JAVA applet, awt and util CLASS REFERENCE

CONTENTS

INTRODUCTION

This Java Class Reference card is one of two Java cards available from SSC. Each section of the card pertains to a particular class or interface. A short explanation is given, followed by a list of any available constructors, defined methods and variables/constants. Self-explanatory methods and variables are listed with no defining text. [] following class or data type indicate an array; more than one set of brackets indicates that the array is n-dimensional, n being the number of sets.

Different type faces are used to indicate the following:
- **Bold** is used for class, interface and method names.
- *Italic* is used for parameter names.
- Roman is used for method class/type, parameter class/type and explanatory text.

This reference card was written by Randy Chapman.

OTHER SSC PRODUCTS:

Java

Java is a new and extremely popular language designed by Sun Microsystems. It is a product of current research, incorporating many features to make the programmer's life easier: multiple threads, garbage collection and networking to name a few. It is also highly portable: the same compiled code will in fact run on all Java-compatible systems. This quick reference does not attempt to cover the language itself. Instead, it covers the tools (class libraries) that come with Java. These allow for GUI programming, data management, network access and more.

JAVA.APPLET

Applet

Base class all applets are derived from. It will contain all of applet's controls. Extends **Panel**.

void **destroy**(); frees up any applet-held resources when browser wishes to destroy object
AppletContext **getAppletContext**();
AudioClip **getAudioClip**(); see **getImage** for syntax
URL **getCodeBase**(); returns URL of .class file
URL **getDocumentBase**(); returns URL of document containing this applet
Image **getImage**(URL *urlName* [, String *name*]); queues retrieval of image at *urlName*+*name* and returns
String **getParameter**(String *paramName*); returns value from <*paramName*> HTML tag
String[ ][ ] **getParameterInfo**();
void **init**(); performs initialization for applet when first called, i.e. before first **start**
boolean **isActive**();
void **play**(); plays audio, see **getImage** for syntax
void **resize**(int *w*, int *h*);
void **resize**(Dimension *theDimension*);
void **start**(); start applet execution
void **stop**(); end applet execution
Also: **getAppletInfo**, **setStub**

AppletContext

Interface defining methods to obtain and alter information about the context in which the applet is executing, such as other applets and current URL.

Applet **getApplet**(String *appletName*);
Enumeration **getApplets**(); get all running applets
AudioClip **getAudioClip**(URL *urlName*);
Image **getImage**(URL *urlName*);
void **showDocument**(URL *urlName* [, String *where*]); displays URL in place, such as a browser frame
void **showStatus**(String *statString*); puts *statString* in browser's status line

AudioClip

Interface defining methods of playing audio piece.

void **loop**(); plays audio, continuously looping
void **play**(); plays audio once; restarts if already playing
void **stop**();

JAVA.AWT

COMPONENTS

Button

Class implements a pushbutton. Extends **Component**.

Button([String *label*]);
String **getLabel**();
void **setLabel**(String *newLabel*);

Canvas

Class offers an area to draw into and to receive events from. It is also useful for implementing custom controls or widgets. Must make a derivative of the class to use it. Extends **Component**.

Canvas();

Checkbox

Class offers both checkbox and radiobox functionality, see also **CheckboxGroup**. Extends **Component**.

Checkbox([String *label* [, CheckboxGroup *theGroup*, boolean *state*]]);
CheckboxGroup **getCheckboxGroup**();
String **getLabel**();
boolean **getState**();
void **setCheckboxGroup**(CheckboxGroup *newGroup*);
void **setLabel**(String *newLabel*);
void **setState**(boolean *checked*);

CheckboxGroup

Class used with **Checkbox** to implement groups of checkboxes, where only one can be currently selected.

CheckboxGroup();
Checkbox **getCurrent**();
void **setCurrent**(Checkbox *theBox*);

Choice

Class implements a choice box, also often known as a ComboBox. Extends **Component**.

Choice();
void **addItem**(String *name*); adds item to end of choices; throws NullPointerException; no method to add item elsewhere
int **countItems**(); returns count of items in choice
String **getItem**(int *whichOne*); returns item's value; no method to change value
item **getSelectedIndex**();
String **getSelectedItem**();
void **select**(int *whichOne*); throws IllegalArgumentException

Component

Base class from which all Java controls, windows and drawing areas are derived.

