|  |  |  |
| --- | --- | --- |
| **MODULE 2** | **COMPUTER PROGRAMMING II** | |
|  |  | |
| **TIME ALLOTMENT** | **1 week** | |
| **INSTRUCTOR** | **Javier, Rodolfo jr. A** | |
|  |  | |
| **FACEBOOK** | **https://www.facebook.com/19javier93** | |
| **EMAIL** | **javierrodolfojr1993@gmail.com** | |
| **CELL NUMBER** | **09504046956** | |
| **I OVERVIEW** | |
| Computer Programming II focuses on software development using the java programming language and implementing the Object-Oriented Programming concepts. This course is designed to advance the knowledge of the students in terms of developing object-oriented software using the java language and architecture. This will also include the use of API’s, libraries, and repositories to further enhance and improve the functionality and development of the software. | |
| **II TARGETED COURSE LEARNING OUTCOME** | |
| CLO3. The students should be able to use internal and external API’s and standard libraries of the assigned programming language. | |
| **III TARGETED TOPIC LEARNING OUTCOME** | |
| The students will be able to use the appropriate API provided by the java programming language. | |
| Content: Search and explore the web about the documentations of java API Specifications. | |
| **IV ASSESSMENT** | |
| **Skills test**: Write a (5) five running java program examples using the different java API Specifications.  *Note:* Add a folder on your repository and name it as **module2\_assessment** and put your activity files then commit it on **github**. | |

Note: Add a folder and name it as **module2\_tla** (tla means Teaching Learning Activities) before you commit your answer/document.

**A. ENGAGE: Misconception**

Commit your answer/document on your remote repository that was shared to your instructor github account.

**Misconception Check**

In the past several years the technology evolved too fast and there’s a lot of updates in java versions that was developed and existing now. Select 2 versions of java and compare them in terms of features, advantages, and disadvantages.

Java SE 8 (March 18, 2014)

* Its codename was **Spider**. Although, codenames have been discontinued, but the codename **Spider** is common among java developers. It includes some features which were proposed for Java SE 7 but added in Java SE 8. Language-level support for Lambda expressions. Allowed developers to embed JavaScript code within applications. Annotation of Java Types. Provided Date and Time API. Repeating Annotations. Launching of JavaFX applications. Removal of permanent generation. Java SE 8 is not supported in Windows XP but after JDK 8 update 25, we can install and run it under Windows XP. Java 8 is set as a default version to download from java.com from October 2014.

Java SE 9 (September 21, 2017)

* In 2016, Oracle discussed some features to release in Java 9. It was hoped that Java 9 would include better support for multi-gigabyte heaps, better native code integration, a different default garbage collector and a self-tuning JVM. The release of Java 9 was postponed many times and finally it was released on September 21, 2017. It includes some specific features: Modularization of the JDK under Project Jigsaw. Provided Money and Currency API. Tight integration with JavaFX. Java implementation of reactive streams. More Concurrency Updates. Provided Java Linker. Automatic scaling and sizing.

**B. EXPLORE: API Specifications list**

1. Enumerate at least 20 following API specifications of java, depending of the java version.

Example: follow the format

## JAVA version: Java SE 8 (March 18, 2014)

**Packages Description**

**Java.awt** - Contains all of the classes for creating user interfaces and for painting graphics and images.

|  |
| --- |
| **Interface**  **ActiveEvent-** An interface for events that know how to dispatch themselves. By implementing this interface an event can be placed upon the event queue and its dispatch() method will be called when the event is dispatched, using the EventDispatchThread.  **Adjustable-** The interface for objects which have an adjustable numeric value contained within a bounded range of values.  **Composite-** The Composite interface, along with [CompositeContext](https://docs.oracle.com/javase/8/docs/api/java/awt/CompositeContext.html" \o "interface in java.awt), defines the methods to compose a draw primitive with the underlying graphics area. After the Composite is set in the [Graphics2D](https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics2D.html) context, it combines a shape, text, or an image being rendered with the colors that have already been rendered according to pre-defined rules.  **CompositeContext-** The CompositeContext interface defines the encapsulated and optimized environment for a compositing operation. CompositeContext objects maintain state for compositing operations. In a multi-threaded environment, several contexts can exist simultaneously for a single [Composite](https://docs.oracle.com/javase/8/docs/api/java/awt/Composite.html) object.  **ItemSelectable-** The interface for objects which contain a set of items for which zero or more can be selected.  **KeyEventDispatcher-** A KeyEventDispatcher cooperates with the current KeyboardFocusManager in the targeting and dispatching of all KeyEvents. KeyEventDispatchers registered with the current KeyboardFocusManager will receive KeyEvents before they are dispatched to their targets, allowing each KeyEventDispatcher to retarget the event, consume it, dispatch the event itself, or make other changes.  **KeyEventPostProcessor-** A KeyEventPostProcessor cooperates with the current KeyboardFocusManager in the final resolution of all unconsumed KeyEvents. KeyEventPostProcessors registered with the current KeyboardFocusManager will receive KeyEvents after the KeyEvents have been dispatched to and handled by their targets.  **LayoutManager-** Defines the interface for classes that know how to lay out Containers.  **LayoutManager2-** Defines an interface for classes that know how to layout Containers based on a layout constraints object. This interface extends the LayoutManager interface to deal with layouts explicitly in terms of constraint objects that specify how and where components should be added to the layout.  **MenuContainer-** The super class of all menu related containers.  **Paint-** extends [Transparency](https://docs.oracle.com/javase/8/docs/api/java/awt/Transparency.html)  This Paint interface defines how color patterns can be generated for [Graphics2D](https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics2D.html) operations. A class implementing the Paint interface is added to the Graphics2D context in order to define the color pattern used by the draw and fill methods.  **PaintContext-**The PaintContext interface defines the encapsulated and optimized environment to generate color patterns in device space for fill or stroke operations on a [Graphics2D](https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics2D.html). The PaintContext provides the necessary colors for Graphics2D operations in the form of a [Raster](https://docs.oracle.com/javase/8/docs/api/java/awt/image/Raster.html) associated with a [ColorModel](https://docs.oracle.com/javase/8/docs/api/java/awt/image/ColorModel.html" \o "class in java.awt.image). The PaintContext maintains state for a particular paint operation. In a multi-threaded environment, several contexts can exist simultaneously for a single [Paint](https://docs.oracle.com/javase/8/docs/api/java/awt/Paint.html) object.  **PrintGraphics-** An abstract class which provides a print graphics context for a page.  **SecondaryLoop-** A helper interface to run the nested event loop. Objects that implement this interface are created with the [EventQueue.createSecondaryLoop()](https://docs.oracle.com/javase/8/docs/api/java/awt/EventQueue.html" \l "createSecondaryLoop--) method. The interface provides two methods, [enter()](https://docs.oracle.com/javase/8/docs/api/java/awt/SecondaryLoop.html#enter--) and [exit()](https://docs.oracle.com/javase/8/docs/api/java/awt/SecondaryLoop.html#exit--), which can be used to start and stop the event loop.  **Shape-** The Shape interface provides definitions for objects that represent some form of geometric shape. The Shape is described by a [PathIterator](https://docs.oracle.com/javase/8/docs/api/java/awt/geom/PathIterator.html" \o "interface in java.awt.geom) object, which can express the outline of the Shape as well as a rule for determining how the outline divides the 2D plane into interior and exterior points.  **Stroke-** The Stroke interface allows a [Graphics2D](https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics2D.html) object to obtain a [Shape](https://docs.oracle.com/javase/8/docs/api/java/awt/Shape.html) that is the decorated outline, or stylistic representation of the outline, of the specified Shape. Stroking a Shape is like tracing its outline with a marking pen of the appropriate size and shape. The area where the pen would place ink is the area enclosed by the outline Shape.  **Transparency-** The Transparency interface defines the common transparency modes for implementing classes.    **Classes**  **AlphaComposite**- extends [Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html) implements [Composite](https://docs.oracle.com/javase/8/docs/api/java/awt/Composite.html)  The AlphaComposite class implements basic alpha compositing rules for combining source and destination colors to achieve blending and transparency effects with graphics and images  **AWTEvent**- extends [EventObject](https://docs.oracle.com/javase/8/docs/api/java/util/EventObject.html)  The root event class for all AWT events. This class and its subclasses supercede the original java.awt.Event class. Subclasses of this root AWTEvent class defined outside of the java.awt.event package should define event ID values greater than the value defined by RESERVED\_ID\_MAX.  **AWTEventMulticaster**- extends [Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html) implements [ComponentListener](https://docs.oracle.com/javase/8/docs/api/java/awt/event/ComponentListener.html), [ContainerListener](https://docs.oracle.com/javase/8/docs/api/java/awt/event/ContainerListener.html), [FocusListener](https://docs.oracle.com/javase/8/docs/api/java/awt/event/FocusListener.html), [KeyListener](https://docs.oracle.com/javase/8/docs/api/java/awt/event/KeyListener.html), [MouseListener](https://docs.oracle.com/javase/8/docs/api/java/awt/event/MouseListener.html), [MouseMotionListener](https://docs.oracle.com/javase/8/docs/api/java/awt/event/MouseMotionListener.html), [WindowListener](https://docs.oracle.com/javase/8/docs/api/java/awt/event/WindowListener.html), [WindowFocusListener](https://docs.oracle.com/javase/8/docs/api/java/awt/event/WindowFocusListener.html), [WindowStateListener](https://docs.oracle.com/javase/8/docs/api/java/awt/event/WindowStateListener.html), [ActionListener](https://docs.oracle.com/javase/8/docs/api/java/awt/event/ActionListener.html), [ItemListener](https://docs.oracle.com/javase/8/docs/api/java/awt/event/ItemListener.html), [AdjustmentListener](https://docs.oracle.com/javase/8/docs/api/java/awt/event/AdjustmentListener.html), [TextListener](https://docs.oracle.com/javase/8/docs/api/java/awt/event/TextListener.html), [InputMethodListener](https://docs.oracle.com/javase/8/docs/api/java/awt/event/InputMethodListener.html), [HierarchyListener](https://docs.oracle.com/javase/8/docs/api/java/awt/event/HierarchyListener.html), [HierarchyBoundsListener](https://docs.oracle.com/javase/8/docs/api/java/awt/event/HierarchyBoundsListener.html), [MouseWheelListener](https://docs.oracle.com/javase/8/docs/api/java/awt/event/MouseWheelListener.html)  **AWTKeyStroke**- extends [Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html)implements [Serializable](https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html) An AWTKeyStroke represents a key action on the keyboard, or equivalent input device. AWTKeyStrokes can correspond to only a press or release of a particular key, just as KEY\_PRESSED and KEY\_RELEASED KeyEvents do; alternately, they can correspond to typing a specific Java character, just as KEY\_TYPED KeyEvents do. In all cases, AWTKeyStrokes can specify modifiers (alt, shift, control, meta, altGraph, or a combination thereof) which must be present during the action for an exact match.  **BorderLayout**- extends [Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html) implements [LayoutManager2](https://docs.oracle.com/javase/8/docs/api/java/awt/LayoutManager2.html), [Serializable](https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html)  A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center.  **BufferCapabilities**- extends [Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html) implements [Cloneable](https://docs.oracle.com/javase/8/docs/api/java/lang/Cloneable.html)  Capabilities and properties of buffers.  . **BufferCapabilities.FlipContents-**extends [Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html) A type-safe enumeration of the possible back buffer contents after page-flipping |
| **BufferCapabilities.FlipContents**- extends [Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html) A type-safe enumeration of the possible back buffer contents after page-flipping  **Button** - extends [Component](https://docs.oracle.com/javase/8/docs/api/java/awt/Component.html) implements [Accessible](https://docs.oracle.com/javase/8/docs/api/javax/accessibility/Accessible.html)  This class creates a labeled button. The application can cause some action to happen when the button is pushed. This image depicts three views of a "Quit" button as it appears under the Solaris operating system:  **Canvas**- extends [Component](https://docs.oracle.com/javase/8/docs/api/java/awt/Component.html) implements [Accessible](https://docs.oracle.com/javase/8/docs/api/javax/accessibility/Accessible.html) A Canvas component represents a blank rectangular area of the screen onto which the application can draw or from which the application can trap input events from the user.  **CardLayout** - extends [Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html) implements [LayoutManager2](https://docs.oracle.com/javase/8/docs/api/java/awt/LayoutManager2.html), [Serializable](https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html)  A CardLayout object is a layout manager for a container. It treats each component in the container as a card. Only one card is visible at a time, and the container acts as a stack of cards.  **Checkbox**- A check box is a graphical component that can be in either an "on" (true) or "off" (false) state. Clicking on a check box changes its state from "on" to "off," or from "off" to "on."  **CheckboxGroup**- The CheckboxGroup class is used to group together a set of Checkbox buttons.  **CheckboxMenuItem**- This class represents a check box that can be included in a menu. Selecting the check box in the menu changes its state from "on" to "off" or from "off" to "on."  **Choice**- The Choice class presents a pop-up menu of choices. The current choice is displayed as the title of the menu.  **Color**- The Color class is used to encapsulate colors in the default sRGB color space or colors in arbitrary color spaces identified by a [ColorSpace](https://docs.oracle.com/javase/8/docs/api/java/awt/color/ColorSpace.html" \o "class in java.awt.color). Every color has an implicit alpha value of 1.0 or an explicit one provided in the constructor.  **Component**- A *component* is an object having a graphical representation that can be displayed on the screen and that can interact with the user. Examples of components are the buttons, checkboxes, and scrollbars of a typical graphical user interface.  **ComponentOrientation**- The ComponentOrientation class encapsulates the language-sensitive orientation that is to be used to order the elements of a component or of text.  **Container**- A generic Abstract Window Toolkit(AWT) container object is a component that can contain other AWT components.  **ContainerOrderFocusTraversalPolicy**- A FocusTraversalPolicy that determines traversal order based on the order of child Components in a Container. From a particular focus cycle root, the policy makes a pre-order traversal of the Component hierarchy, and traverses a Container's children according to the ordering of the array returned by Container.  **Cursor**- A class to encapsulate the bitmap representation of the mouse cursor.  **DefaultFocusTraversalPolicy**- A FocusTraversalPolicy that determines traversal order based on the order of child Components in a Container.  **DefaultKeyboardFocusManager**- The default KeyboardFocusManager for AWT applications. Focus traversal is done in response to a Component's focus traversal keys, and using a Container's FocusTraversalPolicy.  **Desktop**- The Desktop class allows a Java application to launch associated applications registered on the native desktop to handle a [URI](https://docs.oracle.com/javase/8/docs/api/java/net/URI.html) or a file.  **Dialog**- A Dialog is a top-level window with a title and a border that is typically used to take some form of input from the user. The size of the dialog includes any area designated for the border.  **Dimension**- The Dimension class encapsulates the width and height of a component (in integer precision) in a single object. The class is associated with certain properties of components. Several methods defined by the Component class and the LayoutManager interface return a Dimension object.  **DisplayMode**- The DisplayMode class encapsulates the bit depth, height, width, and refresh rate of a GraphicsDevice.  Event-  is a platform-independent class that encapsulates events from the platform's Graphical User Interface in the Java 1.0 event model. In Java 1.1 and later versions, the Event class is maintained only for backwards compatibility.  EventQueue-  is a platform-independent class that queues events, both from the underlying peer classes and from trusted application classes. |

**EventQueue**- EventQueue is a platform-independent class that queues events, both from the underlying peer classes and from trusted application classes.

