-

SHARE Session #9777: WebSphere and Rational Developer Hands-on-Labs – Building Java application on System z with RDz

Lab exercise (estimate duration)

Introduction

This hands-on-lab is designed for those interested in learning how to develop and run Java applications on z/OS. It shows you how to use a single integrated development environment (IDE): IBM Rational Developer for System z (RDz) V8 to write, run, and debug a Java application on your workstation, then have it run and debugged on z/OS by leveraging the IBM Developer Toolkits for z/OS.

About this lab

There are two exercises in this lab.

The first exercise guides you through the process of creating a simple, platform-independent "Hello world" Java application that displays the current date and time in a specific output format. You will execute this program two ways: 1) using the Eclipse Launch Configuration, and 2) using a JCL.

The second exercise guides you through the development of a Java application with affinity to z/OS. This Java application reads the name of a PDS, and returns the list of PDS members as the program output. You will learn how you can execute the program using the Eclipse Launch Configuration.

Pre-requisites

- Basic understanding of Java programming
- Familiarity with basic z/OS concepts

Your User id for this lab is: SHARA__

Your password is: *firstpw*

Exercise 1 - Your first Java application on z/OS

In this exercise, you will learn how to use RDz to write, run, and debug a Java application on Windows. Once that is completed, you will then export the Java application as a remote Jar file to z/OS, and then you will run and debug this again on z/OS

Overview of tasks in this exercise

1. Creating the Java application
2. Running and debugging the application on Windows
3. Exporting the application to z/OS
4. Running the application on z/OS using a Host Java Application Launch Configuration
5. Running the application on z/OS as a batch step from JCL

1. Creating the Java application

a. Launch RDz by double clicking on the desktop icon "**RDz 8.0.1**". From the pull-down menu, select the workspace: "C:\workspace\RDzJava", and click **OK**.

b. Once the application is loaded, you will see, in the title area, "z/OS Projects – IBM Rational Developer for System z with Java". This is the **z/OS Projects perspective**[1]**.**

c. The perspective is further customized for this lab with the addition of the **Package Explorer** view and the **Problems** view.

d. You will see a project "PDSJavaProject" on the **Package Explorer**. This will be used for Exercise 2. For now, just ignore it and the errors you see in the **Problems view**.

e. Create a Java project.

   i. From the menu, click **File > New > Project…**

   ii. Select **Java project** from the list of wizards. Click **Next**.

   iii. Name it *HelloWorldProject*. Accept all other default values. Click **Finish**. The project will appear on the **Package Explorer**.

f. Create a Java package within the project

   i. Select your project, right-click and select **New > Other…** Expand the Java folder and select **Package**. Click **Next**.

   ii. Name the package *com.sharann*, where *sharann* is your assigned userid for this lab. Click **Finish**.

g. Import the Java source program

   i. Select the package *com.sharann*, right-click and select **Import…**

   ii. Expand the **General** folder, and then select **File System**. Click **Next**.

   iii. On "From directory", use the **Browse…** button to select the folder on your desktop: **SHARELabs\RDzJava\Exercise1**. Click **OK**, and you're returned to the wizard.

   iv. On the wizard, you should see a list of available files to be imported, select **HelloWorld.java**. Click **Finish**, and the file is imported.

   v. You will notice that there are a few error markers on your HelloWorld project. This is because the imported java code is currently pointing to another package that doesn't exist. To fix it, open the HelloWorld.java program, click on the marker beside "package" on the first line, to see available "**Quick-fixes"**

   vi. Of the options provided to fix this, select "**Change package declaration to "com.sharann**" to match the name of the package you created.

   vii. Press *CTRL+S* to save the changes. You should see the error markers gone.

[1] In Eclipse, views (or windows) are organized and laid out as a unit known as *Perspective*. A Perspective is a collection of related views that helps users accomplish specific goals. Some examples of perspective are Java, Debug, z/OS Projects, etc.

2. Running and debugging on Windows

a. Run your Java application on Windows

   i. Because this application does not have any z/OS resource dependency, you can run it locally on Windows. To run, highlight your program *HelloWorld.java*, right-click and select **Run As > Java application**. The program output is displayed in the **Console** view at the lower section of your screen.

b. Debug your Java application on Windows

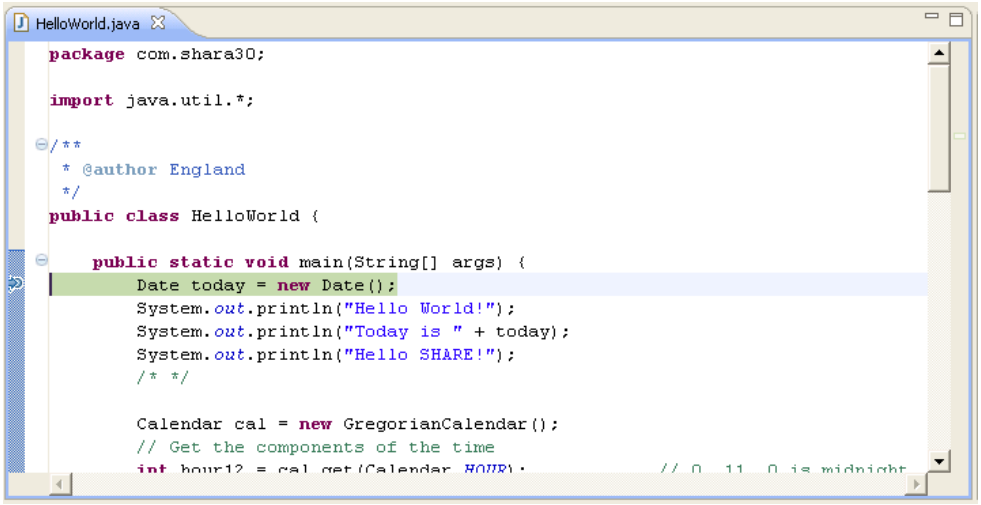
   i. Before we begin debugging, open the *Helloworld.java* program in the editor, and add a couple of breakpoints to the program by double-clicking on the grey-column on the left hand side of the editor. For example, try to set a breakpoint on line 11 (Date today =…)

   ii. To debug the application, go back to the project, highlight *HelloWorld.java*, right-click and select **Debug As > Java Application**.

   iii. Select "Yes" if asked whether you want to switch to the **Debug perspective**.