boolean **action**(Event *theEvent*, Object *theAction*); handles action event. Normally posted by sub-components, such as **Menu** and **Button**. See **handleEvent**.
void **addNotify**(); called when component is actually put on the screen
Rectangle **bounds**(); returns position and size, relative to parent
Image **createImage**(int *width*, int *height*); creates image for double-buffering
Image **createImage**(ImageProducer *theProducer*);
void **deliverEvent**(Event *theEvent*); delivers event for processing
void **disable**();
void **enable**(boolean *enableOrDisable*); enables component if true, otherwise disables
Color **getBackground**();
ColorModel **getColorModel**();
Font **getFont**();
FontMetrics **getFontMetrics**(); will return null if component is not shown
Color **getForeground**();
Graphics **getGraphics**(); will return null if component not shown
Container **getParent**();
Toolkit **getToolkit**();
boolean **gotFocus**(Event *theEvent*, Object *what*); event called when component receives input focus; see **handleEvent**.
boolean **handleEvent**(Event *theEvent*); returns false if event should be passed to component's owner.
void **hide**();
boolean **imageUpdate**(Image *theImage*, int *flags*, int *x*, int *y*, int *w*, int *h*); updates component when image has changed; returns true if image has changed.
boolean **inside**(int *x*, int *y*); returns true if point is inside component
void **invalidate**(); marks component as needing to be laid out
boolean **isEnabled**();
boolean **isShowing**(); returns true if component is visible and in visible container
boolean **isValid**(); true if component has been properly placed
boolean **isVisible**(); returns visibility flag; component might be visible and still not be showing
boolean **keyDown**(Event *theEvent*, int *keyCode*); see **handleEvent**
boolean **keyUp**(Event *theEvent*, int *keyCode*); see **handleEvent**
void **layout**(); called when the layout of subcomponents need to be repositioned
void **list**([PrintStream [, int *indent*]]);
Component **locate**(int *x*, int *y*); returns first subcomponent containing this point, or this component if it contains point and no subcomponents contain point, or null if point not contained

Component (cont)

boolean **lostFocus**(Event *theEvent*, Object *what*); event called when component loses the input focus; see **handleEvent**
Dimension **minimumSize**(); used by **LayoutManager** to determine best layout for this component
boolean **mouseDown**(Event, int *x*, int *y*); see **handleEvent**
Similar: **mouseDrag**, **mouseEnter**, **mouseExit**, **mouseMove**, **mouseDown**, **mouseDrag**
void **move**(int *x*, int *y*); in parent's coordinates
void **nextFocus**(); changes input focus to next component.
void **paint**(Graphics *theComp*); draws this component
void **paintAll**(Graphics *theComp*); draws this component and its children
boolean **postEvent**(Event *theEvent*); handles event, including passing to owner if this component does not handle event
Dimension **preferredSize**(); used by **LayoutManager** to determine best layout for this component
void **print**(Graphics *theComp*); prints this component; calls **paint** if not overridden
void **printAll**(Graphics *theComp*); prints this component and its children; calls **paint** if not overridden
void **removeNotify**(); called when component is removed from the screen
void **repaint**(long *maxWait*); asks component to update itself within *maxWait* milliseconds
void **repaint**(int *x*, int *y*, int *w*, int *h*, long *maxWait*); asks component to update portion of itself within *maxWait* milliseconds
void **requestFocus**(); moves input focus to this component; calls **gotFocus** on success
void **reshape**(int *x*, int *y*, int *w*, int *h*);
void **resize**(Dimension *theDim*);
void **resize**(int *w*, int *h*);
void **setBackground**(Color *theColor*);
void **setFont**(Font *theFont*);
void **setForeground**(Color *theColor*);
void **show**(boolean *showOrHide*);
void **update**(Graphics *theComp*); redraws this component when needed; background has not been cleared
void **validate**(); marks component as having been laid out
Also: **checkImage**, **prepareImage**

Label

Class implements an uneditable label. Extends **Component**.

Label([String *text* [, int *alignment*]]);
int **getAlignment**();
String **getText**();
void **setAlignment**(int *newAlignment*); throws IllegalArgumentException
void **setText**(String *newText*);
Alignment: **LEFT**, **CENTER**, **RIGHT**

List

Class implements a list box. Extends **Component**.

List([int *numRows*, boolean *allowMultiSelect*]); creates listbox of specified height (in items)
void **addItem**(String *name* [, int *where*]); adds item to list at *where*; if *where* is -1, item is added to end
boolean **allowsMultipleSelections**();
void **clear**(); removes all items from **List**
int **countItems**(); returns count of items in **List**
void **delItem**(int *whichOne*);
void **delItems**(int *start*, int *end*);
void **deselect**(int *whichOne*);
String **getItem**(int *whichOne*); returns item's value; no method to change value
int **getRows**(); returns number of visible rows
int **getSelectedIndex**(); returns index of selected item
int[ ] **getSelectedIndexes**(); returns index of selected items
String **getSelectedItem**(); value of selected item
String[ ] **getSelectedItems**();
int **getVisibleIndex**(); item on which **makeVisible** was last used
boolean **isSelected**(int *whichOne*);
void **makeVisible**(int *whichOne*);
Dimension **minimumSize**([int *numRows*]);
Dimension **preferredSize**([int *numRows*]);
void **replaceItem**(String *name*, int *whichOne*);
void **select**(int *whichOne*);
void **setMultipleSelections**(boolean *set*);

Scrollbar

Class implements a standard pushbutton. Extends **Component**.