**FileDialog**- The FileDialog class displays a dialog window from which the user can select a file.

**FlowLayout**- A flow layout arranges components in a directional flow, much like lines of text in a paragraph.

**FocusTraversalPolicy**- A FocusTraversalPolicy defines the order in which Components with a particular focus cycle root are traversed.

**Font**- The Font class represents fonts, which are used to render text in a visible way.

**FontMetrics**- The FontMetrics class defines a font metrics object, which encapsulates information about the rendering of a particular font on a particular screen.

**Frame**- A Frame is a top-level window with a title and a border.

**GradientPaint**- The GradientPaint class provides a way to fill a [Shape](https://docs.oracle.com/javase/8/docs/api/java/awt/Shape.html) with a linear color gradient pattern.

**Graphics**- The Graphics class is the abstract base class for all graphics contexts that allow an application to draw onto components that are realized on various devices, as well as onto off-screen images.

**Graphics2D**- This Graphics2D class extends the [Graphics](https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics.html) class to provide more sophisticated control over geometry, coordinate transformations, color management, and text layout.

**GraphicsConfigTemplate**- The GraphicsConfigTemplate class is used to obtain a valid [GraphicsConfiguration](https://docs.oracle.com/javase/8/docs/api/java/awt/GraphicsConfiguration.html" \o "class in java.awt).

**GraphicsConfiguration**- The GraphicsConfiguration class describes the characteristics of a graphics destination such as a printer or monitor.

**GraphicsDevice**- The GraphicsDevice class describes the graphics devices that might be available in a particular graphics environment.

**raphicsEnvironment**- The GraphicsEnvironment class describes the collection of [GraphicsDevice](https://docs.oracle.com/javase/8/docs/api/java/awt/GraphicsDevice.html" \o "class in java.awt) objects and [Font](https://docs.oracle.com/javase/8/docs/api/java/awt/Font.html) objects available to a Java(tm) application on a particular platform.

**GridBagConstraints**- The GridBagConstraints class specifies constraints for components that are laid out using the GridBagLayout class.

**GridBagLayout**- The GridBagLayout class is a flexible layout manager that aligns components vertically, horizontally or along their baseline without requiring that the components be of the same size.

**GridBagLayoutInfo**- The GridBagLayoutInfo is an utility class for GridBagLayout layout manager. It stores align, size and baseline parameters for every component within a container.

**GridLayout**- The GridLayout class is a layout manager that lays out a container's components in a rectangular grid.

**Image**- The abstract class Image is the superclass of all classes that represent graphical images. The image must be obtained in a platform-specific manner.

**JobAttributes**- A set of attributes which control a print job.

**JobAttributes.DefaultSelectionType**- A type-safe enumeration of possible default selection states.

**JobAttributes.DestinationType**- A type-safe enumeration of possible job destinations.

**JobAttributes.DialogType**- A type-safe enumeration of possible dialogs to display to the user.

**JobAttributes.MultipleDocumentHandlingType**- A type-safe enumeration of possible multiple copy handling states. It is used to control how the sheets of multiple copies of a single document are collated.

**JobAttributes.MultipleDocumentHandlingType**- A type-safe enumeration of possible multiple copy handling states. It is used to control how the sheets of multiple copies of a single document are collated.

**JobAttributes.SidesType**- A type-safe enumeration of possible multi-page impositions. These impositions are in compliance with IPP 1.1.

**KeyboardFocusManager**- The KeyboardFocusManager is responsible for managing the active and focused Windows, and the current focus owner.

**Label**- A Label object is a component for placing text in a container. A label displays a single line of read-only text. The text can be changed by the application, but a user cannot edit it directly.

**LinearGradientPaint**- The LinearGradientPaint class provides a way to fill a [Shape](https://docs.oracle.com/javase/8/docs/api/java/awt/Shape.html) with a linear color gradient pattern.

**List**- The List component presents the user with a scrolling list of text items. The list can be set up so that the user can choose either one item or multiple items.

**MediaTracker**- The MediaTracker class is a utility class to track the status of a number of media objects.

**Menu**- A Menu object is a pull-down menu component that is deployed from a menu bar.

**MenuBar**- The MenuBar class encapsulates the platform's concept of a menu bar bound to a frame. In order to associate the menu bar with a Frame object, call the frame's setMenuBar method.

**MenuComponent**- The abstract class MenuComponent is the superclass of all menu-related components. In this respect, the class MenuComponent is analogous to the abstract superclass Component for AWT components.

**MenuItem**- All items in a menu must belong to the class MenuItem, or one of its subclasses.

**MenuShortcut**- The MenuShortcutclass represents a keyboard accelerator for a MenuItem.

**MouseInfo**- MouseInfo provides methods for getting information about the mouse, such as mouse pointer location and the number of mouse buttons.

**MultipleGradientPaint**- This is the superclass for Paints which use a multiple color gradient to fill in their raster. It provides storage for variables and enumerated values common to LinearGradientPaint and RadialGradientPaint.

**PageAttributes**- A set of attributes which control the output of a printed page.

**PageAttributes.ColorType**- A type-safe enumeration of possible color states.

**Panel**- Panel is the simplest container class. A panel provides space in which an application can attach any other component, including other panels.

**Point**- A point representing a location in (x,y) coordinate space, specified in integer precision.

**PointerInfo** - A class that describes the pointer position. It provides the GraphicsDevice where the pointer is and the Point that represents the coordinates of the pointer.

**Polygon**- The Polygon class encapsulates a description of a closed, two-dimensional region within a coordinate space. This region is bounded by an arbitrary number of line segments, each of which is one side of the polygon.

**PopupMenu**- A class that implements a menu which can be dynamically popped up at a specified position within a component.

**PrintJob**- An abstract class which initiates and executes a print job. It provides access to a print graphics object which renders to an appropriate print device.

**RadialGradientPaint**- The RadialGradientPaint class provides a way to fill a shape with a circular radial color gradient pattern.

**Rectangle**- A Rectangle specifies an area in a coordinate space that is enclosed by the Rectangle object's upper-left point (x,y) in the coordinate space, its width, and its height.

**RenderingHints**- The RenderingHints class defines and manages collections of keys and associated values which allow an application to provide input into the choice of algorithms used by other classes which perform rendering and image manipulation services.

**RenderingHints.Key**- Defines the base type of all keys used along with the [RenderingHints](https://docs.oracle.com/javase/8/docs/api/java/awt/RenderingHints.html" \o "class in java.awt) class to control various algorithm choices in the rendering and imaging pipelines.

**Robot**- This class is used to generate native system input events for the purposes of test automation, self-running demos, and other applications where control of the mouse and keyboard is needed.

**Scrollbar**- The Scrollbar class embodies a scroll bar, a familiar user-interface object.

**ScrollPane**-A container class which implements automatic horizontal and/or vertical scrolling for a single child component.

**ScrollPaneAdjustable**- This class represents the state of a horizontal or vertical scrollbar of a ScrollPane. Objects of this class are returned by ScrollPane methods.

**SplashScreen -** The splash screen can be displayed at application startup, before the Java Virtual Machine (JVM) starts. The splash screen is displayed as an undecorated window containing an image.

**SystemColor**- A class to encapsulate symbolic colors representing the color of native GUI objects on a system. For systems which support the dynamic update of the system colors (when the user changes the colors) the actual RGB values of these symbolic colors will also change dynamically.

**SystemTray**- The SystemTray class represents the system tray for a desktop. On Microsoft Windows it is referred to as the "Taskbar Status Area", on Gnome it is referred to as the "Notification Area", on KDE it is referred to as the "System Tray".

**TextArea**- A TextArea object is a multi-line region that displays text. It can be set to allow editing or to be read-only.

**TextComponent**- The TextComponent class is the superclass of any component that allows the editing of some text.

**TextField**- A TextField object is a text component that allows for the editing of a single line of text.

**TexturePaint**- The TexturePaint class provides a way to fill a [Shape](https://docs.oracle.com/javase/8/docs/api/java/awt/Shape.html) with a texture that is specified as a [BufferedImage](https://docs.oracle.com/javase/8/docs/api/java/awt/image/BufferedImage.html" \o "class in java.awt.image). The size of the BufferedImage object should be small because the BufferedImage data is copied by the TexturePaint object.

**Toolkit**- This class is the abstract superclass of all actual implementations of the Abstract Window Toolkit. Subclasses of the Toolkit class are used to bind the various components to particular native toolkit implementations.

**TrayIcon**- A TrayIcon object represents a tray icon that can be added to the [system tray](https://docs.oracle.com/javase/8/docs/api/java/awt/SystemTray.html). A TrayIcon can have a tooltip (text), an image, a popup menu, and a set of listeners associated with it.

**Window**- A Window object is a top-level window with no borders and no menubar. The default layout for a window is BorderLayout.

**EXCEPTION**

**AWTException**- Signals that an Abstract Window Toolkit exception has occurred.

**FontFormatException**- Thrown by method createFont in the Font class to indicate that the specified font is bad.

**HeadlessException**- Thrown when code that is dependent on a keyboard, display, or mouse is called in an environment that does not support a keyboard, display, or mouse.

**IllegalComponentStateException**- Signals that an AWT component is not in an appropriate

state for the requested operation.

**Packages Description**

**Java**.**awt**.**color** -Provides classes for color spaces.

**CLASSES**

**ColorSpac** - This abstract class is used to serve as a color space tag to identify the specific color space of a Color object or, via a ColorModel object, of an Image, a BufferedImage, or a GraphicsDevice. It contains methods that transform colors in a specific color space to/from sRGB and to/from a well-defined CIEXYZ color space.

**ICC\_ColorSpace-** The ICC\_ColorSpace class is an implementation of the abstract ColorSpace class. This representation of device independent and device dependent color spaces is based on the International Color Consortium Specification ICC.1:2001-12, File Format for Color Profiles

**ICC\_Profile-** A representation of color profile data for device independent and device dependent color spaces based on the International Color Consortium Specification ICC.1:2001-12, File Format for Color Profiles,

**ICC\_ProfileGray-** A subclass of the ICC\_Profile class which represents profiles which meet the following criteria: the color space type of the profile is TYPE\_GRAY and the profile includes the grayTRCTag and mediaWhitePointTag tags.

**ICC\_ProfileRGB-** The ICC\_Profile getInstance method will return an ICC\_ProfileRGB object when these conditions are met. Three-component, matrix-based input profiles and RGB display profiles are examples of this type of profile.

EXCEPTION

**CMMException-** This exception is thrown if the native CMM returns an error.

**ProfileDataException-** This exception is thrown when an error occurs in accessing or processing an ICC\_Profile object.

**Packages Description**

# **java.awt.datatransfer** - Provides interfaces and classes for transferring data between and within applications.

## Interfaces

**ClipboardOwner -** Defines the interface for classes that will provide data to a clipboard. An instance of this interface becomes the owner of the contents of a clipboard (clipboard owner) if it is passed as an argument to [Clipboard.setContents(java.awt.datatransfer.Transferable, java.awt.datatransfer.ClipboardOwner)](https://docs.oracle.com/javase/8/docs/api/java/awt/datatransfer/Clipboard.html" \l "setContents-java.awt.datatransfer.Transferable-java.awt.datatransfer.ClipboardOwner-) method of the clipboard and this method returns successfully. The instance remains the clipboard owner until another application or another object within this application asserts ownership of this clipboard.

**FlavorListener-** Defines an object which listens for [FlavorEvent](https://docs.oracle.com/javase/8/docs/api/java/awt/datatransfer/FlavorEvent.html" \o "class in java.awt.datatransfer)s.

**FlavorMap**- A two-way Map between "natives" (Strings), which correspond to platform- specific data formats, and "flavors" (DataFlavors), which correspond to platform-independent MIME types. FlavorMaps need not be symmetric, but typically are.

**FlavorTable-** A FlavorMap which relaxes the traditional 1-to-1 restriction of a Map. A flavor is permitted to map to any number of natives, and likewise a native is permitted to map to any number of flavors. FlavorTables need not be symmetric, but typically are.

**Transferable-** Defines the interface for classes that can be used to provide data for a transfer operation.For information on using data transfer with Swing, see [How to Use Drag and Drop and Data Transfer](https://docs.oracle.com/javase/tutorial/uiswing/dnd/index.html), a section in The Java Tutorial, for more information.

CLASSES

**Clipboard-** A class that implements a mechanism to transfer data using cut/copy/paste operations.

**DataFlavor-** A DataFlavor provides meta information about data. DataFlavor is typically used to access data on the clipboard, or during a drag and drop operation.

**FlavorEvent-** FlavorEvent is used to notify interested parties that available [DataFlavor](https://docs.oracle.com/javase/8/docs/api/java/awt/datatransfer/DataFlavor.html" \o "class in java.awt.datatransfer)s have changed in the [Clipboard](https://docs.oracle.com/javase/8/docs/api/java/awt/datatransfer/Clipboard.html) (the event source).

