CICS API for Java

CICS and Java Wildfire Workshop

Steve Fowlkes, fowlkes@us.ibm.com
Leigh Compton, lcompton@us.ibm.com

Abstract

There’s never been a better time to take advantage of Java technology in CICS.

This topic discusses the CICS APIs provided for Java. The JCICS and JCICSX classes can be used in any CICS/Java program, including OSGi-based and Liberty Profile environments.

While every effort is made to ensure the information presented is correct, always consult the IBM documentation for CICS.

Agenda

- How CICS supports Java applications
  - Types of applications
  - JVM server environments
- Program invocation
  - Entry points
- Passing data to and from Java programs
  - Commareas, Channels & Containers
  - Code page considerations
- Handling exceptions
- Accesssing CICS resources
  - JCICS classes
  - JCICSX classes

Sources of information

- CICS product documentation:
  - V5.5 -- https://www.ibm.com/docs/en/cics-ts/5.5
  - V5.6 -- https://www.ibm.com/docs/en/cics-ts/5.6
  - V6.1 – https://www.ibm.com/docs/en/cics-ts/6.1
- Developing Java applications for CICS
  - ?topic=applications-developing-java
  - https://www.ibm.com/docs/en/cics-ts/5.6?topic=applications-developing-java
- JCICS Javadoc
  - ?topic=development-jcics-javadoc
  - https://www.ibm.com/docs/en/cics-ts/5.6?topic=development-jcics-javadoc
- Java development with JCICSX (V5.6)
  - https://www.ibm.com/docs/en/cics-ts/5.6?topic=applications-java-development-using-jcicsx
- JCICSX Javadoc
  - https://www.ibm.com/docs/en/cics-ts/5.6?topic=development-jcicsx-javadoc

JVM servers in CICS

- CICS uses a JVMSERVER resource to define the properties of a Java Virtual Machine
- Types of JVM servers:
  - OSGi
  - Liberty Profile
  - Classpath
    - AXIS2 capable
    - Security Token Server (STS) capable
    - Batch capable
    - Mobile capable

Where can you use Java in CICS?

- CICS style programs (OSGi environment)
  - Initial program of a transaction
  - Started by a user at terminal or EXEC CICS START
  - Program named in EXEC CICS HANDLE ABEND command
  - Target of an EXEC CICS LINK
    - Target of EXCI call or DPL including ECI
    - Target of a CICS pipeline Web Service request
    - Pipeline Handler
- JEE style programs (Liberty Profile environment)
  - Web applications
    - Servlets, Java Server Pages
  - JAX-RS applications (REST services)
  - JAX-WS applications (SOAP services)
  - Target of an EXEC CICS LINK
    - Target of EXCI call or DPL including ECI
    - Target of a CICS pipeline Web Service request
    - Pipeline Handler

CICS also uses Java

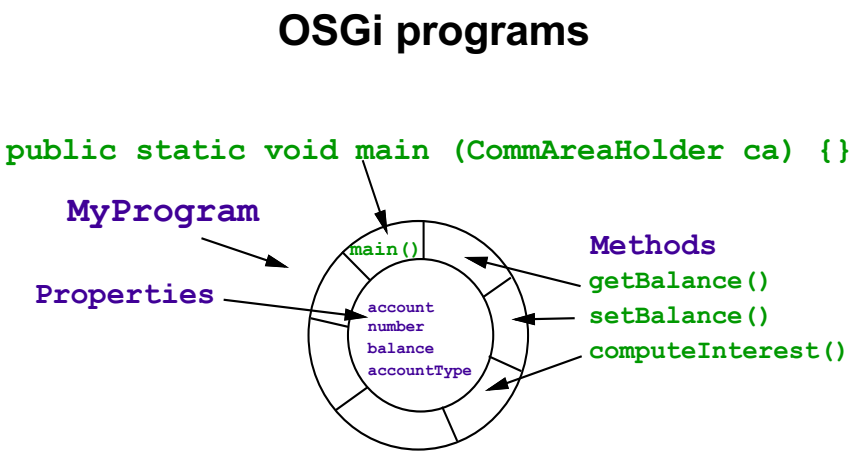
- AXIS2 JVM server
  - SAML
  - JSON transformation
  - Java batch FeaturePack
- Classpath JVM server

Application issues

- How does your program get started?
- How does CICS pass data to your program?
- How does your program interact with the CICS environment?
- How does your program access CICS resources?
- How do you handle error conditions?

Java program entry

- CICS OSGi Java Program
  - public static void main(CommAreaHolder ca)
  - public static void main(String[ ] args)
  - COMMAREA is accessible in a CommAreaHolder
- Web services (Axis2 or Liberty)
  - At the requested method or annotation
- Servlet (Liberty)
  - At the doGet(), doPost(), doPut(), doDelete() method
- Link to Liberty
  - At the @CICSProgram annotation



First program (OSGi environment)

```
import com.ibm.cics.server.*;

public class HelloWorld {

  public static void main(CommAreaHolder ca) {

    System.out.println("Hello CICS World");

  }

}
```

- Import statement indicates package containing JCICS classes
- The HelloWorld class only needs to contain one method
- Execution begins at the `public static void main(CommAreaHolder ca)` method
- A static method always exists (and there is only one per Java environment)
- System.out is a built in ‘output writer’ (writes to stdout)
- Problem: It is static

