Genéricos -Java

http://java.sun.com/developer/technicalArticles/J2SE/generics/index.htm
http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf
http://java.sun.com/docs/books/tutorial/extra/generics/index.html
https://www.arquitecturajava.com/uso-de-java-generics/

Declaración de clases con genéricos

- **Un Genérico permite parametrizar una clase(No un tipobásico).**
- **Se utiliza el operador diamante**
- **Declaración**

```
public class Pila <T>{
private ArrayList<T>items;
public void push(T item) {...}
public T pop() {
T  aux= items.get(0);
return aux;
}
public booleanisEmpty() {...}
}
```

Utilización de clases con genéricos

- **Cuando se utiliza, el parámetro se sustituye por un tipo concreto.**
- **Utilización**

```
Pila<String> p= new  Pila<String>();
p.push("Mesa");
String palabra= p.pop(); //No se necesitacasting
Pila<Persona>  pp= new Pila<Persona> ();
```

No se puede invocar a constructores del tipo genérico

- En una clase genérica **no puedes invocar a constructores** del tipo genérico. Pues el tipo puede ser sustituido por múltiples clasesy obligarías a que todas tuvieran los mismos parámetros en el constructor
- **Declaración**

```
public class Pila <T>{
public método() {
  T t= new T() //  →Esta instrucción no está permitida.
  ……
  }
}
```

Herencia con genéricos

- **Cuando se hereda de una clase que utiliza genéricos hay que colocar el "tipo" en la declaración de herencia:**

```
public classCola <Tipo> {
ArrayDeque<Tipo> cola;

Cola (){
cola = new ArrayDeque<Tipo>( );
}
int insertar (Tipo o){
  cola.addLast(o);
  return0;
}
Tipo extraer(){
  return cola.removeFirst();
}
}
```

```
public class ColaMaxTamano<Tipo> extends Cola<Tipo>{
final static int MAXIMOTAMANO=10;
@Override
int insertar(Tipoo) {
  int aux=0;
  if (!cola.contains(o)){
  if (cola.size()>=MAXIMOTAMANO)
    extraer();
  cola.addLast(o);
}
else {
  cola.remove(o);
  cola.addLast(o);
  aux=1;
}
return aux;
}
void mostrar(){
  for (Tipot: cola)
    System.out.println(t);
}
}
```

Herencia con genéricos

- **Declaración**
- **//Pila para objetosde tipoPersona o sus subclases**

```
public class Pila<T  extends Persona>{
private ArrayList<T> items;
public void push(T item) {}
public T pop() {
T aux=……
return aux;
}
public booleanisEmpty() {}
}
```

T puede ser, además de una **clase**, una interfaz y, aunque se sigue utilizando **extends**se interpreta como "**implementa esa interfaz**". De hecho se puede indicar <T extendsA & B> para obligar a que T implemente las interfaces A y B

Herencia con genéricos

- **Se pueden utilizar varios tipos en la misma clase :**

```
public class Polinomio {
Vector <Integer>coeficientes= new
Vector<Integer>();
void asignar (int posicion, int valorCoeficiente){
if(coeficientes.size()<posicion){
for(int i=coeficientes.size(); i<posicion; i++)
  coeficientes.add(0);
  coeficientes.add( valorCoeficiente);
} else
  coeficientes.setElementAt(valorCoeficiente, posicion);
}
int mayor(){returncoeficientes.size(); }
int elemento(intpos){
  returncoeficientes.elementAt(pos);}
publicStringtoString(){
  String aux="";
  int i=0;
  for(int c: coeficientes){
    if(c!=0)
      aux+= c+"x^"+i+"+";
      i++;
  }
return aux;
}
}
```

```
public class Numero {
int valor;
Numero( int n){
valor=n;
}
int getValor(){
return valor;
}
}
classNumero2 extends Numero{
.....}
```

Collection admite tipos genéricos a partir de la versión 1.5 de java

Interface Collection<E>

Interface List<E>

……

Class LinkedList<E>

LinkedList()

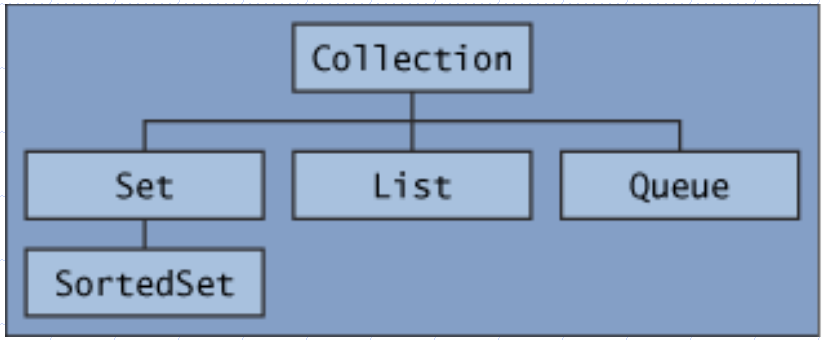
boolean **add**(Eo)

