



Java and Eclipse for Beginner

Part II of III

Session: 1182

Presented by

Theresa Tai
IBM System z New Technology Center
Poughkeepsie, NY
ttai@us.ibm.com



maxim
ove agility save time
enges colleagues



Agenda

- ❖ Exceptions and Exception Handling
 - Exercise 1, Exception Handling and I/O
- ❖ Array List
 - Exercise 2, extending the CommandLine exercise
- ❖ Supplementary Materials
 - JNI (Java Native Interface)
 - Calling Java from C
 - Calling C from Java
 - Remote Debugging
 - The Java (Class) API
 - The Collection API

Exceptions and Exception Handling



- ❖ Exceptions in Java are any abnormal, unexpected events or extraordinary conditions that may occur at runtime
- ❖ On such conditions java **throws** an exception object
- ❖ Exception handling involves **catching** the exception and taking the necessary actions

Exception Handling Syntax

❖ Throwing an exception

```
throw (exception);
```

❖ and catching it

```
try {  
    // code that can throw Exceptions  
}  
catch (...) { }  
catch (...) { }  
finally {  
    // tidy up code...  
}
```



Exception Types 1

- ❖ Checked Exceptions are Exceptions which must be handled programmatically.
- ❖ Methods declare the exceptions that they can throw and the calling method must catch this exception.

```
public void foo() throws <Exception Type> {  
}
```



Exception Types 2

- ❖ Unchecked Exceptions are Exceptions which the compiler doesn't insist are handled.
- ❖ Unchecked exceptions don't need to be declared in a method's throw clause.
- ❖ Unchecked exceptions represent runtime problems which code cannot reasonably be expected to recover from.
- ❖ Examples of unchecked exceptions are:
 - ArithmeticException
 - ArrayStoreException
 - IndexOutOfBoundsException
 - NegativeArraySizeException
 - NullPointerException

Catching Exceptions



SHARE

A screenshot of the Eclipse IDE interface. The main editor window displays the code for `mathExceptionHandled.java`. The code defines a package `mathException` and a class `mathExceptionHandled` with a `main` method and a `divide` method. The `divide` method uses a `try-catch` block to handle a `除以零` (division by zero) exception. The Package Explorer on the left shows the project structure. The Outline view on the right shows the class and method structure. The Console window at the bottom shows the output of the program, indicating a terminated Java application and the exception message.

```
package mathException;

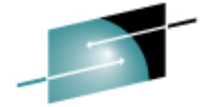
public class mathExceptionHandled {

    public static void main(String[] args) {
        int result = divide(4,0);
    }

    public static int divide(int numerator, int denominator) {
        int result = 0;
        try {
            result = numerator/denominator;
        }
        catch(Exception e){
            System.out.println("Exception : " + e.getMessage());
        }
        return result;
    }
}
```

<terminated> mathExceptionHandled [Java Application] C:\Program Files\IBM\Java50\jre\bin\javaw.exe (31 Jul 2008 01:13:25)
Exception : / by zero

... and not Catching Exceptions



The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations and development tools. The left sidebar shows the Package Explorer with a project named 'SHARE' containing a 'src' folder with files 'ExceptionExample', 'mathException', 'mathException.java', and 'mathExceptionHandled.java'. The 'mathException.java' file is selected. The main editor window displays the code for 'mathException.java':

```
package mathException;

public class mathException {

    public static void main(String[] args) {
        int result = devide(4,0);
    }

    public static int devide(int numerator, int denominator) {
        int result = numerator/denominator;
        return result;
    }
}
```

The right sidebar shows the Outline view with a tree structure of the project, including 'mathException' and 'main(String[])'. The bottom panel shows the Console view with the following output:

```
<terminated> mathExceptionHandled [Java Application] C:\Program Files\IBM\Java50\jre\bin\javaw.exe (31 Jul 2008 01:23:06)
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at mathException.mathException.devide(mathException.java:10)
    at mathException.mathException.main(mathException.java:6)
```

The status bar at the bottom indicates 'Writable', 'Smart Insert', and '6 : 31'.



