

Sun Certified Mobile Application Developer (311-110)

Sun Microsystems, Inc.

Exam Notes

Sathya Srinivasan

02/20/2004

Table of Contents

Requirements of CLDC Specification.....	3
Scope of CLDC Specification.....	3
Differences between CLDC 1.0 and CLDC 1.1.....	4
Differences between CLDC VM and JLS VM.....	5
Comparison between CLDC and J2SE classes.....	6
System Classes.....	6
Datatype Classes.....	6
Utility Classes.....	7
Collection Classes.....	7
Calendar and Time Classes.....	7
Input/Output Classes.....	7
Exception Handling in CLDC.....	9
Errors.....	9
Application Management Exceptions.....	9
Language Exceptions.....	9
Utility/Collection Exceptions.....	10
Input/Output Exceptions.....	10
Garbage Collection in CLDC.....	11

Chapter 2 CLDC 1.0/1.1

Identify correct and incorrect statements or examples about the requirements and scope of the CLDC specification, including the differences between 1.0 and 1.1.

Requirements of CLDC Specification

- Hardware Requirements
 - At least 160KB of non-volatile memory is available for the VM and CLDC libraries.
 - At least 32KB of volatile memory is available for the VM runtime.
- Software Requirements
 - A host OS exists and provides one schedulable entity to run the Java VM.

Scope of CLDC Specification

- What is in scope?
 - Java language and VM features
 - Core Java libraries (`java.lang.*` and `java.util.*`)
 - Input/Output libraries (`java.io.*`)
 - Security
 - Networking
 - Internationalization
- What is not in scope?
 - Application installation and lifecycle management
 - User interface functionality
 - Event handling
 - High-level application model (human-device interaction)
- The items that are not in scope are typically addressed in the profiles.

Differences between CLDC 1.0 and CLDC 1.1

- Floating point support has been added.
 - All floating point byte codes are supported by CLDC 1.1.
 - Classes `Float` and `Double` have been added.
 - Various methods have been added to the other library classes to handle floating point values.
- Weak reference support (small subset of the J2SE weak reference classes) has been added.
- Classes `Calendar`, `Date` and `TimeZone` have been redesigned to be more J2SE-compliant.
- Error handling requirements have been clarified, and one new error class, `NoClassDefFoundError`, has been added.
- In CLDC 1.1, `Thread` objects have names like threads in J2SE do. The method `Thread.getName()` has been introduced, and the `Thread` class has a few new constructors that have been inherited from J2SE.
- Various minor library changes and bug fixes have been included, such as the addition of the following fields and methods:
 - `Boolean.TRUE` and `Boolean.FALSE`
 - `Date.toString()`
 - `Random.nextInt(int n)`
 - `String.intern()`
 - `String.equalsIgnoreCase()`
 - `Thread.interrupt()`
- Minimum memory budget has been raised from 160 to 192 kilobytes, mainly because of the added floating point functionality.
- Specification text has been tightened and obsolete subsections removed.

Describe the ways in which a CLDC virtual machine does and does not adhere to the Java Language Specification (JLS) and the Java Virtual Machine specification.

Differences between CLDC VM and JLS VM

The CLDC VM defines a subset of JLS VM requirements due to the device constraints. The CLDC VM differs from the JLS VM in the following ways.

- Class instances cannot be finalized. There is no `Object.finalize()` method.
- Limited error and exception handling facilities. See [Exception handling in CLDC](#) for more details.
 - In short, the error and exception classes are limited in CLDC and if an implementing VM performs more checks than those mentioned in CLDC, it is expected to throw the nearest appropriate `Error` to the application.
 - Asynchronous exceptions are not available (ex. `InternalError` in JLS).
- User defined class-loaders are not available.
- There is no concept of `Thread` groups (although user can simulate this by using collections).
- There is no concept of daemon threads.
- Class file verification is done in two steps (a pre-verifier which is run on the development system, like your desktop and a runtime verifier which is run on the device during the installation of the application).
 - The verification is a linear pass on the code and does not have complex algorithms like in JLS.
 - An application cannot be installed on a device without running it through the pre-verifier.
- Class lookup order cannot be modified.
 - Application programmer cannot modify CLDC, profile, or manufacturer-specific packages.

