Table of Contents

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

Chapter 15. java.text

Package java.text

Java 1.1

The `java.text` package consists of classes and interfaces that are useful for writing internationalized programs that handle local customs, such as date and time formatting and string alphabetization, correctly.

Copyright Safari Books Online #628024

The `NumberFormat` class formats numbers, monetary quantities, and percentages as appropriate for the default or specified locale. `DateFormat` formats dates and times in a locale-specific way. The concrete `DecimalFormat` and `SimpleDateFormat` subclasses of these classes can be used for customized number, date, and time formatting. `MessageFormat` allows substitution of dynamic values, including formatted numbers and dates, into static message strings. `ChoiceFormat` formats a number using an enumerated set of string values. See the `Format` superclass for a general description of formatting and parsing strings with these classes. `Collator` compares strings according to the customary sorting order for a locale. `BreakIterator` scans text to find word, line, and sentence boundaries following locale-specific rules. The `Bidi` class of Java 1.4 implements the Unicode "bidirectional" algorithm for working with languages such as Arabic and Hebrew that display text right-to-left but display numbers left-to-right.

Interfaces

```
public interface AttributedCharacterIterator extends CharacterIterator;
public interface CharacterIterator extends Cloneable;
```

Classes

```
public class Annotation;
public static class AttributedCharacterIterator.Attribute implements Serializable;
   public static class Format.Field extends AttributedCharacterIterator.Attribute;
      public static class DateFormat.Field extends Format.Field;
      public static class MessageFormat.Field extends Format.Field;
      public static class NumberFormat.Field extends Format.Field;
public class AttributedString;
public final class Bidi;
public abstract class BreakIterator implements Cloneable;
public final class CollationElementIterator;
public final class CollationKey implements Comparable<CollationKey>;
public abstract class Collator implements java.util.Comparator<Object>, Cloneable;
   public class RuleBasedCollator extends Collator;
public class DateFormatSymbols implements Cloneable, Serializable;
public final class DecimalFormatSymbols implements Cloneable, Serializable;
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

```
public class FieldPosition;
public abstract class Format implements Cloneable, Serializable;
   public abstract class DateFormat extends Format;
      public class SimpleDateFormat extends DateFormat;
   public class MessageFormat extends Format;
   public abstract class NumberFormat extends Format;
      public class ChoiceFormat extends NumberFormat;
      public class DecimalFormat extends NumberFormat;
public class ParsePosition;
public final class StringCharacterIterator implements CharacterIterator;
```

Exceptions

```
public class ParseException extends Exception;
```

Annotation java.text

Java 1.2

This class is a wrapper for a the value of a text attribute that represents an annotation. Annotations differ from other types of text attributes in two ways. First, annotations are linked to the text they are applied to, so changing the text invalidates or corrupts the meaning of the annotation. Second, annotations cannot be merged with adjacent annotations, even if they have the same value. Putting an annotation value in an `Annotation` wrapper serves to indicate these special characteristics. Note that two of the attribute keys defined by `AttributedCharaterIterator.Attribute`, `READING` and `INPUT_METHOD_SEGMENT`, must be used with `Annotation` objects.

```
public class Annotation {
// Public Constructors
     public Annotation(Object value);
// Public Instance Methods
     public Object getValue( );
// Public Methods Overriding Object
     public String toString( );
}
```

AttributedCharacterIterator java.text

Java 1.2 ***cloneable***

This interface extends `CharacterIterator` for working with text that is marked up with attributes in some way. It defines an inner class, `AttributedCharaterIterator.Attribute`, that represents attribute keys. `AttributedCharacterIterator` defines methods for querying the attribute keys, values, and runs for the text being iterated over. `getAllAttributeKeys( )` returns the

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

`Set` of all attribute keys that appear anywhere in the text. `getAttributes( )` returns a `Map` that contains the attribute keys and values that apply to the current character. `getAttribute( )` returns the value associated with the specified attribute key for the current character.

`getRunStart( )` and `getRunLimit( )` return the index of the first and last characters in a run. A *run* is a string of adjacent characters for which an attribute has the same value or is undefined (i.e., has a value of `null`). A run can also be defined for a set of attributes, in which case it is a set of adjacent characters for which all attributes in the set hold a constant value (which may include `null`). Programs that process or display attributed text must usually work with it one run at a time. The no-argument versions of `getRunStart( )` and `getRunLimit( )` return the start and end of the run that includes the current character and all attributes that are applied to the current character. The other versions of these methods return the start and end of the run of the specified attribute or set of attributes that includes the current character.

The `AttributedString` class provides a simple way to define short strings of attributed text and obtain an `AttributedCharacterIterator` over them. Most applications that process attributed text are working with attributed text from specialized data sources, stored in some specialized data format, so they need to define a custom implementation of `AttributedCharacterIterator`.

Figure 15-1. java.text.AttributedCharacterIterator


```
public interface AttributedCharacterIterator extends CharacterIterator {
// Nested Types
     public static class Attribute implements Serializable;
// Public Instance Methods
     java.util.Set<AttributedCharacterIterator.Attribute>
        getAllAttributeKeys( );
     Object getAttribute(AttributedCharacterIterator.Attribute attribute);
     java.util.Map<AttributedCharacterIterator.Attribute,Object>
        getAttributes( );
     int getRunLimit( );
     int getRunLimit(java.util.Set<? extends AttributedCharacterIterator.
        Attribute> attributes);
     int getRunLimit(AttributedCharacterIterator.Attribute attribute);
     int getRunStart( );
     int getRunStart(AttributedCharacterIterator.Attribute attribute);
     int getRunStart(java.util.Set<? extends AttributedCharacterIterator.
        Attribute> attributes);
}
```

Passed To

`AttributedString.AttributedString( ),Bidi.Bidi( )`

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

Returned By

`AttributedString.getIterator( )`, `DecimalFormat.formatToCharacterIterator( )`, `Format.formatToCharacterIterator( )`, `MessageFormat.formatToCharacterIterator( )`, `SimpleDateFormat.formatToCharacterIterator( )`

AttributedCharacterIterator.Attribute java.text

Java 1.2 ***serializable***

This class defines the types of the attribute keys used with `AttributedCharacterIterator` and `AttributedString`. It defines several constant `Attribute` keys that are commonly used with multilingual text and input methods. The `LANGUAGE` key represents the language of the underlying text. The value of this key should be a `Locale` object. The `READING` key represents arbitrary reading information associated with text. The value must be an `Annotation` object. The `INPUT_METHOD_SEGMENT` key serves to define text segments (usually words) that an input method operates on. The value of this attribute should be an `Annotation` object that contains `null`. Other classes may subclass this class and define other attribute keys that are useful in other circumstances or problem domains. See, for example, `java.awt.font.TextAttribute` in *Java Foundation Classes in a Nutshell* (O'Reilly).

```
public static class AttributedCharacterIterator.Attribute
        implements Serializable {
// Protected Constructors
     protected Attribute(String name);
// Public Constants
     public static final AttributedCharacterIterator.Attribute
        INPUT_METHOD_SEGMENT;
     public static final AttributedCharacterIterator.Attribute LANGUAGE;
     public static final AttributedCharacterIterator.Attribute READING;
// Public Methods Overriding Object
     public final boolean equals(Object obj);
     public final int hashCode( );
     public String toString( );
// Protected Instance Methods
     protected String getName( );
     protected Object readResolve( ) throws java.io.InvalidObjectException;
}
```

Subclasses

`Format.Field`

Passed To

`AttributedCharacterIterator.{getAttribute( ),getRunLimit( ), getRunStart( )},AttributedString.{addAttribute( ), AttributedString( ),getIterator( )}`

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

AttributedString java.text

Java 1.2

This class represents text and associated attributes. An `AttributedString` can be defined in terms of an underlying `AttributedCharacterIterator` or an underlying `String`. Additional attributes can be specified with the `addAttribute( )` and `addAttributes( )` methods. `getIterator( )` returns an `AttributedCharacterIterator` over the `AttributedString` or over a specified portion of the string. Note that two of the `getIterator( )` methods take an array of `Attribute` keys as an argument. These methods return an `AttributedCharacterIterator` that ignores all attributes that are not in the specified array. If the array argument is `null`, however, the returned iterator contains all attributes.

