



C H A P T E R 4

NotesFactory/NotesThread/ NotesError classes

- | | | | |
|---------------------------|----|---|----|
| 4.1 NotesFactory class | 33 | 4.6 NotesError class | 36 |
| 4.2 What is a thread? | 34 | 4.7 Memory management/recycle
method | 36 |
| 4.3 NotesThread class | 35 | 4.8 Chapter review | 37 |
| 4.4 What is an exception? | 35 | | |
| 4.5 NotesException class | 35 | | |

For your Java code to work in Domino, it must make itself known to the Domino Server. For Java code (other than Domino agents), this can be done using the `NotesFactory` class.

4.1 NOTESFACTORY CLASS

The `NotesFactory` class is used in non-Domino agent code; that is, code running outside of the Domino environment that accesses Domino objects must use the `NotesFactory` class. Applications that make local calls (on the same machine as the Domino server) should use the `createSession` method with no parameters. Code that makes remote calls to a Domino server must use the `createSession-(serverName)` method. Remote code accesses a Domino server via IIOP (Internet InterOrb Protocol).

The `NotesFactory` class has only one method named `createSession`. The method is overloaded, so there are a number of variations of it. Here are the numerous formats of the `createSession` method:

```

createSession();
createSession("Server IP address");
createSession("Server IP address", "username", "password");
createSession("Server IP address", String args[], "username", "password");
createSessionWithIOR("IOR");
createSessionWithIOR("IOR", "username", "password");
createSessionWithIOR("IOR", String args[], "username", "password");
createSession(Applet, "username", "password");
createSession(Applet, org.omg.CORBA.ORB.ORB orb, "username", "password")

```

The only other method is getIOR:

```
getIOR("Server IP address");
```

Example 4.1 shows a standalone application that uses one variation of createSession to connect to a remote server.

Example 4.1 (Standalone application)

```

import lotus.domino.*;
public class Example_4_1 implements Runnable {
    public static void main(java.lang.String[] args) {
        Example1 t = new Example1();
        Thread nt = new Thread((Runnable)t);
        nt.start(); }
    public void run() {
        try {
            Session s = NotesFactory.createSession("200.118.34.8", "tpatton", "password");
            String p = s.getPlatform();
            String cu = s.getCommonUserName();
            String nv = s.getNotesVersion();
            String sn = s.getServerName();
            String un = s.getUserName();
            s.recycle();
        } catch (NotesException n) {
            System.out.println("ID: " + n.id + " -- Name: " + n.text);
        } catch (Exception e) {
            e.printStackTrace(); } } }

```

Don't worry if you don't yet understand what each part of this Java code does. As we work through further examples of java code, I will explain the significance of the various parts of the code in more detail.

4.2 WHAT IS A THREAD?

The book *JAVA Threads*, by Scott Oaks and Henry Wong, defines a thread as follows:

“The term thread is shorthand for thread of control, and a thread of control is, at its simplest, a section of code executed independently of other threads of control within a single program.”

They continue to explain:

“Thread of control is the path taken by an application during execution.... Having multiple threads of control is like executing items from two lists.”

4.3 *NOTESTHREAD CLASS*

The `NotesThread` class extends the basic `Thread` class in the basic Java language. Threads do not have to be incorporated in Java agents; they are included in the `AgentBase` class that is used as the base class for all Domino Java agents.

You can use threads in Domino in one of three ways:

- 1 Extend the `NotesThread` class
- 2 Implement the `Runnable` interface (see Example 4.1)
- 3 Use static methods to initiate a thread (`sinitThread`) and terminate a thread (`stermThread`)

4.4 *WHAT IS AN EXCEPTION?*

Java exception handling allows a Java program to catch all types of exceptions. You can use it to catch specific exceptions as well. Java exception handling is designed to smoothly handle errors rather than abruptly crash a program. Exception handling is useful for the following events:

- handling exceptional and unforeseen situations
- processing errors from other components (such as third-party vendors) that don't handle their own very well
- handling exceptions in a clean fashion

4.5 *NOTESEXCEPTION CLASS*

The `NotesException` class provides the facility of exception handling for Domino Java agents. You should use it in `try/catch/finally` clauses, such as the following, to catch Domino errors:

```
try {  
    // code  
} catch (NotesException n) {  
    // code  
} catch (Exception e) {  
    // code }
```

You should include handling for `NotesException` as well as normal Java exceptions. The previous snippet of code provides that functionality.

