Java™ For RPG Programmers

Don Yantzi (yantzi@ca.ibm.com)
Session: 105-6

Michigan iSeries Technical Education Conference
Acknowledgement:

- This presentation is a collaborative effort of the IBM Toronto AS/400 Application Development presentation team, including work done by:
  - Phil Coulthard, George Farr
- This presentation is based on the books (www.mcpressonline/ibmpress)
  - Java for S/390 and AS/400 COBOL Programmers, 1-58347-011-5
- It also contains information from the related Student Workbook (www.mcpressonline)
  - Java for RPG and COBOL Programmers on AS/400 Student Workbook

Disclaimer:

- The information contained in this document has not been submitted to any formal IBM test and is distributed on an as is basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customers' ability to evaluate and integrate them into the customers' operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will result elsewhere. Customers attempting to adapt these techniques to their own environment do so at their own risk.

Reproduction:

- The base presentation is the property of IBM Corporation. Permission must be obtained PRIOR to making copies of this material for any reason.
• World of Java
• Java versus RPG:
  ➢ RPG IV and ILE Review
  ➢ Application Anatomy
  ➢ Syntax, Data Types, Variables
  ➢ Operators, Statements
  ➢ Arrays, Strings
• OO Terminology
• Exception Handling
What is Java?

• An OO programming language
  ➤ Created by Sun Microsystems Inc, in 1995
  ➤ Adopted by Netscape in 1996
  ➤ Heavy investment by IBM ever since

• Initially for:
  ➤ Applets that run in Web Browsers
  ➤ Applications that are client-GUI or server non-GUI

• Now also for:
  ➤ Servlets that run in a Web Server
  ➤ Enterprise JavaBeans that run on a server
  ➤ and much more...
"Write Once, Run Anywhere"

- Java code is interpreted
- Java interpreter has been ported to just about every operating system, Web browser, Web server, and hand-held device in existence today
- Java language comes with many pre-defined functions and services
  - In the form of "packages"
  - Reduces dramatically the need to rely on operating system APIs

"Learn Once, Use Everywhere"

- Use Java for GUI, Web, Business Logic, Tools, Business Applications, Games, . . .
Three Flavors of Java

• Java 2 Standard Edition (J2SE):
  ➤ For JavaBeans, applets, GUI/non-GUI application
  ➤ START HERE
    • but also use Servlet and JSP support from your Web Application Server

• Java 2 Enterprise Edition (J2EE):
  ➤ For Java Servlets, JavaServer Pages
    • Although also available via Application Servers such as WebSphere
  ➤ For Enterprise JavaBeans, Java Naming and Directory Interface, Java Messaging Service, . . . and much more!
  ➤ GROW HERE

• Java 2 Micro Edition (J2ME):
  ➤ For small, embedded devices: chips, phone, hand-helds...
• How is Java packaged?
  ▶ For developers
    • In a "Java Development Kit" (JDK)
      ▶ Compiler, runtime, command line tools
  ▶ For runtime
    • In a "Java Virtual Machine" (JVM)
      ▶ Interpreter

• How do you get Java?
  ▶ For developers
    • JDK from Sun (www.java.sun.com) or IBM (www.ibm.com/java)
    • JDK also built-in to Java tools like VisualAge for Java, WSSD
  ▶ For runtime
    • JVM built-in to many Operating Systems, Web Browsers, App'n Servers, PDAs, Cell Phones, etc
Command line tools

- JVM / interpreter ("java")
- Bytecode compiler ("javac")
- Doc'n tool ("javadoc")
- Compression tool ("jar")
- ...

Documentation

- JDK 1.1.5 Documentation - Netscape

Packages

- language
- database access
- math
- graphics
- file system
- networking
- utilities
- ...

math

...
JDK Licensing

Sun Microsystems writes JDK for Windows NT/2K/XP, Solaris, and AIX. It licenses JDK to IBM, Netscape, and others. IBM ports to OS/400, which includes OS/390, OS/2, AIX, and Linux. Netscape ports to other platforms. Others port to others.
• Classes

  ➤ **Compilation unit**
    • no matter what you are using Java for!
  ➤ **All fields and executable code are inside classes**
  ➤ **Source files are compiled into class files**

• Bytecode

  ➤ **What are inside class files**
  ➤ **Assembler language for Java**
    • what the JVM "interprets"
Java Beans

• **JavaBeans™**

  ➤ **Classes designed for fine-grained re-use**
    • Java's components, like Microsoft VB's VBX
    • Not to be confused with Enterprise JavaBeans!!
  ➤ **Beans contain**
    • properties (fields),
    • methods (paragraphs),
    • events (eg, button-pressed)
  ➤ **Tools can discover contents dynamically**
    ➤ And present list to use to select from or change

• **JAR™ Files (Java ARchive)**

  ➤ **Java way to group/compress class files**
    • for easy distribution (uses ZIP technology)
• Applications
  ➤ Java command line programs *(you call)*

• Applets™
  ➤ Java Web Browser programs *(Web Browser calls)*

• Enterprise JavaBeans™
  ➤ Enterprise-scale re-usable components *(Application Server calls)*
  ➤ Large scale (eg payroll) versus JavaBeans (eg, tax)

• Java Servlets™
  ➤ Java Web Server programs *(Web Server calls)*

• JavaServer Pages™
  ➤ HTML plus embedded Java *(Servlets call)*
Java Tools

• **Java Tools are**
  - **Optional**
    - minimal requirement: JDK + editor
  - **Productive**
    - eg, wizards and debuggers
  - **Numerous**
    - From IBM, Symantec, Sun, Inprise, ...

• **IBM Java Tool story**
  - **"Next generation" tools are**
    - WebSphere Studio Site Developer (Java, Web, XML tooling)
    - WebSphere Studio Application Developer (+ EJB tooling)
  - **For iSeries, there is**
    - WebSphere Development Studio Client
• Java and Web are a good fit!

  ➤ **Most common use of Java today for business is:**
    • Glue between business logic / transactions, and Web pages

  ➤ **This is done by**
    • Wrapping the business logic / transaction in a JavaBean/EJB
    • Calling the JavaBean/EJB from a **Java Servlet**
    • Creating **JavaServer Pages** (JSPs) that
      ➤ Are simply HTML static pages with "holes" for dynamic data
      ➤ Are called by the Java Servlet, which passes the dynamic data in the form of a simple Java Bean (think of this bean as a data structure)
      ➤ Are resolved into straight HTML by the JSP engine and passed to the Web Browser

• **Java Servlets and JSPs...**

  ➤ **are industry standard**
    • run in a Web Application Server that meets industry standard
Three flavors in 5.0

- Websphere Application Server Express
- Websphere Application Server
- Websphere Application Server Network Deployment

Runtime engine for JSPs and Servlets
- Plugs into Web server such as
  - IBM HTTP Server "classic", Apache, IIS, Domino

Runtime engine for EJBs
- Except in WAS Express

www.ibm.com/software/webservers
www.ibm.com/iseries/websphere
Web Tools

• Web Tools are for
  ➤ Web site management
    • Organizing Web projects and files
    • Publishing to test and product application servers
  ➤ HTML and other static content
    • Images, audio, video, etc
  ➤ Java Servlets and JavaServer Pages
    • For designing JSPs, generating servlets and JSPs via wizards

• IBM Web Tool story
  • WebSphere Studio Site Developer (Java, Web, XML tooling)
    • + iSeries Extensions = WebSphere Development Studio Client
  • WebSphere Studio Application Developer (+ EJB/J2EE tooling)
    • + iSeries Extns = WebSphere Development Studio Client Adv’d
• **Use **javac** to compile**
  ➤ **Use **java** to run from command line**
  • if it is application or to unit-test this individual class

  ![Diagram of Java LifeCycle]

  ![compile](image1)
  ![run](image2)
  ![Your app!](image3)

  ![javac](image4)
  ![class](image5)
  ![java](image6)

  • **For applets**
    • Use HTML/JSP file with `<APPLET>` tag pointing to the applet
  • **For servlets**
    • Use HTML/JSP file with `<FORM>` tag pointing to the servlet
  • **For EJBs**
    • Include in Web Application that is deployed to a J2EE Container

© 2003 IBM Corporation
/* Prototypical Hello World application */
public class Hello
{
    public static void main(String args[])
    {
        System.out.println("Hello World!");
    }
}
E:\mycode>java Hello
Hello World!
# Java Entry Points

<table>
<thead>
<tr>
<th>Where runs</th>
<th>Appliance</th>
<th>Applet</th>
<th>Servlet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anywhere</td>
<td>Web Browser</td>
<td>Web Server</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Who calls</th>
<th>Appliance</th>
<th>Applet</th>
<th>Servlet</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>Browser</td>
<td>Web Server</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>How invoked</th>
<th>Appliance</th>
<th>Applet</th>
<th>Servlet</th>
</tr>
</thead>
<tbody>
<tr>
<td>java command</td>
<td>&lt;APPLET&gt; html tag</td>
<td>By mapping to URL</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Entry point</th>
<th>Appliance</th>
<th>Applet</th>
<th>Servlet</th>
</tr>
</thead>
<tbody>
<tr>
<td>main method</td>
<td>init, then paint</td>
<td>init, then doGet / doPost</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Security restrictions</th>
<th>Appliance</th>
<th>Applet</th>
<th>Servlet</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Yes</td>
<td>Optional</td>
<td></td>
</tr>
</tbody>
</table>
How to access non-Java resources?