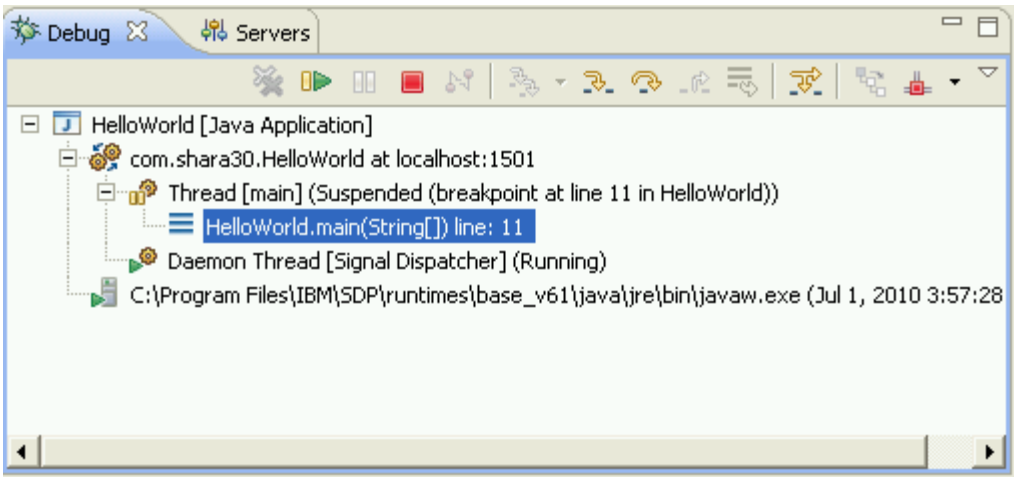
   iv. The **Debug perspective** opens, and you have access to full debugging capabilities such as stepping through code, using breakpoints, and altering variable values.

   v. Since you had previously introduced a breakpoint on line 11, the Debug engine is suspended at this line, waiting for your action.



   vi. On the **Debug** view, there are a number of buttons in the toolbar area. You can select the green **Resume** button to resume debugging. Or you can select one of the Step buttons (**Step into** or **Step over**) to trace the program operations step by step.

   vii. Hover your mouse over the different buttons to see the name of the buttons. Click on the **Step Over** button to advance your debug.



viii. As you continue your debug session, notice how other views, such as the **Variables** view, refresh their contents to provide context sensitive information.

When you reach the end of the Debug session (you will see that all threads are terminated on the **Debug view**), click on "z/OS Projects" button on the top right corner of your perspective to return to the **z/OS Projects** perspective.

---

3. Exporting the application to z/OS (and then we'll run it on z!)

a. Authenticate to the SHARE system.

i. On the **Remote Systems** view, right-click on the connection "mvs1.centers.ihost.com", and select **Connect…**

ii. On the authentication dialog, enter your TSO ID for this lab (It should be: shara*nn*, where shara*nn* is your assigned userid.)[2] Your password is "firstpw"

iii. Under the system name, you should see nodes such as z/OS UNIX Files, z/OS UNIX Shells, MVS Files, etc. Expand on these nodes to browse the file assets under the MVS file system and the USS file system. (Note: please do not delete any files unless explicitly instructed to).

b. Export your Java project.

To run your application on z/OS, you need to first export the project as a Jar file to the Unix System Services (USS, i.e. HFS - the Hierarchical File System).

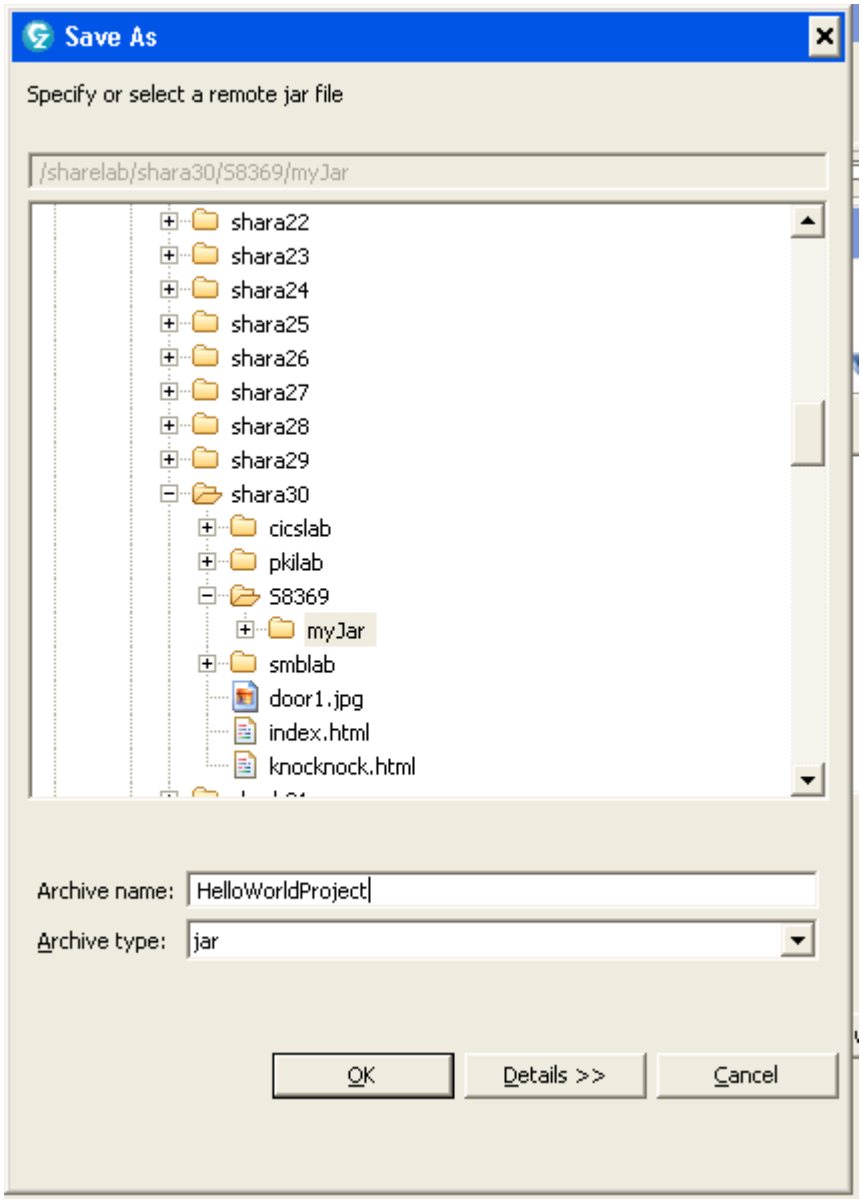
i. On **Package Explorer** view, select your Java project, right-click and select **Export…**

ii. Expand the **Remote Systems** folder, select **Remote Jar file.** Click **Next**.

iii. Under "Select the export destination", click **Browse** to choose a location and a name for the export .jar file. Browse to your connection:

[2] To save time, a connection to the z/OS system has been pre-configured for you in this lab.

mvs1.centers.ihost.com, expand the root node, and then select the directory */sharelab/sharann/S8369/myJar*. On the same dialog, enter a name for the file: *HelloWorldProject*, and specify the archive type to be "jar".