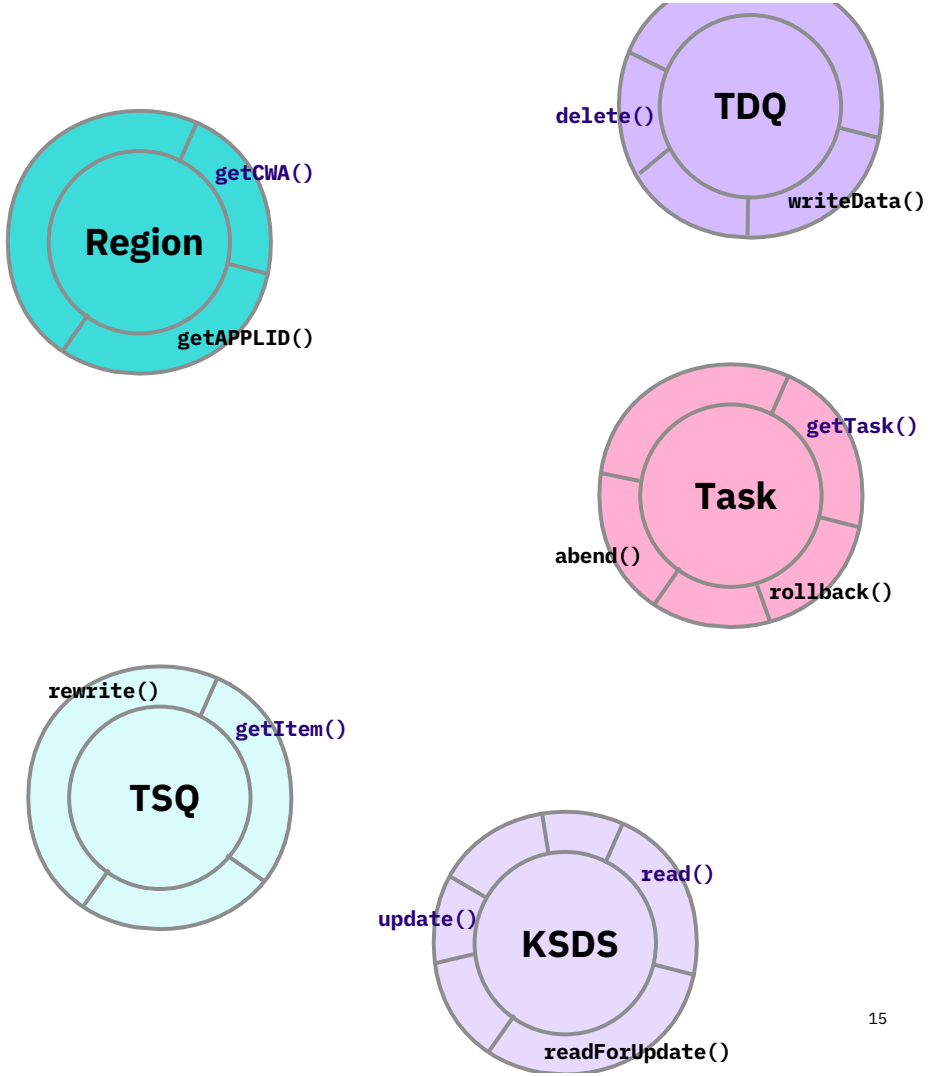
First program; object oriented

```
import com.ibm.cics.server.*;

public class HelloWorld {

  public static void main(CommAreaHolder ca) {

    new HelloWorld();

  }

  public HelloWorld() { // constructor

    System.out.println("Hello CICS World");

  }

}
```

- Execution starts at the static ‘public static void main’ method
- The static method instantiates itself as an object
- As an object, execution starts at the constructor
- A constructor has the same name as the class
- Comments in Java can start with a ‘//’

Interacting with the CICS environment

- CICS information is usually supplied in static or available objects
  - Region, Task, etc.
- Special CICS objects are used to access CICS resources
  - TDQ, TSQ, KSDS, Program



Region and Task objects

```
import com.ibm.cics.server.*;

public class HelloWorld {

  public static void main(CommAreaHolder ca) {

    Task myTask = Task.getTask();

    myTask.out.println("I am running in "+

      Region.getAPPLID()+".");

    myTask.out.println("I am going to abend");

    myTask.abend("BOOM");

  }

}
```

- A printwriter named out (and err) is available to the Task. It writes to a terminal, or to the CESO/CESE destination if not associated with a terminal
- **`Region.getAPPLID()`** is like EXEC CICS ASSIGN APPLID(location) END-EXEC
- The last line of code is equivalent to EXEC CICS ABEND ABCODE('BOOM') END-EXEC

ASSIGN and INQUIRE

```
Task myTask = Task.getTask();
if (myTask.getSTARTCODE() == "SD") {
  System.out.println("This is a started "+
       "task with data");
}
System.out.println("The current task number is "+
       myTask.getTaskNumber());
System.out.println("The current tranid is "+
       myTask.getTransactionName());
```

– CICS documentation lists the ASSIGN, INQUIRE, ADDRESS, and the EIB equivalents that can be accessed from Java

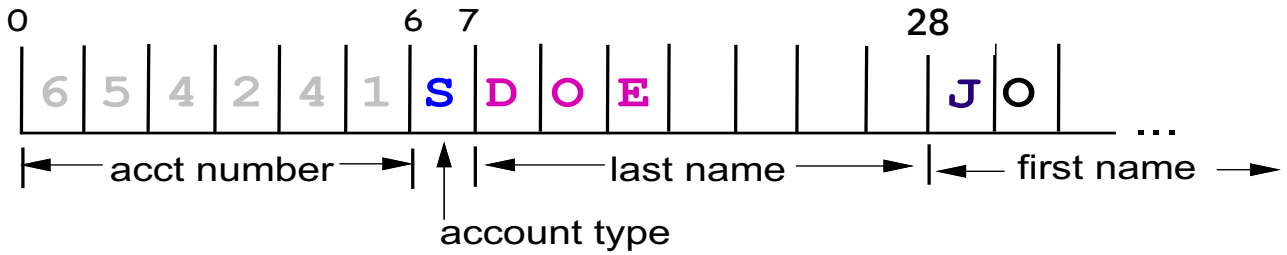
What about CICS data areas

- COMMAREAs
- CONTAINERs
- data read from or written to:
  - files
  - TD Queues
  - TS Queues
  - START data