```
public class AttributedString {
// Public Constructors
     public AttributedString(String text);
     public AttributedString(AttributedCharacterIterator text);
     public AttributedString(String text, java.util.Map<?
        extends AttributedCharacterIterator.Attribute,?> attributes);
     public AttributedString(AttributedCharacterIterator text, int beginIndex,
        int endIndex);
     public AttributedString(AttributedCharacterIterator text, int beginIndex,
        int endIndex, AttributedCharacterIterator.Attribute[ ] attributes);
// Public Instance Methods
     public void addAttribute(AttributedCharacterIterator.Attribute attribute,
        Object value);
     public void addAttribute(AttributedCharacterIterator.Attribute attribute,
        Object value, int beginIndex, int endIndex);
     public void addAttributes(java.util.Map<?
        extends AttributedCharacterIterator.Attribute,?> attributes,
        int beginIndex, int endIndex);
     public AttributedCharacterIterator getIterator( );
     public AttributedCharacterIterator
        getIterator(AttributedCharacterIterator.Attribute[ ] attributes);
     public AttributedCharacterIterator
        getIterator(AttributedCharacterIterator.Attribute[ ] attributes,
        int beginIndex, int endIndex);
}
```

Bidi java.text

Java 1.4

The `Bidi` class implements the " Unicode Version 3.0 Bidirectional Algorithm" for working with Arabic and Hebrew text in which letters run right-to-left and numbers run left-to-right. It is named after the first four letters of "bidirectional." A full description of

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

the bidirectional text handling and the bidirectional algorithim is beyond the scope of this book, but the simplest use case for this class is outlined here. Create a `Bidi` object by passing an `AttributedCharacterIterator` or a `String` and one of the `DIRECTION` constants (to indicate the base direction of the text) to the `Bidi( )` constructor. Or use `createLineBidi( )` to return a substring of an existing `Bidi` object (this is usually done when formatting a paragraph of text to fit on individual lines).

Once you have a `Bidi` object, use `isLeftToRight( )` and `isRightToLeft( )` to determine whether all the text has the same direction. If both of these methods return `false` (which is the same as `isMixed( )` returning `true`) then you cannot treat the text as a single run of uni-directional text. In this case, you must break it into two or more runs of unidirectional text. `getRunCount( )` returns the number of distinct runs of text. For each such numbered run, `getRunStart( )` returns the index of the first character of the run, and `getRunLimit( )` returns the index of the first character past the end of the run. `getRunLevel( )` returns the *level* of the text, which is an integer that represents the direction and nesting level of the text. Even levels represent left-to-right text, and odd levels represent right-to-left text. The level divided by two is the nesting level of the text. For example, left-to-right text embedded within right-to-left text has a level of 2.

```
public final class Bidi {
// Public Constructors
     public Bidi(AttributedCharacterIterator paragraph);
     public Bidi(String paragraph, int flags);
     public Bidi(char[ ] text, int textStart, byte[ ] embeddings,
        int embStart, int paragraphLength, int flags);
// Public Constants
     public static final int DIRECTION_DEFAULT_LEFT_TO_RIGHT;            =-2
     public static final int DIRECTION_DEFAULT_RIGHT_TO_LEFT;            =-1
     public static final int DIRECTION_LEFT_TO_RIGHT;                    =0
     public static final int DIRECTION_RIGHT_TO_LEFT;                    =1
// Public Class Methods
     public static void reorderVisually(byte[ ] levels, int levelStart,
        Object[ ] objects, int objectStart, int count);
     public static boolean requiresBidi(char[ ] text, int start, int limit);
// Public Instance Methods
     public boolean baseIsLeftToRight( );
     public Bidi createLineBidi(int lineStart, int lineLimit);
     public int getBaseLevel( );
     public int getLength( );
     public int getLevelAt(int offset);
     public int getRunCount( );
     public int getRunLevel(int run);
     public int getRunLimit(int run);
     public int getRunStart(int run);
     public boolean isLeftToRight( );
     public boolean isMixed( );
     public boolean isRightToLeft( );
// Public Methods Overriding Object
     public String toString( );
}
```

BreakIterator **java.text**

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

Java 1.1 ***cloneable***

This class determines character, word, sentence, and line breaks in a block of text in a way that is independent of locale and text encoding. As an abstract class, `BreakIterator` cannot be instantiated directly. Instead, you must use one of the class methods `getCharacterInstance( )`, `getWordInstance( )`, `getSentenceInstance( )`, or `getLineInstance( )` to return an instance of a nonabstract subclass of `BreakIterator`. These various factory methods return a `BreakIterator` object that is configured to locate the requested boundary types and is localized to work for the optionally specified locale.

Once you have obtained an appropriate `BreakIterator` object, use `setText( )` to specify the text in which to locate boundaries. To locate boundaries in a Java `String` object, simply specify the string. To locate boundaries in text that uses some other encoding, you must specify a `CharacterIterator` object for that text so that the `BreakIterator` object can locate the individual characters of the text. Having set the text to be searched, you can determine the character positions of characters, words, sentences, or line breaks with the `first( )`, `last( )`, `next( )`, `previous( )`, `current( )`, and `following( )` methods, which perform the obvious functions. Note that these methods do not return text itself, but merely the position of the appropriate word, sentence, or line break.

Figure 15-2. java.text.BreakIterator


```
public abstract class BreakIterator implements Cloneable {
// Protected Constructors
     protected BreakIterator( );
// Public Constants
     public static final int DONE;                                        =-1
// Public Class Methods
     public static java.util.Locale[ ] getAvailableLocales( );     synchronized
     public static BreakIterator getCharacterInstance( );
     public static BreakIterator getCharacterInstance(java.util.Locale where);
     public static BreakIterator getLineInstance( );
     public static BreakIterator getLineInstance(java.util.Locale where);
     public static BreakIterator getSentenceInstance( );
     public static BreakIterator getSentenceInstance(java.util.Locale where);
     public static BreakIterator getWordInstance( );
     public static BreakIterator getWordInstance(java.util.Locale where);
// Protected Class Methods
5.0  protected static int getInt(byte[ ] buf, int offset);
5.0  protected static long getLong(byte[ ] buf, int offset);
5.0  protected static short getShort(byte[ ] buf, int offset);
// Public Instance Methods
     public abstract int current( );
     public abstract int first( );
     public abstract int following(int offset);
     public abstract CharacterIterator getText( );
1.2  public boolean isBoundary(int offset);
     public abstract int last( );
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

```
     public abstract int next( );
     public abstract int next(int n);
1.2  public int preceding(int offset);
     public abstract int previous( );
     public void setText(String newText);
     public abstract void setText(CharacterIterator newText);
// Public Methods Overriding Object
     public Object clone( );
}
```

CharacterIterator **java.text**

Java 1.1 ***cloneable***

This interface defines an API for portably iterating through the characters that make up a string of text, regardless of the encoding of that text. Such an API is necessary because the number of bytes per character is different for different encodings, and some encodings even use variable-width characters within the same string of text. In addition to allowing iteration, a class that implements the `CharacterIterator` interface for non-Unicode text also performs translation of characters from their native encoding to standard Java Unicode characters.