Exercise 1

- ❖ The FilePrinter.java program reads the files passed as arguments
- ❖ The file handling methods throw checked Exceptions which will need to be handled
- ❖ Use a finally block to clean up any resources that the application may have



ArrayList

❖ Example of creating an ArrayList

```
List myList = new ArrayList();
```

❖ Example List methods

<http://java.sun.com/j2se/1.5.0/docs/api/java/util/ArrayList.html>

add(Object element)

get(int index)

contains(Object element)

indexOf(Object element)

size()



ArrayList example

```
import java.util.*;

public class ArrayListExample1 {

    public static void main(String[] args) {

        List theChildren = new ArrayList();

        theChildren.add("Jon");
        theChildren.add("Jane");

        System.out.println("number of children: " + theChildren.size());
        System.out.println("First item: " + theChildren.get(0));
        System.out.println("Second item: " + theChildren.get(1));
    }
}
```



Problems with collections so far

- ❖ In Java 1.4.2 (and before) List entries are of class Object which means that there is no compile time check for what is added to a Collection
- ❖ Type checking was the responsibility of the programmer.
- ❖ Not very nice ...
 1. Prone to mistakes
 2. Casting produces ugly code

Problems with collections so far

- ❖ Collections as described so far worked without compile warnings Java 1.4.2 but not for Java 5
- ❖ Against `theArguments.add(i);` in Java 5 (and later releases) is the warning:
Type safety: The method `add(object)` belongs to the raw type `List`. References to Generic type `List<E>` should be parameterised.
- You'll probably encounter these warnings in the next Exercise



Java 5 and Generics

❖ Java 5 introduces Generics

<http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf>

// Before Java 5 ArrayLists entries are Objects

```
List theArguments = new ArrayList();
```

// In Java 5 it is possible to define the class of entries eg String

```
List <String> theArguments = new ArrayList <String> ();
```



Exercise 2

- ❖ Modify the CommandLine program to store the arguments in a ArrayList
- ❖ Query this array list to see if it contains a specific value



Supplementary Materials

Java Native Interface (JNI)

<http://java.sun.com/j2se/1.5.0/docs/guide/jni/>

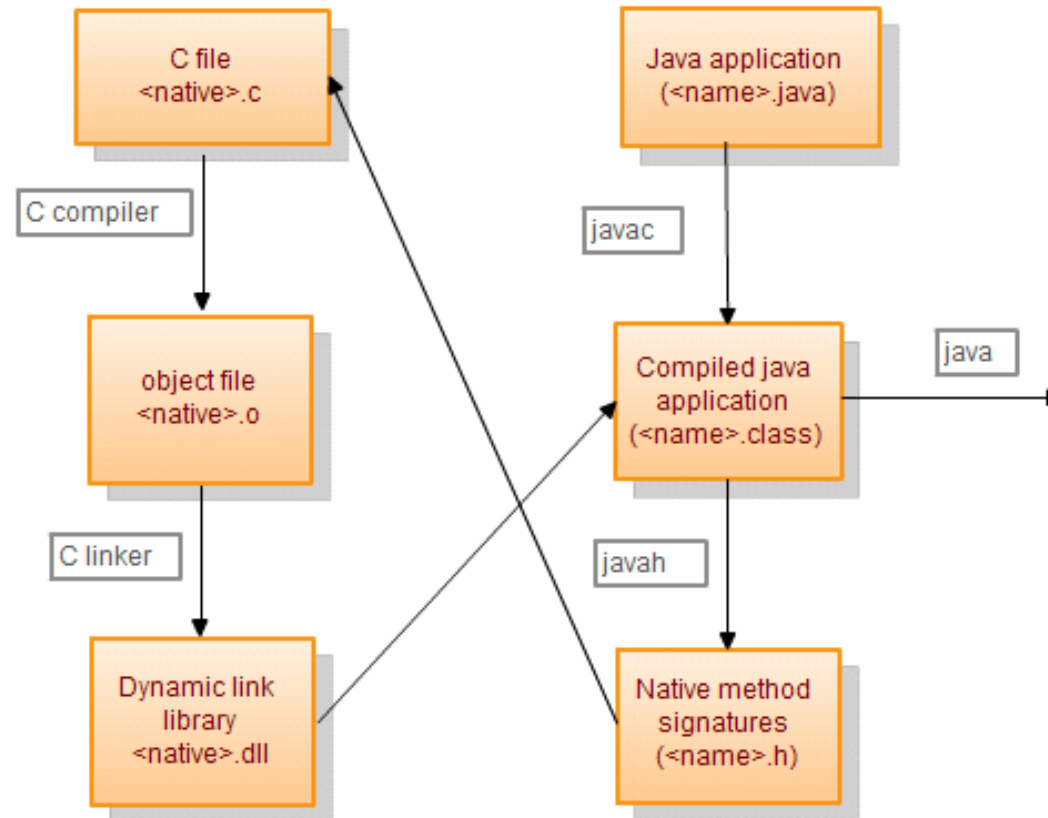
- Call Java code from native code
- Call native code (C, Cobol, Fortran, assembler, ...) from Java. This may be needed where :
 - The Java class library doesn't support platform-dependent features
 - The code required already exists and was written in a different language
 - Requirements demand a bespoke implementation in assembly for example

