APPENDIX

E

JAVA CLASSES

This appendix describes various Java classes and some of their commonly used members that are used or mentioned in this book. For a detailed description of all Java classes, visit the Web site *http://java.sun.com*.

CLASS: Boolean (PACKAGE java.lang)

Constructors

```
public Boolean(boolean param)
  //Creates a Boolean object initialized with the value specified
  //by param
public Boolean(String str)
  //Creates a Boolean object initialized to true if the string
  //specified by str is not null and is equal, ignoring case,
  //to the string "true".
```

Methods

```
public boolean booleanValue()
  //Return true if the value of the object is true;
  //otherwise it returns false.
public int hashCode()
  //Returns the hash code of this object.
public String toString()
  //If the state of the object is true, returns the String
  //object with the value "true"; Otherwise returns the String
  //object with the value "false".
public static Boolean valueOf(String str)
  //Returns the Boolean object initialized to the value
  //specified by str.
```

CLASS: BorderLayout (PACKAGE java.awt)

Constructors

```
public BorderLayout()
  //Creates a border layout with 0 horizontal
  //and 0 vertical gaps between components.
```

```
public BorderLayout(int hgap, int vgap)
  //Creates a border layout with the specified
  //horizontal and vertical gaps between components.
  //The horizontal gap is specified by hgap
  //and the vertical gap is specified by vgap.
```

Methods

```
public int getHgap()
  //Returns the horizontal gap between components.
public void setHgap(int hgap)
  //Sets the horizontal gap between components.
public int getVgap()
  //Returns the vertical gap between components.
public void setVgap(int vgap)
  //Sets the vertical gap between components.
```

Class: `BufferedReader` (Package `java.io`)

Constructors

```
public BufferedReader(Reader rd)
  //Creates a BufferedReader object. The object is initialized
  //using rd.
public BufferedReader(Reader rd, int size)
  //Creates a BufferedReader object. The object is initialized
  //using rd and to the size specified by size. The default size
  //is 8192 characters.
```

Methods

```
public void close() throws IOException
  //Closes the BufferedReader object.
public int read() throws IOException
  //Returns a single character as an int from the
  //BufferedReader stream. Returns -1 if the end of stream
  //is reached.
public String readLine() throws IOException
  //Returns a line of characters as a string.
  //The line of characters ends with \r, \n, or \r\n.
  //Returns null if the end of stream is reached.
public boolean ready() throws IOException
  //Returns true if the object is ready to be read. If the object
  //is nonempty, the state is true.
public void reset() throws IOException
  //Resets the BufferedReader object.
```

```
public long skip(long num) throws IOException
  //Skips the next number of characters specified by num. The
  //number of characters skipped are returned.
```

Class: Character (Package java.lang)

Constructors

```
public Character(char ch)
  //Creates a Character object with the value specified by ch.
```

Methods

```
public static int digit(char ch, int base)
  //Returns the numeric value of ch in the radix base
  //specified by base
public static int forDigit(int digit, int base)
  //Returns the numeric value of digit in the radix base
  //specified by base
public boolean equals(Object obj)
  //Returns true if this object is equal to the object
  //specified by obj; otherwise it returns false
public static int getNumericValue(char ch)
  //Returns the Unicode representation, as a nonnegative integer
  //of the character specified by ch. If ch has no
  //numeric representation -1 is returned; If ch cannot be
  //represented as a nonnegative integer -2 is returned.
public int hashCode()
  //Returns the hash code of this object.
public static boolean isDigit(char ch)
  //Returns true if ch is a digit; false otherwise.
public static boolean isLetter(char ch)
  //Returns true if ch is a letter; false otherwise.
public static boolean isLowerCase(char ch)
  //Returns true if ch is a lowercase letter; false otherwise.
public static boolean isUpperCase(char ch)
  //Returns true if ch is an uppercase letter; false otherwise.
public static boolean isSpaceChar(char ch)
  //Returns true if ch is the space character; false otherwise.
public static boolean isWhitespace(char ch)
  //Returns true if ch is a whitespace character;
  //false otherwise.
public static char toLowerCase(char ch)
  //Returns the character that is the lowercase equivalent of ch.
  //If ch does not have a corresponding lowercase letter,
  //it returns ch.
```

```
public static char toUpperCase(char ch)
  //Returns the character that is the uppercase equivalent of ch.
  //If ch does not have a corresponding uppercase letter,
  //it returns ch.
public String toString()
  //Returns a string representation of the object.
```

Interface: Collection<E> (Package java.util)

Methods

```
boolean add(E obj) throws ClassCastException,
                  NullPointerException, IllegalArgumentException
  //This method ensures that this collection contains the
  //element specified by obj (optional operation). Returns true
  //if this collection changed as a result of the call. Returns
  //false if this collection does not permit duplicates and
  //already contains the specified element.
boolean addAll(Collection<? extends E> coll) throws
               UnsupportedOperationException,
               ClassCastException, NullPointerException,
               IllegalArgumentException
  //Adds all of the elements of coll to this collection
  //(optional operation).
void clear()
  //Removes all of the elements from this collection
  //(optional operation).
boolean contains(Object obj) throws ClassCastException,
                             NullPointerException
  //Returns true if this collection contains the element
  //specified by obj.
boolean containsAll(Collection<?> coll) throws
                    ClassCastException, NullPointerException
  //Returns true if this collection contains all of the elements
  //of coll.
boolean equals(Object obj)
  //Compares the object specified by obj with this collection for
  //equality.
int hashCode()
  //Returns the hash code value for this collection.
boolean isEmpty()
  //Returns true if this collection contains no elements.
Iterator<E> iterator()
  //Returns an iterator over the elements in this collection.
boolean remove(Object obj) throws
               UnsupportedOperationException,
               ClassCastException, NullPointerException
  //Removes a single instance of obj from this collection, if
  //it is present (optional operation). Returns true if this
  //collection contained the specified element.
boolean removeAll(Collection<?> coll) throws
                 UnsupportedOperationException,
                 ClassCastException, NullPointerException,
                 IllegalArgumentException
  //Removes all this collection's elements that are also
  //contained in coll (optional operation).
boolean retainAll(Collection<?> coll) throws
                  ClassCastException, NullPointerException
  //Retains only the elements in this collection that are
  //contained in coll. In other words, all elements in this
  //collection that are not contained in the collection coll are
  //removed. It returns true if this collection is changed as a
  //result of the call.
int size()
  //Returns the number of elements in this collection. If
  //this collection contains more than Integer.MAX_VALUE
  //elements, it returns Integer.MAX_VALUE.
Object [] toArray()
  //Returns an array containing all of the elements in this
  //collection. If this collection makes any guarantees as to
  //what order its elements are returned by its iterator, this
  //method must return the elements in the same order.
<T> T[] toArray(T[] a) throws ArrayStoreException,
                              NullPointerException
  //Returns an array containing all of the elements in this
  //collection; the runtime type of the returned array is that
  //of the specified array. If the collection fits in the
  //specified array, it is returned therein. Otherwise, a new
  //array is allocated with the runtime type of the specified
  //array and the size of this collection.
```

Class: Color (Package java.awt)

Named Constants

```
public static final Color black =  new Color(0, 0, 0);
public static final Color blue =  new Color(0, 0, 255);
public static final Color cyan =  new Color(0, 255, 255);
public static final Color darkGray =  new Color(64, 64, 64);
public static final Color gray =  new Color(128, 128, 128);
public static final Color green =  new Color(0, 255, 0);
public static final Color lightGray =  new Color(192, 192, 192);
public static final Color magenta =  new Color(255, 0, 255);
public static final Color orange =  new Color(255, 200, 0);
public static final Color pink =  new Color(255, 175, 175);
public static final Color red =  new Color(255, 0, 0);
public static final Color white =  new Color (255, 255, 255);
public static final Color yellow =  new Color(255, 255, 0);
```

Constructors

```
public Color(int r, int g, int b)
  //Creates a new Color with the red value r, green value g,
  //and blue value b. In this case, r, g, and b can be
  //between 0 and 255.
public Color(int rgb)
  //Creates a new Color with the red value r, green value g,
  //and blue value b; RGB value consisting of the red component
  //in bits 16-23, the green component in bits 8-15, and the
  //blue component in bits 0-7.
public Color(float r, float g, float b)
  //Creates a new Color with the red value r, green value g,
  //and blue value b. In this case, r, g, and b can be between 0
  //and 1.0.
```

Methods

```
public Color brighter()
  //Returns a Color that is brighter
public Color darker()
  //Returns a Color that is darker
public boolean equals(Object o)
  //Checks whether Color objects have identical RGB values
public int getBlue()
  //Returns the value of the blue component
public static Color getColor(String str)
  //Returns the color specified by str. The string specified by
  //str contains an int value.
public static Color getColor(String str, Color cl)
  //Returns the color specified by str. The string specified by
  //str contains an int value. If the color cannot be
  //determined, the color cl is returned.
public static Color getColor(String str, int cl)
  //Returns the color specified by str. The string specified by
  //str contains an int value. If the color cannot be
  //determined, the color cl is returned.
public int getGreen()
  //Returns the value of the green component
public int getRGB()
  //Returns the RGB value. RGB value is r * 65536 + g * 256 + b
public int getRed()
```

```
  //Returns the value of the red component
public static  Color getColor(String nm)
  //Finds a color in the system properties.
public String toString()
  //Returns a String with information about the color
```

CLASS: Component (PACKAGE java.awt)

Constructors

```
protected Component()
  //Creates a new component
```

Methods

```
public String getName()
  //Returns the name of the component.
public void setName(String name)
  //Sets the name of the component.
public Container getParent()
  //Returns the parent of this component.
public boolean isValid()
  //Returns true if this component is valid.
  //A component is valid if it is correctly sized and placed
  //within its parent container and all its children are also
  //valid.
public boolean isDisplayable()
  //Returns true if this component is displayable.
public boolean isVisible()
  //Returns true if this component is visible.
public void setVisible(boolean tog)
  //If tog is true, sets the component to visible;
  //If tog is false, the component is not shown.
public boolean isShowing()
  //Returns true if this component is showing on screen.
public boolean isEnabled()
  //Returns true if this component is enabled.
public void setEnabled(boolean b)
  //Component is enabled if b is true.
  //Component is disabled if b is false.
public Color getBackground()
  //Returns the background color of this component
public Color getForeground()
  //Returns the foreground color of this component.
public void setBackground(Color c)
  //Sets the background color of this component to color c.
```

```
public void setForeground(Color c)
  //Sets the foreground color of this component to color c.
public Font getFont()
  //Returns the font of this component.
public void setFont(Font ft)
  //Sets the font of this component to ft.
public void setSize(int w, int h)
  //Sets the size of this component.
public boolean isVisible()
  //Returns true if the component is visible.
public void setVisible(boolean tog)
  //If tog is true, sets the component to visible;
  //If tog is false, the component is not shown.
public void paint(Graphics g)
  //Paints the component with the graphic component specified by g.
public void repaint()
  //Repaints the component.
public void repaint(int x, int y, int wid, int ht)
  //Repaints the rectangular portion of the component from (x, y) to
  //(x + wid, y + ht)
public void setLocation(int x, int y)
  //Sets the component at the location (x, y).
public String toString()
  //Returns a string representation of this component.
public void update(Graphics g)
  //Invokes the paint method.
public void validate()
  //Validates this container and all of its subcomponents. The
  //method validate is used to cause a container to lay out its
  //subcomponents once more. Typically called after the components
  //it contains have been added to or modified.
public void addFocusListener(FocusListener lis)
  //Adds the focus listener specified by lis.
public void addKeyListener(KeyListener lis)
  //Adds the key listener specified by lis.
public void addMouseListener(MouseListener lis)
  //Adds the mouse listener specified by lis.
public void addMouseMotionListener(MouseMotionListener lis)
  //Adds the mouse motion listener specified by lis.
public void removeFocusListener(FocusListener lis)
  //Removes the focus listener specified by lis.
public void removeKeyListener(KeyListener lis)
  //Removes the key listener specified by lis.
public void removeMouseListener(MouseListener lis)
  //Removes the mouse listener specified by lis.
public void removeMouseMotionListener(MouseMotionListener lis)
  //Removes the mouse motion listener specified by lis.
```

CLASS: Container (PACKAGE java.awt)

Constructor

```
public Container()
//Creates a new Container.
```

Methods

```
public Component add(Component comp)
  //Appends the specified component to the end of this container.
public Component add(Component comp, int index)
  //Adds the specified component to this container at the
  //given position.
public void setLayout(LayoutManager ob)
  //Method to set the layout of the pane.
public LayoutManager getLayout()
  //Returns the layout manager for this container.
public void setFont(Font ft)
  //Sets the font of this component to ft.
public void validate()
  //Validates this container and all of its subcomponents. The
  //method validate is used to cause a container to lay out its
  //subcomponents once more. Typically called after the
  //components it contains have been added to or modified.
public void remove(int index)
  //Removes the specified component from the container.
public void remove(Component comp)
  //Removes the specified component from this container.
public void removeAll()
  //Removes every component from this container.
public void paint(Graphics g)
  //Paints the container.
public void update(Graphics g)
  //Updates the container.
```

