B.Sc. In Internet Systems Development.
Concurrent Programming.
Internationalization.

LIMERICK INSTITUTE
OF TECHNOLOGY
SCHOOL OF SCIENCE,
ENGINEERING & I.T.
Department of Information Technology

Introduction

- How many countries are there in the world?
- How many languages are there in the world?
- How many people are there in the world?

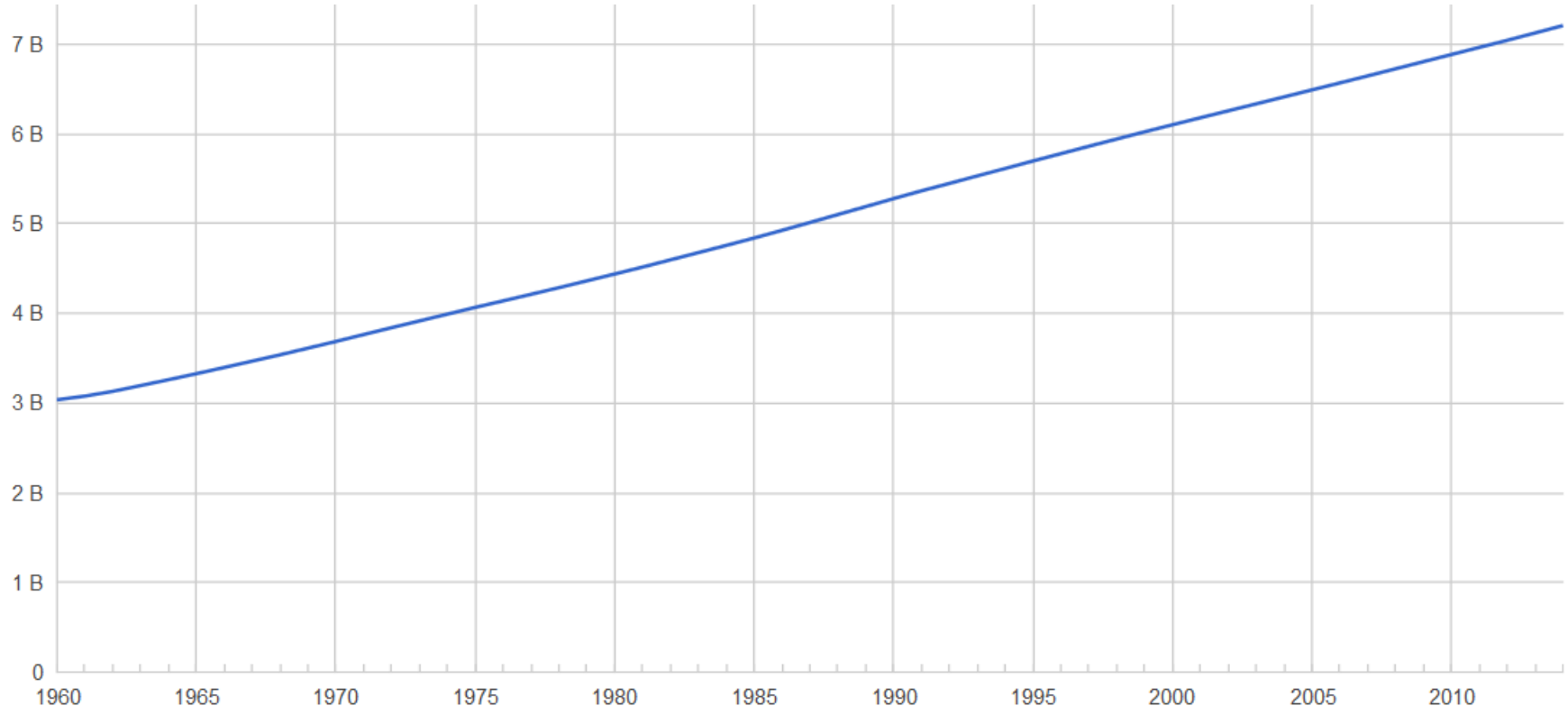
Introduction

- ***How many countries are there in the world?*** – 195 (U.S. Dept of State).

- ***How many languages are there in the world?*** – its difficult to put an exact figure on this. Its estimated to be around 6900.

- ***How many people are there in the world?*** - 7.4bn.