<table>
<thead>
<tr>
<th>Database</th>
<th>Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transactions</td>
<td>File System</td>
</tr>
<tr>
<td>Commands</td>
<td>Other...</td>
</tr>
</tbody>
</table>

Web Server + App Server

Servlet

Web page

Java EJB Application

Java server Application

Java client Application

Web Browser

Network
Standards for Accessing Data

**JDBC™**
- Dynamic SQL access to relational data or stored procedures
- Part of the Java standard
- Patterned after ODBC, but with OO versus C-APIs
- JDBC driver manager comes with Java
- JDBC drivers supplied by DB vendors or others
  - IBM UDB, HIT Software, Oracle, Sybase, Inprise, . . .

**SQL/J**
- Static SQL embedded inside Java
- Created by Oracle, supported by IBM UDB

**Java Stored Procedures**
- IBM UDB supports writing stored procedures in Java
iSeries Toolbox For Java

• 100% Java classes for:
  ➤ JDBC access to DB2/400
  ➤ DDM record level access to DB2/400
  ➤ Data Queue access
  ➤ Print access
  ➤ Program Call, Command Call
  ➤ File system access
  ➤ Client to server connections, remote login
  ➤ Much more...

• Runs anywhere
  ➤ OS/400, Windows, Linux, Unix, . . .

free!

Shipped with OS/400, WDS

© 2003 IBM Corporation
RPG IV
and ILE
review
We compare Java to RPG IV

- closer match to Java than III
- more modern constructs
- easier skills transfer to Java!

Continue your RPG IV journey!
✓ Longer names (10, but 4096 in V3R7)
✓ Mixed case (folded to upper by compiler)
✓ New D spec (Definition) for declares
✓ Free form expressions in factor 2 of some op-codes: EVAL, IF, DOW, DOU, WHEN
✓ New data types
  - Date, Time, Timestamp, Integer, Float, Null, Variable-Length (V4R2), Indicator (V4R2) fields
✓ Built-in functions (like %TRIML / R)
✓ Procedures ("grown up subroutines")
  - fast intra/inter-module calls. New CALLP op-code
ILE Compiling, Binding

**ILE Program**
- **CRTPGM**
  - ILE Program (*PGM)

**ILE Service Program**
- **CRTSRVPGM**
  - ILE Service Program (*SRVPGM)

**ILE Modules**
- **CRTRPGMOD**
  - ILE Modules (*MODULE)

**RPG IV Source Members (RPGLE)**
- **CRTRPGMOD**

**Compile**
- **CRTRPGMOD**
- **CRTBNDRPG**

**Bind**
- **CRTCSPVPGM**
- **CRTBNDRPG**

**ReUsable!**

**Link**
- **CRTBNDRPG**

© 2003 IBM Corporation
allow you to extract out common code

are linked, not bound, to *PGMs
• Modules call each other...
  ➤ by calling procedures

• *PGMs call *SRVPGMs...
  ➤ By calling procedures
RPG IV Procedures

- Local Variables
- Return Values
- PARMS: Value & Reference
- Recursion
- Exporting
- Prototyping

"grown up subroutines"

- Modularity
- Re-Use
- Skills transfer to Java (methods)
**Anatomy of Procedures**

**Pmax**

* return value

**max**

* parameters

**parm1**

**parm2**

* local variables

**temp**

* local code

```
IF parm1 > parm2
  EVAL temp = parm1
ELSE
  EVAL temp = parm2
ENDIF
RETURN temp
```

* end of procedure

**Pmax**
• WDSc has a Procedure Wizard

► generates skeleton code for you
► saves you from memorizing procedure syntax
Java versus RPG
**RPG Module vs Java Class**

- **RPG Module**
  - Fields
  - Procedures

- **Java Class**
  - Variables
  - Methods

**Conversion Process**
- **CRTRPGMOD** to RPG IV
- **javac** to Java
The class statement

```java
public class Customer {
    private int custId;
    private char custCode;
}
```

- **class keyword:** declares what we're defining
- **public modifier:** anyone can use
- **private modifier:** only code in this class can access
- **braces { } delimit start and end of class**

**Global variables**

```java
public static void main(String args[]) {
    ...
}
```

- **"main" method (described later)**
- **Methods (described next)**
public void setId(int idParm) {
    // Code goes here
} // end of the method

private int read() {
    // Code goes here
} // end of the method

private modifier: only code in this class can call

void keyword: nothing returned

public modifier: anyone can call

braces {} delimit start and end of methods

Java

method name

parameter type + name

integer value returned

Comments
int max(int parm1, int parm2) {
    int retval;
    if (parm1 > parm2) {
        retval = parm1;
    } else {
        retval = parm2;
    }
    return retval;
}

no public modifier specified so only classes in this package can call this method

int is the integer data type in Java
• Class names should
  ➤ be all lowercase except
    • first letter of each word
    • eg: OrderEntry

• Method / field names should
  ➤ be all lowercase except
    • first letter of each word other than first
    • eg: processOrder

• Constants should
  ➤ be all uppercase
    • eg SUNDAY
Packages

- **RPG IV**
  - contains procedures

- **orderentry**
  - Order module
  - BackOrder module
  - Inventory module

- **SRVPGM**
  - contains methods

- **Java**
  - orderentry package
    - Order class
    - BackOrder class
    - Inventory class
"package" must be first statement in source file

"package" is like compiler directive

```
package orderentry;

public class Order {
    ...
}
```

```
package orderentry;

public class BackOrder {
    ...
}
```

```
package orderentry;

public class Inventory {
    ...
}
```
"import" enables access to classes in package

"import" is like ADDLIBE. It is NOT like /COPY!

Order.java

```
import orderentry.*;

public class Order
{
    ...
}
```

can import one class or all (*)

Using Packages

Java
• Package names are
  ➤ usually all lowercase
  ➤ usually multi-part, dot separated

• Java-supplied packages
  ➤ all named java.xxx
  ➤ for example: java.awt or java.awt.event

• Your packages
  ➤ will start with com.xxx, where xxx.com is your company's domain name
  ➤ eg, IBM's start with com.ibm.xxx
• Packages
  ➤ have no file system objects!
  ➤ map to directories
    • One per dot-separated name part

• For example
  ➤ Consider package name com.abc.orders
## Comparing Anatomies

<table>
<thead>
<tr>
<th>RPG</th>
<th>JAVA</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>*PGM</td>
<td>Application</td>
<td>Program object == Application</td>
</tr>
<tr>
<td>*SRVPGM</td>
<td>Package</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Class</td>
<td>Compilation unit</td>
</tr>
<tr>
<td>Module</td>
<td>Class</td>
<td>Global variables</td>
</tr>
<tr>
<td>Fields</td>
<td>Variables</td>
<td>Functions</td>
</tr>
<tr>
<td>Procedures</td>
<td>Methods</td>
<td>Local Variable</td>
</tr>
<tr>
<td>Variables</td>
<td>Variables</td>
<td>Executable code</td>
</tr>
<tr>
<td>Code</td>
<td>Code</td>
<td></td>
</tr>
</tbody>
</table>
How are classes found?

- when referred to by code in other classes
- by the compiler (javac) and runtime (java)

Answer!

- by searching the CLASSPATH env variable

CLASSPATH

- is a list of semi-colon separated directories
  - colon separated on OS/400
- much like library list on OS/400!
  - system searches it for first match
• CLASSPATH entries are directories to search for classes

SET CLASSPATH = .;c:\myJava

search current directory

search c:\myJava directory
• For classes in packages
  - name parent directory containing subdirs

• Consider package com.abc.orders:
  - if com\abc\orders is off of c:\myJava . . .

