Development Environment Set up

Created by Rajeev Chamyal, last modified by Ambarish Rapte on Mar 22, 2019

Table of Contents

Setting up build environment.

Pre-requisites

- **[Windows/Linux/Mac] Boot JDK :** jdk for building OpenJDK.
  JAVA SE9 / 10 /11 latest edition from: http://www.oracle.com/technetwork/java/javase/downloads/index.html
  It is advised to use the latest JDK builds as boot JDK (**10 / 11**)
  OR
  JDK bundles from: https://java.se.oracle.com/artifactory/re-release-local/jdk/ And/OR
  http://jre.us.oracle.com/java/re/jdk

  Set env variables: Typical set of exports for Java on **Windows**
  export JAVA_HOME="D:\\jdk_bundle\\10\\46\\jdk-10"

  export JDK_HOME=$JAVA_HOME

  export JAVA_BIN=$JAVA_HOME\\bin

  export PATH=${JAVA_BIN}:${PATH}

- **[Windows/Linux/Mac]Mercurial SCM** (source control management tool)
  - Cygwin includes mercurial. The version would be little older but it would be sufficient
  - Latest mercurial on https://mercurial.selenic.com/wiki/Download
  - [Windows] TortoiseHg GUI Front end to Mercurial http://tortoisehg.bitbucket.org/download/
  - For Mac check the default system python version and install compatible mercurial version.
  - **[Ubuntu] See Virtual Linux OS**
    - sudo apt-get update
    - sudo apt-get install mercurial

- **[Windows/Linux/Mac]IDE**
  - IntelliJ IDEA (Latest Community Edition) - https://www.jetbrains.com/idea/
  - Netbeans latest Development build: https://netbeans.apache.org/download/index.html

- **[Windows] DirectX Software Development Kit**
  - Direct X9 : https://www.microsoft.com/en-us/download/details.aspx?id=6812

- **[Windows] Visual Studio**
  - Visual Studio 2013 / **2015** : https://uno.oraclecorp.com/uno/dssm
  - Microsoft site : https://visualstudio.microsoft.com/vs/older-downloads/
  -
- **[Windows] Cygwin**
  1. Install cygwin [Windows] from https://cygwin.com/install.html
  2. Run the cygwin setup-x86_64.exe (preferable 64 bit)
  3. Select http://mirrors.kernel.org server for downloading packages.
  4. Required cygwin packages for building JDK are Devel and archive
  5. Set JAVA_HOME, JDK_HOME & PATH env variables in ~/.bashrc

- **[Solaris]Solaris**
  1. Download Oracle Solaris 11.2 VM template for Oracle VM Virtual box from http://www.oracle.com/technetwork/server-storage/solaris11/downloads/vm-templates-2245495.html
  2. Follow these steps to install the Oracle 11.2 on Oracle VM, http://www.oracle.com/technetwork/systems/hands-on-labs/s11-vbox-install-1408628.html
     a. Please note above guide installs 32-bit (i386 arch), please download related image to your need.
  3. After installation change proxy to www-proxy.idc.oracle.com:80
  4. Follow these steps and install solaris studio 12.4, http://docs.oracle.com/cd/E37069_01/html/E37072/gozlw.html#scrolltoc
  5. Update the PATH variable
     a. gedit ~/.profile
     b. export PATH=${PATH}:/opt/solarisstudio12.4/bin/
  6. Install necessary packages for build
     a. pfexec pkg install jdk-8
     b. pkg install SUNWmercurial
     c. pfexec pkg install SUNWgmake SUNWcups SUNWzip SUNWunzip SUNWxwhl SUNWxorg-headers SUNWaudh SUNWfreetype2
     d. sudo pkg install pkg:/developer/assembler
  7. Follow below given steps to download the JDK source repo
  8. There is a slight change in build command, please check the build section below.

- **[Mac] Xcode** application for Mac OS X https://developer.apple.com/xcode/

- **Virtual Linux OS**
  - [Windows/Mac] Latest virtual box https://www.virtualbox.org/wiki/Downloads
  - [Windows/Mac] Ubuntu 16.04 **/** 18.04 LTS 64 bit on Virtual box http://www.ubuntu.com/download/desktop : (prefer 16.04)
  - [Linux - Ubuntu] : After installation of Virtual OS - do below steps to get started :
    a. Set the system wide network proxy : www-proxy.idc.oracle.com:80
    b. Open a terminal and issue command : "sudo apt-get update"
    c. The above command may fail to connect to internet. The browser does connect to internet but terminal fails. Use below steps to solve the problem.
       i. Change of proxy:
          1. Change proxy setting to automatic method using configuration URL http://wpad/wpad.dat.
          2. If this does not work follow below steps.
       ii. apt.conf
          1. Launch terminal

2. *cd /etc/apt*
3. *sudo gedit apt.conf*
4. Add below two lines to apt.conf & save the file
   Acquire::http::proxy "http://www-proxy.idc.oracle.com:80/";
   Acquire::ftp::proxy "http://www-proxy.idc.oracle.com:80/";
5. Save the apt.conf file & start new terminal. The terminal should connect to internet.

d. Install **mercurial**: sudo apt-get install mercurial
e. To use shared folder of host system, run below command and restart the VM.
   i. *sudo adduser <user_name> vboxsf*

Building JDK

Step 1: get open + closed sources

a. Dev repository
   i. hg clone http://closedjdk.us.oracle.com/jdk/client
   ii. cd client
   iii. sh get_source.sh **OR** hg clone http://hg.openjdk.java.net/jdk/client open
b. Old repository (**NOT REQUIRED**)
   i. hg clone http://hg.openjdk.java.net/jdk/client
   ii. cd client
   iii. sh get_source.sh
   iv. sh get_source.sh http://closedjdk.us.oracle.com // you need to be on OWAN/VPNed in for this one
   v. To save build time and also to avoid figuring it out since we rarely need these :- rm -rf pubs deploy install

Step 2: Build JDK on Linux(Virtual box)/Windows/Mac OS X

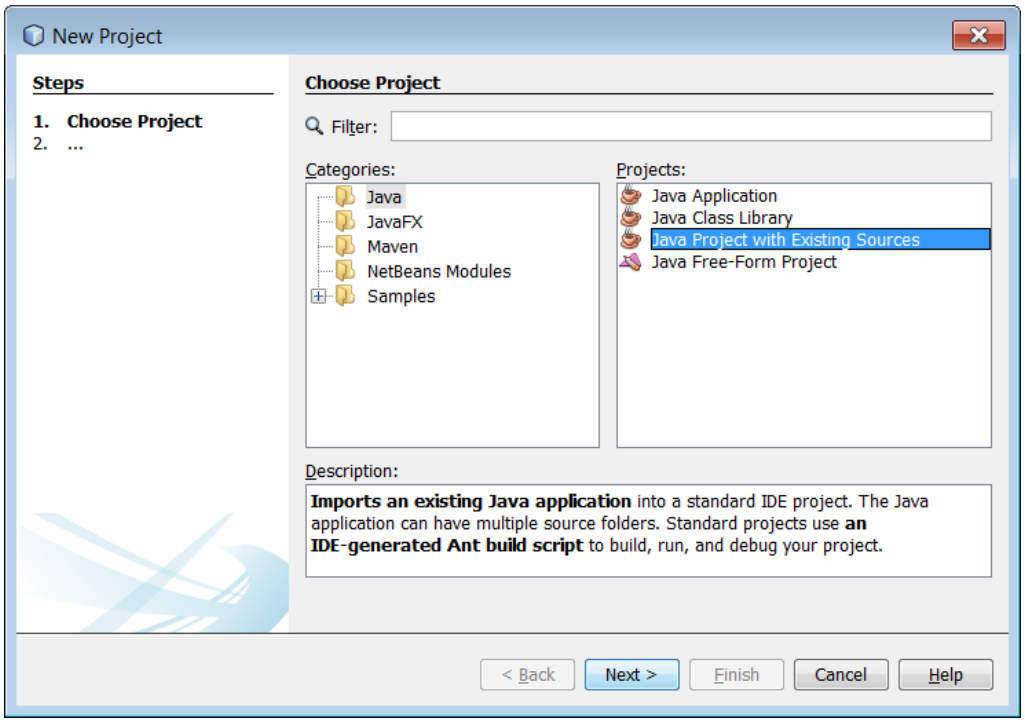
1. **Linux / Windows / Mac**
   - ***Configure the build***
     - navigate to **client** directory in terminal
     - *sh configure*
     - In case of errors follow the error log to install missing packages
     - There are three types of build based of debug level, release, fastdebug & slowdebug.
       - release, **sh configure**
       - fast debug, **sh configure --enable-debug**
       - solw debug, ***sh configure --with-debug-level=slowdebug***
     - Can provide boot jdk with ***--with-boot-jdk*** option.
     - Command for more help on options: ***sh configure --help***
   - **BuildJDK**
     - navigate to **client** directory in terminal
     - make help : up-to-date list of important make targets and make control variables
     - make
     - make images
     - make demos
     - make all
     - make CONF=<conf name> where <conf name> on **ubuntu** is linux-x86-normal-server-release **/** linux-x86-normal-server-fastdebug **/** linux-x86-normal-server-slowdebug
       - release, make CONF=**linux**-x86-normal-server-release
       - fastdebug, make CONF=**linux**-x86-normal-server-fastdebug
       - slowdebug, make CONF=**linux**-x86-normal-server-slowdebug
     - Build directory : client / build / (XXXXX-release **/** fastdebug **/** slowdebug)

2. **Solaris**
   - Steps are same as above but there is slight change as below
   - In **sh configure** command, these two options may be required **--disable-warnings-as-errors** & **--with-native-debug-symbols=internal**
   - Use **gmake** command instead of **make**

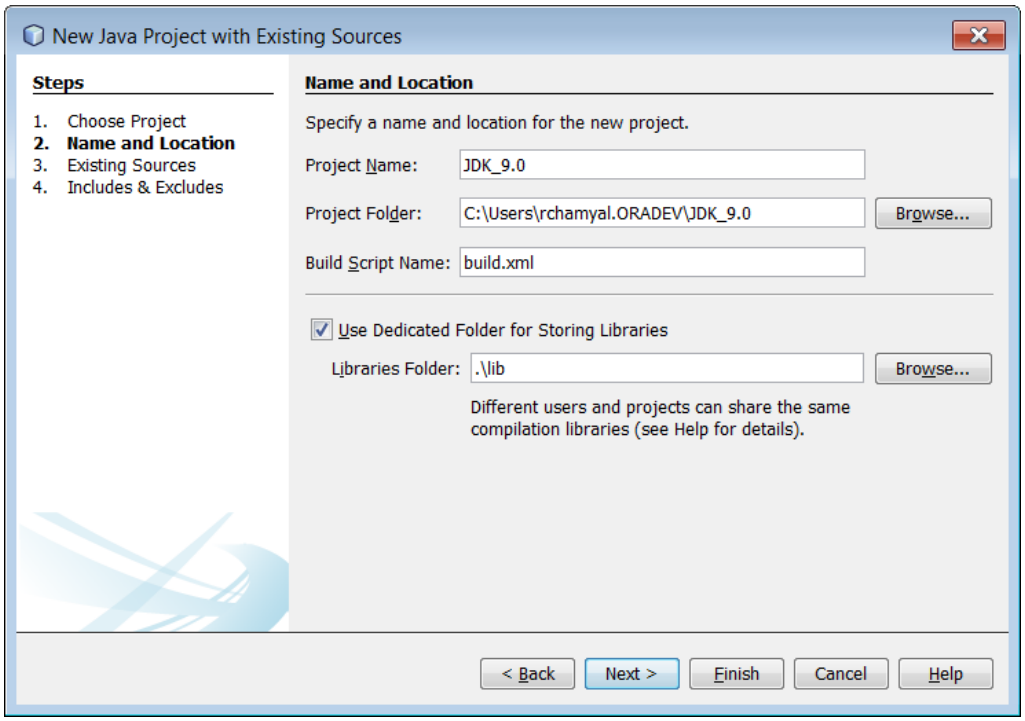
3. **Mac**
   - Steps are same as above but there is slight change as below
   - In **sh configure** command, **--disable-warnings-as-errors** option may be required.

Netbeans: Creating java project with JDK source files

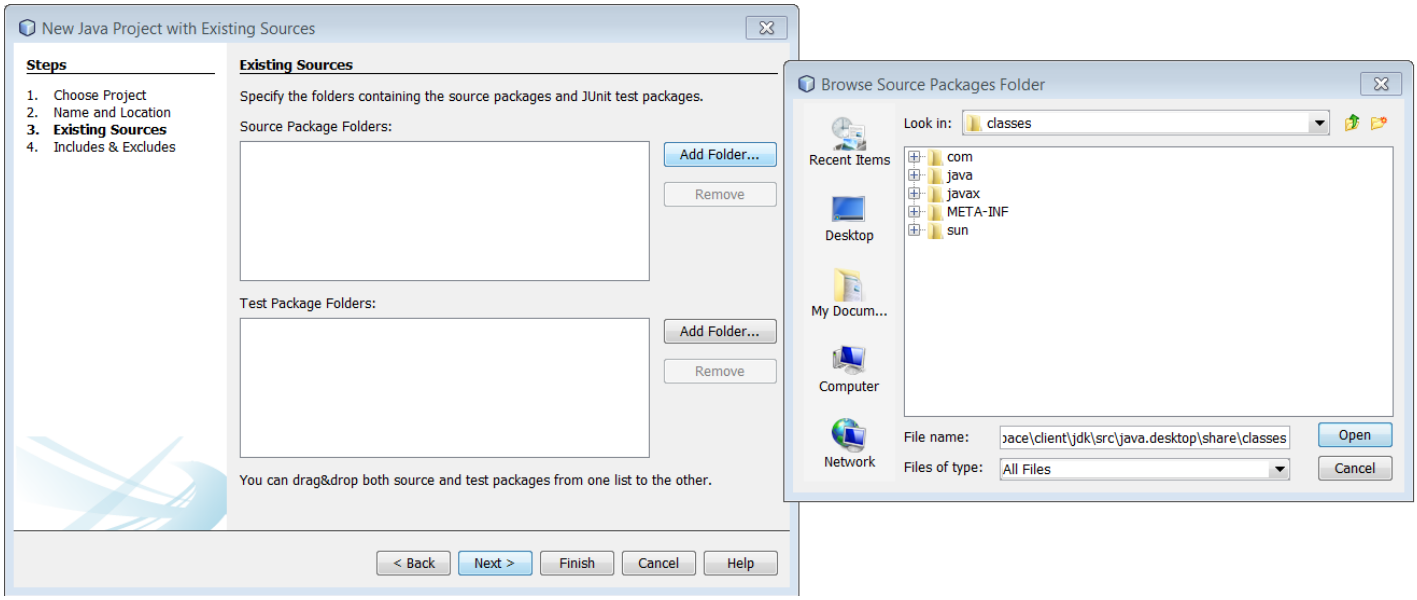
1. Create new java project with existing sources.



2. Give some name to project and click next.



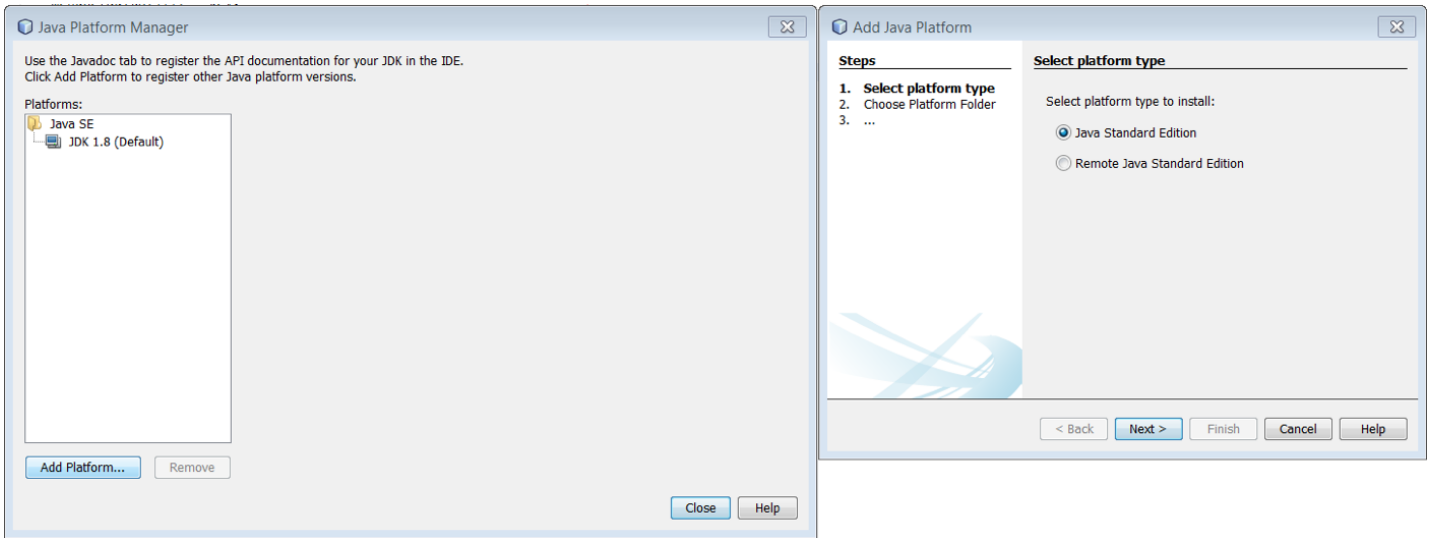
3. Add the relevant source folders depending on the module you are working and then click finish. A new project will be created in netbeans.