Introduction

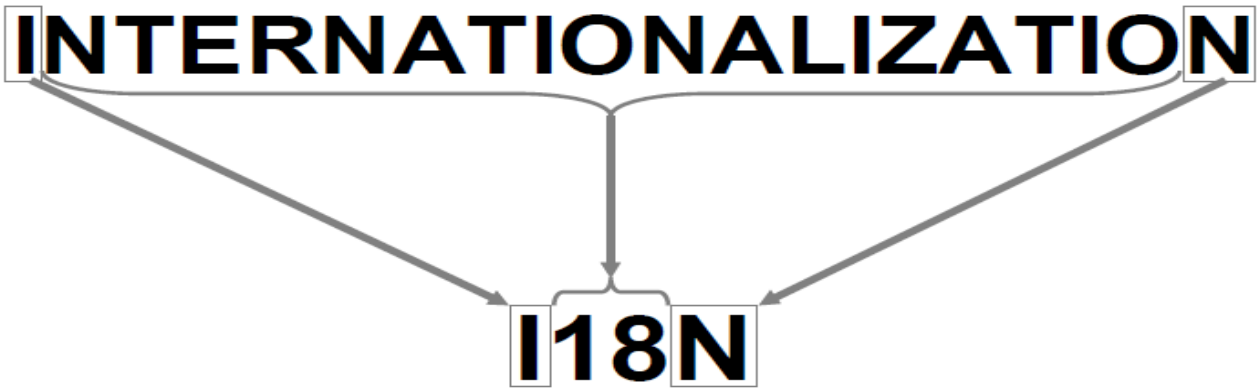


Worlds population 1960 – 2014

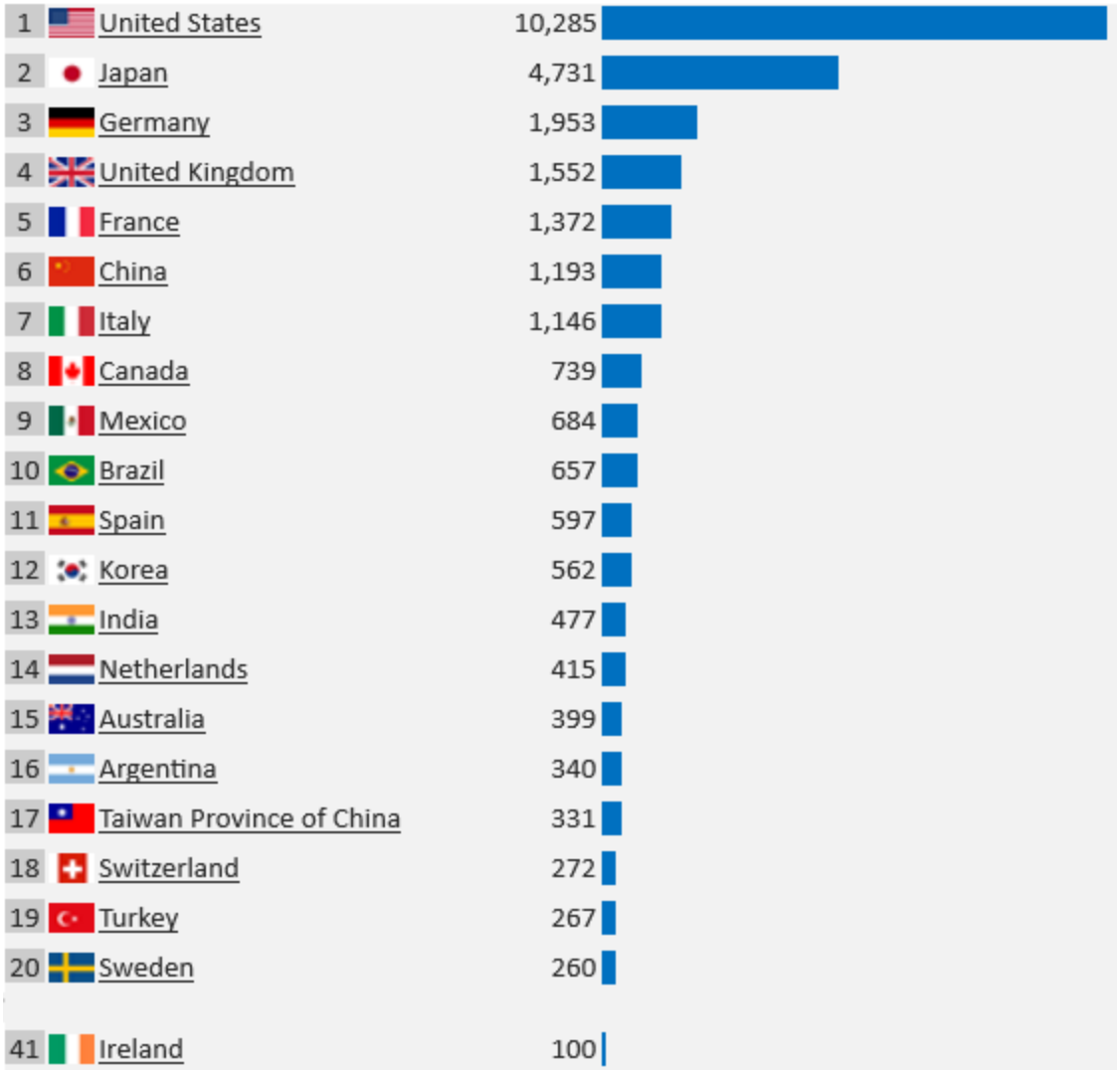
source: google.com

What is Internationalization?

- Internationalization is the process of designing a program from the ground up so that it can be changed to reflect the expectations of a new user community without having to modify its executable code.
- In other words, Internationalization is the process of designing an application so that it can be adapted to various languages, regions and cultures without engineering changes.
- Its often abbreviated to I18N.

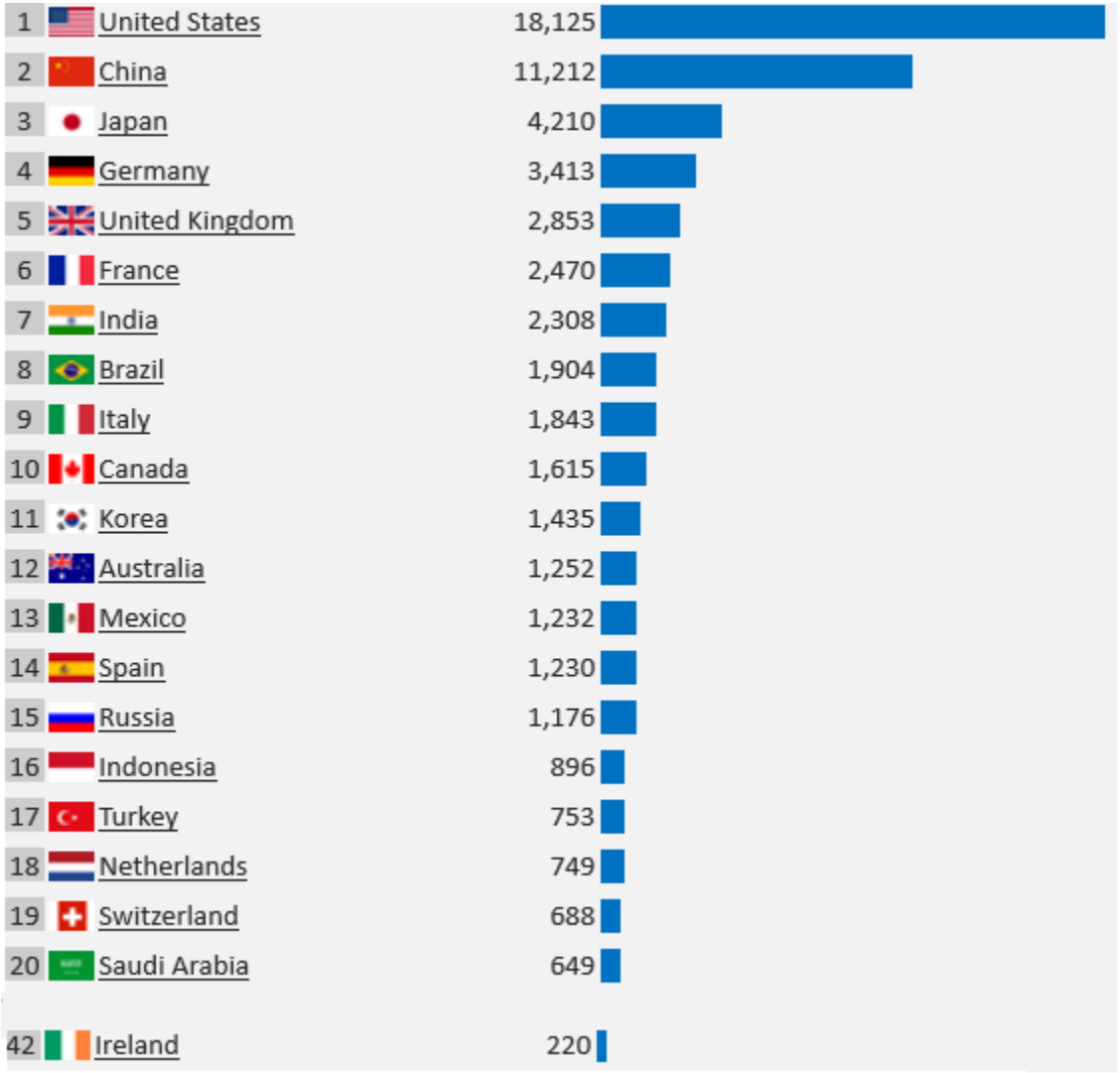


The Case For Internationalization?



World's GDP distribution in USD (billions) 2000.

The Case For Internationalization?



World's GDP distribution in USD (billions). 2015.

The Case For Internationalization?

- The USA is the wealthiest country in the world, but it still only represents about 23% of the world's economy.
- The combined GPA of major EU powerhouses like Germany, France and the UK is still less than that of China.
- China has shown huge growth since 2000 and now represents about 15% of the worlds economy.
- Japan represents about 7% of the world's economy.
- The vast majority of the Japanese and Chinese people don't speak English.

The Case For Internationalization?

Rank	Country	Population	Share of World Pop
1	China	1,393,783,836	19.24%
2	India	1,267,401,849	17.50%
3	U.S.A.	322,583,006	4.45%
4	Indonesia	252,812,245	3.49%
5	Brazil	202,033,670	2.79%
6	Pakistan	185,132,926	2.56%
7	Nigeria	178,516,904	2.46%
8	Bangladesh	158,512,570	2.19%
9	Russia	142,467,651	1.97%
10	Japan	126,999,808	1.75%

Rank	Country	Population	Share of World Pop
11	Mexico	123,799,215	1.71%
12	Philippines	100,096,496	1.38%
13	Ethiopia	96,506,031	1.33%
14	Vietnam	92,547,959	1.28%
15	Egypt	83,386,739	1.15%
16	Germany	82,652,256	1.14%
17	Iran	78,470,222	1.08%
18	Turkey	75,837,020	1.05%
19	Congo	69,360,118	0.96%
20	Thailand	67,222,972	0.93%

World population. 2015.