SET CLASSPATH = .;c:\myJava

- searches inside .\com\abc\orders
- searches inside c:\myJava\com\abc\orders
Two options for distribution:

- ZIP files
  - Industry standard compression technology
- JAR files (Java ARchive)
  - Same as ZIP by written in Java, part of JDK

To compress multiple files together:

- Use WINZIP or PKZIP utilities on Windows or...
- Use jar command that comes with JDK

```
jar -cvf myClasses.jar *.class
```

create **myClasses.jar** file

put all class files in it
• You don't have to uncompressed!
  ➤ JVM can find and read classes directly from ZIP files and JAR files!!
  • That's cool!
• However, the .zip or .jar file must be on the CLASSPATH environment var
  ➤ place actual file name on path, not just dir!

SET CLASSPATH =
  .;c:\myJava;c:\myJava\myClasses.jar
• **Statements are free-format**
  - extra blanks and lines are ignored
  - statements end with semi-colon ;

• **Blocks use braces**
  - start and end of classes
  - start and end of methods
  - start and end of conditional / loop blocks

• **All names are case-sensitive**
  - abc NOT= ABC
  - even source file names are case sensitive
Java is totally free form
extra blanks and lines ignored

```java
void myMethod(int param1) {
    return;
}
```

```java
void myMethod(int param1) {
    return;
}
```

```java
return;
```

```java
return;
```
*Multi-line comment:*

```java
/* this is a multi-line comment */

* Please read these comments *
* as they are very important! *
```

*Single line comment:*

```java
// This whole line is a comment
int myVariable = 10; // Only this part is a comment
```

*JavaDoc comment:*

```java
/**
 * This is the <U>scan package</U>
 * this is the second line.
 * @author George & Phil
 * @version Feb 26, 2000
 */
```
**Can use special tags**

- special meaning to javadoc formatter
- can also use any HTML tags like `<b>bold</b>`

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@author</td>
<td>Author of this class or method</td>
</tr>
<tr>
<td>@see</td>
<td>References another class or method. Generates a link</td>
</tr>
<tr>
<td>@version</td>
<td>Version number of this class or method</td>
</tr>
<tr>
<td>@since</td>
<td>Release or version this class or method has existed since</td>
</tr>
<tr>
<td>@deprecated</td>
<td>This is an obsolete method</td>
</tr>
<tr>
<td>@return</td>
<td>Describes what this method returns</td>
</tr>
<tr>
<td>@param</td>
<td>Describes a parameter to this method</td>
</tr>
</tbody>
</table>
/**
 * Shows a message
 * @param message The msg string to show
 * @return void
 * @see MyClass#myMethod2(String message)
 */

public void myMethod(String message) {

Data Types Overview

RPG

IV

Numeric Types

Other Types

binary
integer
zoned
unsigned
packed
float
d/t/z

graphic
character
indicator
unicode

Numeric Types

Primitive Types

Graphic

integer
d/t/z

Character

unsigned

Java

integer

Graphic

unsigned

Primitive Types

Numeric

Float

byte
short
int
long
float
double

Reference Types

covered later

Boolean

Covered later

cover
<table>
<thead>
<tr>
<th>Type</th>
<th>In Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>int i;</td>
<td>4 byte signed: about +- 2 billion</td>
</tr>
<tr>
<td>Long</td>
<td>long l;</td>
<td>8 byte signed: about +- huge #</td>
</tr>
<tr>
<td>Byte</td>
<td>byte b;</td>
<td>1 byte signed: -128 to + 127</td>
</tr>
<tr>
<td>Short</td>
<td>short s;</td>
<td>2 byte signed: -32768 to 32767</td>
</tr>
<tr>
<td>Character</td>
<td>char c;</td>
<td>2 byte unicode. 1 char only!</td>
</tr>
<tr>
<td>Boolean</td>
<td>boolean flag;</td>
<td>true or false</td>
</tr>
<tr>
<td>Float Single</td>
<td>float f;</td>
<td>32 bit</td>
</tr>
<tr>
<td>Float Double</td>
<td>double d;</td>
<td>64 bit</td>
</tr>
<tr>
<td>RPG</td>
<td>Java</td>
<td>Comments</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>numeric (no decimals)</td>
<td>short or int</td>
<td>depends on length</td>
</tr>
<tr>
<td>numeric (with decimals)</td>
<td>float or double, or</td>
<td>depends on length. BigDecimal is</td>
</tr>
<tr>
<td></td>
<td>BigDecimal class</td>
<td>a Java supplied class</td>
</tr>
<tr>
<td>float (length 4)</td>
<td>float</td>
<td>Both are IEEE standard</td>
</tr>
<tr>
<td>float (length 8)</td>
<td>double</td>
<td>Both are IEEE standard</td>
</tr>
<tr>
<td>character (length one)</td>
<td>char</td>
<td>single character only</td>
</tr>
<tr>
<td>character (length n)</td>
<td>String class</td>
<td>A class, not a primitive type</td>
</tr>
<tr>
<td>graphic</td>
<td>String class</td>
<td>A class, not a primitive type</td>
</tr>
<tr>
<td>unicode</td>
<td>String class</td>
<td>A class, not a primitive type</td>
</tr>
<tr>
<td>indicator</td>
<td>boolean</td>
<td>'1' = true, '0' = false</td>
</tr>
<tr>
<td>date, time, timestamp</td>
<td>GregorianCalendar class</td>
<td>A class, not a primitive type</td>
</tr>
</tbody>
</table>
More on Boolean

- Can be assigned true or false:
  ```java
  boolean myFlag = true;
  ```

- Can be assigned an expression:
  ```java
  boolean myFlag = (rate > 10);
  ```

- Can be in an expression:
  ```java
  if (rate > 10) ... *** or ***
  if (myFlag)
  ```

- Can be negated:
  ```java
  myFlag = !myFlag;
  while (!myFlag) ...
  ```
What about Packed?

- **No packed decimal data type in Java**
  - Could use float / double, but precision is a problem for "fixed decimal" numbers

- **Answer: BigDecimal class**
  - Part of `java.math` package
  - A class, not a built-in "primitive" data type
  - *Software simulation* of fixed decimal numbers
  - Unlimited **precision** (total number of digits)
  - Program control over **scale** (number of decimal digits)
  - Methods include: *add*, *subtract*, *divide*, *multiply*, *setScale*

- **See also: BigInteger class**
Declaring Fields in RPG

On the C specification

Using the Define operation code

On the new Definition specification

**Beginning of data**

FQSYSPPRT O F 80 PRINTER OFLIND(*INOV)

D FIRST S 7A INZ('George ')

D AGE S 2B 0 INZ(25)

C *LIKE DEFINE FIRST LAST -3

C EVAL LAST='FARR'

C MOVE ' AGE WAS--->'AGETEXT 12

C EXCEPT RESULT

C MOVE *ON *INLR

OQSYSPPRT E RESULT

O FIRST 5

O LAST 10

O AGETEXT 22

O AGE 26
### Declaring Variables

Java

```java
modifiers type name = expression;

static final public private protected

byte short int long char float double boolean

class-name

literal, variable, computation
```
public class EmployeeRecord {
{
private int number;
private char type;
private String name;
private String address;
private Date hired;
private BigDecimal salary;
}
You do not specify #digits!

Data Type determines # of bytes
- which determines how much var can hold
- eg: short holds -32768 to 32767

Usually you will use:

- integer ("int") when no decimals (unless numbers > 2 billion)
- BigDecimal class when decimals needed
- String class when dealing with characters
public class EmployeeRecord
{
    private int number = 0;
    private char type = 'R';
    private String name = "Joe Public";
    private String address = "1 Young St";
    private Date hired = new Date();
    private BigDecimal salary = new BigDecimal("30000.00");
}
public class EmployeeRecordDefaults {
    static final int NUMBER = 0;
    static final char TYPE = 'R';
    static final String NAME = "Joe Public";
    static final String ADDRESS = "1 Young St";
    static final Date HIRED = new Date();
    static final BigDecimal SALARY =
        new BigDecimal("30000.00");
}

"static" and "final" keywords define a constant

D*..1....+....2....+....3....+....4....+....5
D*EmpRcdDFT         DS
D numberDFT          C       CONST(0)
D typeDFT            C       CONST('R')
D nameDFT            C       CONST('Joe Public')
D addressDFT         C       CONST('1 Young St')
D hiredDFT           C       CONST(D'1999-12-31')
D salaryDFT          C       CONST(30000)
**Wrapper Classes**

- **Primitive types have wrappers**
  - classes in `java.lang` package
    - always imported for you!
  - sometimes you will need them
    - such as for Vectors as we'll see
    - they also have handy methods and constants

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Wrapper</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>Byte</td>
</tr>
<tr>
<td>short</td>
<td>Short</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>long</td>
<td>Long</td>
</tr>
<tr>
<td>char</td>
<td>Character</td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>float</td>
<td>Float</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
</tbody>
</table>
Casting in RPG

....+... 1 ....+... 2 ....+... 3 ....+... 4 ....+... 5 ....+...

