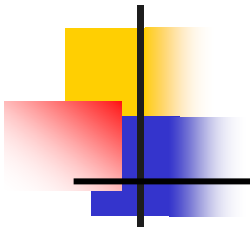
Writing Java Applications for Mobile Information Devices

James E. Osbourn

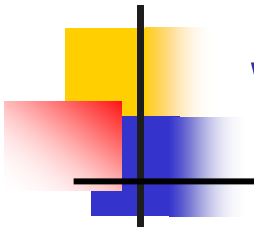
RiverPoint Group LLC

josbourn@riverpoint.com



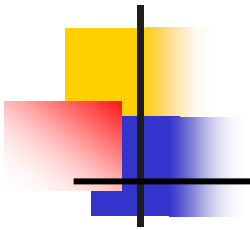
Who Are You?

- You ...
  - already have the fundamentals of object-oriented programming and the Java language.
  - are excited about mobile information devices (like phones, PDAs, pagers, *etc*.)
  - are willing to live on the cutting edge of new platform rollout.
  - are ready to get started writing wireless Java applications.



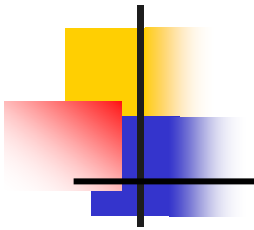
Who Am I?

- I …
  - consult IT teams on organization, architecture, methodology, and governance.
  - Have experience as a programmer and architect.
  - Like to keep my hands dirty with the new technologies.
  - Like people, good food, and sharing ideas.



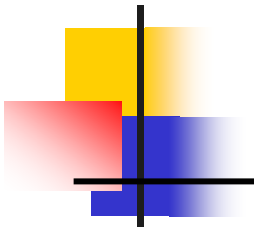
Session's Contents Include:

- Introduction
  - What is J2ME?
  - Focus on MIDP
- Midlets
  - Tools
  - Code, Compile, Preverify
  - Suites, Jars, Descriptors, Permissions & Protection Domains
  - Midlet class and AMS
- Create your interface
  - Abstraction & Discovery
  - Event Handling & Commands
  - Ticker, TextBox, Alert
  - List & Form
  - Custom Item
- Persistence
  - RecordStore & Records



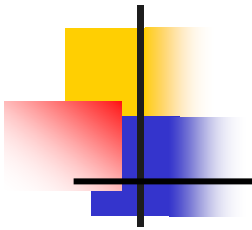
Introduction

- What is J2ME?
  - Java 2 Platform, Micro Edition, was created to support the market for small devices.
  - This is a high growth development arena that holds potential for individuals and corporations.
  - Java offers the advantages of portability and security needed to make it a winner.
  - Applications for Mobile Phones, PDAs, Pagers, and (not yet created) hand-helds are our focus.



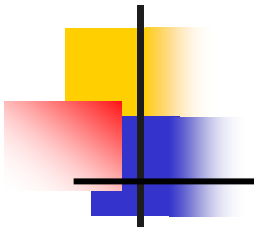
Introduction

- J2ME is broader than this class can cover. It includes embedded systems for appliances and set-top devices that are nearly PCs.
- The specification calls for *configurations*, *profiles*, and *optional APIs*. These work together to form a *stack* for a given device.
- Sun uses the Java Community Process and Java Specification Request (JSR) 185 to guide standardization of J2ME stacks.



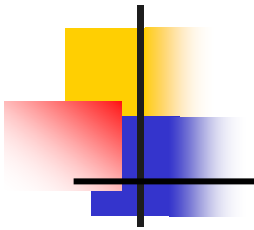
Introduction

- J2ME has ***configurations*** designed for memory constraints and processing power of specific devices, including the JVM and subset of J2SE APIs and other APIs to be used.
- The two configurations of interest are:
  - Connected, Limited Device Configuration (CLDC)
  - Connected Device Configuration (CDC)



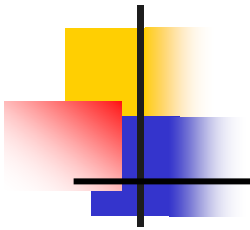
Introduction

- CDC has a minimum of 512KB of ROM, 256KB of RAM, and a network connection.
- Examples include TV set-top boxes, automobile navigation systems, and high-end PDAs/Pocket PCs.
- A full JVM must be supported in this configuration.



Introduction

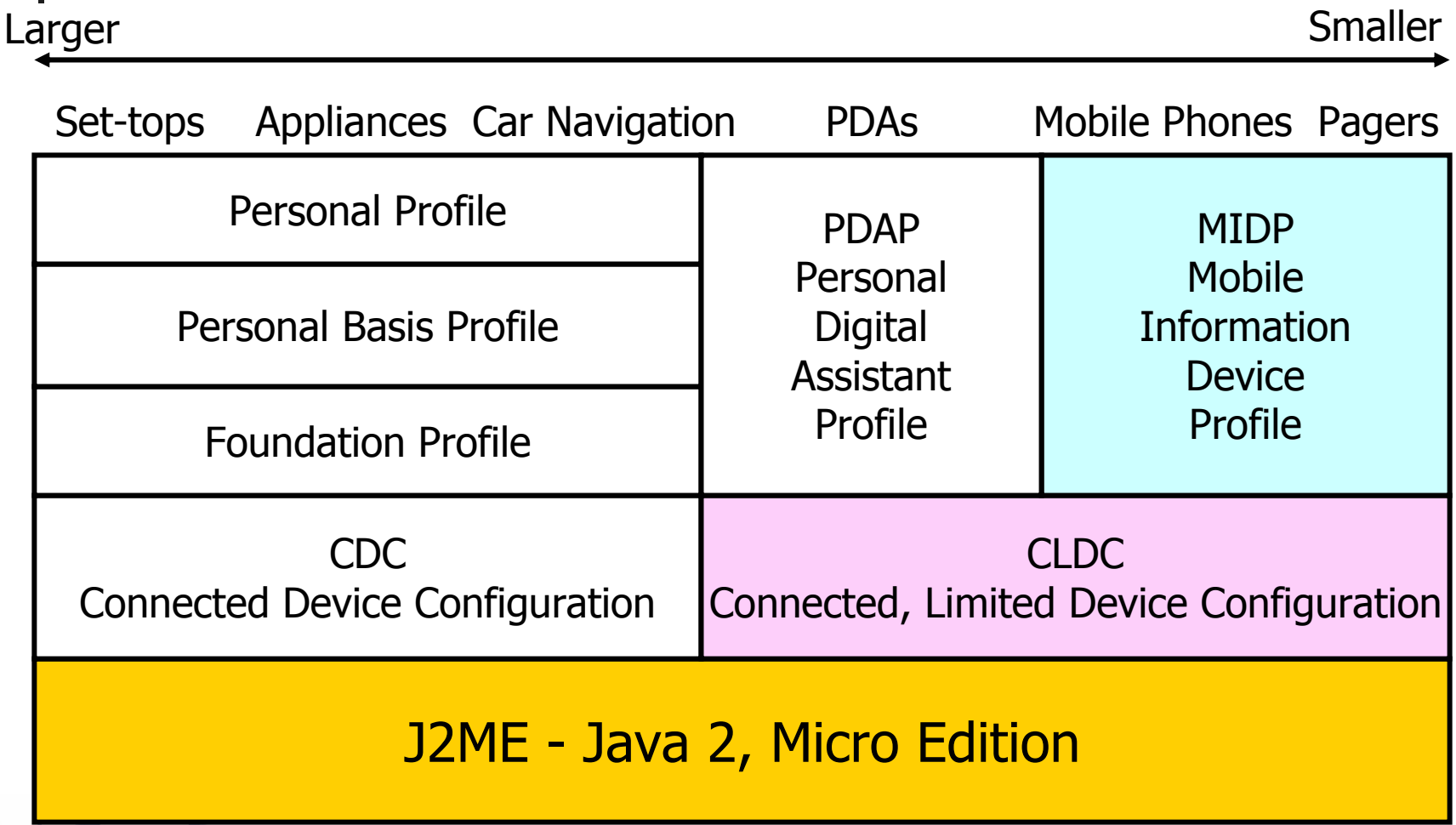
- CLDC mandates only 160-512KB of memory. The *limited* means probably an intermittent and slow connection, a small screen, and not much memory. Java Kilobyte Virtual Machine (KVM) is specified.
  - Native methods must be built in.
  - Bytecode verification is a subset of the JVM standard.