`CharacterIterator` is similar to `java.util.Enumeration`, but is somewhat more complex than that interface. The `first( )` and `last( )` methods return the first and last characters in the text, and the `next( )` and `prev( )` methods allow you to loop forward or backwards through the characters of the text. These methods return the `DONE` constant when they go beyond the first or last character in the text; a test for this constant can be used to terminate a loop. The `CharacterIterator` interface also allows random access to the characters in a string of text. The `getBeginIndex( )` and `getEndIndex( )` methods return the character positions for the start and end of the string, and `setIndex( )` sets the current position. `getIndex( )` returns the index of the current position, and `current( )` returns the character at that position.

Figure 15-3. java.text.CharacterIterator


```
public interface CharacterIterator extends Cloneable {
// Public Constants
     public static final char DONE;        = \uFFFF
// Public Instance Methods
     Object clone( );
     char current( );
     char first( );
     int getBeginIndex( );
     int getEndIndex( );
     int getIndex( );
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

```
        char last( );
        char next( );
        char previous( );
        char setIndex(int position);
    }
```

Implementations

`AttributedCharacterIterator`, `StringCharacterIterator`

Passed To

`BreakIterator.setText( )`, `CollationElementIterator.setText( )`, `RuleBasedCollator.getCollationElementIterator( )`

Returned By

`BreakIterator.getText( )`

ChoiceFormat java.text

Java 1.1 ***cloneable serializable***

This class is a subclass of `Format` that converts a number to a `String` in a way reminiscent of a `switch` statement or an enumerated type. Each `ChoiceFormat` object has an array of doubles known as its *limits* and an array of strings known as its *formats*. When the `format( )` method is called to format a number `x`, the `ChoiceFormat` finds an index `i` such that:

```
limits[i] <= x < limits[i+1]
```

If `x` is less than the first element of the array, the first element is used, and if it is greater than the last, the last element is used. Once the index `i` has been determined, it is used as the index into the array of strings, and the indexed string is returned as the result of the `format( )` method.

A `ChoiceFormat` object may also be created by encoding its limits and formats into a single string known as its *pattern*. A typical pattern looks like the one below, used to return the singular or plural form of a word based on the numeric value passed to the `format( )` method:

```
ChoiceFormat cf = new ChoiceFormat("0#errors|1#error|2#errors");
```

A `ChoiceFormat` object created in this way returns the string "errors" when it formats the number 0 or any number greater than or equal to 2. It returns "error" when it formats the number 1. In the syntax shown here, note the pound sign (`#`) used to separate the limit number from the string that corresponds to that case and the vertical bar (`|`) used to

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

separate the individual cases. You can use the `applyPattern( )` method to change the pattern used by a `ChoiceFormat` object; use `toPattern( )` to query the pattern it uses.

Figure 15-4. java.text.ChoiceFormat


```
public class ChoiceFormat extends NumberFormat {
// Public Constructors
     public ChoiceFormat(String newPattern);
     public ChoiceFormat(double[ ] limits, String[ ] formats);
// Public Class Methods
     public static final double nextDouble(double d);
     public static double nextDouble(double d, boolean positive);
     public static final double previousDouble(double d);
// Public Instance Methods
     public void applyPattern(String newPattern);
     public Object[ ] getFormats( );
     public double[ ] getLimits( );
     public void setChoices(double[ ] limits, String[ ] formats);
     public String toPattern( );
// Public Methods Overriding NumberFormat
     public Object clone( );
     public boolean equals(Object obj);
     public StringBuffer format(long number, StringBuffer toAppendTo,
        FieldPosition status);
     public StringBuffer format(double number, StringBuffer toAppendTo,
        FieldPosition status);
     public int hashCode( );
     public Number parse(String text, ParsePosition status);
}
```

CollationElementIterator java.text

Java 1.1

A `CollationElementIterator` object is returned by the `getCollationElementIterator( )` method of the `RuleBasedCollator` object. The purpose of this class is to allow a program to iterate (with the `next( )` method) through the characters of a string, returning ordering values for each of the collation keys in the string. Note that collation keys are not exactly the same as characters. In the traditional Spanish collation order, for example, the two-character sequence "ch" is treated as a single collation key that comes alphabetically between the letters "c" and "d." The value returned by the `next( )` method is the collation order of the next collation key in the string. This numeric value can be directly compared to the value returned by `next( )` for other `CollationElementIterator` objects. The value returned by `next( )` can also be decomposed into primary, secondary, and tertiary ordering values with the static methods of this class. This class is used by `RuleBasedCollator` to implement its

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

`compare( )` method and to create `CollationKey` objects. Few applications ever need to use it directly.

```
public final class CollationElementIterator {
// No Constructor
// Public Constants
     public static final int NULLORDER;                                    =-1
// Public Class Methods
     public static final int primaryOrder(int order);
     public static final short secondaryOrder(int order);
     public static final short tertiaryOrder(int order);
// Public Instance Methods
1.2  public int getMaxExpansion(int order);
1.2  public int getOffset( );
     public int next( );
1.2  public int previous( );
     public void reset( );
1.2  public void setOffset(int newOffset);
1.2  public void setText(String source);
1.2  public void setText(CharacterIterator source);
}
```

Returned By

`RuleBasedCollator.getCollationElementIterator( )`

CollationKey java.text

Java 1.1 ***comparable***

`CollationKey` objects compare strings more quickly than is possible with `Collation.compare( )`. Objects of this class are returned by `Collation.getCollationKey( )`. To compare two `CollationKey` objects, invoke the `compareTo( )` method of key `A`, passing the key `B` as an argument (both `CollationKey` objects must be created through the same `Collation` object). The return value of this method is less than zero if the key `A` is collated before the key `B`, equal to zero if they are equivalent for the purposes of collation, or greater than zero if the key `A` is collated after the key `B`. Use `getSourceString( )` to obtain the string represented by a `CollationKey`.

Figure 15-5. java.text.CollationKey


```
public final class CollationKey implements Comparable<CollationKey> {
// No Constructor
// Public Instance Methods
     public int compareTo(CollationKey target);    Implements:Comparable
     public String getSourceString( );
     public byte[ ] toByteArray( );
// Methods Implementing Comparable
     public int compareTo(CollationKey target);
// Public Methods Overriding Object
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

```
    public boolean equals(Object target);
    public int hashCode( );
}
```

Returned By

`Collator.getCollationKey( ),RuleBasedCollator.getCollationKey( )`

Collator java.text

Java 1.1 ***cloneable***

This class compares, orders, and sorts strings in a way appropriate for the default locale or some other specified locale. Because it is an abstract class, it cannot be instantiated directly. Instead, you must use the static `getInstance( )` method to obtain an instance of a `Collator` subclass that is appropriate for the default or specified locale. You can use `getAvailableLocales( )` to determine whether a `Collator` object is available for a desired locale.

Once an appropriate `Collator` object has been obtained, you can use the `compare( )` method to compare strings. The possible return values of this method are -1, 0, and 1, which indicate, respectively, that the first string is collated before the second, that the two are equivalent for collation purposes, and that the first string is collated after the second. The `equals( )` method is a convenient shortcut for testing two strings for collation equivalence.

When sorting an array of strings, each string in the array is typically compared more than once. Using the `compare( )` method in this case is inefficient. A more efficient method for comparing strings multiple times is to use `getCollationKey( )` for each string to create `CollationKey` objects. These objects can then be compared to each other more quickly than the strings themselves can be compared.

You can customize the way the `Collator` object performs comparisons by calling `setStrength( )`. If you pass the constant `PRIMARY` to this method, the comparison looks only at primary differences in the strings; it compares letters but ignores accents and case differences. If you pass the constant `SECONDARY`, it ignores case differences but does not ignore accents. And if you pass `TERTIARY` (the default), the `Collator` object takes both accents and case differences into account in its comparison.