Identify correct and incorrect statements or examples about CLDC classes including those derived from J2SE, and the CLDC-specific classes, including identifying which core J2SE classes are NOT included in CLDC, or have different behaviors (example: `java.lang.String`, io classes, etc.)

Comparison between CLDC and J2SE classes

All or most of the classes defined in CLDC that have a counterpart in J2SE have a subset of functionalities.

- Localization is not supported in CLDC classes.
- Only **Basic-Latin** and **Latin-1-Supplement** character sets are required to be supported.

System Classes

Class in CLDC	Limitations/Differences with respect to J2SE counterpart
<code>java.lang.Object</code>	No <code>finalize()</code> method.
<code>java.lang.Class</code>	No support for reflection API.
<code>java.lang.Runtime</code>	No support for native library invocations (<code>exec()</code> , etc.)
<code>java.lang.System</code>	No support for native library invocations (<code>exec()</code> , etc.)
<code>java.lang.Thread</code>	No support for daemon threads.
<code>java.lang.Runnable</code>	
<code>java.lang.String</code>	Minimal support for internationalization. CLDC 1.1 defines support for <code>float</code> and <code>double</code> datatype conversions. CLDC 1.1 defines <code>equalsIgnoreCase()</code> method. CLDC 1.1 defines <code>intern()</code> method.
<code>java.lang.StringBuffer</code>	CLDC 1.1 defines support for <code>float</code> and <code>double</code> datatype conversions. No support for <code>substring()</code> or <code>replace()</code> operations.
<code>java.lang.Throwable</code>	No support for dynamic stack traces. No support for exception chaining.
<code>java.lang.ref.Reference</code>	Since CLDC 1.1
<code>java.lang.ref.WeakReference</code>	Since CLDC 1.1

Datatype Classes

Class in CLDC	Limitations/Differences with respect to J2SE counterpart
<code>java.lang.Boolean</code>	CLDC 1.1 defines <code>Boolean.TRUE</code> and <code>Boolean.FALSE</code> constants.
<code>java.lang.Byte</code>	
<code>java.lang.Character</code>	Conversion operations and properties are valid only for ISO-Latin 1 character set.
<code>java.lang.Double</code>	Since CLDC 1.1
<code>java.lang.Float</code>	Since CLDC 1.1
<code>java.lang.Integer</code>	
<code>java.lang.Long</code>	
<code>java.lang.Short</code>	

Utility Classes

Class in CLDC	Limitations/Differences with respect to J2SE counterpart
<code>java.lang.Math</code>	Only basic math operations are defined. In CLDC 1.1 basic trigonometric methods (<code>sin</code> , <code>cos</code> , <code>tan</code>) have been added. In CLDC 1.1 constants have been added for <code>PI</code> and <code>E</code> . There is no <code>random()</code> method.
<code>java.util.Random</code>	A subset of functionalities is provided to generate pseudo-random numbers.

Collection Classes

- Only thread-safe collections are provided.
- The collection framework interfaces are not available.

Class in CLDC	Limitations/Differences with respect to J2SE counterpart
<code>java.util.Vector</code>	
<code>java.util.Hashtable</code>	The load factor is always 0.75 and cannot be modified.
<code>java.util.Stack</code>	
<code>java.util.Enumeration</code>	

Calendar and Time Classes

Class in CLDC	Limitations/Differences with respect to J2SE counterpart
<code>java.util.Calendar</code>	A subset of the JDK 1.3 counterpart.
<code>java.util.Date</code>	A subset of the JDK 1.3 counterpart.
<code>java.util.TimeZone</code>	Only <code>GMT</code> is required to be supported. Rest are optional.

