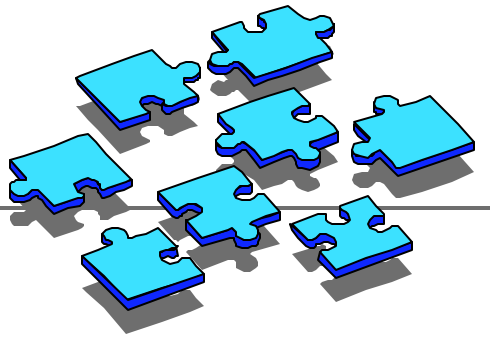
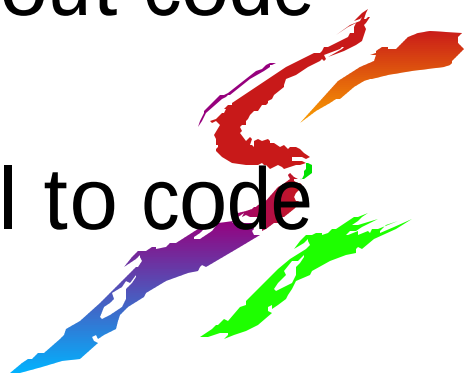
Internationalization



- Why internationalize?
- Many aspects of presentation are language (or country) specific
  - Words
  - Formatting of numbers (especially currency and time)
- We separate language specific parts from code, so the language can be easily translated without code changes.
- Language translation can be in parallel to code finalization.



Java Internationalization

- Java built from ground up for Internationalization
- Uses Unicode for all strings
- Java uses Locales to indicate the current language, country specific formatting
- Java strings and identifiers are put in properties files or *resource bundles* instead of resource files

Java Locale

- Java internationalized functions pay attention to the Locale
- A Locale is always in effect; if you do not specify, you use the default Locale.
- Locales have both a language and a location component (e.g. language=English, location=UnitedStates).
- Locales use the short-hand codes for languages defined by ISO-649 and for countries defined by ISO-3166. (e.g. en_US)

Locale Dependencies

- Language
- Numbers
- Currency
- Date (day names, order of day-month-year, gregorian, time zone)
- Time
- Coallation
- Word breaks
- Message formatting

Example Resource Bundle

```
//Generated by GUIBUILDER  "MessageQueueGUI.java"
package com.ibm.as400.opnav.MsgQueueSample1;
public class MessageQueueGUI extends java.util.ListResourceBundle {
   public Object[ ][ ] getContents() { return contents; }
   static final Object[ ][ ] contents = {
      {"ID_ADD_HELP", "%Help"},
      {"error.text.delete.msg", "Unable to delete messages on system {0}."},
      {"ID_DEL_OK", "OK"},
      {"ID_DEL_CANCEL", "Cancel"},
      {"error.title.msgbox", "AS/400 Operations Navigator"},
      {"menu.help.new", "Creates a new message."},
      {"IDD_MSGQ_ADD.Margins", "18,18,18,18,18,18"},
      {"menu.text.move", "&Move"},
      {"IDC_MSGQ_ADD_TEXT", "Enter your new message:"},
      {"IDC_MSG_DEL_MSGLIST.COLUMN_1", "Messages"},
      {"message.text.deletemsg", "{0} of {1} messages deleted."},
    :
   };
}
```

Displaying Messages

- Create and Initialize a ResourceLoader object
- Get the resource string
- Load any values for substitution
- Format the resource string
- Display the message

ResourceLoader

- setResourceName Searches path for resource
  - Searches for base_lang_country
  - Searches for base_lang
  - Searches for base
- getString returns string for message ID

Message formatting

- Substitution values needed in messages
- Order is critical for formatting program
- Changing order is critical for internationalization
- Messages have {0}, {1}, values for nth parameter (e.g. {0} of {1} copied to clipboard.)
- Result is a string with substituted values
  - (e.g. 2 of 2 copied to clipboard.)
- To display a single quote in the message next to a {, use a pair of single quotes
  - Ex. ""{0}" is not a valid AS400 short name (SNAME)."
- To display a double quote, use \"

Example Resource Loader

```
// Utility class for loading MRI from the sample resource bundle
static private ResourceLoader  m_loader = new ResourceLoader();

static
{
    // Set up to load MRI.  We do this in the static initializer for
    // the class so that we only load the resource bundle once.
    m_loader.setResourceName
            ("com.ibm.MyProject.MyResources");
}
```

Example Message Display

```
   // Load the error message text
String msg = m_loader.getString("error.text.send.msg");

// Build array of substitution text; here, the system name, could have multiples...
Object[ ] args = { m_queue.getSystem().getSystemName() };

// Format the error message
String usrMsg = MessageFormat.format(msg, args);

// Display it to the user
MessageBoxDialog.showMessageDialog(m_owner,
                                   usrMsg,
                                   m_loader.getString("error.title.msgbox"),
                                   JOptionPane.WARNING_MESSAGE);
```

Java Packages

- Java has potential name-space collisions
- Solved by packages, which qualify a class as a member of a code set
- Convention established to assure package uniqueness
- Some similarities to naming conventions of internet
- Package naming
  - ***organization-type.organization.sub-organization.etc***
  - *IBM standards require com.ibm.platform. . . for all our packages*
  - Example: com.ibm.msys.myproject
  - You may choose your project name

