList<E>*
  - Define un grupo de objetos de tipo E a los que se accede a partir del índice.
  - Permite el crecimiento dinámico de la estructura de datos.

```
public class TestArrayList{
   public static void main(String args[]){
      List<Figura> lista = new ArrayList<Figura>();
      lista.add(new Circulo());
       Figura f = lista.get(0);
      Circulo c = (Circulo) lista.get(0);
 }}
```

- El método *get* puede lanzar IndexOutOfBoundsException (RuntimeException)

Restricción de Tipos Genéricos

- Es posible restringir el tipo genérico para trabajar con un tipo específico (y sus subtipos).

```
public class CajaNumeros <T extends Number>
{
   private T dato;
   public T quita(){return dato;}
   public void pon(T dato){this.dato = dato;}
}
```

```
CajaNumeros<Double> caja = new CajaNumeros<Double>(); //OK
```

```
CajaNumeros<String> caja2 = new CajaNumeros<String>(); //ERROR
```

- Error del compilador: "type parameter java.lang.String is not within its bound"

Métodos Genéricos

- Al igual que se definen clases genéricas, es posible definir métodos genéricos (incluso en clases no genéricas).

```
public class Caja <T>
{
   private T dato;
   public T quita(){return dato;}
   public void pon(T dato){this.dato = dato;}
   public static <U> void ponEnCaja(U u, Caja<U> caja){
      caja.pon(u);
    }
}
```

```
Caja<Integer> caja = new Caja<Integer>();
Caja.ponEnCaja(new Integer(52), caja);
```


Instanciando la Genericidad en una Subclase

- Es posible particularizar una clase genérica en una subclase para que utilice un tipo de datos determinado:

```
public class Caja <T> {
    protected T dato;
    public Caja() { super();}
    public T quita() {return dato;}
    public void pon(T d){dato = d;}
}
```

```
public class Manzana {
    private int sabor;
    public Manzana(int sabor){
        this.sabor = sabor;}
    public int getSabor(){return sabor;}
}
```

- La clase CajaManzanas deja de ser genérica.

```
public class CajaManzanas  extends Caja<Manzana>
{
  public int getSabor(){return dato.getSabor();}
}
```

Genericidad con Más de Una Variable

- Una clase genérica (o un método) puede referencias más de una variable genérica.
- Ejemplo:
  - La interfaz Map<K,V> del API de Java
    - public interface Map<K,V>{ … }
  - Permite establecer una correspondencia entre objetos de un tipo (denominados *claves)* y de otro (denominados *valores*)
    - Ambos tipos pueden coincidir.
- Ejemplo de instanciación:
  - Map<Integer, String> m = new SortedMap<Integer, String>();
  - m.put(new Integer(5), "casa");
  - String s = m.get(new Integer(5));

Programación Genérica a Partir de la Herencia

- Reutilizar código requiere:
  - Independizar un algoritmo del tipo de datos sobre el que se aplica, i.e. realizar una Programación Genérica.
- Ejemplo: Problema de la Ordenación de un array de elementos
  - Diversos algoritmos de ordenación:
    - Inserción directa
    - Selección directa
    - Intercambio directo
  - Se debe conocer el tipo base del array si es primitivo, como int, u objeto como Figura o Integer.

Inserción Directa: Dos Métodos Según el Tipo

```
// Ordenación del array int a[]
for( int i = 1; i < a.length ; i++ ) {
  int elemAInsertar =  a[i];
  int posIns = i ;
  for(; posIns>0  &&  elemAInsertar < a[posIns-1]; posIns--)  a[posIns]=a[posIns-1];
  a[posIns] = elemAInsertar;
}
```

```
// Ordenación del array Figura a[]
for( int i = 1; i < a.length ; i++ ) {
  Figura elemAInsertar =  a[i];
  int posIns = i ;
  for(; posIns>0 && elemAInsertar.area() < a[posIns-1].area(); posIns--)
          a[posIns]=a[posIns-1];
  a[posIns] = elemAInsertar; }
```


Herramientas para Programación Genérica

- ¿Qué haría falta para desarrollar un método genérico?
  - Un tipo o clase genérica compatible con el de los elementos a ordenar.
  - Un tipo o clase genérica con un único método abstracto que permita comparar objetos compatibles entre sí:
    - La interfaz Comparable<T>, cuyo único método es:

      int **compareTo**(T o)

- El método int compareTo(T o):
  - Devuelve < 0 si este objeto es menor que el objeto o.
  - Devuelve = 0 si este objeto es igual que el objeto o.
  - Devuelve > 0 si este objeto es mayor que el objeto o.