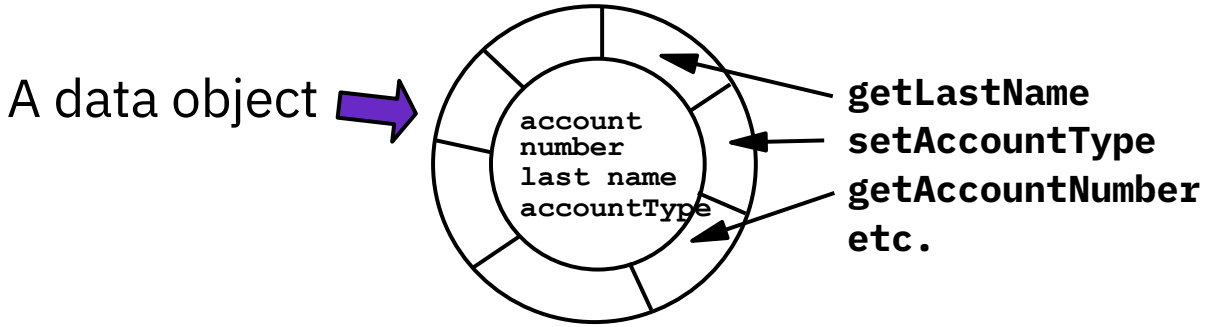
- COBOL:
  - A series of bytes
  - Accessible as ‘fields’
- Java
  - Java ‘String’ data type is unicode (2 bytes per character)
  - A series of bytes is different from the Java ‘String’
  - In Java a series of bytes is a data type called byte array (byte[ ])
  - Individual bytes are accessible by displacement (starting with 0)
  - Several ways to build and access ‘fields’ in Java

Data areas

COBOL



Java



Dealing with code pages

- You write to CICS resources using a byte array, not a String
  - default is to convert into a byte array in the ‘home code page’ of the JVM
- CICS runs Java in different code pages
- Java associated with a ‘PROGRAM’ definition (OSGi environment)
  - EBCDIC - the code page of your Unix System Services environment
- Java in the Liberty environment
  - ASCII

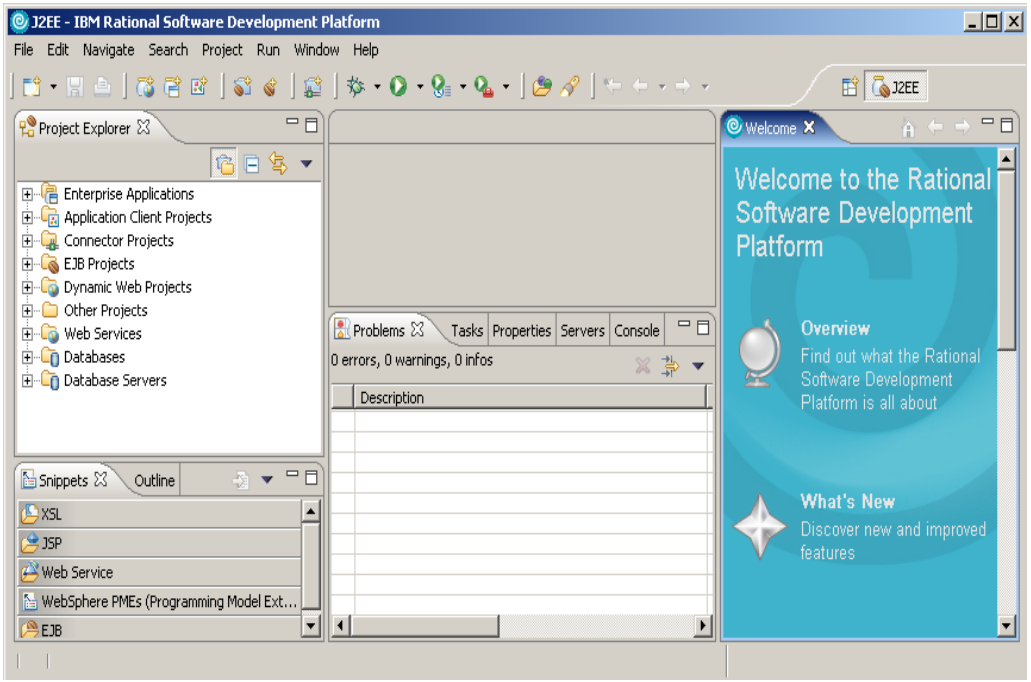
- You always want to write to CICS resources in EBCDIC
- Tell Java the code page you want the bytes in

```
String someText = "Record contents";
String myCodePage =
System.getProperty("com.ibm.cics.jvmserver.local.ccsid");
byte[] myBytes = someText.getBytes(myCodePage);
```

- The com.ibm.cics.jvmserver.local.ccsid property returns the code page specified for the region in the SIT parameter LOCALCCSID

Converting from a series of bytes to a byte array

- Generate Object (Recommended)
  - Rational Application Developer
  - Record Generator for Java
- Write generic Object classes
  - Your code
- Add code as needed
  - Concatenate
  - Substring
  - Indexing into byte array
  - bytestream (see supplied CICS Samples)



Java data bindings – Rational Application Developer and IDz

```
import com.ibm.cics.server.*;

//---- imports for Java Data Bindings go here

import com.dennis.recordlayouts.*;

public PrintAccountNumber class {

  public static void main(CommAreaHolder ca) {

    XYZCommArea myCommArea = new XYZCommArea();

    myCommArea.setBytes(ca.getValue());

    String theAccNum = myCommArea.getAccount__number();

    System.out.println("the Account Number is "+

          theAccNum+".");

  }

}
```

