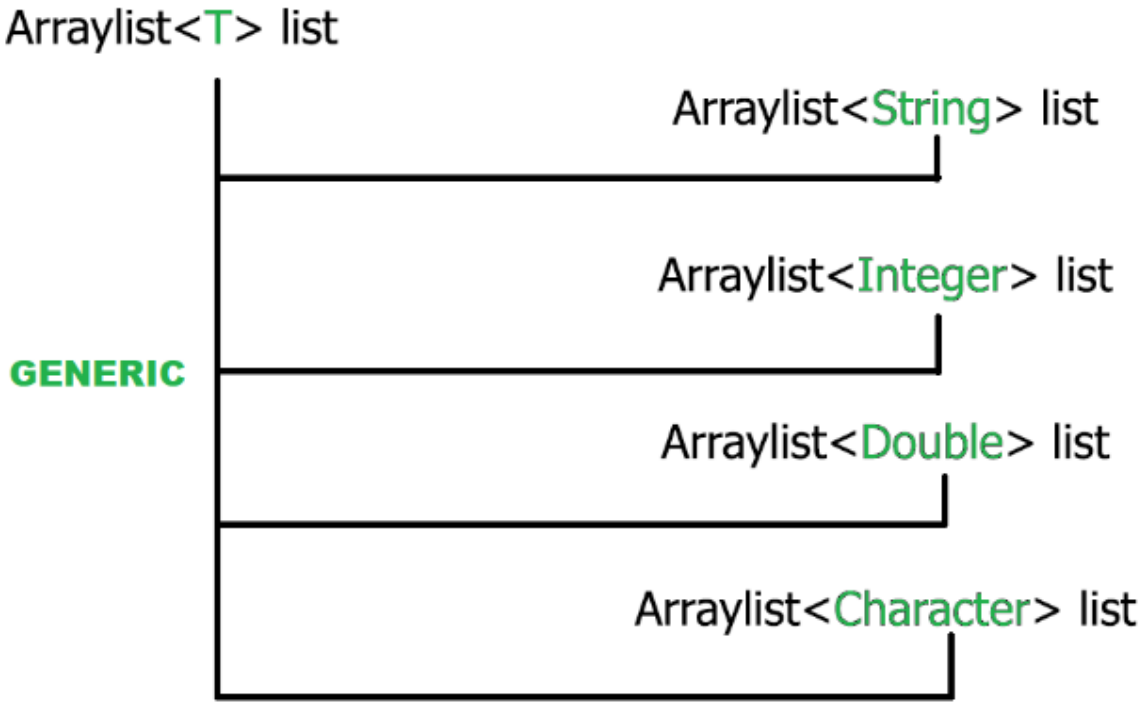
Generics


Arraylist<T> list
GENERIC
Arraylist<String> list
Arraylist<Integer> list
Arraylist<Double> list
Arraylist<Character> list

Generics

- Ưu điểm của Generics:
  - Kiểm tra kiểu dữ liệu trong thời điểm biên dịch để đảm bảo tính chặt chẽ của kiểu dữ liệu.

```
class A{}
class B extends A{
    public void methodB(){}
}
class C {}
```

```
public class GenericEx {
    public static void main(String[] args) {
        List list = new ArrayList();
        list.add(new A());
        list.add(new B());
        list.add(new C());

        ((B)list.get(0)).methodB(); //compile OK
        ((B)list.get(1)).methodB(); //compile OK
        ((B)list.get(2)).methodB(); //compile OK
    }
```

```
Exception in thread "main" java.lang.ClassCastException: class genericex.A cannot be cast to class genericex.B (genericex.A and genericex.B are in unnamed module of loader 'app')
        at genericex.GenericEx.main(GenericEx.java:33)
D:\Hoc tap\Java\GenericEx\nbproject\build-impl.xml:1328: The following error occurred while executing this line:
D:\Hoc tap\Java\GenericEx\nbproject\build-impl.xml:948: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

Generics

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

        List<A> list = new ArrayList<A>();
        list.add(new A());  //OK Compile
        list.add(new B());  //OK Compile
```

incompatible types: C cannot be converted to A

(Alt-Enter shows hints)

```
        list.add(new C());

        for (A item : list){
            if (item instanceof B) ((B)item).methodB();
        }
    }
```