CLASS: DecimalFormat (PACKAGE java.text)

Constructors

```
public DecimalFormat()
  //Creates a DecimalFormat object with the default pattern.
public DecimalFormat(String str)
  //Creates a DecimalFormat object with the pattern specified by str.
public DecimalFormat(String str, DecimalFormatSymbols symbols)
```

```
  //Creates a DecimalFormat object with the pattern specified by
  //symbols.
```

Methods

```
public void applyPattern(String str)
  //Sets the pattern of the object.
public String toPattern()
  //Returns the pattern of the object as a string.
public object clone()
  //Returns a copy of the object.
public boolean equals(Object obj)
  //Returns true if this object is equal to the object
  //specified by obj; otherwise returns false.
public StringBuffer format(double num)
  //Returns a string containing the formatted num.
public StringBuffer format(long num)
  //Returns a string containing the formatted num.
public DecimalFormatSymbols getDecimalFormatSymbols()
  //Returns the decimal number format symbols of the object.
public void setDecimalFormatSymbols(DecimalFormatSymbols symbols)
  //Sets the decimal number format symbols of this object.
public int hashCode()
  //Returns the hash code of the object.
```

CLASS: Double (PACKAGE java.lang)

Named Constants

```
public static final double MAX_VALUE = 1.7976931348623157E308;
public static final double MIN_VALUE = 4.9E-324;

public static final double NEGATIVE_INFINITY = -1.0/0.0;
public static final double POSITIVE_INFINITY = 1.0/0.0;
```

Constructors

```
public Double(double num)
  //Creates a Double object initialized to the value specified
  //by num.
public Double(String str) throws NumberFormatException
  //Creates a Double object initialized to the value specified
  //by the num contained in str.
```

Methods

```
public byte byteValue()
  //Returns the value of the object as a byte value.
public short shortValue()
  //Returns the value of the object as a short value.
public int intValue()
  //Returns the value of the object as an int value.
public long longValue()
  //Returns the value of the object as a long value.
public double doubleValue()
  //Returns the value of the object as a double value.
public float floatValue()
  //Returns the value of the object as a float value.
public int hashCode()
  //Returns the hash code of the object.
public boolean equals(Object obj)
  //Returns true if the value of this object is equal
  //to the value of the object specified by obj;
  //otherwise returns false.
public boolean isInfinite()
  //Returns true if the value of this object is positive
  //or negative infinity; otherwise returns false.
public static boolean isInfinite(double num)
  //Returns true if the value of num is positive
  //or negative infinity; otherwise returns false.
public static double parseDouble(String str) throws
                                       NumberFormatException
  //Returns the value of the number contained in str.
public String toString()
  //Returns the double value, of the object, as a string.
public static String toString(double num)
  //Returns the value of num as a string.
public static Double valueOf(String str) throws
                                       NumberFormatException
  //Returns a Double object initialized to the value
  //specified by str.
```

CLASS: Exception (PACKAGE java.lang)

Constructors

```
public Exception()
  //Creates an Exception object.
public Exception(String str)
  //Creates an Exception object. The
  //parameter str specifies the message string.
```

CLASS: FileReader (PACKAGE java.io)

Constructors

```
public FileReader(FileDescriptor fd)
                  throws FileNotFoundException
  //Creates a FileReader object from a file descriptor.
public FileReader(String fileName) throws FileNotFoundException
  //Creates a FileReader object from a filename.
```

Methods

```
public void close() throws IOException
  //Closes the FileReader.
public int read() throws IOException
  //Returns a single character as an int from the FileReader.
  //Returns -1 if the end of stream is reached.
public boolean ready() throws IOException
  //Returns true if the object is ready to be read. If the
  //object is nonempty, the state is true.
public void reset() throws IOException
  //Resets the object.
```

CLASS: FileWriter (PACKAGE java.io)

Constructors

```
public FileWriter(String fileName) throws IOException
  //Creates a FileWriter object from a filename.
public FileWriter(String fileName, boolean a)
                                   throws IOException
  //Creates a FileWriter object from a filename.
  //The boolean a indicating whether or not to append to
  //the file.
public FileWriter(FileDescriptor fd)
  //Creates a FileWriter object from a file descriptor.
```

Methods

```
public String getEncoding()
  //Returns the name of the character encoding being used.
public void write(int c) throws IOException
  //Writes a single character.
public void write(char[] cbuf, int off, int len)
                               throws IOException
  //Writes a part of an array of characters.
```

```
public void write(String str, int off, int len)
                                  throws IOException
  //Writes a part of a string.
public void flush() throws IOException
  //Empty the stream.
public void close() throws IOException
  //Closes the stream.
```

Class: Float (Package java.lang)

Named Constants

```
public static final float MAX_VALUE = 3.4028235E38;
public static final float MIN_VALUE = 1.4E-45;

public static final float NEGATIVE_INFINITY = -1.0f/0.0f;
public static final float POSITIVE_INFINITY = 1.0f/0.0f;
```

Constructors

```
public Float(float num)
  //Creates a Float object initialized to the value specified
  //by num.
public Float(double num)
  //Creates a Float object initialized to the value specified
  //by num.
public Float(String str) throws NumberFormatException
  //Creates a Float object initialized to the value specified
  //by the num contained in str.
```

Methods

```
public byte byteValue()
  //Returns the value of the object as a byte value.
public short shortValue()
  //Returns the value of the object as a short value.
public int intValue()
  //Returns the value of the object as an int value.
public long longValue()
  //Returns the value of the object as a long value.
public double doubleValue()
  //Returns the value of the object as a double value.
public float floatValue()
  //Returns the value of the object as a float value.
public int hashCode()
  //Returns the hash code of the object.
public boolean equals(Object obj)
```

```
    //Returns true if the value of this object is equal
    //to the value of the object specified by obj;
    //otherwise returns false.
public boolean isInfinite()
    //Returns true if the value of this object is positive
    //or negative infinity; otherwise returns false.
public static boolean isInfinite(float num)
    //Returns true if the value of num is positive
    //or negative infinity; otherwise returns false.
public static float parseFloat(String str) throws
                                   NumberFormatException
    //Returns the value of the number contained in str.
public String toString()
    //Returns the float value, of the object, as a string
public static String toString(float num)
    //Returns the value of num as a string
public static Float valueOf(String str) throws
                                 NumberFormatException
    //Returns a Float object initialized to the value
    //specified by str.
```

Class: FlowLayout (Package java.awt)

Constructors

```
public FlowLayout()
    //Creates a new FlowLayout with a centered alignment.
    //Both vertical and horizontal gaps are set as
    //5 units (default).
public FlowLayout(int alignment)
    //Creates a new FlowLayout with the specified alignment.
    //Both vertical and horizontal gaps are set as
    //5 units (default).
    //The possible values of the alignment argument are
    //FlowLayout.LEFT, FlowLayout.RIGHT, or FlowLayout.CENTER.
public FlowLayout(int alignment, int hgap, int vgap)
    //Creates a new FlowLayout with the specified alignment
    //and the given horizontal and vertical gaps.
    //The possible values of the alignment argument are
    //FlowLayout.LEFT, FlowLayout.RIGHT, or FlowLayout.CENTER.
```

Methods

```
public int getAlignment()
    //Gets the alignment. Possible values are
    //FlowLayout.LEFT, FlowLayout.RIGHT, FlowLayout.CENTER,
    //FlowLayout.LEADING, or FlowLayout.TRAILING.
public void setAlignment(int alignment)
```

```
    //Sets the alignment. Possible values are
    //FlowLayout.LEFT, FlowLayout.RIGHT, FlowLayout.CENTER,
    //FlowLayout.LEADING, or FlowLayout.TRAILING.
public int getHgap()
    //Gets the horizontal gap between components.
public void setHgap(int hgap)
    //Sets the horizontal gap between components.
public int getVgap()
    //Gets the vertical gap between components.
public void setVgap(int vgap)
    //Sets the vertical gap between components.
```

Class: Font (Package java.awt)

Named Constants

```
int PLAIN: 0
int BOLD: 1
int ITALIC: 2
```

Constructors

```
public Font(String name, int style, int size)
    //Creates a new Font from the specified name, style,
    //and point size.
```

Methods

```
public String getFamily()
    //Returns the family name of this font.
public String getFontName()
    //Returns the font face name of this font.
public static Font getFont(Map attributes)
    //Returns a Font appropriate to this attribute set.
public static Font createFont(int fontFormat,
                              InputStream fontStream)
                       throws FontFormatException,
                              IOException
    //Creates a new Font with the specified font type and
    //input data.
public String getPSName()
    //Returns the Postscript name of this font.
public String getName()
    //Returns the logical name of this font.
public int getStyle()
    //Returns the style of this font.
    //The possible values are: PLAIN, BOLD, ITALIC,
    //or BOLD + ITALIC.
public int getSize()
    //Returns the point size of this font.
public boolean isPlain()
    //Returns true if font style is PLAIN; false otherwise.
public boolean isBold()
    //Returns true if font style is BOLD; false otherwise.
public boolean isItalic()
    //Returns true if font style is ITALIC; false otherwise.
public static Font getFont(String nm)
    //Returns a Font object from the system properties list.
public static Font getFont(String nm, Font font)
    //Gets the specified font from the system properties list.
```

CLASS: Graphics (PACKAGE java.awt)

Constructors

```
protected Graphics()
    //Constructs a Graphics object that defines a context in which
    //user can draw. This constructor cannot be called directly.
```

Methods

```
public abstract void clearRect(int x, int y, int w, int h)
    //Draws a rectangle with no fill pattern in the current
    //background color at the position (x, y) having a width w
    //and height h.
public abstract void clipRect(int x, int y, int w, int h)
    //Sets a clipping rectangle area at the position (x, y) of
    //width w and height h.
public abstract void copyArea(int x, int y, int w, int h,
                              int newx, int newy)
    //Copies rectangular area at (x, y) with the width w and
    //height h to new position (newx, newy).
public abstract Graphics create()
    //Returns a copy of the entire graphics context.
public abstract Graphics create(int x, int y, int w, int h)
    //Returns a copy of the rectangular graphics context at
    //position (x, y) having the width w and height h to
    //new position (newx, newy).
public abstract void dispose()
    //Draws a rectangle with no fill pattern in the current
    //background color at the position (x, y) having a width w and
    //height h.
```

```
public void draw3DRect(int x, int y, int w, int h, boolean t)
  //Draws a 3D rectangle at (x, y) with the width w, height h. If t is
  //true, rectangle will appear raised.
public abstract void drawArc(int x, int y, int w, int h,
                             int sangle, int aangle)
  //Draws an arc starting at the position (x, y) having a width w and
  //height h, starting at the angle sangle with an arc angle aangle.
  //Both angles are measured in degrees.
public void drawBytes(byte[] str, int i, int n, int x, int y)
  //Draws n bytes of array str, starting with the array index i, n
  //characters, at (x, y).
public void drawChars(char[] str, int i, int n, int x, int y)
  //Draws n characters of array str, starting with the array index i,
  //n characters, at (x, y).
public abstract boolean drawImage(Image img, int x, int y,
                                  Color c, ImageObserver ob)
  //Draws the image specified by img at (x, y). Any transparent
  //color pixels are drawn in color c, and ob monitors the
  //progress of the image.
public abstract boolean drawImage(Image img, int x, int y,
                                  ImageObserver ob)
  //Draws the image specified by img at (x, y). The ob monitors
  //the progress of the image.
public abstract boolean drawImage(Image img, int x, int w, int h,
                                int y, Color c, ImageObserver ob)
  //Draws the image specified by img at (x, y) having width w and
  //height h. Any transparent color pixels are drawn in color c,
  //and ob monitors the progress of the image.
public abstract boolean drawImage(Image img, int x, int w, int h,
                                  int y, ImageObserver ob)
  //Draws the image specified by img at (x, y) having width w and
  //height h. The ob monitors the progress of the image.
public abstract boolean drawImage(Image img, int xs1, int ys1,
                   int xs2, int ys2, int xd1, int yd1,
                   int xd2, int yd2, Color c, ImageObserver ob)
  //Draws the image specified by img from the area defined by
  //bounding rectangle, (xs1, ys1) to (xs2, ys2) in the area
  //defined by the rectangle (xd1, yd1) to (xd2, yd2). Any
  //transparent color pixels are drawn in color c.
  //The ob monitors the progress of the image.
public abstract boolean drawImage(Image img, int xs1, int ys1,
                            int xs2, int ys2, int xd1, int yd1,
                            int xd2, int yd2, ImageObserver ob)
  //Draws the image specified by img from the area defined by
  //bounding rectangle, (xs1, ys1) to (xs2, ys2) in the area
  //defined by the rectangle (xd1, yd1) to (xd2, yd2). The ob
  //monitors the progress of the image.
public abstract void drawLine(int xs, int ys, int xd, int yd)
  //Draws a line from (xs, ys) to (xd, yd).
```

```
public abstract void drawOval(int x, int y, int w, int h)
  //Draws an oval at position (x, y) of the width w and height h.
public abstract void drawPolygon(int[] x, int[] y, int num)
  //Draws a polygon with the points (x[0], y[0]), ...,
  //(x[num-1], y[num-1]). Here num is the number of points in
  //the polygon.
public void drawPolygon(Polygon poly)
  //Draws a polygon as defined by the object poly.
public void drawRect(int x, int y, int w, int h)
  //Draws a rectangle at the position (x, y) having a width w and
  //height h.
public abstract void drawRoundRect(int x, int y, int w, int h,
                                   int arcw, int arch)
  //Draws a round-cornered rectangle at the position (x, y)
  //having a width w and height h. The shape of the rounded
  //corners are determined by arc with width arcw and
  //height arch.
public abstract void drawString(String s, int x, int y)
  //Draws the string s at (x, y).
public void fill3DRect(int x, int y, int w, int h, boolean t)
  //Draws a 3D filled rectangle at (x, y) with the width w,
  //height h. If t is true, rectangle will appear raised.
  //The rectangle is filled with current color.
public abstract void fillArc(int x, int y, int w, int h,
                             int sangle, int aangle)
  //Draws a filled arc starting at the position (x, y) having a
  //width w and height h. The starting at angle sangle with
  //an arc angle aangle. Both angles are measured in degrees.
  //The arc is filled with current color.
public abstract void fillOval(int x, int y, int w, int h)
  //Draws a filled oval at the position (x, y) having a width w
  //and height h. The oval is filled with current color.
public abstract void fillPolygon(int[] x, int[] y, int num)
  //Draws a filled polygon with the points (x[0], y[0]), ...,
  //(x[num-1], y[num-1]). Here num is the number of points in
  //the polygon. The polygon is filled with the current color.
public void fillPolygon(Polygon poly)
  //Draws a filled polygon as defined the by the object poly.
  //The polygon is filled with the current color.
public abstract void fillRect(int x, int y, int w, int h)
  //Draws a filled rectangle at the position (x, y) having a
  //width w and height h. The rectangle is filled with the
  //current color.
public abstract void fillRoundRect(int x, int y, int w, int h,
                                   int arcw, int arch)
  //Draws a filled round-cornered rectangle at the position (x, y)
  //having a width w and height h. The shape of the rounded
  //corners are determined by the arc with the width arcw and height
  //arch. The rectangle is filled with the current color.
```

```
public abstract Shape getClip()
  //Returns a shape object of the current clipping area for
  //this graphics context.
public abstract Rectangle getClipBounds()
  //Returns the rectangle describing the bounds of the current
  //clipping area for this graphics context.
public abstract Color getColor()
  //Returns the current color for this graphics context.
public abstract void setColor(Color c)
  //Sets the current color for this graphics context.
public abstract Font getFont()
  //Returns the current font for this graphics context.
public abstract void setFont(Font f)
  //Sets the current font for this graphics context.
public FontMetrics getFontMetrics()
  //Returns the font metrics associated with this
  //graphics context.
public FontMetrics getFontMetrics(Font f)
  //Returns the font metrics associated with Font f.
public abstract void setClip(int x, int y, int w, int h)
  //Sets the clipping area as the rectangle at (x, y)
  //with width w and height h.
public abstract void setClip(Shape s)
  //Sets the clipping area to be a specified shape s.
public abstract void setXORmode(Color c)
  //Sets the current graphics context's paint mode to
  //overwrite any subsequent destinations with the alternating
  //current color and color c.
public void String toString()
  //Returns a string representation of this graphics context.
```