- Wizard in RAD/IDz reads COBOL copybook
- Generates Java source code with accessors (GETter and SETter methods) for each field in the copybook
- Type of data returned by getAccount__number() is determined by how the field was defined in the COBOL data layout (character, packed decimal, integer, floating point), returned as an appropriate Java data type

Record Generator for Java

```
import com.ibm.cics.server.*;
//---- imports for Java Data Bindings go here
import com.dennis.recordlayouts.*;
public PrintAccountNumber class {
  public static void main(CommAreaHolder ca) {
    XYZCommArea myCommArea = new XYZCommArea();
    myCommArea.setByteBuffer(ca.getValue());
    String theAccNum = myCommArea.getAccountNumber();
    System.out.println("the Account Number is "+
          theAccNum+".");
  }
}
```

- Record Generator runs as a batch job on z/OS
  - Input is ADATA output from COBOL compilation or assembler
- Generates Java source code with accessors (GETter and SETter methods) for each field in the selected data structure
- Type of data returned by getAccountNumber() is determined by how the field was defined in the COBOL data layout (character, packed decimal, integer, floating point), returned as an appropriate Java data type

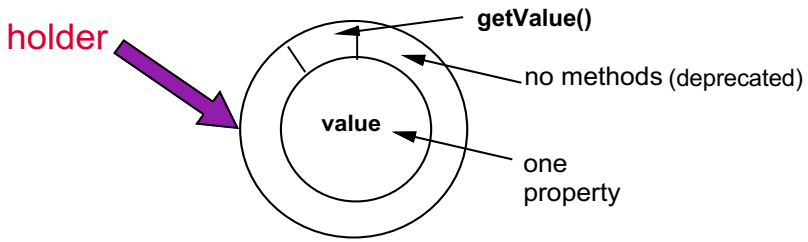
Java code for byte arrays

```
//---- below is substring technique
String theAcctType =    // ‘record’ is a byte Array
   (new String(record)).substring(6,7);
String theLastName =
   (new String(record)).substring(7,28).trim();

//---- indexing data
record[6] = ("S".getBytes())[0];
```

```
//---- below is concatenate technique
theAcctNum = (theAcctNum+"      ").substring(0,6);
theAcctType = (theAcctType+" ").substring(0,1);
theLastName=(theLastName+"       ").substring(0,21);
byte[] record2 = (theAcctNum+theAcctType+
      theLastName).getBytes();
```

Holders --

- Holder is an object with one property and one method
  - Holders (except for one) have a single data property called "value" of type byte array
  - Current versions of CICS use a getter: getValue()
  - The exception is the RetrievedDataHolder which contains additional data
- Byte arrays are received from CICS in "Holders“
- Give CICS a holder object, CICS places the data in the holder
- Examples: CommAreaHolder, ItemHolder, DataHolder, RecordHolder



Using holders – example

```
/ import for CICS Java Data Bindings (my holders)
import my.data.layouts.*;     // <<---
// import for JCICS API
import com.ibm.cics.server.*;
:
public static void main(CommAreaHolder ca) {
   XYZCommArea myCommArea = new XYZCommArea();
   myCommarea.setBytes(ca.getValue()); // <<--
   String theAcctNum = myCommarea.getAccount__Number();
   System.out.println("The Account Number is "+
        theAccNum+".");
```

```
:
  XYZRecordLayout myRecord = new XYZRecordLayout();
  RecordHolder myRecHolder = new RecordHolder();
  // read record that has a key of theAccNum here
  myRecord.setBytes(myRecHolder.getValue());
  String theName = myRecord.getAccount__name();
```

This example is using data objects defined by Java Data Bindings wizard in RAD

Commarea length

COBOL:

```
01  DFHCOMMAREA.
    05 RETURN-CODE              PIC 99.
    05 EMPLOYEE-NUMBER          PIC 9(6).
    05 EMPLOYEE-NAME            PIC X(20).
:
PROCEDURE DIVISION USING DFHEIB, DFHCOMMAREA.
    If EIBCALEN NOT = 28
        EXEC CICS ABEND ABCODE('COML') END-EXEC.
```

Java:

```
public class EmployeeLookup {
   public static void main(CommAreaHolder ca) {
      if (ca.getValue().length != 28) {
         Task.getTask().abend("COML");
      }
```

Every byte array has a property called 'length' that contains the number of bytes in the byte array

Exception handling in COBOL

CICS supports multiple ways to check for exceptional conditions in applications:

- EIBRESP & EIBRESP2
- HANDLE CONDITION
- RESP() and RESP2 operands on API commands
- The example shows the preferred way to work with exceptions

```
EXEC CICS WRITE FILE('MYFILE')
       FROM(RECORD-DATA)
       LENGTH(SIZE OF RECORD-DATA)
       RIDFLD(RECORD-KEY)
       RESP(RESP-FLD) RESP2(RESP2-FLD)
       END-EXEC.
EVALUATE RESP-FLD
    WHEN DFHRESP(DUPREC)
*         ---> code for duplicate record here
    WHEN DFHRESP(NOSPACE)
*         ---> code for out of space condition
END-EVALUATE.
```