Generics

- Cho phép thực hiện các xử lý tổng quát: thực hiện các thuật toán tổng quát với các kiểu dữ liệu tùy chọn khác nhau.

```
public static <T extends Number> double add(T a, T b){
    return a.doubleValue() + b.doubleValue();
}

public static void main(String[] args) {
    System.out.println(add(1,2));
    System.out.println(add(7.5,15.0));
}
```

Generics class

```
public class KeyValue<K, V> {
    private K key;
    private V value;

    public KeyValue(K key, V value) {
        this.key = key;
        this.value = value;
    }
    public K getKey() {
        return key;
    }
    public void setKey(K key) {
        this.key = key;
    }
     public V getValue() {
        return value;
    }
     public void setValue(V value) {
        this.value = value;
    }
}
```

Generics class

```
public static void main(String[] args) {
    // TODO code application logic here
    //KeyValue với kiểu Integer và String
    KeyValue<Integer, String> keyValue = new KeyValue<Integer, String>(1234, "Test");
    Integer id = keyValue.getKey();
    String name = keyValue.getValue();
    System.out.println("ID: " + id + "Name =  " + name);

    //KeyValue với kiểu String và String
    KeyValue<String, String> keyString = new KeyValue<>("ABC","XYZ");
    String key = keyString.getKey();
    String value = keyString.getValue();
    System.out.println("Key: " + key + "Value =  " + value);
}
```

Generic Method

- Kiểu <T> có thể được sử dụng làm kiểu trả về của phương thức.
- Sử dụng generic method khi logic của phương thức giống nhau và chỉ khác biệt nhau về kiểu dữ liệu thì có thể cài đặt phương thức theo generic.
- VD1:

```
public static <T> T getMiddle(T... a){
    return a[a.length/2];
}

public static void main(String[] args) {
    String middle = getMiddle("ABC","DEF","IJK");
    System.out.println(middle);
    int midInt = getMiddle(5,8,1,17,19,20);
    System.out.println(midInt);
}
```

Generic Method

```
public static <T extends Comparable> T timMax(T a,T b,T c){
    T max = a;
    if (max.compareTo(b) < 0)
       max = b;
    if (max.compareTo(c) < 0)
        max = c;
    return max;
}

public static void main(String[] args) {
    int maxInt = timMax(8, 35, 20);
    System.out.println(maxInt);
    double max = timMax(5.7, 8.4, 30.0);
    System.out.println(max);
}
```

Ký tự đại diện Generic

- Ký tự đại diện **<?> (wildcard):** đại diện cho một kiểu không xác định.
- Có thể được sử dụng trong nhiều tình huống: tham số, biến cục bộ, thuộc tính hoặc có thể là một kiểu trả về.
- Không sử dụng như là một đối số cho lời gọi một phương thức generic, khởi tạo đối tượng class generic, hoặc kiểu cha.
- VD:
```
Collection<?> coll = new ArrayList<String>();
Pair<String,?> pair = new Pair<String,Integer>();
```
- Tham số ký tự đại diện không thể tham gia trong toán tử **new**

```
List<? extends Object> list= new ArrayList <? extends Object>(); //Lỗi
```

Ký tự đại diện Generic

- Có thể dùng để hạn chế kiểu dữ liệu của các tham số:

`<? extends type>:` kiểu dữ liệu kế thừa từ type hoặc đối tượng của type.

```
public static <T extends Comparable> T timMax(T a,T b,T c){
    T max = a;
    if (max.compareTo(b) < 0)
       max = b;
    if (max.compareTo(c) < 0)
        max = c;
    return max;
}
```