Class: GridLayout (Package java.awt)

Constructors

```
public GridLayout()
  //Creates a one row, one column grid layout.
public GridLayout(int rows, int cols)
  //Creates a grid layout with the specified number
  //of rows and columns. All grids are given equal size.
public GridLayout(int rows, int cols, int hgap, int vgap)
  //Creates a grid layout with the specified number of
  //rows and columns; and given vertical and horizontal gaps.
  //All grids are given equal size.
```

Methods

```
public int getRows()
  //Gets the number of rows.
public void setRows(int rows)
  //Sets the number of rows.
public int getColumns()
  //Gets the number of columns.
public void setColumns(int cols)
  //Sets the number of columns.
public int getHgap()
  //Gets the horizontal gap between components.
public void setHgap(int hgap)
  //Sets the horizontal gap between components.
public int getVgap()
  //Gets the vertical gap between components.
public void setVgap(int vgap)
  //Sets the vertical gap between components.
```

CLASS: HashMap<K,V> (PACKAGE java.util)

Constructors

```
public HashMap()
  //Constructs a new empty map with the default initial
  //capacity (16) and load factor (0.75).

public HashMap(int initialCapacity) throws
                              IllegalArgumentException
  //Constructs a new empty map with the initial capacity
  //specified by the parameter initialCapacity and load factor
  //of 0.75.
public HashMap(int initialCapacity, float loadFactor) throws
               IllegalArgumentException
  //Constructs a new empty map with the initial capacity
  //specified by the parameter initialCapacity and load factor
  //specified by the parameter loadFactor.
public HashMap (Collection<? extends E> c) throws
               NullPointerException
  //Constructs a new map containing the elements of the
  //collection c. The load factor is 0.75 and the initial
  //capacity is sufficient to contain the elements of the
  //collection c.
```

Methods

The class HashMap<K, V> contains the following methods. The descriptions of these methods are similar to the descriptions of the methods of the interface Map.

```
public void clear()
public Object clone()
public boolean containsKey(Object key)
public boolean containsValue(Object value)
public Set<Map.Entry<K,V>> entrySet()
public V get(Object key)
public boolean isEmpty()
public Set<K> keySet()
public V put(K key, V value)
public void putAll(Map<? extends K,? extends V> m)
public V remove(Object key)
public int size()
public Collection<V> values()
```

Class: HashSet<E> (Package java.util)

Constructors

```
public HashSet()
  //Constructs a new empty set with the default initial
  //capacity (16) and load factor (0.75).
public HashSet(int initialCapacity) throws
               IllegalArgumentException
  //Constructs a new empty set with the initial capacity
  //specified by the parameter initialCapacity and load factor
  //of 0.75.
public HashSet(int initialCapacity, float loadFactor)
               throws IllegalArgumentException
  //Constructs a new empty set with the initial capacity
  //specified by the parameter initialCapacity and load factor
  //specified by the parameter loadFactor.
public HashSet(Collection<? extends E> c) throws
               NullPointerException
  //Constructs a new set containing the elements of the
  //collection c. The load factor is 0.75 and the initial
  //capacity is sufficient to contain the elements of the
  //collection c.
```

Methods

```
public boolean add(E obj)
  //The element specified by the parameter obj is added to this
  //set if it is not already in the set. Returns true if the
  //set did not already contain obj.
public void clear()
  //Removes all of the elements from this set.
public Object clone()
  //Returns a shallow copy of this HashSet instance. The
  //elements themselves are not cloned.
public boolean contains(Object obj)
  //Returns true if this set contains the element specified by
  //the parameter obj.
public boolean isEmpty()
  //Returns true if this set contains no elements.
public Iterator<E> iterator()
  //Returns an iterator over the elements in this set. The
  //elements are returned in no particular order.
public boolean remove(Object obj)
  //Removes the element specified by the parameter obj from
  //this set if it is in the set. Returns true if the set
  //contained obj.
public int size()
  //Returns the number of elements in this set.
```

CLASS: InputStreamReader (PACKAGE java.io)

Constructors

```
public InputStreamReader(InputStream rd)
  //Creates an InputStreamReader object. The object is
  //initialized using rd.
```

Methods

```
public void close() throws IOException
  //Closes the InputStreamReader.
public int read() throws IOException
  //Returns a single character as an int from the
  //InputStreamReader. Returns -1 if the end of stream
  //is reached.
public boolean ready() throws IOException
  //Returns true if the object is ready to be read. If the
  //object is nonempty, the state is true.
public void reset() throws IOException
  //Resets the object.
```

Class: Integer (Package java.lang)

Named Constants

```
public static final int MAX_VALUE = 2147483647;
public static final int MIN_VALUE = -2147483648;
```

Constructors

```
public Integer(int num)
  //Creates an Integer object initialized to the value specified
  //by num.
public Integer(String str) throws NumberFormatException
  //Creates an Integer object initialized to the value specified
  //by the num contained in str.
```

Methods

```
public byte byteValue()
  //Returns the value of the object as a byte value.
public short shortValue()
  //Returns the value of the object as a short value.
public int intValue()
  //Returns the value of the object as an int value.
public long longValue()
  //Returns the value of the object as a long value.
public double doubleValue()
  //Returns the value of the object as a double value.
public float floatValue()
  //Returns the value of the object as a float value.
public int hashCode()
  //Returns the hash code of the object.
public boolean equals(Object obj)
  //Returns true if the value of this object is equal
  //to the value of the object specified by obj;
  //otherwise returns false.
public static int parseInt(String str) throws
                                  NumberFormatException
  //Returns the value of the number contained in str.
public static int parseInt(String str, int base) throws
                                   NumberFormatException
  //Returns the value, in the radix base, of the int
  //contained in str.
public static String toBinaryString(int num)
  //Returns the string representation of num in
```

```
      //binary (base 2).
    public static String toHexString(int num)
      //Returns the string representation of num in
      //hexadecimal (base 16).
    public static String toOctalString(int num)
      //Returns the string representation of num in octal
      //(base 8).
    public String toString()
      //Returns the int value, of the object, as a string.
    public static String toString(int num)
      //Returns the value of num as a string.
    public static String toString(int num, int base)
      //Returns the value of num, in the radix base, as a string.
    public static Integer valueOf(String str) throws
                                        NumberFormatException
      //Returns an Integer object initialized to the value
      //specified by str.
    public static Integer valueOf(String str, int base) throws
                                      NumberFormatException
      //Returns an Integer object initialized to the value,
      //in the radix base, specified by str.
```

Interface: ITERATOR<E> (PACKAGE java.util)

Methods

```
    boolean hasNext()
      //Returns true if the iteration has more elements.
    E next() throws NoSuchElementException
      //Returns the next element in the iteration.
      //If this method is called repeatedly until the method
      //hasNext() returns false, it will return each element in the
      //underlying collection exactly once.
    void remove() throws UnsupportedOperationException,
                         IllegalStateException
      //Removes from the underlying collection the last element
      //returned by the iterator (optional operation). This method
      //can be called only once per call to next. The behavior of
      //an iterator is unspecified if the underlying collection is
      //modified while the iteration is in progress in any way other
      //than by calling this method.
```

Class: JApplet (Package javax.swing)

Constructors

```
public JApplet() throws HeadlessException
  //Creates a swing applet instance.
```

Methods

```
public void init()
  //Called by the browser or applet viewer to inform this applet
  //that it has been loaded into the system.
public void start()
  //Called by the browser or applet viewer to inform this
  //applet that it should start its execution. It is
  //called after the init method and each time the applet is
  //revisited in a Web page.
public void stop()
  //Called by the browser or applet viewer to inform this
  //applet that it should stop its execution. It is called
  //before the method destroy.
public void destroy()
  //Called by the browser or applet viewer. Informs this
  //applet that it is being reclaimed and that it should
  //destroy any resources that it has allocated. The method
  //stop is called before destroy.
public void showStatus(String msg)
  //Displays the string msg in the status bar.
public Container getContentPane()
  //Returns the contentPane object for this applet.
public void setContentPane(Container contentPane)
  //Sets the contentPane object for this applet.
public JLayeredPane getLayeredPane()
  //Returns the layeredPane object for this applet.
public void setLayeredPane(JLayeredPane layeredPane)
  //Sets the layeredPane for this applet.
public JMenuBar getJMenuBar()
  //Returns the JMenuBar object for this applet.
public void setJMenuBar(JMenuBar menuBar)
  //Sets the JMenuBar object for this applet.
public URL getDocumentBase()
  //Returns the URL of the document that contains this applet.
public URL getCodeBase()
  //Returns the URL of this applet.
public void update(Graphics g)
  //Calls the paint() method.
protected String paramString()
```

```
  //Returns a string representation of this applet;
  //mainly used for debugging.
```

CLASS: JButton (PACKAGE javax.swing)

Constructors

```
public JButton()
  //Creates a button with no text or icon.
public JButton(Icon ic)
  //Creates a button with the icon specified by ic.
public JButton(String str)
  //Creates a button with the text specified by str.
public JButton(String str, Icon ic)
  //Creates a button with the text specified
  //by str and icon specified by ic.
```