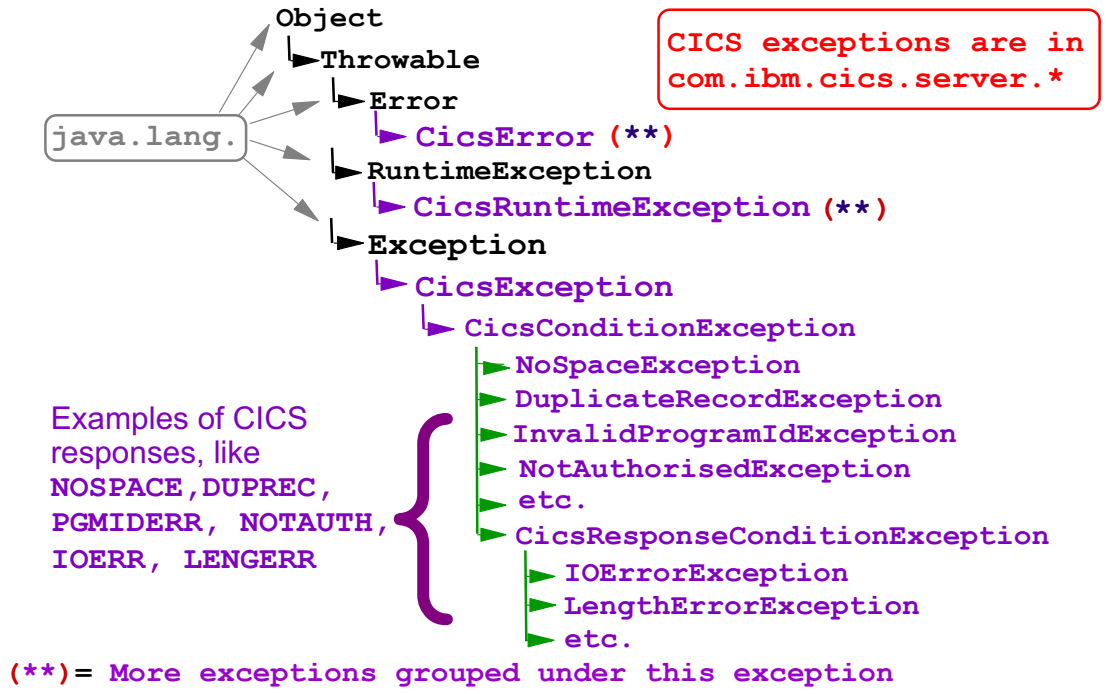
Handling exceptions in Java

- In Java you try an operation and catch an exception if one is ‘thrown’ (try-catch block)
- The getUSERID() method of the task object is written so that you must use a try-catch block
- The getUSERID() method can only throw one exception which is InvalidRequestException
- All exceptions that can be thrown must be caught

```
String strUserId;
try {
   strUserId = myTask.getUSERID();
} catch (InvalidRequestException ire) {
   System.out.println("InvalidRequestException "+
       "caught while trying to get the userid "+
       "from my task object. ");
   ire.printStackTrace();
}
```

Exception inheritance in CICS

You **try** a JCICS method, then **catch** exceptions that correspond to CICS response codes



Catching exceptions – example 1

```
try {
   // try some CICS request here
} catch (InvalidRequestException a) {
     System.out.println("There was an" +
         "invalid request.");
} catch (IOErrorException b) {
     System.out.println("There was an IO Error.");
     System.out.println("The resp2 data "+
         "was "+b.getRESP2());
} catch (LengthErrorException c) {
     System.out.println("There was a length error.");
   // catch any other CICS condition this command can throw
} catch (CicsConditionException cce) {
     System.out.println("Unexpected CICS Condition:"+cce);
} finally {
     System.out.println("The finally is always "+
         "executed");
}
```

- One common approach to handling exceptions is simply to write one catch block for each anticipated (or possible) exception
- Note that the last catch statement is for the top-level exception condition – CicsConditionException – a catch-all for any unhandled individual exceptions

Catching exceptions – example 2

```
try {
   // try some CICS request here
} catch (CicsConditionException t) {
     if (t instanceof InvalidRequestException) {
           System.out.println("There was an" +
               "invalid request.");
     } else if (t instanceof IOErrorException) {
           System.out.println("There was an IO Error. ");
           System.out.println("The resp2 data "+
               "was "+t.getRESP2());
     } else if (t instanceof LengthErrorException) {
           System.out.println("There was a length error.");
       // more test here for any other CICS conditions
     } else { System.out.println("CICS Condition: "+t);
}
```

- Another commonly used technique for handling exceptions is to simply catch the exception and examine it for the type of error which occurred
- This technique is similar to the CICS programming technique of capturing RESP and RESP2 values and then examining the values

Accessing CICS resources

- CICS resources are represented by objects of the appropriate type
  - Files
  - Programs
  - Temporary Storage
  - Etc.
- Use `setName()` to specify resource name
  - set other characteristics as appropriate
- Invoke method specific to desired action
- Data to and from CICS is in byte arrays
- The byte arrays from CICS are in Holders

WRITEQ TS in COBOL & Java

```
MOVE ‘Message to be written to my TSQ’ to TSQ-MSG.
EXEC CICS WRITEQ TS QUEUE('DENNIS')
           MAIN
           FROM(TSQ-MSG)
           LENGTH(LENGTH OF TSQ-MSG)
           RESP(RESP-FLD) RESP2(RESP2-FLD)
           END-EXEC.
     EVALUATE RESP-FLD
         WHEN DFHRESP(LENGERR)
*           ---> code for length error here
         WHEN DFHRESP(NOSPACE)
*           ---> code for out of space condition
         WHEN DFHRESP(ITEMERROR)
*           ---> code for item error here
     END-EVALUATE.
```

```
TSQ myTSQ = new TSQ();          // Create TSQ object
myTSQ.setName("DENNIS");        // Name of the Queue
myTSQ.setType(TSQType.MAIN);    // main is not default
String myMsg = "Message to be written to my TSQ";
String myCodePage = System.getProperty
           ("com.ibm.cics.jvmserver.local.ccsid");
try {
  int numItems =
        myTSQ.writeItem(myMsg.getBytes(myCodePage));
    System.out.println("Just wrote item " + numItems);
} catch (LengthErrorException a) {
  System.out.println("Length error writing item");
} catch (NoSpaceException b) {
  System.out.println("No space to write item");
} catch (ItemErrorException c) {
  System.out.println("There was an item error");
} catch (CicsConditionException cde) {
  System.out.println("Unexpected condition: " + cde);
}
```