`<? super type>:` kiểu dữ liệu là kiểu cha type hoặc đối tượng của type

```
class A{}
class B extends A{}
class C {}

public static void methodB(List<? super B> param){}
```

```
public static void main(String[] args) {
    List<B> listB = new ArrayList<B>();
    methodB(listB);
    List<A> listA = new ArrayList<A>();
    methodB(listA);
    List<Object> listO = new ArrayList<Object>();
    methodB(listO);
    List<C> listC = new ArrayList<C>();
    methodB(listC);
}
```

Hạn chế của Generic

- Không thể khởi tạo generic với kiểu dữ liệu cơ sở

```
KeyValue<int, double> keyValue = new KeyValue<>(1234, 30.3); // Lỗi
```

- Không tạo được đối tượng của kiểu T

```
private T obj;
public GenExample(){
    obj = new T();
}
```

- Không là kiểu static trong class

```
class GenExample<T>
{
    //Lỗi
    static T obj;
    //Lỗi
    static T getObj(){
        return obj;
    }
}
```

Hạn chế của Generic

- Có thể khai báo một mảng generic nhưng không thể khởi tạo mảng Generic

```
public T[] arrayT; //OK
public T[] array = new T[10]; //Lỗi

//Lỗi
GenExample<String> gens[] = new GenExample<>[10];

//OK
GenExample<?> gens[] = new GenExample<?>[10];
gens[0] = new GenExample<Integer>(13);
gens[1] = new GenExample<String>("Gen");
```

Ví dụ mảng Generic

```
public class GenericArray<T> {
    public T[] array;

    public GenericArray(T[] arr){
       this.array = arr;
    }
    //Phương thức lấy phần tử tại vị trí index
    public T Get(int index){
        if (index >= 0 && index < array.length) return array[index];
        return null;
    }
}
public static void main(String[] args) {
    //Tạo một mảng String
    String[] country = new String[]{"US","UK","AU"};
    GenericArray<String> gArr = new GenericArray<>(country);
    String indCoun = gArr.Get(1);
    System.out.println(indCoun);


}
```

Interface Collections

- **Các phương thức trong Interface Collections:**

Phương thức	Mô tả
boolean add(Object element)	Thêm một phần tử vào collection.
boolean addAll(Collection c)	Thêm các phần tử collection được chỉ định.
boolean remove(Object element)	Xóa phần tử từ collection.
boolean removeAll(Collection c)	Xóa tất cả các phần tử của collection được chỉ định.
boolean retainAll(Collection c)	Giữ lại các phần tử collection được chỉ định.
int size()	Tổng số các phần tử trong collection.
void clear()	Xóa tất cả các phần tử khỏi collection.
boolean contains(Object element)	True nếu collection chứa phần tử được chỉ định.
boolean containsAll(Collection c)	True nếu collection chứa collection con được chỉ chỉ định.
Iterator iterator()	Trả về một iterator.
Object[] toArray()	Trả về mảng chứa tất cả phần tử của collection.
boolean isEmpty()	True nếu collection rỗng.
boolean equals(Object element)	So sánh một đối tượng với collection.
int hashCode()	Trả về giá trị hashcode của collection

ArrayList

- Khai báo và khởi tạo ArrayList có 2 cách:
  - Cách 1: khởi tạo một ArrayList rỗng

```
ArrayList<String> list = new ArrayList<String>();
```

  - Cách 2: khởi tạo và cung cấp số lượng phần tử ban đầu

```
ArrayList<Integer> listInt = new ArrayList<>(10);
```

- Truy xuất phần tử dùng phương thức `get(int index)`.

```
String s = list.get(1);
Interger num listInt.get(2);
```

ArrayList

- Duyệt ArrayList dùng vòng lặp For:

```
for (int i = 0; i < list.size(); i++){
System.out.println(list.get(i));}
```