*************** Beginning of data *********************

FQSYSPT O F 80 PRINTER OFLIND(*INOV)
D DS1 DS
D int5 5I 0 INZ(25)
D BIN9 9B 0 INZ(22)
D ZONE9 9S 0 INZ(30)
D PACK9 9P 0 INZ(40)

C \[ MOVE \] BIN9 \[ INT5 \]
C \[ EXCEPT \] RESULT
C \[ MOVE \] PACK9 \[ INT5 \]
C \[ EXCEPT \] RESULT
C \[ MOVE \] ZONE9 \[ INT5 \]
C \[ EXCEPT \] RESULT
C \[ MOVE \] *ON \[ *INLR \]

OQSYSPT E RESULT
O INT5 15

************************** End of data ***********************

casting is always implicit in RPG
public class TestCast {
    public static void main(String args[]) {
        short sValue = 10; // 2 bytes
        long lValue = 30; // 8 bytes
        lValue = sValue; // implicit
        sValue = (short)lValue; // explicit
    }
}

• Casting in Java
  ➤ only implicit if target type larger than source
  ➤ else must explicitly cast: (target-type)source
What About OverFlow?

- Source won't fit in target?

  ➤ Nothing happens!!

    • No overflow indicators in Java!!

  ➤ You're job to check first before casting:

    • Use \texttt{MIN\_VALUE} and \texttt{MAX\_VALUE} constants in wrapper classes

```java
if ((lValue <= \texttt{Short.MAX\_VALUE}) &&
     (lValue >= \texttt{Short.MIN\_VALUE}))
    sValue = (\texttt{short})lValue; // cast
else
    // overflow/underflow error...
```
## Casting Summary Table

<table>
<thead>
<tr>
<th>Java Type</th>
<th>byte</th>
<th>char</th>
<th>short</th>
<th>int</th>
<th>long</th>
<th>float</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>No</td>
<td>Cast(^1)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>char</td>
<td>Cast</td>
<td>No</td>
<td>Cast(^1)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>short</td>
<td>Cast</td>
<td>Cast</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>int</td>
<td>Cast</td>
<td>Cast</td>
<td>Cast</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>long</td>
<td>Cast</td>
<td>Cast</td>
<td>Cast</td>
<td>Cast</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>float</td>
<td>Cast</td>
<td>Cast</td>
<td>Cast</td>
<td>Cast</td>
<td>Cast</td>
<td>Cast</td>
<td>No</td>
</tr>
<tr>
<td>double</td>
<td>Cast</td>
<td>Cast</td>
<td>Cast</td>
<td>Cast</td>
<td>Cast</td>
<td>Cast</td>
<td>No</td>
</tr>
</tbody>
</table>

\(^1\) Potential loss of sign
### RPG IV:
- free-format **EVAL** op-code & equal operator '=='

### Java:
- no op-code, just equal operator "="

<table>
<thead>
<tr>
<th>RPG III</th>
<th>RPG IV</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>C MOVE 0 X</td>
<td>C EVAL X = 0</td>
<td>x = 0;</td>
</tr>
</tbody>
</table>

### Java also allows stringing:
```
A = B = C = 25;
```
• Similar in both languages
• But in Java

➤ Body can be compound or single statement

• Single statement bodies don't need braces
### RPG Example

```c
if (age <= 2)
    EVAL PRICE = 0
else if (age <= 10)
    EVAL PRICE = 5
else
    EVAL PRICE = 10
endif
endif
```

### Java

```java
if (age <= 2)
    price = 0;
else if (age <= 10)
    price = 5;
else
    price = 10;
```

*note single statement in body so braces not required*
• Conditional operator '? :'
  ➤ also called a *ternary* operator
• Short form for *if* statement
  ➤ when only binary decision to make

result = (idx == 20) ? 30 : 35;

// same as...
if (idx == 20)
  result = 30;
else
  result = 35;
<table>
<thead>
<tr>
<th>RPG</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>C SELECT</code></td>
<td><code>switch (day)</code></td>
</tr>
<tr>
<td><code>C WHEN day = MON</code></td>
<td>`{</td>
</tr>
<tr>
<td><code>C* do something</code></td>
<td>`    case MON:</td>
</tr>
<tr>
<td><code>C WHEN day = TUE</code></td>
<td>`        // do something</td>
</tr>
<tr>
<td><code>C* do something</code></td>
<td>`        break;</td>
</tr>
<tr>
<td><code>C WHEN day = WED</code></td>
<td>`    case TUE:</td>
</tr>
<tr>
<td><code>C* do something</code></td>
<td>`        // do something</td>
</tr>
<tr>
<td><code>C WHEN day = THU</code></td>
<td>`        break;</td>
</tr>
<tr>
<td><code>C* do something</code></td>
<td>`    ....</td>
</tr>
<tr>
<td><code>C OTHER</code></td>
<td>`    default:</td>
</tr>
<tr>
<td><code>C* do something</code></td>
<td>`        // default code</td>
</tr>
<tr>
<td><code>C ENDSL</code></td>
<td><code>} // end switch statement</code></td>
</tr>
</tbody>
</table>

Improved readability over nested IFs

Structures are similar in both languages!
### RPG SELECT

- **SELECT**
- **WHEN** or **WHENxx**
- **OTHER**
- **ENDSL**

### Java Switch

- **switch**
- **case**
- **default**
- **end brace**

**Same But Different**

- Each **WHEN** expr evaluated until true
- Code executed until next **WHEN**
- **switch** expression evaluated
- Result compared to each **case**
- In first match, code executed until "**break**;" or end of **switch**
switch (day) {
    case 1:
    case 2:
    case 3:
        // Mon-Wed code
        break;
    case 4:
    case 5:
        // Thur-Fri code
        break;
    default:
} // end switch statement

Control goes to first "case" that matches the expression, then executes until "break" is encountered, or the end brace
RPG and Java, like all other languages have three main loops, they are...

<table>
<thead>
<tr>
<th>RPG</th>
<th>JAVA</th>
</tr>
</thead>
<tbody>
<tr>
<td>C start</td>
<td>DO</td>
</tr>
<tr>
<td>limit</td>
<td>limit</td>
</tr>
<tr>
<td>index</td>
<td>index</td>
</tr>
<tr>
<td>C*</td>
<td>:</td>
</tr>
<tr>
<td>ENDDO</td>
<td>ENDDO</td>
</tr>
<tr>
<td>C</td>
<td>DO</td>
</tr>
</tbody>
</table>

DO

<table>
<thead>
<tr>
<th>FOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOR</td>
</tr>
</tbody>
</table>

DO-WHILE

<table>
<thead>
<tr>
<th>WHILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHILE</td>
</tr>
</tbody>
</table>

DO-UNTIL

<table>
<thead>
<tr>
<th>DO-WHILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DO-WHILE</td>
</tr>
</tbody>
</table>

DO

<table>
<thead>
<tr>
<th>body</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
</tr>
</tbody>
</table>

while (expression)
<declare> and initialize index variable

loop while true

increment / decrement index

---

```java
static final int MAX = 10;

for (int idx=0; idx < MAX; idx++)
{
    // body;
}
```

---

```
C  1   DO  10   I
C* ...
C
C* initial-value DO Limit-value index
C  1   DO  10   I
C* ...
C* ENDDO Increment-value
C ENDDO 1
```
All three parts are optional

Only convention that:

- first part is for initializing index variable,
- expression is for comparing index value and
- increment is for incrementing/decrementing index value

All three can even be empty!

for (; ; )
    System.out.println("looping...");

Never ending loop!
Simple bodies can be done in incrementing part versus body

Comma-separated statements

```java
for (idx = 0; idx < myCharArray.length; myCharArray[idx] = ' ', idx++)
```

all work done in increment part. No need for body

two statements, comma separated
**Example 1: n!**

C*RN01Factor 1------Opcode----Factor 2------Result-Field
C*
C                   EVAL      Factorial = 1
C                   FOR       i = 1 to n
C                   EVAL      Factorial = Factorial * i
C                   ENDFOR

If n = 5,
n! = 5 * 4 * 3 * 2 * 1 = 120 ...