Scrollbar([int *orientation* [, int *value*, int *visible*, int *minimum*, int *maximum*]]); constructs scrollbar with parameters (see appropriate **get...**() methods for explanation); throws IllegalArgumentException
int **getLineIncrement**(); amount scrollbar moves on single up/down
int **getMinimum**();
int **getMaximum**();
int **getOrientation**();
int **getPageIncrement**(); amount scrollbar moves on page up/down command
int **getValue**();
int **getVisible**(); amount of scrollbar taken by position indicator
void **setLineIncrement**(int *increment*);
void **setOrientation**(int *orientation*);
void **setPageIncrement**(int *increment*);
void **setValue**(int *value*);
void **setValues**(int *value*, int *visible*, int *min*, int *max*);
Orientations: **HORIZONTAL**, **VERTICAL**

TextArea

Class is a multi-line scrollable edit control. Extends **TextComponent**.

TextArea([String *text*] [, int *numRows*, int *numCols*]);
void **appendText**(String *newText*);
int **getColumns**();
int **getRows**();
void **insertText**(String *newText*, int *offset*);
Dimension **minimumSize**([int *numRows*, int *numCols*]);
Dimension **preferredSize**([int *numRows*, int *numCols*]);
void **replaceText**(String *newText*, int *start*, int *end*);

TextComponent

Base class for both single-line (**TextField**); and multi-line (**TextArea**); controls. Extends **Component**.

String **getSelectedText**();
int **getSelectionEnd**();
int **getSelectionStart**();
String **getText**();
boolean **isEditable**();
void **select**(int *start*, int *end*);
void **selectAll**();
void **setEditable**(boolean *isEditable*);
void **setText**(String *strText*);

TextField

Class is a single-line edit control. Extends **TextComponent**.

TextField([String *text*] [, int *numCols*]);
boolean **echoCharIsSet**(); see **setEchoCharacter()**
int **getColumns**();
char **getEchoChar**(); see **setEchoCharacter()**
Dimension **minimumSize**([int *numCols*]);
Dimension **preferredSize**([int *numCols*]);
void **setEchoCharacter**(char *theChar*); if echo character is set, all text in **TextField** appear as that one character; useful for password entry field

CONTAINERS

Container

Class used to contain components and containers. It is a common base for windows, panels and applets. Includes the ability to use a **LayoutManager**. Extends **Component**.

Component **add**(Component *newComp* [, int *index*]); adds component at specified index or end
Component **add**(String *layoutParam*, Component *newComp*); adds component; *layoutParam* format depends on **LayoutManager**
int **countComponents**(); returns number of subcomponents
Component **getComponent**(int *index*); throws ArrayIndexOutOfBoundsException
Component[ ] **getComponents**();
LayoutManager **getLayout**();

Container (cont)

Insets **insets**(); returns four border widths
void **paintComponents**(Graphics *theComp*);
void **printComponents**(Graphics *theComp*); defaults to **paintComponents** if not implemented
void **remove**(Component *theComp*);
void **removeAll**();
void **setLayout**(LayoutManager *theMgr*);

Dialog

Class implements a dialog box. Extends **Window**.

Dialog(Frame *parent* [, String *title*], boolean *modal*); creates dialog, does not show it. As of 1.0.1, *modal* flag does not work.
String **getTitle**();
boolean **isModal**();
boolean **isResizable**();
void **setResizable**(boolean *canResize*);
void **setTitle**(String *title*);

FileDialog

Class implements a native file dialog. It cannot be used inside an **Applet** for security reasons. Extends **Dialog**.

FileDialog(Frame *parent*, String *title* [, int *mode*]);
int **getMode**();
String **getDirectory**();
String **getFile**();
FilenameFilter **getFilenameFilter**();
void **setDirectory**(String *dir*);
void **setFile**(String *fileName*);
void **setFilenameFilter**(FilenameFilter *filter*);
Modes: **LOAD**, **SAVE**

Frame

Class is a top-level window, capable of having a menubar, a full window frame and a separate icon. Extends **Window**.

Frame([String *title*]);
void **dispose**(); discards the Frame
int **getCursorType**();
Image **getIconImage**();
MenuBar **getMenuBar**();
String **getTitle**();
boolean **isResizable**();
void **remove**(MenuComponent *theMenu*);
void **setCursor**(int *cursor*);
void **setIconImage**(Image *icon*);
void **setMenuBar**(MenuBar *menu*);
void **setResizable**(boolean *setResize*);
void **setTitle**(String *title*);
Cursors: **CROSSHAIR_CURSOR**, **DEFAULT_CURSOR**, **E_RESIZE_CURSOR**, **HAND_CURSOR**, **MOVE_CURSOR**, **N_RESIZE_CURSOR**, **NE_RESIZE_CURSOR**, **NW_RESIZE_CURSOR**, **S_RESIZE_CURSOR**, **SE_RESIZE_CURSOR**, **SW_RESIZE_CURSOR**, **TEXT_CURSOR**, **W_RESIZE_CURSOR**, **WAIT_CURSOR**

Panel

Class is a container with a default layout of **FlowLayout**. Applets are derived from a **Panel**. Extends **Container**.

