Session 10

Algorithms

Objectives

- Sorting
- Shuffling
- Routine Data Manipulation
- Searching
- Composition
- Finding Extreme Values

Introduction

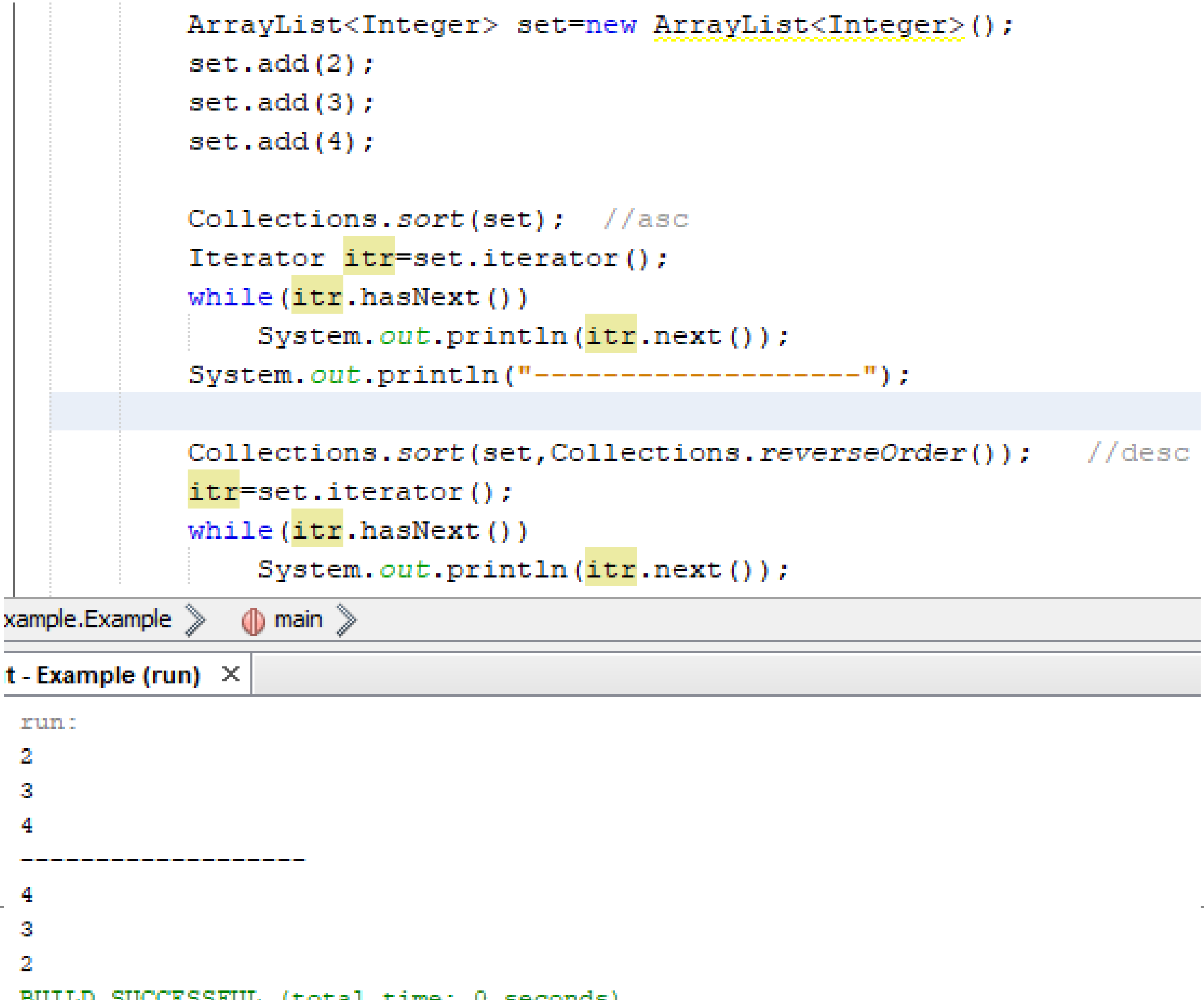
- The *polymorphic algorithms* described here are pieces of reusable functionality provided by the Java platform.
- All of them come from the Collections class, and all take the form of static methods whose first argument is the collection on which the operation is to be performed.