Call Java code from native code

- JNI code from the Java launcher

```
...  
/* Create Java VM */  
res = JNI_CreateJavaVM(&jvm, (void**)&env, &vm_args);  
...  
cls = (*env)->FindClass(env, classname);  
...  
mid = (*env)->GetStaticMethodID(env, cls, methodname, methodsig);  
...  
(*env)->CallStaticVoidMethod(env, cls, mid);  
...  
/* Destroy JVM */  
(*jvm)->DestroyJavaVM(jvm);
```

Call native code from java code





The Java Application

```
/*
 * =====
 * Licensed Materials - Property of IBM
 * "Restricted Materials of IBM"
 *
 * IBM SDK, Java(tm) 2 Technology Edition, v1.4.2
 * (C) Copyright IBM Corp. 1998, 2002. All Rights Reserved
 * =====
 */

public class HelloSanJose
{
    public static native void SayHello();

    public static void main (String[] args) {
        System.loadLibrary("HelloSanJose");
        SayHello();
    }
}
```

The <native>.h file



```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class HelloSanJose */

#ifndef _Included_HelloSanJose
#define _Included_HelloSanJose
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:      HelloSanJose
 * Method:     SayHello
 * Signature:  ()V
 */
JNIEXPORT void JNICALL Java_HelloSanJose_SayHello
    (JNIEnv *, jclass);

#ifdef __cplusplus
}
#endif
#endif
```

The native method implementation

```
#include <stdio.h>
#include <stdlib.h>
#include <jni.h>
#include <string.h>
#include <stdlib.h>

JNIEXPORT void JNICALL Java_HelloSanJose_SayHello
(JNIEnv *env, jclass obj, jstring jstr) {

    printf("\n\nHi San Jose\n\n");

}
```


The build script

```
rm *.h *.o *.so *.x *.class

javac HelloSanJose.java

javah -jni HelloSanJose

c++ -c -o HelloSanJose.o -W c,xplink,dll -W c,exportall -W "c,langlvl(extended)"
-W"c,float(ieee)" -I/u/java9/J1.4/include -I. HelloSanJose.c

c++ -o libHelloSanJose.so -W l,xplink,dll HelloSanJose.o
```



Run the application ...

```
/u/flavell/Tests/JNISanJose:>java -classpath ./SanJ HelloSanJose
```

```
Hi San Jose
```

```
/u/flavell/Tests/JNISanJose:>
```



Remote debugging

- In this exercise an application running on a z/OS box will be debugged from Eclipse running on a windows box
- The application consists of a loop which allocates a chunk of storage and writes a message to stdout, to confirm that the application is running on z/OS, on each iteration
- The exercise consists of 2 parts:
 - starting the application on a remote machine (z/OS)
 - debugging the application from Eclipse