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

Figure 15-6. java.text.Collator


```
public abstract class Collator implements java.util.Comparator<Object>,
        Cloneable {
// Protected Constructors
     protected Collator( );
// Public Constants
     public static final int CANONICAL_DECOMPOSITION;                    =1
     public static final int FULL_DECOMPOSITION;                         =2
     public static final int IDENTICAL;                                  =3
     public static final int NO_DECOMPOSITION;                           =0
     public static final int PRIMARY;                                    =0
     public static final int SECONDARY;                                  =1
     public static final int TERTIARY;                                   =2
// Public Class Methods
     public static java.util.Locale[ ] getAvailableLocales( );   synchronized
     public static Collator getInstance( );                synchronized
     public static Collator
        getInstance(java.util.Locale desiredLocale);     synchronized
// Public Instance Methods
     public abstract int compare(String source, String target);
     public boolean equals(Object that);    Implements:Comparator
     public boolean equals(String source, String target);
     public abstract CollationKey getCollationKey(String source);
     public int getDecomposition( );                synchronized
     public int getStrength( );               synchronized
     public void setDecomposition(int decompositionMode);   synchronized
     public void setStrength(int newStrength);              synchronized
// Methods Implementing Comparator
1.2  public int compare(Object o1, Object o2);
     public boolean equals(Object that);
// Public Methods Overriding Object
     public Object clone( );
     public abstract int hashCode( );
}
```

Subclasses

`RuleBasedCollator`

DateFormat java.text

Java 1.1 ***cloneable serializable***

This class formats and parses dates and times in a locale-specific way. As an abstract class, it cannot be instantiated directly, but it provides a number of static methods that return instances of a concrete subclass you can use to format dates in a variety of ways. The `getDateInstance( )` methods return a `DateFormat` object suitable for formatting dates in either the default locale or a specified locale. A formatting style may also optionally be specified; the constants `FULL`, `LONG`, `MEDIUM`, `SHORT`, and `DEFAULT` specify this style. Similarly, the `getTimeInstance( )` methods return a `DateFormat` object that formats and parses times, and the `getDateTimeInstance( )` methods return a `DateFormat` object that formats both dates and times. These methods also optionally take a format style

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

constant and a `Locale`. Finally, `getInstance( )` returns a default `DateFormat` object that formats both dates and times in the `SHORT` format.

Once you have created a `DateFormat` object, you can use the `setCalendar( )` and `setTimeZone( )` methods if you want to format the date using a calendar or time zone other than the default. The various `format( )` methods convert `java.util.Date` objects to strings using whatever format is encapsulated in the `DateFormat` object. The `parse( )` and `parseObject( )` methods perform the reverse operation; they parse a string formatted according to the rules of the `DateFormat` object and convert it into to a `Date` object. The `DEFAULT`, `FULL`, `MEDIUM`, `LONG`, and `SHORT` constants specify how verbose or compact the formatted date or time should be. The remaining constants, which all end with `_FIELD`, specify various fields of formatted dates and times and are used with the `FieldPosition` object that is optionally passed to `format( )`.

Figure 15-7. java.text.DateFormat


```
public abstract class DateFormat extends Format {
// Protected Constructors
     protected DateFormat( );
// Public Constants
     public static final int AM_PM_FIELD;                                =14
     public static final int DATE_FIELD;                                 =3
     public static final int DAY_OF_WEEK_FIELD;                          =9
     public static final int DAY_OF_WEEK_IN_MONTH_FIELD;                 =11
     public static final int DAY_OF_YEAR_FIELD;                          =10
     public static final int DEFAULT;                                    =2
     public static final int ERA_FIELD;                                  =0
     public static final int FULL;                                       =0
     public static final int HOUR0_FIELD;                                =16
     public static final int HOUR1_FIELD;                                =15
     public static final int HOUR_OF_DAY0_FIELD;                         =5
     public static final int HOUR_OF_DAY1_FIELD;                         =4
     public static final int LONG;                                       =1
     public static final int MEDIUM;                                     =2
     public static final int MILLISECOND_FIELD;                          =8
     public static final int MINUTE_FIELD;                               =6
     public static final int MONTH_FIELD;                                =2
     public static final int SECOND_FIELD;                               =7
     public static final int SHORT;                                      =3
     public static final int TIMEZONE_FIELD;                             =17
     public static final int WEEK_OF_MONTH_FIELD;                        =13
     public static final int WEEK_OF_YEAR_FIELD;                         =12
     public static final int YEAR_FIELD;                                 =1
// Nested Types
1.4  public static class Field extends Format.Field;
// Public Class Methods
     public static java.util.Locale[ ] getAvailableLocales( );
     public static final DateFormat getDateInstance( );
     public static final DateFormat getDateInstance(int style);
     public static final DateFormat getDateInstance(int style,
        java.util.Locale aLocale);
     public static final DateFormat getDateTimeInstance( );
     public static final DateFormat getDateTimeInstance(int dateStyle,
        int timeStyle);
     public static final DateFormat getDateTimeInstance(int dateStyle,
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

```
        int timeStyle, java.util.Locale aLocale);
    public static final DateFormat getInstance( );
    public static final DateFormat getTimeInstance( );
    public static final DateFormat getTimeInstance(int style);
    public static final DateFormat getTimeInstance(int style,
        java.util.Locale aLocale);
// Public Instance Methods
    public final String format(java.util.Date date);
    public abstract StringBuffer format(java.util.Date date,
        StringBuffer toAppendTo, FieldPosition fieldPosition);
    public java.util.Calendar getCalendar( );
    public NumberFormat getNumberFormat( );
    public java.util.TimeZone getTimeZone( );
    public boolean isLenient( );
    public java.util.Date parse(String source) throws ParseException;
    public abstract java.util.Date parse(String source, ParsePosition pos);
    public void setCalendar(java.util.Calendar newCalendar);
    public void setLenient(boolean lenient);
    public void setNumberFormat(NumberFormat newNumberFormat);
    public void setTimeZone(java.util.TimeZone zone);
// Public Methods Overriding Format
    public Object clone( );
    public final StringBuffer format(Object obj, StringBuffer toAppendTo,
        FieldPosition fieldPosition);
    public Object parseObject(String source, ParsePosition pos);
// Public Methods Overriding Object
    public boolean equals(Object obj);
    public int hashCode( );
// Protected Instance Fields
    protected java.util.Calendar calendar;
    protected NumberFormat numberFormat;
}
```

Subclasses

`SimpleDateFormat`

DateFormat.Field java.text

Java 1.4 ***serializable***

This class defines a typesafe enumeration of `AttributedCharacterIterator.Attribute` objects that may be used by the `AttributedCharacterIterator` returned by the `formatToCharacterIterator( )` inherited from `Format`, or that may be used when creating a `FieldPosition` object with which to obtain the bounds of a specific date field in formatted output. Note that the constants defined by this class correspond closely to the integer constants defined by `java.util.Calendar`, and that this class defines methods for converting between the two sets of constants.

```
public static class DateFormat.Field extends Format.Field {
// Protected Constructors
    protected Field(String name, int calendarField);
// Public Constants
    public static final DateFormat.Field AM_PM;
    public static final DateFormat.Field DAY_OF_MONTH;
    public static final DateFormat.Field DAY_OF_WEEK;
    public static final DateFormat.Field DAY_OF_WEEK_IN_MONTH;
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

```
    public static final DateFormat.Field DAY_OF_YEAR;
    public static final DateFormat.Field ERA;
    public static final DateFormat.Field HOUR0;
    public static final DateFormat.Field HOUR1;
    public static final DateFormat.Field HOUR_OF_DAY0;
    public static final DateFormat.Field HOUR_OF_DAY1;
    public static final DateFormat.Field MILLISECOND;
    public static final DateFormat.Field MINUTE;
    public static final DateFormat.Field MONTH;
    public static final DateFormat.Field SECOND;
    public static final DateFormat.Field TIME_ZONE;
    public static final DateFormat.Field WEEK_OF_MONTH;
    public static final DateFormat.Field WEEK_OF_YEAR;
    public static final DateFormat.Field YEAR;
// Public Class Methods
    public static DateFormat.Field ofCalendarField(int);
// Public Instance Methods
    public int getCalendarField( );
// Protected Methods Overriding AttributedCharacterIterator.Attribute
    protected Object readResolve( ) throws java.io.InvalidObjectException;
}
```

DateFormatSymbols java.text

Java 1.1 ***cloneable serializable***

This class defines accessor methods for the various pieces of data, such as names of months and days, used by `SimpleDateFormat` to format and parse dates and times. You do not typically need to use this class unless you are formatting dates for an unsupported locale or in some highly customized way.