Hoặc

```
for (int num : listInt){ System.out.println(num);}
```

- Duyệt ArrayList sử dụng Iterator:
  - Thuộc java.util.Iterator.
  - Khai báo Iterator cùng kiểu với ArrayList muốn duyệt.

```
Iterator<String> itr = list.iterator();
```

- Dùng hàm hasNext() và hàm next() để duyệt.

```
while (itr.hasNext()){  System.out.println(itr.next());}
```

ArrayList

- Duyệt ArrayList sử dụng ListIterator
  - Thuộc java.util.ListIterator.
  - Khai báo ListIterator cùng kiểu với ArrayList muốn duyệt.

```
ListIterator<int> listItr = list.listIterator();
```

- Dùng hàm hasNext() và next() để duyệt list từ đầu tới cuối

```
while (listItr.hasNext()){
System.out.println(listItr.next());}
```

- Dùng hàm hasPrevious() và previous() để duyệt list từ cuối về đầu.

```
while (listItr.hasPrevious()){
System.out.println(listItr.previous());}
```

ArrayList

- Một số phương thức của ArrayList:

- Thêm phần tử vào cuối danh sách: `add(Object o)` `list.add("XYZ");`
- Thêm collection vào cuối danh sách: `addAll(Collection c)`
  `list.addAll(listString);`
- Thêm phần tử vào vị trí bất kỳ trong danh sách:

```
add(int index, object value)
list.add(2, "XYZ");
```

- Thêm collection vào vị trí bất ký trong danh sách

```
addAll(int index, Collection c)
list.addAll(3, listString);
```

- Cập nhật giá trị phần tử: `set(int index, Object o)` `list.set(3, "ABC");`

ArrayList

- Xóa phần tử tại vị trí index: `Remove(int index)`

```
list.Remove(3);
```

- Xóa tất cả các phần tử: `Clear()`

```
list.Clear();
```

ArrayList

- Một số phương thức khác:

Phương thức	Mô tả
boolean isEmpty()	True nếu ArrayList rỗng.
int indexOf (Object o)	Trả về vị trí index trong list của phần tử o xuất hiện đầu tiên, hoặc -1 nếu List không chứa phần tử này
int lastIndexOf(Object o)	Trả về vị trí index của phần tử o cuối cùng, hoặc - 1 nếu List không chứa phần tử này
boolean removeAll(Collection c)	Xóa tất cả các phần tử của ArrayList được chỉ định.
boolean retainAll(Collection c)	Giữ lại các phần tử ArrayList được chỉ định.
void removeRange(int fromIndex, int toIndex)	Gỡ bỏ từ list này tất cả phần tử từ vị trí fromIndex đến toIndex
int size()	Tổng số các phần tử trong ArrayList.
boolean contains(Object element)	True nếu ArrayList chứa phần tử được chỉ định.
void trimToSize()	Cắt kích thước của ArrayList này về kích thước hiện tại.
Object[] toArray()	Trả về một mảng chứa tất cả phần tử của list.
Object clone()	Trả về một bản copy của ArrayList này39

LinkedList

- Khai báo và khởi tạo LinkedList:

○ Khởi tạo một LinkedList rỗng

```
LinkedList<String> list = new LinkedList<String>();
```

○Khởi tạo với danh sách phần tử: `LinkedList(Collection c)`

```
LinkedList<Integer> listInt = new LinkedList<>(lInt);
```

○Truy xuất phần tử dùng phương thức `get(int index)`.

```
String s = list.get(1);
Interger num = listInt.get(2);
```

- Duyệt LinkedList dùng vòng lặp For hoặc Iterator hoặc ListIterator giống với ArrayList

LinkedList

- Một số phương thức của LinkedList:

