Generic Interface Example:

import java.util.ArrayList;

import java.util.List;

// A generic interface named Container with a type parameter T

interface Container<T> {

void add(T item);

T get(int index);

int size();

}

// An implementation of the Container interface for a list

class ListContainer<T> implements Container<T> {

private List<T> items;

public ListContainer() {

this.items = new ArrayList<>();

}

@Override

public void add(T item) {

items.add(item);

}

@Override

public T get(int index) {

return items.get(index);

}

@Override

public int size() {

return items.size();

}

public static void main(String[] args) {

// Using the ListContainer with Integer type

Container<Integer> intContainer = new ListContainer<>();

intContainer.add(1);

intContainer.add(2);

System.out.println("Integer Container Size: " + intContainer.size());

System.out.println("Item at index 0: " + intContainer.get(0));

// Using the ListContainer with String type

Container<String> stringContainer = new ListContainer<>();

stringContainer.add("Hello");

stringContainer.add("World");

System.out.println("String Container Size: " + stringContainer.size());

System.out.println("Item at index 0: " + stringContainer.get(0));

}

}

Bounded Types Example:

class Stats<T extends Number>{

T[] nums;

Stats(T[] o){

nums=o;

}

double average(){

double sum=0;

for(int i=0;i<nums.length;i++){

sum+=nums[i].doubleValue();

}

return sum/nums.length;

}

}

class BoundDemo{

public static void main(String args[]){

Integer nums[]={1,2,3,4,5,6};

Stats<Integer> s1=new Stats<Integer>(nums);

System.out.println(s1.average());

}

}

Sorting Collection Example:

import java.util.\*;

class SortCollection{

public static void main(String args[]){

ArrayList<Integer> al=new ArrayList<Integer>();

al.add(30);

al.add(40);

al.add(10);

al.add(5);

//Collections.sort(al);

Collections.sort(al,Collections.reverseOrder());

Iterator itr=al.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

Generic methods and Constructors Example:

public class GenericMethodExample {

<S extends Number> GenericMethodExample(S o){

System.out.println(o.toString());

}

<S extends String> GenericMethodExample(S o){

System.out.println(o);

}

// Generic method to print elements of an array

public static <T> void printArray(T[] array) {

for (T element : array) {

System.out.print(element + " ");

}

System.out.println();

}

public static void main(String[] args) {

// Creating arrays of different types

Integer[] intArray = {1, 2, 3, 4, 5};

Double[] doubleArray = {1.1, 2.2, 3.3, 4.4, 5.5};

Character[] charArray = {'H', 'E', 'L', 'L', 'O'};

String[] stringArray = {"Java", "Generics", "Are", "Cool"};

// Calling the generic method with different types of arrays

System.out.println("Integer Array:");

printArray(intArray);

System.out.println("Double Array:");

printArray(doubleArray);

System.out.println("Character Array:");

printArray(charArray);

System.out.println("String Array:");

printArray(stringArray);

GenericMethodExample ge=new GenericMethodExample(intArray[0]);

}

}

Wrapper class Example:

public class WrapperExample {

public static void main(String[] args) {

// Primitive data types

int a = 5;

double b = 5.65;

// Converting primitives into objects using wrapper classes

Integer aObj = Integer.valueOf(a);

Double bObj = Double.valueOf(b);

// Autoboxing: Automatically converting primitive to object

Integer aAutoBox = a;

Double bAutoBox = b;

// Displaying object values

System.out.println("Integer object aObj: " + aObj);

System.out.println("Double object bObj: " + bObj);

System.out.println("Autoboxed Integer aAutoBox: " + aAutoBox);

System.out.println("Autoboxed Double bAutoBox: " + bAutoBox);

// Unboxing: Converting objects back to primitives

int aUnbox = aObj.intValue();

double bUnbox = bObj.doubleValue();

// Auto-unboxing: Automatically converting object to primitive

int aAutoUnbox = aObj;

double bAutoUnbox = bObj;

// Displaying primitive values

System.out.println("Unboxed int aUnbox: " + aUnbox);

System.out.println("Unboxed double bUnbox: " + bUnbox);

System.out.println("Auto-unboxed int aAutoUnbox: " + aAutoUnbox);

System.out.println("Auto-unboxed double bAutoUnbox: " + bAutoUnbox);

// Using some useful methods from wrapper classes

String intStr = "123";

String doubleStr = "45.67";

// Parsing strings into primitive data types

int parsedInt = Integer.parseInt(intStr);

double parsedDouble = Double.parseDouble(doubleStr);

System.out.println("Parsed int from String: " + parsedInt);

System.out.println("Parsed double from String: " + parsedDouble);

// Converting a primitive type to String

String intToStr = Integer.toString(a);

String doubleToStr = Double.toString(b);

System.out.println("Integer to String: " + intToStr);

System.out.println("Double to String: " + doubleToStr);

}

}

Generic Class Example:

class Gen<S>{

S ob;

Gen(S o){

ob=o;

}

S getob(){

return ob;

}

void showType(){

System.out.println("Type of S"+ob.getClass().getName());

}

}

class GenDemo{

public static void main(String args[]){

Gen<Integer> iob=new Gen<Integer>(88);

Gen<String> iob2=new Gen<String>("sudheer");

iob.showType();

iob2.showType();

}

}

Functional Interface Example:

interface Mynum{

double getValue(int a);

}

class FunctionalInterface{

public static void main(String args[]){

Mynum num;

num=(a)->a\*10.123;

System.out.println(num.getValue(5));

}

}

Comparable and Comparator example:

Student.java

class Student implements Comparable<Student>{

int age;

String name;

Student(int age,String name){

this.age=age;

this.name=name;

}

public int compareTo(Student i){

return this.name.compareTo(i.name);

}

@Override

public String toString() {

return "Student{" +

"age=" + age +

", name=" + name +

'}';

}

}

StudentCollection.java

import java.util.\*;

class StudentCollection{

public static void main(String args[]){

Student s1=new Student(21,"Alex");

Student s2=new Student(15,"James");

Student s3=new Student(42,"Toe");

Student s4=new Student(25,"Bell");

List<Student> l=new ArrayList<Student>();

l.add(s1);

l.add(s2);

l.add(s3);

l.add(s4);

AgeComparator age=new AgeComparator();

NameComparator name=new NameComparator();

Collections.sort(l,name);

l.forEach(x->System.out.println(x));

}

}

AgeComparator.java

import java.util.\*;

class AgeComparator implements Comparator<Student>{

public int compare(Student i1,Student i2){

if(i1.age>i2.age)

return 1;

else

return -1;

}

}

NameComparator.java

import java.util.\*;

class NameComparator implements Comparator<Student>{

public int compare(Student i1,Student i2){

return i1.name.compareTo(i2.name);

}

}