Input/Output Classes

Class in CLDC	Limitations/Differences with respect to J2SE counterpart
<code>java.io.InputStream</code>	
<code>java.io.ByteArrayInputStream</code>	
<code>java.io.DataInput</code>	
<code>java.io.DataInputStream</code>	
<code>java.io.Reader</code>	
<code>java.io.InputStreamReader</code>	If encoding is not specified, value defined in the <code>microedition.encoding</code> system property is used. If encoding is not available, <code>UnsupportedEncodingException</code> is thrown.
<code>java.io.OutputStream</code>	
<code>java.io.ByteArrayOutputStream</code>	
<code>java.io.PrintStream</code>	
<code>java.io.DataOutput</code>	
<code>java.io.DataOutputStream</code>	<code>UTFDataFormatException</code> is thrown in <code>readUTF()</code> if data is not UTF.

```
java.io.OutputStreamWriter
```

If encoding is not specified, value defined in the `microedition.encoding` system property is used.
If encoding is not available, `UnsupportedEncodingException` is thrown.

Given the differences and limitations of exception/error handling with CLDC devices, handle exceptions correctly.

Exception Handling in CLDC

- A new `Error` called `NoClassDefFoundError` has been added in CLDC 1.1.
- When an `Error` other than those defined in CLDC occurs
 - The VM may halt in an implementation-specific manner
 - The nearest supported CLDC `Error` is thrown.
- Asynchronous errors and exceptions (like `InternalError`) are not supported in CLDC.
- `SecurityException` can be thrown for any operation that needs permission (like opening a network connection).
- `InterruptedIOException` can be thrown by any data transfer operation, if the connection is lost during data transfer. This exception may also be thrown if the application had requested to be notified of connection timeouts, although this is not guaranteed.

Errors

Error	Description
<code>java.lang.Error</code>	
<code>java.lang.NoClassDefFoundError</code>	Since CLDC 1.1
<code>java.lang.VirtualMemoryError</code>	Thrown if enough resources are not available.
<code>java.lang.OutOfMemoryError</code>	

Application Management Exceptions

Exception	Description
<code>java.lang.RuntimeException</code>	
<code>java.lang.SecurityException</code>	
<code>java.lang.ClassNotFoundException</code>	
<code>java.lang.InstantiationException</code>	
<code>java.lang.IllegalMonitorStateException</code>	
<code>java.lang.IllegalThreadStateException</code>	
<code>java.lang.InterruptedException</code>	Thread is interrupted.

Language Exceptions

Exception	Description
<code>java.lang.IllegalAccessException</code>	Appropriate access modifier is not present (no <code>public</code> method, etc.)
<code>java.lang.ClassCastException</code>	
<code>java.lang.NullPointerException</code>	
<code>java.lang.IllegalArgumentException</code>	
<code>java.lang.NumberFormatException</code>	

Utility/Collection Exceptions

Exception	Description
java.lang.ArithmeticException	
java.lang.IndexOutOfBoundsException	
java.lang.ArrayIndexOutOfBoundsException	
java.lang.StringIndexOutOfBoundsException	
java.lang.ArrayStoreException	
java.lang.NegativeArraySizeException	
java.util.EmptyStackException	
java.util.NoSuchElementException	

Input/Output Exceptions

Exception	Description
java.io.IOException	
java.io.EOFException	
java.io.InterruptedIOException	
java.io.UnsupportedEncodingException	
java.io.UTFDataFormatException	
java.microedition.io.ConnectionNotFoundException	

Wirte code that effectively manages memory / garbage collection.

Garbage Collection in CLDC

- Reuse a `GameCanvas` object since a buffer is created for each `GameCanvas` instance. Creating multiple instances will result in multiple buffers, which is bad for performance.
- Call `GameCanvas.getGraphics()` method to get the `Graphics` object BEFORE starting the game and re-use the object again, since each call gets a new instance of the `Graphics` object.
- Set objects to `null` once you are done with the objects.
- Setting `Image` objects to `null` after using them frees up a lot of memory.
- `System.gc()` can be called to explicitly run the Garbage Collector in the same way as that of the J2SE counterpart. Use this sparsely.
- Since `Object.finalize()` method is not available in CLDC, it significantly simplifies the Garbage Collection algorithm.
- Use the `java.lang.ref.WeakReference` as necessary to help in GC (since CLDC 1.1).
- Pooling objects might affect efficient GC since references are not released.