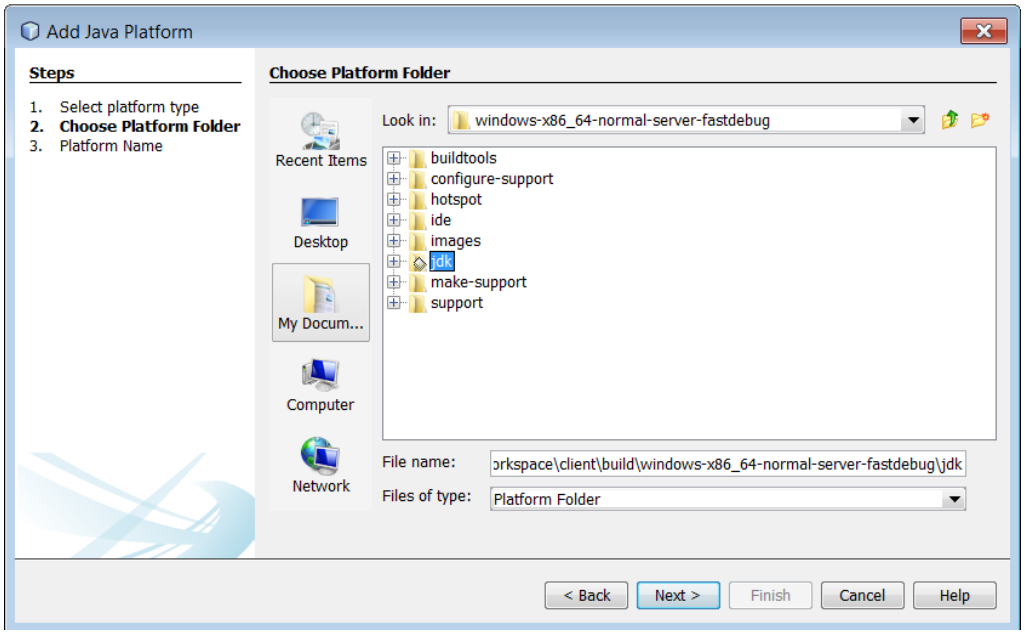
Netbeans: Creating java platform with locally built JDK

1. Add new JAVA platform to Netbeans.Click on tools menu in Netbeans toolbar *Tools-> Java Platform*.

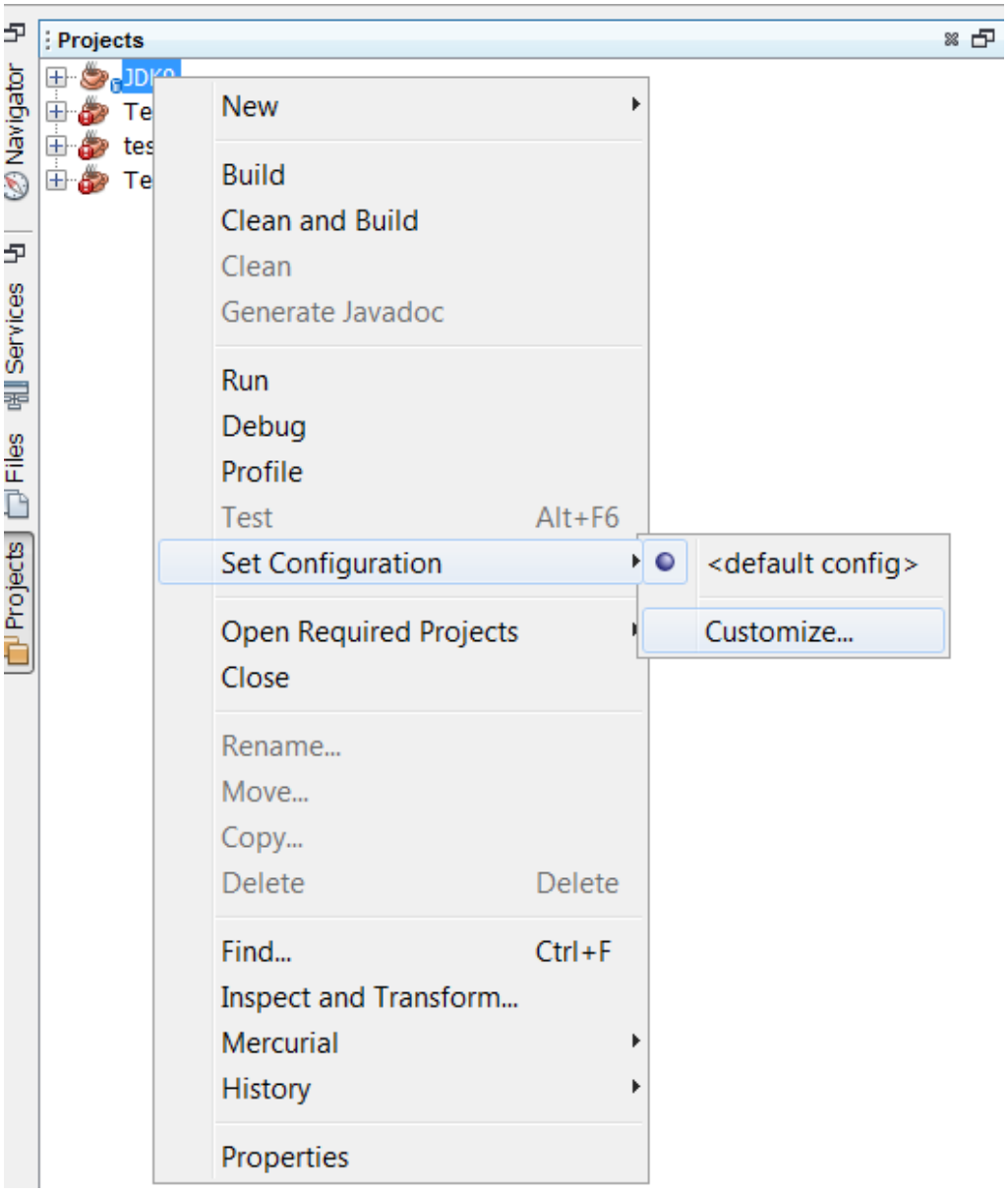
   Click on Add Platform.. button. Select *Java Standard Edition* and click next.



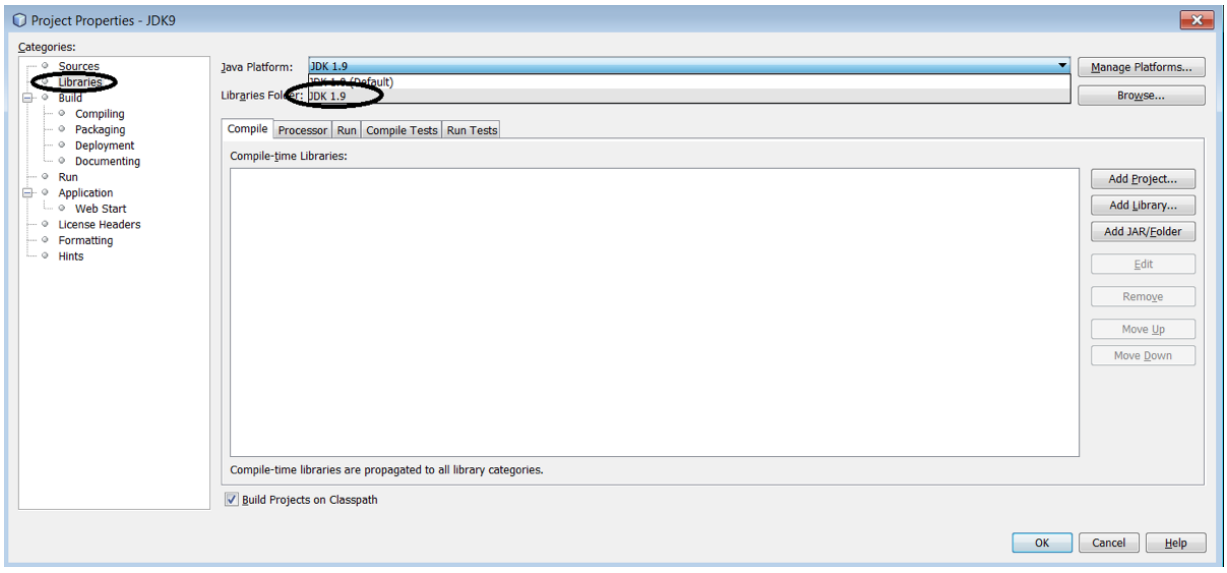
2. Select JDK folder from *<PATH>\client\build\windows-x86_64-normal-server-fastdebug* and click next.Select finish on opened dialog.



3. Right click on the newly created JDK project. In popup menu select *Set Configuration-> Customize*.

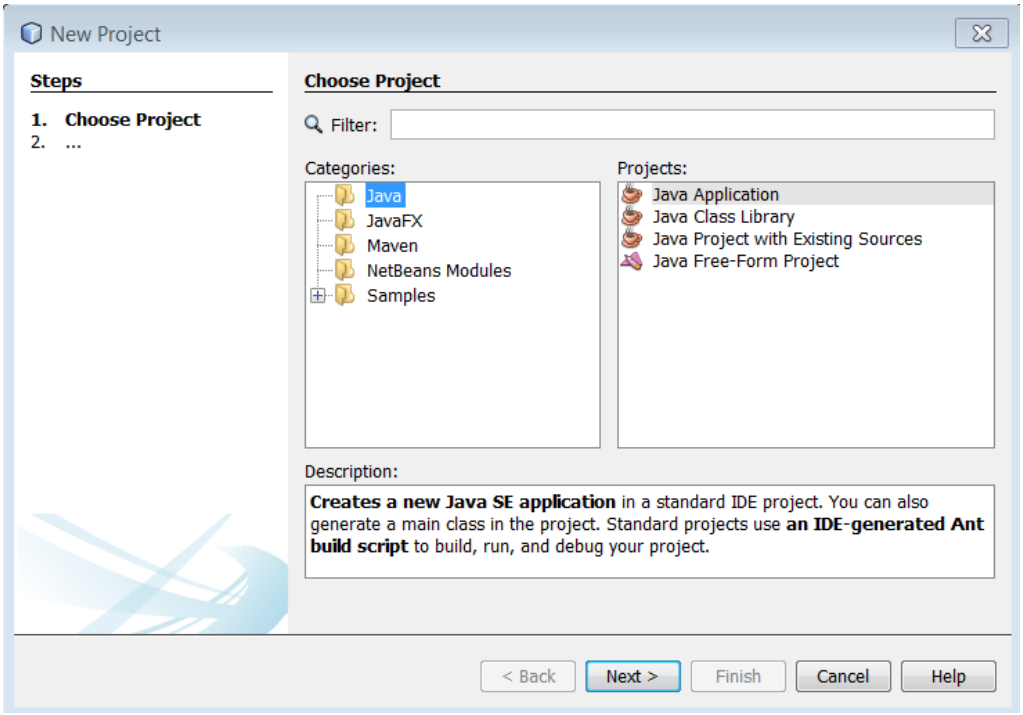


Clicking on customize will open following dialog.Make selections in this dialog as shown in snapshot and click ok

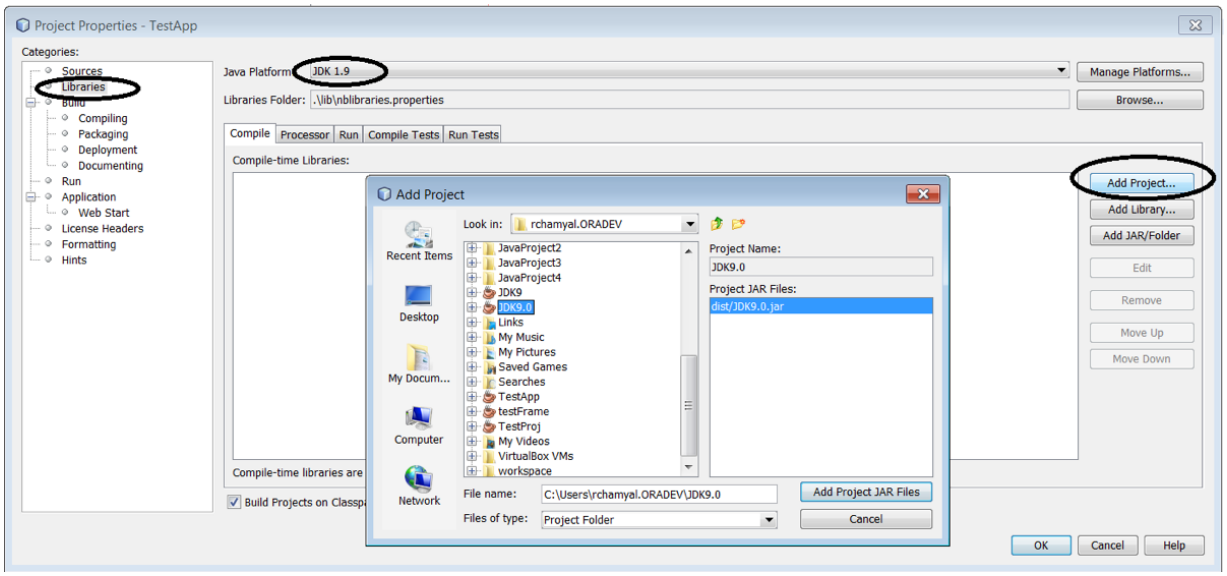


Netbeans: Creating JDK project with java sources / JDK platform

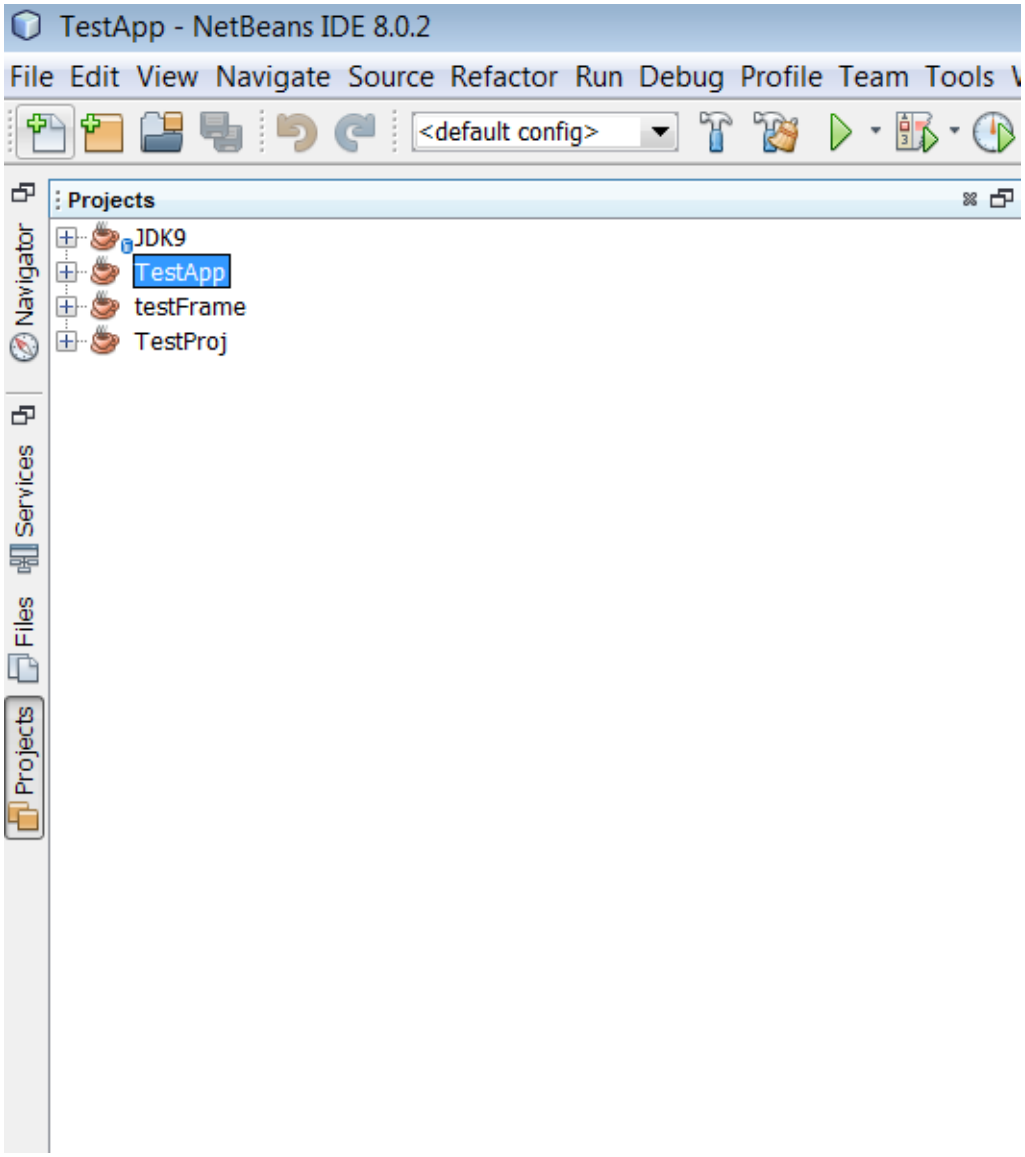
1.For debugging your java applications create a new normal java project.



