



## Advanced Java Programming



## Objectives

### To introduce

- Multithreading
- Streams, Files, Persistence of objects, Serialization
- Java Database Connectivity
- Remote Method Invocation
- Java Beans and Reflection API
- Concepts of Java Naming and Directory Services



Copyright © 2005, Infosys  
Technologies Ltd

2

Infosys®

## References

- Horstmann, Cay S & Cornell, Gary, *Core JAVA 2 Vol-1- Fundamentals*, 7<sup>th</sup> Edition, Prentice Hall/Sunsoft Press
- Horstmann, Cay S & Cornell, Gary, *Core JAVA 2 Vol-2 - Advanced features*, 7<sup>th</sup> Edition, Prentice Hall/Sunsoft Press
- Jaworski, Jamie, *JAVA 2 Unleashed : Expert insight, Powerful Software, Authoritative Advice*, 2<sup>nd</sup> Edition, Techmedia/Sams
- Web Site : <http://java.sun.com/docs/books/tutorial/>  
available on Knowledge Shop also



Copyright © 2005, Infosys  
Technologies Ltd

3

Infosys®

*Core Java2 - Vol 1: Streams and files*

*Core Java2 - Vol 2: RMI, CORBA, JDBC, NativeMethods etc...*

*Java 2 Unleashed:*

- *Java 2 Platform API*
- *Network Programming*
  - *Naming & Directory Services*
- *Distributed Application development using RMI*
- *Database Programming*

Infosys®

POWERED BY INTELLECT  
DRIVEN BY VALUES



## Unit - 1 Multithreading



## What are Threads?

- A thread is a single sequential flow of control within a program
- Facility to allow multiple activities within a single process
- Referred as **lightweight** process
- Each thread has its own **program counter**, **stack** and **local variables**
- Threads share **memory**, **heap area**, **files**



Copyright © 2005, Infosys Technologies Ltd

5

Infosys®

## Thread Advantages

Threads are memory efficient. Many threads can be efficiently contained within a single EXE, while each process will incur the overhead of an entire EXE.

Threads share a common program space, which among other things, means that messages can be passed by queuing only a pointer to the message. Since processes do not share a common program space, the kernel must either copy the entire message from process A's program space to process B's program space - a tremendous disadvantage for large messages, or provide some mechanism by which process B can access the message.

Switching between Threads is faster, since a thread has less context to save than a process.

## Why use Threads?

- To perform asynchronous or background processing
- Increases the responsiveness of GUI applications
- Better utilization of system resources
- Simplify program logic when there are multiple independent entities



Copyright © 2005, Infosys  
Technologies Ltd

6

Infosys®

A very good example of a multi-threaded application is a game software where the GUI display, updation of scores, sound effects and timer display are happening within the same application simultaneously

## Creating the Thread

Two ways:

- ④ Extending the Thread class
- ④ Implementing the Runnable interface



Copyright © 2005, Infosys  
Technologies Ltd

7

Infosys®

In Java we cannot extend from more than one class. In some cases a class might be extending from some parent class and also we need to extend from Thread class. In such cases we can do multithreading by implementing Runnable interface.

## The Thread class (1 of 2)

- ④ Extend the Thread class
- ④ Override the run() method
- ④ Create an object of the sub class and call the start method to execute the Thread



Copyright © 2005, Infosys  
Technologies Ltd

8

Infosys®

A thread can be created by creating a new class that extends Thread, and then by creating an instance of that class. The extending class must override the run() method, which is the entry point for the new thread. It must also call start() to begin execution of the new thread.

During its lifetime, a thread spends some time executing and some time in any of several non-executing states. When a thread gets to execute, it executes the method run().

Java's multithreading system is built upon the Thread class, its methods, and its companion interface Runnable. You can't directly refer to the real state of a running thread, you should deal with it through its proxy, the Thread instance that spawned it.



## The Thread class (2 of 2)

```
class ExtThread extends Thread
{
    ExtThread()
    {
        System.out.println("Child Thread created");
    }
    public void run()
    {
        System.out.println("Child Thread running");
    }
}
```



Copyright © 2005, Infosys  
Technologies Ltd

9

Infosys®

The above code create a new class that extends Thread, and then create an instance of that class.

The extending class must override the run() method, which is the entry point for the new thread. It must also call the start() to begin the execution of the new thread.