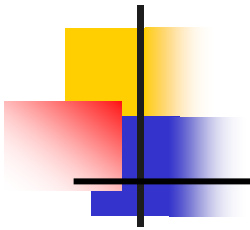


Introduction

- ***Profiles*** add specifications and APIs for certain devices to the configuration.
- Examples include:
  - Foundation Profile, Personal Basis Profile, & Personal profile for CDC
  - Personal Digital Assistant Profile (PDAP) & Mobile Information Device Profile (MIDP) for CLDC
  - We will focus on MIDP in this class

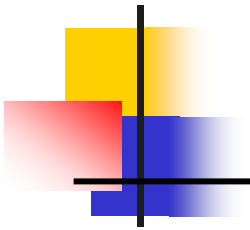
Introduction





Introduction

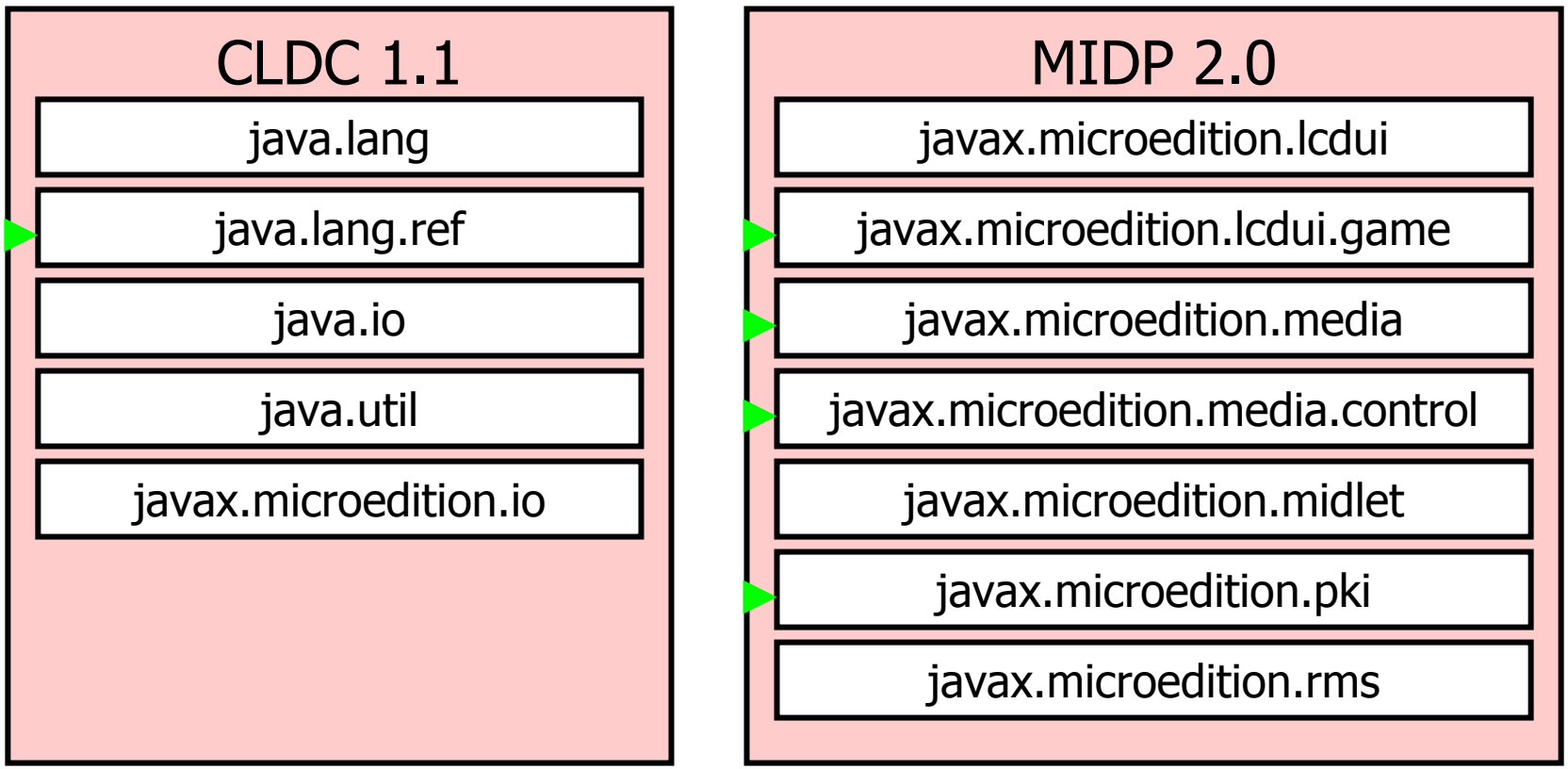
- Characteristics of MIDP
  - 128KB of dedicated non-volatile MIDP memory
  - 32KB of volatile run-time memory
  - 8KB of non-volatile memory for persistent data
  - Screen of at least 96 x 54 pixels
  - Input by either touchscreen, keyboard or keypad
  - Two-way intermittent network connection
- Mobile Phones, high-end Pagers, some PDAs

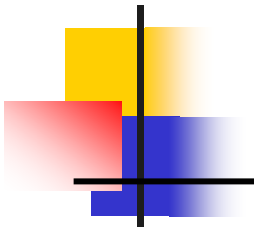


Introduction

- Current MIDP information at
  - http://java.sun.com/products/midp
- Current versions are:
  - CLDC 1.1 which adds floating point support
  - MIDP 2.0 which is backward compatible with 1.0 and adds numerous enhancements which are covered in my other class, "Advanced Features in Java Applications for Mobile Information Devices".

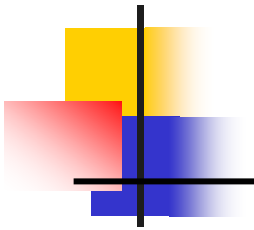
Introduction



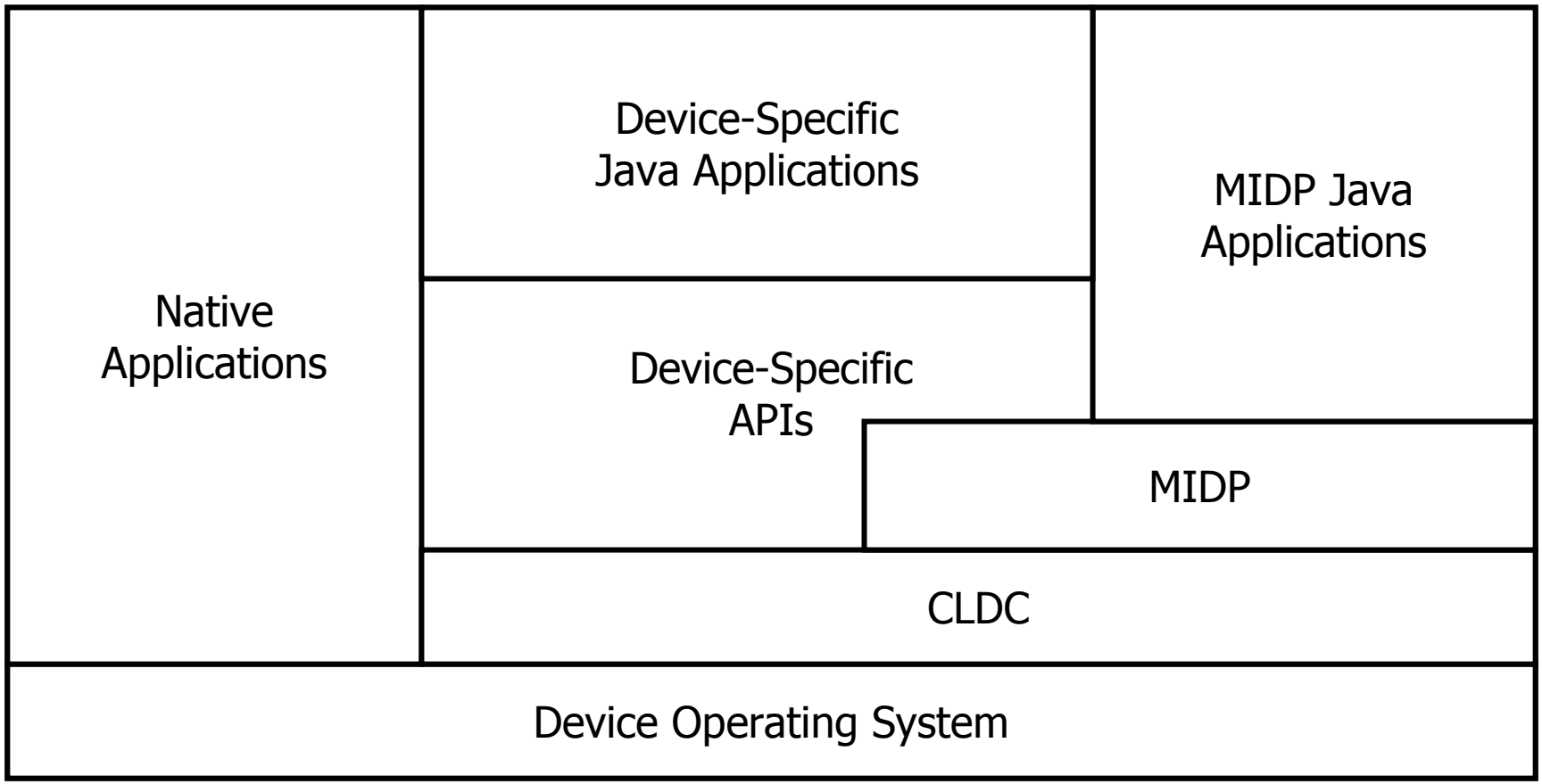


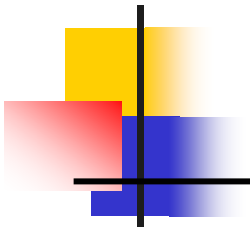
Introduction

- MIDP fits into the larger context of the device vendor's implementation.
- Most devices will offer device-specific APIs for native applications and Java applications, too.
- There are trade-offs to recognize in your choice to use device-specific APIs.



