I18N notes.

General.

Internationalisation.

An application with support for Internationalisation

- a.k.a. I18N
- can be adapted to other languages / regions
- process is quick and easily
- doesn't require engineering / code changes to add support for another language / region (dependencies are stored externally)

Localisation.

- a.k.a. L10N
- addition of language dependent components
- translation of text, etc.

Considerations for I18N applications.

Identification of culturally dependent data.

- common text output (text, dates, times, currency, numbers)
- other text output (measurements, phone numbers, postcodes, titles)
- GUI items (labels, buttons, menus, etc.)
- media (graphics, sounds, icons)

Translation.

Translatable text should be isolated/externalised from the app into ResourceBundles
Compound messages (i.e. those containing several culturally dependent items that may be rendered in a culturally dependent order) must be externalised also.

Java support.

Java provides support for locale specific rendering of numbers, currency, dates, times – use these for rendering culturally dependent data in a local specific manner.

Comparison.

String and characters must be compared using locale-aware functions – e.g. Character.isLetter ('A'), Collator.compare (s1, s2)

Unicode.

Java uses Unicode to represent characters / strings.
If characters / strings are imported into Java they must be converted to Unicode.
If characters / strings are exported from Java they must be written in the required external representation.

Examples :

- `String s = new String (utfBytes, "UTF8")`
- `byte [] bytes = s.getBytes ("UTF8")`
- `InputStreamReader isr = new InputStreamReader (fis, "UTF8");`
- `Writer out = new OutputStreamWriter (fos, "UTF8");`

Java I18N classes.

java.util.Locale

- combination of language and country – e.g. `locale = new Locale("en", "GB");`
- locale-aware classes can be locale instance based but otherwise default to the JVM locale
- can also construct with a *variant* – e.g. `locale = new Locale("en", "GB", "UNIX");`

java.util.ResourceBundle

- acts as a container for locale specific properties
- ResourceBundle.getBundle (NAME, LOCALE) will scan for a class or property file matching NAME_LANGUAGE-CODE_COUNTRY-CODE (e.g. Test_en_GB.class or Test_en_GB.properties)
- ResourceBundle accessors – getString (NAME), getObject (NAME)
- two subclasses available – PropertyResourceBundle and ListResourceBundle
- PropertyResourceBundle (dependencies defined as Strings in a property file)
- ListResourceBundle (dependencies defined as Objects in a subclass of ListResourceBundle)

java.text.NumberFormat

- Provides support for parsing/formatting numbers, currency and percentages in a locale-specific manner using *pre-defined* patterns
- NumberFormat.getNumberInstance (LOCALE).format (NUM)
- NumberFormat.getCurrencyInstance (LOCALE).format (NUM)
- NumberFormat.getPercentageInstance (LOCALE).format (NUM)

java.text.DecimalFormat

- Provides support for *custom* parsing/formatting of numbers using format patterns
- '#' is used to specify digits, ',' for grouping and '.' for decimal points
- '0' is used to specify digits with leading zeros
- "123456.789" with pattern of "0000,###.## " results in "0123,456.79"
- output symbols can be changed – e.g. '.' can be rendered as any requested character

java.text.DateFormat

- Provides support for parsing/formatting dates and times in a locale-specific manner using *pre-defined* patterns. Len of output can be controlled – e.g. DEFAULT, SHORT, MEDIUM, LONG, FULL
- DateFormat.getDateInstance (DateFormat.DEFAULT, LOCALE).format (DATE)
- DateFormat.getTimeInstance (DateFormat.DEFAULT, LOCALE).format (DATE)
- df.getDateTimeInstance (DateFormat.DEFAULT, DateFormat.DEFAULT, LOCALE).format (DATE)

java.text.SimpleDateFormat

- Provides support for *custom* parsing/formatting of dates/times using format patterns
- E.g. pattern "dd/MM/yy HH:mm:ss" results in "06/03/02 02:06:30"
- for correct rendering of dates and times, use locale + pattern (pattern on it's own could leads to inconsistent formatting in other languages)
- date symbols can be changed (e.g. "Mon" can be changed to "MON")

java.text.MessageFormat

- provides support for template based rendering in a locale-specific manner using a pattern string and an array of arguments – similar to placeholders in SQL PreparedStatement

java.text.BreakIterator

- provides support for identifying breaks (by character, word, sentence or line) in text in a locale-specific manner
- getCharacterInstance (), getWordInstance (), getSentenceInstance (), getLineInstance ()
- BreakIterator.first (), BreakIterator.next (), while (BreakIterator.next () != BreakIterator.DONE)