L46

Friday, February 4, 2022 5:25 PM

Utility Classes – Date, DateFormat, Calendar, Randon number generator

Arrays and Collections

Q1. Java collection framework has included a very useful utility class called Arrays.

Please note that the class we are talking about java.util.Arrays is completely different from the class java.lang.reflect.Array.

java.lang.reflect.Array class is part of java.lang.reflect package, and it allows for dynamic creation and manipulation of an array.

Arrays on other hand is a class in java.util package.

Arrays is a utility class. A Utility class is a class which contains static methods and has a private constructor so that it cannot be instantiated.

This class has many useful searching and sorting methods which operate on arrays.

See and retype the below code to know the usage of a few commonly used methods like :

- asList(T...arrayElement)
- sort(int[] array)
- fill(int[] array, int value)
- equals(int[] array, int value)
- copyOfRange(int[] original, int from, int to)

Note in the below code the **main** method uses ellipses(...) instead of a **String[]**.

```
package q11313;
import java.util.*;
public class ArraysDemo {
    public static void main(String ... args) {
        List<Integer> integerList = Arrays.asList(3, 33, 333, 3333);
        System.out.println("integerList : " + integerList);
        Integer[] integerArr = {382, 34, 4, 223, 331};
        System.out.println("Original integerArr : " + Arrays.asList(integerArr));
        Arrays.sort(integerArr);
        System.out.println("After sorting integerArr : " + Arrays.asList(integerArr));
        Integer[] integerArr2 = {4, 34, 223, 331, 382};
        System.out.println("Arrays.equals(integerArr, integerArr2) : " + Arrays.equals(integerArr, integerArr2));
        Integer[] integerArr3 = new Integer[10];
        Arrays.fill(integerArr3, 7);
        System.out.println("After filling integerArr3 with 7 : " + Arrays.asList(integerArr3));
    }
}

```

Q2. Like the class Arrays which deals with arrays, Java collection framework also has a utility class called Collections which deals with collections.

Please note that the class java.util.Collections is completely different from the interface java.util.Collection.

java.util.Collection class is the root interface for all the collection classes.

Collections on other hand is a utility class, which contains static methods and has a private constructor so that it cannot be instantiated.

This class has many useful searching and sorting methods which operate on various collections.

See and retype the below code to know the usage of a few commonly used methods like :

- shuffle(List<?> list)
- sort(List<T> list)
- fill(List<? super T> list, T obj)

Note in the below code the **main** method uses ellipses(...) instead of a **String[]**.

```
package q11314;
import java.util.*;
public class CollectionsDemo {
    public static void main(String ... args) {
        List<Integer> integerList = Arrays.asList(3, 33, 333, 3333, 33333, 333333);
        System.out.println("integerList : " + integerList);
        Collections.shuffle(integerList, new Random(1));
        System.out.println("After shuffle integerList : " + integerList);
        Collections.sort(integerList);
        System.out.println("After sort integerList : " + integerList);
        Collections.fill(integerList, 7);
        System.out.println("After filling integerList with 7 : " + integerList);
    }
}

```

Date, DateFormat and Calendar

Q1. The central classes in Java for working with Date and Time are Date, DateFormat and Calendar.

Date and Calendar classes are present in java.util package, while DateFormat is present in java.text package.

Starting with **Java version 8**, many new classes and enhancements were introduced for handling date and time, these are bundled in a new package named java.time.

We will learn about the new classes introduced in **Java version 8** later.

Since a lot of code written before Java 8, heavily use Date, DateFormat and Calendar classes, we will first learn about them.

1. Date - represents an instance in time (stored as a primitive long). For example this moment. Though the name might suggest that it represents only a calendar date, like 7th July 1977, we should also remember that it actually has the time component to the precision of a millisecond.
2. DateFormat - provides the formating for dates and times on an given Locale
3. Calendar - provides methods to work with the instance of time represented by Date.

It is important to note that earlier Unix systems used a 32-bit signed integer to store time. They started counting time with the value 0 representing 1970-1-1 00:00:00. This is referred as the epoch time.

The time component represented by the Date object counts the milliseconds passed from the above mentioned epoch time.

See and retype the below code.

```
package q11315;
import java.util.*;
public class DateDemo {
    public static void main(String ... args) {
        Date thisMoment = new Date();
        long millisecondsSinceEpochStart = thisMoment.getTime();
        System.out.println("This Moment : " + thisMoment);
        System.out.println("Total milli seconds from epoch to this moment : " + millisecondsSinceEpochStart);
    }
}
```

Q2. The Date class is a simple wrapper over an instance of time, represented as a long, just like the **Integer** class is a wrapper over an **int** value.

A Date object can be created using one of the below two constructors:

1. Date() - creates a new instance of Date object which represents the time at which it is created.
2. Date(long time) - creates a new instance of Date object which represents the specified number of milliseconds specified by the variable time, starting from the "the epoch", namely January 1, 1970, 00:00:00 GMT

The various methods in the Date class like getDay(), getHour(), getYear() etc., are all deprecated **and should not be used.**

We should instead use their alternatives provided in the Calendar class. We will learn more about the Calendar class later.