2. Right click on the new JAVA project .In popup menu select *Set Configuration->customize*.Select below options in the newly opened dialog.



3. New Projects will be created under projects tab in netbeans.

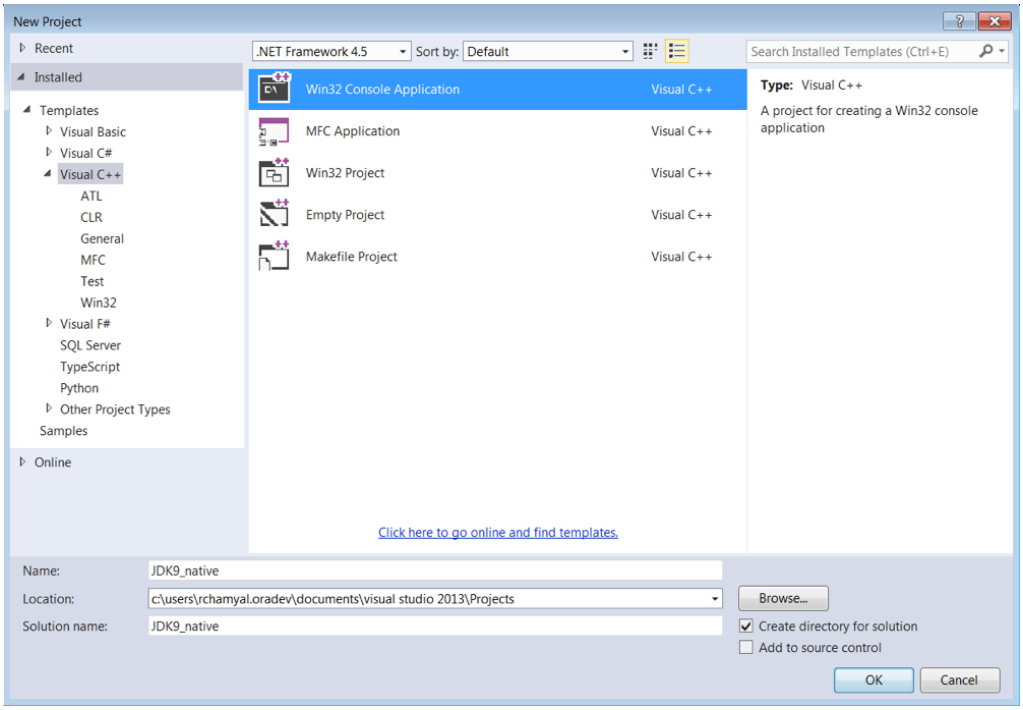


Visual Studio: Native code project setup for debugging

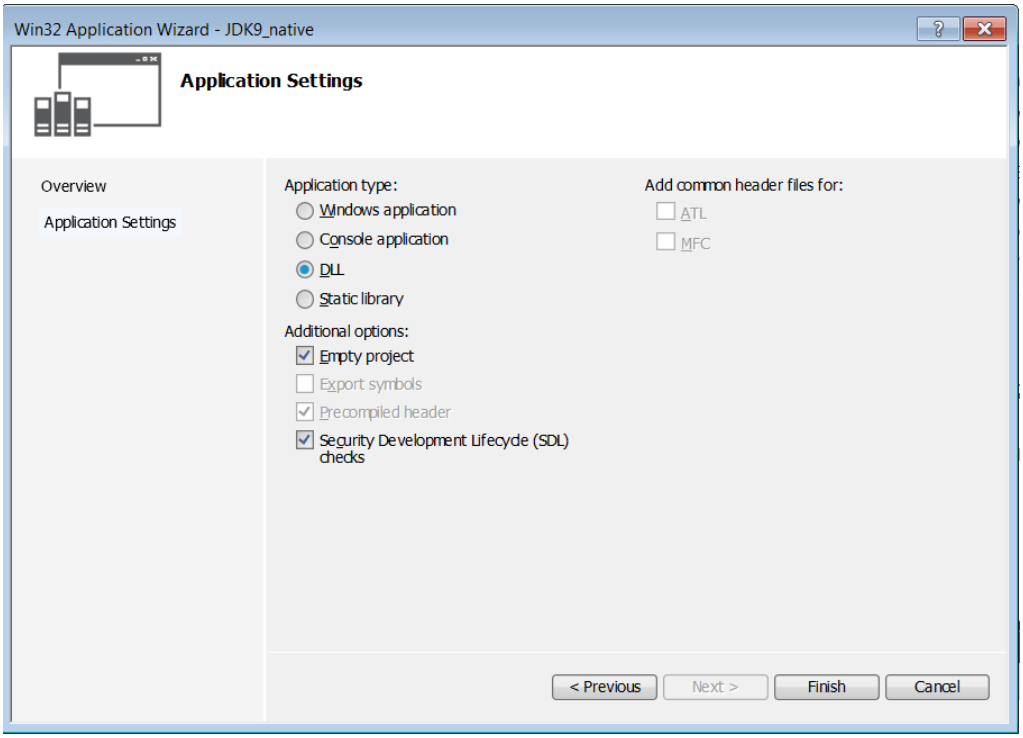
Windows

Visual Studio

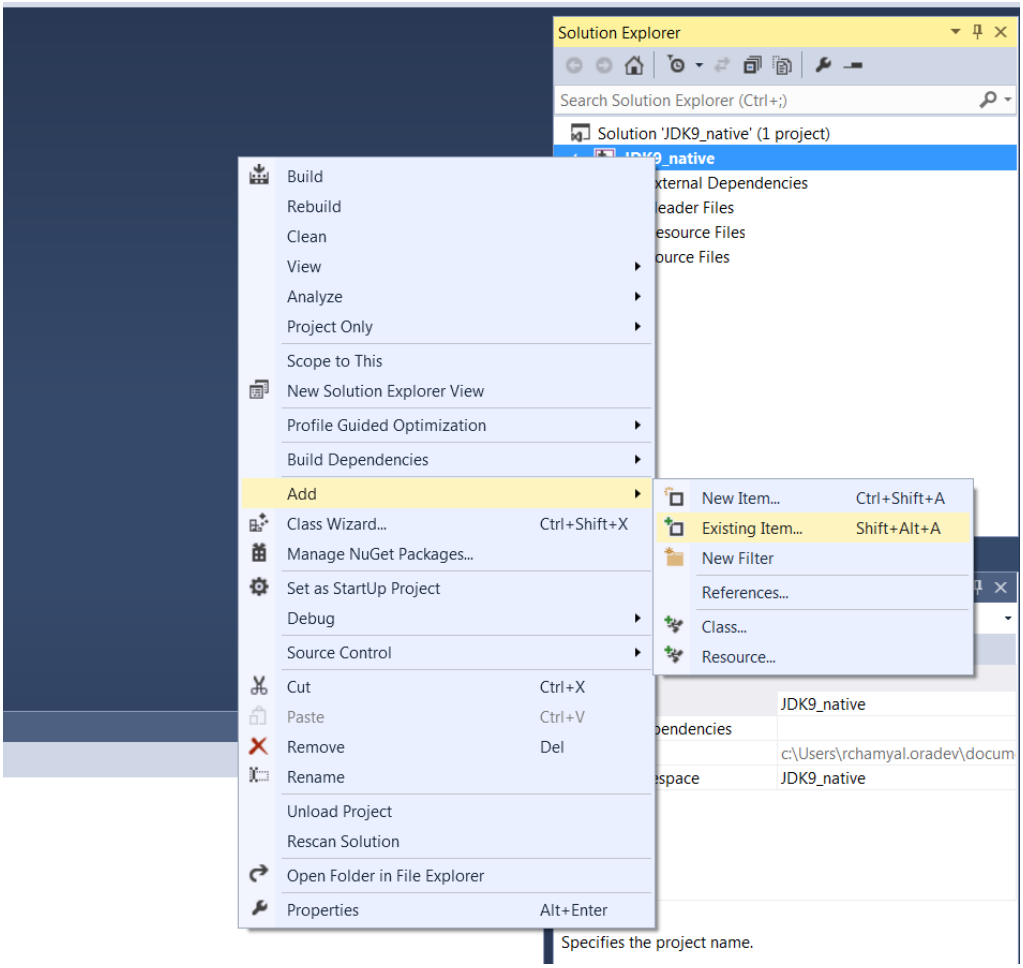
1. Create new Win32 Console Application and click ok and then click next



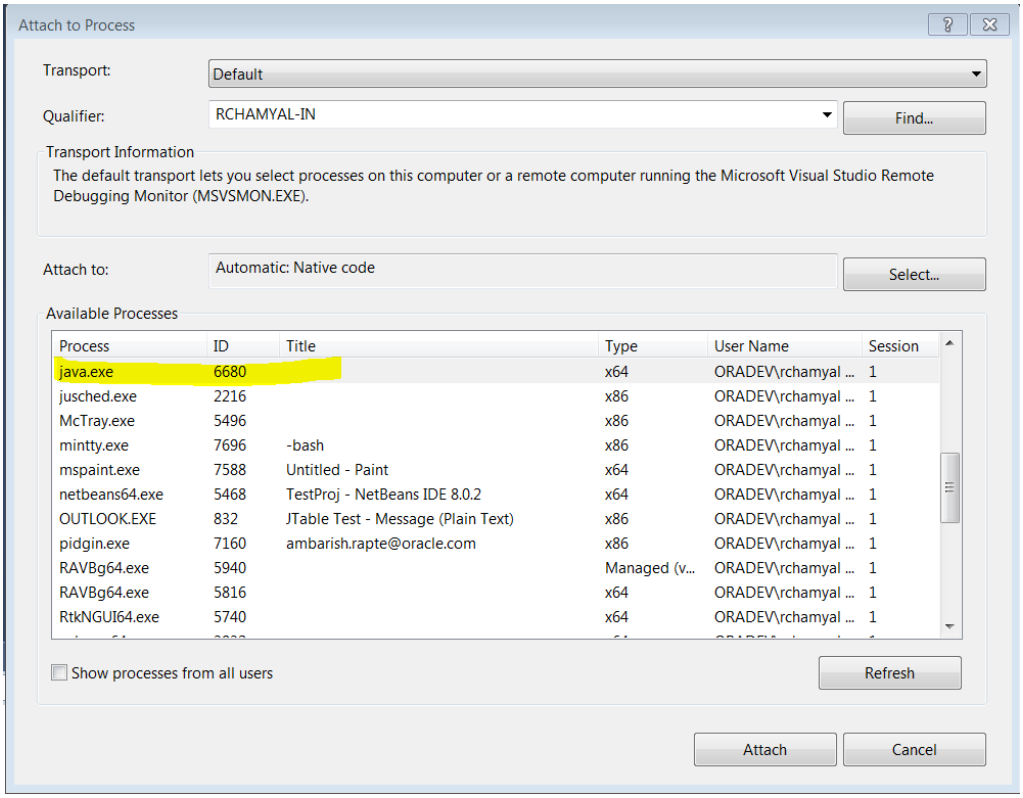
2. Make selections as in the dialog and click finish



3. Add required headers and source files as shown in snapshot.

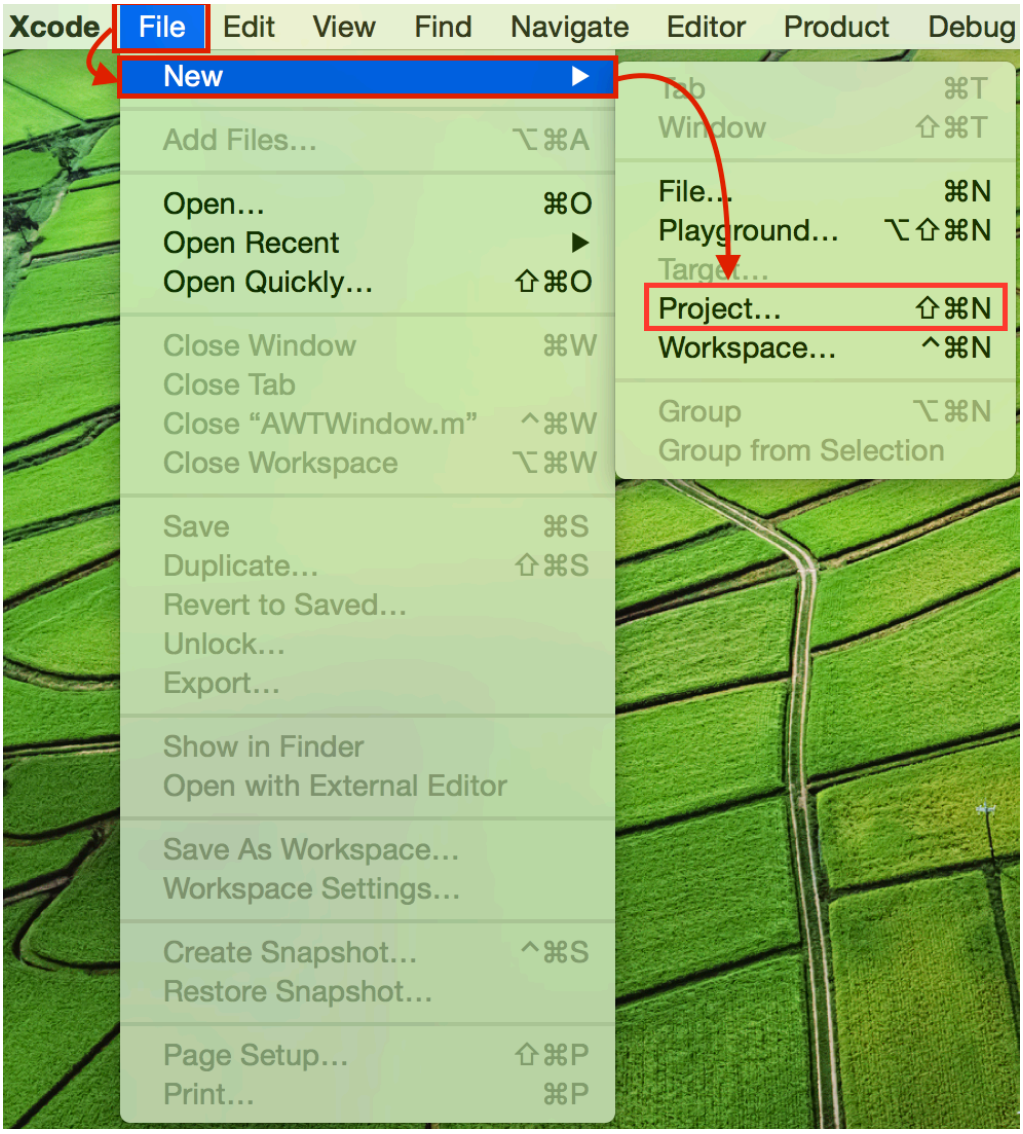


4. For debugging start a java process and attach to visual studio debugger. Click on Debug menu in main menu bar and select *Attach to Process...*