4.5.1 **ID property**

The `id` property of the `NotesException` class returns the Domino error code (Appendix B contains Domino error codes). You can access it as follows:

```
NotesException.id
```

4.5.2 Text property

The `text` property of the `NotesException` class contains the text description of the error message returned:

```
NotesException.text

try {
    // code
} catch (NotesException n) {
    System.out.println("Exception ID: " + n.id);
    System.out.println("Exception description: " + n.text);
} catch (Exception e) {
    // code }
```

The exception handling code appears throughout the examples in this book.

4.6 NOTESERROR CLASS

The `NotesError` class defines error code variables. You can use the defined variables in conjunction with the `NotesException` class to catch and handle specific Domino errors accordingly. Appendix B lists the possible error codes.

4.7 MEMORY MANAGEMENT/RECYCLE METHOD

One of the most talked about features of the Java language is garbage collection. Languages, such as C and C++, require developers to take care of the objects they use; they must manage the memory themselves. Memory leaks can occur in this scenario if you forget to dispose of an object. This is not a problem in Java, because Java performs automatic “garbage collection” of memory.

Every Java class has a method called `finalize`. It is often referred to as the finalizer method. Its job is to return resources used by its object to the system. Java guarantees that the `finalize` method will be called just before garbage collection is performed.

The Java Virtual Machine decides when and if the garbage collector is called. It is out of your hands, so there isn't much to rely on with garbage collection. In fact, the garbage collector may not ever be called.

4.7.1 recycle method

Every Domino Java object has a `recycle` method. This method handles the cleanup of system resources used by the object. The Java garbage collector runs after the `recycle` method is called. The Java garbage collector has been disabled in Domino, so it can only run on objects once the `recycle` method has been called. The garbage collector will not run immediately; it will run when it deems it necessary.

For this reason, you must personally handle the cleanup of your Domino objects. This becomes a real issue in large amounts of code, but it is good practice to always use `recycle`.

Here are a few tips regarding recycling:

- Always create a new Domino session object for each servlet request, and call its `recycle` method when you are finished with the session object.
- Choose your spots in a Domino agent for recycling.
- Keep the memory footprint small.
- A good spot for recycling is just after exiting a loop.

4.7.2 Syntax

The `recycle` method accepts no parameters, so it is called with empty parentheses. Here is the `recycle` method used for a Domino `Session` object:

```
session.recycle();
```

Avoiding the `recycle` method in your Domino code will cause major memory problems. The Java garbage collector is unpredictable and it cannot “garbage collect” Domino objects until the `recycle` method has been called. For this reason, get into the habit of recycling all Domino-related objects.

4.8 CHAPTER REVIEW

There are a number of special Java classes available for accessing Domino objects via Java. The `NotesFactory` class can create sessions with remote Domino servers. The `NotesException` and `NotesError` classes work with Domino exceptions and errors. Garbage collection of Domino objects requires the use of the `recycle` method.

- The `NotesFactory` class is a special class only used to access Domino objects from outside the Domino agent environment.
- The `NotesFactory` class contains the `createSession` method that is overloaded to provide the capability to create a session for use with local and remote calls and calls via Corba.
- The `NotesException` class provides a means to handle errors specific to the Domino environment.
- The `NotesException` class provides `id` and `text` properties for accessing exception data.
- The `NotesError` class defines error variables for use with Domino.
- The `recycle` method is common to all classes in the Domino Java class library.