**StringSelection-** A Transferable which implements the capability required to transfer a String. This Transferable properly supports DataFlavor.stringFlavor and all equivalent flavors. Support for DataFlavor.plainTextFlavor and all equivalent flavors is **deprecated**. No other DataFlavors are supported.

**SystemFlavorMap-** The SystemFlavorMap is a configurable map between "natives" (Strings), which correspond to platform-specific data formats, and "flavors" (DataFlavors), which correspond to platform-independent MIME types. This mapping is used by the data transfer subsystem to transfer data between Java and native applications, and between Java applications in separate VMs.

**EXCEPTION**

**MimeTypeParseException-** A class to encapsulate MimeType parsing related exceptions

**UnsupportedFlavorException-** Signals that the requested data is not supported in this flavor.

**Packages Description**

**Java.awt.dnd** - Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.

**INTERFACE**

**Autoscroll -** During DnD operations it is possible that a user may wish to drop the subject of the operation on a region of a scrollable GUI control that is not currently visible to the user.

**DragGestureListener-** The listener interface for receiving drag gesture events. This interface is intended for a drag gesture recognition implementation. See a specification for DragGestureRecognizer for details on how to register the listener interface.

**DragSourceListener-** The DragSourceListener defines the event interface for originators of Drag and Drop operations to track the state of the user's gesture, and to provide appropriate "drag over" feedback to the user throughout the Drag and Drop operation.

**DragSourceMotionListener-** A listener interface for receiving mouse motion events during a drag operation.

**DropTargetListener-** The DropTargetListener interface is the callback interface used by the DropTarget class to provide notification of DnD operations that involve the subject DropTarget. Methods of this interface may be implemented to provide "drag under" visual feedback to the user throughout the Drag and Drop operation.

CLASSES

**DnDConstants-** This class contains constant values representing the type of action(s) to be performed by a Drag and Drop operation.

**DragGestureEvent-** A DragGestureEvent is passed to DragGestureListener's dragGestureRecognized() method when a particular DragGestureRecognizer detects that a platform dependent drag initiating gesture has occurred on the Component that it is tracking.

**DragGestureRecognizer-** The DragGestureRecognizer is an abstract base class for the specification of a platform-dependent listener that can be associated with a particular Component in order to identify platform-dependent drag initiating gestures.

**DragSource-** The DragSource is the entity responsible for the initiation of the Drag and Drop operation, and may be used in a number of scenarios

**DragSourceAdapter-** An abstract adapter class for receiving drag source events. The methods in this class are empty. This class exists only as a convenience for creating listener objects.

**DragSourceContext-** The DragSourceContext class is responsible for managing the initiator side of the Drag and Drop protocol. In particular, it is responsible for managing drag event notifications to the [DragSourceListeners](https://docs.oracle.com/javase/8/docs/api/java/awt/dnd/DragSourceListener.html" \o "interface in java.awt.dnd) and [DragSourceMotionListeners](https://docs.oracle.com/javase/8/docs/api/java/awt/dnd/DragSourceMotionListener.html" \o "interface in java.awt.dnd), and providing the [Transferable](https://docs.oracle.com/javase/8/docs/api/java/awt/datatransfer/Transferable.html) representing the source data for the drag operation.

**DragSourceDragEvent-** The DragSourceDragEvent is delivered from the DragSourceContextPeer, via the DragSourceContext, to the DragSourceListener registered with that DragSourceContext and with its associated DragSource.

**DragSourceDropEvent-** The DragSourceDropEvent is delivered from the DragSourceContextPeer, via the DragSourceContext, to the dragDropEnd method of DragSourceListeners registered with that DragSourceContext and with its associated DragSource.

**DragSourceEvent-** This class is the base class for DragSourceDragEvent and DragSourceDropEvent. DragSourceEvents are generated whenever the drag enters, moves over, or exits a drop site, when the drop action changes, and when the drag ends.

**DropTarget-** The DropTarget is associated with a Component when that Component wishes to accept drops during Drag and Drop operations.

**DropTarget.DropTargetAutoScroller-** this protected nested class implements autoscrolling

**DropTargetAdapter-** An abstract adapter class for receiving drop target events. The methods in this class are empty. This class exists only as a convenience for creating listener objects.

**DropTarget-** The DropTarget is associated with a Component when that Component wishes to accept drops during Drag and Drop operations.

**DropTarget.DropTargetAutoScroller-** this protected nested class implements autoscrolling

**DropTargetAdapter-** An abstract adapter class for receiving drop target events. The methods in this class are empty. This class exists only as a convenience for creating listener objects.

**DropTargetContext-** A DropTargetContext is created whenever the logical cursor associated with a Drag and Drop operation coincides with the visible geometry of a Component associated with a DropTarget.

**DropTargetDragEvent-** The DropTargetDragEvent is delivered to a DropTargetListener via its dragEnter() and dragOver() methods. The DropTargetDragEvent reports the *source drop actions* and the *user drop action* that reflect the current state of the drag operation.

**DropTargetDropEvent-** The DropTargetDropEvent is delivered via the DropTargetListener drop() method. The DropTargetDropEvent reports the *source drop actions* and the *user drop action* that reflect the current state of the drag-and-drop operation.

**DragSourceEvent-** This class is the base class for DragSourceDragEvent and DragSourceDropEvent. DragSourceEvents are generated whenever the drag enters, moves over, or exits a drop site, when the drop action changes, and when the drag ends.

**DropTarget-** The DropTarget is associated with a Component when that Component wishes to accept drops during Drag and Drop operations.

**DropTarget.DropTargetAutoScroller-** this protected nested class implements autoscrolling

**DropTargetAdapter-** An abstract adapter class for receiving drop target events. The methods in this class are empty. This class exists only as a convenience for creating listener objects.

**DropTargetContext-** A DropTargetContext is created whenever the logical cursor associated with a Drag and Drop operation coincides with the visible geometry of a Component associated with a DropTarget.

**DropTargetDragEvent-** The DropTargetDragEvent is delivered to a DropTargetListener via its dragEnter() and dragOver() methods. The DropTargetDragEvent reports the *source drop actions* and the *user drop action* that reflect the current state of the drag operation.

**DropTargetEvent-** The DropTargetEvent is the base class for both the DropTargetDragEvent and the DropTargetDropEvent. It encapsulates the current state of the Drag and Drop operations, in particular the current DropTargetContext.

**MouseDragGestureRecognizer-** This abstract subclass of DragGestureRecognizer defines a DragGestureRecognizer for mouse-based gestures. Each platform implements its own concrete subclass of this class, available via the Toolkit.createDragGestureRecognizer() method, to encapsulate the recognition of the platform dependent mouse gesture(s) that initiate a Drag and Drop operation.

**EXCEPTION**

**InvalidDnDOperationException-** This exception is thrown by various methods in the java.awt.dnd package. It is usually thrown to indicate that the target in question is unable to undertake the requested operation that the present time, since the underlying DnD system is not in the appropriate state.

**Packages Description**

**Java.awt.event** - Provides interfaces and classes for dealing with different types of events fired by AWT components.

INTERFACE

**ActionListener-** The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method.

**AdjustmentListener-** The listener interface for receiving adjustment events.

**AWTEventListener-** The listener interface for receiving notification of events dispatched to objects that are instances of Component or MenuComponent or their subclasses.

**ComponentListener-** The listener interface for receiving component events. The class that is interested in processing a component event either implements this interface (and all the methods it contains) or extends the abstract ComponentAdapter class (overriding only the methods of interest).

**ContainerListener-** The listener interface for receiving container events. The class that is interested in processing a container event either implements this interface (and all the methods it contains) or extends the abstract ContainerAdapter class (overriding only the methods of interest).

**FocusListener-** The listener interface for receiving keyboard focus events on a component. The class that is interested in processing a focus event either implements this interface (and all the methods it contains) or extends the abstract FocusAdapter class (overriding only the methods of interest).

**HierarchyBoundsListener-** The listener interface for receiving ancestor moved and resized events. The class that is interested in processing these events either implements this interface (and all the methods it contains) or extends the abstract HierarchyBoundsAdapter class (overriding only the method of interest).

**HierarchyListener-** The listener interface for receiving hierarchy changed events. The class that is interested in processing a hierarchy changed event should implement this interface. The listener object created from that class is then registered with a Component using the Component's addHierarchyListener method.

**InputMethodListener-** The listener interface for receiving input method events. A text editing component has to install an input method event listener in order to work with input methods.

**ItemListener-** The listener interface for receiving item events. The class that is interested in processing an item event implements this interface. The object created with that class is then registered with a component using the component's addItemListener method.

**KeyListener-** The listener interface for receiving keyboard events (keystrokes). The class that is interested in processing a keyboard event either implements this interface (and all the methods it contains) or extends the abstract KeyAdapter class (overriding only the methods of interest).

**ActionEvent-** A semantic event which indicates that a component-defined action occurred. This high-level event is generated by a component (such as a Button) when the component-specific action occurs (such as being pressed).

**AdjustmentEvent-** The adjustment event emitted by Adjustable objects like [Scrollbar](https://docs.oracle.com/javase/8/docs/api/java/awt/Scrollbar.html) and [ScrollPane](https://docs.oracle.com/javase/8/docs/api/java/awt/ScrollPane.html" \o "class in java.awt). When the user changes the value of the scrolling component, it receives an instance of AdjustmentEvent.

**AWTEventListenerProxy-** A class which extends the EventListenerProxy specifically for adding an AWTEventListener for a specific event mask. Instances of this class can be added as AWTEventListeners to a Toolkit object.

**ComponentAdapter-** An abstract adapter class for receiving component events. The methods in this class are empty. This class exists as convenience for creating listener objects.

**ComponentEvent-** A low-level event which indicates that a component moved, changed size, or changed visibility (also, the root class for the other component-level events).

**ContainerAdapter-** An abstract adapter class for receiving container events. The methods in this class are empty. This class exists as convenience for creating listener objects.

**ContainerEvent-** A low-level event which indicates that a container's contents changed because a component was added or removed.

**FocusAdapter-** An abstract adapter class for receiving keyboard focus events. The methods in this class are empty. This class exists as convenience for creating listener objects.

**FocusEvent-** A low-level event which indicates that a Component has gained or lost the input focus. This low-level event is generated by a Component (such as a TextField). The event is passed to every FocusListener or FocusAdapter object which registered to receive such events using the Component's addFocusListener method.

**HierarchyBoundsAdapter-** An abstract adapter class for receiving ancestor moved and resized events. The methods in this class are empty. This class exists as a convenience for creating listener objects.

**HierarchyEvent-** Hierarchy events are provided for notification purposes ONLY. The AWT will automatically handle changes to the hierarchy internally so that GUI layout and displayability works properly regardless of whether a program is receiving these events or not.

**InputEvent-** The root event class for all component-level input events. Input events are delivered to listeners before they are processed normally by the source where they originated.

**InputMethodEvent-** Input method events contain information about text that is being composed using an input method. Whenever the text changes, the input method sends an event.

**InvocationEvent-** An event which executes the run() method on a Runnable when dispatched by the AWT event dispatcher thread. This class can be used as a reference implementation of ActiveEvent rather than declaring a new class and defining dispatch().

**ItemEvent-** A semantic event which indicates that an item was selected or deselected. This high-level event is generated by an ItemSelectable object (such as a List) when an item is selected or deselected by the user.

**KeyAdapter-** An abstract adapter class for receiving keyboard events. The methods in this class are empty. This class exists as convenience for creating listener objects.

**KeyEvent-** An event which indicates that a keystroke occurred in a component.

**MouseAdapter-** An abstract adapter class for receiving mouse events. The methods in this class are empty. This class exists as convenience for creating listener objects.

**MouseEvent-** An event which indicates that a mouse action occurred in a component. A mouse action is considered to occur in a particular component if and only if the mouse cursor is over the unobscured part of the component's bounds when the action happens.

**MouseMotionAdapter-** An abstract adapter class for receiving mouse motion events. The methods in this class are empty. This class exists as convenience for creating listener objects.

**MouseWheelEvent-** An event which indicates that the mouse wheel was rotated in a component.

**PaintEvent-**The component-level paint event. This event is a special type which is used to ensure that paint/update method calls are serialized along with the other events delivered from the event queue. This event is not designed to be used with the Event Listener model; programs should continue to override paint/update methods in order render themselves properly.

**TextEvent-** A semantic event which indicates that an object's text changed. This high-level event is generated by an object (such as a TextComponent) when its text changes. The event is passed to every TextListener object which registered to receive such events using the component's addTextListener method.

**WindowAdapter-** An abstract adapter class for receiving window events. The methods in this class are empty. This class exists as convenience for creating listener objects.

**WindowEvent-** A low-level event that indicates that a window has changed its status. This low-level event is generated by a Window object when it is opened, closed, activated, deactivated, iconified, or deiconified, or when focus is transfered into or out of the Window.

**Packages Description**

**Java.awt.font** - Provides classes and interface relating to fonts.

**INTERFACES**

**MultipleMaster-** The MultipleMaster interface represents Type 1 Multiple Master fonts. A particular [Font](https://docs.oracle.com/javase/8/docs/api/java/awt/Font.html) object can implement this interface.

**OpenType** - The OpenType interface represents OpenType and TrueType fonts. This interface makes it possible to obtain *sfnt* tables from the font. A particular Font object can implement this interface.

CLASSES

**FontRenderContext-** The FontRenderContext class is a container for the information needed to correctly measure text. The measurement of text can vary because of rules that map outlines to pixels, and rendering hints provided by an application.

**GlyphJustificationInfo-** The GlyphJustificationInfo class represents information about the justification properties of a glyph. A glyph is the visual representation of one or more characters.

**GlyphMetrics-** The GlyphMetrics class represents information for a single glyph. A glyph is the visual representation of one or more characters. Many different glyphs can be used to represent a single character or combination of characters.

**GlyphVector-** A GlyphVector object is a collection of glyphs containing geometric information for the placement of each glyph in a transformed coordinate space which corresponds to the device on which the GlyphVector is ultimately displayed.

**GraphicAttribute-** This class is used with the CHAR\_REPLACEMENT attribute.

**LayoutPath-** LayoutPath provides a mapping between locations relative to the baseline and points in user space. Locations consist of an advance along the baseline, and an offset perpendicular to the baseline at the advance.