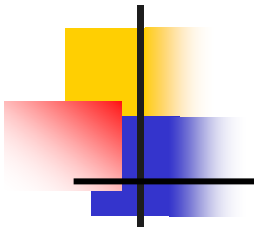
Introduction





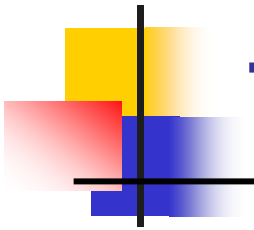
Introduction

- MIDP offers:
  - Portability "Write Once, Run Anywhere"
    - Really?
  - Maintainability
  - Security
    - Really?
    - Bytecode verification may present challenges, but the MIDP application is confined to resources owned by the KVM.



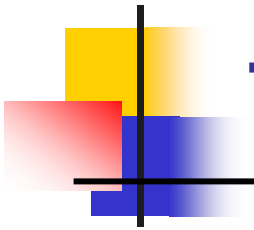
MIDlets

- Applets, then servlets, now MIDlets
- Much of what you know from J2SE applies.
- The development cycle:
  - Write Code
  - Compile
  - Preverify
  - Package
  - Test or Deploy



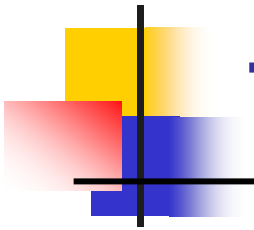
Tools

- Sun offerings include:
- Sun's reference implementation
  - http://java.sun.com/products/midp
- Sun's J2ME Wireless toolkit (J2MEWTK)
  - http://java.sun.com/products/j2mewtoolkit/download-2_0.html
- Sun ONE Studio 5, Standard Edition, features support for J2ME development, too
  - http://wwws.sun.com/software/sundev/jde/index.html



Tools

- Borland JBuilder MobileSet
  - http://www.borland.com/products/downloads/download_jbuilder.html
- IBM Websphere Studio Device Developer
  - http://www-3.ibm.com/software/wireless/wsdd/
- Metrowerks CodeWarrior Wireless Studio
  - http://www.metrowerks.com/MW/Develop/Wireless/Wireless_Studio/
- RIM BlackBerry Java Development Environment
  - http://www.blackberry.net/developers/na/java/start/download.shtml

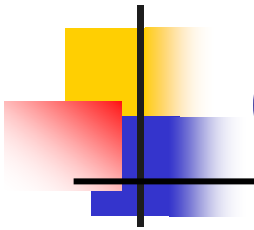


Tools

- Sprint J2ME APIs & Wireless Toolkit
  - http://developer.sprintpcs.com/adp/resources.do?redirect=wirelessjava&bgcolor=3
- Nokia Developer's Suite for J2ME™, Version 2.0
  - http://www.forum.nokia.com/nds_for_j2me.html

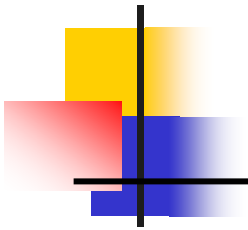
Write Your Code

- You can use your favorite IDE or Editor to write your .java files
- Now look up here. I'll show you something.



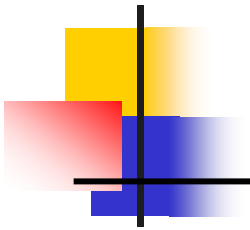
Cross-compiling

- Compile on one and run on another platform
- Using the J2ME Wireless Toolkit takes care of organizing your projects.



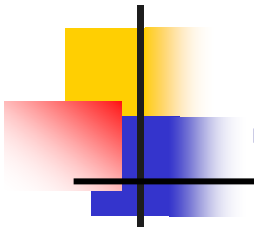
Using the Command Line

- If you want to use the reference implementation and javac,
  - -bootclasspath parameter is needed to specify the fundamental APIs against which you will be compiling
  - javac –bootclasspath \midp\classes sample1.java
- Complex builds are probably better done with ant. (Come to the advanced class.)



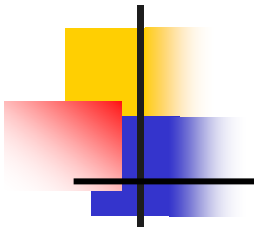
Preverify

- CLDC requires a split of bytecode verification to accommodate the small amount of memory.
- Off-device preverify step performs certain checks and reformats the class file.
- On-device verification will reject a class file that has not been preverified.
- Don’t forget to preverify inner classes, too.



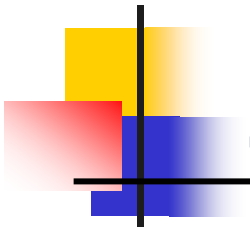
Security Concerns

- Devices should only download code from trusted sources, because some bytecode verification is performed off the device.
- Malicious code could appear to be preverified, and it would load and run.
- MIDlet Suites and deployment techniques can mitigate some of these risks.



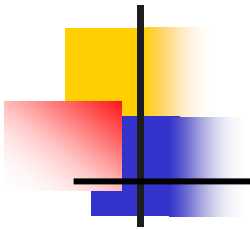
Midlet Suites

- You must use jar to package your application
- J2MEWTK will create a MIDlet Suite for you.
- Extra information in the manifest file
- Also, an **application descriptor** must be created as a separate text file with .jad extension.
- Understand the details, but let the J2MEWTK help you get started.



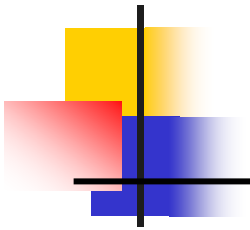
Jars

- Class and resource files for your application just like any jar.
- Manifest file (META-INF/MANIFEST.MF) with extra information needed at run-time by the Application Management System (AMS).
- Let's look at these extras.



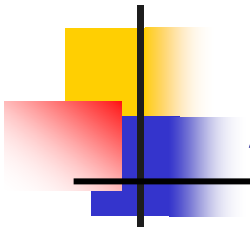
Manifest Attributes

- Required:
  - MIDlet-Name: <name of the entire suite>
  - MIDlet-Version: 1.0
  - MIDlet-Vendor: RiverPoint Group
  - MIDlet-Configuration: CLDC-1.0
  - MIDlet-Profile: MIDP-1.0
  - MIDlet-*n*: <display_name>,<icon>,<class>



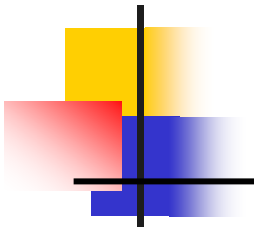
Manifest Attributes

- Optional:
  - MIDlet-Description: <describes the suite>
  - MIDlet-Icon: <icon for the suite>
  - MIDlet-Info-URL: <more on-line information>
  - MIDlet-Data-Size: <persistent data bytes needed>
  - MIDlet-Permissions: <comma-separated list>
  - MIDlet-Permission-Opt: <non-critical list>
  - MIDlet-Extensions: <optional API list>



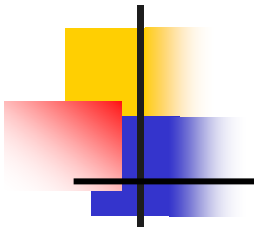
Application Descriptor

- This file is retrieved first and helps the device decide whether to load a MIDlet suite.
- Same information as manifest plus:
  - MIDlet-Jar_URL: <location of jar>
  - MIDlet-Jar_Size: <size in bytes>
- Some devices may be quite strict, so using the J2MEWTK assures well-formed attributes



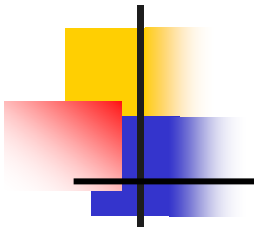
Properties, Too

- Application descriptor file is a good place to stick properties that might change.
- Store them in the .jad files, like this:
  - Sample1.url: http://www.riverpoint.com/
- Retrieve them with:
  - String url = getAppProperty("Sample1.url");
- Useful to distribute the same jar with separate descriptor properties pointing to regional servers.