Sorting

- The sort algorithm reorders a List so that its elements are in ascending order according to an ordering relationship.
- Example

```
public class Sort {
    public static void main(String[] args) {
        List<String> list = Arrays.asList(args);
        Collections.sort(list);
        System.out.println(list);
    }
}
```

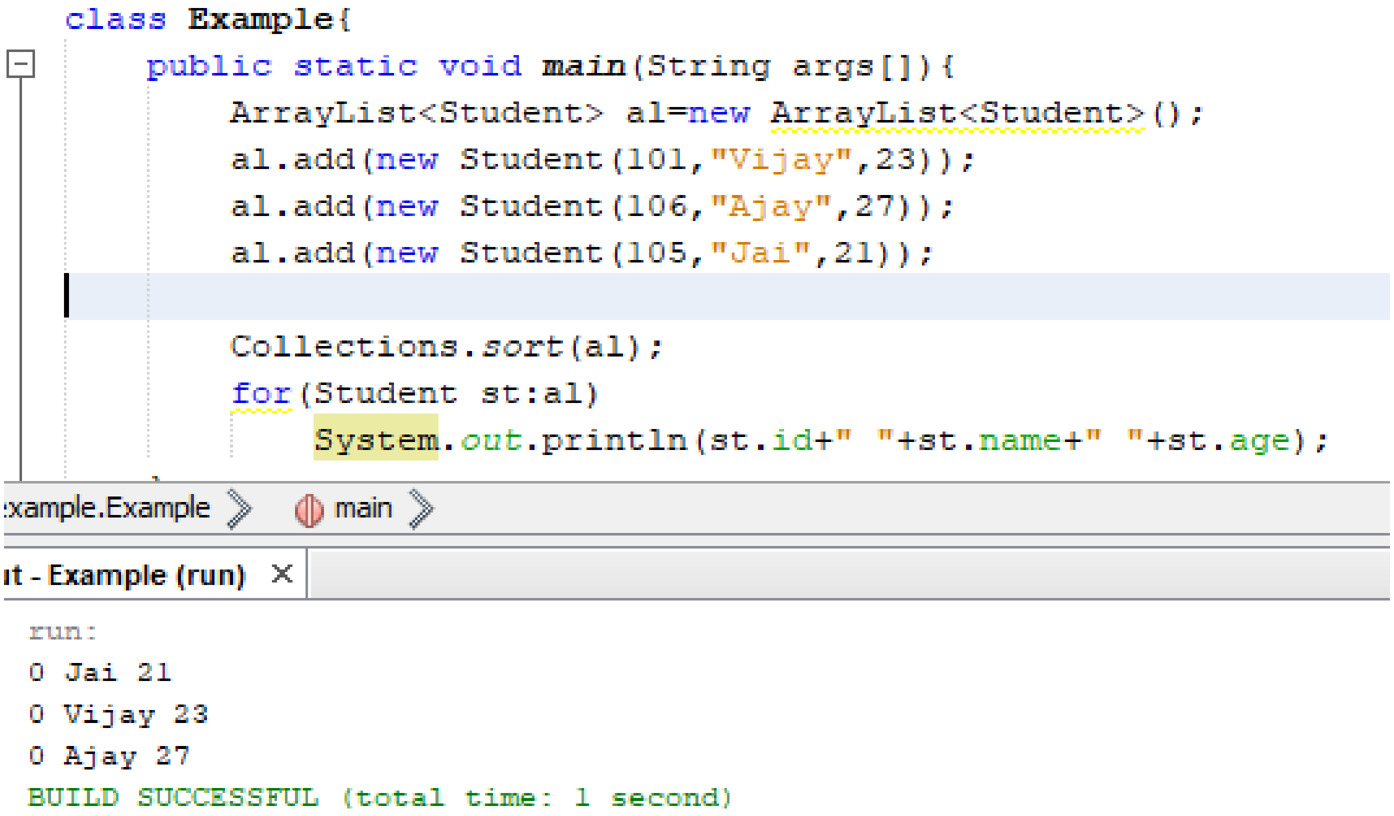
Sorting



```
ArrayList<Integer> set=new ArrayList<Integer>();
set.add(2);
set.add(3);
set.add(4);

Collections.sort(set);  //asc
Iterator itr=set.iterator();
while(itr.hasNext())
    System.out.println(itr.next());
System.out.println("-------------------");

Collections.sort(set,Collections.reverseOrder());   //desc
itr=set.iterator();
while(itr.hasNext())
    System.out.println(itr.next());
```

```
run:
2
3
4
-------------------
4
3
2
BUILD SUCCESSFUL (total time: 0 seconds)
```