**LineBreakMeasurer-** The LineBreakMeasurer class allows styled text to be broken into lines (or segments) that fit within a particular visual advance. This is useful for clients who wish to display a paragraph of text that fits within a specific width, called the **wrapping width**.

**LineMetrics-** The LineMetrics class allows access to the metrics needed to layout characters along a line and to layout of a set of lines. A LineMetrics object encapsulates the measurement information associated with a run of text.

**NumericShaper-** The NumericShaper class is used to convert Latin-1 (European) digits to other Unicode decimal digits. Users of this class will primarily be people who wish to present data using national digit shapes, but find it more convenient to represent the data internally using Latin-1 (European) digits.

**ShapeGraphicAttribute-** The ShapeGraphicAttribute class is an implementation of [GraphicAttribute](https://docs.oracle.com/javase/8/docs/api/java/awt/font/GraphicAttribute.html" \o "class in java.awt.font) that draws shapes in a [TextLayout](https://docs.oracle.com/javase/8/docs/api/java/awt/font/TextLayout.html" \o "class in java.awt.font).

**TextAttribute-** The TextAttribute class defines attribute keys and attribute values used for text rendering.

**TextHitInfo-** The TextHitInfo class represents a character position in a text model, and a **bias**, or "side," of the character. Biases are either leading (the left edge, for a left-to-right character) or trailing (the right edge, for a left-to-right character).

**TextLayout-** TextLayout is an immutable graphical representation of styled character data.

**TextLayout.CaretPolicy-** Defines a policy for determining the strong caret location. This class contains one method, getStrongCaret, which is used to specify the policy that determines the strong caret in dual-caret text. The strong caret is used to move the caret to the left or right.

**TextMeasurer-** The TextMeasurer class provides the primitive operations needed for line break: measuring up to a given advance, determining the advance of a range of characters, and generating a TextLayout for a range of characters.

**TransformAttribute-** The TransformAttribute class provides an immutable wrapper for a transform so that it is safe to use as an attribute.

**Packages Description**

**Java.awt.geom** - Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.

**INTERFACE**

**PathIterator-**The PathIterator interface provides the mechanism for objects that implement the [Shape](https://docs.oracle.com/javase/8/docs/api/java/awt/Shape.html) interface to return the geometry of their boundary by allowing a caller to retrieve the path of that boundary a segment at a time. This interface allows these objects to retrieve the path of their boundary a segment at a time by using 1st through 3rd order Bézier curves, which are lines and quadratic or cubic Bézier splines.

CLASSES

**AffineTransform-** The AffineTransform class represents a 2D affine transform that performs a linear mapping from 2D coordinates to other 2D coordinates that preserves the "straightness" and "parallelness" of lines.

**Arc2D-** Arc2D is the abstract superclass for all objects that store a 2D arc defined by a framing rectangle, start angle, angular extent (length of the arc), and a closure type (OPEN, CHORD, or PIE).

**Arc2D.Double-** This class defines an arc specified in double precision.

**Arc2D.Float-** This class defines an arc specified in float precision.

**Area-** An Area object stores and manipulates a resolution-independent description of an enclosed area of 2-dimensional space. Area objects can be transformed and can perform various Constructive Area Geometry (CAG) operations when combined with other Area objects.

**CubicCurve2D-** The CubicCurve2D class defines a cubic parametric curve segment in (x,y) coordinate space.

**CubicCurve2D.Double-** A cubic parametric curve segment specified with double coordinates.

**CubicCurve2D.Float-** A cubic parametric curve segment specified with float coordinates.

**Dimension2D-** The Dimension2D class is to encapsulate a width and a height dimension.This class is only the abstract superclass for all objects that store a 2D dimension. The actual storage representation of the sizes is left to the subclass.

**Ellipse2D-** The Ellipse2D class describes an ellipse that is defined by a framing rectangle.

**Ellipse2D.Double-** The Double class defines an ellipse specified in double precision.

**Ellipse2D.Float-** The Float class defines an ellipse specified in float precision.

**FlatteningPathIterator-**The FlatteningPathIterator class returns a flattened view of another [PathIterator](https://docs.oracle.com/javase/8/docs/api/java/awt/geom/PathIterator.html" \o "interface in java.awt.geom) object. Other [Shape](https://docs.oracle.com/javase/8/docs/api/java/awt/Shape.html) classes can use this class to provide flattening behavior for their paths without having to perform the interpolation calculations themselves.

**GeneralPath-** The GeneralPath class represents a geometric path constructed from straight lines, and quadratic and cubic (Bézier) curves. It can contain multiple subpaths.

**Line2D-** This Line2D represents a line segment in (x,y) coordinate space. This class, like all of the Java 2D API, uses a default coordinate system called *user space* in which the y-axis values increase downward and x-axis values increase to the right.

**Line2D.Double-** A line segment specified with double coordinates.

**Line2D.Float-** A line segment specified with float coordinates.

**Path2D-** The Path2D class provides a simple, yet flexible shape which represents an arbitrary geometric path. It can fully represent any path which can be iterated by the [PathIterator](https://docs.oracle.com/javase/8/docs/api/java/awt/geom/PathIterator.html" \o "interface in java.awt.geom) interface including all of its segment types and winding rules and it implements all of the basic hit testing methods of the [Shape](https://docs.oracle.com/javase/8/docs/api/java/awt/Shape.html) interface.

**Path2D.Double-** The Double class defines a geometric path with coordinates stored in double precision floating point.

**Path2D.float-** The Double class defines a geometric path with coordinates stored in double precision floating point.

**QuadCurve2D-**The QuadCurve2D class defines a quadratic parametric curve segment in (x,y) coordinate space.

**QuadCurve2D.Double-** A quadratic parametric curve segment specified with double coordinates.

**QuadCurve2D.float-**A quadratic parametric curve segment specified with double coordinates.

**Rectangle2D-** The Rectangle2D class describes a rectangle defined by a location (x,y) and dimension (w x h).

**Rectangle2D.Double-** The Double class defines a rectangle specified in double coordinates.

**Rectangle2D.float-** The Double class defines a rectangle specified in double coordinates.

**RectangularShape-** RectangularShape is the base class for a number of [Shape](https://docs.oracle.com/javase/8/docs/api/java/awt/Shape.html) objects whose geometry is defined by a rectangular frame. This class does not directly specify any specific geometry by itself, but merely provides manipulation methods inherited by a whole category of Shape objects.

**RoundRectangle2D-** The RoundRectangle2D class defines a rectangle with rounded corners defined by a location (x,y), a dimension (w x h), and the width and height of an arc with which to round the corners.

**RoundRectangle2D.Double-** The Double class defines a rectangle with rounded corners all specified in double coordinates.

**RoundRectangle2D.float-** The Double class defines a rectangle with rounded corners all specified in double coordinates.

EXCEPTIONS

**IllegalPathStateException-** The IllegalPathStateException represents an exception that is thrown if an operation is performed on a path that is in an illegal state with respect to the particular operation being performed, such as appending a path segment to a [GeneralPath](https://docs.oracle.com/javase/8/docs/api/java/awt/geom/GeneralPath.html" \o "class in java.awt.geom) without an initial moveto.

**NoninvertibleTransformException-** The NoninvertibleTransformException class represents an exception that is thrown if an operation is performed requiring the inverse of an [AffineTransform](https://docs.oracle.com/javase/8/docs/api/java/awt/geom/AffineTransform.html" \o "class in java.awt.geom) object but the AffineTransform is in a non-invertible state.

**Packages Description**

**Java.awt.im** - Provides classes and interfaces for the input method framework.

INTERFACE

**InputMethodRequests**- InputMethodRequests defines the requests that a text editing component has to handle in order to work with input methods. The component can implement this interface itself or use a separate object that implements it. The object implementing this interface must be returned from the component's getInputMethodRequests method.

CLASSES

**InputContext-** Provides methods to control text input facilities such as input methods and keyboard layouts. Two methods handle both input methods and keyboard layouts: selectInputMethod lets a client component select an input method or keyboard layout by locale, getLocale lets a client component obtain the locale of the current input method or keyboard layout.

**InputMethodHighlight-** An InputMethodHighlight is used to describe the highlight attributes of text being composed. The description can be at two levels: at the abstract level it specifies the conversion state and whether the text is selected; at the concrete level it specifies style attributes used to render the highlight.

**InputSubset-** Defines additional Unicode subsets for use by input methods. Unlike the UnicodeBlock subsets defined in the [Character.UnicodeBlock](https://docs.oracle.com/javase/8/docs/api/java/lang/Character.UnicodeBlock.html" \o "class in java.lang) class, these constants do not directly correspond to Unicode code blocks.

**Packages Description**

**Java.awt.im.spi** - Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.

INTERFACE

**InputMethod**- Defines the interface for an input method that supports complex text input. Input methods traditionally support text input for languages that have more characters than can be represented on a standard-size keyboard, such as Chinese, Japanese, and Korean. However, they may also be used to support phonetic text input for English or character reordering for Thai.

**InputMethodContext-** Provides methods that input methods can use to communicate with their client components or to request other services. This interface is implemented by the input method framework, and input methods call its methods on the instance they receive through [InputMethod.setInputMethodContext(java.awt.im.spi.InputMethodContext)](https://docs.oracle.com/javase/8/docs/api/java/awt/im/spi/InputMethod.html#setInputMethodContext-java.awt.im.spi.InputMethodContext-). There should be no other implementors or callers.

**InputMethodDescriptor**- Defines methods that provide sufficient information about an input method to enable selection and loading of that input method. The input method itself is only loaded when it is actually used.

**Packages Description**

**Java.awt.image**- Provides classes for creating and modifying images.

INTERFACE

**BufferedImageOp**- This interface describes single-input/single-output operations performed on BufferedImage objects. It is implemented by AffineTransformOp, ConvolveOp, ColorConvertOp, RescaleOp, and LookupOp.

**ImageConsumer**- The interface for objects expressing interest in image data through the ImageProducer interfaces. When a consumer is added to an image producer, the producer delivers all of the data about the image using the method calls defined in this interface.

**ImageObserver-** An asynchronous update interface for receiving notifications about Image information as the Image is constructed.

**ImageProducer**

The interface for objects which can produce the image data for Images. Each image contains an ImageProducer which is used to reconstruct the image whenever it is needed, for example, when a new size of the Image is scaled, or when the width or height of the Image is being requested.

**RasterOp**- This interface describes single-input/single-output operations performed on Raster objects. It is implemented by such classes as AffineTransformOp, ConvolveOp, and LookupOp.

**RenderedImage**- RenderedImage is a common interface for objects which contain or can produce image data in the form of Rasters. The image data may be stored/produced as a single tile or a regular array of tiles.

**TileObserver**- An interface for objects that wish to be informed when tiles of a WritableRenderedImage become modifiable by some writer via a call to getWritableTile, and when they become unmodifiable via the last call to releaseWritableTile.

**WritableRenderedImage-**WriteableRenderedImage is a common interface for objects which contain or can produce image data in the form of Rasters and which can be modified and/or written over. The image data may be stored/produced as a single tile or a regular array of tiles.

CLASSES

**AffineTransformOp-**This class uses an affine transform to perform a linear mapping from 2D coordinates in the source image or Raster to 2D coordinates in the destination image or Raster.

**AreaAveragingScaleFilter-** An ImageFilter class for scaling images using a simple area averaging algorithm that produces smoother results than the nearest neighbor algorithm.

**BandCombineOp-** This class performs an arbitrary linear combination of the bands in a Raster, using a specified matrix.

**BandedSampleModel-** This class represents image data which is stored in a band interleaved fashion and for which each sample of a pixel occupies one data element of the DataBuffer. It subclasses ComponentSampleModel but provides a more efficient implementation for accessing band interleaved image data than is provided by ComponentSampleModel.

**BufferedImage-** The BufferedImage subclass describes an [Image](https://docs.oracle.com/javase/8/docs/api/java/awt/Image.html) with an accessible buffer of image data. A BufferedImage is comprised of a [ColorModel](https://docs.oracle.com/javase/8/docs/api/java/awt/image/ColorModel.html" \o "class in java.awt.image) and a [Raster](https://docs.oracle.com/javase/8/docs/api/java/awt/image/Raster.html) of image data. The number and types of bands in the [SampleModel](https://docs.oracle.com/javase/8/docs/api/java/awt/image/SampleModel.html" \o "class in java.awt.image) of the Raster must match the number and types required by the ColorModel to represent its color and alpha components.

**BufferedImageFilter-** The BufferedImageFilter class subclasses an ImageFilter to provide a simple means of using a single-source/single-destination image operator ([BufferedImageOp](https://docs.oracle.com/javase/8/docs/api/java/awt/image/BufferedImageOp.html" \o "interface in java.awt.image)) to filter a BufferedImage in the Image Producer/Consumer/Observer paradigm. Examples of these image operators are: [ConvolveOp](https://docs.oracle.com/javase/8/docs/api/java/awt/image/ConvolveOp.html" \o "class in java.awt.image), [AffineTransformOp](https://docs.oracle.com/javase/8/docs/api/java/awt/image/AffineTransformOp.html" \o "class in java.awt.image) and [LookupOp](https://docs.oracle.com/javase/8/docs/api/java/awt/image/LookupOp.html" \o "class in java.awt.image).

**BufferStrategy-** The BufferStrategy class represents the mechanism with which to organize complex memory on a particular Canvas or Window. Hardware and software limitations determine whether and how a particular buffer strategy can be implemented.

**ByteLookupTable-** This class defines a lookup table object. The output of a lookup operation using an object of this class is interpreted as an unsigned byte quantity. The lookup table contains byte data arrays for one or more bands (or components) of an image, and it contains an offset which will be subtracted from the input values before indexing the arrays.

**ColorConvertOp-**This class performs a pixel-by-pixel color conversion of the data in the source image. The resulting color values are scaled to the precision of the destination image. Color conversion can be specified via an array of ColorSpace objects or an array of ICC\_Profile objects.

**ColorModel-** The ColorModel abstract class encapsulates the methods for translating a pixel value to color components (for example, red, green, and blue) and an alpha component.

**ComponentColorModel-** A ColorModel class that works with pixel values that represent color and alpha information as separate samples and that store each sample in a separate data element.