Click **OK** to exit the pop-up dialog. Click **Next** to advance to the next page.

iv. Ensure that the two checkboxes for exporting class files with compile errors and compile warnings are unchecked. Click **Next**.

v. On this last page of the wizard, click the **Browse…** button for Main Class and select **HelloWorld** from the pop-up.

vi. Click **Finish.**

4. Running the application on z/OS using a **Host Java Application** launch configuration

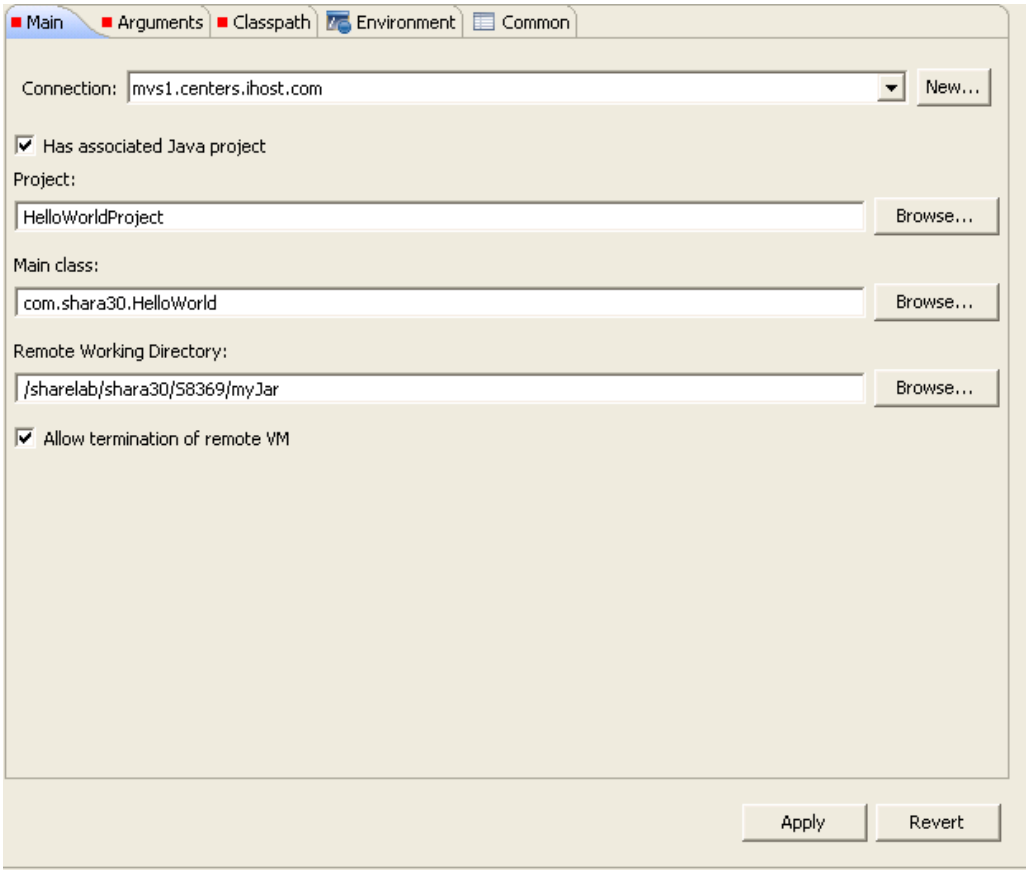
a. First, create the launch configuration[3] file for running the program on z/OS On the menu. To do so, go to the menu, click **Run > Run Configurations…**

b. From the dialog's navigation panel, select **Host Java Application**. Right-click and select **New**. Give this launch configuration a name, such as *RunHelloWorldProject*.

c. Fill in the details on the **Main**, **Classpath**, and **Environment** tabs using the information below.

i. On the **Main** tab, enter the following (Suggestion: Use the Browse… button whenever possible to avoid typos)

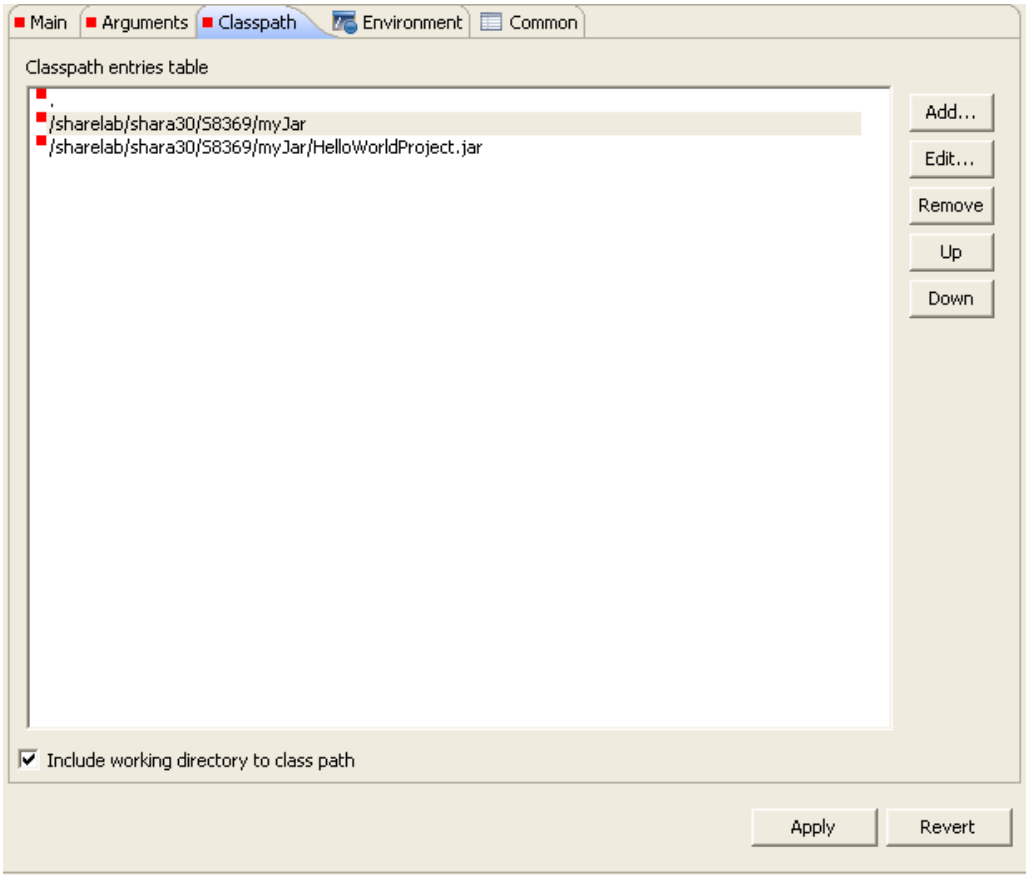
- Connection: *mvs1.centers.ihost.com*
- Select the checkbox "*Has associated Java project*".
- Project: *HelloWorldProject*
- Main class: *com.sharann.HelloWorld*
- Remote working directory: */sharelab/sharann/S8369/myJar*