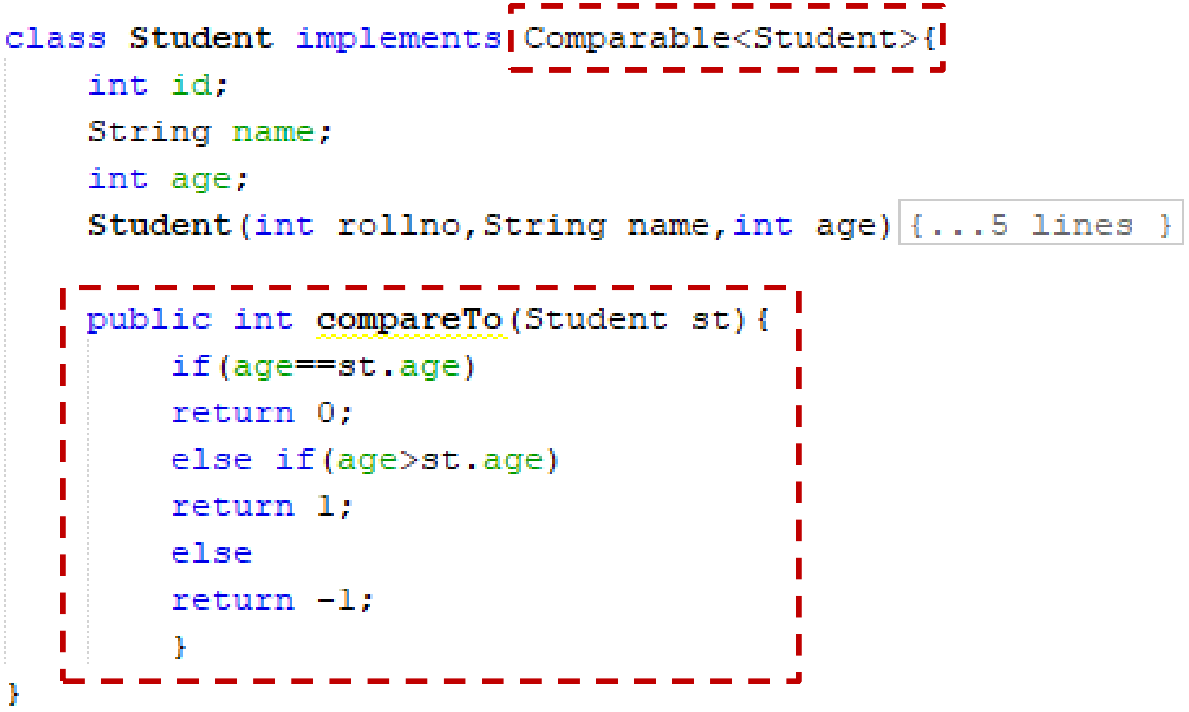
Sorting - compareTo



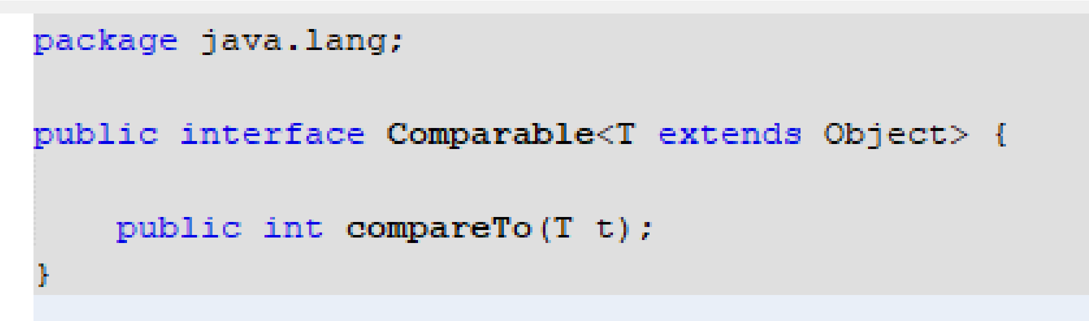
```
class Example{
    public static void main(String args[]){
        ArrayList<Student> al=new ArrayList<Student>();
        al.add(new Student(101,"Vijay",23));
        al.add(new Student(106,"Ajay",27));
        al.add(new Student(105,"Jai",21));

        Collections.sort(al);
        for(Student st:al)
            System.out.println(st.id+" "+st.name+" "+st.age);
```

```
run:
0 Jai 21
0 Vijay 23
0 Ajay 27
BUILD SUCCESSFUL (total time: 1 second)
```