Figure 15-8. java.text.DateFormatSymbols


```
public class DateFormatSymbols implements Cloneable, Serializable {
// Public Constructors
    public DateFormatSymbols( );
    public DateFormatSymbols(java.util.Locale locale);
// Public Instance Methods
    public String[ ] getAmPmStrings( );
    public String[ ] getEras( );
    public String getLocalPatternChars( );
    public String[ ] getMonths( );
    public String[ ] getShortMonths( );
    public String[ ] getShortWeekdays( );
    public String[ ] getWeekdays( );
    public String[ ][ ] getZoneStrings( );
    public void setAmPmStrings(String[ ] newAmpms);
    public void setEras(String[ ] newEras);
    public void setLocalPatternChars(String newLocalPatternChars);
    public void setMonths(String[ ] newMonths);
    public void setShortMonths(String[ ] newShortMonths);
    public void setShortWeekdays(String[ ] newShortWeekdays);
    public void setWeekdays(String[ ] newWeekdays);
    public void setZoneStrings(String[ ][ ] newZoneStrings);
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

```
// Public Methods Overriding Object
     public Object clone( );
     public boolean equals(Object obj);
     public int hashCode( );
}
```

Passed To

`SimpleDateFormat.{setDateFormatSymbols( ),SimpleDateFormat( )}`

Returned By

`SimpleDateFormat.getDateFormatSymbols( )`

DecimalFormat java.text

Java 1.1 ***cloneable serializable***

This is the concrete `Format` class used by `NumberFormat` for all locales that use base 10 numbers. Most applications do not need to use this class directly; they can use the static methods of `NumberFormat` to obtain a default `NumberFormat` object for a desired locale and then perform minor locale-independent customizations on that object.

Applications that require highly customized number formatting and parsing may create custom `DecimalFormat` objects by passing a suitable pattern to the `DecimalFormat( )` constructor method. The `applyPattern( )` method can change this pattern. A pattern consists of a string of characters from the table below. For example:

```
"$#,##0.00;($#,##0.00)"
```

Character	Meaning
#	A digit; zeros show as absent.
0	A digit; zeros show as 0.
.	The locale-specific decimal separator.
,	The locale-specific grouping separator (comma).
-	The locale-specific negative prefix.
%	Shows value as a percentage.
;	Separates positive number format (on left) from optiona negative number format (on right).
'	Quotes a reserved character, so it appears literally in the output (apostrophe).
other	Appears literally in output.

A `DecimalFormatSymbols` object can be specified optionally when creating a `DecimalFormat` object. If one is not specified, a `DecimalFormatSymbols` object suitable for the default locale is used.

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

In Java 5.0, `DecimalFormat` can return `java.math.BigDecimal` values from its `parse( )` method. Call `setParseBigDecimal( )` to enable this feature. This is useful when working with very large numbers, very precise numbers, or financial applications that use `BigDecimal` to avoid rounding errors.

Figure 15-9. java.text.DecimalFormat


```
public class DecimalFormat extends NumberFormat {
// Public Constructors
     public DecimalFormat( );
     public DecimalFormat(String pattern);
     public DecimalFormat(String pattern, DecimalFormatSymbols symbols);
// Public Instance Methods
     public void applyLocalizedPattern(String pattern);
     public void applyPattern(String pattern);
     public DecimalFormatSymbols getDecimalFormatSymbols( );
     public int getGroupingSize( );                          default:3
     public int getMultiplier( );                            default:1
     public String getNegativePrefix( );                     default:"-"
     public String getNegativeSuffix( );                     default:""
     public String getPositivePrefix( );                     default:""
     public String getPositiveSuffix( );                     default:""
     public boolean isDecimalSeparatorAlwaysShown( );        default:false
5.0  public boolean isParseBigDecimal( );    default:false
     public void setDecimalFormatSymbols(DecimalFormatSymbols newSymbols);
     public void setDecimalSeparatorAlwaysShown(boolean newValue);
     public void setGroupingSize(int newValue);
     public void setMultiplier(int newValue);
     public void setNegativePrefix(String newValue);
     public void setNegativeSuffix(String newValue);
5.0  public void setParseBigDecimal(boolean newValue);
     public void setPositivePrefix(String newValue);
     public void setPositiveSuffix(String newValue);
     public String toLocalizedPattern( );
     public String toPattern( );
// Public Methods Overriding NumberFormat
     public Object clone( );
     public boolean equals(Object obj);
5.0  public final StringBuffer format(Object number, StringBuffer toAppendTo,
        FieldPosition pos);
     public StringBuffer format(double number, StringBuffer result,
        FieldPosition fieldPosition);
     public StringBuffer format(long number, StringBuffer result,
        FieldPosition fieldPosition);
1.4  public java.util.Currency getCurrency( );
5.0  public int getMaximumFractionDigits( ); default:3
5.0  public int getMaximumIntegerDigits( );           default:2147483647
5.0  public int getMinimumFractionDigits( );          default:0
5.0  public int getMinimumIntegerDigits( );           default:1
     public int hashCode( );
     public Number parse(String text, ParsePosition pos);
1.4  public void setCurrency(java.util.Currency currency);
1.2  public void setMaximumFractionDigits(int newValue);
1.2  public void setMaximumIntegerDigits(int newValue);
1.2  public void setMinimumFractionDigits(int newValue);
1.2  public void setMinimumIntegerDigits(int newValue);
// Public Methods Overriding Format
1.4  public AttributedCharacterIterator formatToCharacterIterator(Object obj);
}
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

DecimalFormatSymbols java.text

Java 1.1 ***cloneable serializable***

This class defines the various characters and strings, such as the decimal point, percent sign, and thousands separator, used by `DecimalFormat` when formatting numbers. You do not typically use this class directly unless you are formatting dates for an unsupported locale or in some highly customized way.

Figure 15-10. java.text.DecimalFormatSymbols


```
public final class DecimalFormatSymbols implements Cloneable, Serializable {
// Public Constructors
     public DecimalFormatSymbols( );
     public DecimalFormatSymbols(java.util.Locale locale);
// Public Instance Methods
1.4  public java.util.Currency getCurrency( );
1.2  public String getCurrencySymbol( );    default:"$"
     public char getDecimalSeparator( );                          default:.
     public char getDigit( );                                     default:#
     public char getGroupingSeparator( );                         default:,
     public String getInfinity( );                                default:"\u221E"
1.2  public String getInternationalCurrencySymbol( );   default:"USD"
     public char getMinusSign( );                                 default:-
1.2  public char getMonetaryDecimalSeparator( );   default:.
     public String getNaN( );                                     default:"\uFFFD"
     public char getPatternSeparator( );                          default:;
     public char getPercent( );                                   default:%
     public char getPerMill( );                                   default:\u2030
     public char getZeroDigit( );                                 default:0
1.4  public void setCurrency(java.util.Currency currency);
1.2  public void setCurrencySymbol(String currency);
     public void setDecimalSeparator(char decimalSeparator);
     public void setDigit(char digit);
     public void setGroupingSeparator(char groupingSeparator);
     public void setInfinity(String infinity);
1.2  public void setInternationalCurrencySymbol(String currencyCode);
     public void setMinusSign(char minusSign);
1.2  public void setMonetaryDecimalSeparator(char sep);
     public void setNaN(String NaN);
     public void setPatternSeparator(char patternSeparator);
     public void setPercent(char percent);
     public void setPerMill(char perMill);
     public void setZeroDigit(char zeroDigit);
// Public Methods Overriding Object
     public Object clone( );
     public boolean equals(Object obj);
     public int hashCode( );
}
```

Passed To

`DecimalFormat.{DecimalFormat( ),setDecimalFormatSymbols( )}`

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

Returned By

`DecimalFormat.getDecimalFormatSymbols( )`

FieldPosition java.text

Java 1.1

`FieldPosition` objects are optionally passed to the `format( )` methods of the `Format` class and its subclasses to return information about the start and end positions of a specific part or "field" of the formatted string. This kind of information is often useful for aligning formatted strings in columns—for example, aligning the decimal points in a column of numbers.