READQ TS in Java

```
TSQ myTSQ = new TSQ();                  // Create TSQ object
myTSQ.setName("DENNIS");                // Name of the Queue
ItemHolder myItem = new ItemHolder(); // Holder for data
String myCodePage =
   System.getProperty("com.ibm.cics.jvmserver.local.ccsid");
int k;
try {
  for (k = 1; k < 6; k++) {
    myTSQ.readItem(k, myItem);
    System.out.println("Item contents: " +
              new String(myItem.getValue(), myCodePage));
  }
} catch (ItemErrorException i) {
  System.out.println("Item "+k+" does not exist");
} catch (CicsConditionException c) {
  System.out.println("Unexpected condition: " + c);
}
```

- You can use the CICS-provided Javadoc to understand the syntax for any JCICS class
- Many integrated development environments (IDE) offer code assistance with features such as
  - syntax highlighting with visual cues
  - providing language specific auto-completion
  - checking for bugs as code is being written

Accessing TSQs – available methods

- `get/setDescription`
  - local description of resource
- `get/setName()`
  - get or set name of TSQ to be acted on
- `get/setSysid()`
  - get or set the SYSID for the indicated TSQ
- `get/setType()`
  - get or set TSQType (Main or Aux)
- `delete()`
  - delete a TS queue
- `int readItem(int, ItemHolder)`
  - read specific TSQ item
  - returns number of TSQ items in Queue
- `int readNextItem(ItemHolder)`
  - read next TSQ item
  - returns number of TSQ items in Queue
- `rewriteItem(int, byte[]`
  - rewrite indicated TSQ item
- `rewriteItemConditional(int, byte[])`
  - rewrite with NOSUSPEND
- `int writeItem(byte[])`
  - write TSQ item
  - returns item number just written
- `int writeItemConditional(byte[])`
  - write TSQ item with NOSUSPEND
  - returns item number just written
- `int writeString(String)`

CICS TSQ exceptions

INVREQ	- InvalidRequestException
IOERR	- IOErrorException
ITEMERROR	- ItemErrorException
LENGERR	- LengthErrorException
NOSPACE	- NoSpaceException
NOTAUTH	- NotAuthorisedException
QIDERR	- InvalidQueueIdException
SYSIDERR	- InvalidSystemIdExcption

ISCINVREQ - ISCInvalidRequestException

- Exceptions which can occur are listed by command within the CICS documentation
  - READQ TS
  - WRITEQ TS
  - DELETEQ TS
- Exceptions are also listed in the Javadoc for each resource class

VSAM file write in COBOL

```
     MOVE '10101'         TO REC-ACCOUNT.
     MOVE 'Weiand'        TO REC-LAST-NAME.
     MOVE 'Dennis'        TO REC-FIRST-NAME.
     MOVE '1234 Some St.' TO REC-ADDR-1.
     MOVE 'Roanoke'       TO REC-ADDR-2.
     MOVE 'TX'            TO REC-ADDR-3.
     MOVE 1               TO BALANCE.
     EXEC CICS WRITE FILE('EMPLOYEE')
            LENGTH(LENGTH OF RECORD-DATA)
            RIDFLD(REC-ACCOUNT)
            FROM(RECORD-DATA)
            RESP(RESP-FLD) RESP2(RESP2-FLD)
            END-EXEC.
     IF RESP-FLD NOT = DFHRESP(NORMAL)
*       ---> Do appropriate condition checking here,
*            Like duplicate record, no space, etc.
     END-IF.
```

VSAM file write in Java; two versions

Spaces ?? !!

```
String sAccount = "10101";
String sLast    = "Weiand           ";
String sFirst   = "Dennis         ";
String sAddr1   = "1234 Some St.      ";
String sAddr2   = "Roanoke            ";
String sAddr3   = "TX                 ";
String sBalance = "0000000001";
String sRec=sAccount+sLast+sFirst+sAddr1+sAddr2+
         sAddr3+sBalance;
String myCodePage = System.getProperty
           ("com.ibm.cics.jvmserver.local.ccsid");
KSDS myFile = new KSDS();
myFile.setName("EMPLOYEE");
try {
  myFile.write
     (sAccount.getBytes(myCodePage),sRec.getBytes(myCodePage));
  System.out.println("Record write successful");
} catch (CicsConditionException cce) { // do appropriate
   // condition checking, like dup record, no space, etc
  System.out.println("Exception caught: "+cce);
}
```

```
EmployeeRecordLayout emp = new EmployeeRecordLayout();
emp.setAccount("10101“);
emp.setLastName("Weiand“);     // don’t need filler bytes
emp.setFirstName("Dennis“);    // with Data Bindings or RecGen
emp.setAddress1("1234 Some St.“);
emp.setAddress2("Roanoke“);
emp.setAddress3("TX“);
emp.setBalance(1);            // much easier for numbers

KSDS myFile = new KSDS();
myFile.setName("EMPLOYEE");
try {   myFile.write(emp.getAccount().getBytes(),
                           emp.getBytes());
  System.out.println("Record write successful");
} catch (CicsConditionException cce) { // do appropriate
   // condition checking, like dup record, no space, etc
  System.out.println("Exception caught: "+cce);
}
```

VSAM file read in Java

```
KSDS myFile = new KSDS();
myFile.setName("EMPLOYEE");
RecordHolder rh = new RecordHolder();
String myCodePage =
   System.getProperty("com.ibm.cics.jvmserver.local.ccsid");
try {
   myFile.read("11111".getBytes(myCodePage),
                  SearchType.GTEQ, rh);
   System.out.println("Name on account is: "+
        (new String(rh.getValue(), 5, 15, myCodePage)));
} catch (RecordNotFoundException a) {
   System.out.println("Account not found");
} catch (NotOpenException b) {
   System.out.println("File not open");
} catch (NotAuthorisedException c) {
   System.out.println("Not authorized to use file");
}catch (CicsConditionException cce) { // catch the rest
   System.out.println("Unexpected exception: "+cce);
}
```

- This sample code fragment demonstrates one way to use the **`read()`** method for a key sequenced VSAM file.
- Note that not all possible exceptions are explicitly tested.