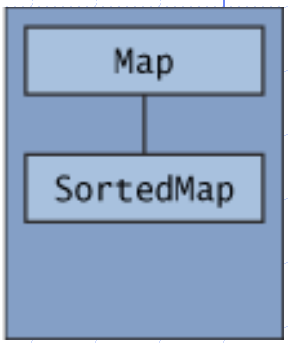
E**get**(intindex)

public Iterator<E> **iterator**()

Class ArrayList<E> ….

Class Vector<E> ….





Relaciones de Herencia

List<String> ls= new LinkedList<String>();

List<Object> lo;

lo y ls No son compatibles.

List<String> no es una subclase de List<Object>

lo=ls; //Error

Relaciones entre Collections varianza

```
public void printAll (Collection<Object> c){
  for (Object o:c)  System.out.println ( o);
}
List<Object> list= new ArrayList<Object>()
printAll(list); //OK
```

Collection<Object> es compatible con List<Object>

Métodos Genéricos

- Pueden ser estáticos, no estáticos o constructores.
- Se parametrizan antes del tipo devuelto con uno o más tipos genéricos.
- La asociación entre el parámetro y el tipo real se realiza en la llamada al método.

Métodos Genéricos-Ejemplo

- Método que pasa de array de objetos a Collection:

```
public class UsoMetodosGenericos{
static void deArrayACollection1(Object []vo, Collection <?> c) {
System.out.println("Método 1");
//  c.add(vo[0] );  //Error No se puede añadir a Collection<?>
}
static<T> void deArrayACollection2 (Object[]vo, Collection<T> c) {
  System.out.println("Método 2");
  for(int i=0; i<vo.length;i++)
    c.add((T)vo[i] );  //OK si voes de tipo T
  }
static<T> voidmostrarCollection(Collection<T> c){
  for(T o: c)
    System.out.println(o);
}
public static void main (String arg[]){
String vs[]= {"Uno", "Dos","Tres"}; //= new String[3];
Collection<String> cs= new LinkedList<String>();
deArrayACollection1(vs,cs);  //La llamada si es correcta
deArrayACollection2(vs,cs);
mostrarCollection(cs);
}
}
```

Resumen-Genericos

- Sin tipos genéricos (≤ JDK 1.4.2)
- List unaLista = new LinkedList();
- unaLista.add(newInteger(0));
- Integer x = (Integer) unaLista.iterator().next();
- Con tipos genéricos (≥ JDK 1.5)

- List<Integer> unaLista = newLinkedList<Integer>();
- unaLista.add(new Integer(0));
- Integer x = unaLista.iterator().next();

```
public interfaceList<E> {
  void add(E x);
  Iterator<E> iterator();
}
public interfaceIterator<E> {
  E next();
  boolean hasNext();
}
```

Wildcards

- **"? extends Type":** Subtipos del tipo Type, incluido Type. T puede ser, además de una clase, una interfaz y, aunque se sigue utilizando extends, se interpreta como "implementa esa interfaz". Se usa en covarianza
- **"? super Type":** Superclases del tipo Type, incluido Type. se usa en contravarianza
- **"?":** Cualquier tipo

Si se usan wildcards no se pueden añadir objetos.

```
Collection<?> c= new ArrayList<String>();
c.add(new Object()); //ERROR de compilación
```

Wildcards: Utilización

```
class PruebaGen {
void imprimir (String s){
   System.out.println (s);
}
void imprimir (Collection <? extends String> c){
    for(String s: c) {
      this.imprimir(s);
    }
 }
//Collection <?> indicará una colección de cualquier cosa
void imprimir2 (Collection <?> c){
    for(Object o: c)
      System.out.println(o);
 }
```

Wildcards: Utilización

```
public static void main(String[] args){

    List<String> palabras = new LinkedList<String>();
    palabras.add("uno");
    palabras.add("dos");
    new PruebaGen().imprimir(palabras);

    new PruebaGen().imprimir2(palabras);
  }
}
```

CoVarianza y ContraVarianza Creación de objetos con genéricos

- Covariance
  - Permite usar un tipo más derivado que el especificado originalmente.
  - Puede asignar ITipo <Base>. =ITipo<Derived>