Internet usage and population statistics wordwide

#	Country	Internet Users (2016)	Penetration (% of Pop)	Population (2016)	Non-Users (internetless)	Users 1 Year Change (%)	Internet Users 1 Year Change	Population 1 Y Change
1	China	721,434,547	52.2 %	1,382,323,332	660,888,785	2.2 %	15,520,515	0.46 %
2	India	462,124,989	34.8 %	1,326,801,576	864,676,587	30.5 %	108,010,242	1.2 %
3	U.S.	286,942,362	88.5 %	324,118,787	37,176,425	1.1 %	3,229,955	0.73 %
4	Brazil	139,111,185	66.4 %	209,567,920	70,456,735	5.1 %	6,753,879	0.83 %
5	Japan	115,111,595	91.1 %	126,323,715	11,212,120	0.1 %	117,385	-0.2 %
6	Russia	102,258,256	71.3 %	143,439,832	41,181,576	0.3 %	330,067	-0.01 %
7	Nigeria	86,219,965	46.1 %	186,987,563	100,767,598	5 %	4,124,967	2.63 %
8	Germany	71,016,605	88 %	80,682,351	9,665,746	0.6 %	447,557	-0.01 %
9	U.K.	60,273,385	92.6 %	65,111,143	4,837,758	0.9 %	555,411	0.61 %
10	Mexico	58,016,997	45.1 %	128,632,004	70,615,007	2.1 %	1,182,988	1.27 %
82	Ireland	3,817,392	81 %	4,713,993	896,601	1 %	39,443	0.54 %

Ordered by the number of internet users in each country

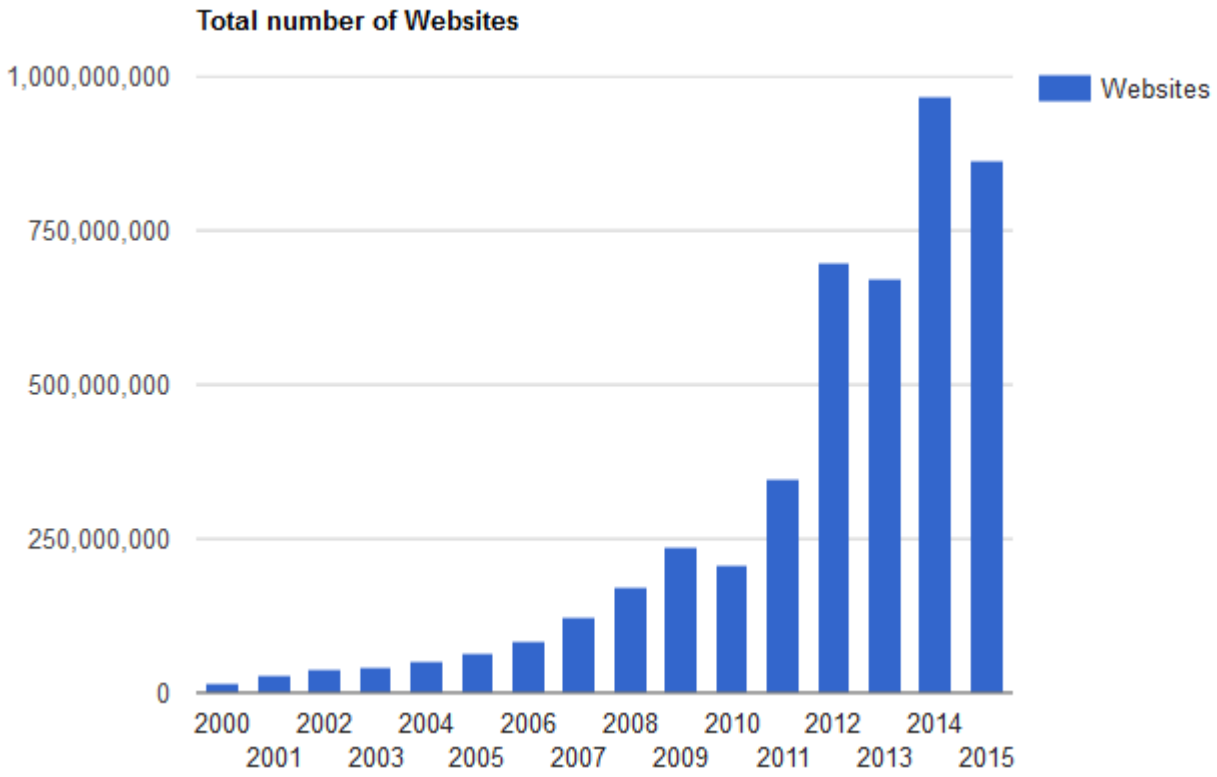
Internet usage and population statistics worldwide

#	Country	Internet Users (2016)	Penetration (% of Pop)	Population (2016)	Non-Users (internetless)	Users 1 Year Change (%)	Internet Users 1 Year Change	Population 1 Y Change
154	Iceland	331,778	100 %	331,778	0	0.9 %	2,975	0.71 %
191	Faeroe Islands	47,515	98.5 %	48,239	724	1.3 %	608	0.08 %
67	Norway	5,167,573	98 %	5,271,958	104,385	1.7 %	87,185	1.17 %
181	Bermuda	60,047	97.4 %	61,662	1,615	-0.3 %	-152	-0.55 %
177	Andorra	66,728	96.5 %	69,165	2,437	-1.6 %	-1,059	-1.86 %
64	Denmark	5,479,054	96.3 %	5,690,750	211,696	0.5 %	25,936	0.38 %
196	Liechtenstein	36,183	95.8 %	37,776	1,593	1 %	342	0.65 %
137	Luxembourg	548,807	95.2 %	576,243	27,436	1.9 %	10,314	1.61 %
36	Netherlands	15,915,076	93.7 %	16,979,729	1,064,653	0.6 %	98,813	0.32 %
45	Sweden	9,169,705	93.1 %	9,851,852	682,147	1 %	94,636	0.74 %

Ordered by the % of population with internet access

Websites Worldwide

- There are over 1bn websites.
- The first was launched in 1991.
- 75% of all websites are inactive.