Panel();

Window

Class is the base for all framed windows. Extends **Container**.

Window(Frame *parent*);
void **dispose**(); closes window
Toolkit **getToolkit**();
String **getWarningString**(); returns string displayed as warning to user if this is insecure window; returns null on secure windows
void **pack**(); invokes current **layoutManager** to resize and reposition all children controls; may result in window changing size
void **show**();
void **toBack**();
void **toFront**();

FONTS

Font

Class defines a platform-independent method of specifying fonts. Use with **FontMetrics** to get more detailed information. There is currently no way of specifying a font not in the predefined list.

Font(String *name*, int *style*, int *size*); *name* can be any one of the following: "Helvetica", "TimesRoman", "Courier", "Dialog", "DialogInput", "ZapfDingbats" and anything from Toolkit **getFontlist**
String **getFamily**(); gets platform-dependent family name
static Font **getFont**(String *name*);
static Font **getFont**(String *propertyName*, Font *backupFont*); retrieves font from property list or uses *backupFont* if font doesn't exist
String **getName**();
int **getSize**();
int **getStyle**();
int **hashCode**();
boolean **isBold**(); Similar **isItalic**, **isPlain**
String **name**;
int **size**;
int **style**;
Styles: **PLAIN**, **BOLD**, **ITALIC**

FontMetrics

Class contains detailed information about one font. This cannot be directly constructed; use the **getFontMetrics** method of the appropriate font instance or use the **Toolkit** class.

int **bytesWidth**(bytes[ ] *theBytes*, int *start*, int *length*);
int **charWidth**(char *theChar*); returns width of character
int **charWidth**(int *theChar*); returns width of Unicode character

FontMetrics (cont)

int **charsWidth**(char[ ] *theChars*, int *start*, int *length*);
Font **getFont**( );
int **getAscent**( ); returns average distance between top of character and baseline
Similar: **getMaxAscent**
int **getDescent**( ); returns average distance between baseline and bottom of a character
Similar: **getMaxDescent**
int **getHeight**( ); returns average total height of character (sum of ascent, descent and leading)
Similar: **getMaxHeight**
int **getLeading**( ); returns distance between bottom of characters on one line and top of those on next line
int **getMaxAdvance**( ); returns maximum character width
int[ ] **getWidths**( ); returns array containing widths of characters in font; only first 256 characters are used
int **stringWidth**(String *theString*);

LAYOUTS

LayoutManager

Interface is the base for all layout managers. Layout managers control the placement of all controls and canvases in all AWT Containers, such as **Applet**. **Dialog** and **Frame**. Each component is added to the container through the use of the add method which takes a string parameter that is an instruction to the **LayoutManager** as to where to place the component. Some managers (such as **CardLayout**) have utility functions that take a parent parameter. This parent is the container that the layout is managing.

BorderLayout

Class used for putting up to five components in a container; understands "North", "East", "South", "West", and "Center". All but "Center" will be placed according to their preferred size; "Center" will get all the remaining space. Implements **LayoutManager**.

BorderLayout( );

CardLayout

Class used for situations when only one subcomponent should be visible at any time; usually only panels are added. It is an excellent choice for implementing the tab-switching part of a tab control or for a slide show. Implements **LayoutManager**.

CardLayout([int *borderWidth*, int *borderHeight*]);
void **first**(Container *parent*); shows first card
void **last**(Container *parent*); shows last card
void **next**(Container *parent*); shows next card
void **previous**(Container *parent*); goes to previous card
void **show**(Container *parent*, String *name*); goes to *name* card

Date (cont)

int **getDate**();
int **getDay**();
int **getHours**();
int **getMinutes**();
int **getMonth**();
int **getSeconds**();
long **getTime**();
long **getTimezoneOffset**();
int **getYear**();
void **setDate**(int *newDate*);
void **setHours**(int *newHour*);
void **setMinutes**(int *newMinute*);
void **setMonth**(int *newMonth*);
void **setSeconds**(int *newSecond*);
void **setTime**(long *newTime*);
void **setYear**(int *newYear*);
String **toGMTString**();
String **toLocaleString**();
String **toString**();

Dictionary

Abstract class is a base for mapping between a key and an element.

Dictionary();
Enumeration **elements**();
Object **get**(Object *key*);
boolean **isEmpty**();
Enumeration **keys**();
Object **put**(Object *key*, Object *element*); returns old element if applicable; throws NullPointerException
Object **remove**(Object *key*); returns old element if found
int **size**();

Enumeration

Interface for retrieving a list of elements.

boolean **hasMoreElements**();
Object **nextElement**(); throws NoSuchElementException

Hashtable

Class provides an efficient method to store and lookup key/value pairs. Extends **Dictionary**.

