Cache images in RMS (Java ME)

This article presents a pattern for caching image objects into RMS and then restoring them when the Java ME application is next launched. This is useful when redrawing complex objects at runtime is too slow.

Overview

When you have large and complex pictures that are painted in run-time by your application using low-level API (e.g. fractal mountains as background picture or the multicolored sun with glares and fuzzy spots), sometimes it takes a long time to repaint them (especially on early java-embedded mobile-phones).

Of course, you can prepare a picture in PNG or JPG format beforehand and draw it instead of using low-level methods. But if your application works on different mobile-phones with different screen-resolutions, then you are required to create a huge number of such pictures at different resolutions and when the algorithm is changed, you need to re-convert them all.

The alternative way is to cache the complex picture when it is painted in run-time on device and then paint only the cached image. For this you should use methods:

- `Image.createImage(width, height)`
- `image.getGraphics()`

After you have an `Image` object that contains a complex picture you can save it in RMS; the image can then be reloaded and used the next time the application is launched.

Tip: You should measure what will be faster – reading cached image from RMS or fully execute complicated algorithm of painting picture into image.

This article was submitted by atiskov as an entry in the St Petersburg Developer Day Competition.

Code example

Here is the code for class that will attend saving images in RMS.

```
public class ImageStorage {
    public static final int ID_MOUNTAIN = 0;//indexes for images that will be cached
    public static final int ID_SUN = 1;
    public static final int NUMBER_OF_IMAGES = ID_SUN + 1;

    private static final Image[] CASHED_IMAGES = new Image[NUMBER_OF_IMAGES];//array of cashed images
    private static final int MAX_AREA_OF_IMAGE = 176 * 220;

// if we need more, it will grow in run-time
    private static final String RMS_IMAGES = "images";

//name of storage in RMS

    /**
     * This color will be used to indicate transparent color,
     * you can change it, but be sure that
     * your color is processed correctly by application.
     * It is setted only once (you can make it final)
     */
    protected static int COLOR_TO_BE_TRANSPARENT = 0xFFFFFF;

    public static boolean isLoaded = false; //flag indicates that images have been loaded from RMS


    /**
     * Class DrawerImageToBuffer for simplify caching images into graphics buffer (Pattern "template method" GOF)
     */
    public static abstract class DrawerImageToBuffer {
        protected int width;
        protected int height;
        private Image imgBuffer = null;

        private DrawerImageToBuffer() {
        }

        /**
         * Saves image, that is produced by overriden method drawImage,
         * in array CASHED_IMAGES
         *
         * @param w          - width of captured image
         * @param h          - height of captured image
         * @param imageIndex - index of of captured image
         */

        protected void process(int w, int h, int imageIndex) {
            width = w;
            height = h;
            imgBuffer = Image.createImage(width, height);
            Graphics g = imgBuffer.getGraphics();
            g.setColor(COLOR_TO_BE_TRANSPARENT);
            g.fillRect(0, 0, imgBuffer.getWidth(), imgBuffer.getHeight());
            drawImage(g);
            saveImageToArray(imageIndex);
        }

        abstract protected void drawImage(Graphics g);//template method

        /**
```

```
     * @param imageIndex - index of of captured image
     */
    private void saveImageToArray(int imageIndex) {
        CASHED_IMAGES[imageIndex] = imgBuffer;
    }
}
//-----------------------------------------------
/**
 * Method paintToBufferAndStoreImages() goes through all images that should be cached,
 * invoking their complicated algorithms of painting, storing result of painting in cash ImageStorage.CASHED_IMAGES,
 * and then saving images in RMS
 */
protected void paintToBufferAndStoreImages() {
    DrawerImageToBuffer TM = null;
    int size = Font.getDefaultFont().getHeight();

    TM = new MountainDrawingToCache();
    TM.process(size, size, ID_MOUNTAIN);

    TM = new SunDrawingToCache();
    TM.process(4, Font.getDefaultFont().getHeight() * 8, ID_SUN);

    storeImagesInRMS(CASHED_IMAGES);//saving images in RMS
}

/**
 * Gets table of colors of image curImage into int array rgbInts
 *
 * @param rgbInts  - result array with table of colors of image
 * @param curImage - image object from which we take table of colors
 * @param w        - width of taken image (we can take not whole region of image)
 * @param h        - height of taken image (we can take not whole region of image)
 * @param s        - area of image (should be w*h)
 */
protected static void getRGB(int[] rgbInts, Image curImage, int w, int h, int s) {
    curImage.getRGB(rgbInts, 0, w, 0, 0, w, h);
    for (int i = 0; i < s; i++)
        if ((rgbInts[i] & 0x00FFFFFF) == COLOR_TO_BE_TRANSPARENT)
            rgbInts[i] = (rgbInts[i] & 0x00FFFFFF);
}

/**
 * Checks if we need to update cash
 * You can use data that is stored in RMS to find is it neccessary to update
 *
 * @param recordStore - RMS store with additional data
 * @return false if data in RMS is up-to-date
 *         otherwise return true
 * @throws RecordStoreException
 * @throws IOException
 */
protected boolean additioanlCheckingOfNeedingReCashing(RecordStore recordStore) throws RecordStoreException, IOException {
    /*additional check e.g. data in RMS is out-of-date because of changed font or resolution*/
    return false;
}

/**
 * Restores array of images from RMS to array of cashed images
 *
 * @param arImages - array of images to be restored
 */
protected void restoreImagesFromRMS(Image[] arImages) {
    int[] intArrayOfRGBforImage = null;
    int w = 0;//width of image
    int h = 0;//height of image