**ComponentSampleModel-** This class represents image data which is stored such that each sample of a pixel occupies one data element of the DataBuffer. It stores the N samples which make up a pixel in N separate data array elements. Different bands may be in different banks of the DataBuffer. Accessor methods are provided so that image data can be manipulated directly.

**ConvolveOp-** This class implements a convolution from the source to the destination. Convolution using a convolution kernel is a spatial operation that computes the output pixel from an input pixel by multiplying the kernel with the surround of the input pixel.

**CropImageFilter-** An ImageFilter class for cropping images. This class extends the basic ImageFilter Class to extract a given rectangular region of an existing Image and provide a source for a new image containing just the extracted region.

**DataBuffer-** This class exists to wrap one or more data arrays. Each data array in the DataBuffer is referred to as a bank. Accessor methods for getting and setting elements of the DataBuffer's banks exist with and without a bank specifier.

**DataBufferByte-** This class extends DataBuffer and stores data internally as bytes. Values stored in the byte array(s) of this DataBuffer are treated as unsigned values.

**DataBufferDouble-** This class extends DataBuffer and stores data internally in double form.

**DataBufferFloat-** This class extends DataBuffer and stores data internally in float form.

**DataBufferInt-**This class extends DataBuffer and stores data internally as integers.

**DataBufferShort-**This class extends DataBuffer and stores data internally as shorts.

**DataBufferUShort-** This class extends DataBuffer and stores data internally as shorts. Values stored in the short array(s) of this DataBuffer are treated as unsigned values.

**DirectColorModel-**The DirectColorModel class is a ColorModel class that works with pixel values that represent RGB color and alpha information as separate samples and that pack all samples for a single pixel into a single int, short, or byte quantity.

**FilteredImageSource-** This class is an implementation of the ImageProducer interface which takes an existing image and a filter object and uses them to produce image data for a new filtered version of the original image

**ImageFilter-** This class implements a filter for the set of interface methods that are used to deliver data from an ImageProducer to an ImageConsumer. It is meant to be used in conjunction with a FilteredImageSource object to produce filtered versions of existing images.

**IndexColorModel-** The IndexColorModel class is a ColorModel class that works with pixel values consisting of a single sample that is an index into a fixed colormap in the default sRGB color space.

**Kernel-** The Kernel class defines a matrix that describes how a specified pixel and its surrounding pixels affect the value computed for the pixel's position in the output image of a filtering operation. The X origin and Y origin indicate the kernel matrix element that corresponds to the pixel position for which an output value is being computed.

**LookupOp-** This class implements a lookup operation from the source to the destination. The LookupTable object may contain a single array or multiple arrays, subject to the restrictions below.

**LookupTable-** This abstract class defines a lookup table object. ByteLookupTable and ShortLookupTable are subclasses, which contain byte and short data, respectively.

**MemoryImageSource-** This class is an implementation of the ImageProducer interface which uses an array to produce pixel values for an Image. Here is an example which calculates a 100x100 image representing a fade from black to blue along the X axis and a fade from black to red along the Y axis:

**MultiPixelPackedSampleModel-** The MultiPixelPackedSampleModel class represents one-banded images and can pack multiple one-sample pixels into one data element. Pixels are not allowed to span data elements.

**PackedColorModel-** The PackedColorModel class is an abstract [ColorModel](https://docs.oracle.com/javase/8/docs/api/java/awt/image/ColorModel.html" \o "class in java.awt.image) class that works with pixel values which represent color and alpha information as separate samples and which pack all samples for a single pixel into a single int, short, or byte quantity

**PixelGrabber-** The PixelGrabber class implements an ImageConsumer which can be attached to an Image or ImageProducer object to retrieve a subset of the pixels in that image.

**PixelInterleavedSampleModel-** This class represents image data which is stored in a pixel interleaved fashion and for which each sample of a pixel occupies one data element of the DataBuffer. It subclasses ComponentSampleModel but provides a more efficient implementation for accessing pixel interleaved image data than is provided by ComponentSampleModel.

**Raster-** A class representing a rectangular array of pixels. A Raster encapsulates a DataBuffer that stores the sample values and a SampleModel that describes how to locate a given sample value in a DataBuffer.

**ReplicateScaleFilter-** An ImageFilter class for scaling images using the simplest algorithm. This class extends the basic ImageFilter Class to scale an existing image and provide a source for a new image containing the resampled image

**RescaleOp-** This class performs a pixel-by-pixel rescaling of the data in the source image by multiplying the sample values for each pixel by a scale factor and then adding an offset. The scaled sample values are clipped to the minimum/maximum representable in the destination image.

**RGBImageFilter-** This class provides an easy way to create an ImageFilter which modifies the pixels of an image in the default RGB ColorModel. It is meant to be used in conjunction with a FilteredImageSource object to produce filtered versions of existing images.

**SampleModel-** This abstract class defines an interface for extracting samples of pixels in an image. All image data is expressed as a collection of pixels. Each pixel consists of a number of samples. A sample is a datum for one band of an image and a band consists of all samples of a particular type in an image.

**ShortLookupTable-** This class defines a lookup table object. The output of a lookup operation using an object of this class is interpreted as an unsigned short quantity. The lookup table contains short data arrays for one or more bands (or components) of an image, and it contains an offset which will be subtracted from the input values before indexing the arrays.

**SinglePixelPackedSampleModel-** This class represents pixel data packed such that the N samples which make up a single pixel are stored in a single data array element, and each data data array element holds samples for only one pixel. This class supports [TYPE\_BYTE](https://docs.oracle.com/javase/8/docs/api/java/awt/image/DataBuffer.html#TYPE_BYTE), [TYPE\_USHORT](https://docs.oracle.com/javase/8/docs/api/java/awt/image/DataBuffer.html#TYPE_USHORT), [TYPE\_INT](https://docs.oracle.com/javase/8/docs/api/java/awt/image/DataBuffer.html#TYPE_INT) data types. All data array elements reside in the first bank of a DataBuffer.

**VolatileImage-** VolatileImage is an image which can lose its contents at any time due to circumstances beyond the control of the application (e.g., situations caused by the operating system or by other applications). Because of the potential for hardware acceleration, a VolatileImage object can have significant performance benefits on some platforms.

**WritableRaster-** This class extends Raster to provide pixel writing capabilities. Refer to the class comment for Raster for descriptions of how a Raster stores pixels.

EXCEPTION

**ImagingOpException-** The ImagingOpException is thrown if one of the [BufferedImageOp](https://docs.oracle.com/javase/8/docs/api/java/awt/image/BufferedImageOp.html" \o "interface in java.awt.image) or [RasterOp](https://docs.oracle.com/javase/8/docs/api/java/awt/image/RasterOp.html" \o "interface in java.awt.image) filter methods cannot process the image.

**RasterFormatException-** The RasterFormatException is thrown if there is invalid layout information in the [Raster](https://docs.oracle.com/javase/8/docs/api/java/awt/image/Raster.html).

**Packages Description**

**Java.awt.image.renderable**- Provides classes and interfaces for producing rendering-independent images.

INTERFACE

**ContextualRenderedImageFactory-** ContextualRenderedImageFactory provides an interface for the functionality that may differ between instances of RenderableImageOp. Thus different operations on RenderableImages may be performed by a single class such as RenderedImageOp through the use of multiple instances of ContextualRenderedImageFactory.

**RenderableImage**- A RenderableImage is a common interface for rendering-independent images (a notion which subsumes resolution independence). That is, images which are described and have operations applied to them independent of any specific rendering of the image. For example, a RenderableImage can be rotated and cropped in resolution-independent terms.

**RenderedImageFactory**- The RenderedImageFactory interface (often abbreviated RIF) is intended to be implemented by classes that wish to act as factories to produce different renderings, for example by executing a series of BufferedImageOps on a set of sources, depending on a specific set of parameters, properties, and rendering hints.

CLASSES

**ParameterBlock-** A ParameterBlock encapsulates all the information about sources and parameters (Objects) required by a RenderableImageOp, or other classes that process images.

**RenderableImageOp-**This class handles the renderable aspects of an operation with help from its associated instance of a ContextualRenderedImageFactory.

**RenderableImageProducer-** An adapter class that implements ImageProducer to allow the asynchronous production of a RenderableImage. The size of the ImageConsumer is determined by the scale factor of the usr2dev transform in the RenderContext. If the RenderContext is null, the default rendering of the RenderableImage is used

**RenderContext-** A RenderContext encapsulates the information needed to produce a specific rendering from a RenderableImage. It contains the area to be rendered specified in rendering-independent terms, the resolution at which the rendering is to be performed, and hints used to control the rendering process.

**Packages Description**

**Java.awt.print**- Provides classes and interfaces for a general printing API.

INTERFACE

**Pageable**- The Pageable implementation represents a set of pages to be printed. The Pageable object returns the total number of pages in the set as well as the [PageFormat](https://docs.oracle.com/javase/8/docs/api/java/awt/print/PageFormat.html" \o "class in java.awt.print) and [Printable](https://docs.oracle.com/javase/8/docs/api/java/awt/print/Printable.html) for a specified page.

**Printable**- The Printable interface is implemented by the print methods of the current page painter, which is called by the printing system to render a page. When building a [Pageable](https://docs.oracle.com/javase/8/docs/api/java/awt/print/Pageable.html), pairs of [PageFormat](https://docs.oracle.com/javase/8/docs/api/java/awt/print/PageFormat.html" \o "class in java.awt.print) instances and instances that implement this interface are used to describe each page.

**PrinterGraphics**- The PrinterGraphics interface is implemented by [Graphics](https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics.html) objects that are passed to [Printable](https://docs.oracle.com/javase/8/docs/api/java/awt/print/Printable.html) objects to render a page. It allows an application to find the [PrinterJob](https://docs.oracle.com/javase/8/docs/api/java/awt/print/PrinterJob.html" \o "class in java.awt.print) object that is controlling the printing.

CLASSES

**Book-** The Book class provides a representation of a document in which pages may have different page formats and page painters. This class uses the [Pageable](https://docs.oracle.com/javase/8/docs/api/java/awt/print/Pageable.html) interface to interact with a [PrinterJob](https://docs.oracle.com/javase/8/docs/api/java/awt/print/PrinterJob.html" \o "class in java.awt.print).

**PageFormat-** The PageFormat class describes the size and orientation of a page to be printed.

**Paper-**The Paper class describes the physical characteristics of a piece of paper.

**PrinterJob-** The PrinterJob class is the principal class that controls printing. An application calls methods in this class to set up a job, optionally to invoke a print dialog with the user, and then to print the pages of the job.

EXCEPTIONS

**PrinterAbortException-** The PrinterAbortException class is a subclass of [PrinterException](https://docs.oracle.com/javase/8/docs/api/java/awt/print/PrinterException.html" \o "class in java.awt.print) and is used to indicate that a user or application has terminated the print job while it was in the process of printing.

**PrinterException-** The PrinterException class and its subclasses are used to indicate that an exceptional condition has occurred in the print system.

**PrinterIOException-** The PrinterIOException class is a subclass of [PrinterException](https://docs.oracle.com/javase/8/docs/api/java/awt/print/PrinterException.html" \o "class in java.awt.print) and is used to indicate that an IO error of some sort has occurred while printing.

**Packages Description**

**Java.beans** - Contains classes related to developing beans -- components based on the JavaBeans™ architecture.

INTERFACE

**AppletInitializer-**This interface is designed to work in collusion with java.beans.Beans.instantiate. The interface is intended to provide mechanism to allow the proper initialization of JavaBeans that are also Applets, during their instantiation by java.beans.Beans.instantiate().

**BeanInfo**- Use the BeanInfo interface to create a BeanInfo class and provide explicit information about the methods, properties, events, and other features of your beans.

**Customizer**- A customizer class provides a complete custom GUI for customizing a target Java Bean.Each customizer should inherit from the java.awt.Component class so it can be instantiated inside an AWT dialog or panel.Each customizer should have a null constructor.

**DesignMode-** This interface is intended to be implemented by, or delegated from, instances of java.beans.beancontext.BeanContext, in order to propagate to its nested hierarchy of java.beans.beancontext.BeanContextChild instances, the current "designTime" property.

**ExceptionListener**- An ExceptionListener is notified of internal exceptions.

**PropertyChangeListener-** A "PropertyChange" event gets fired whenever a bean changes a "bound" property. You can register a PropertyChangeListener with a source bean so as to be notified of any bound property updates.

**PropertyEditor**- A PropertyEditor class provides support for GUIs that want to allow users to edit a property value of a given type.

**VetoableChangeListener-** A VetoableChange event gets fired whenever a bean changes a "constrained" property. You can register a VetoableChangeListener with a source bean so as to be notified of any constrained property updates.

**Visibility**- Under some circumstances a bean may be run on servers where a GUI is not available. This interface can be used to query a bean to determine whether it absolutely needs a gui, and to advise the bean whether a GUI is available.

CLASSES

**BeanDescriptor-** A BeanDescriptor provides global information about a "bean", including its Java class, its displayName, etc.

**Beans-** This class provides some general purpose beans control methods.

**Beans-** This class provides some general purpose beans control methods.

**Encoder-** An Encoder is a class which can be used to create files or streams that encode the state of a collection of JavaBeans in terms of their public APIs.

**EventHandler-** The EventHandler class provides support for dynamically generating event listeners whose methods execute a simple statement involving an incoming event object and a target object.

**EventSetDescriptor-** An EventSetDescriptor describes a group of events that a given Java bean fires.

**Expression-**An Expression object represents a primitive expression in which a single method is applied to a target and a set of arguments to return a result - as in "a.getFoo()".

**FeatureDescriptor-** The FeatureDescriptor class is the common baseclass for PropertyDescriptor, EventSetDescriptor, and MethodDescriptor, etc.

**IndexedPropertyChangeEvent-** An "IndexedPropertyChange" event gets delivered whenever a component that conforms to the JavaBeans™ specification (a "bean") changes a bound indexed property. This class is an extension of PropertyChangeEvent but contains the index of the property that has changed.

**IndexedPropertyDescriptor-** An IndexedPropertyDescriptor describes a property that acts like an array and has an indexed read and/or indexed write method to access specific elements of the array.