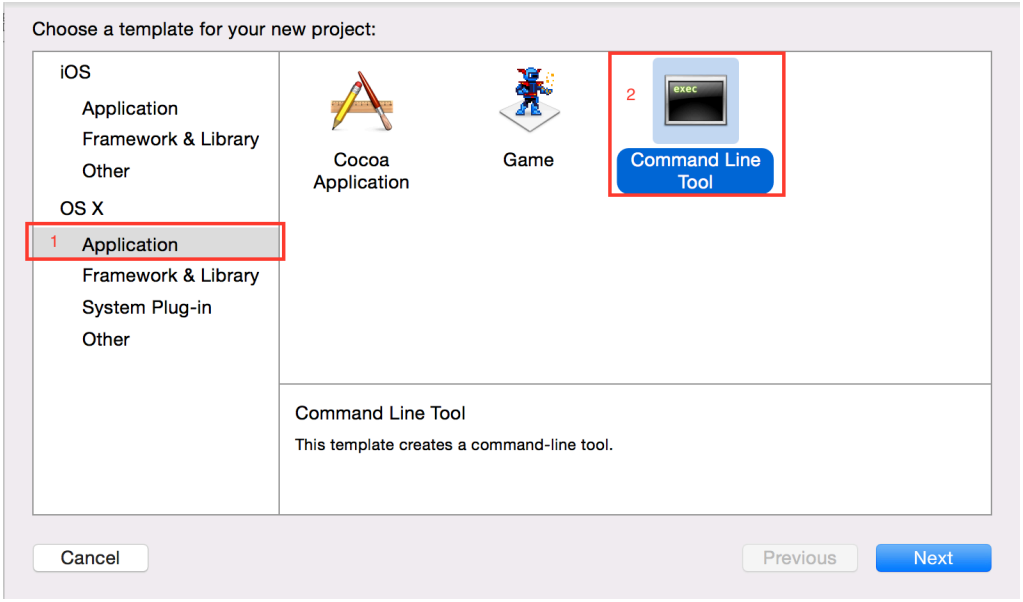


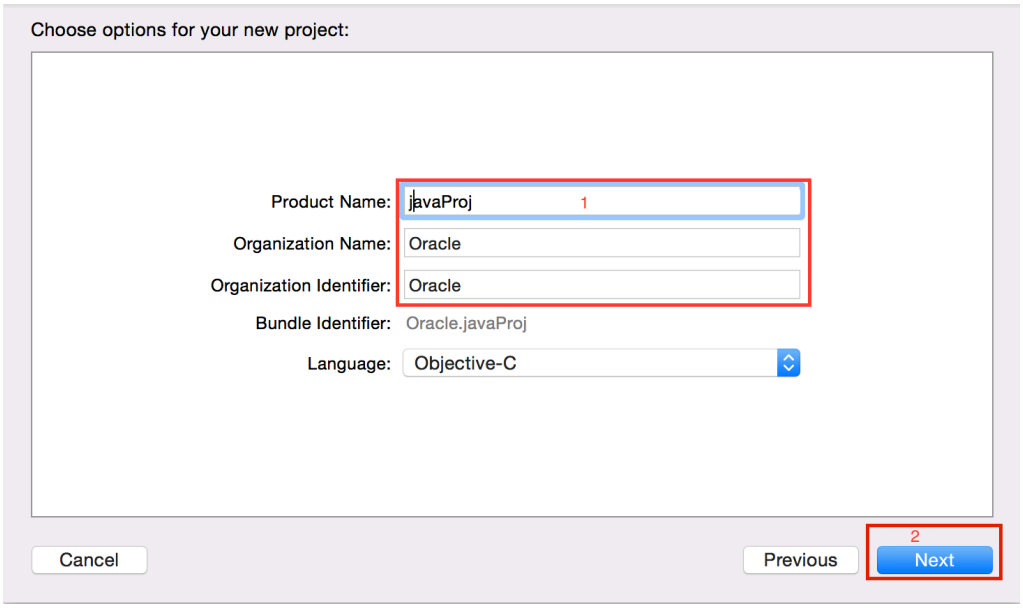
Mac OS X using Xcode

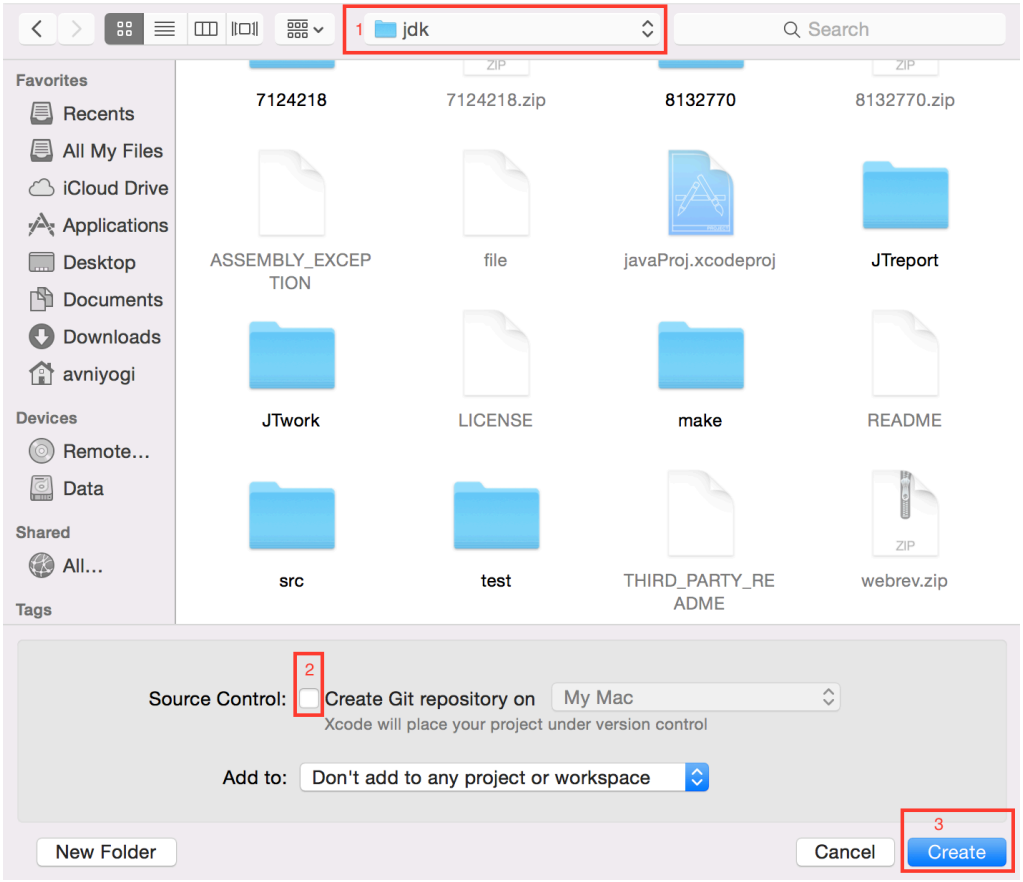
1) Open Xcode application on Mac OS X



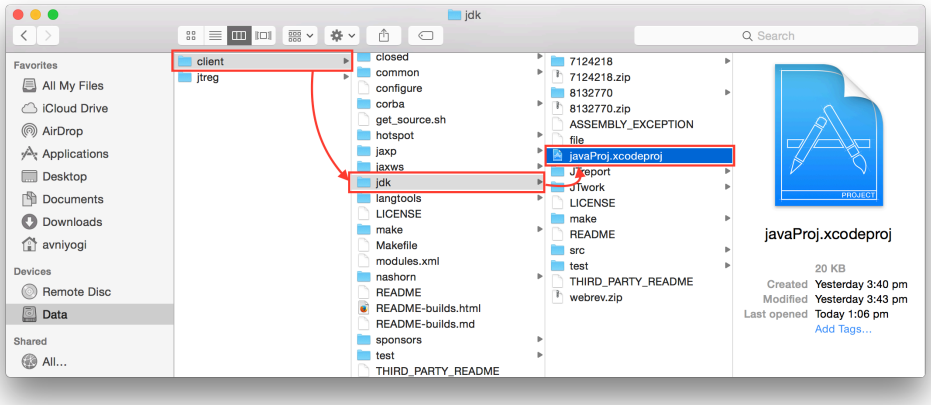
2) Create a "javaProj" (lets assume this is file name, other file name can be used appropriately. For the purpose of this document it is assumed to be javaProj.xcodeProj) in the below manner in steps indicated by order of annotation numerals:


Choose a template for your new project:
iOS
Application
Framework & Library
Other
OS X
1 Application
Framework & Library
System Plug-in
Other
Cocoa Application
Game
2
exec
Command Line Tool
Command Line Tool
This template creates a command-line tool.
Cancel
Previous
Next

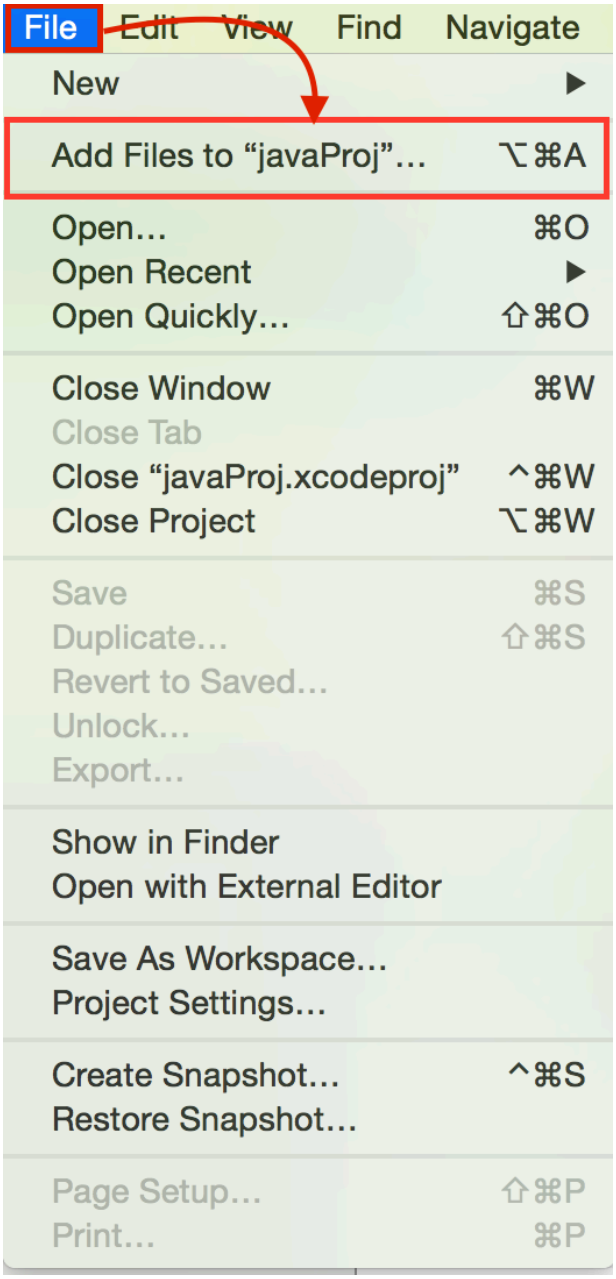

Choose options for your new project:
Product Name: javaProj 1
Organization Name: Oracle
Organization Identifier: Oracle
Bundle Identifier: Oracle.javaProj
Language: Objective-C
Cancel
Previous
2
Next

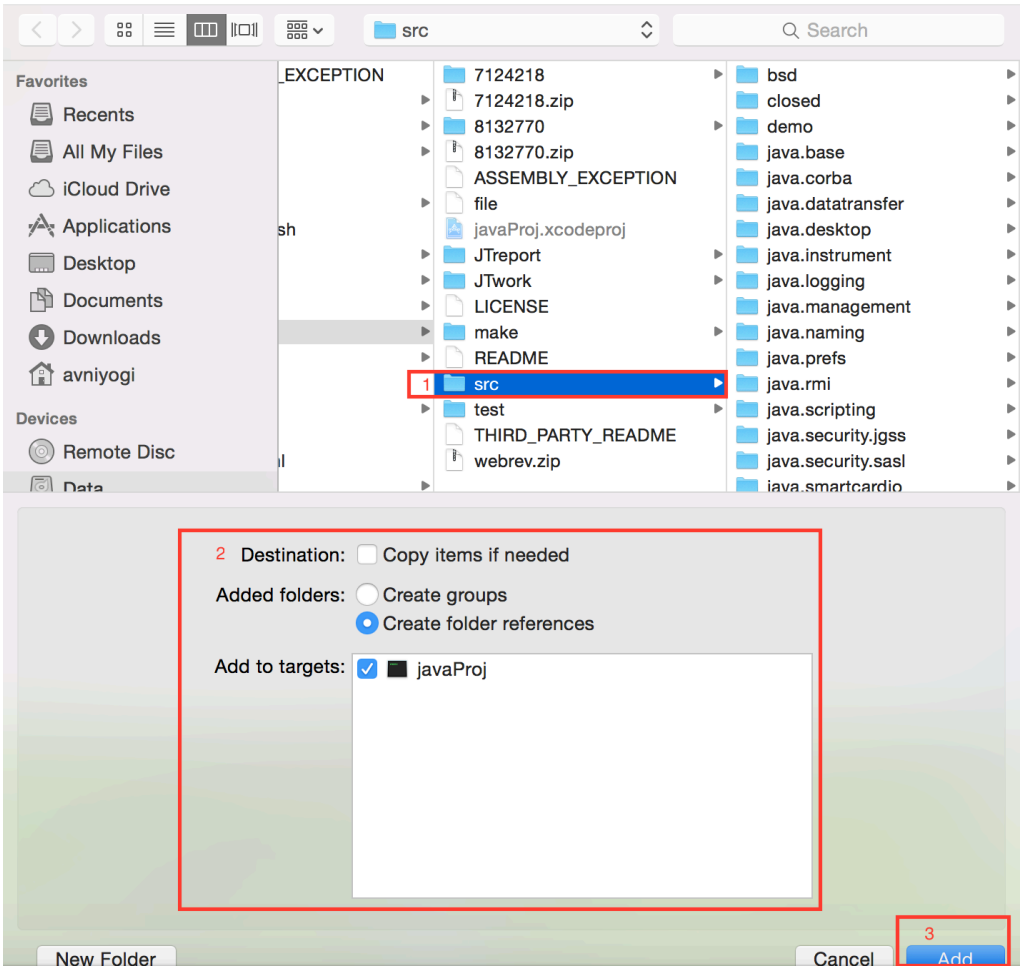


3) Delete auto created files from project menu and make sure the javaProj.xcodeproj is moved to below location.

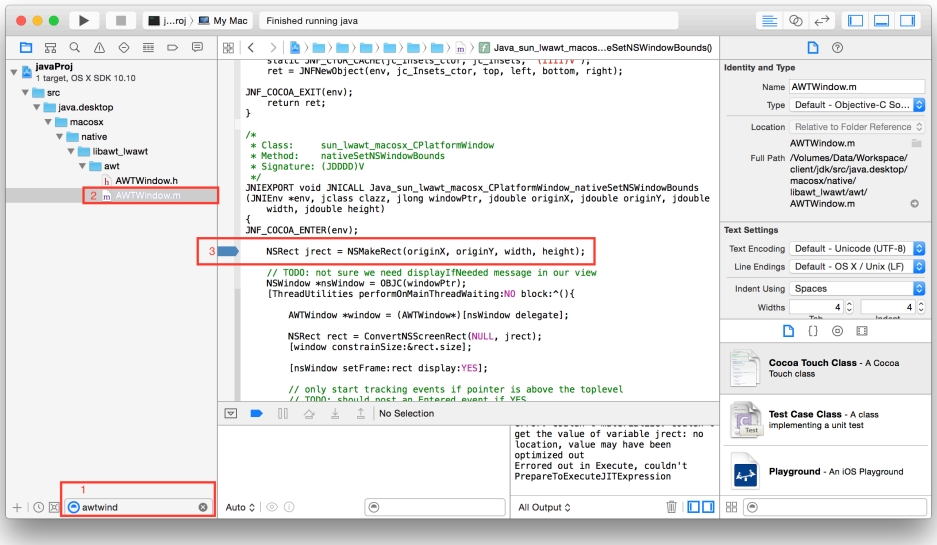


4) Add src folder in below manner:

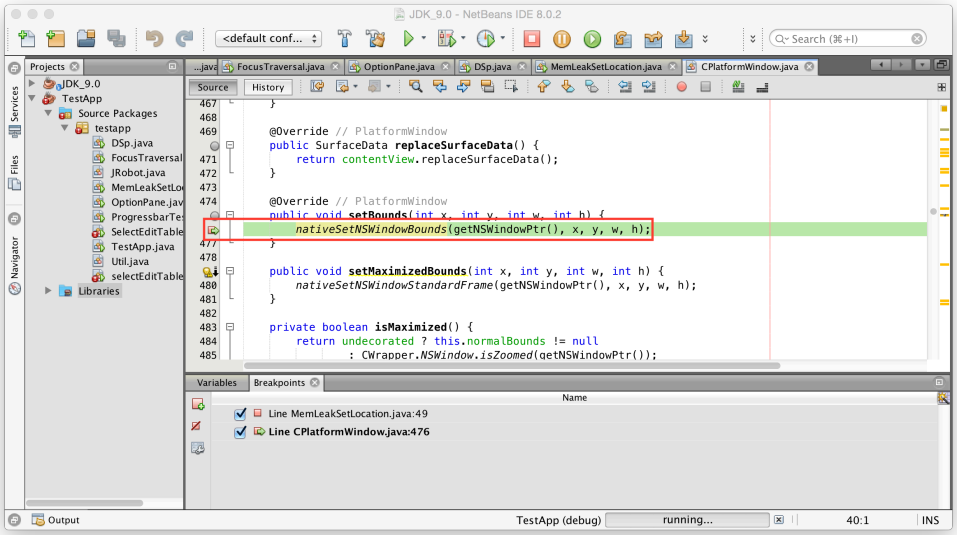

File
Edit
View
Find
Navigate
New
Add Files to “javaProj”... ⌥⌘A
Open... ⌘O
Open Recent
Open Quickly... ⇧⌘O
Close Window ⌘W
Close Tab
Close “javaProj.xcodeproj” ^⌘W
Close Project ⌥⌘W
Save ⌘S
Duplicate... ⇧⌘S
Revert to Saved...
Unlock...
Export...
Show in Finder
Open with External Editor
Save As Workspace...
Project Settings...
Create Snapshot... ^⌘S
Restore Snapshot...
Page Setup... ⇧⌘P
Print... ⌘P



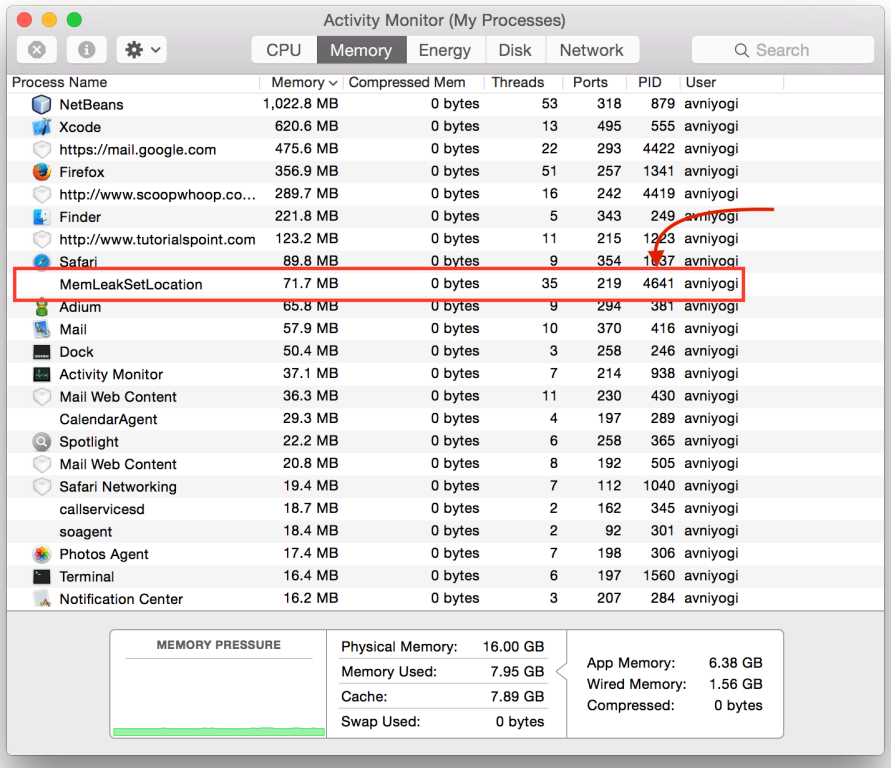
5) Add breakpoint in "required" Objective C (*.m) file. Note: Which file is required is based on understanding of JNI and some Spotlight Searching.