Sorting - compareTo

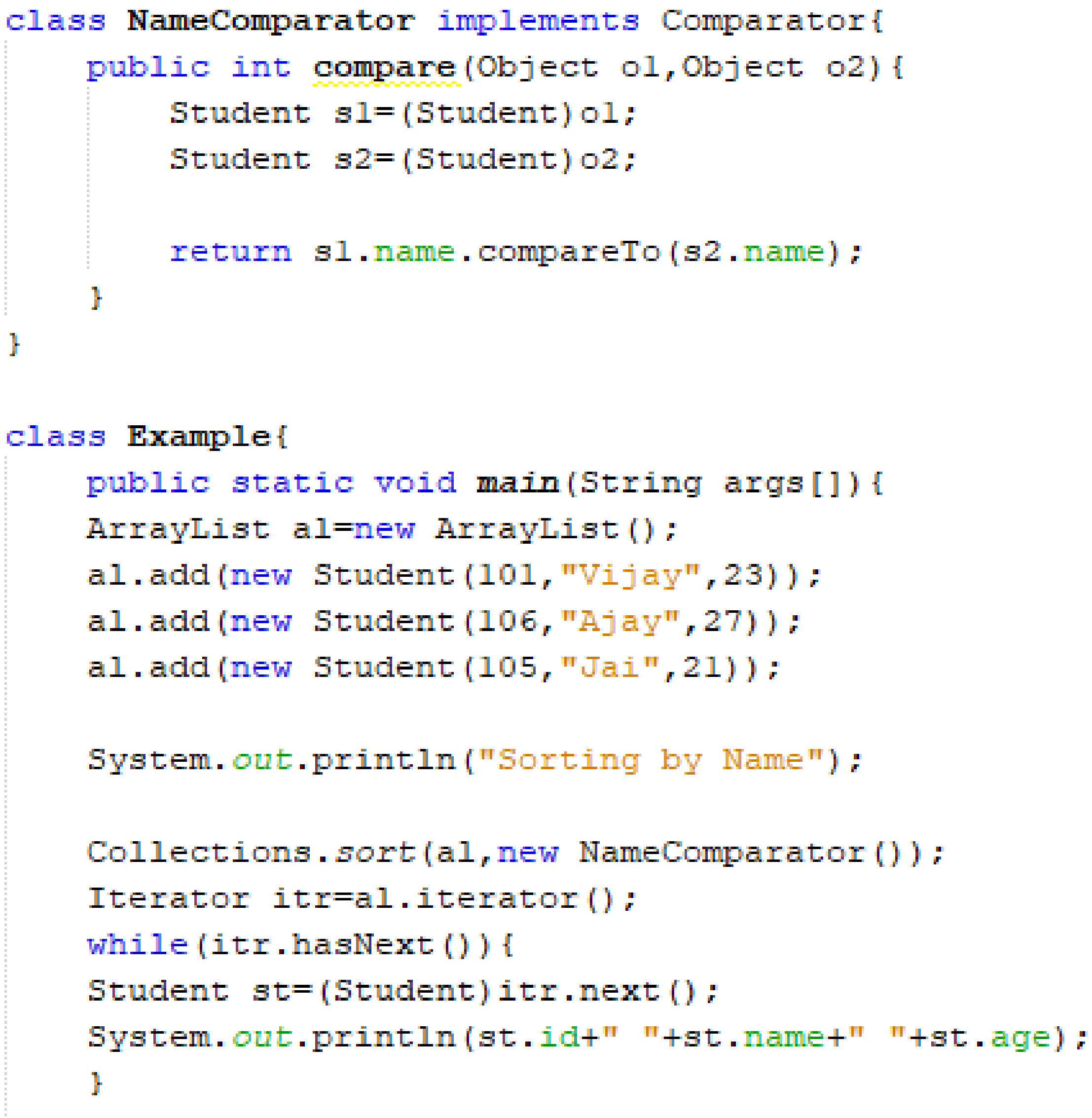


```
class Student implements Comparable<Student>{
    int id;
    String name;
    int age;
    Student(int rollno,String name,int age){...5 lines }

    public int compareTo(Student st){
        if(age==st.age)
        return 0;
        else if(age>st.age)
        return 1;
        else
        return -1;
        }
}
```



```
package java.lang;

public interface Comparable<T extends Object> {

    public int compareTo(T t);
}
```

