Internationalization (I18N)

Introduction

→ The process of designing web applications in such a way that which provides support for various languages & various currencies, automatically without performing any change in the application, is called "Internationalization (I18N)".

→ For ex. If the request is coming from India then the response should be in India-people understandable form.
And If the request is coming from USA then the response should be in USA-people understandable form.

→ we can implement I18N by using the following 3 classes:

1) Locale
2) NumberFormat
3) DateFormat

Locale

→ A locale object represent a geographic location (country) or

Language or both.

~> We can create a Locale object to represent "India."

~> We can create a Locale object to represent "English language."

~> Locale class present in "java.util" package.

~> It is a final class & it is a direct child class of Object class.

~> It implements serializable & cloneable interfaces.

Constructors :-

1. `Locale l = new Locale (String language);`

2. `Locale l = new Locale (String language, String country);`

ex.
`Locale l = new Locale ("pa", "IN");`

pa → Punjabi, IN → India

~> Locale class already defined some constants to represent some standard locales, we can use this constants directly.

ex. Locale.US
Locale.UK
Locale.ITALY
Locale.ENGLISH

Important Methods :-

1. `public static Locale getDefault();`
   - Returns the default Locale object set on system.

2. `public static void setDefault (Locale l);`
   - To set any other Locale object as default locale in system.

3. `public String getCountry();` // US

4. `public String getLanguage();` // en

(5) public String getDisplayCountry();

// United States

(6) public String getDisplayLanguage();

// english

(7) public static String[] getISOLanguages()

• Returns array of Iso languages

(8) public static String[] getISOCountries()

• Returns array of Iso countries.

(9) public static Locale[]
getAvailableLocales();

Ex.

```
import java.util.*;
class LocaleDemo
{
p s v main(Str)
{
    Locale l1 = Locale.getDefault();
    s.o.p(l1.getCountry() + "---" +
          l1.getLanguage());
    s.o.p(l1.getDisplayCountry + "---" +
          l1.getDisplayLanguage());
    Locale l2 = new Locale("pa",
                           "IN");
    Locale.setDefault(l2);
    s.o.p(Locale.getDefault().
          getDisplayLanguage());
    String[] s3 = Locale.
                  getISOLanguages();
    for(String s : s3)
    {
        s.o.p(s);
    }
    String[] s4 = Locale.
                  getISOCountries();
    for(String s : s4)
    {
        s.o.p(s);
    }
```

// US---en
// United States---English
// Panjabi
// aa
ab
ae
af
:
// AD
AE
AF
AG
AL
:

```
Locale[] l = Locale.getAvailableLocales();
for(Locale ltemp : l)
{
    S.O.P(ltemp.getDisplayCountry()
          +"---"+ltemp.getDisplayLanguage());
}
```

// United Arab Emirates --- Arabic
Jordan --- Arabic
:

✱ NumberFormate

→ Various locations follow various styles to represent a java number.

ex.

```
double d = 123456.789;
```

IN ⇒ 1,23,456.789
US ⇒ 123,456.789
ITALY ⇒ 123.456,789

→ we can use NumberFormat class to format a java number according to a particular Locale.

→ NumberFormat class present in [java.text] package & it is an abstract class.

→ `NumberFormat nf = new NumberFormat();` (X)

⇒ Getting NumberFormat Object For default Locale :-

→ NumberFormat class defines the following methods for this purpose :-

1. `public static NumberFormat getInstance();`
2. `public static NumberFormat getCurrencyInstance();`
3. `public static NumberFormat getPercentInstance();`
4. `public static NumberFormat getNumberInstance();`

⇒ Getting NumberFormat Object for Specific Locale :-

→ The above methods are same but we have to pass the corresponding Locale object as argument.