Hashtable([int *initialCapacity* [, float *reloadFactor*]]); *reloadFactor* is between 0-1 and indicates at what point table should be rehashed into larger one; throws IllegalArgumentException
void **clear**();
boolean **contains**(Object *value*); throws NullPointerException
boolean **containsKey**(Object *key*);

Observable

Base class for all objects that can be watched by an Observer.

Observable();
void **addObserver**(Observer *theObserver*);
protected void **clearChanged**(); clear changed flag
int **countObservers**();
void **deleteObserver**(Observer *theObserver*);
void **deleteObservers**();
boolean **hasChanged**();
void **notifyObservers**([Object *arg*]);
protected void **setChanged**();

Observer

Interface that an object must implement in order to watch an Observable.

void **update**(Observable *theObserver*, Object *arg*);

Properties

Class allows for management of user-defined information. Often used to hold setup information. Extends **Hashtable**.

Properties([Properties *defaults*]);
String **getProperty**(String *name* [, String *default*]);
void **list**(PrintStream *inStream*);
void **load**(InputStream *inStream*);
Enumeration **propertyNames**();
void **save**(OutputStream *outStream*, String *comment*);

Random

Class returns random numbers. The range of the numbers is the entire range of the return type.

Random([long *seed*]);
Double **nextDouble**();
Similar: **nextFloat**, **nextInt**, **nextLong**
Double **nextGaussian**();
void **setSeed**(long *seed*);

Stack

Class implements a LIFO (Last-In-First-Out); stack. Extends **Vector**.

Stack();
boolean **empty**(); returns true if stack is empty
Object **peek**(); returns top value without removing it
Object **pop**(); removes and returns topmost item
void **push**(Object *theOBJ*); puts *theObj* on top
int **search**(Object *theOBJ*);

StringTokenizer

Class breaks a string into logical tokens. Implements **Enumeration**.

StringTokenizer(String *theString* [, String *delimiters* [, boolean *delimsAsTokens*]]); if *delimsAsTokens* is true, delimiters are returned as tokens, otherwise they are discarded
int **countTokens**();
boolean **hasMoreTokens**();
String **nextToken**([String *newDelimiters*]); throws NoSuchElementException

Vector

Class is an array of objects that grows automatically.

Vector([int *initialSize*, [, int *growthSize*]]);
void **addElement**(Object *newObject*);
int **capacity**();
boolean **contains**(Object *theElement*);
void **copyInfo**(Object[ ] *dest*);
Object **elementAt**(int *index*); throws ArrayIndexOutOfBoundsException
Enumeration **elements**();
void **ensureCapacity**(int *minSize*);
Object **firstElement**(); throws NoSuchElementException
int **indexOf**(Object *theElement*[, int *index*]);
void **insertElementAt**(Object *theElement*, int *index*); throws ArrayIndexOutOfBoundsException
boolean **isEmpty**();
Object **lastElement**(); throws NoSuchElementException
int **lastIndexOf**(Object *theObject* [, int *start*]); last index of *theObject* before start
void **removeAllElements**();
void **removeElement**(Object *theObject*);
void **removeElementAt**(int *index*); throws ArrayIndexOutOfBoundsException
void **setElementAt**(Object *theElement*, int *index*); throws ArrayIndexOutOfBoundsException
void **setSize**(int *newSize*); loses elements if set to less than current size
int **size**(); number of used entries
void **trimToSize**(); frees unused elements at end of **Vector**

Event (cont)

int **key**; key that was pressed on keyboard event
int **modifiers**; contains status of special keys. Middle and right mouse buttons are reported through ALT and META, respectively, on systems that support these buttons.
Object **target**; item causing event
long **when**; time event occurred
int **x,y**; where pointer was
IDs: **ACTION_EVENT, GOT_FOCUS, LIST_DESELECT, LIST_SELECT, LOAD_FILE, LOST_FOCUS, KEY_ACTION, KEY_ACTION_RELEASE, KEY_PRESS, KEY_RELEASE, MOUSE_DOWN, MOUSE_DRAG, MOUSE_ENTER, MOUSE_EXIT, MOUSE_UP, SAVE_FILE, SCROLL_ABSOLUTE, SCROLL_LINE_DOWN, SCROLL_LINE_UP, SCROLL_PAGE_DOWN, SCROLL_PAGE_UP, WINDOW_DESTROY, WINDOW_DEICONIFY, WINDOW_EXPOSE, WINDOW_ICONIFY, WINDOW_MOVED**
Keys: **DOWN, END, F1...F12, HOME, LEFT, PGDN, PGUP, RIGHT, UP**
Modifiers: **ALT_MASK, CTRL_MASK, META_MASK, SHIFT_MASK**

Graphics

Class defines the interface used to draw an image to the screen or to other drawable areas.