Comparator Interface



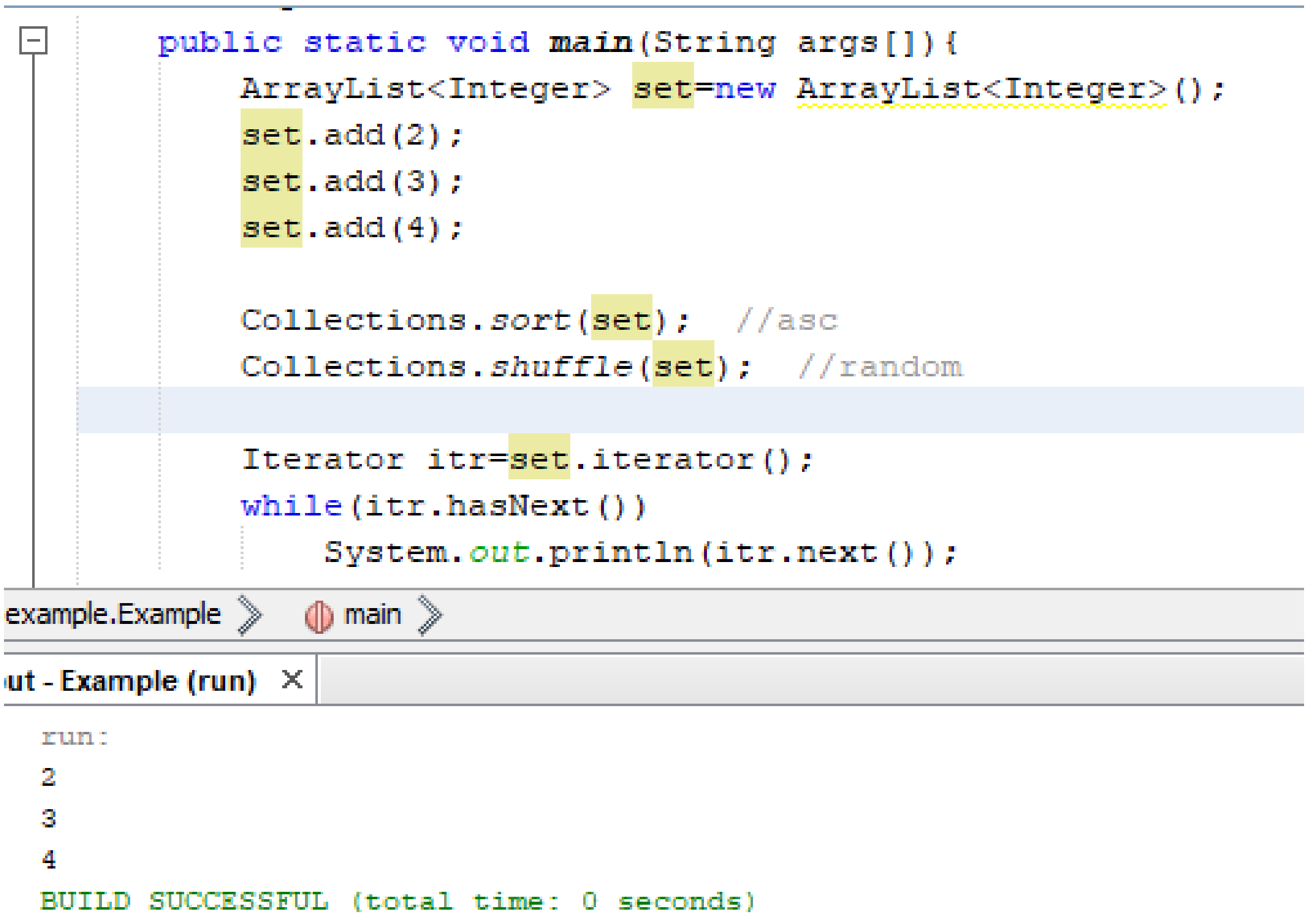
```
class NameComparator implements Comparator{
    public int compare(Object o1,Object o2){
        Student s1=(Student)o1;
        Student s2=(Student)o2;

        return s1.name.compareTo(s2.name);
    }
}

class Example{
    public static void main(String args[]){
    ArrayList al=new ArrayList();
    al.add(new Student(101,"Vijay",23));
    al.add(new Student(106,"Ajay",27));
    al.add(new Student(105,"Jai",21));

    System.out.println("Sorting by Name");

    Collections.sort(al,new NameComparator());
    Iterator itr=al.iterator();
    while(itr.hasNext()){
    Student st=(Student)itr.next();
    System.out.println(st.id+" "+st.name+" "+st.age);
    }
```

Shuffling

- The shuffle algorithm does the opposite of what sort does, destroying any trace of order that may have been present in a List.
- This algorithm reorders the List based on input from a source of randomness.

```
Collections.shuffle(arrlist);
```

Shuffling



```
    public static void main(String args[]){
        ArrayList<Integer> set=new ArrayList<Integer>();
        set.add(2);
        set.add(3);
        set.add(4);

        Collections.sort(set);  //asc
        Collections.shuffle(set);  //random

        Iterator itr=set.iterator();
        while(itr.hasNext())
            System.out.println(itr.next());
```