The ExtThread() is the constructor of the class which is implicitly called when the instance of the extending class is created.

## The Runnable Interface (1 of 2)

- Useful when the class is already extending another class and needs to implement multithreading
- Need to implement the `run()` method
- Create a thread object (the worker) and give it a `Runnable` object (the job)
  - `public Thread(Runnable target);`
- Start the thread by calling the `start()` method



Copyright © 2005, Infosys Technologies Ltd

10

Infosys®

The easiest way to create a thread is to create a class that implements the Runnable interface. Runnable abstracts a unit of executable code.

We can construct a thread on any object that implements Runnable.

To implement Runnable, a class need only implement a single method called `run()`.

Inside `run()`, we define the code that constitutes the new thread. This can call other methods, use other classes, and declare variables, just like the main thread can.

The only difference is that `run()` establishes the entry point for another, concurrent thread of execution within your program. This thread will end when `run()` returns.

## The Runnable Interface (2 of 2)

```
class RunnableThread implements Runnable
{
    RunnableThread()
    {
        System.out.println("Child Thread: ");
    }
    public void run()
    {
        System.out.println("Hi I'm a new thread");
    }
}
```



Copyright © 2005, Infosys  
Technologies Ltd

11

Infosys®

We need to create a class that implements Runnable, we then instantiate an object of type Thread from within that class.

Thread defines several constructors which are called when the instance of the object is created.

After the new thread is created, it will not start running until you call its start() method, which is declared within Thread.

In essence, start() executes a call to run().

## Starting the Thread

### The main() method when extending the Thread class

```
class ExtThreadMain
{
    public static void main(String a[])
    {
        System.out.println("Hi I'm main thread");
        ExtThread obj=new ExtThread ();
        obj.start();
        System.out.println("This is the main thread printing");
    }
}
```



Copyright © 2005, Infosys  
Technologies Ltd

12

Infosys®

The above code explains the instance being created for the extending class which is followed by the call to the start() method. This will invoke the run() method implicitly.

Similarly when implementing the Runnable interface the instance of the implementing class is created and the object of the same is passed to the object of the Thread class which then invokes the start() method, which starts the thread of execution beginning at the run() method.

## Starting the Thread

### ④ The main() method when implementing the Runnable Interface

```
class RunnableThreadMain
{
    public static void main(String a[])
    {
        System.out.println("Hi I'm main thread");
        RunnableThread rt=new RunnableThread();
        Thread t=new Thread(rt);
        t.start();
    }
}
```



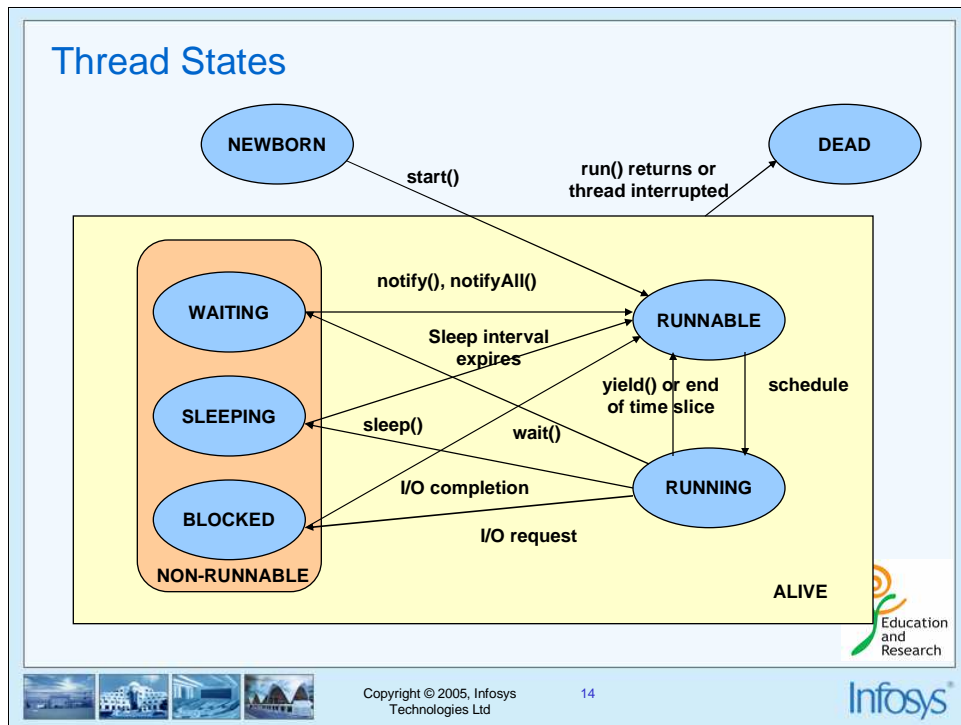
Copyright © 2005, Infosys  
Technologies Ltd