Date class has one useful method called getTime() which returns the number of milliseconds starting from January 1, 1970, 00:00:00 GMT represented by a date object as a long.

Note: System class also has a method called System.currentTimeMillis() which will return the current time as milliseconds from the **epoch time** (January 1, 1970, 00:00:00 GMT) as a long value.

See and retype the below code.

```
package q11397;
import java.util.*;
public class DateExample {
    public static void main(String ... args) throws InterruptedException {
        System.out.println("Sleeping for 1 second...");
        Date timeAsDate = new Date();
        long oldTimeInMillis = timeAsDate.getTime();
        Thread.sleep(1000);
        long newCurrentTimeInMillis = System.currentTimeMillis();
        if (newCurrentTimeInMillis - oldTimeInMillis >= 1000) {
            System.out.println("This thread resumed after 1 or more seconds");
        }
    }
}
```

Q3. DateFormat is an abstract class which provides methods for parsing text to Date objects and formating Date objects to text.

DateFormat class has static methods to create instance of DateFormat objects.

```
DateFormat df1 = DateFormat.getInstance();
// the above code creates a date and time formatter instance which uses the SHORT style
DateFormat df2 = DateFormat.getDateInstance();
// the above line creates a date foramatter instance with system's Locale
DateFormat df3 = DateFormat.getDateInstance(DateFormat.LONG, Locale.FRANCE);
// the above line creates a date formatter instance with French Locale in long format
```

A Locale object represents a specific geographical, political, or cultural region and their attributes. For example, the month August is called **August** in English and **Août** in French.

The Locale class allows for displaying numbers, currencies, date and time information according to user's native language, country and cultural settings.

SimpleDateFormat is one of the most useful implementation sub-class of DateFormat class.

Note that the formatting classes like DateFormat and SimpleDateFormat are present in the java.text package.

Write a class named DateFormatInJapan with a static method getJapaneseShortDate(). The method should accept one parameter of type Date. It should return a formatted date as **String** from the passed parameter using **JAPAN** locale in **SHORT** format.

Sample Output:

```
Oct 2, 1869 in Japan is: 1869/10/02
```

```java
package q11398;
import java.util.*;
import java.text.*;
public class DateFormatInJapan {
    public static String getJapaneseShortDate(Date date) {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd", Locale.JAPAN);
        return sdf.format(date);
    }
}

```

Q4. SimpleDateFormat is a concrete implementation of the abstract class DateFormat.

An instance of SimpleDateFormat class can be created using one of the below four constructors:

1. SimpleDateFormat() - creates default SimpleDateFormat object with default locale.
2. SimpleDateFormat(String pattern) - creates a SimpleDateFormat object with the given pattern.
3. SimpleDateFormat(String pattern, DateFormatSymbols formatSymbols) - creates a SimpleDateFormat object with the given pattern and custom date format symbols.
4. SimpleDateFormat(String pattern, Locale locale) - creates a SimpleDateFormat object with the given pattern and locale.

The pattern string is a sequence of characters which have special meanings. Click here to view all the pattern characters with their meanings under the section titled ***Date and Time Patterns*.**

After a SimpleDateFormat object is created applyPattern(String pattern) method can be used to modify the pattern.

Below are the two important methods of DateFormat that can be called on a SimpleDateFormat instance:

1. format(Date date) - formats the Date object as text.
2. parse(String dateText) - creates a Date object by parsing the dateText.

Note that if text has to be included as it is in the pattern, the text has to be surrounded in **single quotes.**

Write a class with name SimpleDateFormateDemo. In the main method create a Date object by parsing the given dateText 15-08-1947 and print the same.

The output should be

```
parsedDate : Fri Aug 15 00:00:00 GMT 1947
```

```java
package q11399;

import java.util.*;
import java.text.*;

public class SimpleDateFormatDemo {
    public static void main(String... args) throws ParseException {

        String dateText = "15-08-1947";

        SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");

        Date parsedDate = sdf.parse(dateText);

        SimpleDateFormat sdf1 = new SimpleDateFormat("EEE MMM dd HH:mm:ss zzz yyyy", Locale.US);

        System.out.println("parsedDate : " + sdf1.format(parsedDate));
    }
}

```

Q5. Calendar is an abstract class, which means we cannot create an instance of a Calendar class directly.

The Calendar class provides a static method called Calendar.getInstance(), which returns an object of type Calendar.

```
Calendar currentTime = Calendar.getInstance();
```

The above getInstance() method creates a Calendar object which is initialized with the current system time at that moment.

In most cases getInstance() returns an instance of GregorianCalendar which is a subclass of Calendar.

Also note that the above method uses the system's default locale, we can use another method Calendar.getInstance(Locale locale) to create a Calendar object with a specific locale. (We will learn about Locale when we learn about DateFormat)

Calendar also has two methods to reset the time component it represents, after a calendar instance is created.