① public static NumberFormat
getNumberInstance
(Locale l);

four
[all same, just have one arg]
(Locale l)

⇒ once we got NumberFormat object, we can call format() & parse() method on that object

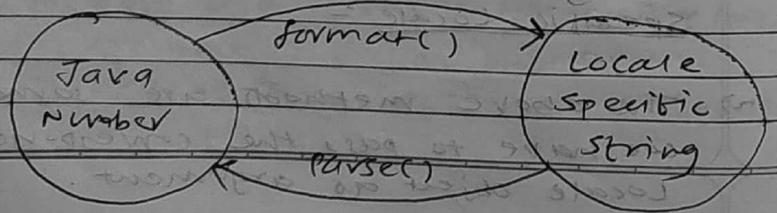
① public String format (long l)

public String format (double d)

To convert java number form to locale specific string.

② public Number parse (String s)
throws ParseException

To convert locale specific string form to java number form.



✱ WAP to display a Java number in ITALY specific form.

⇒
```
import java.util.*;
import java.text.*;

class NumberFormatDemo1
{
    p s v main(Str )
    {
        double d = 123456.789;

        NumberFormat nf =
            NumberFormat.getInstance
                (Locale.ITALY);

        S.O.P ("Italy form : " +
                nf.format(d));
    }
}
```

O/P :- Italy form :
123.456,789

✱ WAP to display a Java number in UK, US & INDIA currency forms.

⇒
```
import java.util.*;
import java.text.*;

class NumberFormatDemo2
{
```

```
p s v main(SA )
{
    double d = 123456.789;

    // For India
    Locale localeIndia = new
        Locale ("pa", "IN");
    NumberFormat nf2 = ~~xxx~~
        NumberFormat.getCurrency
        Instance(localeIndia);
    S.O.P("Indian currency form
        of number :" +
        nf2.format(d));

    // For UK
    NumberFormat nf2 =
        NumberFormat.getCurrency
        Instance(Locale.UK);
    S.O.P("UK currency form
        of number :" +
        nf2.format(d));

    // For US
    NumberFormat nf3 =
        NumberFormat.getCurrency
        Instance(Locale.US);
    S.O.P("US currency form
        of number :" +
        nf3.format(d));
}
```

O/P :-

Indian currency form of number :
INR 123,456.789

UK currency form of number :
£123,456.789

US currency form of number :
$123,456.789

Note :- O/P is coming like .79 instead of .789 bcz in currency we look only 2 digits after decimal point so .789 is converted into .79

Setting Maximum & Minimum Fraction & Integer Digits :-

→ NumberFormat class defines the following methods for this purpose.

1. public void setMaximumFractionDigits (int n)
2. public void setMinimumFractionDigits (int n)
3. public void setMaximumIntegerDigits (int n)
4. public void setMinimumIntegerDigits (int n)

123.456
Integer digits ← → Fraction digits

Ex.

```
NF nf = NF.getInstance();

case 1:

nf.setMaximumFractionDigits(2);
s.o.p(nf.format(123.4567)); //123.46
s.o.p(nf.format(123.4)); // 123.4

case 2:

nf.setMinimumFractionDigits(2);
s.o.p(nf.format(123.4567)); //123.4567
s.o.p(nf.format(123.4)); // 123.40

case3 :

nf.setMaximumIntegerDigits(3);
s.o.p(nf.format(123456.789)); // 456.789
s.o.p(nf.format(1.2345)); // 1.2345

case 4 :

nf.setMinimumIntegerDigits(3);
s.o.p(nf.format(123456.789)); // 123,456.789
s.o.p(nf.format(1.2345)); //001.2345
```

DateFormat

→ various locations follow various styles to represent date

Ex.-

IN : DD-MM-YYYY
US : MM-DD-YYYY

→ we can use DateFormat to format java date according to a particular locale.

→ DateFormat class present in java.text package & it is an abstract class.

```
DateFormat df = new DateFormat();
```
(X)

⇒ Getting DateFormat object for Default locale :-

1. public static DateFormat getInstance();
2. public static DateFormat getDateInstance();
3. public static DateFormat getDateInstance(int style);

⇒ The allowed styles are 0 to 3.