Internet usage

Year (June)	Websites	Change	Internet Users	Users per Website	Websites launched
2017	**1,766,926,408**	69%			
2016	**1,045,534,808**	21%			
2015	**863,105,652**	-11%	3,185,996,155*	3.7	
2014	**968,882,453**	44%	2,925,249,355	3.0	
2013	**672,985,183**	-3%	2,756,198,420	4.1	
2012	**697,089,489**	101%	2,518,453,530	3.6	
2011	**346,004,403**	67%	2,282,955,130	6.6	
2010	**206,956,723**	-13%	2,045,865,660	9.9	Pinterest
2009	**238,027,855**	38%	1,766,206,240	7.4	
2008	**172,338,726**	41%	1,571,601,630	9.1	Dropbox
2007	**121,892,559**	43%	1,373,327,790	11.3	Tumblr
2006	**85,507,314**	32%	1,160,335,280	13.6	Twttr
2005	**64,780,617**	26%	1,027,580,990	16	YouTube, Reddit
2004	**51,611,646**	26%	910,060,180	18	Thefacebook, Flickr

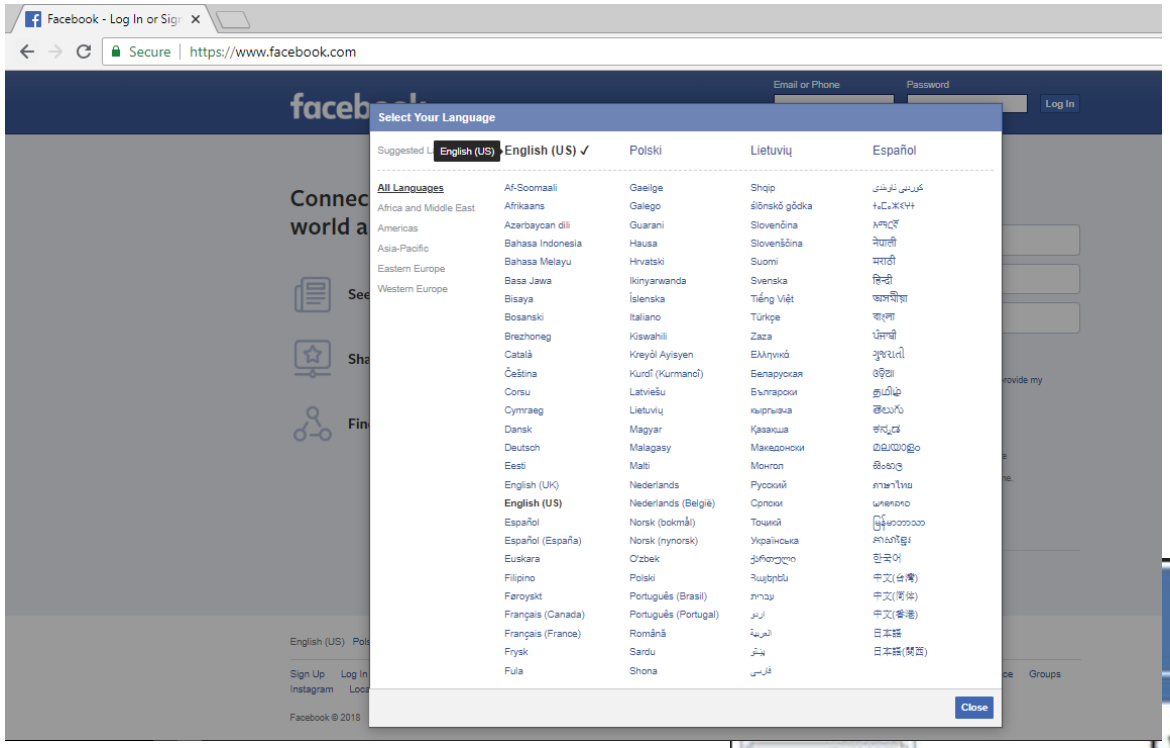
Internet usage

Contd…

2003	**40,912,332**	6%	778,555,680	19	WordPress, LinkedIn
2002	**38,760,373**	32%	662,663,600	17	
2001	**29,254,370**	71%	500,609,240	17	Wikipedia
2000	**17,087,182**	438%	413,425,190	24	Baidu
1999	**3,177,453**	32%	280,866,670	88	PayPal
1998	**2,410,067**	116%	188,023,930	78	Google
1997	**1,117,255**	334%	120,758,310	108	Yandex
1996	**257,601**	996%	77,433,860	301	
1995	**23,500**	758%	44,838,900	1,908	Altavista, Amazon, AuctionWeb
1994	**2,738**	2006%	25,454,590	9,297	Yahoo
1993	**130**	1200%	14,161,570	108,935	
1992	**10**	900%			
Aug. 1991	**1**				World Wide Web Project

Source: ***NetCraft and Internet Live Stats*** **(elaboration of data by** ***Matthew Gray of MIT and Hobbes' Internet Timeline*** **and** ***Pingdom)***

Internationalisation is Big Business



Internationalisation is Big Business

The Case For Internationalization?

- Many companies feel that Internationalization doesn't apply to them.
- What if you land a new client in the future that's based in a foreign country?
- What if one of your existing clients expands their operation to a foreign country?
- What if you wind up with suppliers or other business partners in other countries?
- What if your Web site is getting lots of hits from overseas, or what if you want your Web site to get lots of hits from overseas?
- The thing you definitely don't want to do is write your application in such a way as to inhibit its future internationalization.

Java & Internationalization

- Java is the first language designed from the ground up to support internationalization.
- It allows programs to be customized for any number of countries or languages without requiring cumbersome changes in the code.
- Three major features that support internationalization:
  1. Java characters use *Unicode* (16-bit encoding scheme)
  2. Java provides the Locale class to encapsulate information about a specific locale (date, time, numbers etc).
  3. Classes such as ResourceBundle, Preferences, Collator, MessageFormat, NumberFormat, DateTimeFormatter.

The Locale Class

- When you look at an application that is adapted to the international market, the most obvious difference is the language.

- However, differences will also appear regarding how dates, times, numbers and currency values will be displayed.

- A `Locale` object represents a specific geographical, political, or cultural region. An operation that requires a Locale to perform its task is called *locale-sensitive.*

- You can use `Locale` to tailor information to the user.

The Locale Class

To create a Locale object, you can use the following constructor(s) in Locale class:

```
Locale(String language, String country)
```

Examples:

```
new Locale("en", "US");

new Locale("fr", "CA");
```

The **language** should be a valid language code, one of the **lowercase 2-letter** codes defined by **ISO-639**.

The **country** should be a valid country code, one of the **UPPERCASE 2-letter** codes defined by **ISO-3166.**

The Locale Class

`Locale(String` **language**`, String` **country**`, String` **variant**`)`

The **variant** is rarely used. For example, the Norwegian language has two sets of spelling rules, a traditional one called ***bokmål*** and a new one called ***nynorsk***

```
new Locale("no", "NO", "B");
new Locale("no", "NO", "N");
```

The Locale Class

Locales are described by tags – hyphenated strings of locale elements such as ie-EN (in Germany, you would use a de-DE).

Switzerland has four official languages (French, German, Italian and Rhaeto Romance). A German speaker in Switzerland would use the locale de-CH.

The de-CH locale would use the rules for the German language, but currency values would be expressed as Swiss Francs and not in Euro.

Locale objects can be constructed from these tags:

```
Locale usEnglish =

      Locale.forLanguageTag("en-US");
```

The Locale Class

For convenience, constants exist for commonly used *countries*.

- Locale.CANADA
- Locale.FRANCE
- Locale.ITALY
- Locale.UK
- Locale.US etc…

There are also constants for commonly used *languages*.

- Locale.CHINESE
- Locale.FRENCH
- Locale.KOREAN
- Locale.English etc…

Working With Locales

An operation that requires a Locale to perform its task is called ***locale-sensitive***.

For example. displaying a number as a date or time is a locale-sensitive operation.