[3] "A launch configuration is a description of how to launch a program. The program itself may be a Java program, another Eclipse instance in the form of a runtime workbench, a C program, or something else. Launch configurations are manifested in the Eclipse UI through **Run > Run Configurations...**." (http://wiki.eclipse.org/FAQ_What_is_a_launch_configuration%3F)

ii. On the **Classpath** tab, enter the following for your path:

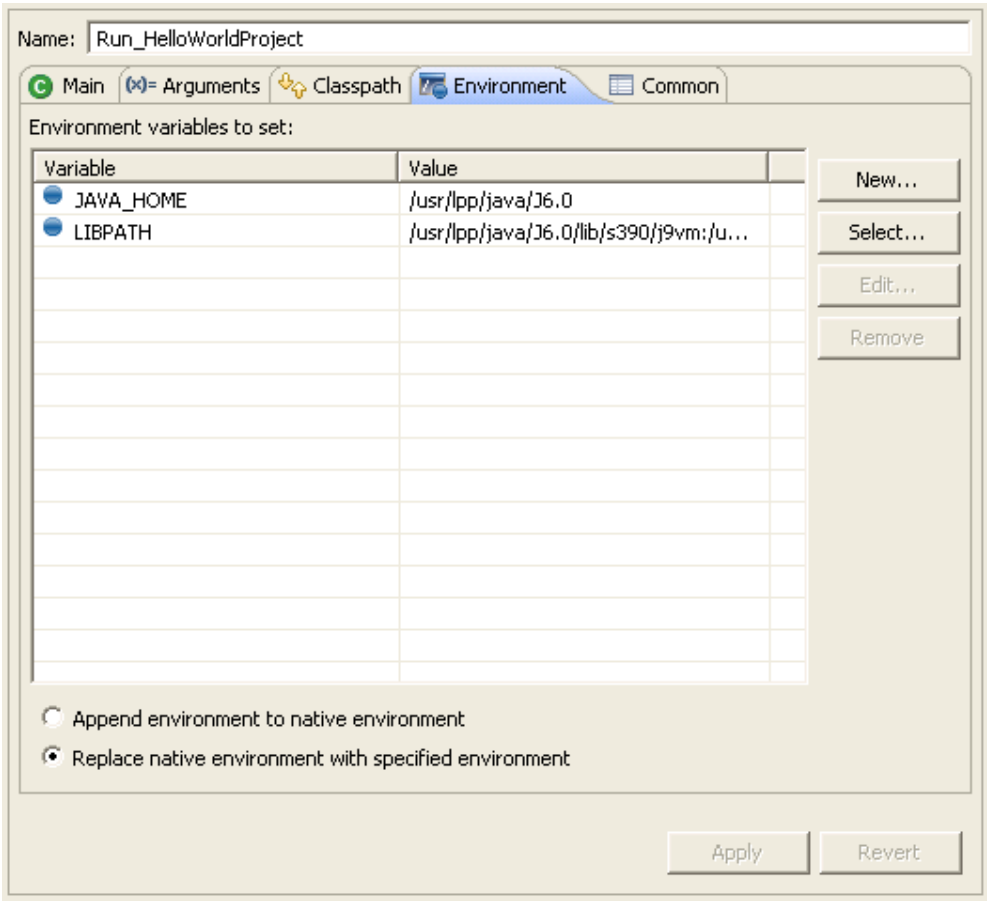
- */sharelab/sharann/S8369/myJar/HelloWorldProject.jar*



iii. On the **Environment** tab, enter your PATH environment variable information, including your paths to Java home and the library paths.

- Click the **Select…** button to select the checkbox for JAVA_HOME and the checkbox for LIBPATH. Click **OK**.
- Also on the Environment tab, select the radio button: **Replace native environment with specified environment**

d. Click **Apply** to save the changes to your launch configuration file.



e. Click **Run** to run the Java application on z/OS. Again, the program output is displayed in the **Console** view. Congratulations! You have just finished running your Java application on z/OS!