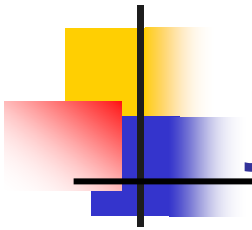
Permissions

- MIDP 2.0 brings a security schema. MIDlets must have permission to perform sensitive operations.
- So far, only network operations are protected by permissions associated with the APIs. Optional APIs are free to define permissions, too.
  - Public final int checkPermission(String permission)



Protection Domains

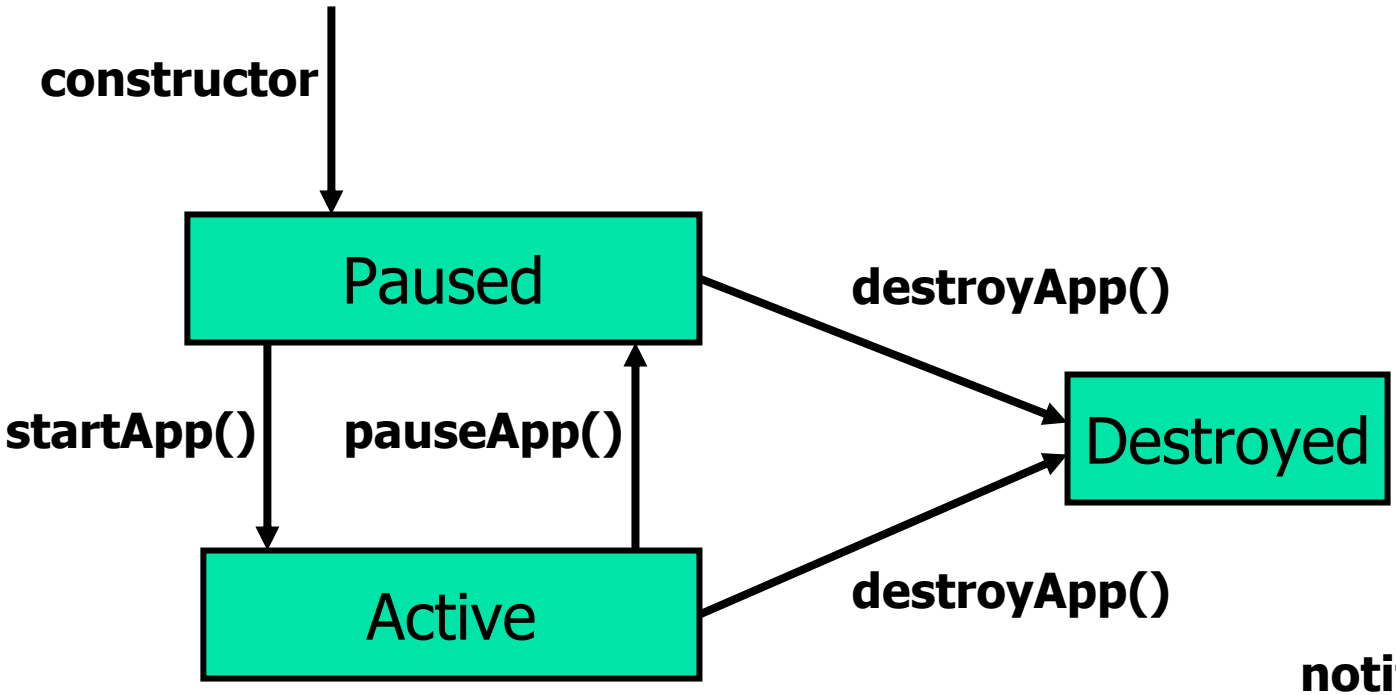
- Device implementations can provide sets of permissions to a defined protection domain, and then let a MIDlet suite enter if it meets the entry rules.
  - MIDlet-Permissions: javax.microedition.io.Connector.http
- This attribute tells the AMS that HTTP traffic is critical.



javax.microedition.midlet.MIDlet

- The application manager controls the lifecycle state of the MIDlet class.
  - Loaded and constructed, then *paused.*
  - AMS calls startApp(), then *active.*
  - AMS suspends it with pauseApp() or MIDlet asks to be paused with notifyPaused().
  - AMS kills it with destroyApp() or MIDlet asks by calling notifyDestroyed().
  - MIDlet can ask resumeRequest().

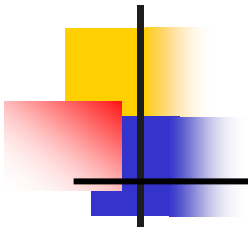
Lifecycle of a MIDlet



notifyDestroyed()

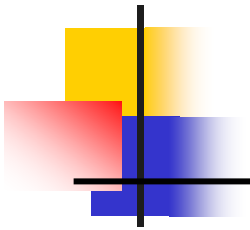
notifyPaused()

resumeRequest()



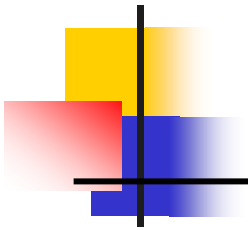
Other Important Restrictions

- Application Management System (AMS) handles class loading. CLDC does not allow you to define your own classloaders
- No finalize() methods, so explicitly clean up yourself.
- No Reflection API, so no RMI.
- No native methods supported by CLDC.
- Runtime & System classes are greatly reduced.



Old Tricks Are Still Very Handy

- Even though J2ME and MIDP present new things to learn, most of your familiar J2SE APIs in java.lang, java.io, and java.util look and act similarly.
- Some tasks get easier, and the restrictions of the platform force one to think more simply and code more efficiently.

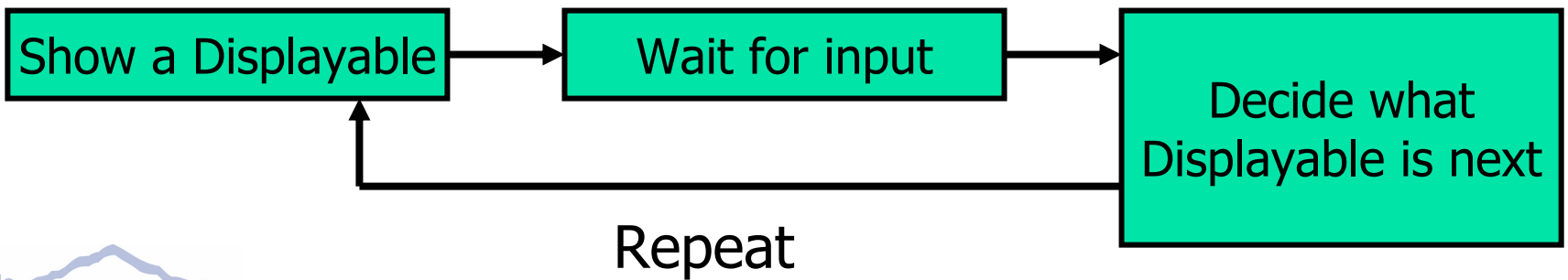


Create Your Interface

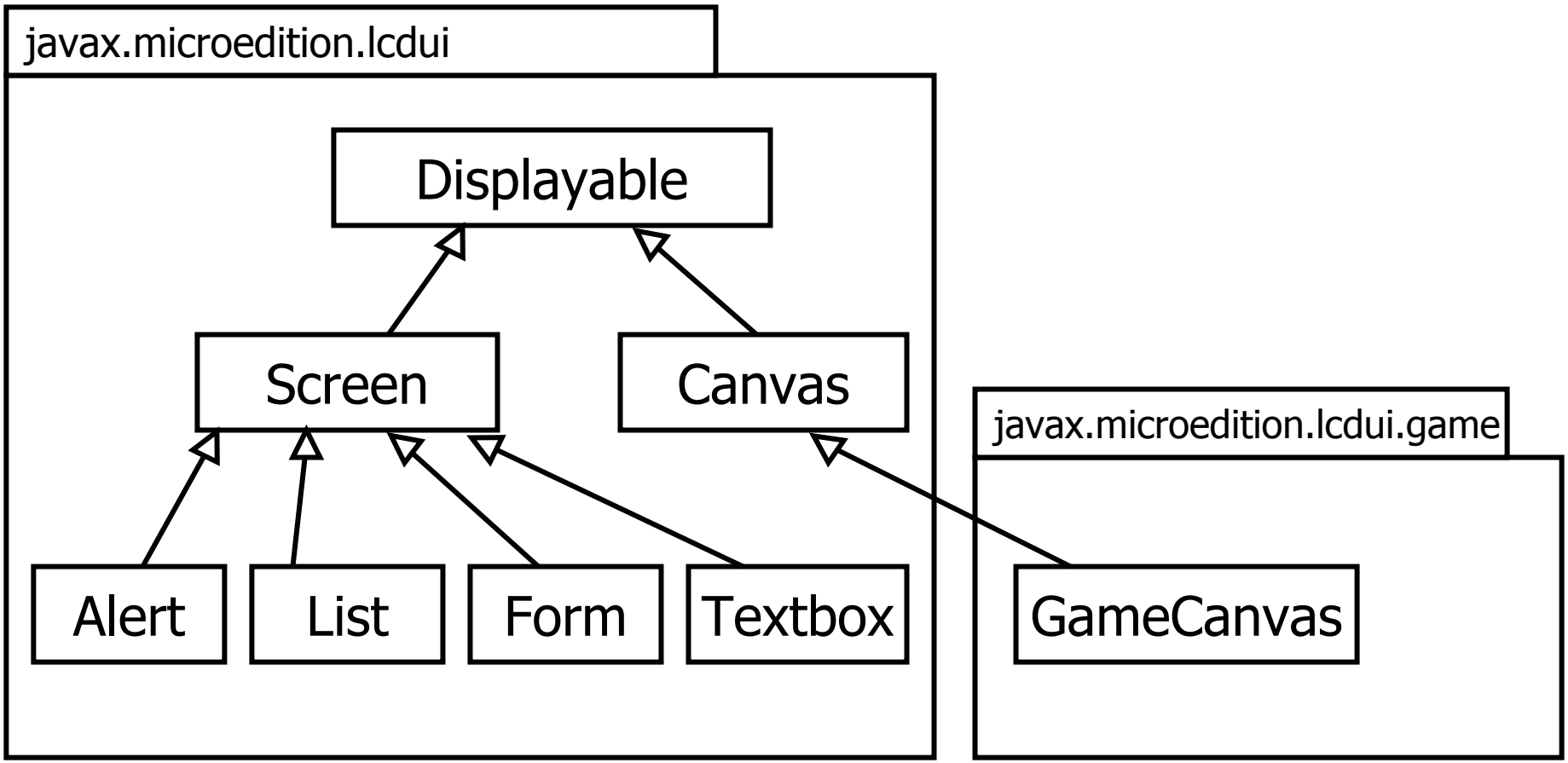
- Wide variety of devices presents a challenge to WORA.
- MIDP strategy includes:
  - **Abstraction**: specify the interface needs and let the implementation create the screen
  - **Discovery**: learn about the device at runtime and tailor UI programmatically
- Prefer Abstraction because it creates the smaller footprint.