LINK from COBOL or Java

```
     MOVE 'Weiand'     TO COMM-EMP-NAME.
     MOVE 1            TO COMM-EMP-ACCOUNT-NO.
*
*      more moves here
*
     EXEC CICS LINK PROGRAM('EMPPROG1')
            COMMAREA(COMM-DATA)
            LENGTH(SIZE OF COMM-DATA)
            RESP(RESP-FLD) RESP2(RESP2-FLD)
            END-EXEC.
     EVALUATE RESP-FLD
         WHEN DFHRESP(NOTAUTH)
*       ---> code for not authorized here
         WHEN DFHRESP(PGMIDERR)
*       ---> code for Program id error here
     END-EVALUATE.
```

```
EMPPROGCommarea ec = new EMPPROGCommarea();
ec.setEmployeeName("Weiand");
ec.setEmployeeAccountNumber(1);
//
// more sets here
//
Program aProg = new Program();
aProg.setName("EMPPROG1");
try {
   aProg.link(ec.getBytes());    // Data Binding
   System.out.println("Link was successful");
} catch (InvalidProgramIdException a) {
   System.out.println("Program not found");
} catch (NotAuthorisedException b) {
   System.out.println("Not authorized to use pgm");
}catch (CicsConditionException cce) { // catch the rest
   System.out.println("Unexpected exception: "+cce);
}
```

START command

```
     EXEC CICS START TRANSID('DDW1')
            AFTER MINUTES(5)
            RESP(RESP-FLD) RESP2(RESP2-FLD)
            END-EXEC.
     EVALUATE RESP-FLD
         WHEN DFHRESP(NOTAUTH)
*       ---> check conditions here
     END-EVALUATE.
```

```
StartRequest startReq = new StartRequest();
startReq.setName("DDW1");
Calendar cal = new GregorianCalendar();
cal.add(Calendar.MINUTES, 5);
startReq.setTime(cal);
try {
  startReq.issue();      // issue start request
  System.out.println("Link was successful");
} catch (CicsConditionException cde) {
            //
            // check for appropriate conditions here
            //
}
```

Commarea on return from a LINK

- When a non-Java program is invoked with a LINK request and passed a Commarea, it receives a pointer to the storage allocated by the caller.
- Upon return from the LINK, whatever data is in that same area of storage is the reply data.
- Java doesn't support pointers and direct addressability to data areas.
- The CICS documentation documents how to copy the reply data into the CommareaHolder.

```
// return a value to CICS in the COMMAREA
System.arraycopy(myByteArray, 0,
                 ca.getValue(), 0,
                 myByteArray.length);
```

The `arraycopy()` copies from myByteArray starting at displacement 0, to the COMMAREA (ca) starting at displacement 0, for a length of the COMMAREA

Channels and Containers

Java programs can use Channels and Containers in the same way as non-Java CICS applications

Containers are named blocks of data designed for passing information between programs

Containers are grouped in sets called **channels**, which act as the interface between programs

Character data – DATATYPE(CHAR) – is passed to Java programs as String objects

Binary data – DATATYPE(BIT) – is passed to Java programs as Byte[] arrays

Character data is automatically converted at the API level

Binary data is flowed to the Java program unconverted

While COMMAREAs and START data are confined to a 32KB limit, Containers allow much more data to be passed

Not only can an individual container exceed 32K, multiple containers can be passed between programs within a channel

The use of multiple containers provides a high degree of flexibility about how to structure data

Get Current Channel

- Non-Java programs have implicit use of the Current Channel as default on all Container-based commands
- Java programs must explicitly instantiate a Channel object from the Current Channel – analogous to issuing ASSIGN CHANNEL command

```
import com.ibm.cics.server.*;
:
   Task myTask = Task.getTask();
   :
   Channel myChannel = myTask.getCurrentChannel();
   Container myContainer =
            myChannel.getContainer("myContainer");
   System.out.println("The container contents was: " +
            myContainer.getString());
```

Create a Container in a Channel

```
import com.ibm.cics.server.*;
:
:
   Task myTask = Task.getTask();
   :
   Channel myChannel =
        myTask.createChannel("myChannel");
   Container myContainer =
        myChannel.createContainer("myContainer");
   myContainer.putString("Some Container Data");
```

- Exceptions were omitted to simplify this example
- The **putString()** method is only valid with CHAR containers
- The **put()** method is available to place a byte[] into a Container
  - The put() method can create either a CHAR or BIT container

Browse through the Containers in a Channel

```
Task myTask = Task.getTask();
ContainerIterator ci = myTask.containerIterator();
if (ci != null) {
    int containerCount = 0;
    while (ci.hasNext()) {
      containerCount++;
      Container con = (Container)ci.next();
      String pData = con.getString());
      myTask.out.println("Container " + con.getName()+
                 " Contents are " + pData);
    }
} else {
    System.out.println("There are no containers");
}
```