Methods

```
public String getText()
  //Method to return the text contained in the button.
public void setText(String str)
  //Method to set the text of the button to the string specified
  //by str.
public boolean isSelected()
  //Returns true if the button is selected, false otherwise.
public void setSelected(boolean b)
  //Sets the state of the button to b. This method does not
  //trigger an actionEvent. For that, call doClick.
public void addActionListener(ActionListener obj)
  //Method to register a listener object to the button object.
public void doClick()
  //Programmatically perform a "click."
public void doClick(int msec)
  //Programmatically perform a "click." Button appears pressed
  //for msec milliseconds.
public Icon getIcon()
  //Returns the default icon.
public void setIcon(Icon icon)
  //Sets the default icon.
public Icon getPressedIcon()
  //Returns the pressed icon.
public Icon setPressedIcon()
  //Sets the pressed icon.
public Icon getSelectedIcon()
  //Returns the selected icon.
public void setSelectedIcon(Icon icon)
```

```
  //Sets the selected icon.
public Icon getDisabledIcon()
  //Returns the icon used when the button is disabled.
  //If there is no disabled icon, one from the default
  //icon is constructed.
public void setDisabledIcon(Icon icon)
  //Sets the disabled icon.
public int getVerticalTextPosition()
  //Returns the vertical position of the text.
  //Returns one of the following constant values:
  //SwingConstants.CENTER (the default)
  //SwingConstants.TOP
  //SwingConstants.BOTTOM
public void setVerticalTextPosition(int pos)
  //Sets the vertical position of the text.
  //possible values are:
  //SwingConstants.CENTER (the default)
  //SwingConstants.TOP
  //SwingConstants.BOTTOM
public int getHorizontalTextPosition()
  //Returns the horizontal position of the text.
  //Returns one of the following constant values:
  //SwingConstants.RIGHT
  //SwingConstants.LEFT
  //SwingConstants.CENTER
  //SwingConstants.LEADING
  //SwingConstants.TRAILING (the default)
public void setHorizontalTextPosition(int pos)
  //Sets the horizontal position of the text.
  //Possible values are:
  //SwingConstants.RIGHT
  //SwingConstants.LEFT
  //SwingConstants.CENTER
  //SwingConstants.LEADING
  //SwingConstants.TRAILING (the default)
public void setActionCommand(String actionCommand)
  //Sets the action command.
public String getActionCommand()
  //Returns the action command.
public int getMnemonic()
  //Returns the keyboard mnemonic. The mnemonic is the key
  //which when combined with the meta key (usually Alt) will
  //"click" this button if focus is within the button’s
  //ancestor window.
public void setMnemonic(int mnemonic)
  //Sets the keyboard mnemonic. The mnemonic is the key
  //which when combined with the meta key (usually Alt) will
  //"click" this button if focus is within the button’s
  //ancestor window.
```

```
public void setEnabled(boolean b)
  //Sets the button enabled if b is true;
  //disabled if b is false.
```

CLASS: JCheckBox (PACKAGE javax.swing)

Constructors

```
public JCheckBox()
  //Creates an initially unselected check box
  //with no label and no icon.
public JCheckBox(Icon icon)
  //Creates an initially unselected check box
  //with the specified icon and no label.
public JCheckBox(Icon icon, boolean selected)
  //Creates a check box with the specified
  //icon and selection state, but without label.
public JCheckBox(String text)
  //Creates an unselected check box with
  //the specified label.
public JCheckBox(String text, boolean selected)
  //Creates a check box with the specified
  //label and selection state.
public JCheckBox(String text, Icon icon)
  //Creates a check box with the specified image
  //and with specified label.
public JCheckBox(String text, Icon icon, boolean selected)
  //Creates a check box with the specified image
  //and selection state, and with the specified text.
```

Methods

```
public String getText()
  //Returns the text contained in the check box.
public void setText(String str)
  //Sets the text of check box to the string specified by str.
public boolean isSelected()
  //Returns true if the check box is selected, false otherwise.
public void setSelected(boolean b)
  //Sets the state of the check box to b. This method does not
  //trigger an actionEvent. For that, call doClick.
public void addActionListener(ActionListener obj)
  //Method to register a listener object to the check box
  //object
public void doClick()
  //Programmatically perform a "click."
public int getVerticalTextPosition()
  //Returns the vertical position of the text.
  //Returns one of the following constant values:
  //SwingConstants.CENTER (the default)
  //SwingConstants.TOP
  //SwingConstants.BOTTOM
public void setVerticalTextPosition(int pos)
  //Sets the vertical position of the text.
  //Possible values are:
  //SwingConstants.CENTER (the default)
  //SwingConstants.TOP
  //SwingConstants.BOTTOM
public int getHorizontalTextPosition()
  //Returns the horizontal position of the text.
  //Returns one of the following constant values:
  //SwingConstants.RIGHT
  //SwingConstants.LEFT
  //SwingConstants.CENTER
  //SwingConstants.LEADING
  //SwingConstants.TRAILING (the default)
public void setHorizontalTextPosition(int pos)
  //Sets the horizontal position of the text.
  //Possible values are:
  //SwingConstants.RIGHT
  //SwingConstants.LEFT
  //SwingConstants.CENTER
  //SwingConstants.LEADING
  //SwingConstants.TRAILING (the default)
public void setActionCommand(String actionCommand)
  //Sets the action command.
public String getActionCommand()
  //Returns the action command.
public int getMnemonic()
  //Returns the keyboard mnemonic. The mnemonic is the key
  //which when combined with meta key (usually Alt) will
  //"click" this check box if focus is within the check box's
  //ancestor window.
public void setMnemonic(int mnemonic)
  //Sets the keyboard mnemonic. The mnemonic is the key
  //which when combined with meta key (usually Alt) will
  //"click" this check box if focus is within the check box's
  //ancestor window.
public void setEnabled(boolean b)
  //Sets the check box enabled if b is true;
  //disabled if b is false.
```

CLASS: JComboBox (PACKAGE javax.swing)

Constructors

```
public JComboBox()
  //Creates a JComboBox with no items to select.
public JComboBox(Vector v)
  //Creates a JComboBox to display elements
  //in the vector provided as an input parameter.
public JComboBox(Object[] o)
  //Creates a JComboBox that displays
  //elements in the object array provided as an input parameter.
```

Methods

```
public void addItem(Object ob)
  //Adds an item to the list of items. In order for this
  //method to work, the JComboBox must use a mutable
  //data model.
public void insertItemAt(Object ob, int index)
  //Inserts an item at a given index in the list of items.
  //In order for this method to work, the JComboBox
  //must use a mutable data model.
public void removeItem(Object ob)
  //Removes an item from the list of items.
  //In order for this method to work, the JComboBox
  //must use a mutable data model.
public void removeItemAt(int index)
  //Removes the item at index from the list of items.
  //This method works only if the JComboBox uses a mutable
  //data model.
public void removeAllItems()
  //Removes all items from the item list.
  //This method works only if the JComboBox uses a mutable
  //data model.
public void addItemListener(ItemListener aListener)
  //Adds an ItemListener.
  //aListener will receive one or two ItemEvents when the
  //selected item changes.
public void setEnabled(boolean b)
  //If b is true, the combo box is enabled; thus items can
  //be selected.
public int getItemCount()
  //Returns the number of items in the list.
public Object getItemAt(int index)
  //Returns the list item at index. If index is out of range
  //a null value is returned.
public void setEditable(boolean b)
  //JComboBox field is editable if b is true. An editable
```

```
    //JComboBox allows the user to type into the field or
    //select an item from the list to initialize the field.
    //By default, a combo box is not editable.
  public boolean isEditable()
    //Returns false if the JComboBox is not editable.
    //By default, a combo box is not editable.
  public void setMaximumRowCount(int count)
    //Sets the maximum number of rows to be displayed.
    //If the number of items is greater than count, scrollbar
    //is used.
  public int getMaximumRowCount()
    //Returns the maximum number of items that can
    //display without a scrollbar.
```

CLASS: JFrame (PACKAGE javax.swing)

Constructors

```
  public JFrame()
    //Creates a JFrame object without any title.
  public JFrame(String s)
    //Creates a JFrame object with the title specified by s.
  public JFrame(GraphicsConfiguration gc)
    //Creates a JFrame object in the specified
    //GraphicsConfiguration of a screen device with no title.
  public JFrame(String t, GraphicsConfiguration gc)
    //Creates a JFrame object in the specified
    //GraphicsConfiguration of a screen device and title.
```

Methods

```
  public void setSize(int w, int h)
    //Sets the size of the window.
  public void setTitle(String s)
    //Sets the title of the window.
  public void setVisible(boolean b)
    //Method to display window in the program. If the value of
    //b is true the window will be displayed on the screen.
  public int getDefaultCloseOperation()
    //Returns the operation that occurs when the user closes
    //this frame.
  public void setDefaultCloseOperation(int operation)
    //Method to determine the action to be taken when the user
    //clicks on the window closing button to close the
    //window. Choices for the parameter operation are the named
    //constants – EXIT_ON_CLOSE, HIDE_ON_CLOSE, DISPOSE_ON_CLOSE,
    //and DO_NOTHING_ON_CLOSE. The named constant EXIT_ON_CLOSE
    //is defined in the class JFrame. The last three named
  //constants are defined in javax.swing.WindowConstants.
public void addWindowListener(WindowListener e)
  //Method to register a window listener object to a JFrame.
public void update(Graphics g)
  //Invokes the method paint(g).
public void setJMenuBar(JMenuBar mbar)
  //Sets the menubar for this JFrame.
public JMenuBar getJMenuBar()
  //Returns the menubar set on this JFrame.
public JRootPane getRootPane()
  //Returns the rootPane object for this JFrame.
protected void setRootPane(JRootPane r)
  //Sets the rootPane property.
public Container getContentPane()
  //Returns the contentPane object for this JFrame.
public void setContentPane(Container pane)
  //Sets the contentPane for this JFrame.
public JLayeredPane getLayeredPane()
  //Returns the layeredPane object for this JFrame.
public void setLayeredPane(JLayeredPane layeredPane)
  //Sets the layeredPane for this JFrame.
```

Class: JLabel (Package javax.swing)

Constructors

```
public JLabel()
  //Creates a JLabel object with no text or icon.
public JLabel(String str)
  //Creates a JLabel object with the left-aligned text specified
  //by str.
public JLabel(String str, int align)
  //Creates a JLabel object with the text specified by str.
  //The value of align can be any one of the following:
  //    SwingConstants.LEFT,
  //    SwingConstants.RIGHT,
  //    SwingConstants.CENTER
  //These constants are defined in the class SwingConstants.
public JLabel(Icon icon)
  //Constructs a JLabel with an icon.
public JLabel(Icon icon, int align)
  //Creates a JLabel object with an icon.
  //The value of align can be any one of the following:
  //    SwingConstants.LEFT,
  //    SwingConstants.RIGHT,
  //    SwingConstants.CENTER
  //These constants are defined in the class SwingConstants.
public JLabel(String t, Icon icon, int align)
  //Constructs a JLabel with both text and an icon.
  //The icon is to the left of the text.
  //The value of align can be any one of the following:
  //    SwingConstants.LEFT,
  //    SwingConstants.RIGHT,
  //    SwingConstants.CENTER
  //These constants are defined in the class SwingConstants.
```

Methods

```
public Icon getIcon()
  //Returns the graphic image (glyph, icon) that
  //the label displays.
public void setIconTextGap(int IconTextGap)
  //Sets the gap between the icon and the text properties if
  //both are set.
public int getIconTextGap()
  //Returns the gap between the icon and the text displayed
  //in this label.
public void setVerticalAlignment(int Align)
  //Sets the label's alignment along the Y-axis.
  //The value of align can be any one of the following:
  //    SwingConstants.TOP,
  //    SwingConstants.CENTER (default),
  //    SwingConstants.BOTTOM.
  //These constants are defined in the class SwingConstants.
public int getVerticalAlignment()
  //Returns the label's alignment along the Y-axis.
public void setHorizontalAlignment(int Align)
  //Sets the label's alignment along the X-axis.
  //The value of align can be any one of the following:
  //    SwingConstants.LEFT,
  //    SwingConstants.RIGHT,
  //    SwingConstants.CENTER
  //These constants are defined in the class SwingConstants.
public int getHorizontalAlignment()
  //Returns the label's alignment along the X-axis.
public void setVerticalTextPosition(int TextPos)
  //Sets the vertical text position relative to the
  //icon. The value of align can be any one of the following:
  //    SwingConstants.TOP,
  //    SwingConstants.CENTER (default),
  //    SwingConstants.BOTTOM.
  //These constants are defined in the class SwingConstants.
public int getVerticalTextPosition()
  //Returns the vertical text position.
public void setHorizontalTextPosition(int TextPos)
  //Sets the horizontal text position relative to the
  //icon. The value of align can be any one of the following:
```

```
  //    SwingConstants.LEFT,
  //    SwingConstants.RIGHT,
  //    SwingConstants.CENTER
  //These constants are defined in the class SwingConstants.
public int getHorizontalTextPosition()
  //Returns the horizontal text position.
```

CLASS: JList (PACKAGE javax.swing)

Constructors

```
public JList()
  //Creates a JList with no items to select.
public JList(ListModel dataModel)
  //Constructs a JList that displays the elements in the
  //specified, non-null model.
public JList(Object[] o)
  //Creates a JList for selection that displays elements
  //in the object array provided as an input parameter.
public JList(Vector<?> listData)
  //Constructs a JList that displays the elements in the
  //specified Vector.
```