13

Infosys®

When it comes to choosing an approach, the Thread class defines several methods that can be overridden by a derived class. Of these methods, the only one that must be overridden is run().

This is the same method required when you implement Runnable.



### Newborn State:

When the new instance of the thread is created by executing this constructor `Thread(...)`, Thread is newborn. In this state no resource are allocated to the thread. Calling any other methods on the newborn thread results into throwing an `IllegalStateException`.

### Runnable State:

When a `start()` method is called on the newborn thread instance, thread makes the transition to Runnable state. In this state the thread is waiting for the scheduler to schedule it on the processor.

### Running State:

When the `run()` method is invoked the thread is being executed by the CPU bringing it to the running state.

### Non-Runnable State:

A running state can be suspended, which temporarily suspends its activity. This is done by invoking `sleep()` or `wait()` method. A suspended thread can then be resumed, allowing it to pick up where it left off, by calling `notify()` method.

A thread can be blocked when waiting for a resource.

### Dead State:

A thread comes to a dead state if it over with the task which forms the part of its code. At any time, a thread can be terminated, which halts its execution immediately. Once terminated, a thread cannot be resumed.

## Thread scheduling

- There are 2 techniques for thread scheduling
  - Pre-emptive and time-sliced
- In preemptive scheduling, the thread with a higher priority preempts threads with lower priority and grabs the CPU
- In time-sliced or round robin scheduling, each thread will get some time of the CPU
- Java runtime system's thread scheduling algorithm is preemptive but it depends on the implementation
- Solaris is preemptive, Macintosh and Windows are time sliced
- In theory, a thread with high priority should get more CPU time, but practically it may depend on the platform



Copyright © 2005, Infosys Technologies Ltd

15

Infosys®

The Java runtime system's thread scheduling algorithm is *Fixed Priority preemptive*. If at any time a thread with a higher priority than all other runnable threads becomes runnable, the runtime system chooses the new higher priority thread for execution. The new higher priority thread is said to *preempt* the other threads.

Some systems, such as Windows 95/NT, fight selfish thread behavior with a strategy known as *time-slicing*. Time-slicing comes into play when there are multiple "Runnable" threads of equal priority and those threads are the highest priority threads competing for the CPU. A time-sliced system divides the CPU into time slots and iteratively gives each of the equal-and-highest priority threads a time slot in which to run. The time-sliced system iterates through the equal-and-highest priority threads, allowing each one a bit of time to run, until one or more of them finishes or until a higher priority thread preempts them. Notice that time-slicing makes no guarantees as to how often or in what order threads are scheduled to run.

**Note:** The Java runtime does not implement (and therefore does not guarantee) time-slicing. However, some systems on which you can run Java do support time-slicing. Your Java programs should not rely on time-slicing as it may produce different results on different systems.

## Thread Priorities

- Thread priority can be used by the scheduler to decide which thread is to be run
- The priority of a Thread can be set using the method **setPriority()**
  - `t.setPriority(7);`
- The method **getPriority()** will return the priority of a Thread
  - `t.getPriority()`
- The priority can vary from 1 to 10 or Thread.MIN\_PRIORITY to Thread.MAX\_PRIORITY
- Normally a Thread will have the priority 5 or Thread.NORM\_PRIORITY



Copyright © 2005, Infosys Technologies Ltd

16

Infosys®

Java assigns to each thread a priority that determines how that thread should be treated with respect to each others. Thread priorities are integers that specify the relative priority of one thread to another. As an absolute value, a priority is meaningless; a higher –priority thread doesn't run any faster than a lower-priority thread if it is the only thread running.

Instead, a thread's priority is used to decide when to switch from one running thread to the next. This is called a **context switch**.