5. Running the application as a batch step from JCL

   Besides using the host launch configuration, you can also run Java applications on z/OS using the capabilities provided by the JZOS batch toolkit. This allows you to invoke any Java programs from a z/OS batch environment.

a. Return to the **z/OS Projects** perspective.

b. On the **Remote Systems** view, expand the node **MVS Files,** then expand the "My Data Sets" filter. Locate the data set SHARAnn.S8369.JCL. Expand it and then highlight the member HELLOW.jcl, right mouse click to **copy**, and then **paste** it into the data set SHARAnn.S8369.JZOS.JCL

c. Open the newly copied HELLOW.JCL.

d. Follow the instruction below to customize it:

   i. Verify that

   PROCLIB JCLLIB ORDER = KIRK.JZOS.SAMPJCL

   ii. **Update**

   JAVACLS='com.sharann.HelloWorld'

   iii. Verify that

   e*xport JZOS_HOME= /usr/lpp/java/J6.0/lib/ext*

   export JAVA_HOME=/usr/lpp/java/J6.0

   iv. **Update the CLASSPATH to point to the remote jar file, e.g.:**

   CLASSPATH=/sharelab/sharann/S8369/myJar/HelloWorldProject.jar

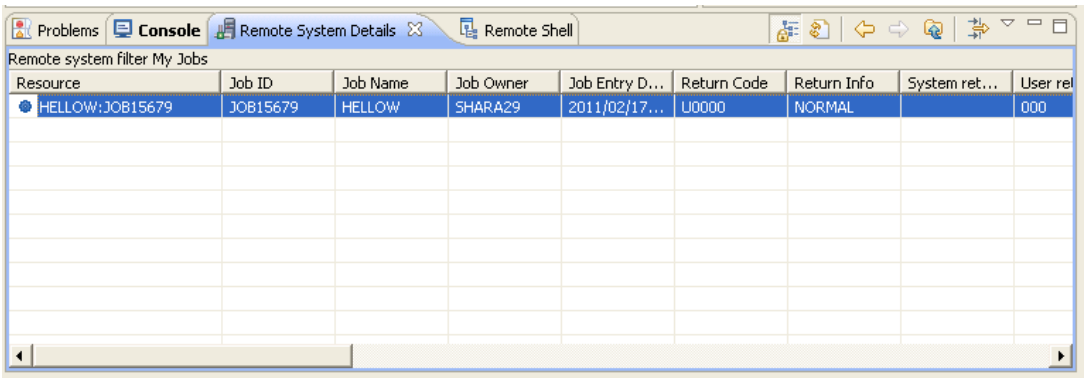
   Save the changes.

e. Next, submit this JCL to run the batch job. To submit the JCL. Do one of the following:

   i. On the editor, press ESC to set focus on the editor's command line. Type "sub" and press Return. Make note of your JOBID.

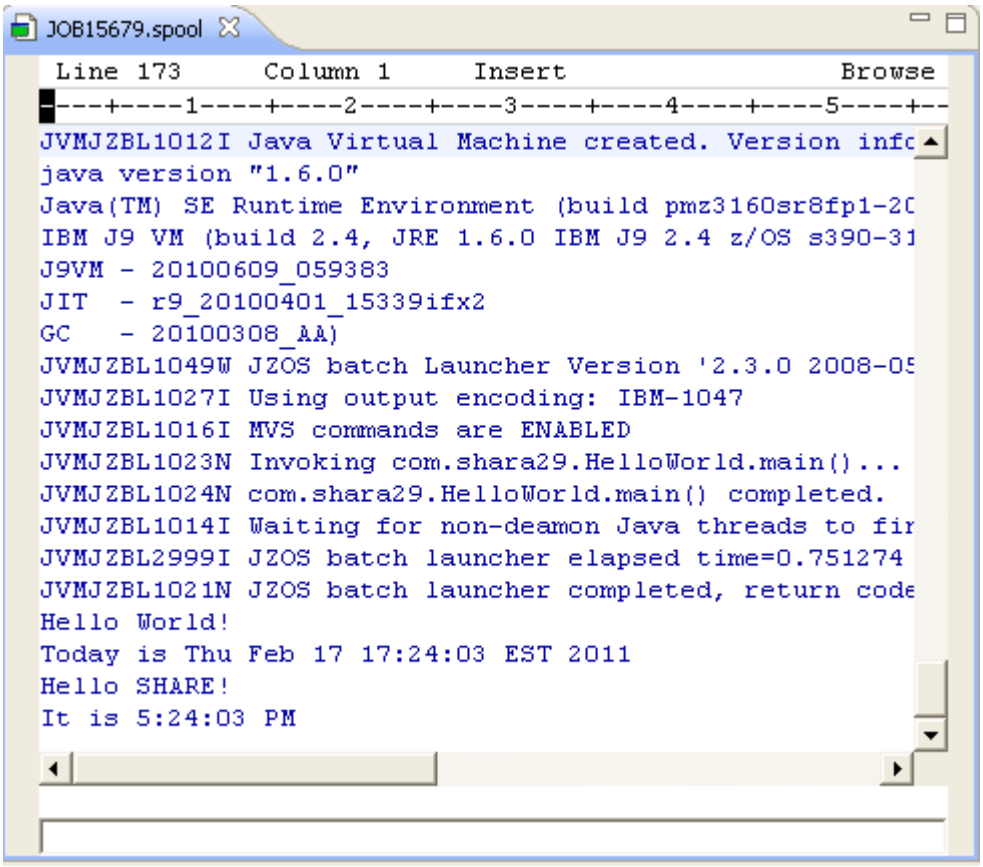
   ii. Alternatively, on the **Remote Systems** view, select HELLOW.JCL, right-click and then select **Submit**. Make note of your JOBID.

f. To view the job output, locate the **JES** node on the **Remote Systems** view, expand it to see the filter "**My Jobs**". Select it, right-click and then select **Show in Table**.



g. The **Remote System Details** view should be visible. Select the job output from the queue, right-click to select **Open**. If you do not see your job, press the **Refresh** button that is located on the toolbar area of this view.

h. Your job should have a return code of 0, and if all goes well, you will find the output of the program displayed at the end of job output.



Congratulations! You have just completed the task of customizing a batch step from a JCL to execute Java on z/OS!

Exercise 2 – Writing and running Java applications with an affinity to z/OS

Overview of tasks in this exercise:
1. Open the included Java project *PDSJavaProject*
2. Update the build path to include the external Jar file for compilation
3. Export the project as a remote jar file to the HFS
4. Running the application on z/OS using a Host Java Application Launch Configuration

1. On the **Package Explorer** view, you will find the Java project *PDSJavaProject*. Expand the project and package until you see the Java program. Open it in the editor and familiarize yourself with the code. You will see a number of error markers in the code. This is because the project has a few errors, including some unresolved references to the class com.ibm.jzos.PdsDirectory located in the ibmjzos.jar[4] file.

---

2. Make the following changes:

a. In the project, use the Refactor action to rename the package name to **com.shara*nn***. Next, open the program, and update the package name at the top of the file.

b. Modify the following line

   myPDS = "//'SHARAnn.S1112.COBOL'" with your assigned userid.