**Example 2: Last non-blank character**

C*RN01Factor 1------Opcode----Factor 2------Result-Field
C*
C                   FOR       i = %len(SayWhat) DOWNTO 1
C                   IF        %SUBST(SayWhat:i:1) <> ' '
C                     LEAVE
C                   ENDFOR

if SayWhat = 'New For RPG4',
Last non-blank = 12
boolean in30 = false;
while (!in30) {
    if (endOfFile())
        in30 = true;
    else
        readLine();
}

Java

- loop while true
- set variable to force end of loop
- loop iterations >= 0

RPG

Free Form Factor 2
Java

- Loop until true
- Set variable to force end of loop
- Loop iterations >= 1

boolean in30 = false;

\[
\text{do}
\{
    \text{if (endOfFile())}
    \text{in30 = true;}
    \text{else}
    \text{readLine();}
\} \text{while (!in30)};
\]
Note: `continue` and `break` can specify a labeled loop to explicitly iterate or leave.

```java
out: for (int i = 0; i < 10; i++)
{
    for (int j = 0; j < 10; j++)
    {
        if (intArray[i][j] == -1)
        {
            // some code
            continue out;
        }
        if (intArray[i][j] == -2)
            break;
    }
    // end inner for-loop
    // outside inner loop
} // end outer for-loop
```
### Operators: Relational

<table>
<thead>
<tr>
<th>Operation</th>
<th>Java Operator</th>
<th>RPG Op-Codes</th>
<th>RPG IV Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal</td>
<td>==</td>
<td>EQ</td>
<td>=</td>
</tr>
<tr>
<td>Not Equal</td>
<td>!=</td>
<td>NE</td>
<td>&lt;&gt;</td>
</tr>
<tr>
<td>Greater Than</td>
<td>&gt;</td>
<td>GT</td>
<td>&gt;</td>
</tr>
<tr>
<td>Less Than</td>
<td>&lt;</td>
<td>LT</td>
<td>&lt;</td>
</tr>
<tr>
<td>GT or Equal</td>
<td>&gt;=</td>
<td>GE</td>
<td>&gt;=</td>
</tr>
<tr>
<td>Lt or Equal</td>
<td>&lt;=</td>
<td>LE</td>
<td>&lt;=</td>
</tr>
<tr>
<td>Or</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>And</td>
<td>&amp;&amp;</td>
<td>ANDxx</td>
<td>AND</td>
</tr>
<tr>
<td>Negation</td>
<td>!</td>
<td>NOT</td>
<td>NOT</td>
</tr>
</tbody>
</table>
Relational Example

Java

```java
if ( (age <= 2) ||
    (age >= 65) &&
    (currDay == SENIORS_DAY) ) )
price = 0;
```

note double equals: ==

RPG

```rpg
C AGE IFLE 2
C AGE ORGE 65
C CURDAY ANDEQSENDAY
C MOVE 0 PRICE
C ENDIF
```

RPG

```rpg
C IF (age <= 2) OR
    ((age >= 65) AND
    (currday = SENIORS-DAY) )
C EVAL price = 0
C ENDIF
```
## Operators: Math

<table>
<thead>
<tr>
<th>Operation</th>
<th>Java Operator</th>
<th>RPG Op-Codes</th>
<th>RPG Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>+</td>
<td>ADD, Z-ADD</td>
<td>+</td>
</tr>
<tr>
<td>Subtract</td>
<td>-</td>
<td>SUB, Z-SUB</td>
<td>-</td>
</tr>
<tr>
<td>Multiply</td>
<td>*</td>
<td>MULT</td>
<td>*</td>
</tr>
<tr>
<td>Divide</td>
<td>/</td>
<td>DIV</td>
<td>/</td>
</tr>
<tr>
<td>Modulus</td>
<td>%</td>
<td>DIV and MVR</td>
<td>n / a</td>
</tr>
<tr>
<td>Power</td>
<td>Use exp or pow in Math class</td>
<td>n / a</td>
<td>**</td>
</tr>
</tbody>
</table>
Math Examples

RPG

C*  A = B+C
C  B  ADD  C  A  50
C*  A = (B+C)/12
C  B  ADD  C  A  50
C  A  DIV  12  A

C  EVAL  a = b + c
C  EVAL  a = (b+c)/12

Java

a = b + c;
a = (b+c)/12;
What does this mean?

```
x += 10;
```

**Answer:** short form for...

```
x = x + 10;
```

**All binary operators supported:**

```
x *= 10; x /= 2; y -= 1;
```

Same as using **ADD** op-code in RPG and *not* specifying factor 1 value
What does this mean?

\[ x++; \]

Answer: short form for

\[ x = x + 1; \]

Also supports decrementing:

\[ x--; \]

Can be before or after variable:

\[ ++x; --x; \]
Always changes variable

if (\(X++ > 10\))

Prefix:
- Increment variable, use value
  \(X = 10;\)
  \(Y = ++X + 2;\)

Suffix:
- Use value, increment variable
  \(X = 10;\)
  \(Y = X++ + 2;\)
**Bitwise Operators**

- RPG has op-codes
  - TESTB, BITON, BITOFF
- Java has operators...

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>Bitwise AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>Bitwise Exclusive OR</td>
</tr>
<tr>
<td>~</td>
<td>Bitwise negation</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>Left Shift</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Right Shift</td>
</tr>
<tr>
<td>&gt;&gt;&gt;</td>
<td>Zero fill right shift</td>
</tr>
</tbody>
</table>

*They work only on integer types!*
Reference Data Types

Java

Primitive Types
- Numeric
  - Integer
    - byte
    - short
    - int
    - long
  - Float
    - char
    - float
    - double
- boolean

Reference Types
- Java-Supplied
- User-Defined
  - Classes
  - Arrays
  - Strings
The following information will take some time (and more reading) to absorb

★ RELAX!

★ Focus on the "how" ...

★ ..the "why" will follow with time
To use a class, you must do two things:

• Declare an object reference variable:
  • Declare a variable using class as the type:

  MyClass myVariable;

• Instantiate an object using "new"

  myVariable = new MyClass();
Object reference variables are

- merely pointers, or references, to objects
- initially point to "null"
  - a keyword in Java

The **new** operator:

- allocates memory for the class ("instantiates")
  - Total memory needed by all global variables in class

Allocated memory known as

- object or
- instance of class
public class Customer {
    private int id;
    private String name;
    private String address;
    private int phone;
    private BigDecimal accountBalance;

    public void setId(int custId) {
        id = custId;
    }

    public boolean readInfo() {
        boolean readok = false;
        // read customer info from database
        return readok;
    }

    // more methods...
}

Customer aCust = new Customer();

// how to call the methods?

// you can declare & instantiate in one step!

Object Example Example
To call a method

- use *dot operator* on object reference variable

```
public class ProcessCustomer
{
    public static void main(string args[])
    {
        Customer aCust = new Customer();
        aCust.setId(100012);
        aCust.readInfo();
    }
}
```

Must use object reference variable, not Class name

Can also access non-private variables with dot operator
Why must we instantiate?

Because you can instantiate more than one!

Customer cust1 = new Customer();
cust1.setId(100011);

Customer cust2 = new Customer();
cust2.setId(100012);

Each gets their own memory

Each hold unique values for their variables

Hence we call global variables "instance variables"
• Classes are like templates
  ➤ or "cookie cutters"

• Classes have no memory allocated
  ➤ Objects have the actual memory
    • object == "instance of class"
    • object == actual "cookies"
Class vs Objects

- Classes like DDS source members?
- Objects like compiled *FILE objects?
- "new" operator like DDS compiler?
• Possible classes
  ➤ Customer
  ➤ Employee
  ➤ StateTax
  ➤ CustomerId
  ➤ EmployeeId
  ➤ Payroll
  ➤ Order
  ➤ PushButton

• Eg, in a GUI application
• Each instance might want different label text
Master object1 = new Master();
Master object2 = new Master();

object1 = object2;

Nobody points to object1 now so it is swept up by Garbage Collector
• Consider a **Stack class**
  ➤ for managing LIFO list of integers

```java
public class Stack {
    private int list[] = new int[100];
    private int topIndex = 0;

    public void push(int topValue) {
        list[topIndex++] = topValue;
    }

    public int pop() {
        return list[--topIndex];
    }
}
```

*instance variables*

*Warning: no error checking!*
Objects allow us multiple stacks simultaneously

```java
Stack myList = new Stack(); // allocate instance of stack
Stack myList2 = new Stack(); // allocate another instance

myList.push(100);          // stack contents: 100
myList.push(200);          // stack contents: 100, 200

myList2.push(1000);        // stack2 contents: 1000
myList2.push(2000);        // stack2 contents: 1000, 2000

int topValue;              // declare an integer variable
topValue = myList.pop();   // topValue: 200
topValue = myList2.pop();  // topValue: 2000
```
### Three ways to call methods:

- **Assignment statement**
  - returned result is saved in a variable

- **Expression**
  - returned result used in expression but not saved

- **Expression**
  - Runs the method and disregards return value

#### JRPG vs Java

<table>
<thead>
<tr>
<th><strong>RPG</strong></th>
<th><strong>Java</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>EVAL myVar = myProc(p1 : p2)</td>
<td>myVar = myObject.myProc(p1,p2);</td>
</tr>
<tr>
<td>IF myProc(p1 : p2) = 10</td>
<td>if (myObject.myProc(p1,p2) == 10)</td>
</tr>
<tr>
<td>CALLP myProc(p1 : p2)</td>
<td>myObject.myProc(p1, p2);</td>
</tr>
<tr>
<td>EVAL myVar = noParms</td>
<td>myVar = myObject.noParms();</td>
</tr>
<tr>
<td>IF noParms = 10</td>
<td>if (myObject.noParms() == 10)</td>
</tr>
<tr>
<td>CALLP noParms</td>
<td>myObject.noParms();</td>
</tr>
</tbody>
</table>
• **Method Overloading**

  ➤ Methods in same class *with same name!* But:
  • Number or type of parameters are different
  • `method name + nbr and type of parms == "signature"`

  ➤ *Official name 'method overloading'*

```java
public int max(int parm1, int parm2)
{
    // code to return max of two integers
}

public float max(float parm1, float parm2)
{
    // code to return max of two floats
}
```
Java variables can be **static**:

- Use static modifier (like RPG's STATIC keyword)
  - Cannot specify it on local variables in methods!
- Static variables are called **class variables**
  - Versus instance variables
- All objects share same value for static vars
  - Qualify *with the class name* to access them

```java
class RentalCar {
    static int totalRented = 0;
    public void rentCar() {
        // . . .
        ++totalRented;
    }
}
```

```java
if (RentalCar.totalRented > MAX_CARS)
```
• Methods can be static too
  ➤ called class methods
  ➤ Equivalent to standalone procedure
    • Call by qualifying with class name, not obj ref variable
  ➤ Cannot reference instance variables in the method

```java
class MyHelperRoutines {
    static int max(int p1, int p2) {
        if (p1 > p2) return p1;
        else return p2;
    }
}
```

If your method does not reference or use any instance variables, it should be static!

```java
int maxValue = MyHelpers.max(1000, 2000);
```
Classes can have constructors

- Special methods identified by:
  - Same name as class
  - No return value specified (not even `void`)

- Called by JVM when object created with `new`
  - Right after allocating memory for the object

- Your opportunity to do initialization
  - like *INZSR in RPG

```java
public class MyClass {
    public MyClass() {
        // . . .
    }
}
```
Constructors can take parms

- Declared same as in all methods
  - On method signature
- Passed by caller in parens after `new`
  - `MyClass myClass = new MyClass(10);`
- Usually to allow caller-specified initial values
  - For the instance variables

```java
public class MyClass
{
    private int myVariable;

    public MyClass(int parm1)
    {
        myVariable = parm1;
    }  // end ctor
}
```

"ctor" is common shorthand for "constructor"
Constructors can be overloaded

- Same as with all methods
  - Number or type of parameters must be unique

- Compiler, Runtime determine which to call
  - By matching number, type of new parameters

```java
private int myVariable;
private int myOtherVariable = 0;

public MyClass(int parm1)
{
    myVariable = parm1;
}

public MyClass(int parm1, int parm2)
{
    myVariable = parm1;
    myOtherVariable = parm2;
}
```

Constructor with no parms called **default constructor**

```java
MyClass obj1 = new MyClass(10);

MyClass obj1 = new MyClass(10, 20);
```
class AS400
{
    private String userId;
    private String password;

    AS400() // default constructor
    {
        this("PHIL", "GREATGUY");
    }

    AS400(String userId, String password)
    {
        this.userId = userId;
        this.password = password;
    }
} // end AS400 class

AS400 host1 = new AS400(); // Call default constructor
AS400 host2 = new AS400("GEORGE","OKGUY"); // Two parm ctor

use "this()" to call another constructor
/** Represents a single card in a deck */
public class Card
{

    // public constants...
    public static final int HEART = 0;
    public static final int CLUB = 1;
    public static final int SPADE = 2;
    public static final int DIAMOND = 3;

    // private instance variables...
    private int number; // value of card
    private int suit; // heart, spade, club, diamond
    private boolean played = false; // card been played yet?

    public Card(int number, int suit)
    {
        this.number = number;
        this.suit = suit;
    }

    public int getNumber()
    {
        return number;
    }

    public int getSuit()
    {
        return suit;
    }

    public boolean isPlayed()
    {
        return played;
    }

    public void setPlayed(boolean played)
    {
        this.played = played;
    }

} // end of class Card
# Main: RPG and Java

<table>
<thead>
<tr>
<th>What</th>
<th>RPG IV</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>How called</td>
<td>CALL command</td>
<td>java command</td>
</tr>
<tr>
<td>What compile unit gets control</td>
<td>first *MODULE without NOMAIN keyword</td>
<td>class identified on java command</td>
</tr>
<tr>
<td>What code gets control</td>
<td>first C-specs</td>
<td>main method</td>
</tr>
</tbody>
</table>

**Java's `main` method must look like:**

```java
public static void main(String args[]) {
    ...
}
```

- `Array of Strings: parameters from command line`
- `so JVM can call`
- `so JVM doesn't have to instantiate class`
- `nothing returned`
- `JVM looks for this name`
Writing To Console

* Prototype of this program main entry
DMAIN            PR             EXTPGM('HWORLD')
D STRING        1000A    OPTIONS(*VARSIZE)

* Definition of this program main entry
DMAIN            PI
D STRING        1000A    OPTIONS(*VARSIZE)

* Global variables
DOutString      S     52A

* Main logic
C              EVAL     OutString = 'Input: ' +
C                                       %TRIMR(%SUBST(STRING:1:45))
C              OutString  DSPLY

* End of program
C               MOVE     *ON     *INLR

// main class
public class HelloWorld
{
    // main method
    public static void main(String[] args)
    {
        // print first parameter passed
        System.out.println("Input: " + args[0]);
    }
}
public class Time
{
    private int hour, minute, second;

    public Time(int hour, int minute, int second)
    {
        this.hour = hour;
        this.minute = minute;
        this.second = second;
    }

    public String toString()
    {
        return "Time: ", " +
            minute + ", ", " +
    }

    public static void main(String args[])
    {
        Time torontoTime = new Time(08,30,0);
        Time sanFranTime = new Time(05,30,0);
        System.out.println(torontoTime);
        System.out.println(sanFranTime);
    }
Arrays, I Need Arrays

RPG

Array Types:
- One-dimension
- Tables
- Dynamic APIs

Initializing:
- Compile time
- Pre-runtime
- Runtime

Java

Array Types:
- One-dimension
- Multi-dimension
- Hashtable class
- Vector class

Initializing:
- Compile time
- Runtime
What About MODs?

What about Multiple Occurring Data Structures?

- In RPG these are arrays of structures
- In Java these are arrays of objects

✓ The object's class = the DS in RPG
Arrays in RPG

**CTDATA NAMES**
George
Phil
Sandra
Jennifer
Angelica

O* END OF SOURCE CODE
Arrays in Java

• Declaration

```java
int thisArray[ ];
long anotherArray[ ] [ ];
char[ ] orThisOne[ ];
```

• Declaration and Definition

```java
int thisArray[] = new int[1000];
long anotherArray[ ][ ] = new long[10][10];
char[ ] orThisOne[ ] = new char[20][20];
```

➤ Spacing not important
➤ # bracket pairs = # dimensions
➤ Type is same for all elements
➤ Arrays are objects! Require new
class TestMultiArrayRT
{
    public static void main(String args[])
    {
        int rtArray[][] = new int[3][3]; // Two dim array
        int value = 1;
        // Loop through all rows...
        for (int xIdx=0; xIdx < rtArray.length; xIdx++)
        {
            // Loop through all columns...
            for (int yIdx=0; yIdx < rtArray[xIdx].length; yIdx++)
            {
                rtArray[xIdx][yIdx] = value++;// assign and incr't
                System.out.print(rtArray[xIdx][yIdx] + " ");
            }
            System.out.println();
        }
    } // end main method
} // end TestMultiArrayRT class

- Arrays are zero-based in Java!
- Length is array instance variable

Use [var] to access elements
Java allows initializing at declaration time (**compile time**):

```java
String employee[] = {"ABC", "DEF", "GHI", "JKL"};
```

**Note:** `String` objects are covered shortly

```java
employee[0] = "ABC"
employee[1] = "DEF"
employee[2] = "GHI"
employee[3] = "JKL"
```

**Special Java syntax:**

- Values specified between curly braces
- Semi-colon needed after last brace
- Values for each element separated by commas
- No need to use `new` operator (implied)
In both RPG and Java, once an array is created, its size is fixed

- It cannot be resized
- However, Java at least allows deferring creation (using `new`) until after size has been determined

However, Java also offers *Vectors*!