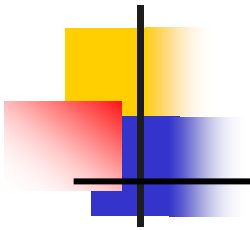
Overview

- The device display is represented by an instance of Display. Use the factory method, getDisplay(), to access it.
- Keeps track of the current instance of Displayable. MIDlets change this by calling setCurrent().



Displayable Family

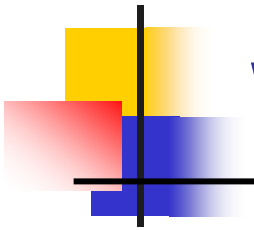




Get Your Display Reference

- MIDlet application can get a reference to the device display in your startApp() method.
  - Display's static method getDisplay(MIDlet m)

```
Public void startApp() {
        Display d = Display.getDisplay(this);
        // …. More startup code
}
```



Working a Display

- Pass the Displayable you create with:

  setCurrent(Displayable nextDisplayable)

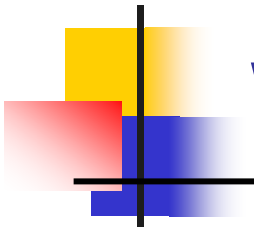
  setCurrent(Alert alert, Displayable next)

- Get a reference to the current one with:

  getCurrent()

  - Note: this may not be actually shown on the device at the time it is returned.

- Query the Display to get its capabilities.



Working a Display

int isColor()

int numColors()

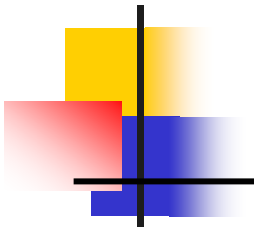
int numAlphaLevels()

int getBorderStyle(boolean highlighted)

int getColor(int colorspecifier)

boolean flashBacklight(int duration)

boolean vibrate(int duration)



Displayable Commands

- Displayable offers the ability to add commands which the device will take care of showing. Each with a title, type and priority.

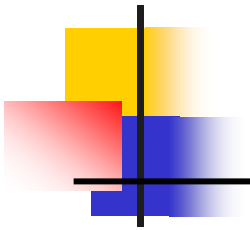
  addCommand(Command cmd)

  removeCommand(Command cmd)

- Create a Command with:

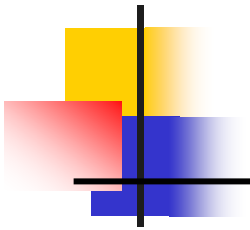
  Command cmd = new Command("OK", Command.OK, 0);

  Command cmd = new Command("Sync", Command.SCREEN, 0);



Command Types

- Constants defined:
  - OK — Confirm a selection
  - CANCEL — Cancel pending change
  - BACK — Move to previous screen
  - STOP — Stop current operation
  - HELP — Show instructions
  - SCREEN — General for application use



CommandListeners

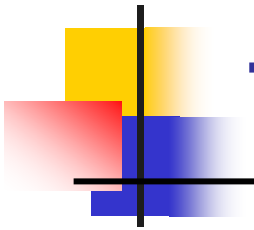
- Displayable can have one listener which it will call whenever a Command is invoked.

  setListener(CommandListener l)

- All a CommandListener needs is a single method.

  commandAction(Command c, Displayable s)

- Warning: Do not tie up the system's event-handling thread in your commandAction.

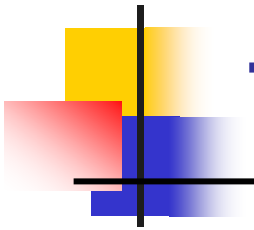


Titles and Tickers

- MIDP 1.0 Screen had methods for titles and Tickers. MIDP 2.0 moved them to Displayable, leaving Screen with no methods.

```
public void setTitle(String newTitle)
public String getTitle()
public void setTicker(Ticker newTicker)
public Ticker getTicker()
```

- Ticker is wrapper for String making it scroll.

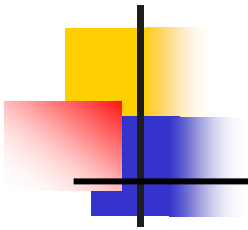


TextBox

- Simple Screen that allows user to enter a string.
- Specify Screen title, initial text, and max text size. Use constraints and flags to limit input.
  - ANY, NUMERIC, DECIMAL, PHONENUMBER, EMAILADDR, URL
  - PASSWORD, UNEDITABLE, SENSITIVE, NON_PREDICTIVE, INITIAL_CAPS_WORD, INITIAL_CAPS_SENTENCE

Demo

- Let's run a sample application to show our TextBox.



Alerts

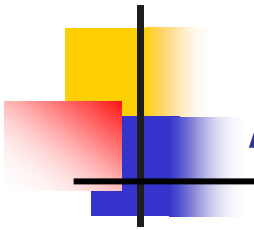
- Alert is a message shown to the user. Two kinds: timed and modal

  public Alert()

  public Alert(String title, String text, Image alertImage, AlertType alertType)

- You could create a timed Alert with:

  Alert alert = new Alert("Hello", "Hi, Wayne..", null, null);



Alerts

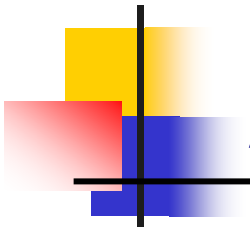
- Default timeout can be overridden with:

```
alert.setTimeout(5000);  // 5 seconds
```

- For a modal Alert use:

```
alert.setTimeout(Alert.FOREVER);
```

- Device implementation will supply a way to dismiss a modal alert. Make your own approach with addCommand(), and the system default will be removed.

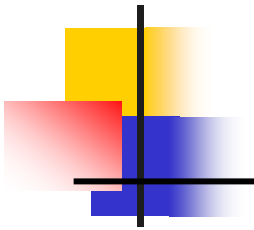


Alerts

- Advances to the next Displayable when the Alert is dismissed. Specify next screen by passing it with the Alert on setCurrent().
- Device may use alert type to decide what kind of sound to play when the alert is shown. Type include:
  - ALARM, CONFIRMATION, ERROR, INFO, and WARNING
- Add a progress indicator with setIndicator().

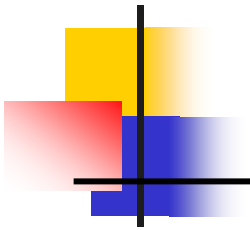
Demo

- Let's go add some alerts to our sample and see the effect.



Lists

- Allow the user to select elements by image and/or text. Multiple or exclusive choices.
- Navigate using the up/down arrows, then press the device select button. Confirm by a Command that your MIDlet furnishes.
- Implicit list is an exclusive list that fires a confirmation event when selection is made. User has no chance to change mind.



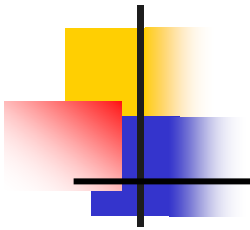
Implicit List Selection Event

- commandAction() is invoked and passed a special value.

```
public static final Command SELECT_COMMAND
```

- Test like this:

```
public void commandAction(Command c, Displayable s) {
  if (c == nextCommand)
  // …
  else if (c == List.SELECT_COMMAND)
  // …
}
```



Create a List

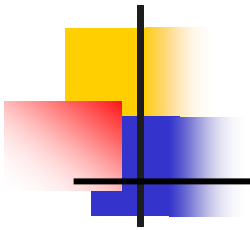
- Specify title and list type, pass the element names and images if you have them.

  public List(String title, int type)

  Public List(String title, int type,

  String[] stringElements, Image[] imageElements)

- If there are more elements than can be displayed on one screen, the device implementation will handle scrolling.