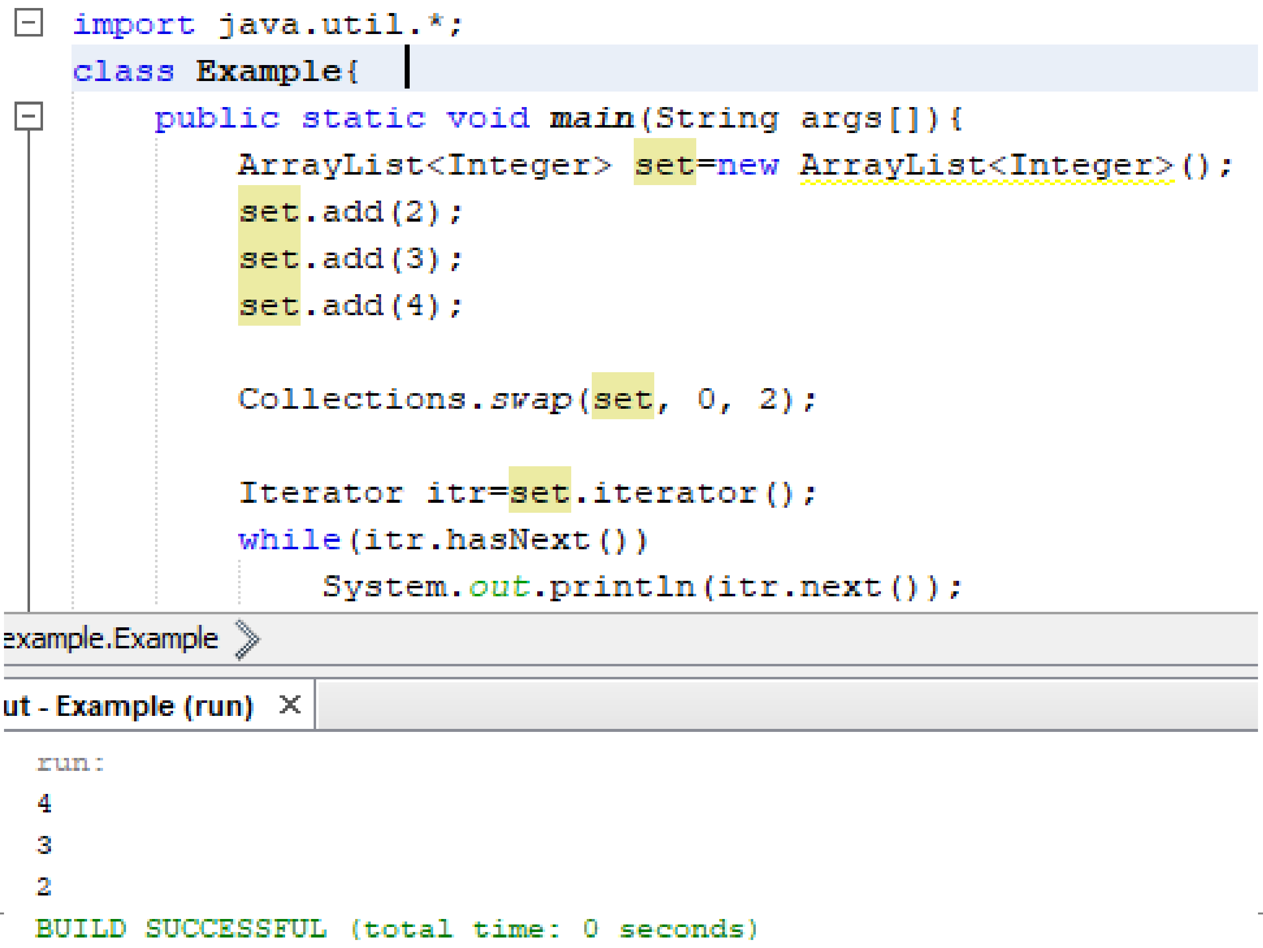
```
run:
2
3
4
BUILD SUCCESSFUL (total time: 0 seconds)
```

Routine Data Manipulation

- The Collections class provides five algorithms for doing routine data manipulation on List objects, including:
  - reverse()
  - fill()
  - copy()
  - swap()
  - addAll()

Routine Data Manipulation

- swap()



```
import java.util.*;
class Example{
    public static void main(String args[]){
        ArrayList<Integer> set=new ArrayList<Integer>();
        set.add(2);
        set.add(3);
        set.add(4);

        Collections.swap(set, 0, 2);

        Iterator itr=set.iterator();
        while(itr.hasNext())
            System.out.println(itr.next());
```

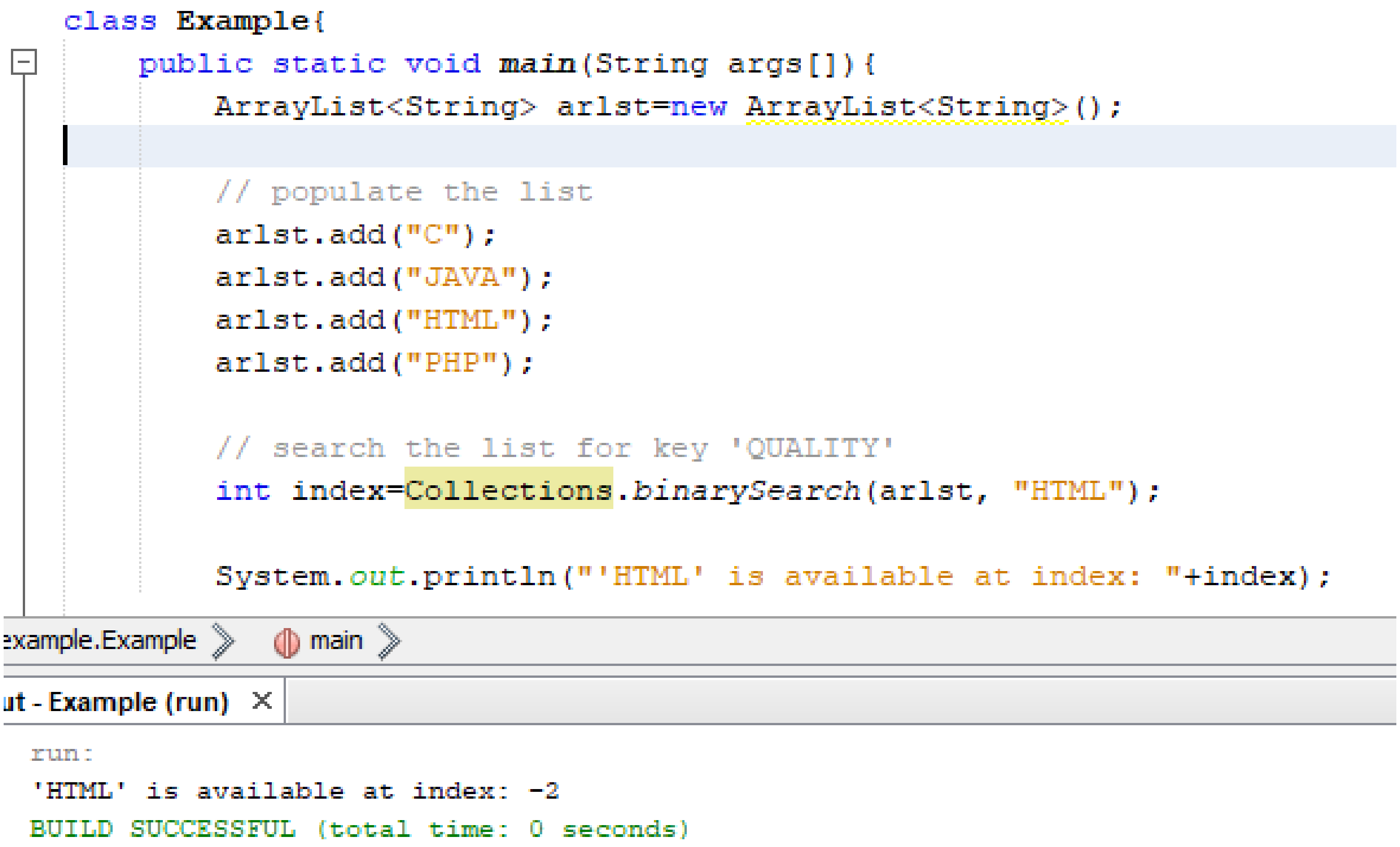
```
run:
4
3
2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Searching

- The binarySearch algorithm searches for a specified element in a sorted List.