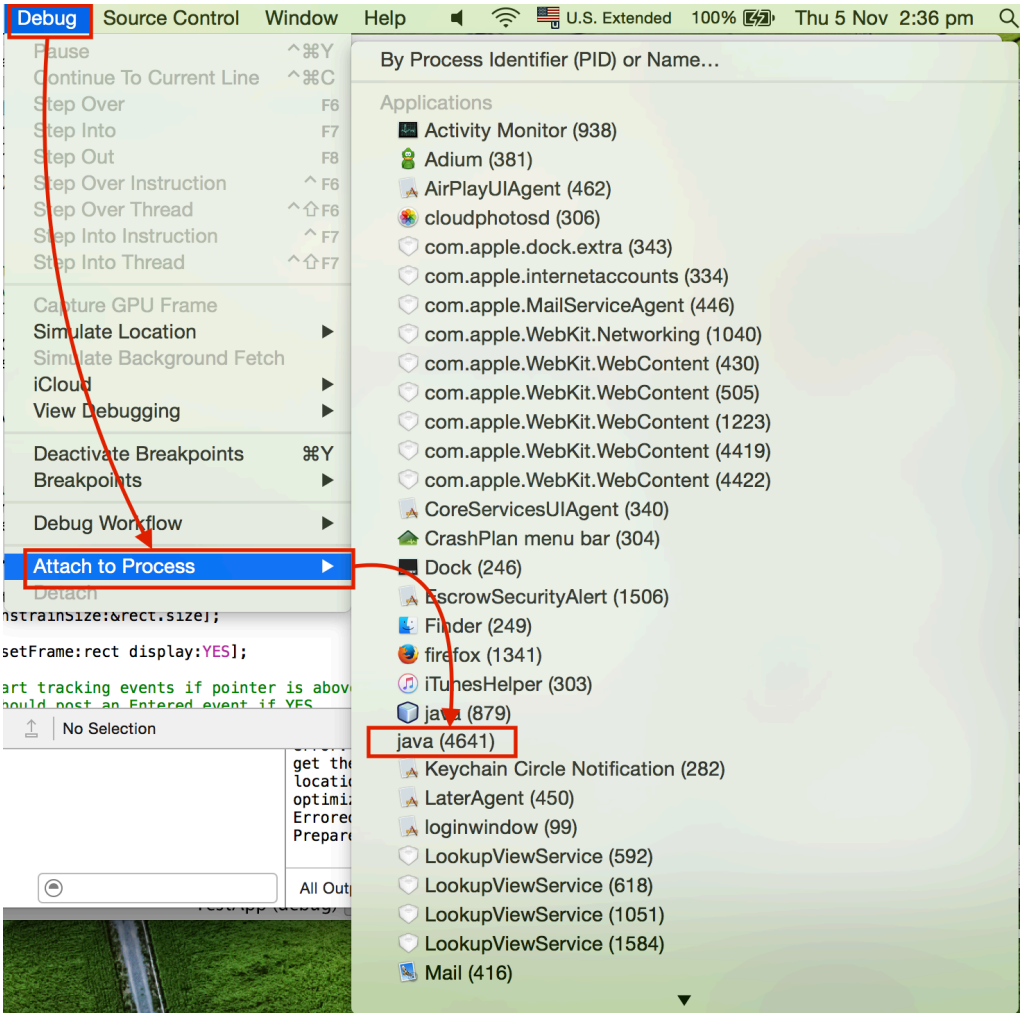
6) Add breakpoint to Netbeans project and make sure it is the native call method used by the running java file



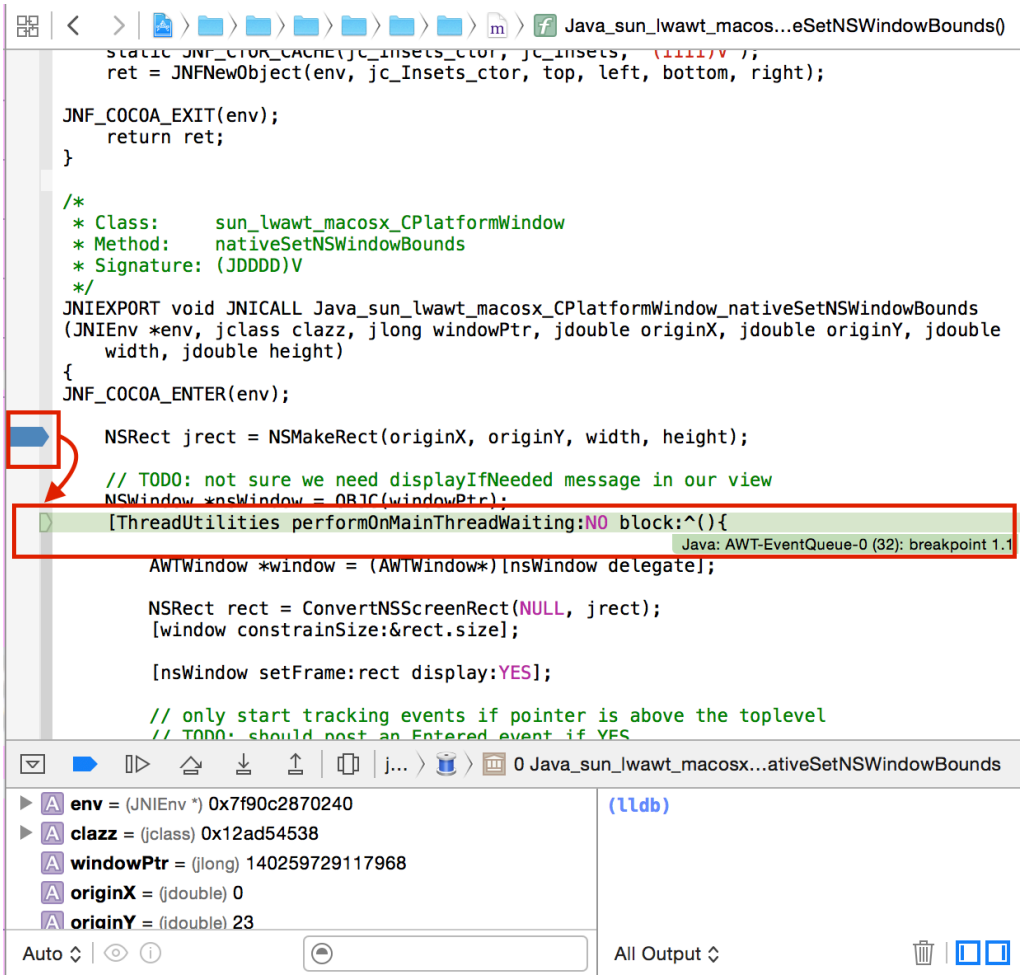
7) Look for process ID of running java file in Activity Monitor after running debugger in Netbeans. Note it.



8) Attach the java process to Xcode debugger in following manner.



9) Find the execution stop at breakpoint in Xcode



Note: Some values are not available with debugger. Use command line code like "po" to get the values of the same on the console.

JTREG Test Execution

Note : To run jtreg for JDK9 build having Jigsaw download jtreg from :
http://jre.us.oracle.com/java/re/jtreg/4.2/promoted/latest/bundles/

Setup

1. General information can be found at http://openjdk.java.net/jtreg/index.html
2. Download jtreg [jtreg4.1-b12.tar.gz] from, https://adopt-openjdk.ci.cloudbees.com/job/jtreg/lastSuccessfulBuild/artifact/
3. Extract the jtreg at convenient location on machine.

Execution

1. Open terminal and navigate to jdk/test folder. ex. *cd ~/workspace/jdk1.9/client/jdk/test/*
   Run a test using jtreg command jtreg -jdk:*test-jdk-ptah* *test-or-folder...*
   where,
   a. **jtreg,** is the bin/jtreg in extracted jtreg package.
   b. ***test-jdk-ptah**, is the absolute path to locally built java home folder to be tested against.*
   c. **test-or-folder**, is the path of test folder, or individual test file to be executed.
   Example:
   ~/workspace/jtreg/bin/jtreg -jdk:/home/user/workspace/jdk1.9/client/build/linux-x86-normal-server-fastdebug/jdk/ java/awt/Mixing/AWT_Mixing/OpaqueOverlappingChoice.java

3. Test report will be generated at, *jdk1.9/ client/ jdk/ test/* JTreport

More information on running jtreg can be found in http://openjdk.java.net/jtreg/runtests.html

JCK Test Execution

1. Download JCK-runtime-9.jar
http://jre.us.oracle.com/java/re/jck/9/promoted/latest/bundles/JCK-runtime-9.jar

2. Download JCK-extra-9.zip
http://jre.us.oracle.com/java/re/jck/9/promoted/latest/extra/bundles/JCK-extra-9.zip

3. Put files downloaded above to the same directory

4. Extract the jar content
> <jdk> / bin / java -jar JCK-runtime-9.jar

5. Unzip JCK-extra-9.zip

6. Build JDK images on your working JDK with the applied fix:
> make images

7. Run the JCK tests with the built JDK from images: (I believe it is placed in <build-jdk>*/build/*<os-normal-server>/images/jdk, but it could be changed)
> <jdk from images>/bin/java -jar JCK-runtime-9/lib/jtjck.jar "api/javax_swing/JTabbedPane"
or using the -jdk option:
> <jdk>/bin/java -jar JCK-runtime-9/lib/jtjck.jar -jdk <jdk from images> "api/javax_swing/JTabbedPane"

Note : above example is for api/javax_swing/JTabbedPane JCK tests. Please refer to JCK-runtime-9/tests directory to check and provide path of the particular set of tests you are interested in.

To run interactive tests (and with added vm options), you need to execute the following

bin/java -jar JCK-runtime-12/lib/jtjck.jar -vmoptions:-Dswing.defaultlaf=com.sun.java.swing.plaf.motif.MotifLookAndFeel -v -k:interactive "api/javax_swing/interactive/JTextAreaTests.java

JPRT (JDK Putback Reliability Testing) build

Steps to do JPRT build

1. Install autofs: sudo apt-get install autofs
2. Modify /etc/auto.master file and add "/net /etc/auto.net" as the first entry
3. Restart autofs service: /etc/init.d/autofs restart
4. Check to see if you can access /net/scaaa637.us.oracle.com/ [reboot if autofs restart does not work]
   Execute command "/net/scaaa637.us.oracle.com/export/archives/data/jprt/archive/west/dist/bin/jprt" to see if it can access jprt script. You should see this output
   a. JPRT Version: 3.3.227: (2015-11-15) Case Of the Ancient Romeo [53a2573e96dc]
      JPRT System: west [scaaa563.us.oracle.com:5325]
      Java Version: 1.8.0_45
   Execute command from "client" directory or topmost directory of the client workspace
   a. /net/scaaa637.us.oracle.com/export/archives/data/jprt/archive/west/dist/bin/jprt submit -stree . -bo -lt -noqa -quickabortcount 1000 -email prasanta.sadhukhan@oracle.com
   b. the above command will only build the workspace for different targets. it will not run any tests. If you want to run all tests add parameters "-testset all" to the above commandline.
   c. It will bundle your source code and upload to the jprt server and start the jprt build and send you notification to your email once it completes which might take 2-3 hrs depending on slowness of server.
   d. you can see the progress in http://scaaa637.us.oracle.com/archive/west.html [search for you email id]

- N.B: Alternatively, if scaaa637.us.oracle.com server is not working, we can use sthnas01.se.oracle.com.
- The jprt path to be used : /net/sthnas01.se.oracle.com/export/archives/stockholm/dist/bin/jprt
- The http server to see the progress will then be: http://sthjprt.uk.oracle.com/archives/2016/12/2016-12-28-

112832.PRSADHUK.jdk8u-dev [where PRSADHUK is the user id used to submit the job and jdk8u-dev is the workspace to be compiled] used
- Run 7u-cpu and 6u-cpu on Legacy queue: http://jdk-services.us.oracle.com/jprt/archive/legacy.html [location is: /net/scanas415.us.oracle.com/export/jprt_storage/archive/legacy/dist/bin/jprt]
- HINT: Use 'jprt rerun [-comment <arg>] <JOB ID> ' to rerun this job if you suspect the verdict was wrong. Example: '/net/scanas415.us.oracle.com/export/jprt_storage/archive/legacy/dist/bin/jprt rerun 2017-03-28-094346.prem.jdk7u-cpu'
- HINT: Delete install folder i.e., ../jdk7u-cpu/install if getting errors as follows: LINK : fatal error LNK1181: cannot open input file 'c:/jprt/T/P1/064108.manajit/s/build/windows-x86-normal-clientANDserver-release/install/obj/installer-full.exe/full.res' make[2]: *** [/cygdrive/c/jprt/T/P1/064108.manajit/s/build/windows-x86-normal-clientANDserver-release/install/bin/installer-full.exe] Error 157 make[2]: Leaving directory `/cygdrive/c/jprt/T/P1/064108.manajit/s/install/make' make[1]: *** [/cygdrive/c/jprt/T/P1/064108.manajit/s/build/windows-x86-normal-clientANDserver-release/install/_install.timestamp] Error 2 make[1]: Leaving directory `/cygdrive/c/jprt/T/P1/064108.manajit/s/jdk/make/closed' make: *** [installer-only] Error 2
- HINT:If Targets failed due to INFRA-8393(INFRA-8393 - LongLink: typeflag 'L' not recognized on JPRT legacy OPEN). i.e., Target solaris_sparc_5.8-fastdebug sick. Target solaris_sparc_5.8-product sick. Target solaris_sparcv9_5.8-fastdebug sick. Target solaris_sparcv9_5.8-product sick. Target solaris_i586_5.8-fastdebug sick. Target solaris_i586_5.8-product sick. then use the workaround : export TAR_OPTIONS="--format=ustar" before running the jprt submit command.

Mach5

Run Mach5 client

Mach 5 User Guide for Building and Testing JDK 10

1. **Configuring mach5 ‘s jobs :**
   Configuration file(client-test.js) location: <jdk-source>/ client/ closed/ task-definitions/ jobs/

Commit - howto

1. Download defpath.py from http://jre.us.oracle.com/java/jdk/lib/hgext/
2. Place defpath.py in your home folder (or to any directory of your choice)
3. Add defpath = ~/defpath.py in your ~/.hgrc file under [extensions]
   so you should have something like this
   [extensions]
   trees = ~/trees.py
   jcheck = ~/jcheck.py
   defpath = ~/defpath.py

   [hooks] // this will help in catching spaces and other anomalies during commit before push

   pretxnchangegroup.jcheck = python:jcheck.hook
   pretxncommit.jcheck = python:jcheck.hook

4. Clone a fresh repo for your commits
5. Run "hg defpath -du <username>" in all your freshly cloned repo like in
   jdk, jdk/test/closed from where you need to commit files

6. Check that in <repo>/.hg/hgrc file you should now have this
   default-push = ssh://<username>@closedjdk.us.oracle.com/jdk9/client/jdk/test/closed

7. Download jdk.patch and do "hg import --no-commit jdk.patch" to import the patch
8. If there is a new binary file like image, hg import will not import the binary file we have to explicitly copy the binary file to proper path in the workspace which we are using to commit the changes.
   After this execute "hg add <new_file>". Also make sure that binary file to be added is not executable. If it is executable

"hg push" will fail, so remove executable permission of file and then perform "hg add <new_file>"

9. Create a comments.txt with following information
   <bugid>: <description>
   Reviewed-by: <reviewer1>, <Reviewer2>
   Contributed-by: <emailid of contributer> [optional] (The developers guide says the contributed-by line is just in case the author does not have commit rights/ author ID.)

10. Execute "hg commit -u <username of the fixer> -l comments.txt" [-u is needed if you are committing for someone else]
11. Execute "hg push"

12. Steps 9-10 can be done in tortoise hg. Install "sudo apt-get install tortoisehg"

PIT (sync and integration steps)

- **Sync from jdk/jdk to jdk/client**

1. Get a fresh clone of **http://closedjdk.us.oracle.com/jdk/client. cd client;**
2. Run "sh get_source.sh"
3. hg pull -u http://closedjdk.us.oracle.com/jdk/jdk
4. hg merge
5. hg commit -m Merge
6. cd open
7. hg pull -u http://hg.openjdk.java.net/jdk/jdk
8. hg merge
9. hg commit -m Merge
10. cd ..
11. mach5/bin/mach5 --remote-build [or use jib $ sh jib.sh mach5 -- remote-build --email prasanta.sadhukhan@oracle.com]
12. Now wait (30 mins or more) to be sure it builds.
13. CHECK INCOMING from client

    hg in
    cd open
    hg in

    If there are new changes you will need to pull (hg pull -u) and re-merge (hg merge) before you can push.