1. setTime(Date newTime) - (note it has a corresponding getTime() method which returns a Date object for the Calendar's time component.)
2. setTimeInMillis(long timeInMilliSeconds)

Calendar has a generic method called get(int field). The field value can be any of the fields declared in the Calendar class like: Calendar.YEAR, Calendar.MONTH, Calendar.DAY_OF_MONTH, Calendar.WEEK_OF_MONTH, Calendar.DAY_OF_YEAR, Calendar.HOUR, Calendar.HOUR_OF_DAY, Calendar.MINUTE, Calendar.SECOND, Calendar.MILLISECOND, etc.

Similar to the get(int field) method, Calendar also has a set(int field, int value) method which is used to reset any of the above mentioned fields in a Calendar object.

(Note that the value returned for MONTH field starts with 0. For example - value for January is 0, February is 1 ...)

Write the missing code in the below program. A calendarDemo class is created with a setTime method which passes one argument. SimpleDateFormat is used to parse the string and produce the date. Calendar setTime method is used to set time to the value of date.

```java
package q11589;
import java.util.*;
import java.text.SimpleDateFormat;
import java.text.ParseException;
public class CalendarDemo {
    public String setTime(String dateString) throws ParseException {
        // use the correct format string as argument for the constructor
        SimpleDateFormat sd = new SimpleDateFormat("MMM-dd-yyy");

        // parses text from the beginning of the given string to produce a date
        Date date =  sd.parse(dateString);


        Calendar currentTime = Calendar.getInstance();

        //// configuring the current object to the value of date
        currentTime.setTime(date);

        // format method on SimpleDateFormat returns a string representation of the passed date
        return sd.format(currentTime.getTime());
    }
}
```

Q6. Calendar class provides many useful methods for date and time manipulations.

1. roll(int field, int amount)
2. add(int field, int amount)
3. compareTo(Calendar anotherCalendar) - returns 1 if greater, 0 if equal and -1 if lesser

Write the missing code in the below program.

In this program CalendarAddition class is created with a method name addDays which passes two arguments to add the specified amount of time to the given calendar field. Parse the string using simpleDateFormate. In add method add the number of days to the calendar field.

Consider the following example for your understanding.
Adding (10) days to Feb-19-2005 gives Mar-01-2005

```java
package q11590;

import java.util.*;
import java.text.SimpleDateFormat;
import java.text.ParseException;

public class CalendarAddition {
    public String addDays(String dateString, int days) throws ParseException {
        SimpleDateFormat sd = new SimpleDateFormat("MMM-dd-yyy");

        Date date = sd.parse(dateString);

        Calendar cal = Calendar.getInstance();

        cal.setTime(date);

        cal.add(Calendar.DAY_OF_MONTH, days);

        return sd.format(cal.getTime());

    }
}
```

Q7. Given :

```
import java.util.Date;
import java.text.DateFormat;
DateFormat df;
Date date = new Date();
// insert code here
String s = df.format(date);
```

Which code fragment, inserted at `// insert code here`, allows the code to compile?

[Hint: Click on Date and DateFormat to explore the methods.]

- [ ] df = new DateFormat();
- [ ] df = Date.getFormat();
- [ ] df = date.getFormat();
- [ ] df = DateFormat.getFormat();
- [x] df = DateFormat.getInstance();

Q8. The java.time package introduced in **Java 8** contains many new classes apart from replacements for the existing Date, DateFormat and Calendar classes.

For a complete list of new classes in **java.time** package click here.

Below are list of a few important classes:

1. Instant - represents a moment on the time-scale. Unlike the java.util.Date which measures till milliseconds, an Instant records even nanoseconds.

2. LocalDate - represents a date without the time component
3. LocalTime - represents a time without the date component
4. LocalDateTime - represents an object with both date and time components
5. ZonedDateTime - represents a complete date-time object with time-zone

See and retype the below code.

```java
package q11592;
import java.time.*;
import java.time.temporal.*;
public class LocalDateDemo {
    public static void main(String ... args) {
        LocalDate javaBirthday = LocalDate.of(1995, Month.MAY, 23);
        TemporalAdjuster adjuster = TemporalAdjusters.next(DayOfWeek.SUNDAY);
        LocalDate nextSunday = javaBirthday.with(adjuster);
        System.out.printf("Java's birthday was on %s, and the cake was cut on next Sunday : %s.%n", javaBirthday, nextSunday);
    }
}

```

Random number generator

Q1. The Random class in java.util package can be used to generate pseudo random numbers.

See and retype the below code which generates 5 random numbers below 100. Click on the class Random to know more about the available methods.

```java
package q11316;
import java.util.*;
public class RandomDemo {
    public static void main(String[] args) {
        Random rand = new Random(10);
        for (int i = 0; i < 5; i++) {
            System.out.println(rand.nextInt(100));
        }
    }
}

```