The Application

```
package eclipseLab;

import eclipseLab.RemoteMemGrab;

public class RemoteRunIt {
    public static void main(String[]
args) {
        int i=0;
        System.out.println("Exercise 3");
        while (i++ < 3) {
            RemoteMemGrab aGrab =
new
RemoteMemGrab();
        }
    }
}
```

```
package eclipseLab;

public class RemoteMemGrab {
    public RemoteMemGrab()
    {
        System.out.println("About to
allocate a      StringBuffer");
        StringBuffer buf= new
StringBuffer(10000);
    }
}
```

On the remote machine

- ❖ The files and directory structure on z/OS

```
total 56
drwx-----  3 FLAVELL  JAVA      8192 Aug  6 02:02 .
drwxrwx--- 25 FLAVELL  JAVA      8192 Aug  6 01:57 ..
drwx-----  2 FLAVELL  JAVA      8192 Aug  6 02:51 eclipseLab
-rwx-----  1 FLAVELL  JAVA       163 Aug  6 02:56 runMeRemote

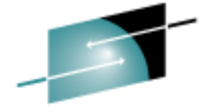
./eclipseLab:
total 64
drwx-----  2 FLAVELL  JAVA      8192 Aug  6 02:51 .
drwx-----  3 FLAVELL  JAVA      8192 Aug  6 02:02 ..
-rw-----  1 FLAVELL  JAVA       574 Aug  6 03:02 RemoteMemGrab.class
-rw-r----- 1 FLAVELL  JAVA       201 Aug  6 02:29 RemoteMemGrab.java
-rw-----  1 FLAVELL  JAVA       686 Aug  6 03:02 RemoteRunIt.class
-rw-r----- 1 FLAVELL  JAVA       246 Aug  6 03:09 RemoteRunIt.java
/u/flavell/Tests/JAVATests/RemoteTest:>
```

- ❖ run runMeRemote which starts RemoteRunIt suspended waiting on debug instructions on port 8001

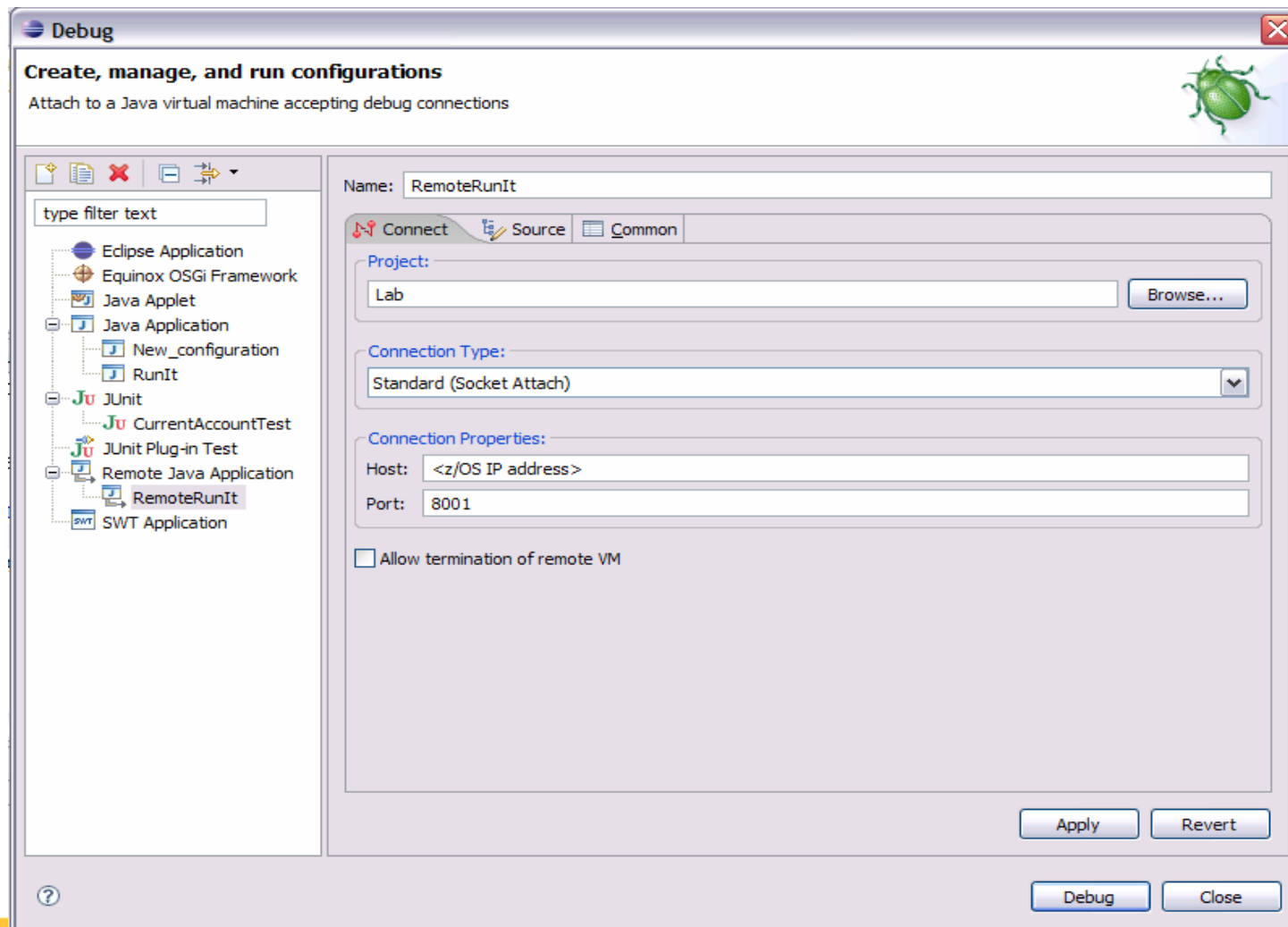
```
javac -g eclipseLab/Remote*.java
java -Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,address=8001,suspend=y,server=y eclipseLab.RemoteRunIt
```