void **clearRect**(int *x*, int *y*, int *toX*, int *toY*);
void **clipRect**(int *x*, int *y*, int *w*, int *h*); sets new clipping area common between existing clipping rectangle and new one
void **copyArea**(int *x*, int *y*, int *w*, int *h*, int *distX*, int *distY*); copies area to a place *distX*, *distY* distance from original location
Graphics **create**([int *x*, int *y*, int *w*, int *h*]); duplicates object, sets *x*, *y* to origin and clips to *w*idth, *h*eight
void **dispose**(); release resources taken by this interface
void **draw3DRect**(int *x*, int *y*, int *toX*, int *toY*, boolean *raised*);
void **drawArc**(int *x*, int *y*, int *w*, int *h*, int *start*, int *overAngle*);
void **drawBytes**(byte[ ] *byteData*, int *start*, int *length*, int *x*, int *y*);
void **drawChars**(char[ ] *charData*, int *start*, int *length*, int *x*, int *y*);
boolean **drawImage**(Image *theImage*, int *x*, int *y* [, int *w*, int *h*] [, Color *theColor*], ImageObserver *theObserver*); draws image at location; if incomplete, notifies observer and returns false asynchronously
void **drawLine**(int *x*, int *y*, int *toX*, int *toY*);
void **drawOval**(int *x*, int *y*, int *w*, int *h*);
void **drawPolygon**(int[ ] *x*, int[ ] *y*, int *nPoints*);
void **drawRect**(int *x*, int *y*, int *toX*, int *toY*);
void **drawRoundRect**(int *x*, int *y*, int *toX*, int *toY*, int *arcW*, int *arcH*);
void **drawString**(String *theString*, int *x*, int *y*);
void **fill3DRect**(int *x*, int *y*, int *toX*, int *toY*, boolean *raised*);
void **fillArc**(int *x*, int *y*, int *w*, int *h*, int *start*, int *overAngle*);

Graphics (cont)

void **fillOval**(int *x*, int *y*, int *w*, int *h*);
void **fillPolygon**(int[ ] *x*, int[ ] *y*, int *nPoints*);
void **fillRect**(int *x*, int *y*, int *toX*, int *toY*);
void **fillRoundRect**(int *x*, int *y*, int *toX*, int *toY*, int *arcW*, int *arcH*);
Rectangle **getClipRect**();
Color **getColor**();
Font **getFont**(); returns current font
FontMetrics **getFontMetrics**();
void **setColor**(Color *theColor*);
void **setFont**(Font *theFont*);
void **setPaintMode**(); sets drawing mode to opaque
void **setXORMode**(Color *xorMask*); sets drawing mode to exclusive **or**, e.g. for rubber banding
void **translate**(int *x*, int *y*); sets x,y as origin

Image

Abstract class is a base for an image in the AWT.

void **flush**(); releases all resources associated with current image
Graphics **getGraphics**(); returns Graphics object that can be used to modify image
int **getHeight**(ImageObserver *theObserver*); returns height of image; if information is not yet available, returns -1 and notifies observer when it is
Similar: **getWidth**
Object **getProperty**(String *propertyName*, ImageObserver *theObserver*); returns value of property or **UndefinedProperty** if property doesn't exist in image. Property names are specific to image types. If properties for image are not yet available, returns null and notifies observer when they are.
static Object **UndefinedProperty**; returned by **getProperty** when requested property is not in image

MediaTracker

Class used to wait for images to load and to check their status. Multiple images can have the same ID, which allows them to be loaded in parallel.

MediaTracker(Component *caller*);
void **addImage**(Image *newImage*, int *id* [, int *width*, int *height*]);
boolean **checkAll**([boolean *startLoading*]); returns true if all images loaded
boolean **checkID**(int *id* [, boolean *startLoading*]);
Object[ ] **getErrorsAny**(); returns array of images in which a loading error occurred
boolean[ ] **getErrorsID**(int *ID*);
boolean **isErrorAny**(); checks for error
boolean **isErrorID**(int *ID*);
int **statusAll**(boolean *startLoading*);
int **statusID**(int *ID*, boolean *startLoading*);
void **waitForAll**(); throws InterruptedException
boolean **waitForAll**(long *timeout*); throws InterruptedException
void **waitForID**(int *ID*, long *maxWait*); throws InterruptedException
Status: **ABORTED, COMPLETE, DONE, ERRORED, LOADING, LOADSTARTED**

Toolkit

Abstract class used to determine information global to the running AWT, such as display parameters.

int **checkImage**(Image *theImage*, int *width*, ImageObserver *theObserver*); returns bitwise **or** of operations currently completed in process of preparing image
Image **createImage**(ImageProducer *theProducer*); creates image from producer
ColorModel **getColorModel**(); returns display's color model; AWT does not have the ability to use multiple ColorModels on hardware that support it
static Toolkit **getDefaultToolkit**(); returns default toolkit, as defined by "awt.toolkit" property; throws AWTError
String[ ] **getFontList**(); list of font names this Toolkit supports
FontMetrics **getFontMetrics**(Font *theFont*);
Image **getImage**(String *fileName*);
image **getImage**(URL *urlName*);
int **getScreenResolution**(); returns what system thinks is dots per inch of screen; many systems cannot accurately determine this information
Dimension **getScreenSize**();
boolean **prepareImage**(Image *theImage*, int *w*, int *h*, ImageObserver *theObserver*); asynchronously starts retrieval, scaling and any other operation needed to display image
void **sync**(); ensures all drawing has been flushed

JAVA.UTIL

BitSet

Class provides a variable size array of boolean bits.