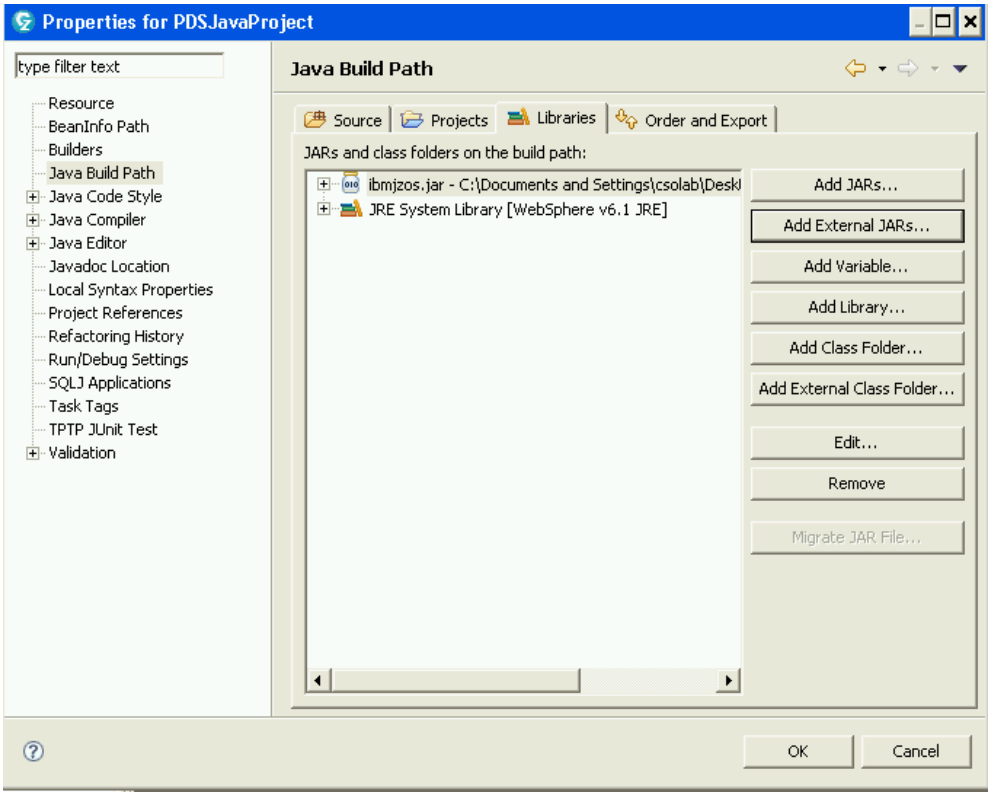
   Save your changes to the program.

c. Update the Java project properties to include the ibmjzos.jar file for compilation.

   i. Go back to the **Package Explorer** view, select *PDSJavaProject*, right-click and select **Properties**.

[4] The ibmjzos.jar file is part of the JZOS Batch Toolkit

ii. Select **Java Build Path** from the left hand side, then select the **Libraries** tab, and click on the **Add External JARs…** button. Browse to the location **SHARELabs\RDzJava\Exercise2** to select *ibmjzos.jar*[5]. Click **Open**.



iii. Go to the **Order and Export** tab, and select the checkbox for *ibmjzos.jar*.

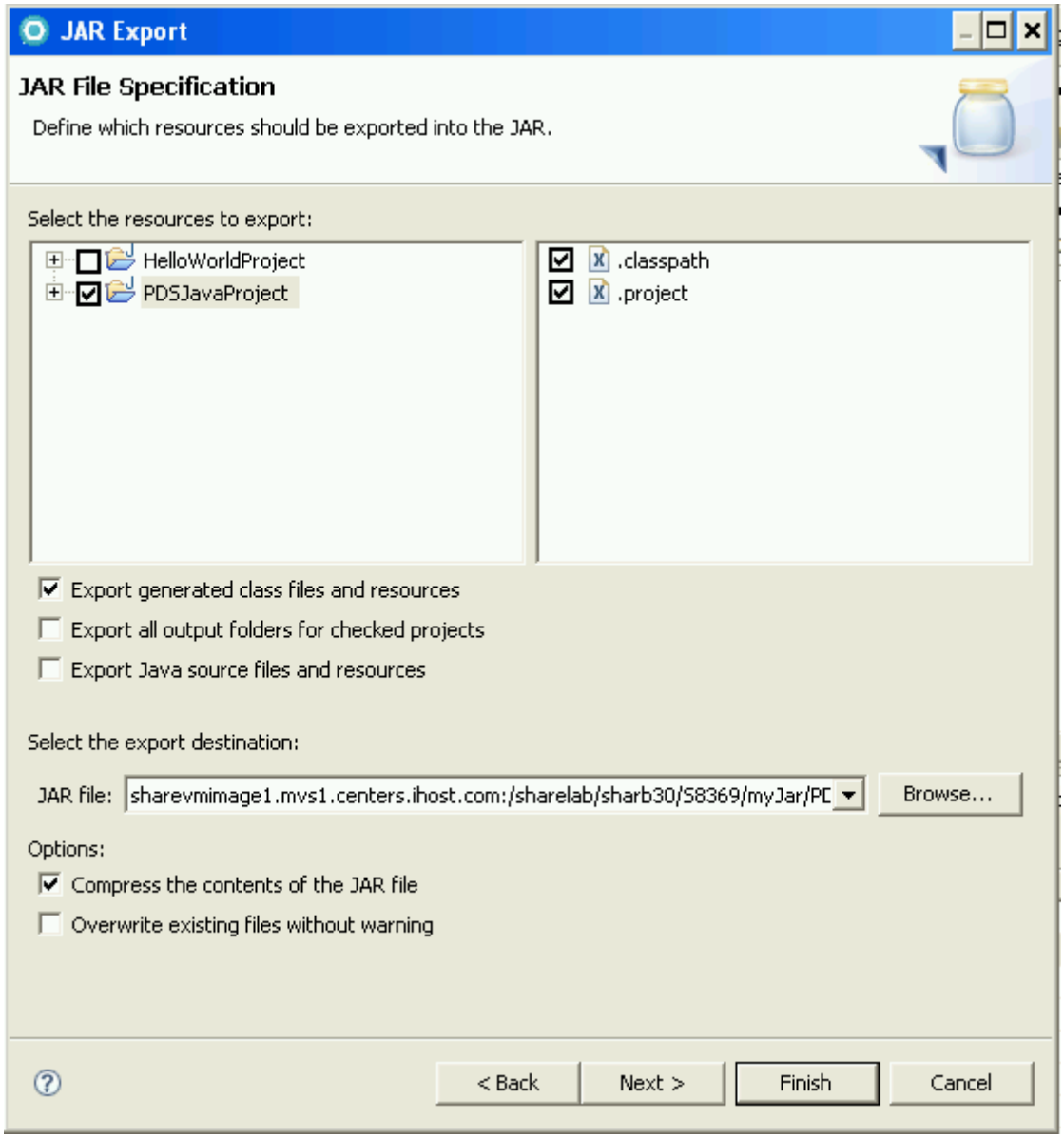
iv. Click **OK** to exit the **Properties** dialog. Expand the *PDSJavaProject* again, and you shall see the ibmjzos.jar file added to your project (under the node Referenced Libraries)

v. All the unresolved reference errors should be gone now. On the **Problems** view, verify that you have no error shown.