## Ensuring time-slicing (1 of 2)

- The programmer should write code to ensure that time-slicing occurs even when the application is running in a preemptive environment
- The static method `sleep` of the `Thread` class will make a thread sleep for specified number of milliseconds  

```
Thread.sleep(1000);
```
- A sleeping thread can be interrupted by another thread using the method `interrupt()`
- When interrupted, sleep method will throw an ***InterruptedException***



Copyright © 2005, Infosys  
Technologies Ltd

17

Infosys®

***Thread.sleep()*** and other methods that can pause a thread for periods of time can be interrupted. Threads can call another thread's ***interrupt()*** method, which signals the paused thread with an `InterruptedException`.

## Ensuring time-slicing (2 of 2)

- The `yield` method of the class `Thread` will give a chance to other threads to run
  - `t.yield();`
- The `yield()` method ensures that the thread behaves like a polite thread and not a selfish thread



Copyright © 2005, Infosys  
Technologies Ltd

18

Infosys®

### ***yield():***

It is used to give the other threads of the same priority a chance to execute. If other threads at the same priority are runnable, ***yield()*** places the calling thread in the running state into the runnable pool and allows another thread to run. If no other threads are runnable at the same priority, ***yield()*** does nothing.

## isAlive() and join() methods

- The state of a thread can be queried using the isAlive() method
  - Returns true if the thread has been started but not completed its task
  - A thread can wait for another thread to finish by using the join() method



Copyright © 2005, Infosys  
Technologies Ltd

19

Infosys®

Thread can be in unknown state.

**isAlive()** method is used to determine if a thread is still alive. The term alive does not imply that the thread is running; it returns true for a thread that has been started but not completed its task.

***final boolean isAlive()***

The method that you will more commonly use to wait for a thread to finish is called join(), shown here:

***final void join() throws InterruptedException***

This method waits until the thread on which it is called terminates

## Thread synchronization

- In a multithreaded environment 2 or more threads may access a shared resource
- There should be some means to ensure that the shared resource is accessed by only one thread at a time. This is termed as **synchronization**
- Every such shared object has a mutually exclusive lock called **monitor**
- Only one thread can get the monitor of an object at a time



Copyright © 2005, Infosys  
Technologies Ltd

20

Infosys®

## Thread synchronization

- Synchronization can be ensured by using the keyword ***synchronized***
- A method or block of code can be made synchronized
- Example

```
public class Account{  
    int bankBalance;  
  
    public synchronized void creditAccount (int amount) {  
        bankBalance+= amount;  
    }  
}
```



## Inter-thread communication (1 of 3)

- Thread must wait for the state of an object to change while executing a synchronized method, it may call wait()
  - void wait()
  - void wait (long timeout)
- The thread calling wait will release the lock on that particular object
- The thread will wait till it is notified by another thread owning the lock on the same object
  - void notify()
  - void notifyAll()



Copyright © 2005, Infosys  
Technologies Ltd

22

Infosys®

Threads also provide a secondary benefit: they do away with polling. Polling is usually implemented by a loop that is used to check some condition repeatedly. Once the condition is true, appropriate action is taken. So, in this case the thread needs to repeatedly check if the monitor lock is available in order to enter a synchronized method thereby wasting CPU cycles

To avoid polling, java includes an elegant inter-thread communication mechanism via the wait(), notify(), and notifyAll() methods. These methods belong to the Object class and can be called from within synchronized context only.

## Inter-thread communication (2 of 3)

- If there are multiple threads waiting on the same object, the notify() method will notify one among them
- The thread that would be notified cannot be predicted
- It is safer to call notifyAll(), since it will notify all waiting threads
- Code for synchronization should be written carefully else may lead to a **dead-lock** situation



Copyright © 2005, Infosys Technologies Ltd

23

Infosys®

wait() tells the calling thread to give up the monitor until some other thread enters the same monitor and calls notify().

notify() wakes up the one thread that called wait() on the same object (if there are more than one thread in wait state, then it is not known which thread would wake up.)

notifyAll() wakes up all the threads that called wait() on the same object.

## Inter-thread communication (3 of 3)

### Example

```
public class Account {  
  
    int bankBalance;  
    public synchronized void debitAccount (int amount) {  
        while((bankBalance - amount)<0) wait();  
        bankBalance -= amount;  
        ...  
    }  
    public synchronized void creditAccount (int amount) {  
        bankBalance += amount;  
        notify();  
        ...  
    }  
}
```



Copyright © 2005, Infosys  
Technologies Ltd

24

Infosys®

In the above code, the bankBalance is a sharable resource between threads that are executing debitAccount() and creditAccount() method. In case if one is operating and making the changes in the value of the same the other should not be allowed to access the bankBalance.

