Java II

:: Activity

Activity: Data Storage (1h)

(Due: Thu, 08 Aug | Status: Not Completed)

Data Storage implementation will provide Android developers a Java API to store information on mobile devices. This lesson provides insight into File methods and practices associated with storing data to Internal Storage.

Objectives & Outcomes

Upon completion of this activity, the student will be able to...

- Identify the proper applications for storing information on internal storage.
- Utilize Java read/write file methods to store data in a String format into internal storage.

Level of Effort

This activity should take approximately 60m to complete. It will require:

- 60m Research
- 0m Prep & Delivery
- 0m Work

If you find that this activity takes you significantly less or more time than this estimate, please contact me for guidance.

Reading & Resources

Android Data Storage (related)

Android developer reference for storing data on Android devices.

Instructions

One of the biggest strengths of Android is the variety of data storage options provided through the API. We are going to explore Internal Storage of a String. Internal Storage represents storage that is built directly into the device for storing system data. This **does not** include storage to the device cache, internal non-removeable storage, or external storage devices such as SD cards.

Writing to Internal Storage

Writing to Internal Storage is a 3 step process involving opening an output to stream to the storage system, writing byte data to the system, and then closing the byte stream and

finalizing the file. This is accomplished through the methods *openFileOutput()*, *write()*, and *close()* respectively. It is important to note that both *write* and *close* are methods of the *FileOutputStream* object, which is returned by the *openFileOutput* method.

openFileOutput(String name, int mode) – This method opens a private file associated with an application package context to write to the file system. The first parameter represents the name of the file to be written. If a file with that name does not exist, one will be created. If a file with the name does exist, the mode will determine how the existing data is treated in conjunction with the new data. The mode will also determine the permissions for the file. Valid modes are as follows:

- **MODE_PRIVATE** – *Default* – Writes to the file system, granting access permission only to the application context.
- **MODE_APPEND** – Append data to the existing file, keeping existing user permissions.
- **MODE_WORLD_READABLE** – Write data granting read access to all applications.
- **MODE_WORLD_WRITEABLE** – Write data granding full access to all applications.

write(byte[] buffer) – This method accepts a byte array to be written to the file system. Depending on the type of data needing to be written, it will need to be converted into a byte array. For example, the Java String object has an integrated method which converts String data into bytes called *getBytes()*. Writing a string to storage is relatively simple:

```
String content = "Java Rules!";

FileOutputStream fos = openFileOutput("MyFile", Context.MODE_PRIVATE);
fos.write(content.getBytes());
fos.close();
```

Note that *Context* is a reserved keyword that represents the current application context within an Android application.

Writing other data types to internal storage isn't much different, with the exception of converting the data into a byte array. That said, it is important to remember that the steps to write a file are the same – Open, Write Byte Stream, Close. To illustrate the point, here is some possible code to write image data to Internal Storage:

```
Bitmap bmp = BitmapFactory.decodeStream( [image] );
FileOutputStream fos = openFileOutput("MyImage.jpg", Context.MODE_PRIVATE);
bmp.compress(CompressFormat.JPEG, 100, fos);
fos.close();
```

Note that *[image]* represents an image object within the Java application. Furthermore, rather than using the *write()* method, we are using Java's *Bitmap.compress()* method to write the image byte stream to the *FileOutputStream*.

Reading from Internal Storage

Similar to writing a file to Internal Storage, data can be read. Rather than opening a *FileOutputStream*, we open a *FileInputStream* with the method *openFileInput()*. Like when

we generate a byte stream for writing a file, reading a *FileInputStream* will require converting the byte stream to a working type. Typically, the process of converting an incoming byte stream into a working data set involves buffering the incoming stream data with *BufferedInputStream*. Once the incoming file stream is buffered, we can parse the bytes into the required type using *BufferedInputStream.read()*. The body of a method for opening a file would resemble the following:

```
FileInputStream fin = openFileInput([file]);
BufferedInputStream bin = new BufferedInputStream(fin);
byte[] contentBytes = new byte[1024];
int bytesRead = 0;
String content;
StringBuffer contentBuffer = new StringBuffer();

while( (bytesRead = bin.read(contentBytes)) != -1){
content = new String(contentBytes,0,bytesRead);
contentBuffer.append(content);
}

return contentBuffer;
```

Handling Exceptions

Note that the above code blocks will likely result in compilation errors in that it is neccessary to handle possible exceptions thrown by the *FileOutputStream* and *FileInputStream* classes. As such, it is good practice to wrap file access calls within try catch blocks which handle the exceptions as shown below with reading files...

```
String readTextFile(String filename){
try{
FileInputStream fin = openFileInput([file]);
BufferedInputStream bin = new BufferedInputStream(fin);
byte[] contentBytes = new byte[1024];
int bytesRead = 0;
String content;
StringBuffer contentBuffer = new StringBuffer();

while( (bytesRead = bin.read(contentBytes)) != -1){
content = new String(contentBytes,0,bytesRead);
contentBuffer.append(content);
}

return contentBuffer;
}
catch(FileNotFoundException e){
return "";
}
catch(IOException e){
return "";
}
}
```

Deliverables

There is no deliverable information associated with this activity