- This sample assumes that all Containers are of DATATYPE_CHAR
- This loop iterates over the channel and for each container
  - It gets the name – **`con.getName()`**
  - And the contents – **`con.getString()`**
- If Channel contains mixed CHAR and BIT Containers, program can use **`getDataType()`** method to determine the type of data in the Container
  - DATATYPE_CHAR
  - DATATYPE_BIT

JCICS classes, part 1

- BMS and Terminal Control
  - converse(), receive(), send(), sendControl(), sendText()
  - no SEND MAP, RECEIVE MAP, HANDLE AID or WAIT TERMINAL
- Document API
- Common equivalents of ASSIGN, ADDRESS, INQUIRE
- FILE Control, including BROWSE
- LINK
  - no SUSPEND
- CANCEL, RETRIEVE, START
- Temp Storage
- Transient Data
- ENQUEUE, DEQUEUE
- APPC mapped conversations
- TRACE
- SYNCPOINT, ROLLBACK
- Asynchronous API
  - AsyncService
  - ChildResponse

JCICS classes, part 2

- WEB
  - HttpSession
  - HttpRequest
  - HttpResponse
  - HttpHeader
  - HttpClientRequest
  - HttpClientResponse
  - CertificateInfo
- SIGNAL EVENT
- TRANSFORM
  - XmlTransform
- INVOKE SERVICE
- WS-Addressing
- Channels and Containers
- IsCICS – Test whether we are in a CICS environment

Restrictions, limitations, and unsupported functions

- Cannot use JCICS API in an activator class of an OSGi bundle
  - Can use the CICSExecutorService in a bundle activator
- Must not use System.exit() method
- BMS Send Map and Receive Map **
- APPC unmapped conversations
- CICS Business Transaction Services (BTS)
- XCTL
- DUMP services
- Journal services
- Storage services
  - no GETMAIN – use normal Java storage management
- Timer services
  - START & CANCEL are supported
  - no POST, DELAY, WAIT EVENT
- System Programmer Interface (INQUIRE/SET/PERFORM, etc)

JCICX

- JCICSX API introduced in CICS TS V5.6
- JCICSX API classes extend parts of the JCICS API with the capability of remote development and mocking
- The JCICSX API classes support only a subset of CICS functionality
  - focused on linking to CICS programs using channels and containers
- Client-side tooling is available to enable Liberty users to use JCICSX to access CICS from a servlet

Benefits

- easy mocking and stubbing
- can be run remotely in development environments
- syntax is simplified and natural with more recent Java constructors
- Code written using the JCICSX API classes can execute without change, both in remote development mode and when deployed to run in CICS
- compatible with the JCICS API

JCICSX

- Supported functions
- Channels and Containers
  - Create Channel
  - Create Containers
  - PUT data into Containers
  - GET data from Containers
- Program LINK
  - LINK to program passing Channel
  - LINK to program with no data
- Note, there is no support for LINK passing Commarea
- Mocking out the CICS calls enables you to independently unit test the logic of your application
  - There are many mocking frameworks you can use
  - Example in documentation shows how to use Mockito to return some mocked contents of a container

Link to Liberty

- Java EE application is required to contain a plain old Java object (POJO)
  - packaged as a web archive (WAR) or enterprise archive (EAR) file
- A method in the Java EE application can be made a CICS program by use of the @CICSProgram annotation
- CICS creates the program resource defined by the @CICSProgram annotation
- Data passed between non-Java and Java programs using Channels and Containers
  - COMMAREA is not supported
- Java application runs under the same unit-of-work (UOW) as the calling program
  - updates made to recoverable CICS resources are committed or backed out when the transaction ends
  - when the Java application is invoked, there is no JTA transaction context
  - If the application starts a JTA transaction, a syncpoint is performed to commit the CICS UOW, and create a new one

Link to Liberty security

- Link to Spring Boot application from a CICS program, the CICS user ID is not passed to Spring security
- Link to a Java EE application from a CICS program, the user ID of the CICS task is passed into the Java EE application
  - Liberty does not authenticate the user, but trusts the identity that is passed in by CICS
  - Liberty does check that the user ID is present in the configured user registry
    - Where possible, use the SAF registry in Liberty

Summary

- How CICS supports Java applications
  - Types of applications
  - JVM server environments
- Program invocation
  - Entry points
- Passing data to and from Java programs
  - Commareas, Channels & Containers
  - Code page considerations
- Handling exceptions
- Accesssing CICS resources
  - JCICS classes
  - JCICSX classes

Read more

IBM Rational Application Developer for WebSphere Software

– https://www.ibm.com/docs/en/radfws/9.7

IBM Developer for z/OS 16.0.x

– https://www.ibm.com/docs/en/developer-for-zos/16.0

IBM Record Generator for Java

– https://www.ibm.com/docs/en/record-generator/3.0

IBM SDK, Java Technology Edition 8

– https://www.ibm.com/docs/en/sdk-java-technology/8

Redbook SG24-8038 – CICS and the JVM server: Developing and Deploying Java Applications

– https://www.redbooks.ibm.com/abstracts/sg248038.html?Open

Redbook SG24-8418 – Liberty in CICS: Deploying and Managing Java EE Applications

– https://www.redbooks.ibm.com/abstracts/sg248418.html?Open

Redbook SG24-8335 – IBM CICS and Liberty: What You Need to Know

– https://www.redbooks.ibm.com/abstracts/sg248335.html?Open

Learn more – IBM Redbook courses

Architecting Java solutions for CICS

– https://www.redbooks.ibm.com/abstracts/crse0301.html?Open

Extending a CICS web application with JCICS

– https://www.redbooks.ibm.com/abstracts/crse0302.html?Open

Developing a RESTful Web application for Liberty in CICS

– https://www.redbooks.ibm.com/abstracts/crse0300.html?Open