Methods

```
public void setSelectionBackground(Color sbColor)
  //Sets the color of the background of a
  //selected item.
public void addListSelectionListener(ListSelectionListener lsl)
  //Adds a listener class to take action when an
  //item in the list is selected.
public void removeListSelectionListener
                                (ListSelectionListener lsnr)
  //Removes a listener of the list that is notified every
  //time change to the selection occurs.
public void setSelectionMode(int selectionMode)
  //Sets the selection mode. The selectionMode values are:
  //ListSelectionModel.SINGLE_SELECTION - Only one list index
  //can be selected.
  //ListSelectionModel.SINGLE_INTERVAL_SELECTION - One contiguous
  //index interval can be selected.
  //ListSelectionModel.MULTIPLE_INTERVAL_SELECTION - There's no
  //restriction on what can be selected (default).
public int getSelectionMode()
  //Returns the current selectionMode.
public int getAnchorSelectionIndex()
  //Returns the first index argument from the most recent
```

```
    //addSelectionModel orsetSelectionInterval method invocation.
  public int getLeadSelectionIndex()
    //Returns the second index argument from the most recent
    //addSelectionInterval or setSelectionInterval method
    //invocation.
  public int getMinSelectionIndex()
    //Returns the smallest selected index.
  public int getMaxSelectionIndex()
    //Returns the largest selected index.
  public boolean isSelectedIndex(int index)
    //Returns true if the specified index is selected.
  public boolean isSelectionEmpty()
    //Returns true if the selected list is empty.
  public void clearSelection()
    //Clears the selection list.
  public int[] getSelectedIndices()
    //Returns an array of all of the selected indices in the
    //increasing order.
  public void setSelectedIndex(int index)
    //Selects a single item.
  public void setSelectedIndices(int[] indices)
    //Selects a set of items.
  public Object[] getSelectedValues()
    //Returns an array of the values for the selected items.
  public int getSelectedIndex()
    //When an item in the list is selected, this method returns
    //the index of the first item (0 to number of items - 1).
    //Returns -1 if nothing is selected.
  public Object getSelectedValue()
    //Returns the first selected value, or null if there is no
    //selected item.
```

Class: JMenu (Package javax.swing)

Constructors

```
  public JMenu()
    //Creates a JMenu object with no text.
  public JMenu(String t)
    //Creates a JMenu object with the specified text.
```

Methods

```
  public boolean isSelected()
    //Returns true if the menu is currently selected
    //(highlighted).
```

```
public void setSelected(boolean b)
  //Sets the menu selected or not selected as specified.
public boolean isPopupMenuVisible()
  //Returns the visibility status of a popup menu. The value
  //returned is true if the menu's popup window is visible.
public void setPopupMenuVisible(boolean b)
  //Sets the visibility status of a popup menu.
protected Point getPopupMenuOrigin()
  //Returns the origin for the JMenu's popup menu.
public int getDelay()
  //Returns the time, in milliseconds, delay before submenus
  //are displayed.
public void setDelay(int dtime)
  //Sets the time, in milliseconds, delay before submenus
  //are displayed.
public void setMenuLocation(int x, int y)
  //Sets the location of the popup component.
public JMenuItem add(JMenuItem menuItem)
  //Adds a menu item to the end of this menu.
public Component add(Component c)
  //Adds a component to the end of this menu.
public Component add(Component c, int index)
  //Adds a component at the specified position of this menu.
public JMenuItem add(String s)
  //Creates a new menu item with the specified text and adds
  //it to the end of this menu.
public void addSeparator()
  //Adds a new separator to the end of the menu.
public void insert(String s, int pos)
  //Creates and inserts a new menu item with the specified text
  //at a given position.
public JMenuItem insert(JMenuItem mi, int pos)
  //Inserts the specified JMenuItem at the given position of
  //this menu.
public void insertSeparator(int index)
  //Inserts a menu separator at the specified position.
public JMenuItem getItem(int pos)
  //Returns the JMenuItem at the specified position.
public int getItemCount()
  //Returns the number of items on this menu. The count
  //includes separators.
public void remove(JMenuItem item)
  //Deletes the specified menu item from this menu.
public void remove(int pos)
  //Deletes the menu item at the specified index from
  //this menu.
public void remove(Component c)
  //Deletes the specified component from this menu.
```

```
public void removeAll()
  //Deletes all the menu items from this menu.
public int getMenuComponentCount()
  //Returns the number of components on this menu.
public Component getMenuComponent(int n)
  //Returns the component at the specified position
public Component[] getMenuComponents()
  //Returns an array of components of the menu's subcomponents
  //including separators.
public boolean isTopLevelMenu()
  //Returns true if the menu is a top-level menu.
public boolean isMenuComponent(Component c)
  //Returns true if the specified component is part of this
  //menu's submenu hierarchy.
public JPopupMenu getPopupMenu()
  //Returns the popup menu of this menu. If no popup menu exists,
  //it will create one.
public void addMenuListener(MenuListener l)
  //Adds a listener for the menu events.
public void removeMenuListener(MenuListener l)
  //Removes a listener for the menu events.
public void doClick(int pressTime)
  //Programmatically performs a "click". This method overrides
  //the one in the class AbstractButton.
```

Class: JMenuBar (Package javax.swing)

Constructor

```
public JMenuBar()
  //Creates a new menu bar.
```

Methods

```
public JMenu add(JMenu c)
  //Appends the specified menu to the end of the menu bar.
public JMenu getMenu(int index)
  //Returns the menu at the specified position in the menu bar.
public int getMenuCount()
  //Returns the number of items in the menu bar.
public void setHelpMenu(JMenu menu)
  //Sets the help menu for the menu bar.
public JMenu getHelpMenu()
  //Gets the help menu for the menu bar.
public int getComponentIndex(Component c)
  //Returns the index of the specified component.
```

```
public void setSelected(Component sel)
  //Sets the currently selected component.
public boolean isSelected()
  //Returns true if a component of the menu bar is selected.
public boolean isBorderPainted()
  //Returns true if the menu bar's border is painted.
public void setBorderPainted(boolean b)
  //Border is painted if b is true.
protected void paintBorder(Graphics g)
  //Method that paints the menu bar's border if BorderPainted
  //property is true.
public void setMargin(Insets m)
  //Sets the margin between the menu bar's border and its menus.
  //To get the default margin, set it to null.
public Insets getMargin()
  //Returns the margin between the menu bar's border and its
  //menus.
```

CLASS: JMenuItem (PACKAGE javax.swing)

Constructors

```
public JMenuItem()
  //Constructs a JMenuItem with no set text or icon.
public JMenuItem(Icon icon)
  //Constructs a JMenuItem with the specified icon.
public JMenuItem(String t)
  //Constructs a JMenuItem with the specified text.
public JMenuItem(String t, Icon icon)
  //Constructs a JMenuItem with the specified text and
  //icon.
public JMenuItem(String t, int mnemonic)
  //Constructs a JMenuItem with the specified text and
  //keyboard mnemonic.
```

Methods

```
protected void init(String t, Icon icon)
  //Initializes the menu item with the specified text and icon.
public void setEnabled(boolean b)
  //Enables or disables this menu item as specified.
public void setAccelerator(KeyStroke keyStroke)
  //Sets the accelerator key, that is, the key combination
  //that invokes the menu item's action listeners without
  //navigating the menu hierarchy using a mouse.
```

```
public KeyStroke getAccelerator()
  //Returns the KeyStroke which serves as an accelerator
  //for the menu item.
```

Class: JRadioButton (Package javax.swing)

Constructors

```
public JRadioButton()
  //Creates an initially unselected radio button
  //with no label and no icon.
public JRadioButton(Icon icon)
  //Creates an initially unselected radio button
  //with the specified icon and no label.
public JRadioButton(Icon icon, boolean selected)
  //Creates a radio button with the specified
  //image and selection state, but without label.
public JRadioButton(String text)
  //Creates an unselected radio button with the
  //specified label.
public JRadioButton(String text, boolean selected)
  //Creates a radio button with the specified
  //label and selection state.
public JRadioButton(String text, Icon icon)
  //Creates a radio button with the specified image and
  //with specified label.
public JRadioButton(String text, Icon icon, boolean selected)
//Creates a radio button with the specified
//image and selection state, and with the specified text.
```

Methods

```
public String getText()
  //Returns the text contained in the radio button.
public void setText(String str)
  //Sets the text of the radio button to the string
  //specified by str.
public boolean isSelected()
  //Returns true if the radio button is selected, false
  //otherwise.
public void setSelected(boolean b)
  //Sets the state of the radio button to b. This method does
  //not trigger an actionEvent. For that, call doClick.
public void addActionListener(ActionListener obj)
  //Registers a listener object to the radio button
  //object.
public void doClick()
  //Programmatically performs a "click."
```

```
public int getVerticalTextPosition()
  //Returns the vertical position of the text.
  //Returns one of the following constant values:
  //   SwingConstants.CENTER (the default)
  //   SwingConstants.TOP
  //   SwingConstants.BOTTOM
public void setVerticalTextPosition(int pos)
  //Sets the vertical position of the text.
  //Possible values are:
  //   SwingConstants.CENTER (the default)
  //   SwingConstants.TOP
  //   SwingConstants.BOTTOM
public int getHorizontalTextPosition()
  //Returns the horizontal position of the text.
  //Returns one of the following constant values:
  //   SwingConstants.RIGHT
  //   SwingConstants.LEFT
  //   SwingConstants.CENTER
  //   SwingConstants.LEADING
  //   SwingConstants.TRAILING (the default)
public void setHorizontalTextPosition(int pos)
  //Sets the horizontal position of the text.
  //Possible values are:
  //   SwingConstants.RIGHT
  //   SwingConstants.LEFT
  //   SwingConstants.CENTER
  //   SwingConstants.LEADING
  //   SwingConstants.TRAILING (the default)
public void setActionCommand(String actionCommand)
  //Sets the action command.
public String getActionCommand()
  //Returns the action command.
public int getMnemonic()
  //Returns the keyboard mnemonic. The mnemonic is the key which
  //when combined with the meta key (usually Alt) will "click"
  //this radio button if focus is within the radio button's
  //ancestor window.
public void setMnemonic(int mnemonic)
  //Sets the keyboard mnemonic. The mnemonic is the key which
  //when combined with the meta key (usually Alt) will "click"
  //this radio button if focus is within the radio button's
  //ancestor window.
public void setEnabled(boolean b)
  //Sets the radio button enabled if b is true;
  //disabled if b is false.
```

CLASS: JTextArea (PACKAGE javax.swing)

Constructors

```
public JTextArea()
  //Creates a JTextArea instance with 0 number of rows and
  //0 number of columns.
public JTextArea(int r, int c)
  //Creates a JTextArea instance with r number of rows and
  //c number of columns.
public JTextArea(String t)
  //Creates a JTextArea instance with 0 number of rows,
  //0 number of columns and initial text t.
public JTextArea(String t, int r, int c)
  //Creates a JTextArea instance with r number of rows,
  //c number of columns and initial text t.
```

Methods

```
public void setColumns(int c)
  //Sets the number of columns to c.
public int getColumns()
  //Returns the number of columns.
public void setRows(int r)
  //Sets the number of rows to r.
public int getRows()
  //Returns the number of rows.
public void append(String t)
  //Concatenates the text already in the JTextArea with t.
public void setLineWrap(boolean b)
  //If b is true, lines are wrapped.
public boolean getLineWrap()
  //Returns true if line wrapping is set.
public void setWrapStyleWord(boolean b)
  //If b is true, lines are wrapped at the word boundaries.
  //If b is false, the word boundaries are not considered.
public void getWrapStyleWord()
  //Returns true if lines are wrapped at the word boundaries.
  //Returns false if the word boundaries are not considered.
public void setTabSize(int c)
  //Sets tab stops every c columns.
public int getTabSize()
  //Returns the number of characters used to expand a tab stop.
public void setText(String t)
  //Changes the text of the text area to t.
public String getText()
  //Returns the text contained in the text area.
public void setEditable(boolean b)
  //If b is false, user cannot type in the text area. In this
  //case, the text area is used as a tool to display the result.
public int getLineCount()
  //Determines the number of lines contained in the text area.
public void insert(String str, int p)
  //Inserts the specified text at a given position.
public void replaceRange(String str, int start, int end)
  //Replaces text from the specified start to end position
  //with the new text given.
public void setFont(Font font)
  //Sets the current font.
```

Class: JTextField (Package javax.swing)

Constructors

```
public JTextField()
  //Creates a JTextField object with 0 columns.
  //Initial text is set to null.
public JTextField(int columns)
  //Creates a JTextField object with the number of columns
  //specified by columns.
public JTextField(String str)
  //Creates a JTextField object with the text
  //specified by str
public JTextField(String str, int columns)
  //Creates a JTextField object with the text specified by
  //str and sets the size of the text field
```

Methods

```
public void setColumns(int c)
  //Sets the number of columns to c.
public int getColumns()
  //Returns the number of columns.
public void setText(String str)
  //Sets the text of the text field to the string
  //specified by str.
public String getText()
  //Returns the text contained in the text field.
public void setEditable(boolean b)
  //If the value of the Boolean variable b is false, user
  //cannot type in the text field.
  //In this case, the text field is used as a tool to
  //display the result.
public void addActionListener(ActionListener obj)
  //Registers a listener object to a JTextField.
public void removeActionListener(ActionListener obj)
  //Removes a registered listener object.
```

```
public int getHorizontalAlignment()
  //Returns the horizontal alignment of the text.
  //Valid values are:
  //   JTextField.LEFT
  //   JTextField.CENTER
  //   JTextField.RIGHT
  //   JTextField.LEADING
  //   JTextField.TRAILING
public void setHorizontalAlignment(int a)
  //Sets the horizontal alignment of the text.
  //Valid values are:
  //   JTextField.LEFT
  //   JTextField.CENTER
  //   JTextField.RIGHT
  //   JTextField.LEADING
  //   JTextField.TRAILING
public void setFont(Font font)
  //Sets the current font.
```