BitSet([int *numberOfBits*]);
void **and**(BitSet *other*); **and**s *other* with this bit set
void **clear**(int *index*); sets bit to false
boolean **get**(int *index*); get status of the bit
void **or**(BitSet *other*); **or**s bits in *other* with this bit set
void **set**(int *index*); sets bit to true
int **size**(); in bits
void **xor**(BitSet *other*); does bitwise exclusive **or**

Date

Class allows for parsing and presenting of dates. Milliseconds (UTC) are measured from 00:00:00 GMT, January 1, 1970, years from 1900, months in range 0-11, hours in 0-23 and days from 0-6, 0 being Sunday.

Date([long *milliseconds*]);
Date(int *yr*, int *mon*, int *day* [, int *hr*, int *min* [, int *sec*]]);
Date(String *theString*); see **parse**
boolean **after**(Date *when*);
boolean **before**(Date *when*);
static long **UTC**(int *yr*, int *mon*, int *day*, int *hr*, int *min*, int *sec*);
static long **parse**(String *theString*); understands number of formats, especially IETF from RFC822

Point

Class specifies a point in the coordinate system.

Point(int *x*, int *y*);
void **move**(int *x*, int *y*); moves point to coordinates
void **translate**(int *h*, int *v*); moves point by amount
int **x**, **y**;

Polygon

Class contains the information needed to represent and draw a polygon.

Polygon([int[ ] *xpoints*, int[ ] *ypoints*, int *npoints*]);
void **addPoint**(int *x*, int *y*);
Rectangle **getBoundingBox**(); returns smallest possible rectangle that contains entire polygon
boolean **inside**(int *x*, int *y*); returns whether or not point is inside polygon; uses alternating rule
int **npoints**;
int[ ] **xpoints**;
int[ ] **ypoints**;

Rectangle

Class contains the information needed to represent and draw a rectangle, as well as do basic operations on it.

Rectangle([Dimension *theDim*]);
Rectangle([int *x*, int *y*,] int *w*, int *h*);
Rectangle(Point *newPoint* [, Dimension *theDim*]);
void **add**(int *x*, int *y*); increases rectangle by minimum amount needed to fully contain original rectangle and new point
void **add**(Point *newPoint*); increases rectangle by minimum amount needed to fully contain original rectangle and new point
void **add**(Rectangle *newRect*); changes rectangle to be a union of itself and new rectangle
void **grow**(int *w*, int *h*); moves upper-left coordinate up and left by amount specified and moves lower-right coordinate down and right by same amount
boolean **inside**(int *x*, int *y*); returns true if point is inside rectangle
Rectangle **intersection**(Rectangle *newRect*); returns new rectangle containing all points in both
boolean **intersects**(Rectangle *newRect*); returns true if two rectangles share any common area
boolean **isEmpty**(); returns true if rectangle has zero width and height
void **move**(int *x*, int *y*);
void **reshape**(int *x*, int *y*, int *w*, int *h*);
void **resize**(int *w*, int *h*);
void **translate**(int *horz*, int *vert*); shifts rectangle by specified amount
Rectangle **union**(Rectangle *newRect*); returns smallest rectangle that completely contains both
int **x**, **y**, **width**, **height**;

MENUS

CheckboxMenuItem

Class implements a menu item with an option check beside it. Extends **MenuItem**.

CheckboxMenuItem(String *label*);
boolean **getState**();
void **setState**(boolean *checked*);

Menu

Class is a menu such as a pulldown menu in the menubar or a submenu of another menu. Extends **MenuItem** and implements **MenuContainer**.

Menu(String *label* [, boolean *canTearOff*]); creates menu that can optionally be placed in separate window by user on platforms that support it
MenuItem **add**(MenuItem *newItem*);
void **add**(String *label*);
void **addSeparator**();
int **countItems**();
MenuItem **getItem**(int *index*);
boolean **isTearOff**();
void **remove**(int *index*);
void **remove**(MenuComponent *theMenu*);

MenuBar

Class contains a frame's main menu. Extends **MenuComponent**.

MenuBar();
Menu **add**(Menu *newMenu*);
int **countMenus**();
Menu **getHelpMenu**(); help menus are special in that they are right justified or otherwise specially placed on systems where that is standard
Menu **getMenu**(int *index*);
void **remove**(int *index*);
void **remove**(MenuComponent *theMenu*);
void **setHelpMenu**(Menu *theMenu*);

MenuComponent

Class provides methods common to all menu types.