Phương thức	Mô tả
boolean add(Object o)	Thêm phần tử vào cuối list
boolean add(int index, Object o)	Thêm phần tử vào vị trí index của list.
boolean addAll(Collection c)	Thêm một collection vào cuối list.
boolean addAll(int index, Collection c)	Thêm một collection vào vị trí index của list
boolean addFirst(Object o)	Thêm phần tử vào đầu list
boolean addLast(Object o)	Thêm phần tử vào cuối list
void clear()	Xóa tất cả các phần tử khỏi list.
boolean contains(Object o)	True nếu list chứa phần tử được chỉ định.
Object get(int index)	Trả về phần tử tại vị trí index của list
Object getFirst()	Trả về phần tử đầu tiên của list
Object getLast()	Trả về phần tử cuối của list
int indexOf(Object o)	Trả về vị trí index của phần tử o xuất hiện đầu tiên trong list

LinkedList

- Một số phương thức của LinkedList:

Phương thức	Mô tả
int lastIndexOf(Object o)	Trả về vị trí index của phần tử o cuối cùng, hoặc - 1 nếu List không chứa phần tử này
Object remove(int index)	Xóa phần tử tại vị trí index trong list.
boolean remove(Object o)	Xóa phần tử o xuất hiện đầu tiên trong list.
Object removeFirst()	Xóa phần tử đầu tiên trong list.
Object removeLast()	Xóa phần tử cuối cùng trong list.
Object set(int index, Object o)	Thay thế phần tử tại vị trí index bằng phần tử o
int size()	Trả về số phần tử trong list
Object[] toArray()	Trả về một mảng chứa tất cả phần tử của list.

TreeSet

- Khai báo và khởi tạo TreeSet:
  - Khởi tạo một TreeSet rỗng

```
TreeSet<String> list = new TreeSet <>();
```

  - Khởi tạo với danh sách phần tử: TreeSet(Sorter<> s)

```
TreeSet <Integer> listInt = new TreeSet <>(lInt);
```

  - Khởi tạo với với bộ so sánh Comparator tùy chỉnh.

```
TreeSet<String> list = new
TreeSet<>(String.CASE_INSENSITIVE_ORDER);
```

Hoặc

```
TreeSet<String> list = new TreeSet<>(Comparator.reverseOrder());
```

TreeSet

Hoặc

```
TreeSet<String> list = new TreeSet<>(new  Comparator<String>() {
      @Override
      public int compare(String s1, String s2) { return
      s2.compareTo(s1);
      }
  });
```

TreeSet

- Các phương thức trong TreeSet

Phương thức	Mô tả
void add(Object o)	Thêm phần tử vào TreeSet
boolean addAll(Collection c)	Thêm một collection vào TreeSet.
boolean remove(Object o)	Xóa phần tử khỏi TreeSet
void clear()	Xóa tất cả các phần tử khỏi TreeSet
boolean contains(Object o)	True nếu TreeSet chứa phần tử được chỉ định.
Object first()	Trả về phần tử đầu tiên (nhỏ nhất) của TreeSet
Object last()	Trả về phần tử cuối cùng (lớn nhất) của TreeSet
SortedSet subSet(Object fromElement, Object toElement)	Trả về SortedSet từ fromElement đến phần tử đứng trước toElement
SortedSet headSet(E toElement)	Trả về SortedSet từ phần tử đầu tiên đến phần tử đứng trước toElement
SortedSet tailSet(E fromElement)	Trả về SortedSet từ phần tử lớn hơn hoặc bằng fromElement đến phần tử cuối cùng
int size()	Trả về số phần tử trong TreeSet
boolean isEmpty()	True nếu TreeSet rỗng.

TreeMap

- Khai báo và khởi tạo TreeMap:
  - Khởi tạo một TreeMap rỗng

```
TreeMap<Integer, String> list = new TreeMap <>();
```

  - Khởi tạo với danh sách phần tử:

```
TreeMap <Integer , String> list = new TreeMap <>(map);
```

  - Hiển thị toàn bộ TreeMap: entry