Class: LinkedList<E> (Package java.util)

Constructors

```
public LinkedList()
  //Constructs an empty list.
public LinkedList(Collection<? extends E> coll) throws
                 NullPointerException
  //Constructs a list containing the elements of the collection
  //specified by the parameter coll, in the order they are
  //returned by the collection's iterator.
```

Methods

```
public boolean add(E obj)
  //Appends the element specified by the parameter obj to the
  //end of this list. It returns true if this collection changed
  // as a result of the call; false otherwise.
public void add(int index, E elem) throws
               IndexOutOfBoundsException
  //Inserts the element specified by the parameter elem at
  //the position index in this list. The element currently at
  //that position (if any) and any subsequent elements are
  //shifted to the right.
public boolean addAll(Collection<? extends E> coll) throws
                     NullPointerException
  //Appends all of the elements in the collection coll to the
  //end of this list, in the order that they are returned by the
  //the iterator of coll. It returns true if this list is
  //changed as a result of the call. If the collection coll
  //is modified while this operation is in progress, then the
  //behavior of this operation is undefined. It returns true if
  //this list changed as a result of the call.
public boolean addAll(int index, Collection<? extends E> coll)
                           throws NullPointerException,
                                  IndexOutOfBoundsException
  //Inserts all of the elements of the collection coll into this
  //list, starting at the position index. The element currently
  //at that position (if any) and any subsequent elements are
  //shifted to the right (increases their indices). The new
  //elements appear in the list in the order that they are
  //returned by the iterator of coll. It returns true if this
  //list changed as a result of the call.
public void addFirst(E obj)
  //Inserts the element specified by the parameter obj at the
  //beginning of this list.
public void addLast(E obj)
  //Appends the element specified by the parameter obj to the
  //end of this list.
public void clear()
  //Removes all of the elements from this list.
public Object clone()
  //Returns a shallow copy of this LinkedList. (The elements
  //themselves are not cloned.)
public boolean contains(Object obj)
  //Returns true if this list contains the element specified
  //by the parameter obj. Otherwise it returns false.
public E element() throws NoSuchElementException
  //Retrieves, but does not remove, the first element of this
  //list.
public E get(int index) throws IndexOutOfBoundsException
  //Returns the element at the position specified by index in
  //this list.
public E getFirst() throws NoSuchElementException
  //Returns the first element in this list.
public E getLast() throws NoSuchElementException
  //Returns the last element in this list.
public int indexOf(Object obj)
  //Returns the index in this list of the first occurrence of
  //the element specified by the parameter obj. If this list
  //does not contain the specified element, it returns -1.
public int lastIndexOf(Object obj)
  //Returns the index, in this list, of the last occurrence of
  //the element specified by the parameter obj. If this list
  //does not contain the specified element, it returns -1.
public ListIterator<E> listIterator(int index) throws
                              IndexOutOfBoundsException
  //Returns a list-iterator of the elements in this list (in
  //proper sequence), starting at the position specified by
  //index in the list.
```

```
public boolean offer(E obj)
  //Adds the element specified by the parameter obj as the tail
  //(last element) of this list. It returns true if it was
  //possible to add the element to this list; false otherwise.
public E peek()
  //Retrieves, but does not remove, the first element of this
  //list. If this list is empty it returns null.
public E poll()
  //Retrieves and removes the first element of this list. If
  //this list is empty it returns null.
public E remove(int index) throws IndexOutOfBoundsException
  //Removes the element at the position specified by index in
  //this list. Any subsequent elements are shifted to the left
  //(subtracts one from their indices). It also returns the
  //element that was removed from the list.
public E remove() throws NoSuchElementException
  //Retrieves and removes the first element of this list.
public boolean remove(Object obj)
  //Removes the first occurrence of the element specified by
  //the parameter obj in this list. If the list does not contain
  //the element, it is unchanged. It returns true if the list
  //contained the element obj.
public E removeFirst() throws NoSuchElementException
  //Removes and returns the first element from this list.
public E removeLast() throws NoSuchElementException
  //Removes and returns the last element from this list.
public E set(int index, E element) throws
                             IndexOutOfBoundsException
  //Replaces the element at the position specified by index, in
  //this list, with the element specified by the parameter
  //element. It also returns the element previously at the
  //position index.
public int size()
  //Returns the number of elements in this list.
public Object[] toArray()
  //Returns an array containing all of the elements in this
  //list in the correct order.
```

Interface: ListIterator<E> (PACKAGE java.util)

Methods

```
void add(E obj) throws UnsupportedOperationException,
                       ClassCastException,
                       IllegalArgumentException
  //Inserts the element specified by the parameter obj into the
  //list. It is an optional operation. The element is inserted
  //immediately before the next element that would be returned
  //by next, if any, and after the next element that would be
  //returned by previous, if any. (If the list was empty, the
  //new element becomes the sole element on the list.) The new
  //element is inserted before the implicit cursor: a
  //subsequent call to next would be unaffected, and a subsequent
  //call to previous would return the new element.
boolean hasNext()
  //Returns true if this list iterator has more elements when
  //traversing the list in the forward direction.
boolean hasPrevious()
  //Returns true if this list iterator has more elements when
  //traversing the list in the reverse direction.
E next() throws NoSuchElementException
  //Returns the next element in the list. This method may be
  //called repeatedly to iterate through the list, or it may be
  //intermixed with calls to previous to go back and forth.
int nextIndex()
  //Returns the index of the element that would be returned by
  //a subsequent call to next. It returns list-size if the list
  //iterator is at the end of the list.
E previous() throws NoSuchElementException
  //Returns the previous element in the list. This method may
  //be called repeatedly to iterate through the list backwards,
  //or it may be intermixed with calls to next to go back and
  //forth.
int previousIndex()
  //Returns the index of the element that would be returned by
  //a subsequent call to previous. If the list iterator is at
  //the beginning of the list, it returns -1.
void remove() throws UnsupportedOperationException,
                     IllegalArgumentException
  //Removes from the list the last element that was returned by
  //the method next or previous (optional operation). This
  //method can be called once per call to next or previous. It
  //can be made only if ListIterator.add has not been called
  //after the last call to next or previous.
void set(E obj) throws UnsupportedOperationException,
              ClassCastException, IllegalArgumentException,
              IndexOutOfBoundsException, IllegalStateException
  //Replaces the last element returned by the method next or
  //previous with the element specified by the parameter obj. It
  //is an optional operation. This method can be called only if
  //neither ListIterator.remove nor ListIterator.add have been
  //called after the last call to next or previous.
```

Class: Long (Package java.lang)

Named Constants

```
public static final long MAX_VALUE = 9223372036854775807;
public static final long MIN_VALUE = -9223372036854775808;
```

Constructors

```
public Long(long num)
  //Creates a Long object initialized to the value specified
  //by num.
public Long(String str) throws NumberFormatException
  //Creates a Long object initialized to the value specified
  //by the number contained in str.
```

Methods

```
public byte byteValue()
  //Returns the value of the object as a byte value.
public short shortValue()
  //Returns the value of the object as a short value.
public int intValue()
  //Returns the value of the object as an int value.
public long longValue()
  //Returns the value of the object as a long value.
public double doubleValue()
  //Returns the value of the object as a double value.
public float floatValue()
  //Returns the value of the object as a float value.
public int hashCode()
  //Returns the hash code of the object.
public boolean equals(Object obj)
  //Returns true if the value of this object is equal
  //to the value of the object specified by obj;
  //otherwise returns false.
public static long parseLong(String str) throws
                                    NumberFormatException
  //Returns the value of the number contained in str.
public static long parseLong(String str, int base) throws
                                    NumberFormatException
  //Returns the value, in the radix base, of the long value
  //contained in str.
public static String toBinaryString(long num)
  //Returns the string representation of num in binary (base 2).
public static String toHexString(long num)
  //Returns the string representation of num in
```

E

```
  //hexadecimal (base 16).
public static String toOctalString(long num)
  //Returns the string representation of num in octal (base 8).
public String toString()
  //Returns the long value of the object as a string.
public static String toString(long num)
  //Returns the value of num as a string.
public static String toString(long num, int base)
  //Returns the value of num, in the radix base, as a string.
public static Long valueOf(String str) throws
                                        NumberFormatException
  //Returns a Long object initialized to the value
  //specified by str.
public static Long valueOf(String str, int base) throws
                                        NumberFormatException
  //Returns a Long object initialized to the value,
  //in radix base, specified by str.
```

Interface: Map<K, V> (Package java.util)

Methods

```
public void clear() throws UnsupportedOperationException
  //Removes all the mappings from this map. It is an optional
  //operation.
public boolean containsKey(Object key) throws
                          ClassCastException, NullPointerException
  //Returns true if this map contains a mapping for the
  //specified key.
public boolean containsValue(Object†value) throws
                          ClassCastException, NullPointerException
  //Returns true if this map maps one or more keys to the
  //value specified by the parameter value.
public boolean equals(Object obj)
  //Compares the object specified by the parameter obj with this
  //map for equality. Returns true if the given object is also
  //a map and the two Maps represent the same mappings.
public V get(Object key) throws ClassCastException,
                                NullPointerException
  //Returns the value to which this map maps the key specified
  //by the parameter key. Returns null if the map contains no
  //mapping for this key. Note that a return value of null does
  //not necessarily indicate that the map contains no mapping
  //for the key; it's also possible that the map explicitly
  //maps the key to null.
boolean isEmpty()
```

```
    //Returns true if this map does not contains any key/value
    //mappings.
public Set<K> keySet()
    //This method returns the keys contained in the map as a Set
    //object.
public V put(K key, V value) throws
              UnsupportedOperationException,
              ClassCastException, IllegalArgumentException,
              NullPointerException
    //This method associates the value specified by the parameter
    //value with the key specified by the parameter key in this
    //map. It is an optional operation. If the map previously
    //contained a mapping for this key, the old value is replaced
    //by the specified value. This method also returns the
    //previous value associated with specified key, or null if
    //there was no mapping for key.
public V remove(Object key) throws
                 UnsupportedOperationException,
                 ClassCastException, NullPointerException
    //Removes the mapping for the key specified by the parameter
    //key from this map if it is present. It is an optional
    //operation. This method also returns the previous value
    //associated with specified key, or null if there was no
    //mapping for the key.
public int size()
    //Returns the number of key/value mappings in this map. If the
    //number of elements in the map is more than Integer.MAX_VALUE
    //elements, it returns Integer.MAX_VALUE.
public Collection<V> values()
    //Returns a collection view of the values contained in this
    //map.
```

Class: Math (Package java.lang)

Methods

```
public static int abs(int x)
    //Returns the absolute value of x.
public static long abs(long x)
    //Returns the absolute value of x.
public static double abs(double x)
    //Returns the absolute value of x.
public static float abs(float x)
    //Returns the absolute value of x.
public static double cbrt(double x)
    //Returns the cube root of x.
public static double ceil(double x)
```

```
  //Returns a value of the type double, which is the
  //smallest integer value that is not less than x.
public static double exp(double x)
  //Returns e^x, where e is approximately 2.7182818284590455.
public static double floor(double x)
  //Returns a value of the type double, which is the
  //largest integer value less than x.
public static double hypot(double x, double y)
  //Returns sqrt(x^2 + y^2) without intermediate overflow
  //or underflow.
public static double log(double x) throws ArithmeticException
  //Returns a value of the type double, which is
  //the natural logarithm of x.
public static double log10(double x)
  //Returns a value of the type double, which is the common
  //logarithm (base 10) of x.
public static int max(int x, int y)
  //Returns the larger of x and y.
public static long max(long x, long y)
  //Returns the larger of x and y.
public static double max(double x, double y)
  //Returns the larger of x and y.
public static float max(float x, float y)
  //Returns the larger of x and y.
public static int min(int x, int y)
  //Returns the smaller of x and y.
public static long min(long x, long y)
  //Returns the smaller of x and y.
public static double min(double x, double y)
  //Returns the smaller of x and y.
public static float min(float x, float y)
  //Returns the smaller of x and y.
public static double pow(double x, double y)
  //Returns x^y.
public static double random()
  //Returns a random number between 0.0 and 1.0.
public static int round(float x)
  //Returns the value which is the integer closest to x.
public static long round(double x)
  //Returns the value which is the integer closest to x.
public static float signum(float x)
  //Returns the signum function of x.
  //Returns 0 if x == 0.
  //Returns 1.0f if x > 0.
  //Returns -1.0f if x < 0.
public static double sqrt(double x)
  //Returns the positive square root of x; x must be
  //nonnegative.
public static double cos(double x)
```

```
  //Returns the cosine of x measured in radians.
public static double sin(double x)
  //Returns the sine of x measured in radians.
public static double tan(double x)
  //Returns the tangent of x measured in radians.
public static double acos(double x)
  //Returns the arc cosine of x measured in radians.
public static double asin(double x)
  //Returns the arc sine of x measured in radians.
public static double atan(double x)
  //Returns the arc tangent of x measured in radians.
public static double cosh(double x)
  //Returns the hyperbolic cosine of x.
public static double sinh(double x)
  //Returns the hyperbolic sine of x.
public static double tanh(double x)
  //Returns the hyperbolic tangent of x.
```