To take care of this both the methods have been declared as synchronized.

While a thread is inside a synchronized method, all the other threads that try to call it on the same instance have to wait.

The wait() method makes the thread to wait till there is some amount in the bankbalance to debit. Similarly the notify() method intimates the other thread that some amount is credited in the bankbalance so the amount could be debited.



## Thread Groups

- Represents a set of threads
- Can also contain other thread groups, creating a hierarchy of thread groups
- Provides a single-point control on the threads belonging to the thread group
- Creation time association is for the life time of the thread
  - `ThreadGroup threadGroup = new ThreadGroup("MyGroup");`
- Some of the important methods of ThreadGroup are
  - `activeCount()`, `destroy()`, `setMaxPriority()`, `setDaemon()`



Copyright © 2005, Infosys Technologies Ltd

25

Infosys®

The ThreadGroup class is used to manage a group of threads as a single unit. This provides you with a means to finely control thread execution for a series of threads. Thread groups can also contain other thread groups, allowing for a nested hierarchy of threads. Another benefit to using thread groups is that they can keep threads from being able to affect other threads, which is useful for security.

*/\*demo for depicting various methods of threads\*/*

```
class ThreadMethodsDemo
```

```
{
```

```
    public static void main(String args[]) throws InterruptedException  
    {
```

```
        Thread t = Thread.currentThread();
```

```
        // threadname,priority,threadgroup
```

```
        System.out.println("Current Thread : " + t);
```

```
        System.out.println("Name of thread : " + t.getName());
```

```
        System.out.println("Priority : " + t.getPriority());
```

```
        System.out.println("Active count : " +
```

```
Thread.activeCount());
```

```
        System.out.println("Thread Group : " +
```

```
t.getThreadGroup());
```

```
        System.out.println("Is Daemon ? " + t.isDaemon());
```

```
        System.out.println("Is Alive ? " + t.isAlive());
```

```
        //IllegalArgumentException is thrown here as the value
```

```
of priority is not in range 1 to 10
```

```
        //t.setPriority(11);
```

## Daemon Thread

- Daemon Thread is a thread that will run in the background, serving other threads
- So, a program can stop, if all non-daemon threads have finished execution
- A thread can be made a daemon by calling the method ***setDaemon(true)*** before calling its start() method
- Can query thread status using the method ***isDaemon()***



Copyright © 2005, Infosys  
Technologies Ltd

26

Infosys®

The system itself always has a few daemon threads running, one of which is the garbage collector.

## Summary

### Multithreading

- Creating and managing threads
- Priority management
- Thread synchronization
- Thread groups and daemon threads



Copyright © 2005, Infosys  
Technologies Ltd

27

Infosys®



## Unit - 2

# Input Output Stream & Serialization



## Input and Output

- The Java Input/Output system is designed to make it device independent
- The classes for performing input and output operations are available in the package `java.io`



Copyright © 2005, Infosys  
Technologies Ltd

29

Infosys®

## Streams

- Streams are channels of communication between programs and source/destination of data
  - A stream is either a source of bytes or a destination for bytes.
- Provide a good abstraction between the source and destination
- Abstract away the details of the communication path from I/O operation
- Streams hide the details of what happens to the data inside the actual I/O devices.
- Streams can read/write data from/to blocks of memory, files and network connections



Copyright © 2005, Infosys  
Technologies Ltd

30

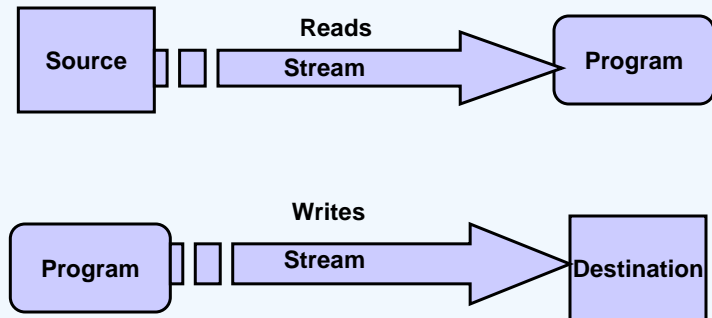
Infosys®

A stream is a path of communication between the source of some information and its destination. The source and the destination can be a file, the computer's memory, or even the Internet. An input stream allows you to read data from a source, and an output stream allows you to write data to a destination.