The field of interest is specified when the `FieldPosition( )` constructor is called. The `NumberFormat` and `DateFormat` classes define integer various constants (which end with the string `_FIELD`) that can be used here. In Java 1.4 and later you can also construct a `FieldPosition` by specifying the `Format.Field` object that identifies the field. (For constant `Field` instances, see `DateFormat.Field`, `MessageFormat.Field` and `NumberFormat.Field`.)

After a `FieldPosition` has been created and passed to a `format( )` method, use `getBeginIndex( )` and `getEndIndex( )` methods of this class to obtain the starting and ending character positions of the desired field of the formatted string.

```
public class FieldPosition {
// Public Constructors
1.4  public FieldPosition(Format.Field attribute);
     public FieldPosition(int field);
1.4  public FieldPosition(Format.Field attribute, int fieldID);
// Public Instance Methods
     public int getBeginIndex( );
     public int getEndIndex( );
     public int getField( );
1.4  public Format.Field getFieldAttribute( );
1.2  public void setBeginIndex(int bi);
1.2  public void setEndIndex(int ei);
// Public Methods Overriding Object
1.2  public boolean equals(Object obj);
1.2  public int hashCode( );
1.2  public String toString( );
}
```

Passed To

`ChoiceFormat.format( )`, `DateFormat.format( )`, `DecimalFormat.format( )`, `Format.format( )`, `MessageFormat.format( )`, `NumberFormat.format( )`, `SimpleDateFormat.format( )`

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

Format **java.text**

Java 1.1 ***cloneable serializable***

This abstract class is the base class for all number, date, and string formatting classes in the `java.text` package. It defines the key formatting and parsing methods that are implemented by all subclasses. `format( )` converts an object to a string using the formatting rules encapsulated by the `Format` subclass and optionally appends the resulting string to an existing `StringBuffer`. `parseObject( )` performs the reverse operation; it parses a formatted string and returns the corresponding object. Status information for these two operations is returned in `FieldPosition` and `ParsePosition` objects.

Java 1.4 defined a variant on the `format( )` method. `formatToCharacterIterator( )` performs the same formating operation as `format( )` but returns the result as an `AtttributedCharacterIterator` which uses attributes to identify the various parts (such the integer part, the decimal separtator, and the fractional part of a formatted number) of the formatted string. The attribute keys are all instances of the `Format.Field` inner class. Each of the `Format` subclasses define a `Field` subclass that defines a set of `Field` constants, (such as `NumberFormat.Field.DECIMAL_SEPARATOR`) for use by the character iterator returned by this method. See `ChoiceFormat`, `DateFormat`, `MessageFormat`, and `NumberFormat` for subclasses that perform specific types of formatting.

Figure 15-11. java.text.Format


```
public abstract class Format implements Cloneable, Serializable {
// Public Constructors
     public Format( );
// Nested Types
1.4  public static class Field extends AttributedCharacterIterator.Attribute;
// Public Instance Methods
     public final String format(Object obj);
     public abstract StringBuffer format(Object obj, StringBuffer toAppendTo,
        FieldPosition pos);
1.4  public AttributedCharacterIterator formatToCharacterIterator(Object obj);
     public Object parseObject(String source) throws ParseException;
     public abstract Object parseObject(String source, ParsePosition pos);
// Public Methods Overriding Object
     public Object clone( );
}
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

Subclasses

`DateFormat`, `MessageFormat`, `NumberFormat`

Passed To

`MessageFormat.{setFormat( ), setFormatByArgumentIndex( ), setFormats( ), setFormatsByArgumentIndex( )}`

Returned By

`MessageFormat.{getFormats( ), getFormatsByArgumentIndex( )}`

Format.Field java.text

Java 1.4 ***serializable***

This inner class extends `AttributedCharacterIterator.Attribute` and serves as the common superclass for `DateFormat.Field`, `MessageFormat.Field`, and `NumberFormat.Field`. See those specific subclasses for details.

```
public static class Format.Field extends AttributedCharacterIterator.Attribute {
// Protected Constructors
     protected Field(String name);
}
```

Subclasses

`DateFormat.Field`, `MessageFormat.Field`, `NumberFormat.Field`

Passed To

`FieldPosition.FieldPosition( )`

Returned By

`FieldPosition.getFieldAttribute( )`

MessageFormat java.text

Java 1.1 ***cloneable serializable***

This class formats and substitutes objects into specified positions in a message string (also known as the pattern string). It provides the closest Java equivalent to the `printf( )` function of the C programming language. If a message is to be displayed only a single time, the simplest way to use the `MessageFormat` class is through the static `format( )` method. This method is passed a message or pattern string and an array of argument objects to be formatted and substituted into the string. If the message is to be displayed several times, it makes more sense to create a `MessageFormat` object, supplying the pattern string, and then call the `format( )` instance method of this object, supplying the array of objects to be formatted into the message.

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

The message or pattern string used by the `MessageFormat` contains digits enclosed in curly braces to indicate where each argument should be substituted. The sequence "`{0}`" indicates that the first object should be converted to a string (if necessary) and inserted at that point, while the sequence "`{3}`" indicates that the fourth object should be inserted. If the object to be inserted is not a string, `MessageFormat` checks to see if it is a `Date` or a subclass of `Number`. If so, it uses a default `DateFormat` or `NumberFormat` object to convert the value to a string. If not, it simply invokes the object's `toString( )` method to convert it.

A digit within curly braces in a pattern string may be followed optionally by a comma, and one of the words "date", "time", "number", or "choice", to indicate that the corresponding argument should be formatted as a date, time, number, or choice before being substituted into the pattern string. Any of these keywords can additionally be followed by a comma and additional pattern information to be used in formatting the date, time, number, or choice. (See `SimpleDateFormat`, `DecimalFormat`, and `ChoiceFormat` for more information.)

You can pass a `Locale` to the constructor or call `setLocale( )` to specify a nondefault locale that the `MessageFormat` should use when obtaining `DateFormat` and `NumberFormat` objects to format dates, time, and numbers inserted into the pattern. You can change the `Format` object used at a particular position in the pattern with the `setFormat( )` method, or change all `Format` objects with `setFormats( )`. Both of these methods depend on the order of in which arguments are displayed in the pattern string. The pattern string is often subject to localization and the arguments may appear in different orders in different localizations of the pattern. Therefore, in Java 1.4 and later it is usually more convenient to use the "ByArgumentIndex" versions of the `setFormat( )`, `setFormats( )` methods, and `getFormats( )` methods.

You can set a new pattern for the `MessageFormat` object by calling `applyPattern( )`, and you can obtain a string that represents the current formatting pattern by calling `toPattern( )`. `MessageFormat` also supports a `parse( )` method that can parse an array of objects out of a specified string, according to the specified pattern.

Figure 15-12. java.text.MessageFormat