```
int pos = Collections.binarySearch(list, key);
if (pos < 0) l.add(-pos-1, key);
```

Searching (2)



```
class Example{
    public static void main(String args[]){
        ArrayList<String> arlst=new ArrayList<String>();

        // populate the list
        arlst.add("C");
        arlst.add("JAVA");
        arlst.add("HTML");
        arlst.add("PHP");

        // search the list for key 'QUALITY'
        int index=Collections.binarySearch(arlst, "HTML");

        System.out.println("'HTML' is available at index: "+index);
```

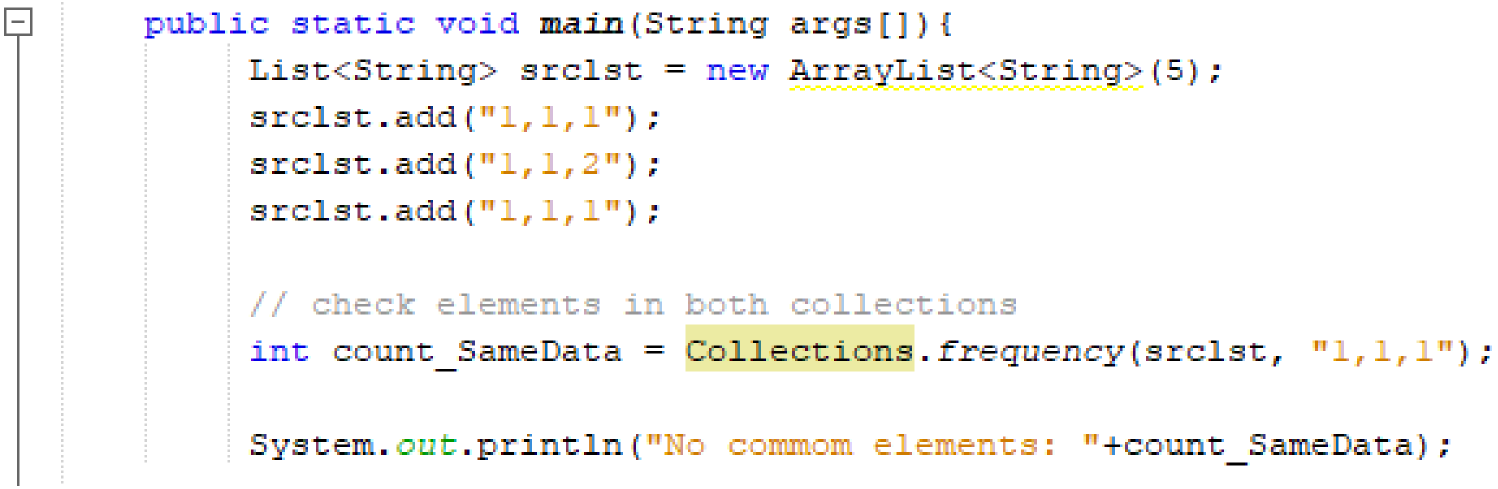
```
run:
'HTML' is available at index: -2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Composition