Class: Point (Package java.awt)

Variables

```
public int x;
public int y;
```

Constructors

```
public Point()
  //Creates a Point object and initializes x and y to 0.
public Point(int a, int b)
  //Creates a Point object and initializes x to a and y to b.
public Point(Point p)
  //Creates a Point object and initializes it using the object p.
```

Methods

```
public boolean equals(Object ob)
  //Returns true if this object is same as the object ob;
  //otherwise returns false.
public Point getLocation()
  //Returns the location of the object.
public void move(int a, int b)
  //Moves the object at the location (a, b).
public void setLocation(int a, int b)
  //Sets the location of the object at (a, b).
public void setLocation(Point p)
  //Sets the location of the object at p.
```

```
public int hashCode()
  //Returns the hash code of the object.
public String toString()
   //Returns a string representation of the object.
```

CLASS: PrintWriter (PACKAGE java.io)

Constructors

```
public PrintWriter(File file)
                        throws FileNotFoundException
  //Creates a new PrintWriter with the specified file.
  //No automatic line flushing.
public PrintWriter(File file, String csn)
                        throws FileNotFoundException,
                              UnsupportedEncodingException
  //Creates a new PrintWriterwith the specified file
  //and charset. No automatic line flushing.
public PrintWriter(String fileName)
                        throws FileNotFoundException
  //Creates a new PrintWriter with the specified filename.
  //No automatic line flushing.
public PrintWriter(String fileName, String csn)
                        throws FileNotFoundException,
                              UnsupportedEncodingException
  //Creates a new PrintWriter with the specified filename
  //and charset. No automatic line flushing.
public PrintWriter(Writer out)
  //Creates a PrintWriter object. No automatic line flushing.
public PrintWriter(Writer out, boolean af)
  //Create a PrintWriter object.
  //boolean af determines whether or not automatic
  //line flushing is enabled.
public PrintWriter(OutputStream out)
  //Creates a PrintWriter object from an OutputStream.
  //No automatic line flushing.
public PrintWriter(OutputStream out, boolean autoFlush)
  //Create a PrintWriter object from an existing OutputStream.
```

Methods

```
public void flush()
  //Flushes the stream.
public void close()
  //Closes the stream.
public boolean checkError()
```

```
  //Flushes the stream and checks its error state.
protected void setError()
  //Sets an error state.
public void write(int c)
  //Writes a single character.
public void write(char[] b, int offset, int length)
  //Writes a part of an array of characters.
public void write(char[] buf)
  //Writes an array of characters.
public void write(String s, int offset, int length)
  //Writes a part of a string.
public void write(String s)
  //Writes a string.
public void print(boolean b)
  //Prints a boolean value.
public void print(char c)
  //Prints a character.
public void print(int i)
  //Prints an integer.
public void print(long l)
  //Prints a long integer.
public void print(float f)
  //Prints a floating-point number.
public void print(double d)
  //Prints a double-precision floating-point number.
public void print(char[] s)
  //Prints an array of characters.
public void print(String s)
  //Prints a string.
public void print(Object obj)
  //Prints an object.
public void println()
  //Terminates the current line by printing the line
  //separator string.
public void println(boolean x)
  //Prints a boolean value followed by the line separator string.
public void println(char x)
  //Prints a character followed by the line separator string.
public void println(int x)
  //Prints an integer followed by the line separator string.
public void println(long x)
  //Prints a long integer followed by the line separator string.
public void println(float x)
  //Prints a floating-point number followed by the line
  //separator string.
public void println(double x)
  //Prints a double-precision floating-point number
  //followed by the line separator string.
public void println(char[] x)
```

```
    //Prints an array of characters followed by the line
    //separator string.
  public void println(String x)
    //Prints a String followed by the line separator string.
  public void println(Object x)
    //Prints an Object followed by the line separator string.
```

CLASS: Scanner (PACKAGE java.util)

Constructors

```
  public Scanner(InputStream source)
    //Constructs a new Scanner that produces values scanned
    //from the specified input stream. Bytes from the stream
    //are converted into characters using the underlying
    //platform's default charset.
    //The parameter source specified the input stream to be
    //scanned.
  public Scanner(File source) throws FileNotFoundException
    //Constructs a new Scanner that produces values scanned
    //from the specified file.
    //The parameter source specified the input stream to be
    //scanned.
  public Scanner(File source, String charsetName)
                                  throws FileNotFoundException
    //Constructs a new Scanner that produces values scanned
    //from the specified file. Bytes from the file are
    //converted into characters using the specified charset.
    //The parameter source specified the input stream to be
    //scanned and charsetName specifies the encoding type
    //used to convert bytes from the stream into characters
    //to be scanned.
  public Scanner(String source)
    //Constructs a new Scanner that produces values scanned
    //from the specified string.
    //The parameter source specifies string to be scanned.
```

Methods

```
  public void close()
    //Closes this scanner.
  public Pattern delimiter()
    //Returns the Pattern this Scanner is currently using to
    //match delimiters.
  public Scanner useDelimiter(Pattern pattern)
    //Sets this scanner's delimiting pattern to the specified
    //pattern.
    //The parameter pattern specifies the delimiting pattern.
  public Scanner useDelimiter(String pattern)
    //Sets this scanner's delimiting pattern to a pattern
  //constructed from the specified String.
  //The parameter pattern specifies a delimiting pattern.
public String toString()
 //Returns the string representation of this Scanner.
public boolean hasNext()
  //Returns true if this scanner has another token in
  //its input.
  //If this scanner is closed it throws IllegalStateException.
public String next()
  //Finds and returns the next complete token from
  //this scanner.
  //It throws NoSuchElementException if no more tokens are
  //available and IllegalStateException if this scanner is
  //closed.
public boolean hasNext(String pattern)
  //Returns true if the next token matches the pattern
  //constructed from the specified string. The scanner does
  //not advance past any input.
  //The parameter pattern specifies the pattern to be scanned.
  //If this scanner is closed it throws IllegalStateException.
public String next(String pattern)
  //Returns the next token if it matches the pattern
  //constructed from the specified string. If the match is
  //successful, the scanner advances past the input that
  //matched the pattern.
  //The parameter pattern specifies the pattern to be scanned.
  //It throws NoSuchElementException if no more tokens are
  //available and IllegalStateException if this scanner is
  //closed.
public boolean hasNextLine()
  //Returns true if there is another line in the input of
  //this scanner. This method may block while waiting for
  //input. The scanner does not advance past any input.
  //If this scanner is closed it throws IllegalStateException.
public String nextLine()
  //Advances this scanner past the current line and returns
  //the input that was skipped. This method returns the rest
  //of the current line, excluding any line separator at the
  //end. The position is set to the beginning of the next
  //line. It throws NoSuchElementException if no more
  //tokens are available and IllegalStateException if this
  //scanner is closed.
public String findInLine(String pattern)
  //Attempts to find the next occurrence of a pattern
  //constructed from the specified string, ignoring
  //delimiters. The parameter pattern specifies the pattern
  //to search for. If this scanner is closed it throws
  //IllegalStateException.
public Scanner skip(String pattern)
  //Skips input that matches a pattern constructed from the
```

```
    //specified string.
    //If this scanner is closed it throws IllegalStateException.
  public boolean hasNextBoolean()
    //Returns true if the next token in this scanner's input
    //can be interpreted as a boolean value using a case
    //insensitive pattern created from the string
    //"true or false". The scanner does not advance past the
    //input that matched.
    //If this scanner is closed it throws IllegalStateException.
  public boolean nextBoolean()
    //Scans the next token of the input into a boolean value
    //and returns that value.
    //Returns the boolean scanned from the input.
    //It throws InputMismatchException if the next token is not
    //a valid boolean, NoSuchElementException if input is
    //exhausted, and IllegalStateException if this scanner is
    //closed.
    //If the match is successful, the scanner advances past the
    //input that matched.
  public byte nextByte()
    //Scans the next token of the input as a byte.
    //Returns the byte scanned from the input.
    //It throws InputMismatchException if the next token does not
    //match the Integer regular expression, or is out of range,
    //NoSuchElementException if input is exhausted, and
    //IllegalStateException if this scanner is closed.
  public boolean hasNextShort()
    //Returns true if the next token in this scanner's input
    //can be interpreted as a short value in the default radix
    //using the nextShort() method. The scanner does not advance
    //past any input.
    //If this scanner is closed it throws IllegalStateException.
  public short nextShort()
    //Scans the next token of the input as a short.
    //Returns the short scanned from the input.
    //It throws InputMismatchException if the next token does
    //not match the Integer regular expression, or is out of
    //range, NoSuchElementException if input is exhausted, and
    //IllegalStateException if this scanner is closed.
  public boolean hasNextInt()
    //Returns true if the next token in this scanner's input
    //can be interpreted as an int value in the default radix
    //using the nextInt() method.
    //If this scanner is closed it throws IllegalStateException.
  public int nextInt()
    //Scans the next token of the input as an int.
    //Returns the int scanned from the input.
    //It throws InputMismatchException if the next token does
    //not match the Integer regular expression, or is out of
  //range, NoSuchElementException if input is exhausted, and
  //IllegalStateException if this scanner is closed.
public boolean hasNextLong()
  //Returns true if the next token in this scanner's input
  //can be interpreted as a long value in the default radix
  //using the nextLong() method. The scanner does not advance
  //past any input.
  //If this scanner is closed it throws IllegalStateException.
public long nextLong()
  //Scans the next token of the input as a long.
  //Returns the long scanned from the input.
  //It throws InputMismatchException if the next token does not
  //match the Integer regular expression, or is out of range,
  //NoSuchElementException if input is exhausted, and
  //IllegalStateException if this scanner is closed.
public boolean hasNextFloat()
  //Returns true if the next token in this scanner's input can
  //be interpreted as a float value using the nextFloat()
  //method. The scanner does not advance past any input.
  //If this scanner is closed it throws IllegalStateException.
public float nextFloat()
  //Scans the next token of the input as a float.
  //Returns the float scanned from the input.
  //It throws InputMismatchException if the next token does
  //not match the Float regular expression, or is out of range,
  //NoSuchElementException if input is exhausted, and
  //IllegalStateException if this scanner is closed.
public boolean hasNextDouble()
  //Returns true if the next token in this scanner's input
  //can be interpreted as a double value using the
  //nextDouble() method. The scanner does not advance past
  //any input.
  //If this scanner is closed it throws IllegalStateException.
public double nextDouble()
  //Scans the next token of the input as a double.
  //Returns the double scanned from the input.
  //It throws InputMismatchException if the next token does
  //not match the Float regular expression, or is out of
  //range, NoSuchElementException if the input is exhausted,
  //and IllegalStateException if this scanner is closed.
```

Interface: Set<E> (Package java.util)

Methods

The descriptions of the following methods are similar to the descriptions of the methods of the `interface` `Collection` given earlier in this appendix.

```
boolean add(E o)
boolean addAll(Collection<? extends E> c)
void clear()
boolean contains(Object o)
boolean containsAll(Collection<?> c)
boolean equals(Object o)
int hashCode()
boolean isEmpty()
Iterator<E> iterator()
boolean remove(Object o)
boolean removeAll(Collection<?> c)
boolean retainAll(Collection<?> c)
int size()
Object[] toArray()
```

Class: String (Package java.lang)

Constructors

```
public String()
  //Creates a String object with no characters.
public String(char[] arg)
  //Creates a String object with the character array specified
  //by arg.
public String(char[] arg, int index, int length) throws
                        StringIndexOutOfBoundsException
  //Creates a String object with the character array specified
  //by arg, starting at index and of length
  //specified by length.
public String(String str)
  //Creates a String object and initializes the string
  //object with the characters specified by str.
public String(StringBuffer str)
  //Creates a String object and initializes the string
  //object with the characters specified by str.
```