The foundations of this stream framework in the Java class hierarchy are abstract classes for doing input and output operation. Since the streams are abstract, the same I/O classes and methods can be applied to any type of I/O devices. All the I/O classes are in the io package.

Java works with two types of I/O streams i.e. the Byte-Oriented streams and the Character-Oriented streams.

## Streams(Contd...)



Copyright © 2005, Infosys  
Technologies Ltd

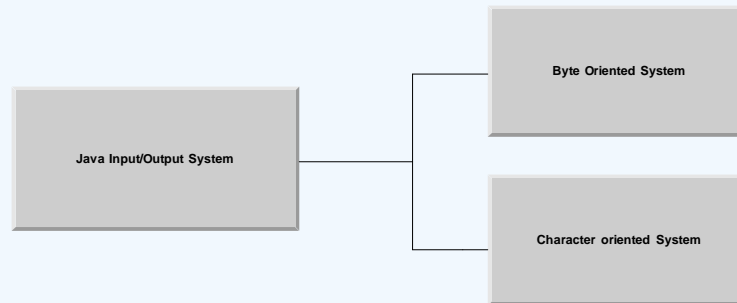
31

Infosys®

## Java's Input-Output System

• The basic I/O system of Java can handle only two types of data

- Bytes
- Characters



Copyright © 2005, Infosys Technologies Ltd

32

Infosys®



## Byte I/O

### Byte-Oriented System

- The following are the two abstract classes provided for reading and writing bytes

- InputStream - Reading bytes
- OutputStream - Writing bytes

- For performing I/O operations, we dependent on the sub classes of these two classes



Copyright © 2005, Infosys  
Technologies Ltd

33

Infosys®

A general stream, which receives and sends information as bytes, is called a byte-oriented stream. At the top of the hierarchy are the two abstract classes: `InputStream` and `OutputStream` for input and output respectively. These classes define many important methods for handling the streams. The `read()` and `write()` methods in the `InputStream` and `OutputStream` are used for reading and writing to and from streams respectively. Both these methods are abstract and are implemented by all the classes derived from these `InputStream` and `OutputStream` classes .

## Methods in InputStream

- `int available()`
- `void close()`
- `void mark(int numberOfBytes)`
- `boolean markSupported()`
- `int read()`
- `int read(byte buffer[])`
- `int read(byte buffer[], int offset, int numberOfBytes)`
- `void reset()`
- `long skip(long numberOfBytes)`



Copyright © 2005, Infosys  
Technologies Ltd

34

Infosys®

## Methods in OutputStream

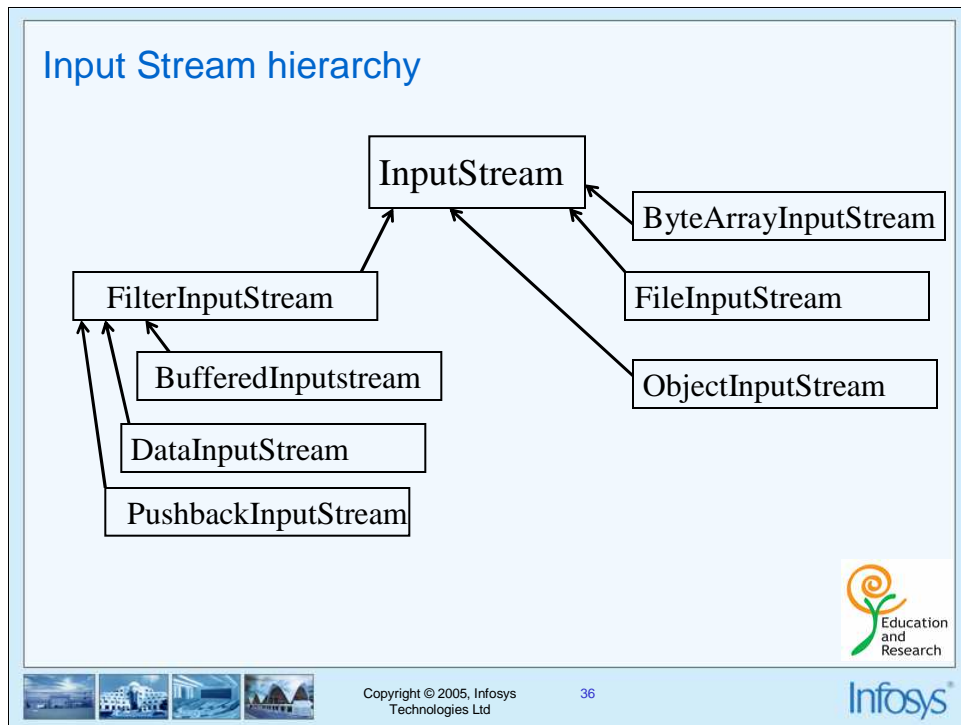
- ❏ `void close()`
- ❏ `void flush()`
- ❏ `void write(int b)`
- ❏ `void write(byte buffer[])`
- ❏ `void write(byte buffer[], int offset, int numberOfBytes)`