(int style)

DateFormat.FULL → 0 Wednesday 10th September 2014

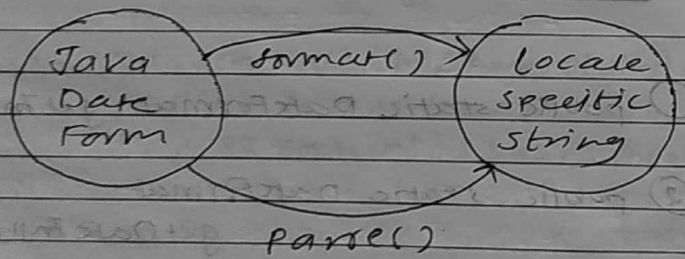
DateFormat.LONG → 1 10th September 2014

DateFormat.MEDIUM → 2 (Default) 10th sep 2014

DateFormat.SHORT → 3 10/09/14

⇒ Getting DateFormat object for specific Locale :-

① public static DateFormat getDateInstance (int style, Locale l)



⇒ once we got DateFormat object we can call the following methods on that object :-

① public String format (Date d);

- To convert java Date form to locale specific string form

② public Date parse (String s) throws ParseException

- To convert locale specific string form to java Date form.

(*) WAP to display current system date in all possible styles of US forms.

```
⇒ import java.util.*;
  import java.text.*;

  class DateFormatDemo1
  {
      p s v main (str)
      {
          S.O.p ("Full form: " +
                  DateFormat.
                  getDateInstance (0,
                  Locale.US).format
                  (new Date ());
```

```
S.O.P("Long form:"+
      DateFormat.
      getDateInstance(1,Locale.US).
      format(new Date());

S.O.P("Medium form:"+
      DateFormat.
      getDateInstance(2,Locale.US).
      format(new Date());

S.O.P("short form:"+
      DateFormate.
      getDateInstance(3,Locale.US).
      format(new Date());

  }
}
```

O/P :-

```
Full form : Sunday, July 5, 2020
Long form : July 5, 2020
Medium form : Jul 5, 2020
Short form : 7/5/20
```

(*) WAP to display current system date in UK, US, ITALY styles.

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

class DateFormatDemo2
{
  p s v main(Str~)
  {
    DateFormat Uk =
      DateFormat.
      getDateInstance(0,Locale.UK);

    DateFormat US =
      DateFormat.
      getDateInstance(0,Locale.US);

    DateFormat italy =
      DateFormat.
      getDateInstance(0,Local.ITALY);

    S.O.P("Uk style:"+
          Uk.format(new
          Date());
    S.O.P("US style:"+
          us.format(new
          Date());
    S.O.P("ITALY style:"+
          italy.format(new
          Date());
  }
}
```

O/P :-

```
Uk style : Sunday, 5 July 2020
US style : Sunday, July 5, 2020
ITALY style : domenica 5 luglio 2-2-
```

Getting DateFormat object to display both Date & Time :-

→ DateFormat class defines the following methods for this purpose :-

① public DateFormat getDateTimeInstance()

② public DateFormat getDateTimeInstance (int dateStyle, int timeStyle)

③ public DateFormat getDateTimeInstance (int dateStyle, int timStyle, Locale l)

→ The allowed time styles are also : 0 to 3 only.

(*) WAP to display Date & Time for US style.

→
```
import java.util.*;
import java.text.*;
class DateFormatDemo3
{
    p s v main(Sn)
    {
        DateFormat us =
        DateFormat.
        getDateTimeInstance(0,0,Locale.US);
        S.O.P("US style:"+
            us.format(new
            Date());
    }
}
```

O/P :-

US style : Sunday, July 5, 2020
7:55:44 PM IST

End of Internationalization (I18N)

Start of Regular Expressions