In Eclipse

- ❖ Select **Run > Debug** , enter the following settings and select **Debug**

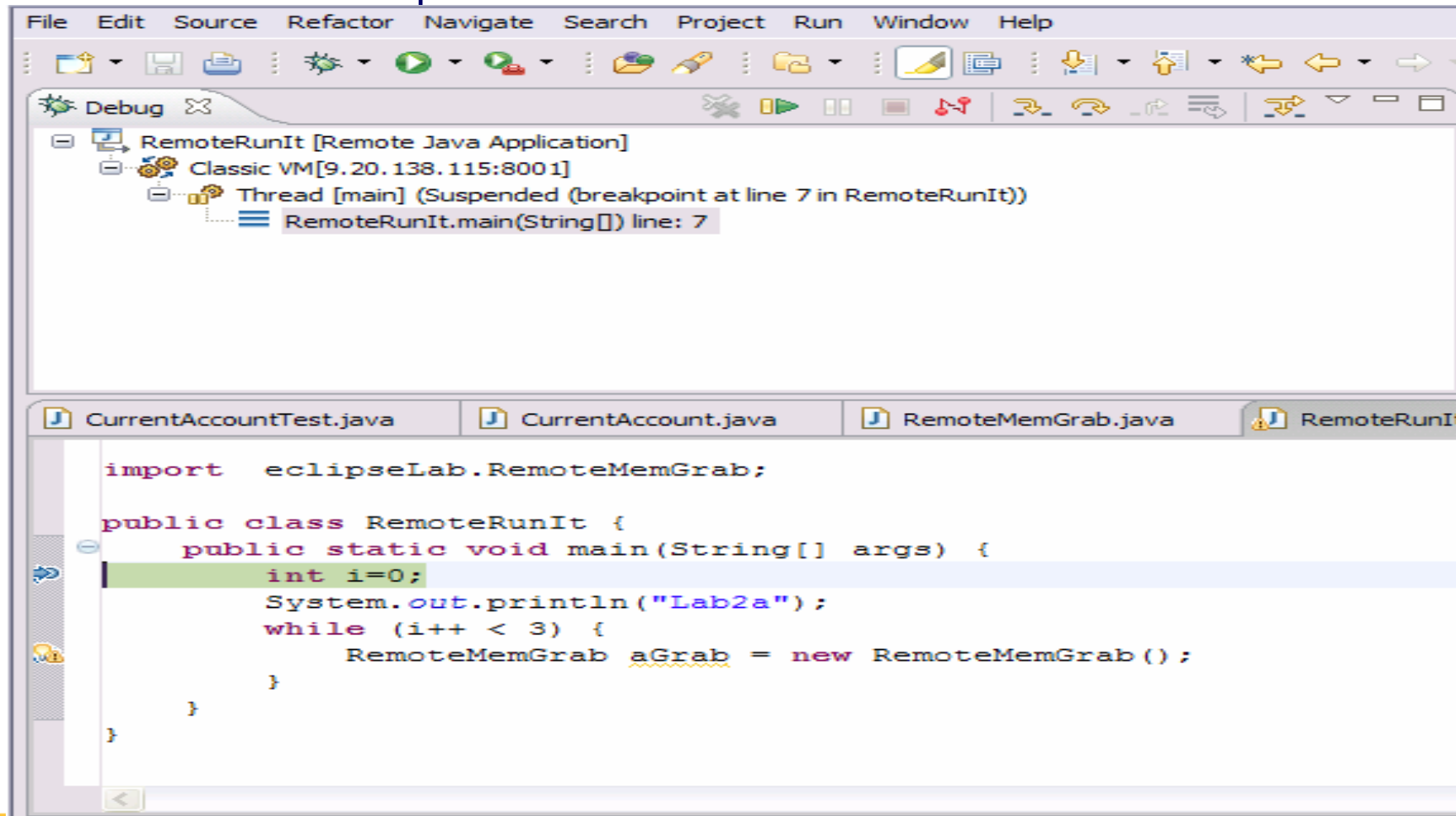


SHARE
Technology - Connections - Results



In Eclipse

- ❖ A breakpoint was set at the start of the main method and execution is suspended here.



An in flight snapshot



Debug Console:

- RemoteRunIt [Remote Java Application]
- Classic VM[9.20.138.115:8001]
- Thread [main] (Suspended)
- RemoteRunIt.main(String[]) line: 9

Variables:

Name	Value
args	String[0] (id=11)
i	2 [0x2]

Code Editor:

```
package eclipseLab;

import eclipseLab.RemoteMemGrab;

public class RemoteRunIt {
    public static void main(String[] args) {
        int i=0;
        System.out.println("Lab2a");
        while (i++ < 3) {
            RemoteMemGrab aGrab = new RemoteMemGrab();
        }
    }
}
```

Outline:

- eclipseLab
- import declarations
- RemoteRunIt
- main(String[])

Console:

No consoles to display at this time.

Console Window:

```
eclipseLab runMeRemote
/u/flavell/Tests/JAVATests/RemoteTest:>runMeRemote
JVMHP030: Unable to switch to IFA processor - issue "extattr +a libhpi.so"
JVMHP030: Unable to switch to IFA processor - issue "extattr +a libhpi.so"
Lab2a
About to allocate a StringBuffer
About to allocate a StringBuffer
```

Click to add



Java (Class) API

- ❖ As well as the Virtual Machine the Java sdk contains a class library

<http://java.sun.com/j2se/1.5.0/docs/api/index.html>

<http://java.sun.com/j2se/1.4.2/docs/api/index.html>

<http://java.sun.com/javase/6/docs/api/index.html>

- ❖ Java applications are written to this API
- ❖ Exercise 1 uses the I/O API available in the **java.io** package.

<http://java.sun.com/j2se/1.4.2/docs/api/java/io/package-tree.html>



Java (Class) API

The main Java packages are:

java.awt Contains all of the classes for creating user interfaces and for painting graphics and images.

java.io Provides for system input and output through data streams, serialization and the file system.

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

java.net Provides the classes for implementing networking applications.

java.security Provides the classes and interfaces for the security framework.

java.util Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes.



The Collections API

<http://java.sun.com/j2se/1.5.0/docs/api/index.html>

- A Collection is an object that groups multiple elements into a single unit.
- Collections are used to store, retrieve, manipulate, and communicate aggregate data.

		Implementations				
		Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Interfaces	Set	HashSet		TreeSet		LinkedHashSet
	List		ArrayList		LinkedList	
	Map	HashMap		TreeMap		LinkedHashMap

Thank You!



The Future Runs on System z