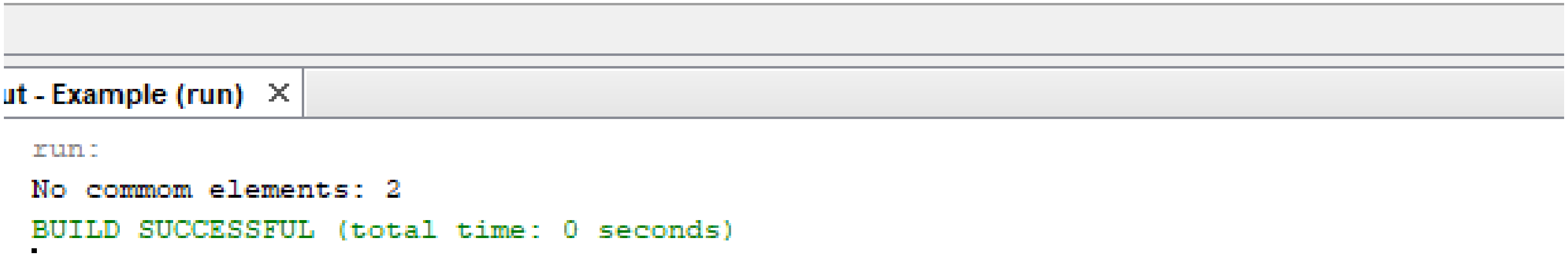
- `frequency` — counts the number of times the specified element occurs in the specified collection.
- `disjoint` — determines whether two Collections are disjoint; that is, whether they contain no elements in common.

Composition

- frequency



```
public static void main(String args[]){
    List<String> srclst = new ArrayList<String>(5);
    srclst.add("1,1,1");
    srclst.add("1,1,2");
    srclst.add("1,1,1");

    // check elements in both collections
    int count_SameData = Collections.frequency(srclst, "1,1,1");

    System.out.println("No commom elements: "+count_SameData);
```

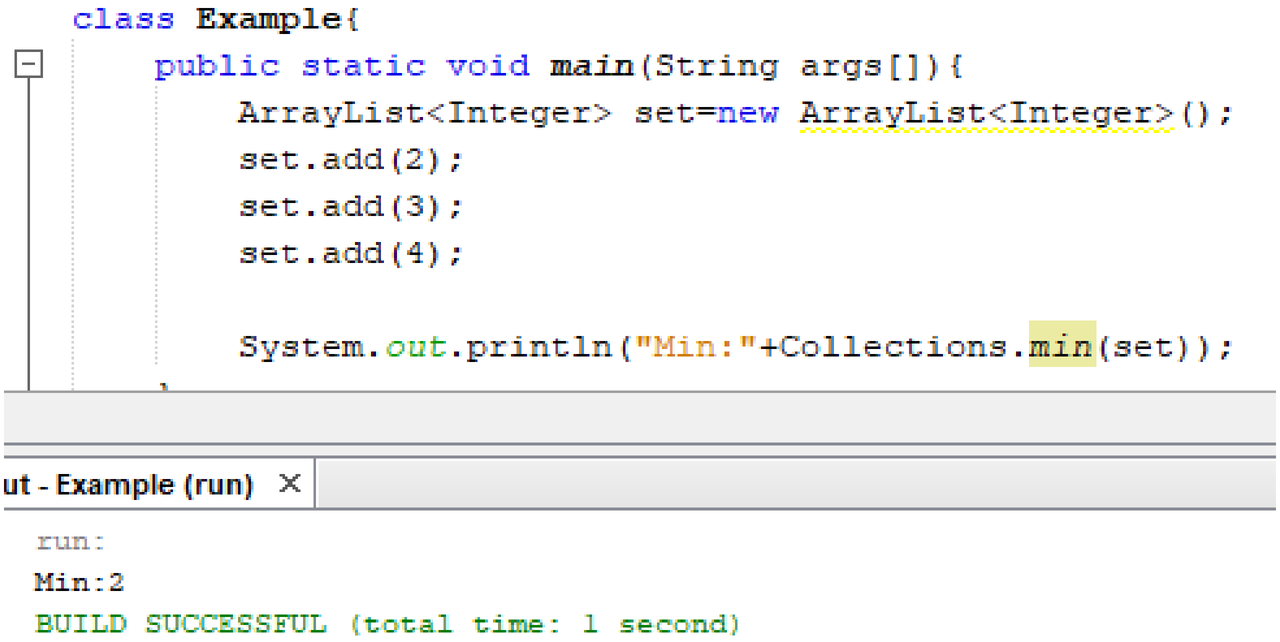


```
run:
No commom elements: 2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Finding Extreme Values

- The min and the max algorithms return, respectively, the minimum and maximum element contained in a specified Collection.

Finding Extreme Values



```
class Example{
    public static void main(String args[]){
        ArrayList<Integer> set=new ArrayList<Integer>();
        set.add(2);
        set.add(3);
        set.add(4);

        System.out.println("Min:"+Collections.min(set));
```

```
run:
Min:2
BUILD SUCCESSFUL (total time: 1 second)
```

Summary

- Sorting
- Shuffling
- Routine Data Manipulation
- Searching
- Composition
- Finding Extreme Values

Student Activity

- Use Student Class
- **Entry 5 students**
- **Print Student List with GPA desc**
- Print Stundents whose year of birth =2000