Several classes in the Java class libraries contain locale-sensitive methods

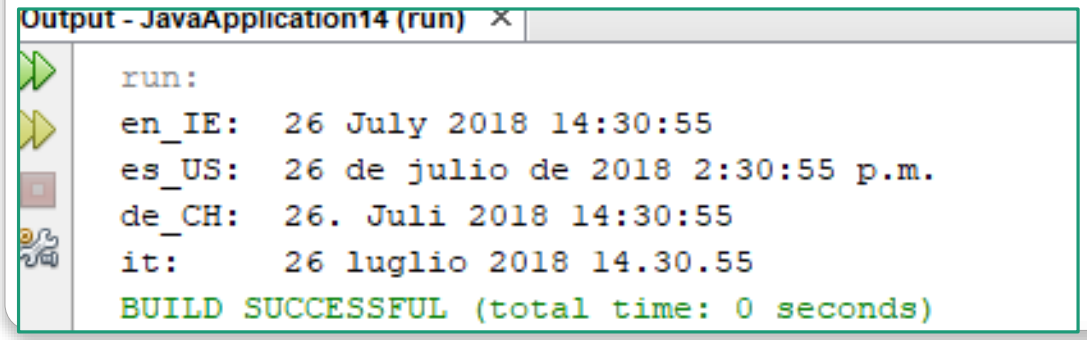
All locale-sensitive classes contain a static method `getAvailableLocales()`, which returns an array of all locales known to the virtual machine.


```
Locale[] availableLocales =
                    Calender.getAvailableLocales();
```

Working With Locales

```
GregorianCalendar cal = new GregorianCalendar();

Locale locales[] = new Locale[4];

//English-Ireland
locales[0] = new Locale("en", "IE");

//Spanish-USA
locales[1] = new Locale("es","US");

//German-Switzerland
locales[2] = Locale.forLanguageTag("de-CH");

locales[3] = Locale.ITALIAN;

for (Locale locale : locales) {
    DateFormat formatter = DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.MEDIUM, locale);
    System.out.println(locale + ":\t" + formatter.format(cal.getTime()));
}
```

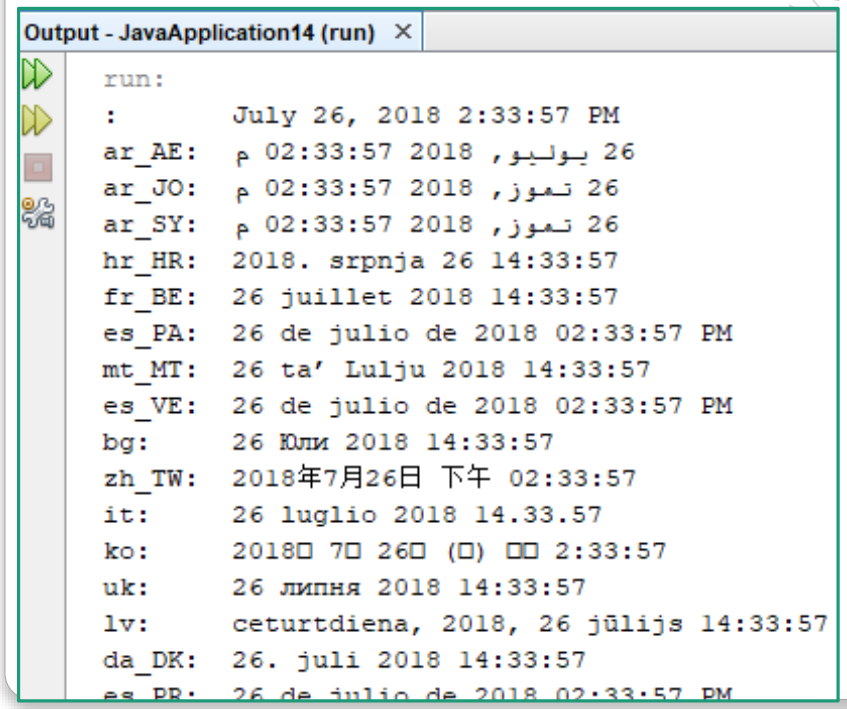
src.working_with_locales.LocaleEX1.java



Working With Locales

```
GregorianCalendar cal = new GregorianCalendar();

Locale locales[] = GregorianCalendar.getAvailableLocales();

for (Locale locale : locales) {
    DateFormat formatter = DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.MEDIUM, locale);
    System.out.println(locale + ":\t" + formatter.format(cal.getTime()));
}
```

src.working_with_locales.LocaleEX2.java



Working With Locales

```
GregorianCalendar cal = new GregorianCalendar();

Locale locales[] = GregorianCalendar.getAvailableLocales();

for (Locale locale : locales) {
    DateFormat formatter = DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.MEDIUM, locale);
    System.out.println(locale.getDisplayName(Locale.GERMAN) + ":\t" + formatter.format(cal.getTime()));
}
```

```
Arabisch (Syrien):      م 02:44:11 2018 ,تموز 26
Kroatisch (Kroatien):   2018. srpnja 26 14:44:11
Französisch (Belgien):  26 juillet 2018 14:44:11
Spanisch (Panama):      26 de julio de 2018 02:44:11 PM
Maltesisch (Malta):     26 ta' Lulju 2018 14:44:11
Spanisch (Venezuela):   26 de julio de 2018 02:44:11 PM
Bulgarisch:     26 Юли 2018 14:44:11
Chinesisch (Taiwan):    2018年7月26日 下午 02:44:11
Italienisch:    26 luglio 2018 14.44.11
Koreanisch:     2018□ 7□ 26□ (□) □□ 2:44:11
Ukrainisch:     26 липня 2018 14:44:11
Lettisch:       ceturtdiena, 2018, 26 jūlijs 14:44:11
Dänisch (Dänemark):     26. juli 2018 14:44:11
Spanisch (Puerto Rico): 26 de julio de 2018 02:44:11 PM
Vietnamesisch (Vietnam):        14:44:11 Ngày 26 tháng 7
```

Formatting Numbers

Formatting numbers as currency or percentages is highly locale dependent.

For example, number 5000.50 is displayed as $5,000.50 in the US currency, but the same number is displayed as 5 000,50€ in Euro in France

The NumberFormat class allows you to format numbers. Useful methods of this class include:

getNumberInstance() *gets a number format.*

getCurrencyInstance() *gets the currency number format.*

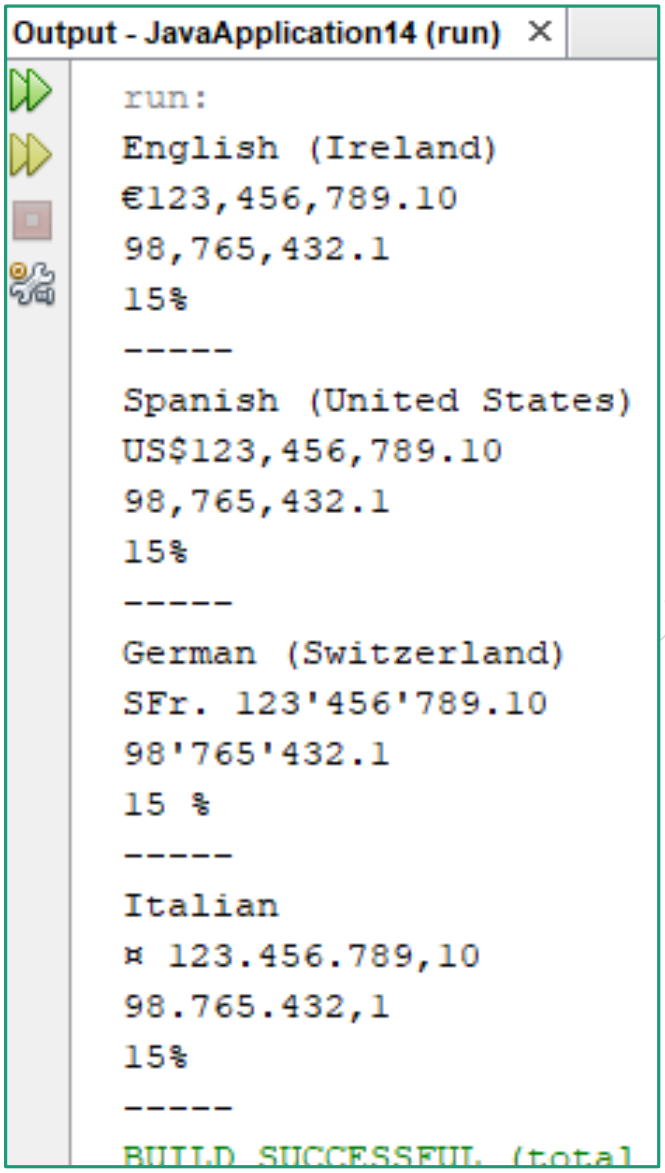
getPercentInstance() *gets a format for displaying percentages. With this format, a fraction like 0.53 is displayed as 53%.*