    Once that is done you can push
14. hg push;
15. cd open
16. hg push

Wait for nightly builds

- **Integrate from jdk/client to jdk/jdk**

1. Look at the client mach5 nightly build, eg http://java.se.oracle.com:10065/mdash/buildIds/2018-09-26-1022502.jdk-client-confidential_ww_grp.source
   Verify all builds + tests passed .
   If so prepare an integration workspace based on the revisions listed at that location (see Sources-> revisions)

   In the top-level repo of your local clone of http://closedjdk.us.oracle.com/jdk/client, copy-paste these commands to sync your forest to this version of the code:

   sh ./get_source.sh
   hg -R . update -c -r 46216031f21a7977727a34a4fb8ba671da27be4d
   hg -R open update -c -r 54937a08689bb21a60382be8663ce8fa532eaf99

   ---
2. Pull down the build artifacts to each platform on which we need to run the HEADFUL jtreg test suite.

   On EACH platform you will need a matching copy of the workspace because you need

the exact set of regression tests that are in the workspace corresponding to the builds.
3. Start a jtreg job on each platform

   jtreg -a -ignore:quiet -exclude:open/test/jdk/ProblemList.txt -exclude:closed/test/jdk/ProblemList.txt open/test/jdk:jdk_desktop closed/test/jdk:jdk_desktop
   This will take at least several hours on each platform (windows, linux, mac)
4. Inspect the results for each platform.
5. Many tests fail when run as a batch. Many tests fail or give errors regularly because we still need to update the ProblemList.txt files. For tests that you think should NOT have failed, run them one at a time.

   They'll probably pass this time, if so you can ignore them for now. If they fail REPEATEDLY, then file a new bug if there's none already, and add whatever bug you created or found to the problem list in the ususal way - platform-specific or generic, and get a code review

   If the bug is SERIOUS, discuss whether it is an integration stopper.This becomes more important the later you get in the rease
6. If all is good to integrate you need to sync the integration w/s with jdk/jdk
7. cd <integrationws>/client [**Note**: Ensure step 1 of updating the workspace to correct revision is performed before doing step 8]
8. hg pull -u http://closedjdk.us.oracle.com/jdk/jdk
9. hg merge
10. hg commit -m Merge
11. cd open
12. hg pull -u http://hg.openjdk.java.net/jdk/jdk
13. hg merge [N.B: You need to use "hg merge -r tip" if there are more commits after the nightly build or after the ws updation done above hg -R open update -c -r 54937a08689bb21a60382be8663ce8fa532eaf99]
14. hg commit -m Merge
15. ~/mach5/mach5/bin/mach5 remote-build
16. Now, wait for that job to complete.

    When it has completed, on EACH platform, pull down

    (1) the JDK build artifact as .tar.gz or .zip

    (2) The tests + demos artifact which contains SwingSet2, J2Demo

    unpack these where ever you want

    (3) Run systematically through these demos, verifying options, text input, mouse input

    each L&F etc etc

    $ cd jdk/demo/jfc/SwingSet2

    $ /../../bin/java -jar SwingSet2.jar

    $ cd ../J2Ddemo

    $ ../../../bin/java -jar J2Ddemo.jar

    If all works then you may INTEGRATE : - push to jdk/jdk
17. HOWEVER, most likely someone pushed to jdk/jdk since you did so first check
    $ hg in http://closedjdk.us.oracle.com/jdk/jdk
    comparing with http://closedjdk.us.oracle.com/jdk/jdk
    searching for changes
    no changes found
18. cd open; hg in http://hg.openjdk.java.net/jdk/jdk

    comparing with http://hg.openjdk.java.net/jdk/jdk
    searching for changes
    changeset: 51938:760ca4ba79ce
    tag: tip
    parent: 51936:11fd6c8188d9
    user: gadams
    date: Thu Sep 27 07:33:13 2018 -0400
    summary: 8210984: [TESTBUG] hs203t003 fails with "# ERROR: hs203t003.cpp, 218: NSK_CPP_STUB2 (ResumeThread, jvmti, thread)"
19. In which case
    (1) sync changes from master again, remerge, or rebase

$ hg pull -u http://hg.openjdk.java.net/jdk/jdk
pulling from http://hg.openjdk.java.net/jdk/jdk
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 2 changes to 2 files (+1 heads)
abort: not updating: not a linear update
(merge or update --check to force update)

(2) verify at least locally this still builds

(3) Hope no one pushed AGAIN in the 15 mins since you just synced.

If they have go back to step (1) and seriously consider rebase to avoid

more merge changesets.

(4) Once all is in sync push open+closed

20. hg push ssh://psadhukhan@closedjdk.us.oracle.com/jdk/jdk
21. cd open; hg push ssh://psadhukhan@hg.openjdk.java.net/jdk/jdk

[N.B: you need to use "hg push -r tip" if the jdk/client actual tip has more commits than local jdk/client ws which is updated to -r 54937a08689bb21a60382be8663ce8fa532eaf99]

- **Integration is DONE, Sync again from jdk/jdk to jdk/client so that both master & client are on same page.**

1. Immediately sync jdk/jdk to jdk/client
2. cd sync/client
3. Run "sh get_source.sh"
4. hg pull -u http://closedjdk.us.oracle.com/jdk/jdk
5. hg merge
6. hg commit -m Merge
7. cd open
8. hg pull -u http://hg.openjdk.java.net/jdk/jdk
9. hg merge
   hg commit -m Merge
10. hg push
11. cd ..
12. hg push
    pushing to ssh://psadhukhan@closedjdk.us.oracle.com/jdk/client

Building JavaFX

- Detailed build instructions for OpenJFX are available at: https://wiki.openjdk.java.net/display/OpenJFX/Building+OpenJFX.
- The below steps are an example of using those instructions for Mac (Mac OS 10.12) and Windows. It also contains settings and test program execution examples.

Clone workspace

a. hg clone http://closedjdk.us.oracle.com/openjfx/jfx-dev
b. cd jfx-dev
c. hg clone http://hg.openjdk.java.net/openjfx/jfx-dev/rt
d. hg clone http://closedjdk.us.oracle.com/openjfx/jfx-dev/rt-closed

2. Set gradle (4.4+) latest in path - https://gradle.org/releases

3. Set ant (1.9.7+) latest in path - http://ant.apache.org/bindownload.cgi
   Set boot jdk to build OpenJFX - https://java.se.oracle.com/artifactory/re-release-local/jdk/9/

   a. Typical set of exports for step, 2,3,4 would be as [Windows]

      export JDK_HOME="D:\\jdk_bundle\\9\\181\\jdk-9"

      export JAVA_HOME="D:\\jdk_bundle\\9\\181\\jdk-9"

      export JAVA_BIN="D:\\jdk_bundle\\9\\181\\jdk-9\\bin"

      export GRADLE_HOME="D:\\fx\\tools\\gradle-4.4"

      export ANT_HOME="D:\\fx\\tools\\apache-ant-1.9.7"

      export PATH=${JAVA_BIN}:${GRADLE_HOME}\\bin:${ANT_HOME}\\bin:${PATH}
5. Additionally set the following env variable to allow gradle to run with the latest JDK 9 as boot JDK:

   export _JAVA_OPTIONS="-Dsun.reflect.debugModuleAccessChecks=true --add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.util=ALL-UNNAMED --add-opens=java.base/java.lang.invoke=ALL-UNNAMED --add-opens=java.base/java.io=ALL-UNNAMED --add-opens=java.base/java.util.concurrent=ALL-UNNAMED --add-opens=java.base/java.text=ALL-UNNAMED"

   Set proxy for gradle if you are connected to vnc:
   a. For Win & Linux, execute gradle command as : `gradle -Dhttps.proxyHost=www-proxy.idc.oracle.com -Dhttps.proxyPort=80`
      This command is required only once, afterwards only `gradle` command is sufficient.
   b. For Mac OS:
      Go to your home folder:(For Windows C:\Users\<user>.ORADEV\ )
      cd ~/
      .gradle folder should be available in this folder. Create gradle.properties file inside .gradle folder and add the below proxy configuration:
      systemProp.proxyHost=www-proxy.idc.oracle.com
      systemProp.proxyPort=80
      cat .gradle/gradle.properties
      systemProp.proxyHost=www-proxy.idc.oracle.com
      systemProp.proxyPort=80

   Build OpenJFX
   a. cd rt
   b. gradle (or gradle sdk)

   Info:
   a. _JAVA_OPTIONS is available in file "README-java-options"
   b. Sample source location: rt/apps/toys/Hello/src/main/java/hello/
   c. Mac glass code location: rt/modules/javafx.graphics/src/main/native-glass/mac
   d. Java glass code location: rt/modules/javafx.graphics/src/main/java/com/sun/glass/ui
   e. Java glass peer code location: rt/modules/javafx.graphics/src/main/java/com/sun/glass/ui/mac

Working with stand alone JavaFX & JDK 11

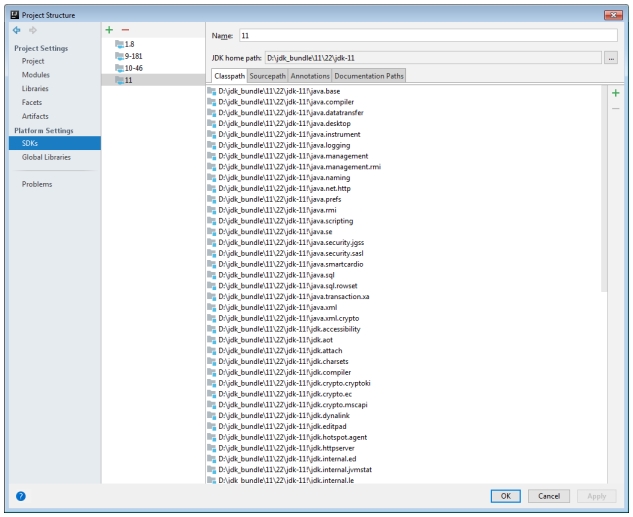
Background

Since JDK 11, Java FX is removed from JDK build, which means JavaFX programs will not compile or execute with JDK 11 build.

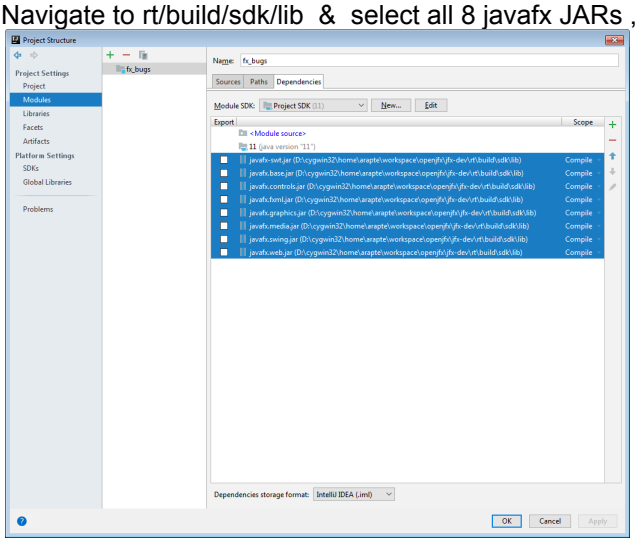
Compile & Execute JavaFX program on terminal

1. Compile the program : javac @"PATH_TO /jfx-dev/rt/build/compile.args" testApp.java
2. Execute the program : java @"PATH_TO /jfx-dev/rt/build/run.args" testApp

IntelliJ setup to Compilie JavaFX program



1. Setup JDK 11 in IntelliJ → (Image)
2. Open Module settings of project of JavaFX project.
3. Navigate to Modules → Dependencies.
4. Click + icon and select JARs or Directories.
5. Navigate to rt/build/sdk/lib & select all 8 javafx JARs , click Ok. → (Image)



6. The Java FX programs should compile & execute in IntelliJ IDE.
7. The FX source links might get disturbed with this setting, re link the FX source to navigate to files.
8. Debugging may not work with this setup, Please update the steps in case you find solution.

Testing Public member additions to Java FX

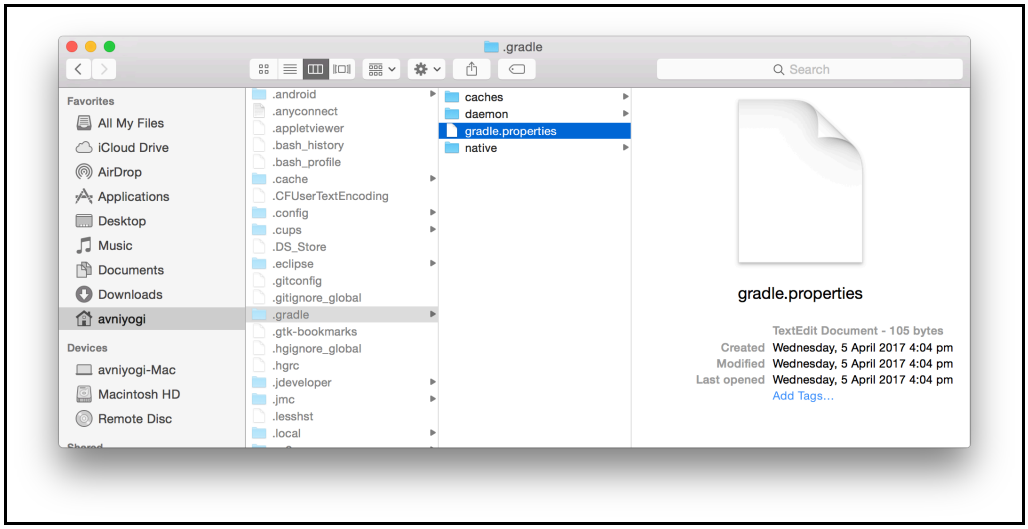
Below are steps for verifying public additions to Java FX without building JDK.
Below steps are command line to be executed on terminal.

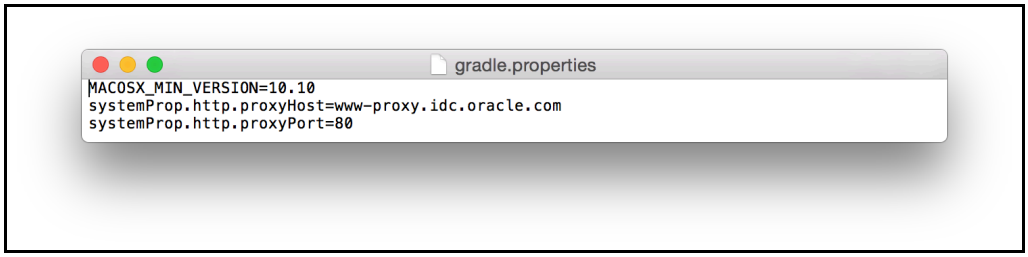
1. Add public API / classes to Java FX
2. Build the Java FX normally without building JDK source
3. Write a test application which uses the newly added public entities.
   Compile & execute the program using jdk bundle
   a. Compile the program : javac @"PATH_TO / jfx-dev/rt/build/compile.args" testApp.java
   b. Execute the program : java @"PATH_TO / jfx-dev/rt/build/run.args" testApp

The program should compile and execute.
Currently IntelliJ IDE does not have any option to solve this problem.
If the compile.args can be provided through IntelliJ or any other IDE, then it should be possible to debug.