- Contravariance
  - Permite usar un tipo más genérico (menos derivado) que el especificado originalmente.
  - Puede asignar ITipo <Derived>= ITipo <Base>

  - https://learn.microsoft.com/es-es/dotnet/standard/generics/covariance-and-contravariance
  - https://dzone.com/articles/covariance-and-contravariance

CoVarianza y ContraVarianza

- Ver ejemplo UsoCoyContraVarianza.java

Creación de objetos con genéricos

- No se pueden instanciar:
  T elem = new T(); //ERROR
  T elem [] = new T[3]; //ERROR

- Si se pueden crear listas
  List<T> lista =new LinkedList<T>();

Collection admite tipos genéricos a partir de la versión 1.5 de java

Interface Collection<E>

Interface List<E>

.....

Class LinkedList<E>

LinkedList()

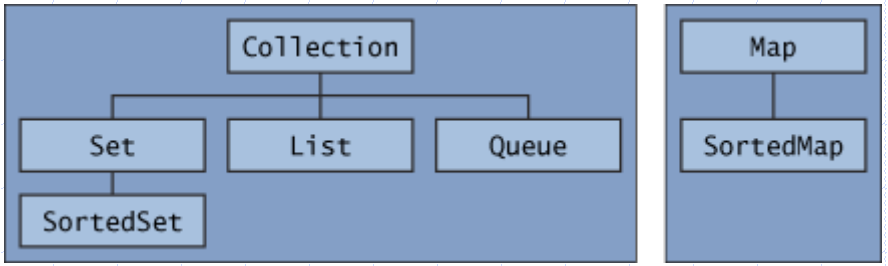
boolean **add**(E o)

E **get**(int index)

public Iterator<E> **iterator**()

Class ArrayList<E> ....

Class Vector<E> ....



Relaciones de Herencia

List<String> ls= new LinkedList<String>();

List<Object> lo;

lo y ls No son compatibles.

List<String> no es una subclase de List<Object>

lo=ls; //Error

Relaciones entre Collections

```
public void printAll (Collection<Object> c){
    for (Object o:c)  System.out.println ( o);
}
List<Object> list= new ArrayList<Object>()
  printAll(list); //OK
```

Collection<Object> es compatible con List<Object>

Métodos Genéricos

- Pueden ser estáticos, no estáticos o constructores.
- Se parametrizan antes del tipo devuelto con uno o más tipos genéricos.

  Ejem: public <T> List<T>devolverLista(){}
- La asociación entre el parámetro y el tipo real se realiza en la llamada al método.

https://www.baeldung.com/java-generics

Métodos Genéricos-Ejemplo

Método que pasa de array de objetos a Collection:

```
public class UsoMetodosGenericos {
  static void deArrayACollection1(Object []vo, Collection <?> c) {
     System.out.println("Método 1");
     //  c.add(vo[0] );  //Error No se puede añadir a Collection<?>
  }
   static <T> void deArrayACollection2 (Object []vo, Collection <T> c) {
     System.out.println("Método 2");
     for (int i=0; i<vo.length;i++)
       c.add((T)vo[i] );  //OK si vo es de tipo T
   }
   static <T> void mostrarCollection (Collection <T> c){
      for (T o: c)
     System.out.println(o);
  }
   public static void main (String arg[]){
      String vs[]= {"Uno", "Dos","Tres"}; //= new String [3];
      Collection<String> cs = new LinkedList<String>();
      deArrayACollection1(vs,cs);  //La llamada si es correcta
      deArrayACollection2(vs,cs);
     mostrarCollection (cs);
   }
}
```

OTRO EJEMPLO:

```
public static <T> void listar(List <T>lista) {
  for (T o: lista)
     System.out.println(o);
}
```

UTILIZACIÓN

```
listar(Files.list(dir).toList());
listar(Files.list(dir)
       .filter(p->!Files.isDirectory(p)).toList());
```

Resumen-Genericos

- Sin tipos genéricos (≤ JDK 1.4.2)

```
List unaLista = new LinkedList();
unaLista.add(new Integer(0));
Integer x = (Integer)
    unaLista.iterator().next();
```

- Con tipos genéricos (≥ JDK 1.5)

```
List<Integer> unaLista = new
    LinkedList<Integer>();
unaLista.add(new Integer(0));
Integer x = unaLista.iterator().next();
```

```
public interface
List<E> {
  void add(E x);
  Iterator<E> iterator();
}
public interface
Iterator<E> {
  E next();
  boolean hasNext();
}
```