**PersistenceDelegate-** The PersistenceDelegate class takes the responsibility for expressing the state of an instance of a given class in terms of the methods in the class's public API.

**PropertyChangeEvent-** A "PropertyChange" event gets delivered whenever a bean changes a "bound" or "constrained" property. A PropertyChangeEvent object is sent as an argument to the PropertyChangeListener and VetoableChangeListener methods.

**PropertyChangeListenerProxy-** A class which extends the EventListenerProxy specifically for adding a PropertyChangeListener with a "bound" property. Instances of this class can be added as PropertyChangeListeners to a bean which supports firing property change events.

**PropertyChangeSupport-** This is a utility class that can be used by beans that support bound properties. It manages a list of listeners and dispatches [PropertyChangeEvent](https://docs.oracle.com/javase/8/docs/api/java/beans/PropertyChangeEvent.html" \o "class in java.beans)s to them. You can use an instance of this class as a member field of your bean and delegate these types of work to it. The [PropertyChangeListener](https://docs.oracle.com/javase/8/docs/api/java/beans/PropertyChangeListener.html" \o "interface in java.beans) can be registered for all properties or for a property specified by name.

**PropertyDescriptor-** A PropertyDescriptor describes one property that a Java Bean exports via a pair of accessor methods.

**PropertyEditorManager-** The PropertyEditorManager can be used to locate a property editor for any given type name. This property editor must support the java.beans.PropertyEditor interface for editing a given object.

PropertyEditorSupport-This is a support class to help build property editors.It can be used either as a base class or as a delegate.

**SimpleBeanInfo-** This is a support class to make it easier for people to provide BeanInfo classes.

**Statement-** A Statement object represents a primitive statement in which a single method is applied to a target and a set of arguments - as in "a.setFoo(b)".

**VetoableChangeListenerProxy-** A class which extends the EventListenerProxy specifically for adding a VetoableChangeListener with a "constrained" property. Instances of this class can be added as VetoableChangeListeners to a bean which supports firing vetoable change events.

**VetoableChangeSupport-** This is a utility class that can be used by beans that support constrained properties. It manages a list of listeners and dispatches [PropertyChangeEvent](https://docs.oracle.com/javase/8/docs/api/java/beans/PropertyChangeEvent.html" \o "class in java.beans)s to them. You can use an instance of this class as a member field of your bean and delegate these types of work to it.

**XMLDecoder-** The XMLDecoder class is used to read XML documents created using the XMLEncoder and is used just like the ObjectInputStream.

**XMLEncoder** - The XMLEncoder class is a complementary alternative to the ObjectOutputStream and can used to generate a textual representation of a JavaBean in the same way that the ObjectOutputStream can be used to create binary representation of Serializable objects.

EXCEPTION

**IntrospectionException-** Thrown when an exception happens during Introspection. Typical causes include not being able to map a string class name to a Class object, not being able to resolve a string method name, or specifying a method name that has the wrong type signature for its intended use.

**PropertyVetoException-** A PropertyVetoException is thrown when a proposed change to a property represents an unacceptable value.

**Packages Description**

**Java.beans.beancontext -** Provides classes and interfaces relating to bean context.

INTERFACE

**BeanContext-** The BeanContext acts a logical hierarchical container for JavaBeans.

**BeanContextChild-** JavaBeans wishing to be nested within, and obtain a reference to their execution environment, or context, as defined by the BeanContext sub-interface shall implement this interface.

**BeanContextChildComponentProxy-** This interface is implemented by BeanContextChildren that have an AWT Component associated with them.

**BeanContextContainerProxy-** This interface is implemented by BeanContexts' that have an AWT Container associated with them.

**BeanContextMembershipListener-** Compliant BeanContexts fire events on this interface when the state of the membership of the BeanContext changes.

**BeanContextProxy-** This interface is implemented by a JavaBean that does not directly have a BeanContext(Child) associated with it (via implementing that interface or a subinterface thereof), but has a public BeanContext(Child) delegated from it.

**BeanContextServiceProvider-** One of the primary functions of a BeanContext is to act a as rendezvous between JavaBeans, and BeanContextServiceProviders.

**BeanContextServiceProviderBeanInfo-** A BeanContextServiceProvider implementor who wishes to provide explicit information about the services their bean may provide shall implement a BeanInfo class that implements this BeanInfo subinterface and provides explicit information about the methods, properties, events, etc, of their services.

**BeanContextServiceRevokedListener-** The listener interface for receiving BeanContextServiceRevokedEvent objects. A class that is interested in processing a BeanContextServiceRevokedEvent implements this interface.

**BeanContextServices-** The BeanContextServices interface provides a mechanism for a BeanContext to expose generic "services" to the BeanContextChild objects within.

**BeanContextServicesListener-** The listener interface for receiving BeanContextServiceAvailableEvent objects. A class that is interested in processing a BeanContextServiceAvailableEvent implements this interface.

CLASSES

**BeanContextChildSupport-** This is a general support class to provide support for implementing the BeanContextChild protocol. This class may either be directly subclassed, or encapsulated and delegated to in order to implement this interface for a given component.

**BeanContextEvent-** BeanContextEvent is the abstract root event class for all events emitted from, and pertaining to the semantics of, a BeanContext. This class introduces a mechanism to allow the propagation of BeanContextEvent subclasses through a hierarchy of BeanContexts. The setPropagatedFrom() and getPropagatedFrom() methods allow a BeanContext to identify itself as the source of a propagated event.

**beanContextMembershipEvent-** A BeanContextMembershipEvent encapsulates the list of children added to, or removed from, the membership of a particular BeanContext. An instance of this event is fired whenever a successful add(), remove(), retainAll(), removeAll(), or clear() is invoked on a given BeanContext instance.

**BeanContextServiceAvailableEvent-** This event type is used by the BeanContextServicesListener in order to identify the service being registered.

**BeanContextServiceRevokedEvent-** This event type is used by the BeanContextServiceRevokedListener in order to identify the service being revoked.

**BeanContextServicesSupport-** This helper class provides a utility implementation of the java.beans.beancontext.BeanContextServices interface.

**BeanContextServicesSupport.BCSSServiceProvider-** subclasses may subclass this nested class to add behaviors for each BeanContextServicesProvider.

**BeanContextSupport-** This helper class provides a utility implementation of the java.beans.beancontext.BeanContext interface.

**BeanContextSupport.BCSIterator-** protected final subclass that encapsulates an iterator but implements a noop remove() method.

**Packages Description**

**Java.io -** Provides for system input and output through data streams, serialization and the file system.

INTERFACES

**Closeable-** A Closeable is a source or destination of data that can be closed. The close method is invoked to release resources that the object is holding (such as open files).

**DataInput-** The DataInput interface provides for reading bytes from a binary stream and reconstructing from them data in any of the Java primitive types. There is also a facility for reconstructing a String from data in [modified UTF-8](https://docs.oracle.com/javase/8/docs/api/java/io/DataInput.html#modified-utf-8) format.

**DataOutput**- The DataOutput interface provides for converting data from any of the Java primitive types to a series of bytes and writing these bytes to a binary stream.

**Externalizable-** Only the identity of the class of an Externalizable instance is written in the serialization stream and it is the responsibility of the class to save and restore the contents of its instances.

**FileFilter**- A filter for abstract pathnames. Instances of this interface may be passed to the [listFiles(FileFilter)](https://docs.oracle.com/javase/8/docs/api/java/io/File.html" \l "listFiles-java.io.FileFilter-) method of the [File](https://docs.oracle.com/javase/8/docs/api/java/io/File.html) class.

**FilenameFilter**- Instances of classes that implement this interface are used to filter filenames. These instances are used to filter directory listings in the list method of class File, and by the Abstract Window Toolkit's file dialog component.

**Flushable**- A Flushable is a destination of data that can be flushed. The flush method is invoked to write any buffered output to the underlying stream.

**ObjectInput-** ObjectInput extends the DataInput interface to include the reading of objects. DataInput includes methods for the input of primitive types, ObjectInput extends that interface to include objects, arrays, and Strings.

**ObjectInputValidation**- Callback interface to allow validation of objects within a graph. Allows an object to be called when a complete graph of objects has been deserialized.

**ObjectOutput-** ObjectOutput extends the DataOutput interface to include writing of objects. DataOutput includes methods for output of primitive types, ObjectOutput extends that interface to include objects, arrays, and Strings.

**ObjectStreamConstants**- Constants written into the Object Serialization Stream.

**Serializable**- Serializability of a class is enabled by the class implementing the java.io.Serializable interface. Classes that do not implement this interface will not have any of their state serialized or deserialized.

CLASSES

**BufferedInputStream-** A BufferedInputStream adds functionality to another input stream-namely, the ability to buffer the input and to support the mark and reset methods.

**BufferedOutputStream-** The class implements a buffered output stream. By setting up such an output stream, an application can write bytes to the underlying output stream without necessarily causing a call to the underlying system for each byte written.

**BufferedReader-** Reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines.

**BufferedWriter-**Writes text to a character-output stream, buffering characters so as to provide for the efficient writing of single characters, arrays, and strings.

**ByteArrayInputStream-** A ByteArrayInputStream contains an internal buffer that contains bytes that may be read from the stream. An internal counter keeps track of the next byte to be supplied by the read method.

**ByteArrayOutputStream-** This class implements an output stream in which the data is written into a byte array. The buffer automatically grows as data is written to it. The data can be retrieved using toByteArray() and toString().

**CharArrayReader-** This class implements a character buffer that can be used as a character-input stream.

**CharArrayWriter-** This class implements a character buffer that can be used as an Writer. The buffer automatically grows when data is written to the stream. The data can be retrieved using toCharArray() and toString().

**Console-** Methods to access the character-based console device, if any, associated with the current Java virtual machine.

**DataInputStream-** A data input stream lets an application read primitive Java data types from an underlying input stream in a machine-independent way. An application uses a data output stream to write data that can later be read by a data input stream.

**DataOutputStream-** A data output stream lets an application write primitive Java data types to an output stream in a portable way. An application can then use a data input stream to read the data back in.

**File-** An abstract representation of file and directory pathnames.

**FileDescriptor-** Instances of the file descriptor class serve as an opaque handle to the underlying machine-specific structure representing an open file, an open socket, or another source or sink of bytes. The main practical use for a file descriptor is to create a FileInputStream or FileOutputStream to contain it.

**FileInputStream-** A FileInputStream obtains input bytes from a file in a file system. What files are available depends on the host environment.

**FileOutputStream-** A file output stream is an output stream for writing data to a File or to a FileDescriptor. Whether or not a file is available or may be created depends upon the underlying platform.

**FilePermission-** This class represents access to a file or directory. A FilePermission consists of a pathname and a set of actions valid for that pathname.

**FileReader-** Convenience class for reading character files. The constructors of this class assume that the default character encoding and the default byte-buffer size are appropriate.

**FilterWriter-** Abstract class for writing filtered character streams. The abstract class FilterWriter itself provides default methods that pass all requests to the contained stream.

**InputStream-** This abstract class is the superclass of all classes representing an input stream of bytes.

**InputStreamReader-** An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified [charset](https://docs.oracle.com/javase/8/docs/api/java/nio/charset/Charset.html).

**LineNumberInputStream-** This class is an input stream filter that provides the added functionality of keeping track of the current line number.

**LineNumberReader-** A buffered character-input stream that keeps track of line numbers. This class defines methods [setLineNumber(int)](https://docs.oracle.com/javase/8/docs/api/java/io/LineNumberReader.html" \l "setLineNumber-int-) and [getLineNumber()](https://docs.oracle.com/javase/8/docs/api/java/io/LineNumberReader.html" \l "getLineNumber--) for setting and getting the current line number respectively.

**ObjectInputStream-** An ObjectInputStream deserializes primitive data and objects previously written using an ObjectOutputStream.

**ObjectInputStream.GetField-** Provide access to the persistent fields read from the input stream.

**ObjectOutputStream-** An ObjectOutputStream writes primitive data types and graphs of Java objects to an OutputStream. The objects can be read (reconstituted) using an ObjectInputStream.

**ObjectOutputStream.PutField-** Provide programmatic access to the persistent fields to be written to ObjectOutput.

**ObjectStreamClass-** Serialization's descriptor for classes. It contains the name and serialVersionUID of the class. The ObjectStreamClass for a specific class loaded in this Java VM can be found/created using the lookup method.

**ObjectStreamField-** A description of a Serializable field from a Serializable class. An array of ObjectStreamFields is used to declare the Serializable fields of a class.

**OutputStream-** This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

**OutputStreamWriter-** An OutputStreamWriter is a bridge from character streams to byte streams: Characters written to it are encoded into bytes using a specified [charset](https://docs.oracle.com/javase/8/docs/api/java/nio/charset/Charset.html).

**PipedInputStream-** A piped input stream should be connected to a piped output stream; the piped input stream then provides whatever data bytes are written to the piped output stream.

**PipedOutputStream-** A piped output stream can be connected to a piped input stream to create a communications pipe. The piped output stream is the sending end of the pipe.

**PipedReader-** Piped character-input streams.

**PipedWriter-** Piped character-output streams.

**PrintStream-** A PrintStream adds functionality to another output stream, namely the ability to print representations of various data values conveniently. Two other features are provided as well.

**PrintWriter-** Prints formatted representations of objects to a text-output stream. This class implements all of the print methods found in [PrintStream](https://docs.oracle.com/javase/8/docs/api/java/io/PrintStream.html" \o "class in java.io). It does not contain methods for writing raw bytes, for which a program should use unencoded byte streams.

**PushbackInputStream-** A PushbackInputStream adds functionality to another input stream, namely the ability to "push back" or "unread" one byte. This is useful in situations where it is convenient for a fragment of code to read an indefinite number of data bytes that are delimited by a particular byte value; after reading the terminating byte, the code fragment can "unread" it, so that the next read operation on the input stream will reread the byte that was pushed back.

**PushbackReader-** A character-stream reader that allows characters to be pushed back into the stream.

**RandomAccessFile-** Instances of this class support both reading and writing to a random access file. A random access file behaves like a large array of bytes stored in the file system.