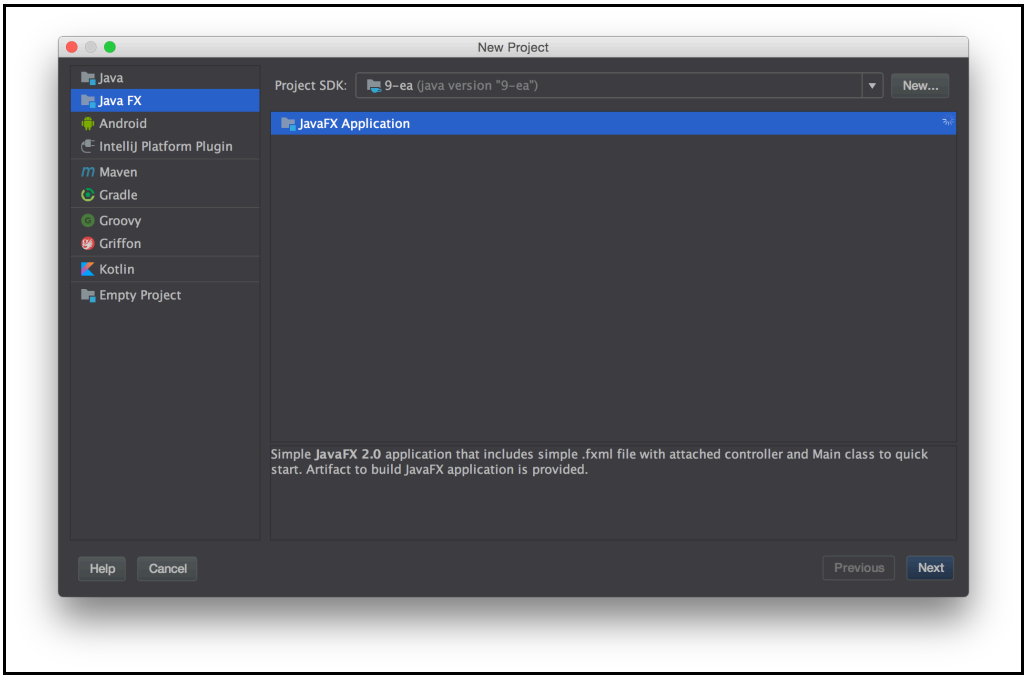
Configure IntelliJ IDE for Java FX

1. This configuration assumes the following:
    a. Java FX is setup for respective OS versions with steps available for perusal at this link.
    b. Java FX with JDK9 is shown here
    c. Java FX is running well from Terminal/Console/Command Line
    d. All environment variables are setup
    e. All gradle properties are setup. For Example:

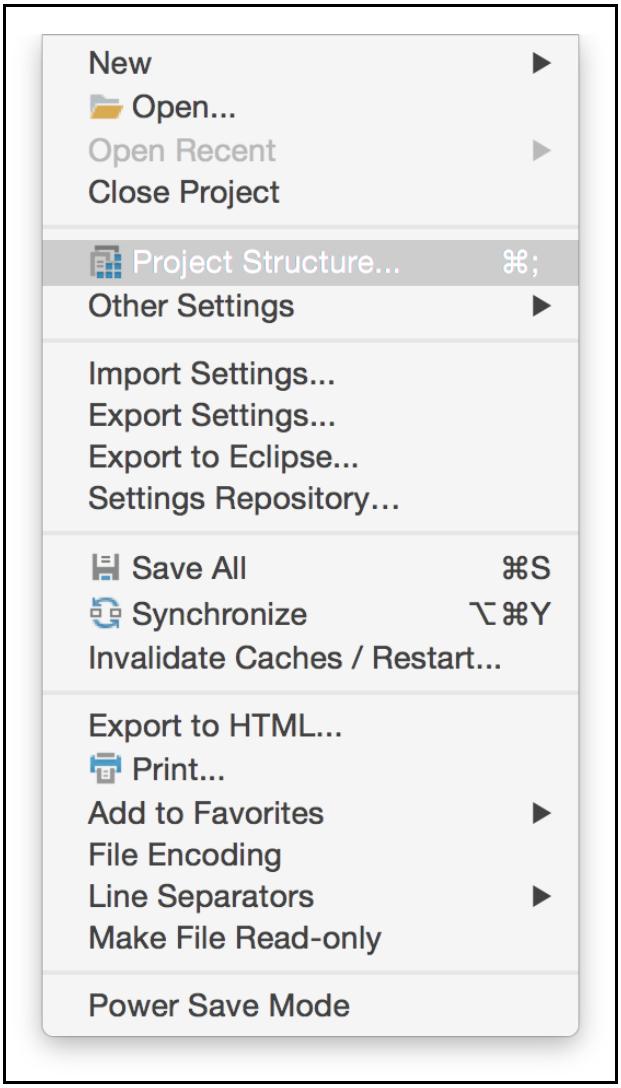


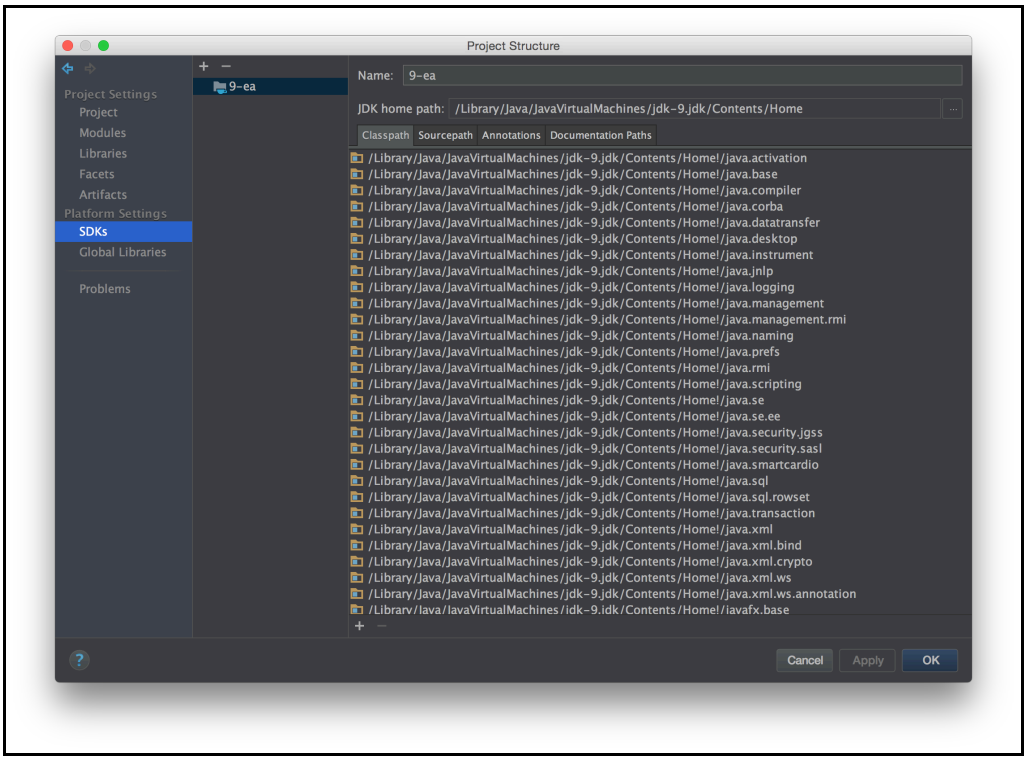


2. Download the latest IntelliJ IDE from here and install it.
3. Open IntelliJ IDE (configure LAF and other settings as per requirement) and select **New Project**.
4. Select **Java FX** as below:



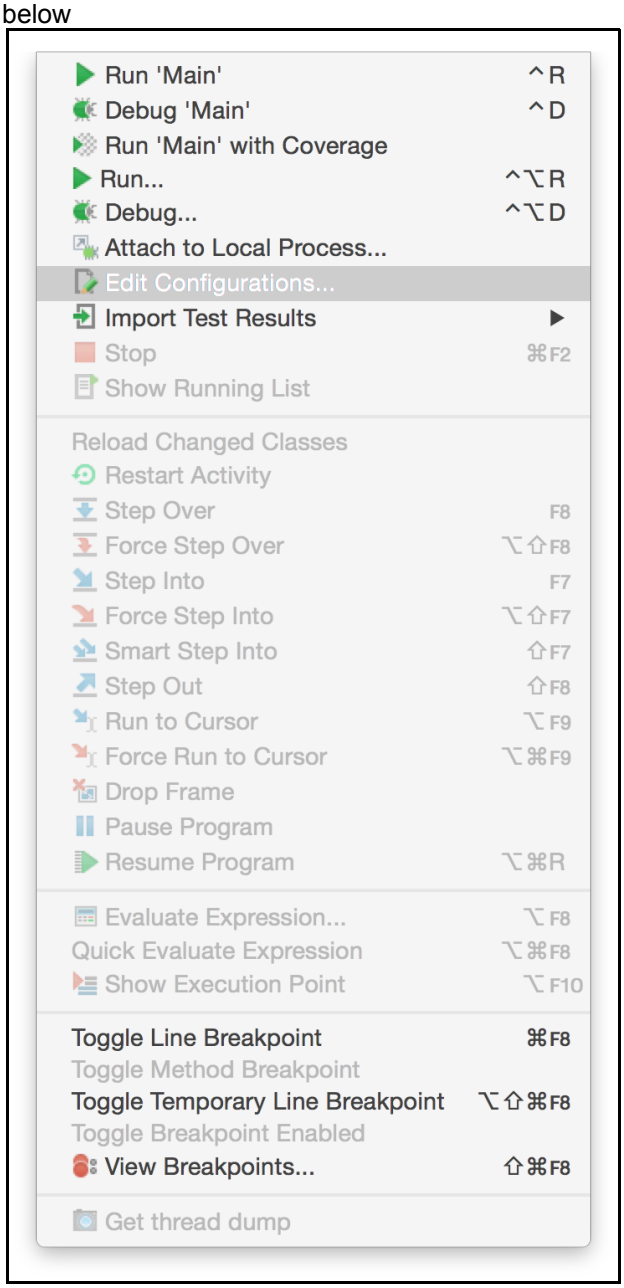
5. Create TestApp and select Project Structure:

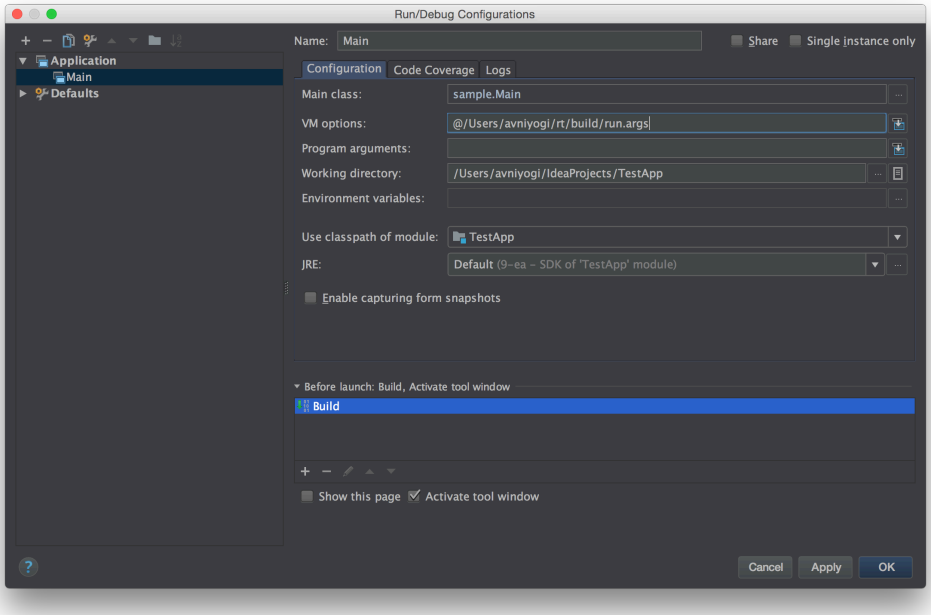




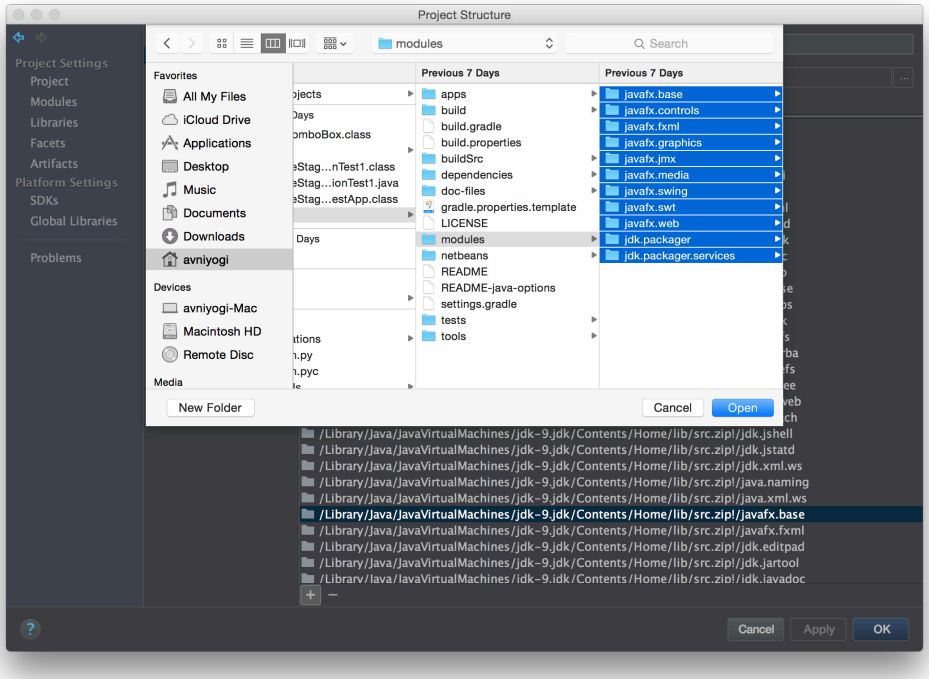
6. Verify that the correct SDK classpath is referred to.
7. Run the sample Main.java to verify that the Hello World is running. Quit "Hello World" java application.
8. Select **Run/Debug Configurations** titled as **Edit Configurations..** and add **@<full_path_of_run.args>** as shown

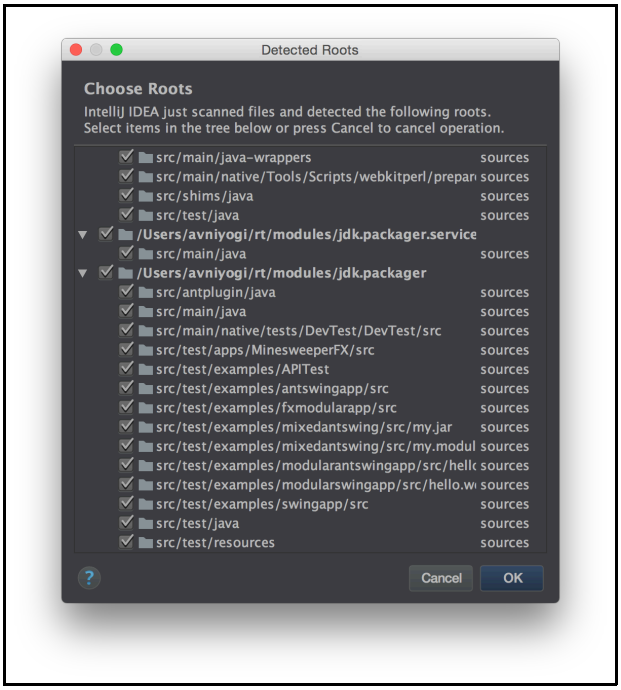
below

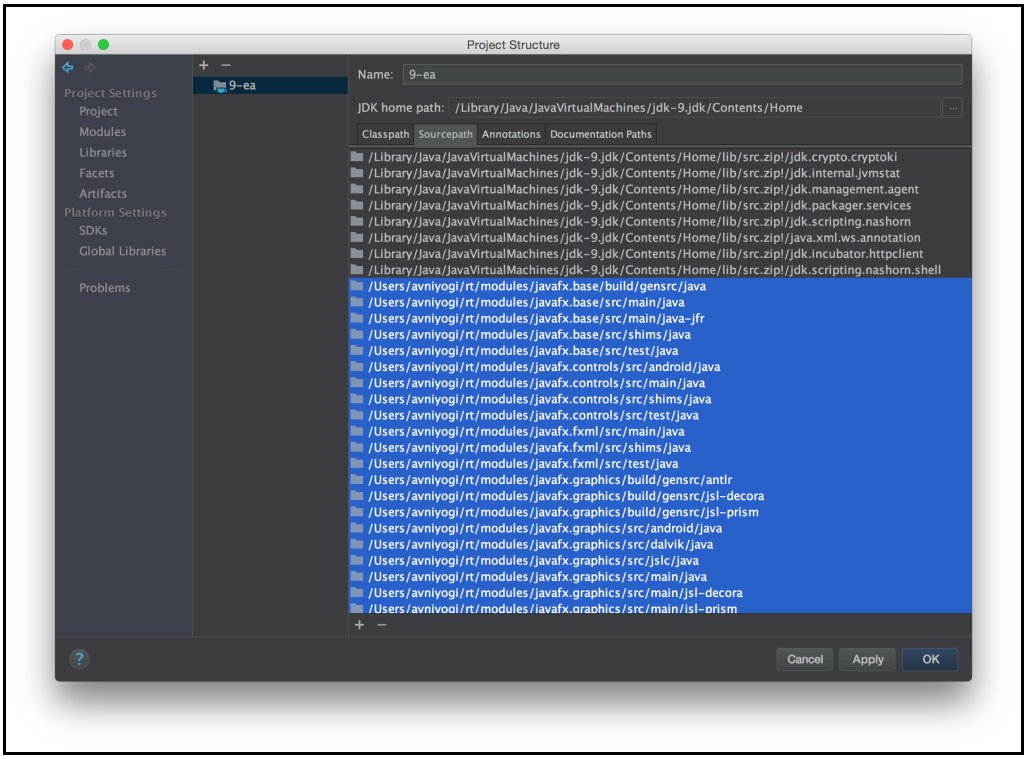

Run 'Main' ^R
Debug 'Main' ^D
Run 'Main' with Coverage
Run... ^⌥R
Debug... ^⌥D
Attach to Local Process...
Edit Configurations...
Import Test Results ▶
Stop ⌘F2
Show Running List
Reload Changed Classes
Restart Activity
Step Over F8
Force Step Over ⌥⇧F8
Step Into F7
Force Step Into ⌥⇧F7
Smart Step Into ⇧F7
Step Out ⇧F8
Run to Cursor ⌥F9
Force Run to Cursor ⌥⌘F9
Drop Frame
Pause Program
Resume Program ⌥⌘R
Evaluate Expression... ⌥F8
Quick Evaluate Expression ⌥⌘F8
Show Execution Point ⌥F10
Toggle Line Breakpoint ⌘F8
Toggle Method Breakpoint
Toggle Temporary Line Breakpoint ⌥⇧⌘F8
Toggle Breakpoint Enabled
View Breakpoints... ⇧⌘F8
Get thread dump



