Java programming

Nathan Warner

Northern Illinois University

Computer Science
Northern Illinois University
United States

Contents

User Input (scanner)

The Scanner class is used to get user input, and it is found in the **java.util** package.

```java
import java.util.Scanner;  // Import the Scanner class

class Main {
    public static void main(String[] args) {
        Scanner myObj = new Scanner(System.in);  // Create a
    ↪  Scanner object
        System.out.println("Enter username");

        String userName = myObj.nextLine();  // Read user input
        System.out.println("Username is: " + userName);  //
    ↪  Output user input
    }
}
```

1.1 Input Types

- **nextBoolean()** Reads a boolean value from the user
- **nextByte()** Reads a byte value from the user
- **nextDouble()** Reads a double value from the user
- **nextFloat()** Reads a float value from the user
- **nextInt()** Reads a int value from the user
- **nextLine()** Reads a String value from the user
- **nextLong()** Reads a long value from the user
- **nextShort()** Reads a short value from the user

1.2 Checks

- **hasNextBoolean()**
- **hasNextByte()**
- **hasNextDouble()**
- **hasNextFloat()**
- **hasNextInt()**
- **hasNextLine()**
- **hasNextLong()**
- **hasNextShort()**

Arrays

2.1 Important methods

These static methods are found in `java.util.Arrays`

- **Arrays.fill()**: Fills all elements of the specified array with the specified value.
- **Arrays.equals()**: Returns a Boolean true value if both arrays are of the same type and all of the elements within the arrays are equal to each other.
- **Arrays.copyOf()**: Copies the specified array, truncating or padding with default values if necessary so the copy has the specified length.
- **Arrays.copyOfRange()**: Copies the specified range from the index1 element up to, but not including, the index2 element of the specified array into a new array
- **Arrays.sort()**
- **Arrays.binarySearch**

2.2 Sorting

2.3 The Comparable Interface

In Java, the `Comparable<T>` interface (in java.lang) lets a class define its natural ordering by implementing a single method:

```java
public interface Comparable<T> {
    int compareTo(T other);
}
```

Enables objects to be sorted (e.g. by Collections.sort() or Arrays.sort()), or used in sorted collections (e.g. TreeSet, TreeMap).

Contract:

- **this.compareTo(other)** $< \mathbf{0}$ means this precedes other
- $== \mathbf{0}$ means they're considered equal in ordering
- $> \mathbf{0}$ means this follows other

```java
import java.util.Scanner;
import java.util.Collections;
import java.util.ArrayList;
import java.util.List;

public class t1 implements Comparable<t1> {
    public int x,y;

    public t1(int x, int y) { this.x = x; this.y = y; }

    @Override
    // Ascending
    public int compareTo(t1 other) {
        if (this.x == other.x) return 0;
        else if (this.x > other.x) return 1;
        else return -1;
    }

    @Override
    // Descending
    public int compareTo(t1 other) {
        if (this.x == other.x) return 0;
        else if (this.x > other.x) return -1;
        else return 1;
    }

    public static void main(String[] args) {
        ArrayList<t1> arr = new ArrayList<>(List.of(new t1(4,2),
  ↪ new t1(2,6), new t1(1,8), new t1(9,18), new t1(5,0)));

        Collections.sort(arr);

        for (t1 item : arr) {
            System.out.println("(" + item.x + "," + item.y +
  ↪ ")");
        }
    }
}
```

2.4 Comparator

The Comparator<T> interface (in java.util) defines a custom ordering for objects—even if the class itself doesn't implement Comparable. It has one primary method

```
public interface Comparator<T> {
    /**
    * Compares its two arguments for order.
    *
    * @param o1 the first object to be compared.
    * @param o2 the second object to be compared.
    * @return a negative integer if o1 <  o2,
    *         zero                if o1 == o2,
    *         a positive integer if o1 >  o2.
    */
    int compare(T o1, T o2);

}
```

We can use it to define an ordering for objects without implementing the Comparable interface

```java
import java.util.Collections;
import java.util.Comparator;
import java.util.ArrayList;
import java.util.List;

public class t1 {
    public int x, y;

    public t1(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public static void main(String[] args) {
        List<t1> arr = new ArrayList<>(List.of( new t1(4,2), new
↪   t1(2,6), new t1(1,8), new t1(9,18), new t1(5,0)));

        // 1) Create a Comparator that compares by x:
        Comparator<t1> byX = new Comparator<>() {
            @Override
            public int compare(t1 a, t1 b) {
                // Integer.compare handles a.x < b.x, ==, >
                return Integer.compare(a.x, b.x);
            }
        };

        // 2) Sort using that Comparator:
        Collections.sort(arr, byX);

        // 3) Print out:
        for (t1 item : arr) {
            System.out.println("(" + item.x + "," + item.y +
↪   ")");
        }
    }
}
```