Formatting Numbers

```
double aMoneyValue = 123456789.10;
double aNumberValue = 98765432.10;
double aPercentageValue = .15;

Locale locales[] = new Locale[4];

//English-Ireland
locales[0] = new Locale("en", "IE");

//Spanish-USA
locales[1] = new Locale("es","US");

//German-Switzerland
locales[2] = Locale.forLanguageTag("de-CH");

locales[3] = Locale.ITALIAN;

for (Locale locale : locales) {
    NumberFormat currencyFrmt = NumberFormat.getCurrencyInstance(locale);
    NumberFormat numberFrmt = NumberFormat.getNumberInstance(locale);
    NumberFormat percentFrmt = NumberFormat.getPercentInstance(locale);
    System.out.println(locale.getDisplayName());
    System.out.println(currencyFrmt.format(aMoneyValue));
    System.out.println(numberFrmt.format(aNumberValue));
    System.out.println(percentFrmt.format(aPercentageValue));
    System.out.println("-----");
}
```

src.formatting_numbers.FormattingNumbersEX1.java

Formatting Numbers



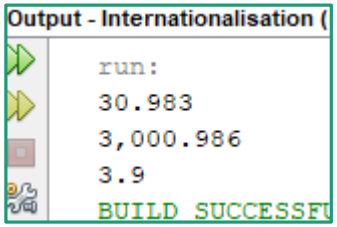
Formatting Numbers

For even more control over the format or parsing of decimal numbers, a NumberFormat object can be cast into a DecimalFormat object.

DecimalFormat is a subclass of NumberFormat.

The applyPattern() method of DecimalFormat can be used to specify the pattern for displaying the number.

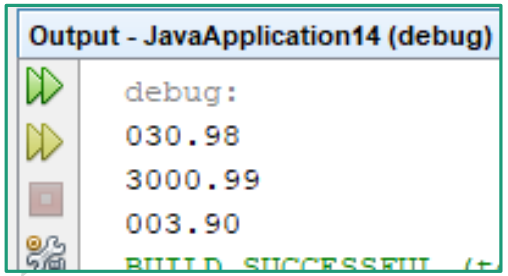
```
//English-Ireland
Locale l = new Locale("en", "IE");

NumberFormat frmt = NumberFormat.getNumberInstance(l);
DecimalFormat df = (DecimalFormat) frmt;
System.out.println(df.format(30.983));
System.out.println(df.format(3000.9856));
System.out.println(df.format(3.9));
```



src.formatting_numbers.FormattingNumbersEX2.java

Using a DecimalFormat for a specific Locale

```
Locale l = Locale.UK;
NumberFormat numberForm = NumberFormat.getNumberInstance(l);
DecimalFormat df = (DecimalFormat) numberForm;
df.applyPattern("000.00");

System.out.println(df.format(30.983));
System.out.println(df.format(3000.9856));
System.out.println(df.format(3.9));
```




```
Locale l = Locale.GERMAN;
NumberFormat numberForm = NumberFormat.getNumberInstance(l);
DecimalFormat df = (DecimalFormat) numberForm;
df.applyPattern("000.00");

System.out.println(df.format(30.983));
System.out.println(df.format(3000.9856));
System.out.println(df.format(3.9));
```

src.formatting_numbers.FormattingNumbersEX3.java

Currencies

As demonstrated, NumberFormat.getCurrencyInstance() can be used to format currency values.

An alternative is to se the Currency class. Usage Examples include:


```
Currency c = Currency.getInstance("EUR"); //Common Codes Include "USD", "GBP", "JPY", "RUB"
System.out.println(c.getDisplayName() + "\t" + c.getSymbol() + "\t" + c.getCurrencyCode());

c = Currency.getInstance(Locale.JAPAN); //Pass a locale to the getInstance method
System.out.println(c.getDisplayName() + "\t" + c.getSymbol() + "\t" + c.getCurrencyCode());
```

```
Output - JavaApplication14 (run)
run:
Euro    €       EUR
Japanese Yen    JPY     JPY
```

Currencies

```
11          NumberFormat ukFormat = NumberFormat.getCurrencyInstance(Locale.UK);
12          ukFormat.setCurrency(Currency.getInstance("GBP"));
13          System.out.println(ukFormat.format(123456789.10));
14
15          ukFormat.setCurrency(Currency.getInstance("USD"));
16          System.out.println(ukFormat.format(123456789.10));
17
18          ukFormat.setCurrency(Currency.getInstance("EUR"));
19          System.out.println(ukFormat.format(123456789.10));
20
21          System.out.println("");
22
23          NumberFormat itaFormat = NumberFormat.getCurrencyInstance(Locale.ITALY);
24          itaFormat.setCurrency(Currency.getInstance("GBP"));
25          System.out.println(itaFormat.format(123456789.10));
26
27          itaFormat.setCurrency(Currency.getInstance("USD"));
28          System.out.println(itaFormat.format(123456789.10));
29
30          itaFormat.setCurrency(Currency.getInstance("EUR"));
31          System.out.println(itaFormat.format(123456789.10));
```

```
Output - Internationalization (run)
run:
£123,456,789.10
USD123,456,789.10
€123,456,789.10

GBP 123.456.789,10
USD 123.456.789,10
€ 123.456.789,10
BUILD SUCCESSFUL (tot
```

src.currencies.WorkingWithCurrenciesEX1.java

Currencies

```
Set<Currency> currencies = Currency.getAvailableCurrencies();

for (Currency curr : currencies) {
    System.out.println(curr.getDisplayName() + "\t" + curr.getSymbol() + "\t" + curr.getCurrencyCode());
}
```

```
Output - JavaApplication14 (run)
run:
Eritrean Nakfa  ERN     ERN
Colombian Peso  COP     COP
Cuban Peso      CUP     CUP
Tajikistani Somoni      TJS     TJS
Zimbabwean Dollar (2009)        ZWL     ZWL
Ghanaian Cedi   GHS     GHS
```



```
Locale.setDefault(Locale.GERMAN);

Set<Currency> currencies = Currency.getAvailableCurrencies();

for (Currency curr : currencies) {
    System.out.println(curr.getDisplayName() + "\t" + curr.getSymbol() + "\t" + curr.getCurrencyCode());
}
```

```
Output - JavaApplication14 (run)
run:
Nakfa   ERN     ERN
Kolumbianischer Peso    COP     COP
Kubanischer Peso        CUP     CUP
Tadschikistan Somoni    TJS     TJS
Simbabwe-Dollar (2009)  ZWL     ZWL
Ghanaische Cedi GHS     GHS
```

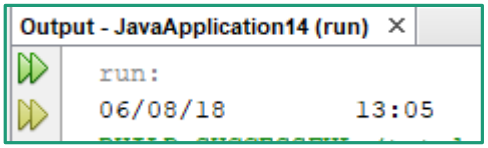
Date and Time Formatting

When formatting dates/times, consider the following:

1. The names of months/days etc should appear in the local language.
2. Observe the local preference for ordering the day/month/year when displaying a date.
3. The Gregorian calendar might not be the local preference for representing dates.
4. The local time zone must be observed.

