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.

```
0  import java.util.Scanner;  // Import the Scanner class
1
2  class Main {
3      public static void main(String[] args) {
4          Scanner myObj = new Scanner(System.in);  // Create a
           ↪  Scanner object
5          System.out.println("Enter username");
6
7          String userName = myObj.nextLine();  // Read user input
8          System.out.println("Username is: " + userName);  //
           ↪  Output user input
9      }
10 }
```

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:

```
0  public interface Comparable<T> {
1      int compareTo(T other);
2  }
```

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)** $<$ **0** means this precedes other
- $==$ **0** means they're considered equal in ordering
- $>$ **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

```
0  public interface Comparator<T> {
1      /**
2      * Compares its two arguments for order.
3      *
4      * @param o1 the first object to be compared.
5      * @param o2 the second object to be compared.
6      * @return a negative integer if o1 <  o2,
7      *         zero                 if o1 == o2,
8      *         a positive integer if o1 >  o2.
9      */
10     int compare(T o1, T o2);
11
12 }
```

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 + ")");
        }
    }
}
```

Classes and Objects: A deeper look

3.1 Default and No Argument Constructors

Every class must have at least one constructor. If you do not provide any in a class's declaration, the compiler creates a default constructor that takes no arguments when it's invoked. The default constructor initializes the instance variables to the initial values specified in their declarations or to their default values (zero for primitive numeric types, false for boolean values and null for references)

Recall that if your class declares constructors, the compiler will not create a default constructor. In this case, you must declare a no-argument constructor if default initialization is required.

3.2 Enum types

Like classes, all enum types are reference types. An enum type is declared with an enum declaration, which is a comma-separated list of enum constants.

The declaration may optionally include other components of traditional classes, such as constructors, fields and methods

- enum constants are implicitly final.
- enum constants are implicitly static.
- Any attempt to create an object of an enum type with operator new results in a compilation error.

The enum constants can be used anywhere constants can be used, such as in the case labels of switch statements and to control enhanced for statements.

```java
public enum e {
    // First, declare constants
    a(1,2),
    b(2,3),
    c(3,4);

    // The fields of those constants
    private final int x,y;

    // Constructor
    e(int x, int y) {
        this.x = x;
        this.y = y;
    }

    // Optional sets and gets

    // So we can run the program to see no errors
    public static void main(String[] args)  {

    }
}
```

For every enum, the compiler generates the static method values that returns an array of the enum's constants.

When an enum constant is converted to a String, the constant's identifier is used as the String representation.

```java
public enum e{
    a,b,c;

    public static void main(String[] args) {
        e E = a;

        for (e item : E.values()) {
            System.out.println(item);
        }
        // a
        // b
        // c
    }
}
```

```java
import java.util.EnumSet;

public enum e{
    a,b,c,d;

    public static void main(String[] args) {
        e E = a;

        EnumSet<e> es = EnumSet.range(e.a, e.d);

        for (e item : es) System.out.print(item + " ");
        // a b c d
    }
}
```

3.3 Static import

Imports all static members of a class (which is known as static import on demand)

The following syntax imports a particular static member

```java
import static packageName.ClassName.staticMemberName;
```

Validating and Converting data types

4.1 String to integer

In Java, you can validate and convert a string to an integer safely using either Integer.parseInt() or Integer.valueOf(), but you should handle exceptions in case the string isn't a valid integer.

```
String input = "123";
int number;

try {
    number = Integer.parseInt(input);
    System.out.println("Valid integer: " + number);
} catch (NumberFormatException e) {
    System.out.println("Invalid input: not a valid integer.");
}
```

Java regex

5.1 Includes

```
import java.util.regex.*;
```

5.2 Create a pattern

```
Pattern pattern = Pattern.compile(string pattern);
```

5.3 Create a matcher

```
Matcher matcher = pattern.matcher(string source);
```

5.4 Yes / no match test

```
boolean found = matcher.matches(); // returns true if entire
↪  string matches
```

5.5 Matcher methods

- **boolean matches()**: Checks if the entire input matches the pattern
- **boolean find()**: Looks for the next occurrence of the pattern
- **String group()**: Returns the text matched by the last find() (default to zero)
- **String group(int number)**: Returns specific capture (zero is full match)
- **int start() / end()**: Returns the start/end indices of the match
- **int start(int groupNumber) / end(int groupNumber)**: Returns the start/end indices of the match
- **string replaceAll(String)**: Replaces all matches with a given string
- **string replaceFirst(String)**: Replaces first match only

5.5.1 replaceAll

This method replaces every substring that matches the regex with a replacement string. We can also use back references as arguments.

```
String old = "abcfooxyz";
// Replace string with captured "foo"
System.out.println(Pattern.compile("\\w*(foo)\\w*").matcher(old)
    .replaceAll("$1"));
```

5.6 Getting captures

We use .find() and .group()

```
Pattern pattern = Pattern.compile("(\\w+)@(\\w+)\\.(\\w+)");
Matcher matcher = pattern.matcher("a@b.com and c@d.org");

while (matcher.find()) {
    System.out.println("Full: " + matcher.group(0));
    System.out.println("User: " + matcher.group(1));
    System.out.println("Domain: " + matcher.group(2));
    System.out.println("TLD: " + matcher.group(3));
    System.out.println();
}
```

5.7 Flags

- **Pattern.CASE_INSENSITIVE**: Makes matching ignore case
- **Pattern.MULTILINE**: Makes ∧ and $ match line boundaries
- **Pattern.DOTALL**: Makes . match newline \n as well
- **Pattern.UNICODE_CASE**: Enables Unicode-aware case folding

5.8 Shortcut

You don't always need Pattern and Matcher:

```
System.out.println("abc123".matches("\\w+\\d+")); // true
```