**Reader-** Abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

**SequenceInputStream-** A SequenceInputStream represents the logical concatenation of other input streams. It starts out with an ordered collection of input streams and reads from the first one until end of file is reached, whereupon it reads from the second one, and so on, until end of file is reached on the last of the contained input streams.

**SerializablePermission-** This class is for Serializable permissions. A SerializablePermission contains a name (also referred to as a "target name") but no actions list; you either have the named permission or you don't.

**StreamTokenizer-** The StreamTokenizer class takes an input stream and parses it into "tokens", allowing the tokens to be read one at a time. The parsing process is controlled by a table and a number of flags that can be set to various states.

**StringBufferInputStream -** This class allows an application to create an input stream in which the bytes read are supplied by the contents of a string. Applications can also read bytes from a byte array by using a ByteArrayInputStream.

**StringReader-** A character stream whose source is a string.

**StringWriter-** A character stream that collects its output in a string buffer, which can then be used to construct a string.

**Writer-** Abstract class for writing to character streams. The only methods that a subclass must implement are write(char[], int, int), flush(), and close(). Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

EXCEPTIONS

**CharConversionException-** Base class for character conversion exceptions.

**EOFException-** Signals that an end of file or end of stream has been reached unexpectedly during input.

**FileNotFoundException-** Signals that an attempt to open the file denoted by a specified pathname has failed.

**InterruptedIOException-** Signals that an I/O operation has been interrupted. An InterruptedIOException is thrown to indicate that an input or output transfer has been terminated because the thread performing it was interrupted.

**InvalidClassException-** Thrown when the Serialization runtime detects one of the following problems with a Class.

**InvalidObjectException-** Indicates that one or more deserialized objects failed validation tests. The argument should provide the reason for the failure.

**IOException-** Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.

**NotActiveException-** Thrown when serialization or deserialization is not active.

**NotSerializableException-** Thrown when an instance is required to have a Serializable interface. The serialization runtime or the class of the instance can throw this exception. The argument should be the name of the class.

**ObjectStreamException-** Superclass of all exceptions specific to Object Stream classes.

**OptionalDataException-** Exception indicating the failure of an object read operation due to unread primitive data, or the end of data belonging to a serialized object in the stream.

**StreamCorruptedException-** Thrown when control information that was read from an object stream violates internal consistency checks.

**SyncFailedException-** Signals that a sync operation has failed.

**SyncFailedException-** Signals that a sync operation has failed.

**UnsupportedEncodingException-** The Character Encoding is not supported.

**UTFDataFormatException-** Signals that a malformed string in [modified UTF-8](https://docs.oracle.com/javase/8/docs/api/java/io/DataInput.html#modified-utf-8) format has been read in a data input stream or by any class that implements the data input interface.

**WriteAbortedException-** Signals that one of the ObjectStreamExceptions was thrown during a write operation. Thrown during a read operation when one of the ObjectStreamExceptions was thrown during a write operation.

**Packages Description**

**Java.lang** - Provides classes that are fundamental to the design of the Java programming language.

INTERFACES

**Appendable**- *An* object to which char sequences and values can be appended. The Appendable interface must be implemented by any class whose instances are intended to receive formatted output from a [Formatter](https://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html).

**AutoCloseable**- An object that may hold resources (such as file or socket handles) until it is closed. The [close()](https://docs.oracle.com/javase/8/docs/api/java/lang/AutoCloseable.html#close--) method of an AutoCloseable object is called automatically when exiting a try-with-resources block for which the object has been declared in the resource specification header.

**CharSequence-** A CharSequence is a readable sequence of char values. This interface provides uniform, read-only access to many different kinds of char sequences. A char value represents a character in the *Basic Multilingual Plane (BMP)* or a surrogate. Refer to [Unicode Character Representation](https://docs.oracle.com/javase/8/docs/api/java/lang/Character.html#unicode) for details.

**Cloneable**- A class implements the Cloneable interface to indicate to the [Object.clone()](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html" \l "clone--) method that it is legal for that method to make a field-for-field copy of instances of that class.

**Comparable<T>**- This interface imposes a total ordering on the objects of each class that implements it. This ordering is referred to as the class's *natural ordering*, and the class's compareTo method is referred to as its *natural comparison method*.

**Iterable<T>-** Implementing this interface allows an object to be the target of the "for-each loop" statement. See [**For-each Loop**](https://docs.oracle.com/javase/8/docs/technotes/guides/language/foreach.html)

**Readable**- A Readable is a source of characters. Characters from a Readable are made available to callers of the read method via a [CharBuffer](https://docs.oracle.com/javase/8/docs/api/java/nio/CharBuffer.html" \o "class in java.nio).

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called run.

**Runnable**- The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called run.

**Thread.UncaughtExceptionHandler**- Interface for handlers invoked when a Thread abruptly terminates due to an uncaught exception.

**CLASSES**

**Boolean-** The Boolean class wraps a value of the primitive type boolean in an object. An object of type Boolean contains a single field whose type is boolean.

**Byte-** The Byte class wraps a value of primitive type byte in an object. An object of type Byte contains a single field whose type is byte.

**Character-** The Character class wraps a value of the primitive type char in an object. An object of class Character contains a single field whose type is char.

**Character.Subset-** Instances of this class represent particular subsets of the Unicode character set. The only family of subsets defined in the Character class is [Character.UnicodeBlock](https://docs.oracle.com/javase/8/docs/api/java/lang/Character.UnicodeBlock.html" \o "class in java.lang). Other portions of the Java API may define other subsets for their own purposes.

**Character.UnicodeBlock-** A family of character subsets representing the character blocks in the Unicode specification. Character blocks generally define characters used for a specific script or purpose. A character is contained by at most one Unicode block.

**Class<T>-** Instances of the class Class represent classes and interfaces in a running Java application. An enum is a kind of class and an annotation is a kind of interface.

**ClassLoader-** A class loader is an object that is responsible for loading classes. The class ClassLoader is an abstract class. Given the [binary name](https://docs.oracle.com/javase/8/docs/api/java/lang/ClassLoader.html#name) of a class, a class loader should attempt to locate or generate data that constitutes a definition for the class.

**ClassValue<T>-** Lazily associate a computed value with (potentially) every type. For example, if a dynamic language needs to construct a message dispatch table for each class encountered at a message send call site, it can use a ClassValue to cache information needed to perform the message send quickly, for each class encountered.

**Compiler** - The Compiler class is provided to support Java-to-native-code compilers and related services. By design, the Compiler class does nothing; it serves as a placeholder for a JIT compiler implementation.

**Double-** The Double class wraps a value of the primitive type double in an object. An object of type Double contains a single field whose type is double.

**Enum<E extends Enum<E>>-** This is the common base class of all Java language enumeration types. More information about enums, including descriptions of the implicitly declared methods synthesized by the compiler, can be found in section 8.9 of The Java™ Language Specification.

**Float-** The Float class wraps a value of primitive type float in an object. An object of type Float contains a single field whose type is float. In addition, this class provides several methods for converting a float to a String and a String to a float, as well as other constants and methods useful when dealing with a float.

**InheritableThreadLocal<T>-** This class extends ThreadLocal to provide inheritance of values from parent thread to child thread: when a child thread is created, the child receives initial values for all inheritable thread-local variables for which the parent has values. Normally the child's values will be identical to the parent's; however, the child's value can be made an arbitrary function of the parent's by overriding the childValue method in this class.

**Integer-** The Integer class wraps a value of the primitive type int in an object. An object of type Integer contains a single field whose type is int.

**Long-** The Long class wraps a value of the primitive type long in an object. An object of type Long contains a single field whose type is long.

**Math-** The class Math contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

**Number-** The abstract class Number is the superclass of platform classes representing numeric values that are convertible to the primitive types byte, double, float, int, long, and short. The specific semantics of the conversion from the numeric value of a particular Number implementation to a given primitive type is defined by the Number implementation in question.

**Object**- Class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.

**Package-** Package objects contain version information about the implementation and specification of a Java package. This versioning information is retrieved and made available by the [ClassLoader](https://docs.oracle.com/javase/8/docs/api/java/lang/ClassLoader.html" \o "class in java.lang) instance that loaded the class(es). Typically, it is stored in the manifest that is distributed with the classes.

**ProcessBuilder-** This class is used to create operating system processes. Each ProcessBuilder instance manages a collection of process attributes. The [start()](https://docs.oracle.com/javase/8/docs/api/java/lang/ProcessBuilder.html#start--) method creates a new [Process](https://docs.oracle.com/javase/8/docs/api/java/lang/Process.html) instance with those attributes. The [start()](https://docs.oracle.com/javase/8/docs/api/java/lang/ProcessBuilder.html#start--) method can be invoked repeatedly from the same instance to create new subprocesses with identical or related attributes.

**ProcessBuilder.Redirect-** Represents a source of subprocess input or a destination of subprocess output.

**Runtime**

extends [Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html)

Every Java application has a single instance of class Runtime that allows the application to interface with the environment in which the application is running. The current runtime can be obtained from the getRuntime method.

**Runtime-** Every Java application has a single instance of class Runtime that allows the application to interface with the environment in which the application is running. The current runtime can be obtained from the getRuntime method.

**RuntimePermission-** This class is for runtime permissions. A RuntimePermission contains a name (also referred to as a "target name") but no actions list; you either have the named permission or you don't.

**SecurityManager-** The security manager is a class that allows applications to implement a security policy. It allows an application to determine, before performing a possibly unsafe or sensitive operation, what the operation is and whether it is being attempted in a security context that allows the operation to be performed.

**Short-** The Short class wraps a value of primitive type short in an object. An object of type Short contains a single field whose type is short.

**StackTraceElement-** An element in a stack trace, as returned by [Throwable.getStackTrace()](https://docs.oracle.com/javase/8/docs/api/java/lang/Throwable.html" \l "getStackTrace--). Each element represents a single stack frame. All stack frames except for the one at the top of the stack represent a method invocation. The frame at the top of the stack represents the execution point at which the stack trace was generated. Typically, this is the point at which the throwable corresponding to the stack trace was created.

**StrictMath-** The class StrictMath contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

**String-** The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

**StringBuffer-** A thread-safe, mutable sequence of characters. A string buffer is like a [String](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html), but can be modified. At any point in time it contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls.

**StringBuilder-** A mutable sequence of characters. This class provides an API compatible with StringBuffer, but with no guarantee of synchronization. This class is designed for use as a drop-in replacement for StringBuffer in places where the string buffer was being used by a single thread (as is generally the case).

**System-** The System class contains several useful class fields and methods. It cannot be instantiated.

**Thread-** A *thread* is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.

**ThreadGroup-** A thread group represents a set of threads. In addition, a thread group can also include other thread groups. The thread groups form a tree in which every thread group except the initial thread group has a parent.

**ThreadLocal<T>-** This class provides thread-local variables. These variables differ from their normal counterparts in that each thread that accesses one (via its get or set method) has its own, independently initialized copy of the variable. ThreadLocal instances are typically private static fields in classes that wish to associate state with a thread (e.g., a user ID or Transaction ID).

**Throwable-** The Throwable class is the superclass of all errors and exceptions in the Java language. Only objects that are instances of this class (or one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java throw statement. Similarly, only this class or one of its subclasses can be the argument type in a catch clause.

**Void-** The Void class is an uninstantiable placeholder class to hold a reference to the Class object representing the Java keyword void.

EXCEPTION

**ArithmeticException-** Thrown when an exceptional arithmetic condition has occurred. For example, an integer "divide by zero" throws an instance of this class. ArithmeticException objects may be constructed by the virtual machine as if [suppression were disabled and/or the stack trace was not writable](https://docs.oracle.com/javase/8/docs/api/java/lang/Throwable.html#Throwable-java.lang.String-java.lang.Throwable-boolean-boolean-).

**ArrayIndexOutOfBoundsException-** Thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

**ArrayStoreException-** Thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects.

**ClassCastException-** Thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance.

**ClassNotFoundException**

Thrown when an application tries to load in a class through its string name using:The forName method in class Class.The findSystemClass method in class ClassLoader .The loadClass method in class ClassLoader.but no definition for the class with the specified name could be found.

**CloneNotSupportedException-** Thrown to indicate that the clone method in class Object has been called to clone an object, but that the object's class does not implement the Cloneable interface.

**EnumConstantNotPresentException-** Thrown when an application tries to access an enum constant by name and the enum type contains no constant with the specified name. This exception can be thrown by the [API used to read annotations reflectively](https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/AnnotatedElement.html).

**Exception-** The class Exception and its subclasses are a form of Throwable that indicates conditions that a reasonable application might want to catch.

**IllegalAccessException-** An IllegalAccessException is thrown when an application tries to reflectively create an instance (other than an array), set or get a field, or invoke a method, but the currently executing method does not have access to the definition of the specified class, field, method or constructor.

**IllegalArgumentException-** Thrown to indicate that a method has been passed an illegal or inappropriate argument.

**IllegalMonitorStateException-** Thrown to indicate that a thread has attempted to wait on an object's monitor or to notify other threads waiting on an object's monitor without owning the specified monitor.

**IllegalStateException-** extends [RuntimeException](https://docs.oracle.com/javase/8/docs/api/java/lang/RuntimeException.html)

Signals that a method has been invoked at an illegal or inappropriate time. In other words, the Java environment or Java application is not in an appropriate state for the requested operation.

**IllegalThreadStateException**

extends [IllegalArgumentException](https://docs.oracle.com/javase/8/docs/api/java/lang/IllegalArgumentException.html)

Thrown to indicate that a thread is not in an appropriate state for the requested operation. See, for example, the suspend and resume methods in class Thread.

**IndexOutOfBoundsException-** Thrown to indicate that an index of some sort (such as to an array, to a string, or to a vector) is out of range.

**InstantiationException-** Thrown when an application tries to create an instance of a class using the newInstance method in class Class, but the specified class object cannot be instantiated.

**InterruptedException-** Thrown when a thread is waiting, sleeping, or otherwise occupied, and the thread is interrupted, either before or during the activity. Occasionally a method may wish to test whether the current thread has been interrupted, and if so, to immediately throw this exception.

**NegativeArraySizeException-** Thrown if an application tries to create an array with negative size.

**NoSuchFieldException-** Signals that the class doesn't have a field of a specified name.

**NoSuchMethodException-** Thrown when a particular method cannot be found.

**NullPointerException-** Thrown when an application attempts to use null in a case where an object is required.

**NumberFormatException-** Thrown to indicate that the application has attempted to convert a string to one of the numeric types, but that the string does not have the appropriate format.