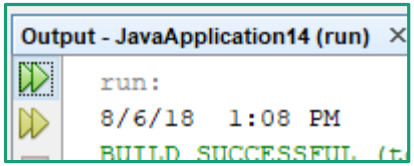
Date and Time Formatting

```
//create the format
FormatStyle style = FormatStyle.SHORT;
DateTimeFormatter dateFormatter = DateTimeFormatter.ofLocalizedDate(style);
DateTimeFormatter timeFormatter = DateTimeFormatter.ofLocalizedTime(style);

//get the date
LocalDate date = LocalDate.now();

//get the time - could use LocalTime instead of ZonedDateTime
ZonedDateTime time = ZonedDateTime.now();

//display the formatted date and time
System.out.println(dateFormatter.format(date) + "\t" + timeFormatter.format(time));
```

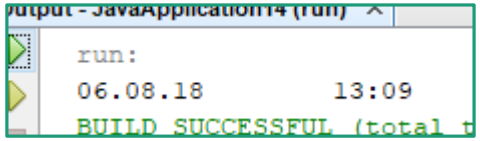


src.date_and_timeDateAndTimeEX1.java

Run the code again, but beforehand set the locale to the US - *Locale.setDefault(Locale.US);*



Set the locale to Germany - *Locale.setDefault(Locale.GERMANY);*



Date and Time Formatting

Locale Specific Formatting Styles:

Style	Date	Time
SHORT	06/08/18	13:12
MEDIUM	06-Aug-2018	13:12:10
LONG	06 August 2018	13:12:35 IST
FULL	06 August 2018	13:12:58 o'clock IST

Date and Time Formatting

When accessing weekdays and months of the year use the DayOfWeek and Month enumerations.

```
for (Month m: Month.values()) {
    //TextStyle.FULL -> January,  TextStyle.SHORT -> Jan,  TextStyle.NARROW -> J
    System.out.print(m.getDisplayName(TextStyle.FULL, Locale.UK) + ", ");
}

System.out.println("");

for (DayOfWeek d: DayOfWeek.values()) {
    //TextStyle.FULL -> Monday,  TextStyle.SHORT -> Mon,  TextStyle.NARROW -> M
    System.out.print(d.getDisplayName(TextStyle.NARROW, Locale.UK)+ ", ");
}
```

src.date_and_timeDateAndTimeEX2.java


```
Output - JavaApplication14 (run)
run:
January, February, March, April, May, June, July, August, September, October, November, December,
Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday,
BUILD SUCCESSFUL (total time: 0 seconds)
```

Output for German Locale.

```
Output - JavaApplication14 (run)
run:
Januar, Februar, März, April, Mai, Juni, Juli, August, September, Oktober, November, Dezember,
Montag, Dienstag, Mittwoch, Donnerstag, Freitag, Samstag, Sonntag,
BUILD SUCCESSFUL (total time: 0 seconds)
```

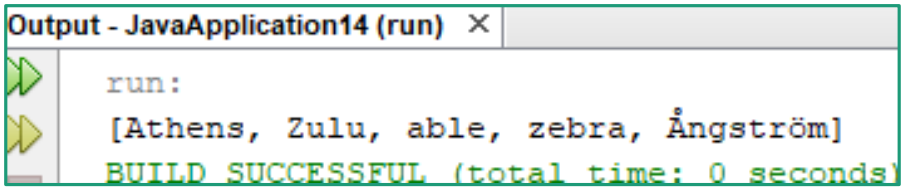
Collation

If you were to sort an array containing the following strings

"Zulu", "Athens", "Ångström", "able", "zebra“

what would the output be?

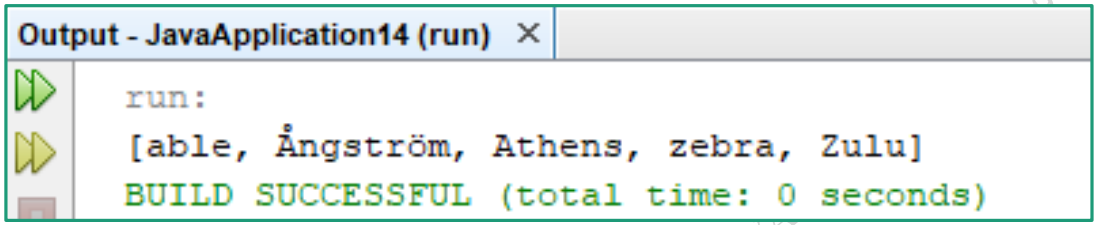

```
String[] words = {"Zulu", "Athens", "Ångström", "able", "zebra"};

Arrays.sort(words);

System.out.println(Arrays.toString(words));
```



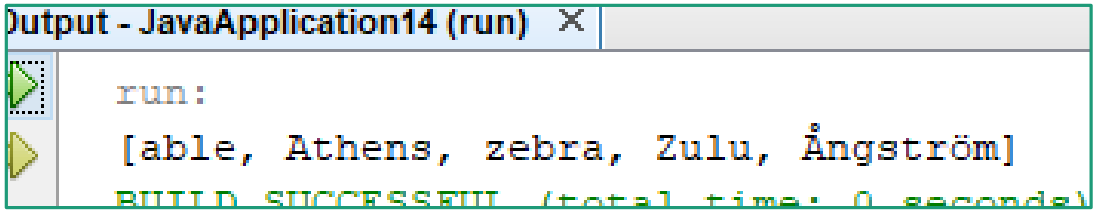
Collation

For dictionary ordering you want to consider upper and lower case letters as equivalent – accents should not be significant.

For an English speaker you would desire the following ordering.



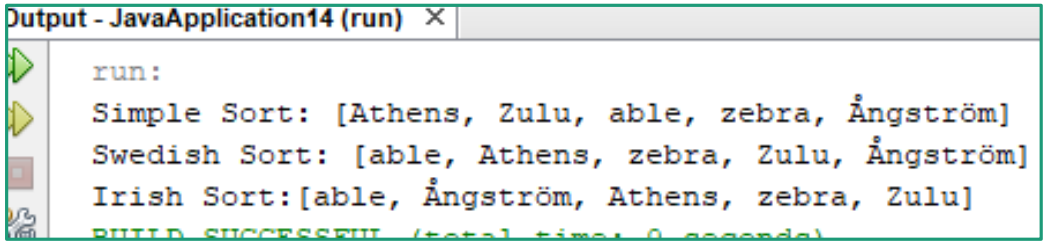
For a Swedish speaker you would want the following ordering.



Collation

```
String[] words = {"Zulu", "Athens", "Ångström", "able", "zebra"};

Arrays.sort(words);

System.out.println("Simple Sort: " + Arrays.toString(words));

Collator coll = Collator.getInstance(new Locale("sv", "SE"));

Arrays.sort(words, coll);

System.out.println("Swedish Sort: " + Arrays.toString(words));

coll = Collator.getInstance(new Locale("en", "IE"));

Arrays.sort(words, coll);

System.out.println("Irish Sort:" + Arrays.toString(words));
```

src.collation.CollationEX1.java



Collation

You can set a collator to adjust how selective it is,

Character differences are set as primary, secondary or tertiary.

