Creating Client jtreg tests

Backlog / deficit

There are a lot of tests and we have decent functional coverage but even so there's a lot of accumulated issues that we should address.
In rough priority order the main categories for the tests themselves are :

- Fix tests that are currently failing and / or are problem listed
- Update tests that use the Applet API to remove that dependency
- Migrate more tests to OpenJDK from closed test suites.
- Update manual tests to use a common framework for reporting the results
- Make more manual tests into automated tests

Writing stable automated client tests

Reliable automated GUI testing is hard work since rendering may be

1. different from platform to platform
2. Affected by O/S desktop changes
3. Affected by platform specifics such as display resolution, color profiles, keyboard shortcuts
4. Affected by timing issues

etc.

We expect automated tests to be resilient against these and pass reliably 99.99% of the time
Tests which are considered to be "important" should pass with even greater reliability.

Headless tests are those which do nothing that would cause a java.awt.HeadlessException to be thrown if there is no display.

So you can't even **create** (never mind show) a Window/Frame/Dialog.

Headful tests are therefore the opposite - they do create a UI on screen. Tests which want to verify D3D/OpenGL/other accelerated pipelines will be headful.

Headless tests run entirely in software mode, eg drawing to a Java-heap allocated BufferedImage.

Tests which ARE headful must specify "@key headful" as a jtreg tag. This is a convention used so that test frameworks can assign tests to appropriate testing resources.

Do not specify that if your test does not need it since headless tests can be run (1) in parallel on a host, (2) on "cloud" / "server" test hosts which don't require any graphics resources.

Many of our automated tests that are headful make use of the java.awt.Robot API to

- deliver mouse and key input events
- grab portions of the screen for comparison against expected rendering.
  These have additional considerations.

The following are some guidelines that should be followed in writing client tests and to ensure stability and reliability of those tests. This will be added to and refined over time.

- Tests should be automated. If it does not need to be manual - automate it
- Headless tests are preferred - they run faster and for our CI testing on much available hardware and they may be better candidates to run in github actions (some day).
  For extra credit make a UI "optional" - via a command line option to help someone evaluating a failure
- Headless tests can grab screen pixels from a BufferedImage if testing rendering and should be reliable
- Automated tests that need to test screen rendering can use java.awt.Robot to grab screen pixels
  But beware of (1) rounded corners of application windows (2) Unexpected overlapping (3) Shadow effects near the edges of windows.
  - Try to ensure you sample safely inside a window in an area where all colours should be as expected
- Beware of tests that may be unstable due to some other test leaving a modifier key pressed or CAPS LOCK activated.
- Make sure YOUR test always cleans up after itself so that no matter of error, or failure, or what it leaves the keyboard state as it found it.
- Make sure to hide() and dispose() all windows
- Use Robot.setAutoDelay() so you don't have to insert delays after every key press
- Use additional delays where the auto delay may not be enough but don't over do it !
- Use SwingUtilities.invokeAndWait() etc to make sure the UI is created before your test runs
- In general follow threading rules
- There may be multiple paint events delivered - so make sure you don't just paint the first time throught - behave like a real application and repaint on demand and grab results only once the UI is stablised.
- Use Toolkit/Robot sync to flush all requests
- use Robot.waitForIdle() to ensure all events are processed Use volatile or synchronized or Atomic classes or whatever is appropriate to make sure threads see changes, or see consistent state
- Make sure the test can safely run in the default time allowed by jtreg (180 seconds). Otherwise increase the time out
- If the test fails log as much info as you can as to what failed "test failed" is terribly unhelpful if a test failure can't be reproduced. Use printStackTrace(), print the state of variables, save as PNG (which is a loss-less format) a BufferedImage that isn't rendered as expected etc .. make sure everything points back to the source of the failure, and you aren't losing that and just summarising the result.
- The fewer windows the better.
- Be wary of edge effects - shadows, rounded corners etc. Don't grab pixels near the boundaries of a window.

- Be aware that the desktop may be "scaled" meaning that 1 user pixel is not 1 device pixel. Robot will down-sample from the device to create a BufferedImage with the number of pixels you request. So even within that image, make sure you sample within an area where there won't be any edge effects where colours change.