Copyright © 2005, Infosys  
Technologies Ltd

35

Infosys®



#### **ByteArrayInputStream**

To create an input stream *from* an array of bytes. Allows a buffer in memory to be used as an **InputStream**.

#### **FileInputStream**

For reading information from a file.

#### **ObjectInputStream**

It supports retrieving the stored state of an object.

**Filtered Streams:** Wrappers around underlying I/O streams that transparently provide some extended level of functionality to the other **InputStream** classes.

#### **BufferedInputStream**

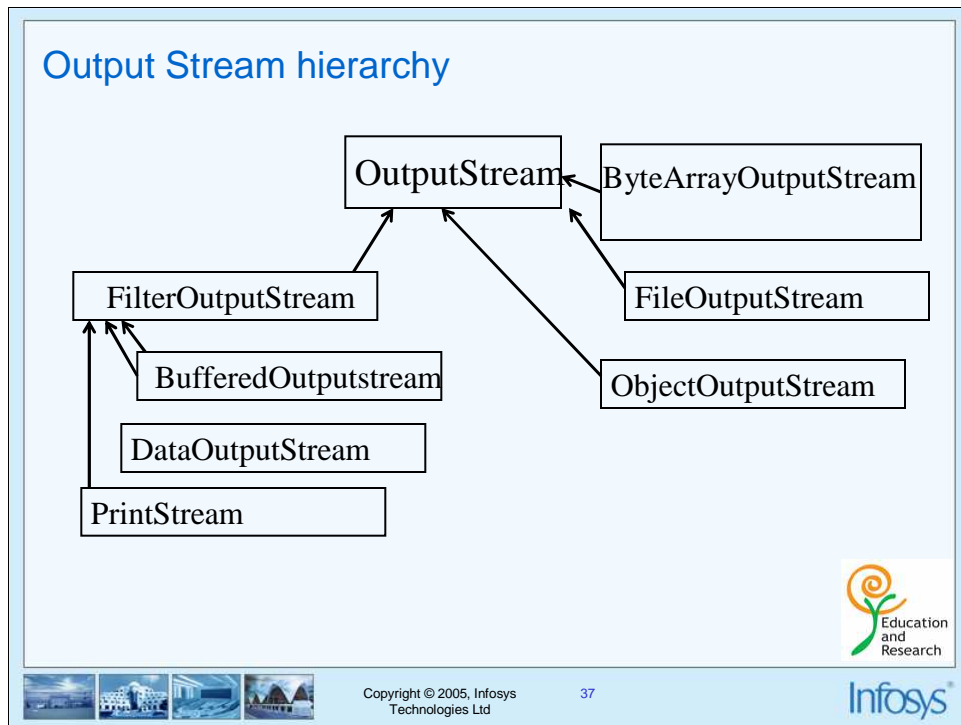
This is one of the most valuable of all streams. Using a buffer, it prevents a physical read every time when more data is needed. It uses buffered array of bytes that acts as a cache for future reading.

#### **DataInputStream**

Used to read primitives from a stream in a portable fashion. All the methods that instances of this class understand are defined in a separate interface, and implemented by this stream.