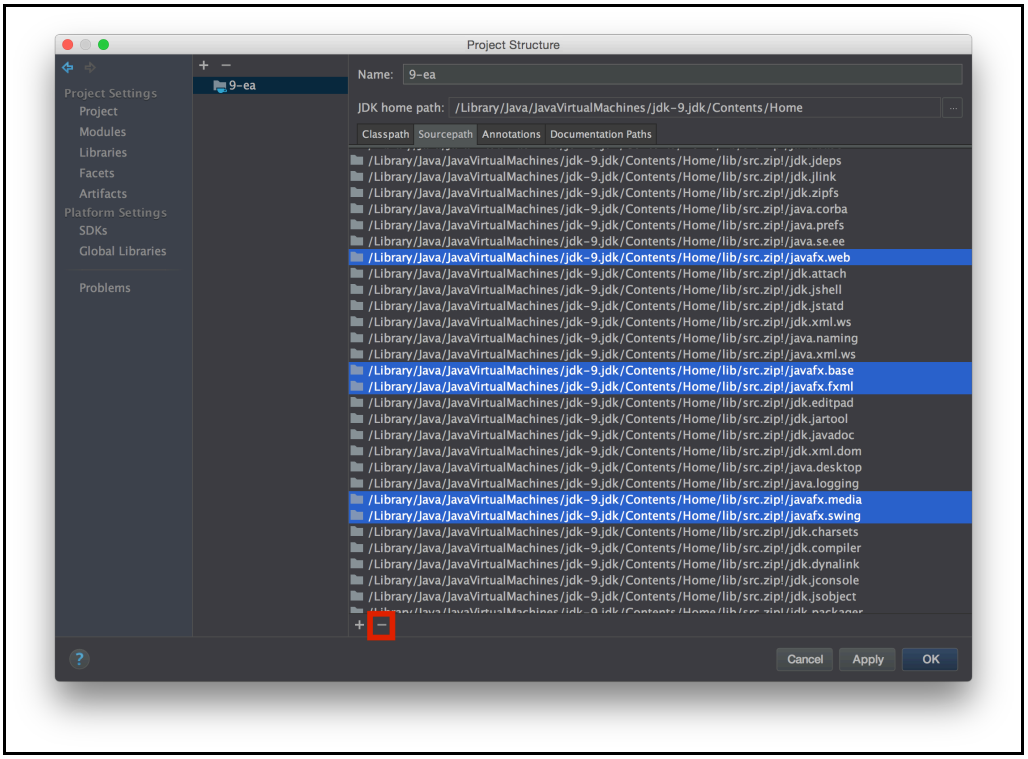
9. Add the modules of Java FX that are part of your local sandbox



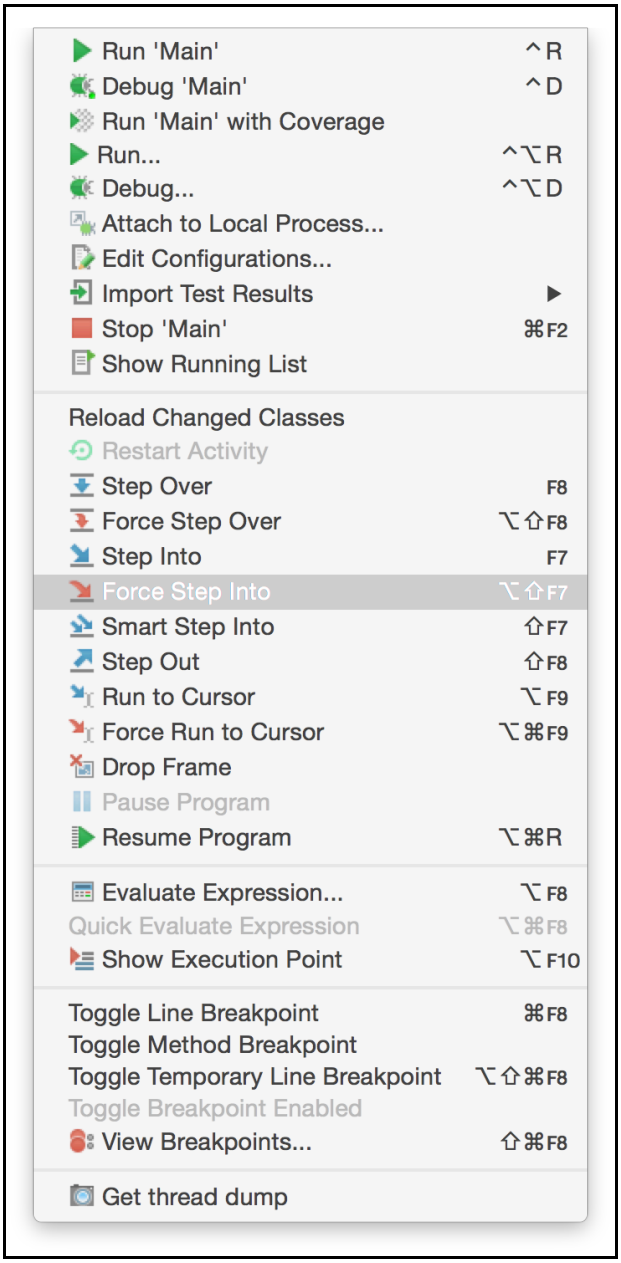


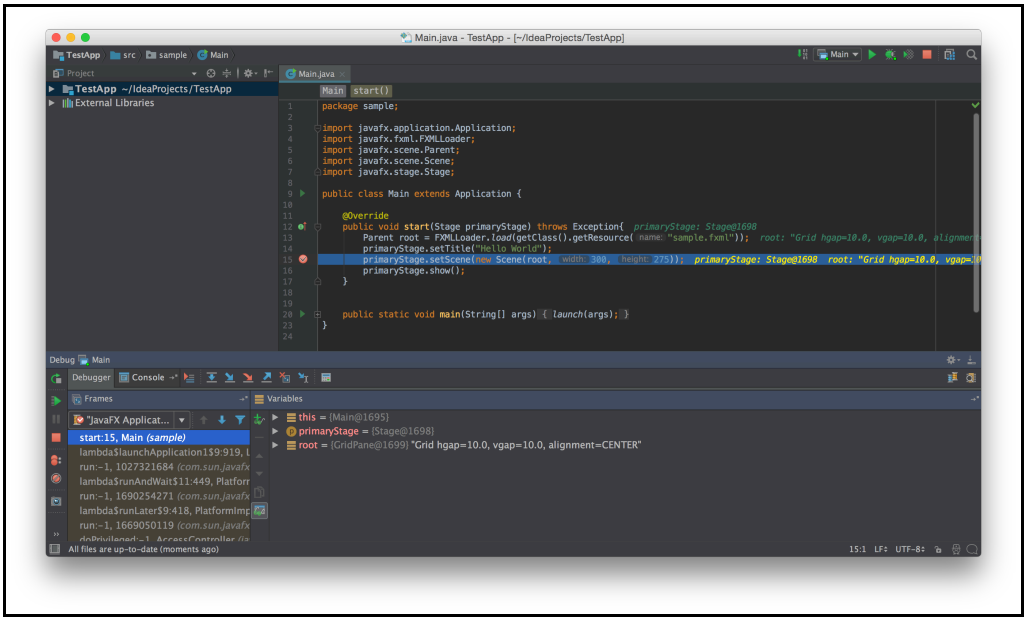


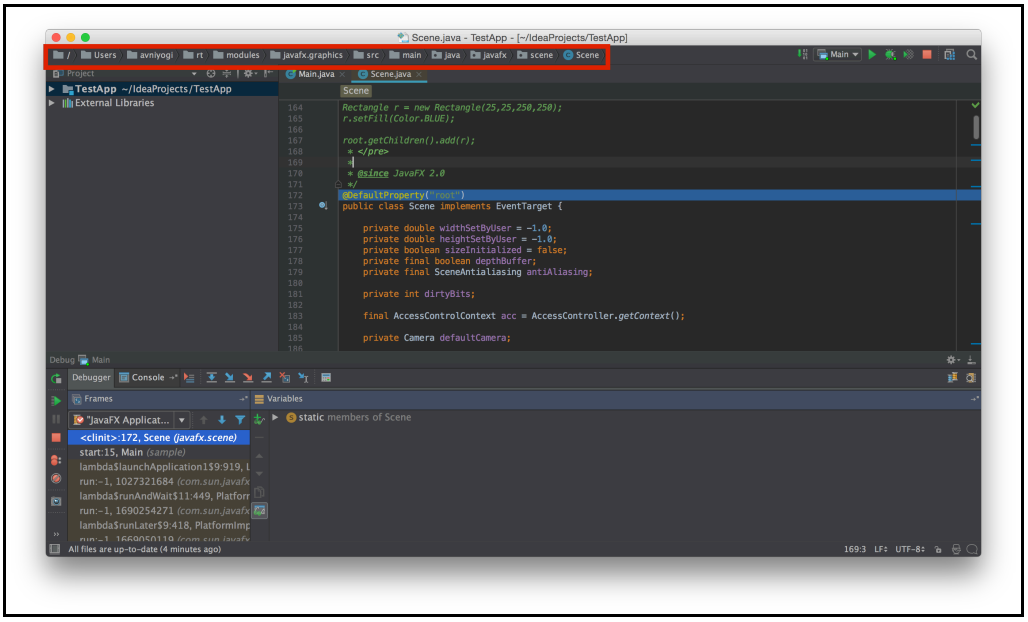
10. Remove ALL the references to Java FX modules which exist as part of JDK (only a few shown in screenshot as example) so that the Java FX used by IDE is the one in the local sandbox.



11. Debug the sample app using below menu items and check if it refers to local sandbox.


Run 'Main' ^R
Debug 'Main' ^D
Run 'Main' with Coverage
Run... ^⌥R
Debug... ^⌥D
Attach to Local Process...
Edit Configurations...
Import Test Results
Stop 'Main' ⌘F2
Show Running List
Reload Changed Classes
Restart Activity
Step Over F8
Force Step Over ⌥⇧F8
Step Into F7
Force Step Into ⌥⇧F7
Smart Step Into ⇧F7
Step Out ⇧F8
Run to Cursor ⌥F9
Force Run to Cursor ⌥⌘F9
Drop Frame
Pause Program
Resume Program ⌥⌘R
Evaluate Expression... ⌥F8
Quick Evaluate Expression ⌥⌘F8
Show Execution Point ⌥F10
Toggle Line Breakpoint ⌘F8
Toggle Method Breakpoint
Toggle Temporary Line Breakpoint ⌥⇧⌘F8
Toggle Breakpoint Enabled
View Breakpoints... ⇧⌘F8
Get thread dump


Main.java - TestApp - [~/IdeaProjects/TestApp]
TestApp ~/IdeaProjects/TestApp
External Libraries
Main.java
package sample;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;
public class Main extends Application {
@Override
public void start(Stage primaryStage) throws Exception{
Parent root = FXMLLoader.load(getClass().getResource("sample.fxml"));
primaryStage.setTitle("Hello World");
primaryStage.setScene(new Scene(root, 300, 275));
primaryStage.show();
}
public static void main(String[] args) { launch(args); }
}
Debug Main
Debugger Console
Frames Variables
start:15, Main (sample)
this = {Main@1695}
primaryStage = {Stage@1698}
root = {GridPane@1699} "Grid hgap=10.0, vgap=10.0, alignment=CENTER"
All files are up-to-date (moments ago)



12. Setup is done for Java FX project and the sandbox of Java FX.
13. Normal debug does not penetrate beyond peer files. Setup Native code as shown in **Steps for Native Code Project Setup** as per the required OS types.
    To debug native files, use respective IDEs
    a. Xcode for Mac OS X, refer to Step 5 of **Steps for Native Code Project Setup > Mac OS X using Xcode**
    b. Visual Studio for Windows, refer to Step 4 of **Steps for Native Code Project Setup > Windows > Visual Studio**

JavaFX open+closed Test Execution

1. Latest jdk
2. gradle 4.8
3. ant 1.8.2 (exactly this version)
4. Clone fx forest
    a. hg clone http://closedjdk.us.oracle.com/openjfx/jfx-dev
    b. cd jfx-dev
    c. bash ./hg-clone-forest.sh
5. Update cache
    a. cd jfx-dev
    b. ant update-cache
6. Build and run test
    a. cd jfx-dev/rt
    b. gradle sdk >& build.log
    c. gradle --continue --info -PFULL_TEST=true -PUSE_ROBOT=true test >& test.log

JavaFX Unit Test Execution

Under rt project there are several sub projects.
    a. Command to list the projects: gradle projects
    b. The command will list several sub projects like, **:apps**, **:base**, **:closedTests**, **:controls**, **:fxml**, **:fxpackager**, **:fxpackagerservices**, **:graphics**, **:jmx**, **:media**, **:swing**, **:swt**, **:systemTests**, **:web**
2. The module sub projects like **:base**, **:controls**, **:graphics** include unit tests as sub project named **:test**
3. So the command to execute Unit tests of **controls** from **rt** folder is
   >> **gradle :controls:test**
   >> Test results will be stored here: \rt \ modules \ javafx.controls \ build \ reports \ tests \ test \ index.html

4. Similarly the unit test of other sub projects can be executed. as **gradle :base:test** , **gradle :graphics:test**

System Tests:

To exclude tests : add in build.gradle -

```
exclude("test/com/**");
exclude("test/javafx/**");
exclude("test/memoryleak/**");
exclude("test/renderlock/**");
exclude("test/sandbox/**");
exclude("test/shutdowntest/**");
exclude("test/util/**")
```

Command to run test:

gradle -PFULL_TEST=true :systemTests:cleanTest :systemTests:test --tests <package.testClassName>

gradle -PFULL_TEST=true :systemTests:cleanTest :systemTests:test --tests test.javafx.scene.shape.meshmanagercacheleaktest.MeshManagerCacheLeakTest

export option: --add-exports javafx.graphics/com.sun.glass.ui=ALL-UNNAMED

JavaFX :apps Execution

1. There are several applications to show case the JavaFX functionalities.
2. These apps are very useful to perform Sanity testing, Fix verification.
3. Command to build the apps,
   >> **gradle :apps**
   Command to run **Hello** apps,
   a. HelloSanity - rt-folder > **java @<path_to>/run.args -cp apps/toys/Hello/dist/Hello.jar hello.HelloSanity**
   b. HelloTabPane - rt-folder > **java @<path_to>/run.args -cp apps/toys/Hello/dist/Hello.jar hello.HelloTabPane**
   c. HelloTableView - rt-folder > **java @<path_to>/run.args -cp apps/toys/Hello/dist/Hello.jar hello.HelloTableView**
   d. There are several Hello samples present here, rt / apps / toys / Hello / src / main / java / hello
   Sample Applications
   a. Ensemble8: sample application includes demo of controls, graphics, animation, 2D, 3D -- run below command from rt-folder
      > **java @<path_to>/run.args -jar apps/samples/Ensemble8/dist/Ensemble8.jar**
   b. Modena: sample application to showcase Modena CSS & custom CSS -- run below command from rt-folder
      > **java @<path_to>/run.args -jar apps/samples/Modena/dist/Modena.jar**
   c. ShapeT3D :-- run below command from rt-folder
      > **java @<path_to>/run.args -jar apps/toys/Shape3DToy/dist/ShapeT3D.jar**
   d. MandelbrotSet :-- run below command from rt-folder
      > **java @<path_to>/run.args -jar apps/samples/MandelbrotSet/dist/MandelbrotSet.jar**

JavaFX 8u-psu-dev commands

1. Setup
   a. gradle 4.8
   b. ant 1.8.2
   c. Latest JDK 8u (delete jfxrt.jar)
2. Clone repo

    a. hg clone http://closedjdk.us.oracle.com/openjfx/8u-psu-dev
    b. cd 8u-psu-dev
    c. sh hg-clone-forest.sh
    d. remove deploy repo, mv deploy _deploy
3. Build
    a. cd 8u-psu-dev/rt
    b. gradle
4. Build javadoc
    a. cd 8u-psu-dev/rt
    b. gradle javadoc -PBUILD_JAVADOC=true
5. Build apps
    a. cd 8u-psu-dev/rt
    b. gradle :apps
6. Run app
    a. cd 8u-psu-dev/rt
    b. java -Xbootclasspath/a:build/sdk/rt/lib/ext/jfxrt.jar -jar apps/samples/Ensemble8/dist/Ensemble8.jar

JavaFX Command line switches

1. JavaFX Properties and command-line switches
2. http://werner.yellowcouch.org/log/javafx-8-command-line-options/
3. -Dglass.win.uiScale=125%
4. -Dprism.order=sw
5. -Dquantum.debug=true

CSR - Imp Links

1. Main page : https://wiki.openjdk.java.net/display/csr/Main
2. FAQs - https://wiki.openjdk.java.net/display/csr/CSR+FAQs
3. Fields of CSR request: https://wiki.openjdk.java.net/display/csr/Fields+of+a+CSR+Request
4. Kinds of compatibility - https://wiki.openjdk.java.net/display/csr/Kinds+of+Compatibility

Crucible review

You should create a patch with the following command (only then the patch can get anchored to a particular repo)

hg diff --config diff.git=false <file1> <file2>....> jdk.patch

and then upload as a "pre-commit" patch into to crucible https://java.se.oracle.com/code/cru/.

Ubuntu Related Stuff

1. Install GNOME window manager, `sudo apt-get install ubuntu-gnome-desktop`

Useful Information

1. JavaFX Command line options: http://werner.yellowcouch.org/log/javafx-8-command-line-options/
2. Linux printer configuration: https://confluence.oraclecorp.com/confluence/pages/viewpage.action?pageId=812831794

The developers guide says the contributed-by line is just in case the author does not have commit rights (nor an author ID I suppose).

No labels