**ReflectiveOperationException-** Common superclass of exceptions thrown by reflective operations in core reflection.

**RuntimeException-** RuntimeException is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine.

**SecurityException-**Thrown by the security manager to indicate a security violation.

**StringIndexOutOfBoundsException-**Thrown by String methods to indicate that an index is either negative or greater than the size of the string.

**TypeNotPresentException-** Thrown when an application tries to access a type using a string representing the type's name, but no definition for the type with the specified name can be found.

**UnsupportedOperationException-** Thrown to indicate that the requested operation is not supported.

**Packages Description**

**Java.lang.annotation** - Provides library support for the Java programming language annotation facility.

INTERFACES

**Annotation**- The common interface extended by all annotation types. Note that an interface that manually extends this one does *not* define an annotation type. Also note that this interface does not itself define an annotation type. More information about annotation types can be found in section 9.6 of *The Java™ Language Specification*. The [AnnotatedElement](https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/AnnotatedElement.html" \o "interface in java.lang.reflect) interface discusses compatibility concerns when evolving an annotation type from being non-repeatable to being repeatable.

EXCEPTION

**AnnotationTypeMismatchException-** Thrown to indicate that a program has attempted to access an element of an annotation whose type has changed after the annotation was compiled (or serialized). This exception can be thrown by the [API used to read annotations reflectively](https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/AnnotatedElement.html).

**IncompleteAnnotationException-** Thrown to indicate that a program has attempted to access an element of an annotation type that was added to the annotation type definition after the annotation was compiled (or serialized). This exception will not be thrown if the new element has a default value. This exception can be thrown by the [API used to read annotations reflectively](https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/AnnotatedElement.html).

**Packages Description**

**Java.lang.instrument -** Provides services that allow Java programming language agents to instrument programs running on the JVM.

INTERFACES

**ClassFileTransformer**- An agent provides an implementation of this interface in order to transform class files. The transformation occurs before the class is defined by the JVM.

**Instrumentation**- This class provides services needed to instrument Java programming language code. Instrumentation is the addition of byte-codes to methods for the purpose of gathering data to be utilized by tools. Since the changes are purely additive, these tools do not modify application state or behavior. Examples of such benign tools include monitoring agents, profilers, coverage analyzers, and event loggers.

CLASSES

**ClassDefinition-** This class serves as a parameter block to the Instrumentation.redefineClasses method. Serves to bind the Class that needs redefining together with the new class file bytes.

EXCEPTION

**IllegalClassFormatException-** Thrown by an implementation of [ClassFileTransformer.transform](https://docs.oracle.com/javase/8/docs/api/java/lang/instrument/ClassFileTransformer.html" \l "transform-java.lang.ClassLoader-java.lang.String-java.lang.Class-java.security.ProtectionDomain-byte:A-) when its input parameters are invalid. This may occur either because the initial class file bytes were invalid or a previously applied transform corrupted the bytes.

**UnmodifiableClassException-** Thrown by an implementation of [Instrumentation.redefineClasses](https://docs.oracle.com/javase/8/docs/api/java/lang/instrument/Instrumentation.html" \l "redefineClasses-java.lang.instrument.ClassDefinition...-) when one of the specified classes cannot be modified.

**Packages Description**

**Java.lang.invoke** - The java.lang.invoke package contains dynamic language support provided directly by the Java core class libraries and virtual machine.

INTERFACES

**MethodHandleInfo**- A symbolic reference obtained by cracking a direct method handle into its consitutent symbolic parts. To crack a direct method handle, call [Lookup.revealDirect](https://docs.oracle.com/javase/8/docs/api/java/lang/invoke/MethodHandles.Lookup.html" \l "revealDirect-java.lang.invoke.MethodHandle-).

CLASSES

**CallSite-** A CallSite is a holder for a variable [MethodHandle](https://docs.oracle.com/javase/8/docs/api/java/lang/invoke/MethodHandle.html" \o "class in java.lang.invoke), which is called its target. An invokedynamic instruction linked to a CallSite delegates all calls to the site's current target. A CallSite may be associated with several invokedynamic instructions, or it may be "free floating", associated with none.

**ConstantCallSite-** A ConstantCallSite is a [CallSite](https://docs.oracle.com/javase/8/docs/api/java/lang/invoke/CallSite.html" \o "class in java.lang.invoke) whose target is permanent, and can never be changed. An invokedynamic instruction linked to a ConstantCallSite is permanently bound to the call site's target.

**LambdaMetafactory-** Methods to facilitate the creation of simple "function objects" that implement one or more interfaces by delegation to a provided [MethodHandle](https://docs.oracle.com/javase/8/docs/api/java/lang/invoke/MethodHandle.html" \o "class in java.lang.invoke), possibly after type adaptation and partial evaluation of arguments. These methods are typically used as bootstrap methods for invokedynamic call sites, to support the lambda expression and method reference expression features of the Java Programming Language.

**MethodHandle-** A method handle is a typed, directly executable reference to an underlying method, constructor, field, or similar low-level operation, with optional transformations of arguments or return values. These transformations are quite general, and include such patterns as [conversion](https://docs.oracle.com/javase/8/docs/api/java/lang/invoke/MethodHandle.html#asType-java.lang.invoke.MethodType-), [insertion](https://docs.oracle.com/javase/8/docs/api/java/lang/invoke/MethodHandle.html#bindTo-java.lang.Object-), [deletion](https://docs.oracle.com/javase/8/docs/api/java/lang/invoke/MethodHandles.html#dropArguments-java.lang.invoke.MethodHandle-int-java.util.List-), and [substitution](https://docs.oracle.com/javase/8/docs/api/java/lang/invoke/MethodHandles.html#filterArguments-java.lang.invoke.MethodHandle-int-java.lang.invoke.MethodHandle...-).

**MethodHandles.Lookup-** A lookup object is a factory for creating method handles, when the creation requires access checking. Method handles do not perform access checks when they are called, but rather when they are created. Therefore, method handle access restrictions must be enforced when a method handle is created.

**MethodType-** A method type represents the arguments and return type accepted and returned by a method handle, or the arguments and return type passed and expected by a method handle caller. Method types must be properly matched between a method handle and all its callers, and the JVM's operations enforce this matching at, specifically during calls to [MethodHandle.invokeExact](https://docs.oracle.com/javase/8/docs/api/java/lang/invoke/MethodHandle.html" \l "invokeExact-java.lang.Object...-) and [MethodHandle.invoke](https://docs.oracle.com/javase/8/docs/api/java/lang/invoke/MethodHandle.html" \l "invoke-java.lang.Object...-), and during execution of invokedynamic instructions.

**MutableCallSite-** A MutableCallSite is a [CallSite](https://docs.oracle.com/javase/8/docs/api/java/lang/invoke/CallSite.html" \o "class in java.lang.invoke) whose target variable behaves like an ordinary field. An invokedynamic instruction linked to a MutableCallSite delegates all calls to the site's current target. The [dynamic invoker](https://docs.oracle.com/javase/8/docs/api/java/lang/invoke/CallSite.html#dynamicInvoker--) of a mutable call site also delegates each call to the site's current target.

**SerializedLambda-** Serialized form of a lambda expression. The properties of this class represent the information that is present at the lambda factory site, including static metafactory arguments such as the identity of the primary functional interface method and the identity of the implementation method, as well as dynamic metafactory arguments such as values captured from the lexical scope at the time of lambda capture.

**SwitchPoint-** A SwitchPoint is an object which can publish state transitions to other threads. A switch point is initially in the valid state, but may at any time be changed to the invalid state. Invalidation cannot be reversed. A switch point can combine a guarded pair of method handles into a guarded delegator. The guarded delegator is a method handle which delegates to one of the old method handles. The state of the switch point determines which of the two gets the delegation.

**VolatileCallSite-** A VolatileCallSite is a [CallSite](https://docs.oracle.com/javase/8/docs/api/java/lang/invoke/CallSite.html" \o "class in java.lang.invoke) whose target acts like a volatile variable. An invokedynamic instruction linked to a VolatileCallSite sees updates to its call site target immediately, even if the update occurs in another thread. There may be a performance penalty for such tight coupling between threads.

EXCEPTIONS

**LambdaConversionException-** LambdaConversionException

**WrongMethodTypeException-** Thrown to indicate that code has attempted to call a method handle via the wrong method type. As with the bytecode representation of normal Java method calls, method handle calls are strongly typed to a specific type descriptor associated with a call site.

**Packages Description**

**Java.lang.management-** Provides the management interfaces for monitoring and management of the Java virtual machine and other components in the Java runtime.

INTERFACES

**BufferPoolMXBean-** The management interface for a buffer pool, for example a pool of [direct](https://docs.oracle.com/javase/8/docs/api/java/nio/ByteBuffer.html#allocateDirect-int-) or [mapped](https://docs.oracle.com/javase/8/docs/api/java/nio/MappedByteBuffer.html) buffers.

**ClassLoadingMXBean-** The management interface for the class loading system of the Java virtual machine.

**CompilationMXBean-** The management interface for the compilation system of the Java virtual machine.

**GarbageCollectorMXBean-** The management interface for the garbage collection of the Java virtual machine. Garbage collection is the process that the Java virtual machine uses to find and reclaim unreachable objects to free up memory space. A garbage collector is one type of [memory manager](https://docs.oracle.com/javase/8/docs/api/java/lang/management/MemoryManagerMXBean.html).

**MemoryManagerMXBean-** The management interface for a memory manager. A memory manager manages one or more memory pools of the Java virtual machine.

**MemoryMXBean-** The management interface for the memory system of the Java virtual machine.

**MemoryPoolMXBean-** The management interface for a memory pool. A memory pool represents the memory resource managed by the Java virtual machine and is managed by one or more [memory managers](https://docs.oracle.com/javase/8/docs/api/java/lang/management/MemoryManagerMXBean.html).

**OperatingSystemMXBean-** The management interface for the operating system on which the Java virtual machine is running.

**PlatformLoggingMXBean-** The management interface for the [logging](https://docs.oracle.com/javase/8/docs/api/java/util/logging/package-summary.html) facility.

**PlatformManagedObject-** A platform managed object is a [JMX MXBean](https://docs.oracle.com/javase/8/docs/api/javax/management/MXBean.html) for monitoring and managing a component in the Java platform. Each platform managed object has a unique [object name](https://docs.oracle.com/javase/8/docs/api/java/lang/management/ManagementFactory.html#MXBean) for the [platform MBeanServer](https://docs.oracle.com/javase/8/docs/api/java/lang/management/ManagementFactory.html#getPlatformMBeanServer--) access. All platform MXBeans will implement this interface.

**RuntimeMXBean-** The management interface for the runtime system of the Java virtual machine.

**ThreadMXBean-** The management interface for the thread system of the Java virtual machine.

CLASSES

**LockInfo-** Information about a lock. A lock can be a built-in object monitor, an ownable synchronizer, or the [Condition](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/Condition.html) object associated with synchronizers.

**ManagementFactory-** The ManagementFactory class is a factory class for getting managed beans for the Java platform. This class consists of static methods each of which returns one or more *platform MXBeans* representing the management interface of a component of the Java virtual machine.

**ManagementPermission-** The permission which the SecurityManager will check when code that is running with a SecurityManager calls methods defined in the management interface for the Java platform.

**MemoryNotificationInfo-** The information about a memory notification.

**MemoryUsage-** A MemoryUsage object represents a snapshot of memory usage. Instances of the MemoryUsage class are usually constructed by methods that are used to obtain memory usage information about individual memory pool of the Java virtual machine or the heap or non-heap memory of the Java virtual machine as a whole.

**MonitorInfo-** Information about an object monitor lock. An object monitor is locked when entering a synchronization block or method on that object.

**ThreadInfo-** Thread information. ThreadInfo contains the information about a thread including:

|  |
| --- |
| **C. EXPLAIN: Reading**  To understand the module activities, read and practice the reading materials on the internet, search it in google to explore more information about the usage and features of java API specifications in different versions. |
| **D. ELABORATE: Additional Details**  More in java API Specifications:  Give the Requirements for Writing a Java API Specifications.  [Top-Level Specification](https://www.oracle.com/java/technologies/javase/api-specifications.html#top-level)  [Package Specification](https://www.oracle.com/java/technologies/javase/api-specifications.html#package)  [Package Examples](https://www.oracle.com/java/technologies/javase/api-specifications.html#packageExample1)  [Class/Inteface Specification](https://www.oracle.com/java/technologies/javase/api-specifications.html#class)  [Interface Example](https://www.oracle.com/java/technologies/javase/api-specifications.html#interfaceExample1)  [Class Examples](https://www.oracle.com/java/technologies/javase/api-specifications.html#classExample1)  [Field Specification](https://www.oracle.com/java/technologies/javase/api-specifications.html#field)  [Field Examples](https://www.oracle.com/java/technologies/javase/api-specifications.html#fieldExample1)  [Method Specification](https://www.oracle.com/java/technologies/javase/api-specifications.html#method)  [Method Examples](https://www.oracle.com/java/technologies/javase/api-specifications.html#methodExample1) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **E.EVALUATE**  **Self-Assessment.**  Kindly check (✔) the box of your answer for each question. In this way, we will be able to assess how much we have learned and what are the things that needs to be improved. | | | | |
| **Questions** | **YES** | **NO** | **MAYBE** |
| **1. Did I work hard on this module?** | ✔ |  |  |
| **2. Did I understand what my teacher asked me to do?** | ✔ |  |  |
| **3. Did I spend enough time to finish answering this module?** | ✔ |  |  |
| **4. Did I make good use of available resources?** | ✔ |  |  |
| **5. Did I check/ review my work for possible errors?** | ✔ |  |  |
| **6. Did I learn something in this module?** | ✔ |  |  |
| **7. Did I ask questions if I needed help?** | ✔ |  |  |
| **8. Did I read the instructions carefully?** | ✔ |  |  |
| **9. Did I set high standards for myself?** | ✔ |  |  |
| **10. Did I meet the success criteria?** |  |  | ✔ |