[5] To save time, we have copied this file from the system to your desktop folder "SHARELABs\RDzJava\Exercise2". You can easily transfer files between host and workstation using the **Remote Systems** view since, in addition to browsing your host files, you can also browse your workstation folders and files using the **Local Files** node on this view.

3. Export the project as a remote jar file to HFS

a. Export the project as a Remote Jar file name: *PDSJavaProject.jar*, to */Sharelab/sharann/S8369/myJar*. Follow the same steps as you did in the first exercise.



b. Verify that the export is successful by expanding the directory /ShareLab/sharann/S8369/myJar/ to view the PDSJavaProject.jar file. (Note, if you don't see the jar file, try to do a **Refresh** from the context menu on the "myJar" folder)

4. Run the application on z/OS using the Launch Configuration

   Repeat the steps you performed in Exercise 1 by creating a new host Java application launch configuration.

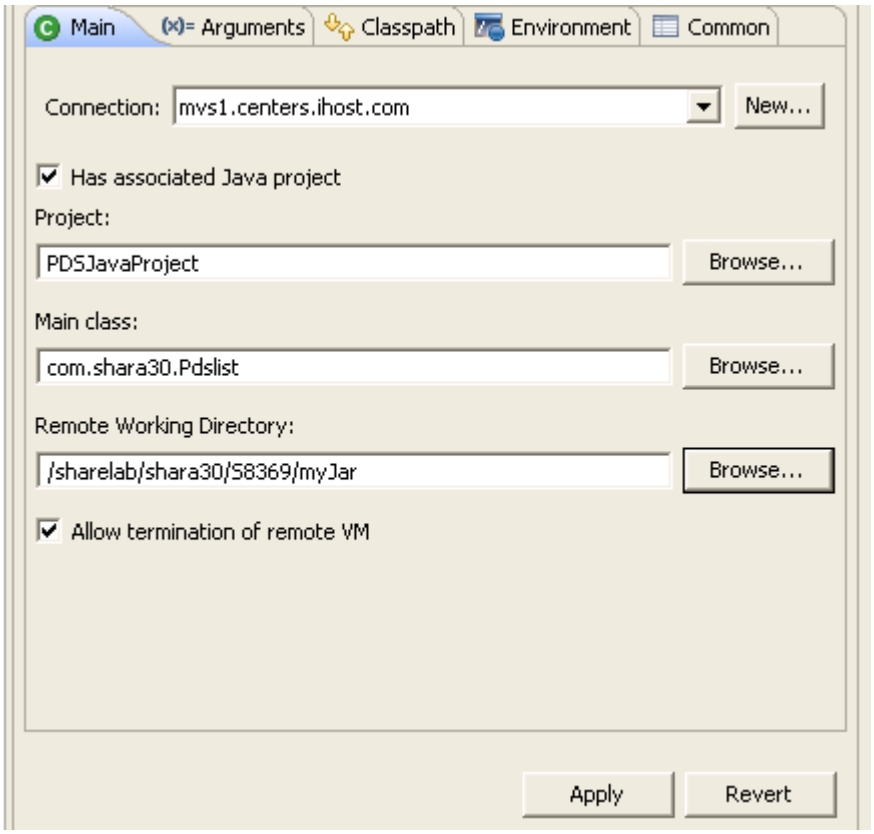
a. Go to the menu, click **Run > Run Configurations…**

b. From the dialog's navigation panel, locate the previous configuration "RunHelloWorldProject" you created, right-click on it, select **Duplicate**. Give this new launch configuration a name, such as *RunPDSJavaProject*.

c. Fill in the details on the **Main**, **Classpath**, and **Environment** tabs using the information below.

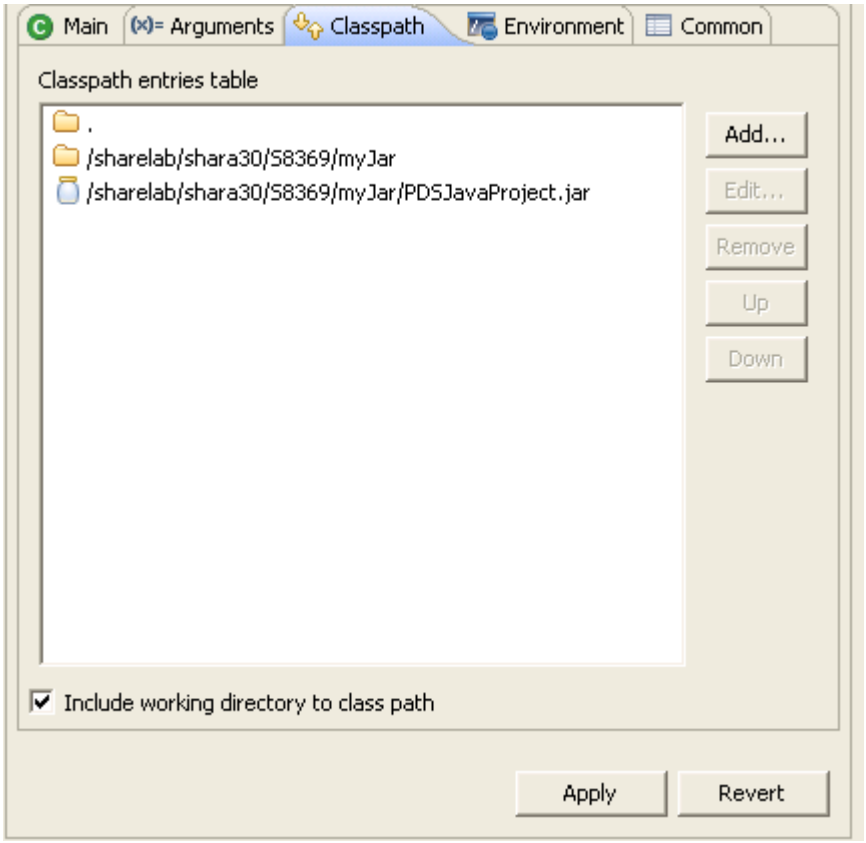
   i. On the **Main** tab, enter the following (Suggestion: Use the **Browse…** button whenever possible to avoid typos)