In English, the difference between *a* and *b* is considered primary, the difference between e and é is secondary and the difference between e and E is tertiary.

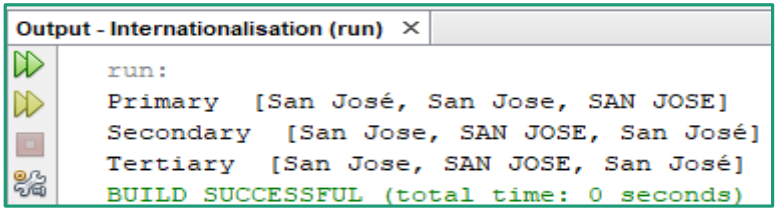
For example, when processing these city names - *San José, San Jose, SAN JOSE*, you may not care about the differences between them.

In this case, you could set the collator's strength to primary.

Collation

```
String[] words = {"San José", "San Jose", "SAN JOSE"};

Collator coll = Collator.getInstance(Locale.UK);

coll.setStrength(Collator.PRIMARY);

Arrays.sort(words, coll);

System.out.println("Primary   " + Arrays.toString(words));

coll.setStrength(Collator.SECONDARY);

Arrays.sort(words, coll);

System.out.println("Secondary   " + Arrays.toString(words));

coll.setStrength(Collator.TERTIARY);

Arrays.sort(words, coll);

System.out.println("Tertiary   " + Arrays.toString(words));
```

src.collation.CollationEX2.java



Resource Bundles

- A *ResourceBundle* is a Java class file or a text file that provides locale-specific information. This information can be accessed by Java programs dynamically.

- When your program needs a locale-specific resource, a message string for example, your program can load the string from the resource bundle that is appropriate for the desired locale. In this way, you can write program code that is largely independent of the user's locale isolating most, if not all, of the locale-specific information in resource bundles.

Resource Bundles

- Resource bundles allow programs to separate the locale-sensitive part of the code from the locale independent part.
- Programs can handle multiple locales and can easily be modified later to support more locales.
- Resource bundles contain *key/value* pairs where each key uniquely identifies a locale-specific object in the bundle.
- A `MissingResourceBundleException` can be raised if a resource bundle or resource object is not found.

Resource Bundles

- Keys are case sensitive.
- If all the keys are strings they can be placed in a text file with extension .properties

```
#MyResourceBundle_de.properties for Germanlanguage
Choose_Locale = Wählen Sie Ihren Schauplatz vor
Enter_Name=Name
Enter_Address=Adresse
Enter_Phone = Telefonnummer
Enter_Job = Besetzung
Submit_Button = Reichen Sie ein
Clear_Button = Freier Raum
Frame_Title = Tragen Sie bitte Ihre persönlichen Details ein
```

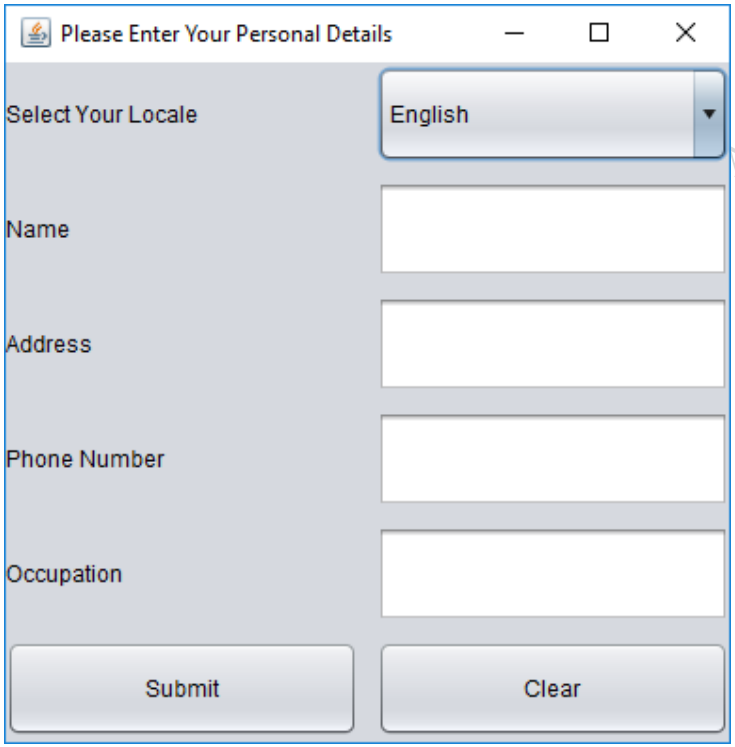
Resource Bundles

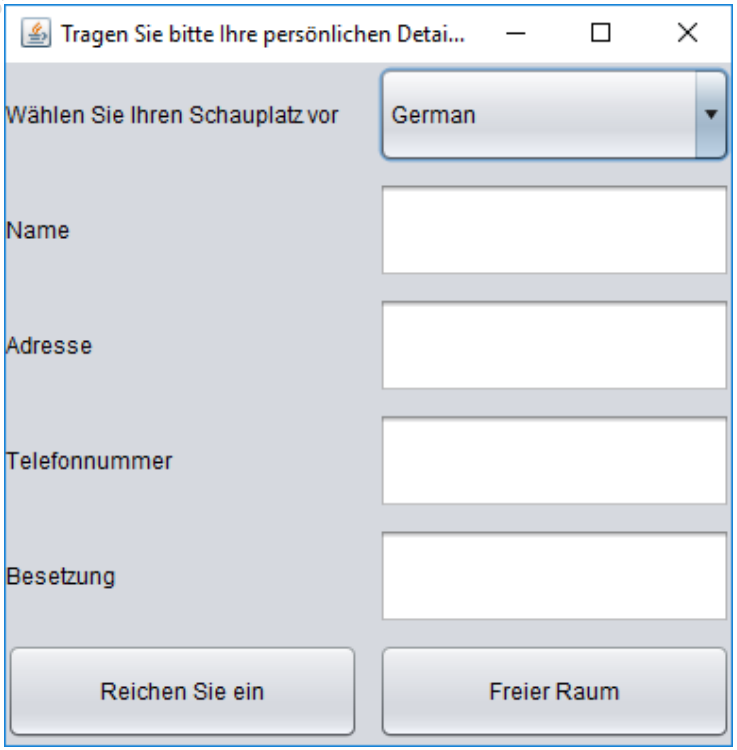
Once a resource bundle object is created, you can use the getObject() method or the getString() to retrieve the value according to the key.

```
Locale l = new Locale(“fr”);


ResourceBundle res =
   ResourceBundle.getBundle("MyResource“, l);


String locale = res.getString("Choose_Locale");
String name =  res.getString("Enter_Name");
```

Example: Using Resource Bundles

- An example appears on Moodle (src.resourcebundle_example.Main.java) which also demonstrates using Resource Bundles.





Further Reading

Date and Time API.

Preferences class.

Character Encodings.

References

Y. Daniel Liang (2017) *Intro to Java Programming and Data Structures, Comprehensive Version.* 11/E. Pearson. ISBN-13 978-0134670942 (Link)

Paul J Deitel (2016) *Java How To Program.* 10/E. ISBN-13 9780134800271 (Link)

Cay S. Horstmann (2018) Core Java SE 9 For the Impatient. 2/E. ISBN-13 978-0-13-469472-6 (Link)

http://www.internetlivestats.com