MenuComponent();
Font **getFont**();
MenuContainer **getParent**();
boolean **postEvent**(Event *theEvent*);
void **setFont**(Font *theFont*);

MenuContainer

Interface for all classes that contain menu items.

Font **getFont**();
boolean **postEvent**(Event *theEvent*);
void **remove**(MenuComponent *theComponent*);

MenuItem

Class is your basic single item on a menu. Extends **Component**.

MenuItem(String *label*);
void **disable**();
void **enable**([boolean *enableOrDisable*]);
boolean **isEnabled**();
String **getLabel**();
void **setLabel**(String *label*);

MISC

Color

Class contains color types and conversions.

Color(int *theBits*); creates color with blue from bits 0-7, green from bits 8-15 and red from bits 16-23
Color(float *r*, float *g*, float *b*); creates color from *r*, *g* and *b*, each in the range 0.0-1.0
Color(int *r*, int *g*, int *b*); creates a color from *r*, *g* and *b*, each in range 0-255
Color **brighter**(); returns new, brighter color
Color **darker**(); returns new color, darker than original
int **getBlue**(); returns blue component as index 0-255
Similar: **getGreen**, **getRed**
static Color **getColor**(String *newColor* [, Color *backup*]); returns color from property; *backup* or null if not found
static Color **getColor**(String *newColor*, int *backup*); returns color from property or *backup* if not found
static Color **getHSBColor**(float *hue*, float *saturation*, float *brightness*);
int **getRGB**(); creates color with blue from bits 0-7, green from bits 8-15, and red from bits 16-23
static int **HSBtoRGB**(float *hue*, float *saturation*, float *brightness*); returns packed color in same format as **getRGB**
static float[ ] **RGBtoHSB**(int *r*, int *g*, int *b*, float[ ] *vals*); converts *r*, *g*, *b* to *h*, *s*, *b* and puts it in *vals*[0-2], respectively; returns *vals*
Constants: **black**, **blue**, **cyan**, **darkGray**, **gray**, **green**, **lightGray**, **magenta**, **orange**, **pink**, **red**, **white**, **yellow**

Event

Class holds all relevant information for an event.

Event(Object *target*, long *when*, int *id*, int *x*, int *y*, int *key*, int *modifiers* [, Object *arg*]);
Event(Object *target*, int *id*, Object *arg*);
void **translate**(int *x*, int *y*); shifts all coordinates of event
boolean **metaDown**(); queries status of alt (meta) key
Similar: **controlDown**, **shiftDown**
void **translate**(int *x*, int *y*); shifts all coordinates of event
Object **arg**; message-specific argument
int **clickCount**; number of consecutive clicks that resulted in this message; some versions of Java have problems reliably reporting this information
Event **evt**; next event in queue
int **id**; type of event

FlowLayout

Class is a simple layout design in the vein of word wrapping in a word processer. Implements **LayoutManager**.

FlowLayout([int *align* [, int *horizGap*, int *vertGap*]]);

GridBagConstraints

Class used with **GridBagLayout**.

GridBagConstraints( );
int **anchor**;
int **fill**;
int **gridx**, **gridy**, **gridwidth**, **gridheight**;
int **ipadx**, **ipady**;
double **weightx**, **weighty**;
Anchors: **NORTH**, **NORTHEAST**, **EAST**, **SOUTHEAST**, **SOUTH**, **SOUTHWEST**, **WEST**, **NORTHWEST**
Fills: **NONE**, **HORIZONTAL**, **VERTICAL**, **BOTH**, **RELATIVE**, **REMAINDER**

GridBagLayout

Class is a powerful layout manager, excellent for creating complicated forms and dialogs. Implements **LayoutManager**.

GridBagLayout( );
GridBagConstraints **getConstraints**(Components *theComp*); returns copy of Component's constraints
int[ ][ ] **getLayoutDimensions**( ); returns copy of the layout dimensions
Point **getLayoutOrigin**( );
double[ ][ ] **getLayoutWeights**( ); returns copy of the layout weights
Point **location**(int *x*, int *y*);
void **setConstraints**(Component, GridBagConstraints);
int[ ] **columnWidths**;
int[ ] **rowHeights**;
double[ ] **columnWeights**;
double[ ] **rowHeights**;

GridLayout

Class is a simple grid layout where each item is the same size. Implements **LayoutManager**.

GridLayout(int *rows*, int *cols* [, int *horizGap*, int *vertGap*]);

MEASUREMENTS

Dimension

Class specifies a size or distance.

Dimension([Dimension *theDim*]);
Dimension(int *w*, int *h*);
int **width**, **height**;

Insets

Class contains information on the size of the borders around a panel.

Insets(int *top*, int *left*, int *bottom*, int *right*);
int **top**, **left**, **bottom**, **right**;