```
public class MessageFormat extends Format {
// Public Constructors
     public MessageFormat(String pattern);
1.4  public MessageFormat(String pattern, java.util.Locale locale);
// Nested Types
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

```
1.4  public static class Field extends Format.Field;
// Public Class Methods
     public static String format(String pattern, Object... arguments);
// Public Instance Methods
     public void applyPattern(String pattern);
     public final StringBuffer format(Object[ ] arguments, StringBuffer result,
        FieldPosition pos);
     public Format[ ] getFormats( );
1.4  public Format[ ] getFormatsByArgumentIndex( );
     public java.util.Locale getLocale( );
     public Object[ ] parse(String source) throws ParseException;
     public Object[ ] parse(String source, ParsePosition pos);
     public void setFormat(int formatElementIndex, Format newFormat);
1.4  public void setFormatByArgumentIndex(int argumentIndex, Format newFormat);
     public void setFormats(Format[ ] newFormats);
1.4  public void setFormatsByArgumentIndex(Format[ ] newFormats);
     public void setLocale(java.util.Locale locale);
     public String toPattern( );
// Public Methods Overriding Format
     public Object clone( );
     public final StringBuffer format(Object arguments, StringBuffer result,
        FieldPosition pos);
1.4  public AttributedCharacterIterator
        formatToCharacterIterator(Object arguments);
     public Object parseObject(String source, ParsePosition pos);
// Public Methods Overriding Object
     public boolean equals(Object obj);
     public int hashCode( );
}
```

MessageFormat.Field java.text

Java 1.4 ***serializable***

This class defines an `ARGUMENT AttributedCharacterIterator.Attribute` constant that is be used by the `AttributedCharacterIterator` returned by `MessageFormat.formatToCharacterIterator( )` to identify portions of the formatted message that are derived from the arguments passed to `formatToCharacterIterator( )`. The value associated with this `ARGUMENT` attribute will be an `Integer` specifying the argument number.

```
public static class MessageFormat.Field extends Format.Field {
// Protected Constructors
     protected Field(String name);
// Public Constants
     public static final MessageFormat.Field ARGUMENT;
// Protected Methods Overriding AttributedCharacterIterator.Attribute
     protected Object readResolve( ) throws java.io.InvalidObjectException;
}
```

NumberFormat java.text

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

Java 1.1 ***cloneable serializable***

This class formats and parses numbers in a locale-specific way. As an abstract class, it cannot be instantiated directly, but it provides a number of static methods that return instances of a concrete subclass you can use for formatting. The `getInstance( )` method returns a `NumberFormat` object suitable for normal formatting of numbers in either the default locale or in a specified locale. `getIntegerInstance( )`, `getCurrencyInstance( )`, and `getPercentInstance( )` return `NumberFormat` objects for formatting numbers that are integers, or represent monetary amounts or percentages. These methods return a `NumberFormat` suitable for the default locale, or for the specified `Locale` object. `getAvailableLocales( )` returns an array of locales for which `NumberFormat` objects are available. In Java 1.4 and later, use `setCurrency( )` to provide a `java.util.Currency` object for use when formating monetary values. Note that the `NumberFormat` class is not intended for the display of very large or very small numbers that require exponential notation, and it may not gracefully handle infinite or `NaN` (not-a-number) values.

Once you have created a suitable `NumberFormat` object, you can customize its localeindependent behavior with `setMaximumFractionDigits( )`, `setGroupingUsed( )`, and similar `set` methods. In order to customize the locale-dependent behavior, you can use `instanceof` to test if the `NumberFormat` object is an instance of `DecimalFormat`, and, if so, cast it to that type. The `DecimalFormat` class provides complete control over number formatting. Note, however, that a `NumberFormat` customized in this way may no longer be appropriate for the desired locale.

After creating and customizing a `NumberFormat` object, you can use the various `format( )` methods to convert numbers to strings or string buffers, and you can use the `parse( )` or `parseObject( )` methods to convert strings to numbers. You can also use the `formatToCharacterIterator( )` method inherited from `Format` (and overridden by `DecimalFormat`) in place of `format( )`. The constants defined by this class are to be used by the `FieldPosition` object.

Figure 15-13. java.text.NumberFormat


```
public abstract class NumberFormat extends Format {
// Public Constructors
     public NumberFormat( );
// Public Constants
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

```
    public static final int FRACTION_FIELD;                              =1
    public static final int INTEGER_FIELD;                               =0
// Nested Types
1.4 public static class Field extends Format.Field;
// Public Class Methods
    public static java.util.Locale[ ] getAvailableLocales( );
    public static final NumberFormat getCurrencyInstance( );
    public static NumberFormat getCurrencyInstance(java.util.Locale inLocale);
    public static final NumberFormat getInstance( );
    public static NumberFormat getInstance(java.util.Locale inLocale);
1.4 public static final NumberFormat getIntegerInstance( );
1.4 public static NumberFormat getIntegerInstance(java.util.Locale inLocale);
    public static final NumberFormat getNumberInstance( );
    public static NumberFormat getNumberInstance(java.util.Locale inLocale);
    public static final NumberFormat getPercentInstance( );
    public static NumberFormat getPercentInstance(java.util.Locale inLocale);
// Public Instance Methods
    public final String format(long number);
    public final String format(double number);
    public abstract StringBuffer format(long number, StringBuffer toAppendTo,
       FieldPosition pos);
    public abstract StringBuffer format(double number, StringBuffer toAppendTo,
       FieldPosition pos);
1.4 public java.util.Currency getCurrency( );
    public int getMaximumFractionDigits( );
    public int getMaximumIntegerDigits( );
    public int getMinimumFractionDigits( );
    public int getMinimumIntegerDigits( );
    public boolean isGroupingUsed( );
    public boolean isParseIntegerOnly( );
    public Number parse(String source) throws ParseException;
    public abstract Number parse(String source, ParsePosition parsePosition);
1.4 public void setCurrency(java.util.Currency currency);
    public void setGroupingUsed(boolean newValue);
    public void setMaximumFractionDigits(int newValue);
    public void setMaximumIntegerDigits(int newValue);
    public void setMinimumFractionDigits(int newValue);
    public void setMinimumIntegerDigits(int newValue);
    public void setParseIntegerOnly(boolean value);
// Public Methods Overriding Format
    public Object clone( );
    public StringBuffer format(Object number, StringBuffer toAppendTo,
       FieldPosition pos);
    public final Object parseObject(String source, ParsePosition pos);
// Public Methods Overriding Object
    public boolean equals(Object obj);
    public int hashCode( );
}
```

Subclasses

ChoiceFormat, DecimalFormat

Passed To

DateFormat.setNumberFormat( )

Returned By

DateFormat.getNumberFormat( )

Type Of

DateFormat.numberFormat

NumberFormat.Field java.text

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

Java 1.4 ***serializable***

This class defines a typesafe enumeration of `AttributedCharacterIterator.Attribute` objects that may be used by the `AttributedCharacterIterator` returned by `formatToCharacterIterator( )` method inherited from the `Format` class, or that may be used when creating a `FieldPosition` object to pass to `format( )` in order to obtain the bounds of a specific number field (such as the decimal point for aligning numbers) in formatted output.