   - Connection: *mvs1.centers.ihost.com*
   - Select the checkbox "*Has associated Java project*".
   - Project: *PDSJavaProject*
   - Main class: *com.sharann.Pdslist*
   - Remote working directory: */sharelab/sharann/S8369/myJar*



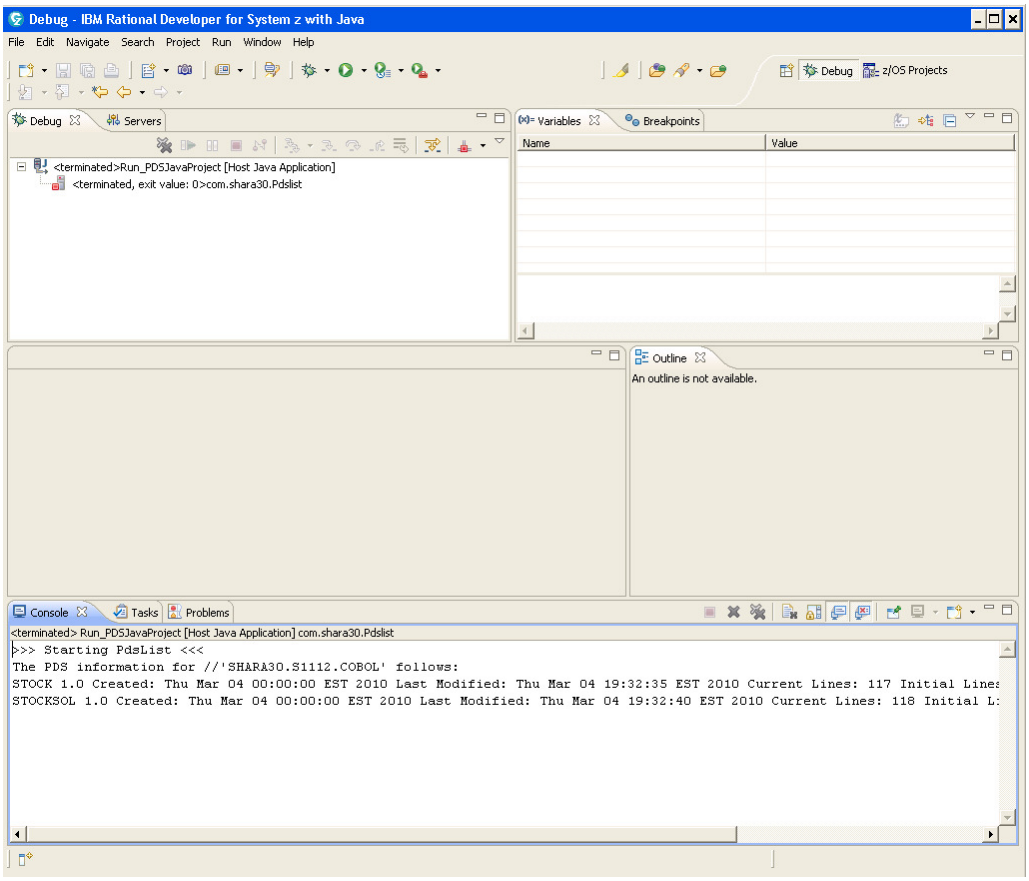
ii. On the **Classpath** tab, update it to include the location of the jar file:

- */Sharelab/sharann/S8369/myJar/PDSJavaProject.jar*



iii. The Environment variables are the same as previous, so no changes need to be made.

iv. With all the above changes made, press **Run** to execute. You should see the output in the **Console view**:



--- End of Lab ---

Appendix

About the technology

Java is a programming language that continues to gain popularity. It is supported on a variety of computing platforms including Windows®, Sun Solaris, and z/OS. Yes, the mainframe.

Eclipse, an open source project, provides a sophisticated, extendable, integrated development environment (IDE) for writing Java applications. With the exception of its team support, which enables groups of users to work with a code repository such as CVS, Eclipse operates as a single user IDE in which all work is performed in a workspace on your local hard drive.

Many commercial products have been built to extend the base Eclipse capabilities. ***IBM Rational Developer for System z*** provides access to the mainframe and lets you manipulate artifacts that reside on the mainframe with a modern, intuitive, and familiar user interface. It "includes capabilities that can help make traditional mainframe development, Web development, and integrated service-oriented architecture (SOA)-based composite development fast and efficient. COBOL, PL/I, C, C++, High-Level Assembler, and Java™ developer communities can also be more productive when they take advantage of these functions. IBM Rational Developer for System z integrates with and extends the IBM Rational Software Delivery Platform (SDP).[6]

IBM JZOS Batch Toolkit for z/OS® SDK is "a set of tools that addresses many of the functional and environmental shortcomings in current Java™ batch capabilities on z/OS. It includes a native launcher for running Java applications directly as batch jobs or started tasks, and a set of Java methods that make access to traditional z/OS data and key system services directly available from Java applications. Additional system services include console communication, multiline WTO (write to operator), and return code passing capability. In addition, JZOS provides facilities for flexible configuration of the run-time environment, and it allows intermediate data to be seen via z/OS System Display and Search Facility (SDSF). Java applications can be fully integrated as job steps in order to augment existing batch applications."[7]

Reference

Title	Location
Writing Java applications on System z	http://www.ibm.com/developerworks/websphere/library/techarticles/0703_england/0703_england.html
IBM Rational Developer for System z	http://www-306.ibm.com/software/awdtools/rdz/
IBM JZOS Batch Toolkit for z/OS® SDK	http://www-03.ibm.com/servers/eserver/zseries/software/java/products/jzos/overview.html

[6] http://www-306.ibm.com/software/awdtools/rdz/
[7] http://www.alphaworks.ibm.com/tech/zosjavabatchtk