Edit a List

- First element is at index 0.

  public void set(int elementNum, String stringPart, Image imagePart)

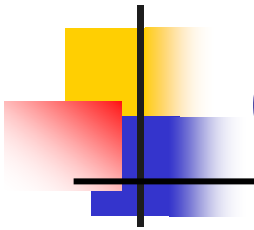
  public void insert(int elementNum, String stringPart, Image imagePart)

  public int append(int elementNum, String stringPart, Image imagePart)

  public void delete(int elementNum)

  public String getString(int elementNum)

  public Image getImage(int elementNum)



Getting List Selections

- You can examine whether an element is selected, with:

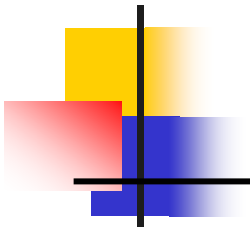
  public boolean isSelected(int index)

- EXPLICIT & IMPLICIT lists return the selected element, with:

  public int getSelectedIndex()

- For MULTIPLE lists, try:

  Public int getSelectedFlags(boolean[] selectedArray_return)

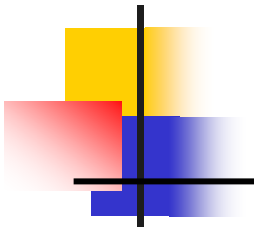


Set List Selections

- If you want to programmatically set list selection, use:

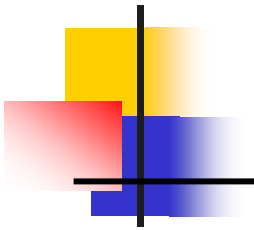
  public void setSelectedIndex(int index, boolean selected)

  public void setSelectedFlags(boolean[] selectedArray)



New Control over Lists

- MIDP 2.0 offers some control over how lists are shown. setFitPolicy() offers Choices: TEXT_WRAP_ON, TEXT_WRAP_OFF, TEXT_WRAP_DEFAULT.
- setFont() to suggest a font.
- These are hints to the device. Its implementation will decide whether fit and font requests can be honored.



Images

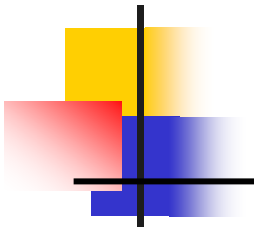
- MIDP must be able to load PNG images.
- Load from PNG data:

  public static Image createImage(String name)

  public static Image createImage(byte[] imagedata, int imageoffset, int imagelength)

  - public static Image createImage(Inputstream stream)

  public static Image createImage(int width, int height)



Images

public static Image createImage(Image image)

- public static Image createImage(Image image, int x, int y, int width, int height, int transform)

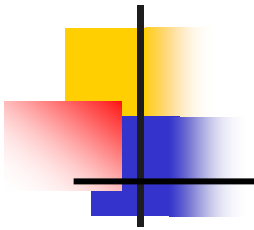
- All images for Alert, ChoiceGroup, ImageItem or List must be immutable.
- Get optimal image sizes for each with:

public int getBestImageHeight(int imageType)

public int getBestImageWidth(int imageType)

Demo

- Let's see a sample program with various kinds of lists.



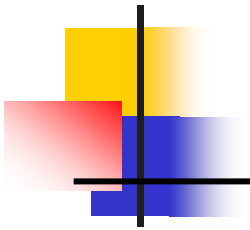
Forms

- Put together UI controls, called items, and gather different types of data on one scrollable screen.

  public Form(String title)

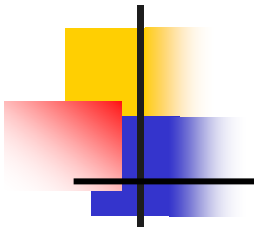
  public Form(String title, Item[] items)

- Keep it simple and clean. It's easy to err with an overly complex form.



Showing Your Form

- Naturally, you will use setCurrent() to show your form.
- MIDP 2.0 offers setCurrentItem() which makes the form containing the item visible, scrolls the form to show the item, and gives it the input focus.



Editing Your Form

- Items are added to the Form in order with append methods:

  public int append(Item item)

  public int append(String str)

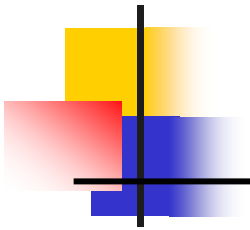
  public int append(Image image)

- Each item has an index.

  public int set(int index, Item item)

  public void insert(int index, Item item)

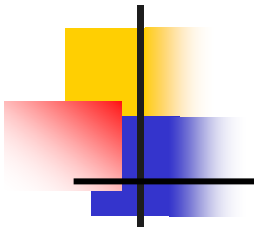
  public void delete(int index)



Form Layout

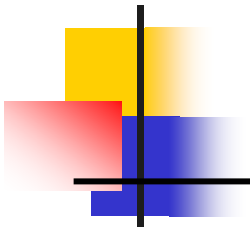
- Form tries to lay out items left to right in rows, then stack rows top to bottom.
- Items have labels and commands just like Displayables, which are shown when the item is selected. Handling is similar to Displayable commands. Use ItemCommandListener interface's single method:

```
public void commandAction(Command c, Item
    item)
```



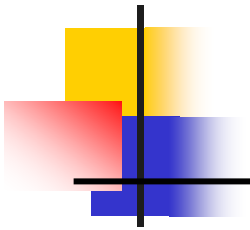
Form Layout Control

- Use an Item's minimum width & height to help make layout decisions.
- Let the implementation decide most of the time. But set the preferred size for items that you want a certain way.
- Setting the preferred size is no guarantee from one device to another.
- Try out the MIDP 2.0 layout directives.



Layout Directives

- For setLayout()
  - LAYOUT_2
  - LAYOUT_LEFT
  - LAYOUT_RIGHT
  - LAYOUT_CENTER
  - LAYOUT_TOP
  - LAYOUT_VCENTER
  - LAYOUT_BOTTOM
- Shrink to minimum or expand to space available:
  - LAYOUT_SHRINK
  - LAYOUT_EXPAND
  - LAYOUT_VSHRINK
  - LAYOUT_VEXPAND
- New line directives:
  - LAYOUT_NEWLINE_AFTER
  - LAYOUT_NEWLINE_BEFORE

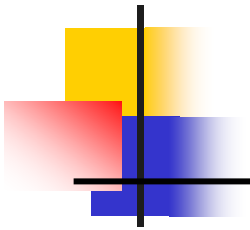


StringItem

- Simple label and value pair for display to user.

```
Form form = new Form("MyForm");
StringItem sI = new StringItem("Name: ", "James");
Form.append(sI);
```

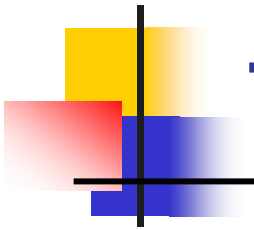
- Make it simple, use Form's append(String)
- Appearance modes: PLAIN, HYPERLINK, BUTTON change the look, but you have to handle the item commands.



Spacer

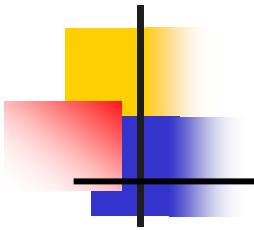
- New in MIDP 2.0, Spacer allows empty space on a Form.

```
public Spacer(minWidth, minHeight)
```



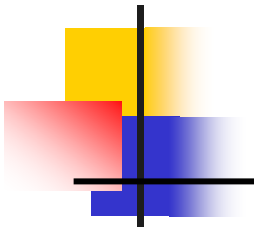
TextField

- Use for an editable input field.
- Limit with constants and flags:
  - ANY, NUMERIC, DECIMAL, PHONENUMBER, EMAILADDR, URL
  - PASSWORD, SENSITIVE, UNEDITABLE, NON_PREDICTIVE, INITIAL_CAPS_WORD, INITIAL_CAPS_SENTENCE
- Handling is pretty much same as TextBox.



ImageItem

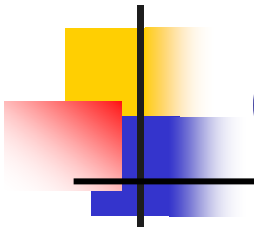
- Show an image on your Form with associated label, layout request, and alternate text in case the device cannot show it.
- Similar handling to StringItem, including appearance modes.