    int l = 0;//area of image
    int curPointerToImage = 0;

    try {
        RecordStore recordStore = RecordStore.openRecordStore(RMS_IMAGES, true);
        RecordEnumeration re = recordStore.enumerateRecords(null, null, true);

        /* Here you can place code for taking additional info for re-cashing into RMS (you should simply skip it)
         * because it is already processed) */
        /*...*/

        try {
            while (re.hasNextElement()) {
                int id = re.nextRecordId();
                ByteArrayInputStream bais = new ByteArrayInputStream(recordStore.getRecord(id));
                DataInputStream inputStream = new DataInputStream(bais);

                try {
                    l = inputStream.readInt();
                    w = inputStream.readInt();
                    h = inputStream.readInt();
                    intArrayOfRGBforImage = new int[l];
                    for (int j = 0; j < l; j++)
                        intArrayOfRGBforImage[j] = inputStream.readInt();
                } catch (EOFException ioe) {
                    ioe.printStackTrace();
                }

                arImages[curPointerToImage++] = Image.createRGBImage(intArrayOfRGBforImage, w, h, true);
                System.gc();
            }
        }
        catch (IOException ioe) {
            ioe.printStackTrace();
        }
        recordStore.closeRecordStore();
    } catch (Exception rse) {
        rse.printStackTrace();
    }
}

/**
 * Stores array of images in RMS from array of cashed images
 *
 * @param images - array of images to be stored
 */
public void storeImagesInRMS(Image[] images) {
    int w, h, l;
    int[] rgbImage = new int[MAX_AREA_OF_IMAGE];
    try {
        try {// clear record store
            RecordStore.deleteRecordStore(RMS_IMAGES);
        } catch (RecordStoreException e) {
            e.printStackTrace();
        }
```

```
            RecordStore recordStore = RecordStore.openRecordStore(RMS_IMAGES, true);

            /*save additional info for checking necessity of re-cashing into RMS*/
            /*...*/

            for (int i2 = 0; (i2 < images.length) && (images[i2] != null); i2++) {
                ByteArrayOutputStream baos = new ByteArrayOutputStream();
                DataOutputStream outputStream = new DataOutputStream(baos);
                Image curImage = images[i2];

                w = curImage.getWidth();
                h = curImage.getHeight();
                l = w * h;
                if (l > MAX_AREA_OF_IMAGE)
                    rgbImage = new int[l];

                getRGB(rgbImage, curImage, w, h, l);
                try {
                    outputStream.writeInt(l);
                    outputStream.writeInt(w);
                    outputStream.writeInt(h);
                    for (int j = 0; j < l; j++)
                        outputStream.writeInt(rgbImage[j]);
                    System.gc();
                } catch (IOException ioe) {
                    ioe.printStackTrace();
                }

                byte[] b = baos.toByteArray();
                int id = recordStore.addRecord(b, 0, b.length);
            }
            recordStore.closeRecordStore();
        } catch (Exception rse) {
            rse.printStackTrace();
        }
    }

    /**
     * Checks if data in RMS (that contains cashed images) is up-to-date and if it is then
     * loads cashed images from RMS. If it is not, then the application cashes images by using
     * proper method drawImage, saves them into RMS, and then loads cashed images from RMS.
     *
     * @throws Exception
     */
    public void loadImages() throws Exception {
        RecordStore recordStore = RecordStore.openRecordStore(RMS_IMAGES, true);
        if (recordStore.getNumRecords() == 0)
            paintToBufferAndStoreImages();
        else {
            boolean isNeedReCash = additioanlCheckingOfNeedingReCashing(recordStore);
            if (isNeedReCash)
                paintToBufferAndStoreImages();
        }
        restoreImagesFromRMS(CASHED_IMAGES);
        isLoaded = true;
    }

    /**
     * Use this method instead of Graphics.drawImage to draw complicated cashed images
     *
     * @param g      - object of class {@link javax.microedition.lcdui.Graphics}
     * @param index  - index of cashed image in array of cashed images
     * @param x      - x coordinate of painted image
     * @param y      - y coordinate of painted image
     * @param anchor - anchor of painted image
     */
    public static void drawImage(Graphics g, int index, int x, int y, int anchor) {
        if (ImageStorage.isLoaded)
            try {
                g.drawImage(CASHED_IMAGES[index], x, y, anchor);
            } catch (Exception e) {
                isLoaded = false;
                e.printStackTrace();
            }
    }
}
```

In class Mountain you should define method drawSingleMountain that contains all complicated code of painting fractal mountain into object of Graphics. Also you should do in class Sun in method drawSingleSun.

```
// Overriden methods (you can place these classes into appropriate class (Mountain and Sun))
    public static class MountainDrawingToCache extends DrawerImageToBuffer {
        public void drawImage(Graphics g) {
            Mountain.drawSingleMountain(g/*plus other parametres if needed*/);
        }
    }

    public static class SunDrawingToCache extends DrawerImageToBuffer {
        public void drawImage(Graphics g) {
            Sun.drawSingleSun(g/*plus other parametres if needed*/);
        }
    }
```

In your Midlet class you can add such code in startApp method:

```
protected void startApp() throws MIDletStateChangeException {
        ImageStorage ids = new ImageStorage();
        try {
            ids.loadImages();
        } catch (Exception e) {
            ids.isLoaded = false;
            ids.paintToBufferAndStoreImages();
        }

        /* other code... */
    }
```

Now in order to draw a cashed image you should use class `ImageStorage` method

```
drawImage(Graphics g, int index, int x, int y, int anchor)
```