Methods

```
public char charAt(int index) throws
                            StringIndexOutOfBoundsException
```

```
  //Returns the character at the position specified by index.
public int compareTo(String str)
  //Compares two strings character by character.
  //Returns a negative value if this string is less than str.
  //Returns 0 if this string is same as str.
  //Returns a positive value if this string is greater
  //than str.
public String concat(String str)
  //Returns the string that is this string concatenated
  //with str.
public static String copyValueOf(char[] arg)
  //Returns the string containing the characters of arg.
public static String copyValueOf(char[] arg, int index,
                                 int length)
  //Returns the string containing the characters, starting
  //at index and of length specified by length, of arg.
public boolean endsWith(String str)
  //Returns true if the string ends with the string specified
  //by str. Otherwise, returns false.
public boolean equals(Object anObject)
  //Compares this string to the object specified.
  //Returns true if argument is not null and is a String object
  //that represents the same sequence of characters as this
  //object. Otherwise, it returns false.
public boolean equalsIgnoreCase(String str)
  //Returns true if this string is the same as str; case of the
  //letters is ignored. Otherwise it returns false.
public static String format(String format, Object... args)
  //Returns a formatted string using the specified format
  //string and arguments.
public int indexOf(int ch)
  //Returns the index of the first occurrence of the character
  //specified by ch; if the character specified by ch does not
  //appear in the string, it returns -1.
public int indexOf(int ch, int pos)
  //Returns the index of the first occurrence of the character
  //specified by ch; the parameter pos specifies from where to
  //begin the search. If the character specified by ch does not
  //appear in the string, it returns -1.
public int indexOf(String str)
  //Returns the index of the first occurrence of the string
  //specified by str; if the string specified by str does not
  //appear in the string, it returns -1.
public int indexOf(String str, int pos)
  //Returns the index of the first occurrence of the string
  //specified by str. The parameter pos specifies where to
  //begin the search; if the string specified by str does not
  //appear in the string, it returns -1.
```

```
public int length()
   //Returns the length of the string.
public boolean regionMatches(int ind, String str, int strIndex,
                             int len)
  //Returns true if the substring of str starting at strIndex
  //and length specified by len is the same as the substring
  //of this String object starting at ind and having the
  //same length.
public boolean regionMatches(boolean ignoreCase, int ind,
                     String str, int strIndex, int len)
  //Returns true if the substring of str starting at strIndex
  //and length specified by len is the same as the substring of
  //this String object starting at ind and having the same
  //length. If ignoreCase is true, then during character
  //comparison case is ignored.
public String replace(char charToBeReplaced,
                      char charReplacedWith)
  //Returns the string in which every occurrence of
  //charToBeReplaced is replaced with charReplacedWith.
public boolean startsWith(String str)
  //Returns true if the string begins with the string
  //specified by str; otherwise returns false.
public String substring(int startIndex)
             throws StringIndexOutOfBoundsException
  //Returns the string which is a substring of this string
  //starting at startIndex until the end of the string.
public String substring(int startIndex, int endIndex)
             throws StringIndexOutOfBoundsException
  //Returns the string which is a substring of this string
  //starting at startIndex until endIndex - 1.
public String toLowerCase()
  //Returns the string that is the same as this string except that
  //all uppercase letters of this string are replaced with
  //their equivalent lowercase letters.
public String toUpperCase()
  //Returns the string that is the same as this string except
  //that all lowercase letters of this string are replaced
  //with their equivalent uppercase letters.
public char[] toCharArray()
  //Returns the object as an array of characters.
public String toString()
  //Returns the object as a string.
public static String valueOf(boolean b)
  //Returns a string representation of b.
public static String valueOf(char ch)
  //Returns a string representation of ch.
public static String valueOf(int num)
  //Returns a string representation of num.
```

```
public static String valueOf(Long num)
  //Returns a string representation of num.
public static String valueOf(double decNum)
  //Returns a string representation of decNum.
public static String valueOf(float decNum)
  //Returns a string representation of decNum.
```

Class: StringBuffer (Package java.lang)

Constructors

```
public StringBuffer()
  //Creates a StringBuffer object with no characters.
public StringBuffer(int length) throws NegativeArraySizeException
  //Creates an empty StringBuffer object with initial capacity
  //specified by length.
public StringBuffer(String str)
  //Creates a StringBuffer object and initializes the
  //object with characters specified by str.
```

Methods

```
public StringBuffer append(boolean b)
  //Appends the string specified by b to the string of
  //this object.
public StringBuffer append(char ch)
  //Appends the string specified by ch to the string of
  //this object.
public StringBuffer append(char[] arg)
  //Appends the character array specified by arg to the
  //string of this object.
public StringBuffer append(char[] arg, int index, int length)
  //Appends the character array specified by arg, starting at
  //index and of length specified by length, to the
  //string of this object.
public StringBuffer append(double num)
  //Appends the string representation of num to the string of
  //this object.
public StringBuffer append(float num)
  //Appends the string representation of num to the string of
  //this object.
public StringBuffer append(int num)
  //Appends the string representation of num to the string of
  //this object.
public StringBuffer append(long num)
  //Appends the string representation of num to the string of
  //this object.
public StringBuffer append(Object ob)
```

```
  //Appends the string representation of ob to the string of
  //this object.
public StringBuffer append(String str)
  //Appends the string representation of str to the string of
  //this object.
public int capacity()
  //Returns the capacity of the object.
public char charAt(int index) throws
                            StringIndexOutOfBoundsException
  //Returns the character at the position specified by index.
public void ensureCapacity(int m)
  //Sets the minimum capacity of the object to m. The new
  //capacity may be greater than m.
public void getChars(int st, int end, char[] chArray,
              int index) throws StringIndexOutOfBoundsException
  //Copies the characters of this object starting at st until
  //end into chArray starting at index.
public StringBuffer insert(int index, boolean b)
                   throws StringIndexOutOfBoundsException
  //Inserts the string representation of b into this object
  //starting at index.
public StringBuffer insert(int index, char ch)
                  throws StringIndexOutOfBoundsException
  //Inserts the string representation of ch into this object
  //starting at index.
public StringBuffer insert(int index, char[] chArray)
                       throws StringIndexOutOfBoundsException
  //Inserts the string representation of chArray into this
  //object starting at index.
public StringBuffer insert(int index, double num)
                       throws StringIndexOutOfBoundsException
  //Inserts the string representation of num into this object
  //starting at index.
public StringBuffer insert(int index, float num)
                       throws StringIndexOutOfBoundsException
  //Inserts the string representation of num into this object
  //starting at index.
public StringBuffer insert(int index, int num)
                       throws StringIndexOutOfBoundsException
  //Inserts the string representation of num into this object
  //starting at index.
public StringBuffer insert(int index, long num)
                       throws StringIndexOutOfBoundsException
  //Inserts the string representation of num into this object
  //starting at index.
public StringBuffer insert(int index, Object ob)
                       throws StringIndexOutOfBoundsException
  //Inserts the string representation of ob into this object
  //starting at index.
public StringBuffer insert(int index, String str)
                       throws StringIndexOutOfBoundsException
  //Inserts the string representation of str into this object
```

```
    //starting at index.
public int length()
    //Returns the length of the string.
public StringBuffer reverse()
    //Returns a StringBuffer object with the characters of the
    //string of this object reversed.
public void setCharAt(int index, char ch)
    //Sets the character at index to the character specified
    //by ch.
public void setLength(int length) throws
                          StringIndexOutOfBoundsException
    //Sets length of the string of this object to length; may
    //truncate the string.
public String toString()
    //Returns the object as a string.
```

Class: StringTokenizer (Package java.util)

Constructors

```
public StringTokenizer(String str)
    //Creates a StringTokenizer object. The object is initialized
    //using the string specified by str. The delimiting characters
    //are the default characters.
public StringTokenizer(String str, String delimits)
    //Creates a StringTokenizer object. The object is initialized
    //using the string specified by str. The string specified by
    //delimits specifies the delimiters.
public StringTokenizer(String str, String delimits, boolean tok)
    //Creates a StringTokenizer object. The object is initialized
    //using the string specified by str. The string delimits
    //specifies the delimiters. If the boolean variable tok
    //is true, the delimiters are treated as words.
```

Methods

```
public int countTokens()
    //Returns the number of tokens in the string.
public boolean hasMoreElements()
    //Returns the value true if there are tokens left in the string;
    //otherwise returns false.
public boolean hasMoreTokens()
    //Returns the value true if there are tokens left in the string;
    //otherwise returns false.
public String nextElement() throws NoSuchElementException
    //Returns the next token in the string.
```

```
public String nextToken() throws NoSuchElementException
  //Returns the next token in the string.
public String nextToken(String delimits) throws
                                        NoSuchElementException
  //Returns the next token in the string.
  //The string delimits specifies the delimiters.
```

CLASS: Throwable (PACKAGE java.lang)

Constructors

```
public Throwable()
//Default constructor.
//Creates a Throwable object with an empty message string.
public Throwable(String strMessage)
//Constructor with parameters.
//Creates a Throwable object with message string
//specified by the parameter strMessage.
```

Variables

```
public String getMessage()
  //Returns the detailed message stored in the object.
public void printStackTrace()
  //Prints the stack trace showing the sequence of
  //method calls when an exception occurs.
public void printStackTrace(PrintStream stream)
  //Prints the stack trace showing the sequence of
  //method calls when an exception occurs. Output is sent
  //to the stream specified by the parameter stream.
public void printStackTrace(PrintWriter stream)
  //Prints the stack trace showing the sequence of
  //method calls when an exception occurs.
  //Output is sent to the stream specified by the parameter stream.
public String toString()
  //Returns a string representation of the Throwable object.
```

CLASS: TreeSet<E> (PACKAGE java.util)

Constructors

```
public TreeSet()
  //Constructs a new, empty set. Elements are sorted according
  //to the elements' natural order.
```

```
public TreeSet(Comparator<? super E> coll)
  //Constructs a new, empty set. Elements are sorted according
  //to the comparator specified by the parameter coll.
public TreeSet(Collection<? extends E> coll)
  //Constructs a new set containing the elements of the
  //collection, specified by the parameter coll. The elements are
  //sorted according to the elements' natural order.
```

Methods

```
public boolean add(E obj) throws ClassCastException
  //Adds the element specified by the parameter obj to this set
  //if it is not already present. This method also returns true
  //if did not contain obj.
public boolean addAll(Collection<? extends E> coll) throws
                     ClassCastException, NullPointerException
  //Adds all of the elements of the collection specified by the
  //parameter coll to this set. If the set is changed as a result
  //of this operation, it returns true.
public void clear()
  //Removes all of the elements from this set.
public Object clone()
  //Returns a shallow copy of this TreeSet instance.
public boolean contains(Object obj) throws ClassCastException
  //Returns true if this set contains the element specified by
  //the parameter obj.
public E first()
  //Returns the first element currently in this sorted set.
public boolean isEmpty()
  //Returns true if this set contains no elements.
public Iterator<E> iterator()
  //Returns an iterator over the elements in this set. The
  //elements are returned in ascending order.
public E last() throws NoSuchElementException
  //Returns the last element currently in this sorted set.
public boolean remove(Object†obj) throws ClassCastException
  //If the element specified by the parameter obj is in the set,
  //it is removed. It also returns true if the set contained the
  //object obj.
public int size()
  //Returns the number of elements in this set.
```

E

CLASS: Vector<E> (PACKAGE java.util)

Variables

```
protected int capacityIncrement;
protected int elementCount;
protected Object[] elementData; //Array of references
```

Constructors

```
public Vector()
  //Creates an empty Vector object of 10 components.
public Vector(int size)
  //Creates an empty Vector object of the length specified by size.
public Vector(int size, int increm)
 //Creates an empty Vector object of the length specified by size and
 //the capacity increment specified by increm.
```

Methods

```
public void addElement(E insertObj)
  //Adds the object insertObj at the end.
public int capacity()
  //Returns the capacity of the vector.
public Object clone()
  //Returns a copy of the vector
public boolean contains(Object obj)
  //Returns true if the Vector object contains the specified
  //by obj; otherwise it returns false.
public void copyInto(Object[] dest)
  //Copies the elements of this vector into the array dest.
public Object elementAt(int index) throws
                                 ArrayIndexOutOfBoundsException
  //Returns the element of the vector at the location specified
  //by index.
public void ensureCapacity(int size)
  //Sets the capacity of this vector to size.
public Object firstElement() throws NoSuchElementException
  //Returns the first element of the vector.
public int indexOf(Object obj)
  //Returns the position of the first occurrence of the element
  //specified obj in the vector.
  //If item is not in the vector, it returns -1.
public int indexOf(Object obj, int index)
  //Starting at index, the method returns the position of the
  //first occurrence of the element specified by obj in the
  //vector. If item is not in the vector, it returns -1.
```

```
public void insertElementAt(E insertObj, int index)
                          throws ArrayIndexOutOfBoundsException
  //Inserts the object insertObj at the position specified
  //by index.
public boolean isEmpty()
  //Returns true if the vector is empty; otherwise it
  //returns false.
public Object lastElement() throws NoSuchElementException
  //Returns the last element of the vector.
public int lastIndexOf(Object obj)
  //Starting at the last element, using a backward search, this
  //method returns the position of the first occurrence of the
  //element specified by obj in the vector.
  //If obj is not in the vector, it returns -1.
public int lastIndexOf(Object item, int index)
  //Starting at the position specified by index and using a
  //backward search, this method returns the position of the
  //first occurrence of the element specified obj in the vector.
  //If item is not in the vector, the method returns -1.
public void removeAllElements()
  //Removes all the elements of the vector.
public boolean removeElement(Object obj)
  //If an element specified by obj exists in the list, the
  //element is removed and true is returned; otherwise
  //false is returned.
public void removeElementAt(int index) throws
                                ArrayIndexOutOfBoundsException
  //If an element at the position specified by index exists, it is
  //removed from the vector.
public void setElementAt(E obj, int index) throws
                             ArrayIndexOutOfBoundsException
  //The element specified by obj is stored at the position specified
  //by index.
public int size()
  //Returns the number of elements in the vector.
public void trimToSize()
  //Reduces the length of the vector to the number of elements
  //currently in the vector.
public String toString()
  //Returns a string representation of this vector.
```