DateField

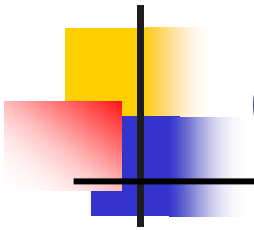
- Very handy way to let the device implementation figure out how the user can enter a date or time. Editor for java.util.Date
- Just need a label and mode: DATE, TIME, or DATE_TIME.

```
public DateField(String label, int mode)
public DateField(String label, int mode, Timezone
    timezone)
```



ChoiceGroup

- Like List, ChoiceGroup implements the Choice interface. Program it in just the same fashion as List.
- Since it is an Item on a Form, not a whole Screen like List, IMPLICIT is not allowed, and POPUP is added to emulate a combo box or drop-down menu.

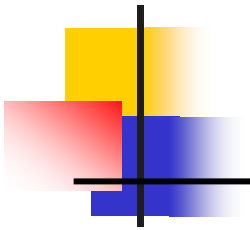


Gauge

- Represents some integer value and lets the device figure out how to display it.

  Public Gauge(String label, boolean interactive, int maxValue, int initialValue)

- Use getValue() and setValue() to update an interactive gauge. Three noninteractive gauges for use as progress indicators.



Item Change Fires Events

- Register ItemStateListener with the Form:

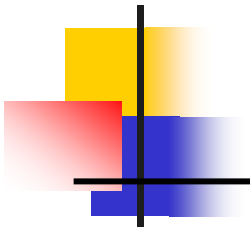
```
public void setItemStateListener(ItemStateListerner
    iListener)
```

- Write your method to handle the event:

```
public void itemStateChanged(Item item) {
  if (item == myGauge)
  // ....  Do something
}
```

Demo

- Let's go build some of these Items on our Form and see how they behave.

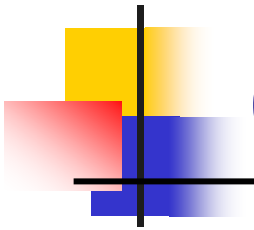


Custom Items

- New MIDP 2.0 CustomItem has only 5 abstract methods for you to define:

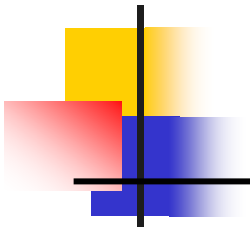
```
protected int getPrefContentWidth(int height)
protected int getPrefContentHeight(int width)
protected int getMinContentWidth()
protected int getMinContentHeight()

protected void paint(Graphics g, int w, int h)
```



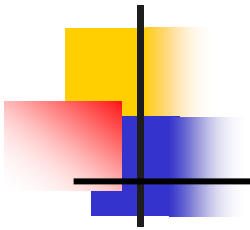
Graphics

- In your CustomItem, the Graphics class can be used to draw text, shapes, lines and images in the content area.
- It will be covered in detail in my next class, "Advanced Features in Java Applications for Mobile Information Devices".
- So, naturally, you'll want to attend that.



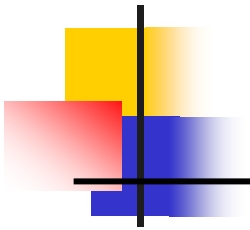
CustomItem Painting

- CustomItem's paint() is a callback method used by the device implementation. You define it and it is called when the time is right.
- Request a refresh with repaint(), and the paint() callback will again be called.
- Use Display's getColor() and getFont() to understand the device's look & feel.



CustomItem Painting

- When the custom item appears, showNotify() is called, and hideNotify() when it is scrolled out of sight.
- If some other change in the Form forces a size change in your custom item, sizeChanged().
- You can force a re-layout with invalidate().



Handling CustomItem events

- Override these methods:

  protected void keyPressed(int keyCode)

  protected void keyReleased(int keyCode)

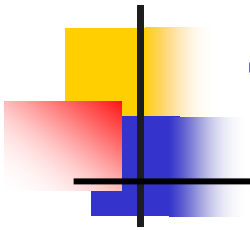
  protected void keyRepeated(int keyCode)

  protected void pointerPressed(int x, int y)

  protected void pointerReleased(int x, int y)

  protected void pointerDragged(int x, int y)

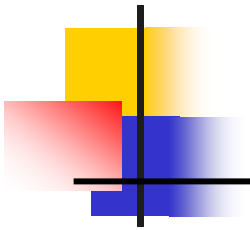
- Use getInteractionModes() to find out which ones the device can generate.



Traversing Items

- Moving from item to item is much as one expects, but items with Choices internally, like ChoiceGroup, may be confusing.

```
protected boolean traverse(int dir, int
    viewportWidth, int viewportHeight, int[]
    visRect_inout);
protected void traverseOut();
```

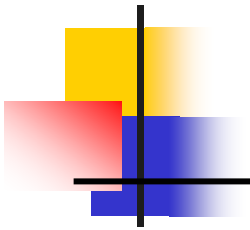


Internal Traversal

- Dir indicates traversal direction:

  Canvas.UP, Canvas.DOWN, Canvas.LEFT, Canvas.RIGHT, CustomItem.NONE

- viewportWidth and viewportHeight basically equal the content area of the Form.
- visRect_inout is an integer array where you give the bounds of the item's content area when you call traverse() and get back the bounds of the selected choice in the item.



Understand the Device

- Each device may have different capabilities, so use getInteractionModes().

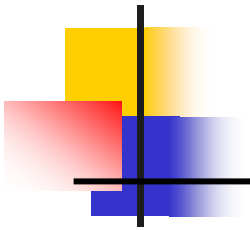
  CustomItem.TRAVERSE_HORIZONTAL,

  CustomItem.TRAVERSE_VERTICAL, or both.

- When the user traverses out of your item, return *false* from the traverse() method. Most of the time, the device will call traverseOut(), if the focus has something else to move on to.

Demo

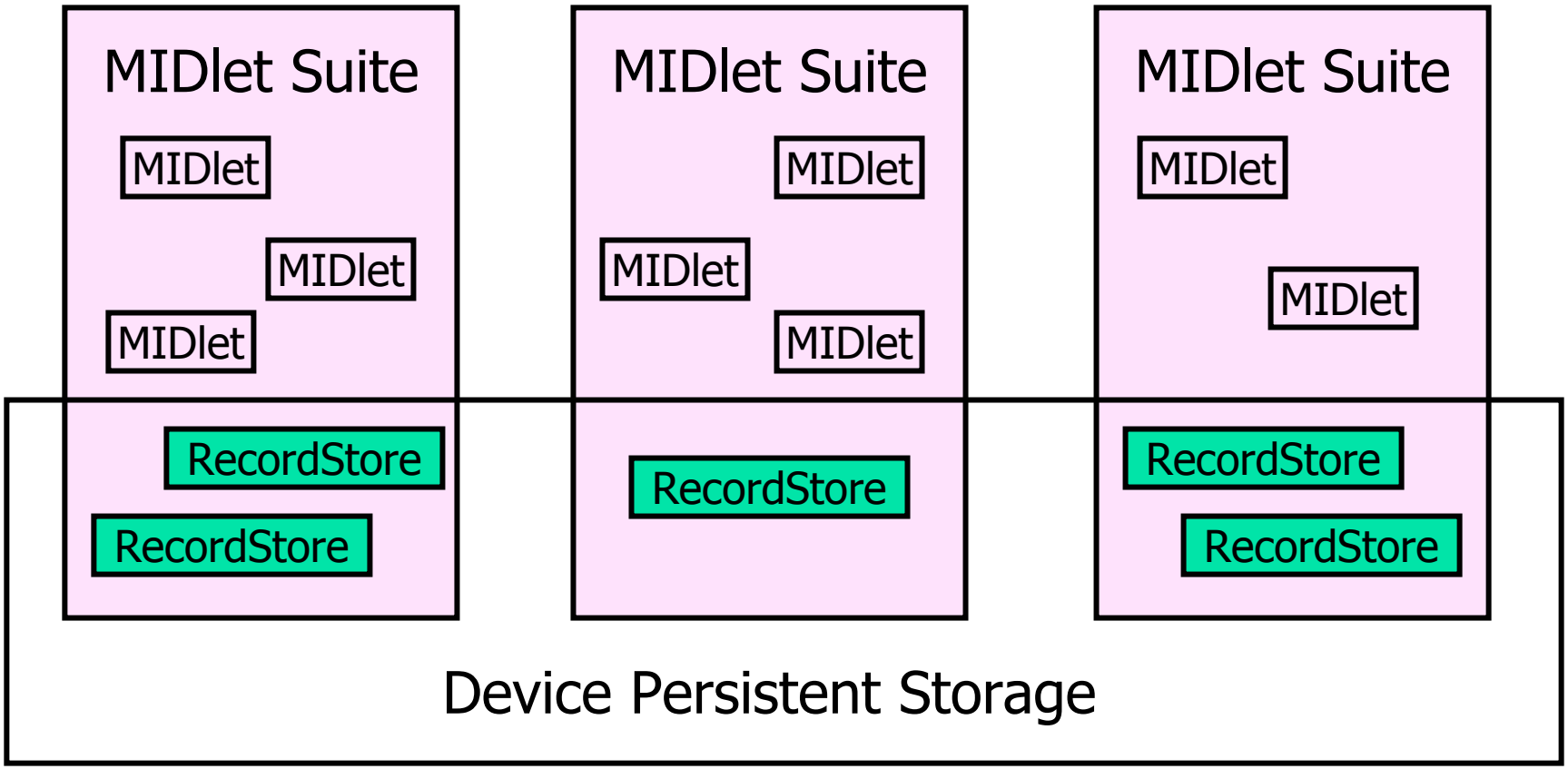
- Let's look at some custom items and run them to watch their behavior.