Método Genérico de Inserción Directa

- Un código genérico de ordenación requiere:
  - Restringir el tipo de datos de los elementos del vector a aquellos que sean comparables.
  - La clase de los elementos del vector debe implementar la interfaz Comparable<T>.

```
public static  <T extends Comparable<T>> void insercionDirecta( T a[] )
  for( int i = 1; i < a.length ; i++ ) {
    T elemAInsertar = a[i];
     int posIns = i ;
    for(; posIns>0 && elemAInsertar.compareTo(a[posIns-1]) < 0; posIns--) {
          a[posIns]=a[posIns-1];
     }
    a[posIns] = elemAInsertar;
}}
```


Uso de insercionDirecta

- Uso del método estático insercionDirecta en la clase Ordenacion del paquete ordenacionArray:

```
import ordenacionArray.*;
public class TestOrdenacionInteger {
    public static void main(String args[]){
            Integer a[] = new Integer[4];
            .... //Rellenar el vector a
            Ordenacion.insercionDirecta(a);
            .... //El vector a está ordenado
          }
    }
```

1. Importar el paquete que contiene la clase Ordenacion.
2. Invocar al método genérico, tras inicializar el array.
3. La clase base (o tipo) del array debe implementar el interfaz Comparable<T>, proporcionando código al método compareTo.

Implementación de la Interfaz Comparable

- int compareTo(T o):
  - Devuelve < 0 si este objeto es menor que el objeto o.
  - Devuelve = 0 si este objeto es igual que el objeto o.
  - Devuelve > 0 si este objeto es mayor que el objeto o.

```
public abstract class Figura implements Comparable<Figura> {
   ...
   public int compareTo(Figura o){
          double areaf = o.area();
          double areathis = this.area();
          if ( areathis < areaf ) return -1;
          else if ( areathis > areaf ) return  1;
          else  return  0;
   }
}
```


Ejercicio: ¿Qué líneas producen error?

1. Figura f1 = new Circulo();
2. Figura f2 = new Cuadrado();
3. System.out.println(f1.compareTo(f2));
4. Object x = f1;
5. System.out.println(x.compareTo(f1));
6. System.out.println((Comparable) x.compareTo(f1));
7. System.out.println(((Comparable) x).compareTo(f1));
8. System.out.println(f2.compareTo(x));

Los métodos equals y compareTo

- Se recomienda que las definiciones de equals y compareTo sean compatibles:
  - (x.compareTo(y) == 0) ←→ x.equals(y)

```
public int compareTo(Figura o){
   double areaF = o.area(),  area = this.area();
   if ( area < areaF ) return -1;
   else
   if (tipo.equals(f.tipo) && color.equals(f.color) && area==areaF) return  0;
   else  return  1;
}
```

```
public boolean equals(Object x){
   return ( this.compareTo( (Figura) x) == 0 ) ;
 }
```


Ejercicios Propuestos de Programación Genérica

- **Ejercicio 1**. Diseñar la Clase Genérica Operaciones, en el paquete ordenacionArray del proyecto Bluej util de librerias, con la siguiente funcionalidad:
  - Calcular el elemento mínimo de un array;
  - Calcular el elemento máximo de un array;
  - Buscar un Objeto en un array, devolviendo la posición de la primera aparición del Objeto en el array o -1 si no existe.
  - Además, diséñese una aplicación que use la Clase Operaciones para un array de Integer
- **Ejercicio 2.** Añádase a la clase Operaciones un método genérico que borre la primera aparición en el array de un objeto dado, devolviendo el objeto eliminado del array o null si no se encuentra.