```
public static class NumberFormat.Field extends Format.Field {
// Protected Constructors
     protected Field(String name);
// Public Constants
     public static final NumberFormat.Field CURRENCY;
     public static final NumberFormat.Field DECIMAL_SEPARATOR;
     public static final NumberFormat.Field EXPONENT;
     public static final NumberFormat.Field EXPONENT_SIGN;
     public static final NumberFormat.Field EXPONENT_SYMBOL;
     public static final NumberFormat.Field FRACTION;
     public static final NumberFormat.Field GROUPING_SEPARATOR;
     public static final NumberFormat.Field INTEGER;
     public static final NumberFormat.Field PERCENT;
     public static final NumberFormat.Field PERMILLE;
     public static final NumberFormat.Field SIGN;
// Protected Methods Overriding AttributedCharacterIterator.Attribute
     protected Object readResolve( ) throws java.io.InvalidObjectException;
}
```

ParseException java.text

Java 1.1 ***serializable checked***

Signals that a string has an incorrect format and cannot be parsed. It is typically thrown by the `parse( )` or `parseObject( )` methods of `Format` and its subclasses, but is also thrown by certain methods in the `java.text` package that are passed patterns or other rules in string form. The `getErrorOffset( )` method of this class returns the character position at which the parsing error occurred in the offending string.

Figure 15-14. java.text.ParseException


```
public class ParseException extends Exception {
// Public Constructors
     public ParseException(String s, int errorOffset);
// Public Instance Methods
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

```
    public int getErrorOffset( );
}
```

Thrown By

`DateFormat.parse( )`, `Format.parseObject( )`, `MessageFormat.parse( )`, `NumberFormat.parse( )`, `RuleBasedCollator.RuleBasedCollator( )`

ParsePosition java.text

Java 1.1

`ParsePosition` objects are passed to the `parse( )` and `parseObject( )` methods of `Format` and its subclasses. The `ParsePosition` class represents the position in a string at which parsing should begin or at which parsing stopped. Before calling a `parse( )` method, you can specify the starting position of parsing by passing the desired index to the `ParsePosition( )` constructor or by calling the `setIndex( )` of an existing `ParsePosition` object. When `parse( )` returns, you can determine where parsing ended by calling `getIndex( )`. When parsing multiple objects or values from a string, a single `ParsePosition` object can be used sequentially.

```
public class ParsePosition {
// Public Constructors
     public ParsePosition(int index);
// Public Instance Methods
1.2  public int getErrorIndex( );
     public int getIndex( );
1.2  public void setErrorIndex(int ei);
     public void setIndex(int index);
// Public Methods Overriding Object
1.2  public boolean equals(Object obj);
1.2  public int hashCode( );
1.2  public String toString( );
}
```

Passed To

`ChoiceFormat.parse( )`, `DateFormat.{parse( ),parseObject( )}`, `DecimalFormat.parse( )`, `Format.parseObject( )`, `MessageFormat.{parse( ),parseObject( )}`, `NumberFormat.{parse( ),parseObject( )}`, `SimpleDateFormat.parse( )`

RuleBasedCollator java.text

Java 1.1 ***cloneable***

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

This class is a concrete subclass of the abstract `Collator` class. It performs collations using a table of rules that are specified in textual form. Most applications do not use this class directly; instead they call `Collator.getInstance( )` to obtain a `Collator` object (typically a `RuleBasedCollator` object) that implements the default collation order for a specified or default locale. You should need to use this class only if you are collating strings for a locale that is not supported by default or if you need to implement a highly customized collation order.

Figure 15-15. java.text.RuleBasedCollator


```
public class RuleBasedCollator extends Collator {
// Public Constructors
     public RuleBasedCollator(String rules) throws ParseException;
// Public Instance Methods
1.2  public CollationElementIterator getCollationElementIterator(CharacterIterator source);
     public CollationElementIterator getCollationElementIterator(String source);
     public String getRules( );
// Public Methods Overriding Collator
     public Object clone( );
     public int compare(String source, String target);                synchronized
     public boolean equals(Object obj);
     public CollationKey getCollationKey(String source);              synchronized
     public int hashCode( );
}
```

SimpleDateFormat java.text

Java 1.1 ***cloneable serializable***

This is the concrete `Format` subclass used by `DateFormat` to handle the formatting and parsing of dates. Most applications should not use this class directly; instead, they should obtain a localized `DateFormat` object by calling one of the static methods of `DateFormat`.

`SimpleDateFormat` formats dates and times according to a pattern, which specifies the positions of the various fields of the date, and a `DateFormatSymbols` object, which specifies important auxiliary data, such as the names of months. Applications that require highly customized date or time formatting can create a custom `SimpleDateFormat` object by specifying the desired pattern. This creates a `SimpleDateFormat` object that uses the `DateFormatSymbols` object for the default locale. You may also specify an locale explicitly, to use the `DateFormatSymbols` object for that locale. You can even provide

Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

an explicit `DateFormatSymbols` object of your own if you need to format dates and times for an unsupported locale.

You can use the `applyPattern( )` method of a `SimpleDateFormat` to change the formatting pattern used by the object. The syntax of this pattern is described in the following table. Any characters in the format string that do not appear in this table appear literally in the formatted date.

Field	Full form	Short form
Year	`yyyy` (4 digits)	`yy` (2 digits)
Month	`MMM` (name)	`MM` (2 digits), `M` (1 or 2 digits)
Day of week	`EEEE`	`EE`
Day of month	`dd` (2 digits)	`d` (1 or 2 digits)
Hour (1-12)	`hh` (2 digits)	`h` (1 or 2 digits)
Hour (0-23)	`HH` (2 digits)	`H` (1 or 2 digits)
Hour (0-11)	`KK`	`K`
Hour (1-24)	`kk`	`k`
Minute	`mm`	
Second	`ss`	
Millisecond	`SSS`	
AM/PM	`a`	
Time zone	`zzzz`	`zz`
Day of week in month	`F` (e.g., 3rd Thursday)	
Day in year	`DDD` (3 digits)	`D` (1, 2, or 3 digits)
Week in year	`ww`	
Era (e.g., BC/AD)	`G`	

Figure 15-16. java.text.SimpleDateFormat


```
public class SimpleDateFormat extends DateFormat {
// Public Constructors
     public SimpleDateFormat( );
     public SimpleDateFormat(String pattern);
     public SimpleDateFormat(String pattern, java.util.Locale locale);
     public SimpleDateFormat(String pattern, DateFormatSymbols formatSymbols);
// Public Instance Methods
     public void applyLocalizedPattern(String pattern);
     public void applyPattern(String pattern);
1.2  public java.util.Date get2DigitYearStart( );
     public DateFormatSymbols getDateFormatSymbols( );
1.2  public void set2DigitYearStart(java.util.Date startDate);
     public void setDateFormatSymbols(DateFormatSymbols newFormatSymbols);
     public String toLocalizedPattern( );
     public String toPattern( );
// Public Methods Overriding DateFormat
     public Object clone( );
     public boolean equals(Object obj);
     public StringBuffer format(java.util.Date date, StringBuffer toAppendTo,
        FieldPosition pos);
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.

```
     public int hashCode( );
     public java.util.Date parse(String text, ParsePosition pos);
// Public Methods Overriding Format
1.4  public AttributedCharacterIterator formatToCharacterIterator(Object obj);
}
```

StringCharacterIterator java.text

Java 1.1 ***cloneable***

This class is a trivial implementation of the `CharacterIterator` interface that works for text stored in Java `String` objects. See `CharacterIterator` for details.

Figure 15-17. java.text.StringCharacterIterator


```
public final class StringCharacterIterator implements CharacterIterator {
// Public Constructors
     public StringCharacterIterator(String text);
     public StringCharacterIterator(String text, int pos);
     public StringCharacterIterator(String text, int begin, int end, int pos);
// Public Instance Methods
1.2  public void setText(String text);
// Methods Implementing CharacterIterator
     public Object clone( );
     public char current( );
     public char first( );
     public int getBeginIndex( );
     public int getEndIndex( );
     public int getIndex( );
     public char last( );
     public char next( );
     public char previous( );
     public char setIndex(int p);
// Public Methods Overriding Object
     public boolean equals(Object obj);
     public int hashCode( );
}
```


Java in a Nutshell, 5th Edition By David Flanagan ISBN: 0596007736 Publisher: O'Reilly
Print Publication Date: 2005/03/01
Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
User number: 628024 Copyright 2008, Safari Books Online, LLC.
This PDF is exclusively for your use in accordance with the Safari Terms of Service. No part of it may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher. Redistribution or other use that violates the fair use priviledge under U.S. copyright laws (see 17 USC107) or that otherwise violates the Safari Terms of Service is strictly prohibited.