Java Packages (con't)

- The hierarchical appearance of package names has nothing to do with inheritance
- In this way any class can be explicitly called out
  - package.classname
  - Ex. com.ibm.myproject.myclass
- The package name is represented on the hard drive as a directory structure
  - Ex. com\ibm\myproject\ which contains myclass.class

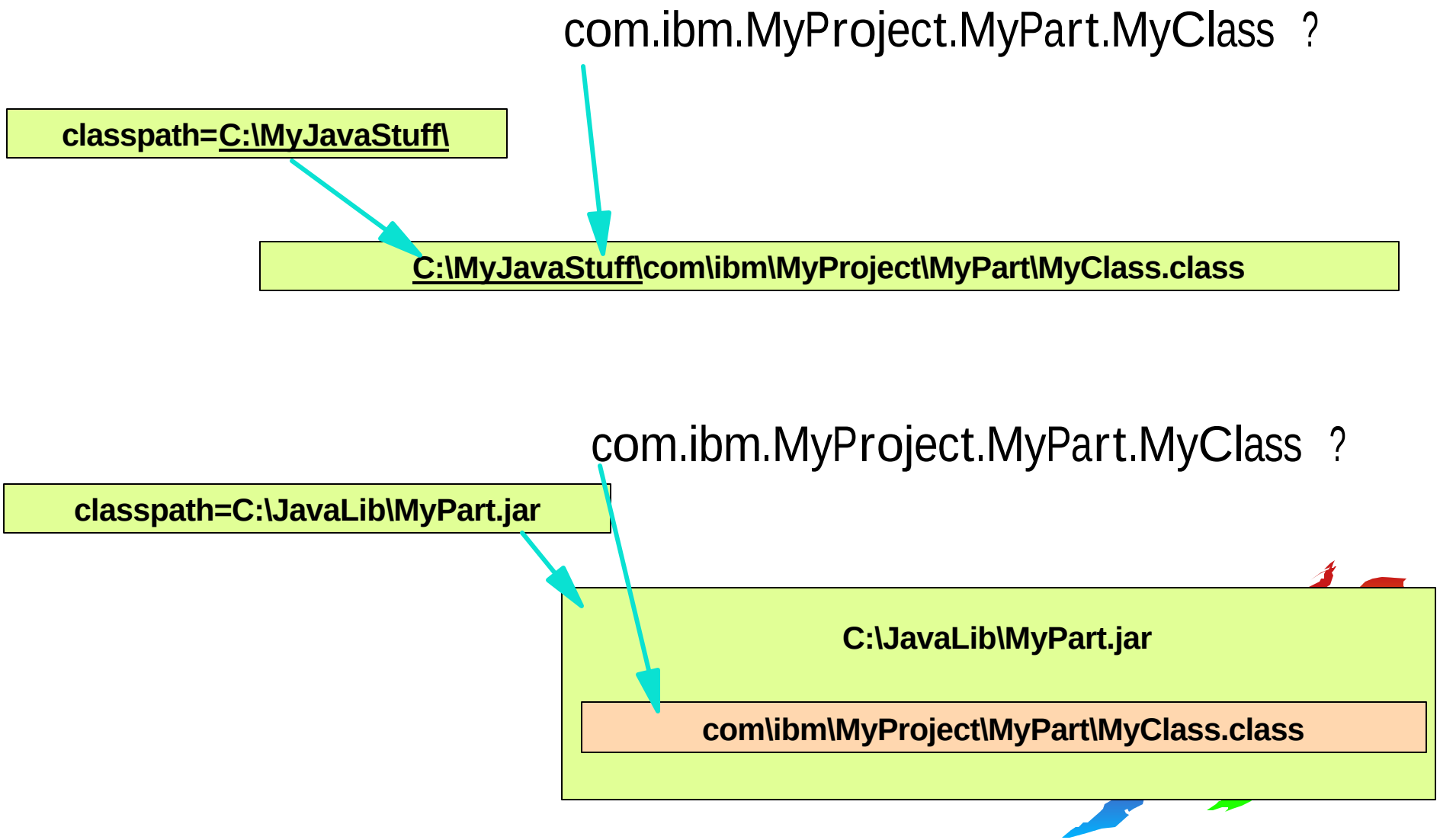
Java Jars

- "jar" stands for Java ARchive
- A collection of classes and resources in a single file
- Has internal directory structure
  - describes the package
- Similar to a zip file

Java Classpath

- Tell operating system the locations of java runtime and classes
- Each entry in the classpath represents
  - Directory containing classes
  - Directory of the base of a package
  - Full path and name of a jar file

Classpath Diagram



Package example

- Java finds your class by combining the location + package name
- To find com.ibm.myproject.myclass you must have a directory structure set up com\ibm\myproject\myclass.class
- The root of your package exists somewhere, and the classpath points at the root.
  - Actual file c:\java\development\com\ibm\ \myproject\myclass.class
  - Class path includes c:\java\development\
  - Package references to com.ibm.myproject.myclass

Package example (con't)

- Jar file myjar.jar is created for package com.ibm.myproject
  - Jar file is built from the actual files in c:\java\development\com\ibm\myproject
- The root of your package exists at the "base" of the jar file
  - Actual file c:\java\lib\myjar.jar
  - Class path includes jar name c:\java\lib\myjar.jar
  - Package references to com.ibm.myproject.myclass