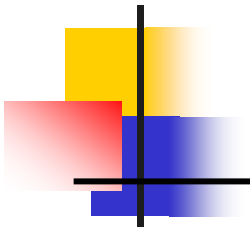


Persistent Storage

- Each device must provide 8KB of persistent storage. That's not much, believe me.
- Your MIDP application doesn't care how the device implements its storage. It just knows about small databases called record stores.
- MIDP 1.0 scoped access to the MIDlet suite.
- MIDP 2.0 now allows optional sharing.

Scope of Record Store Access





Managing Record Stores

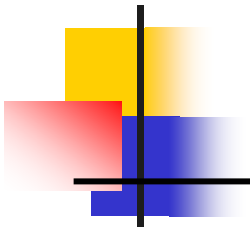
- Open a RecordStore with a name:

  public static RecordStore openRecordStore(String recordStoreName, boolean createIfNecessary) throws RecordStoreException, RecordStoreFullException, RecordStoreNotFoundException

- Close with:

  public void closeRecordStore()

- Don't keep record stores open for no reason.



Managing RecordStores

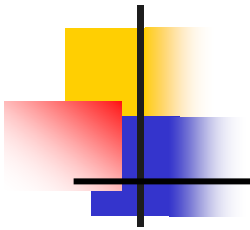
- List available record stores with:

  public static String[] listRecordStores()

- Delete a record store with:

  public static void deleteRecordStore(String recordStoreName)

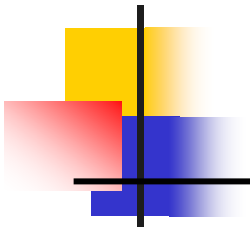
- These may take time, so consider placing record access on its own thread.



Sharing Record Stores

- MIDP 2.0 allows authorization modes to share record stores:

```
public static RecordStore openRecordStore(String
    recordStoreName, boolean createIfNecessary,
    byte authMode, boolean writable) throws
    RecordStoreException,
    RecordStoreFullException,
    RecordStoreNotFoundException
```



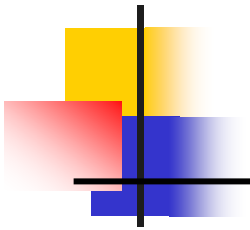
Sharing Record Stores

- authMode accepts:

  AUTHMODE_PRIVATE, AUTHMODE_ANY

- Do not to put secrets in a shared store.
- A MIDlet belonging to the suite that created the record store can change the authMode with:

```
public void setMode(byte authmode, boolean
  writable) throws RecordStoreException
```

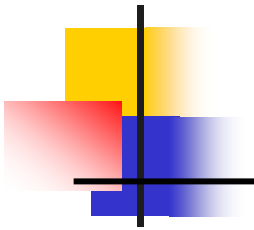


Sharing Record Stores

- Open a shared record store with:

  public static RecordStore openRecordStore(String recordStoreName, String vendorName, String suiteName) throws RecordStoreExeception, RecordStoreNotFoundException

- You must know the owning suite and vendor.

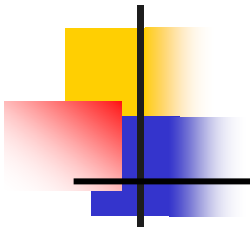


Records

- Records are byte arrays with an integer identification number.

```
public int addRecord(byte[] data, int offset, int
    numBytes) throws
    RecordStoreNotOpenException,
    RecordStoreException, RecordStoreFullException
```

- New record's identification number is returned.



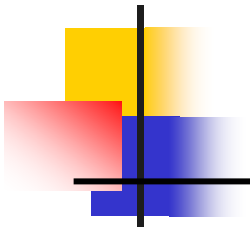
Records

- Get a Record with:

  public byte[] getRecord(int recordId) throws RecordStoreNotOpenException, InvalidRecordIDException, RecordStoreException

  public int getRecord(int recordId, byte[] buffer, int offset) throws RecordStoreNotOpenException, InvalidRecordIDException, RecordStoreException

- Try getRecordSize() if you don't know.

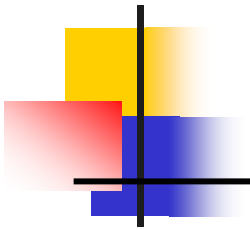


Records

- Delete by passing the ID number to deleteRecord().
- Replace an existing record with:

```
public void setRecord(int recordID, byte[]
    newData, int offset, int numBytes) throws
    RecordStoreNotOpenException,
    InvalidRecordIDException,
    RecordStoreException, RecordStoreFullException
```

- getNumRecords() & getNextRecordID()



Record Listening

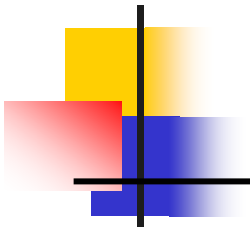
- Changes to your records create events and you can register listeners.

  public void addRecordListener(RecordListener listener)

  public void removeRecordListener(RecordListener listener)

- RecordListener interface has 3 methods:

  recordAdded(), recordChanged(), recordDeleted()

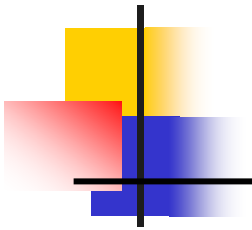


Performing a Query

- No SQL needed!

```
public RecordEnumeration
    enumerateRecords(RecordFilter filter,
    RecordComparator comparator,
    boolean keepUpdated) throws
    RecordStoreNotOpenException
```

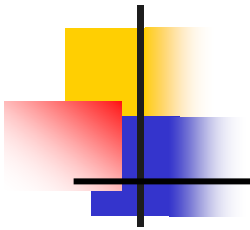
- Returns a sorted subset what's in the RecordStore. RecordFilter determines the subset. RecordComparator does the sort.



RecordFilter

- All records in the record store are pulled and the RecordFilter's matches() is called. It returns true for records in the result set.

```
public class SevenFilter implements
    javax.microeditionrms.RecordFilter {
    public boolean matches(byte[] candidate) {
    if (candidate.length == 0) return false;
    return (candidate[0] == 7);
```



RecordComparator

- Similar to java.util.Comparator. Needs just compare() method.

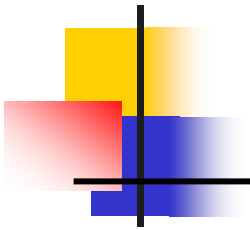
  Public int compare(byte[] rec1, byte[] rec2)

- Examine data in rec1 & rec2 and return:

  PRECEDES meaning rec1 before rec2

  FOLLOWS meaning rec1 after rec2

  EQUIVALENT meaning they are the same as far as sorting is concerned.



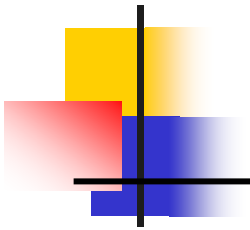
RecordEnumeration

hasNextElement() to find out if there is one.

nextRecord() to retrieve it.

nextRecordId() to retrieve its ID.

- You get the whole record or its ID, but the index cursor advances.
- Usually you'll walk through and retrieve records. Filter can be null to get all records. Comparator can be null, too, which results in no guarantee of the order of the subset.



RecordEnumeration

hasPreviousElement() to find out if there is one.

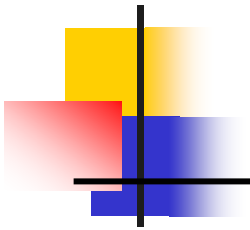
previousRecord() to retrieve it.

previousRecordId() to retrieve its ID.

reset() moves the index cursor back to the start.

destroy() to release the resources.

- Record store could change, so call rebuild() which reuses your same filter & comparator, or register for updates by passing true for keepUpdated. There's a price to pay.



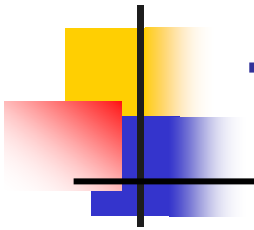
Resource Files

- You can, of course, keep resource files in your MIDlet suite jar for read-only data.
- Access it with an InputStream with getResourceAsStream()

```
InputStream in =
this.getClass().getResourceAsStream("/riverpoint.png");
```

Demo

- Let's put our understanding of Record Stores and Records to practice, ready?



Thank You!

- Please fill out your evaluation form now.
- I am a first-year presenter and really need your feedback.
- If I didn't get to something you wanted to cover, please ask me at the fireplace.