- **Vector** is a class in `java.util` package
  - To use, need "import java.util.*;"
- Vectors are like dynamically sizable arrays
  - Do not need to specify initial size
  - Size grows as needed when items are added
To use Vectors:

- Create empty Vector by instantiation
- Add items using `addElement` method

```java
Vector myVector = new Vector();
String inputString = getFirstInput();
while (inputString != null)
{
    myVector.addElement(inputString);
    inputString = getNextInput();
}
```

- Query number of elements using `size` method
- Query specific element using `elementAt` method

```java
for (int idx = 0 ; idx < myVector.size() ; idx++)
    System.out.println(myVector.elementAt(idx));
```
• Java supplies a class for simple lookup tables
  ➤ **Hashtable** in package `java.util`
  ➤ Contains pairs of objects
    • A key object and a value object
  ➤ Objects can be of any class
  ➤ Use **put** to insert, **get** to retrieve

```java
Hashtable customers = new Hashtable();
customers.put("011002", "Phil Company");
customers.put("110034", "George Limited");
...
String georgeEntry = customers.get("110034");
```
• Strings are objects in Java
  • of the class `String` (in `java.lang` package)
• Language has special support:
  • You can concatenate with the "+" operator
  • You don't *have* to use the `new` operator

```java
String text1 = new String("George");
String text2 = new String("Phil");
String finalText = new String(text1);
finalText = finalText.concat(" and ");
finalText = finalText.concat(text2);
```

OR

```java
String text1 = "George";
String text2 = "Phil";
String finalText = text1 + " and " + text2;
```
• Be careful of these common mistakes:

  • **not assigning result of methods:**
    ```java
    String textField = "Java";
    textField.concat(" and RPG");
    ```
    ```java
    textField = textField.concat("and RPG");
    ```

  • **comparing with '==' versus equals method**
    ```java
    if (name == "Bob")
    ```
    ```java
    if (name.equals("Bob"))
    ```

  • **copying with '=' versus clone method**
    ```java
    String newName = oldName;
    ```
    ```java
    String newName = oldName.clone();
    ```

String is an *immutable* class: all methods return new objects versus changing existing.
# Strings: Java vs RPG

<table>
<thead>
<tr>
<th>RPG o/c</th>
<th>RPG built-in</th>
<th>Description</th>
<th>Java Method(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAT ('+')</td>
<td>Concatenate two strings</td>
<td>concat(string) or '+' operator</td>
<td></td>
</tr>
<tr>
<td>SUBST</td>
<td>%SUBST</td>
<td>Extract a substring from a string</td>
<td>substring(int start, int end) or substring(int start)</td>
</tr>
<tr>
<td>SCAN</td>
<td>%SCAN</td>
<td>Scan for a substring</td>
<td>indexOf()</td>
</tr>
<tr>
<td>%TRIM</td>
<td></td>
<td>Trim begin, end blanks</td>
<td>trim()</td>
</tr>
<tr>
<td>%LEN</td>
<td></td>
<td>Return length of string</td>
<td>length()</td>
</tr>
<tr>
<td>XLATE</td>
<td></td>
<td>Translate a string</td>
<td>Not Available</td>
</tr>
<tr>
<td>CHECK</td>
<td></td>
<td>Check for characters</td>
<td>Not Available</td>
</tr>
<tr>
<td>CHECKR</td>
<td></td>
<td>Check in reverse</td>
<td>Not Available</td>
</tr>
<tr>
<td>%TRIML</td>
<td></td>
<td>Trim leading blanks</td>
<td>Not Available</td>
</tr>
<tr>
<td>%TRIMR</td>
<td></td>
<td>Trim trailing blanks</td>
<td>Not Available</td>
</tr>
<tr>
<td>%CHAR</td>
<td>V4R2. Converts to string</td>
<td>valueOf(datatype value) in String class</td>
<td></td>
</tr>
<tr>
<td>%REPLACE</td>
<td>(V4R2) Allows replacement of substring</td>
<td>Not Available</td>
<td></td>
</tr>
<tr>
<td>METHOD</td>
<td>DESCRIPTION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>compareTo</td>
<td>Compares two Strings lexicographically</td>
<td></td>
<td></td>
</tr>
<tr>
<td>endsWith, startsWith</td>
<td>Test if String ends or starts with the specified string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>equals, equalsIgnoreCase</td>
<td>Compares this String to another, ignoring case</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBytes</td>
<td>Convert this String into a byte array</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getChars</td>
<td>Copies characters from this substring into the destination character array</td>
<td></td>
<td></td>
</tr>
<tr>
<td>regionMatches</td>
<td>Tests if two String regions are equal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>toCharArray</td>
<td>Converts this String to a new character array</td>
<td></td>
<td></td>
</tr>
<tr>
<td>toLowerCase</td>
<td>Converts all characters in String to lower case</td>
<td></td>
<td></td>
</tr>
<tr>
<td>toUpperCase</td>
<td>Converts all characters in String to upper case</td>
<td></td>
<td></td>
</tr>
<tr>
<td>valueOf</td>
<td>Converts primitive data type value to a String (this is a static method)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Java Date / Time

<table>
<thead>
<tr>
<th>Class</th>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>java.util</td>
<td>Simple date/time capture. No manipulation methods</td>
</tr>
<tr>
<td>GregorianCalendar</td>
<td>java.util</td>
<td>Rich date/time functionality, including comparing, adding, subtracting, extracting</td>
</tr>
<tr>
<td>SimpleDateFormat</td>
<td>java.text</td>
<td>For creating &quot;formatting objects&quot; that will format any given Date object to the specified format pattern</td>
</tr>
<tr>
<td>TimeZone</td>
<td>java.util</td>
<td>For creating timezone objects representing any timezone. Apply to GregorianCalendar or SimpleDateFormat objects to get equivalent date/time in that timezone</td>
</tr>
</tbody>
</table>
import java.util.*;

public class TestDate {
    public static void main(String args[]) {
        GregorianCalendar gc = new GregorianCalendar();
        System.out.println("Date before addition: " + gc.getTime());
        gc.add(Calendar.DATE, 2); // add two days
        System.out.println("Date after addition: " + gc.getTime());
        gc.add(Calendar.MONTH, 2); // add three months
        System.out.println("Date after addition: " + gc.getTime());
        GregorianCalendar gc2 = new GregorianCalendar(2012, 0, 30);
        if (gc.before(gc2))
            System.out.println("Yes, it is");
    }
}

Cool methods:
- add
- before / after
- isLeapYear
- get (extracting parts)

GregorianCalendar knows all about dates

Date before addition: Thu Sep 30 22:02:20 EDT 1999
Date after addition: Sat Oct 02 22:02:20 EDT 1999
Date after addition: Wed May 16 22:02:20 EDT 2001
Yes, it is
Agenda

00 Terminology
What does "Object Oriented" mean?

Three attributes:

- Encapsulation
- Inheritance
- Polymorphism
- Hide data from direct public programmer access
- Force access only via procedures or methods

ReUse with **Modules** and **Service Programs** of modules
"Expose" certain procedures or data with **EXPORT** keyword

ReUse with **Classes** and **Packages** of classes
"Expose" certain methods or variables with **public** modifier keyword
// class Employee
public class Employee {
    string name;
    int salary;
    public Employee(string id, int sal) {
        name = id;
        salary = sal;
    }
    public void printEmployee() {
        System.out.print("My name is "+name);
    }
}

// class SubEmployee
public class SubEmployee extends Employee {
}

- Child class extends parent class
  - inherits methods, variables
- Child can also:
  - Add new methods, variables
  - Override methods

Java
Java

00: Inheritance

Class Animal
- eat()
- sleep()
- talk()

extends

Class Dog
- talk()
  { println("bark"); }
- overrides "talk()"

Class Cat
- talk()
  { println("meow"); }
- purr()
- overrides "talk()"
- adds "purr()"

Notes:
- Can only extend one class
- Extended class called parent
- Extending class called child
- Signature important when overriding methods

Cat fluff = new Cat();
fluff.eat(); fluff.talk(); fluff.purr();

can call inherited methods as though locally defined
No limit to inheritance tree

- All child classes inherit methods of all parents

Just remember

- You can only extend one class
- Only one immediate parent
  - But grandparents allowed

Java

```java
TomCat gruff = new TomCat();
gruff.eat();
gruff.talk();
gruff.purr();
```
• When `ClassChild` extends `ClassParent`:

  ▶ You can assign `ClassChild` objects to `ClassParent` object reference variables
  • either direct child or indirect child (eg: grandchild)

  ```java
  ClassParent obj = new ClassChild();
  ```

  ▶ You can then call any method in the `ClassChild` object
  • as long as it is defined in the parent class too
  • if not, compile will fail (it searches declared class type)

  ```java
  obj.commonMethod();
  ```
Why is this important?:

- You can write generic code that calls parent's "base" methods.

  But actually calls child's methods at runtime, if they are overridden in the child class.

```
Class ParentClass {
    public void doSomething() {
        // ...
    }
}

Class ChildClass extends ParentClass {
    public void doSomething() {
        // different algo
    }
}
```

```
ParentClass obj1 = new ParentClass();
ParentClass obj2 = new ChildClass();
obj1.doSomething();
obj2.doSomething();
```

"Poly"...."morphism"
"Many".."faces"
public class Employee
{
    protected int id;
    protected String name;
    protected BigDecimal wage;
    protected BigDecimal hoursWorked;

    public Employee(int iD)
    {
        id = iD;
        // not shown: reading info from database
    }

    public BigDecimal calculatePay()
    {
        return wage * hoursWorked;
    }

    // other methods: getName, setName, etc
}

Consider this "parent" class

"protected" modifier allows only this class and child classes access

What about salaried employees?
public class SalaryEmp extends Employee
{
    public SalaryEmp(int iD)
    {
        id = iD;
        // not shown: reading info from database
    }
    public BigDecimal calculatePay()
    {
        return wage / 26;
    }
}

public class ContractorEmp extends Employee
public class PartTimeEmp extends Employee
public class xxxEmp extends Employee
public class Payroll {

    public static void main(String args[]) {

        Employee allEmps[] = new Employee[100];
        // populate with Employee, SalaryEmp, // and PartTimeEmp objects (not shown)
        for (int idx=0; idx < allEmps.length; idx++) {

            BigDecimal pay =
                allEmps[idx].calculatePay();
            // not shown: rest of code

        }
    }

}
Recommended reading:

- **OBJECT-ORIENTED DESIGN IN JAVA**
  - Stephen Gilbert and Bill McCarty
  - WAITE GROUP PRESS

- Any book on UML
- Any book on OOA and D
- Any book on Design Patterns

Look at any online bookstore

- www.amazon.com
- www.chapters.com
- etc
Exception Handling
Java has "exceptions"
- objects of classes that extend Exception
- Java supplies many existing Exception classes
- You can create your own Exception classes
  ```java
  public class MyException extends Exception
  ```

Methods can throw exceptions
- by using `throw` operator with exception object
  ```java
  if (inputParameter < 100)
      throw new IOException();
  ```
- Usually done when error situation detected
  - preferred over returning special return codes
If a method throws an exception:

- It must declare which exceptions it throws using the `throws` clause on method declaration.

```java
public void MyMethod() throws MyException, OtherException {
    ...
}
```

- Many Java-supplied methods throw exceptions.

To call such a method:

- Calling code must be inside a `try` block.

```java
try {
    myObj.myMethod();
}
```
If any method call throws an exc:

The appropriate `catch` block is executed

```java
try {
    myObj.myMethod();
}
catch (MyException exc) {
    System.out.println(
        exc.getMessage());
    exc.printStackTrace();
}
catch (OtherException exc) {
    // do something
}
```

- Comes here is exception of class `MyException` or its children is thrown.
- All exceptions support these methods.
- Must try to catch all exceptions listed in `throws` clause; else no compile!
Java exceptions similar to OS/400 exceptions!

- Exception objects like OS/400 messages
- Throw like SNDPGMMSG
- Try/catch like MONMSG

**catch with a parent class is like**

- Using MONMSG with message ID ending in 00
  - Catches any exception in this family (or range)

**catch with Exception class like**

- Using MONMSG with CPF9999
  - Catches any exception!
Thanks for coming!!
<table>
<thead>
<tr>
<th>Website URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.ibm.com/software/awdtools/wds400">www.ibm.com/software/awdtools/wds400</a></td>
<td>IBM WebSphere Development Studio for iSeries</td>
</tr>
<tr>
<td><a href="http://www.ibm.com/websphere/developer">www.ibm.com/websphere/developer</a></td>
<td>WebSphere Developer Domain</td>
</tr>
<tr>
<td><a href="http://www.ibm.com/java">www.ibm.com/java</a></td>
<td>IBM Java</td>
</tr>
<tr>
<td><a href="http://www.ibm.com/iseries/java">www.ibm.com/iseries/java</a></td>
<td>IBM Java</td>
</tr>
<tr>
<td><a href="http://www.ibm.com/websphere">www.ibm.com/websphere</a></td>
<td>IBM WebSphere</td>
</tr>
<tr>
<td><a href="http://www.ibm.com/iseries/websphere">www.ibm.com/iseries/websphere</a></td>
<td>IBM iSeries WebSphere</td>
</tr>
<tr>
<td><a href="http://www.java.sun.com/products">www.java.sun.com/products</a></td>
<td>Sun Java</td>
</tr>
<tr>
<td><a href="http://www.ibm.com/iseries/infocenter.html">www.ibm.com/iseries/infocenter.html</a></td>
<td>IBM iSeries online books and help</td>
</tr>
<tr>
<td><a href="http://www.ibm.com/rochester/as400bks">www.ibm.com/rochester/as400bks</a></td>
<td>IBM Redbooks</td>
</tr>
<tr>
<td><a href="http://www.ibm.com/redbooks">www.ibm.com/redbooks</a></td>
<td>IBM Redbooks</td>
</tr>
<tr>
<td>Book, URL</td>
<td>By, ISBN</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>------------------------------------------------------------</td>
</tr>
<tr>
<td>-Java for RPG Programmers</td>
<td>Phil Coulthard, George Farr. IBM Press.</td>
</tr>
<tr>
<td>-Java for COBOL Programmers</td>
<td></td>
</tr>
<tr>
<td>-Student WorkBook for Java for RPG and COBOL Programmers</td>
<td></td>
</tr>
<tr>
<td><a href="http://www.mcpressonline.com">www.mcpressonline.com</a></td>
<td></td>
</tr>
<tr>
<td>JAVA and the AS/400</td>
<td>Daniel Darnell. ISBN 1-58304-033-1</td>
</tr>
<tr>
<td><a href="http://www.29thStreetPress.com">www.29thStreetPress.com</a></td>
<td></td>
</tr>
<tr>
<td><a href="http://www.mc-store.com/mc-store/">www.mc-store.com/mc-store/</a></td>
<td></td>
</tr>
<tr>
<td>Core Java series</td>
<td>Now a series of 3 books. Horstmann, Cornell</td>
</tr>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td></td>
</tr>
<tr>
<td>Teach Yourself Java in 21 Days</td>
<td>Rogers Cadenhead, Laura Lamay. ISBN 1575213907</td>
</tr>
<tr>
<td><a href="http://www.amazon.com">www.amazon.com</a></td>
<td></td>
</tr>
</tbody>
</table>
Trademarks & Disclaimers

© IBM Corporation 1994-2003. All rights reserved.

References in this document to IBM products or services do not imply that IBM intends to make them available in every country.
The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

<table>
<thead>
<tr>
<th>AS/400</th>
<th>IBM (logo)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS/400e</td>
<td>iSeries</td>
</tr>
<tr>
<td>e (logo) business</td>
<td>OS/400</td>
</tr>
<tr>
<td>IBM</td>
<td></td>
</tr>
</tbody>
</table>

Lotus, Freelance Graphics, and Word Pro are registered trademarks of Lotus Development Corporation and/or IBM Corporation.
Domino is a trademark of Lotus Development Corporation and/or IBM Corporation.

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.
UNIX is a registered trademark of The Open Group in the United States and other countries.
SET and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.
Other company, product and service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Information in this presentation concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by IBM. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Contact your local IBM office or IBM authorized reseller for the full text of the specific Statement of Direction.

Some information in this presentation addresses anticipated future capabilities. Such information is not intended as a definitive statement of a commitment to specific levels of performance, function or delivery schedules with respect to any future products. Such commitments are only made in IBM product announcements. The information is presented here to communicate IBM's current investment and development activities as a good faith effort to help with our customers' future planning.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Photographs shown are of engineering prototypes. Changes may be incorporated in production models.