#### **PushbackInputStream**

Has a one byte push-back buffer with which it is possible to push back the last read character. The filter stream class **PushbackInputStream** is commonly used in parsers, to "push back" a single character in the input (after reading it)



### **ByteArrayOutputStream**

It is an implementation of an output stream that uses a byte array as the destination. The inverse of `ByteArrayInputStream`, which creates an input stream from an array of bytes, is `ByteArrayOutputStream`, which directs an output stream *into* an array of bytes:

### **FileOutputStream**

For writing information to a file.

### **ObjectOutputStream**

It supports permanently saving the state object serialization.

**Filtered Streams:** wrappers around underlying I/O streams that transparently provide some extended level of functionality to the other **OutputStream** classes.

### **BufferedOutputStream**

Using a buffer, it prevents a physical write every time data is sent.

### **DataOutputStream**

Used to write primitives to a stream in a portable fashion.

### **PrintStream**

For producing formatted output.

## Character I/O

### Character-Oriented System

- Since byte-oriented streams are inconvenient for Unicode-compliant character-based I/O, Java has the following abstract classes for performing character I/O
  - Reader class
  - Writer class
- Support internationalization of Java I/O
- Subclasses of Reader and Writer are used for handling Unicode characters



Copyright © 2005, Infosys Technologies Ltd

38

Infosys®

Streams, which receive and send information as characters are called character oriented streams. The abstract classes Reader and Writer are the base classes for all the character-oriented streams. Two of the most important methods are, read() and write(), which read and write characters of data, respectively. These methods are overridden by derived stream classes. Java supports internationalization and for that UNICODE character set is used

## Methods in Reader

- `void close()`
- `void mark( int )`
- `int read()`
- `int read(char[] c, int len)`
- `int read(char [] c, int off , int len)`
- `long skip(long num)`



Copyright © 2005, Infosys  
Technologies Ltd

39

Infosys®

## Methods in Writer

- abstract void close()
- abstract void flush()
- void write( int c)
- void write(char[] c, int off, int len)
- void write(char[] c, int len)
- void write(String str)
- void write(String str, int off, int len)

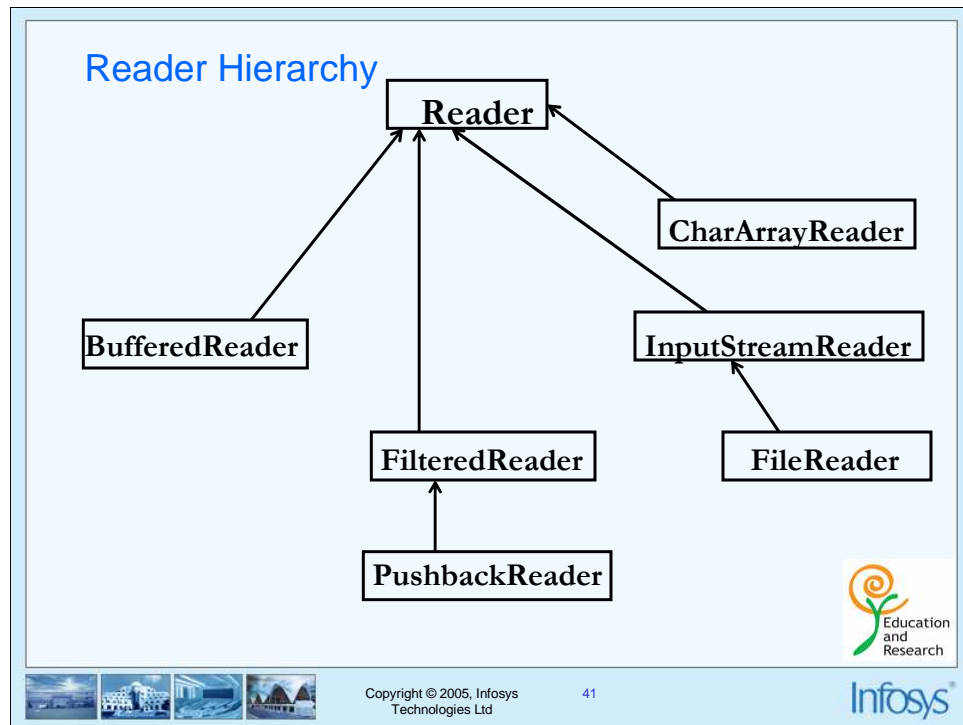


Copyright © 2005, Infosys  
Technologies Ltd

40

Infosys®





**BufferedReader:** Buffered input character stream

**InputStreamReader:** Input stream that translates bytes to characters

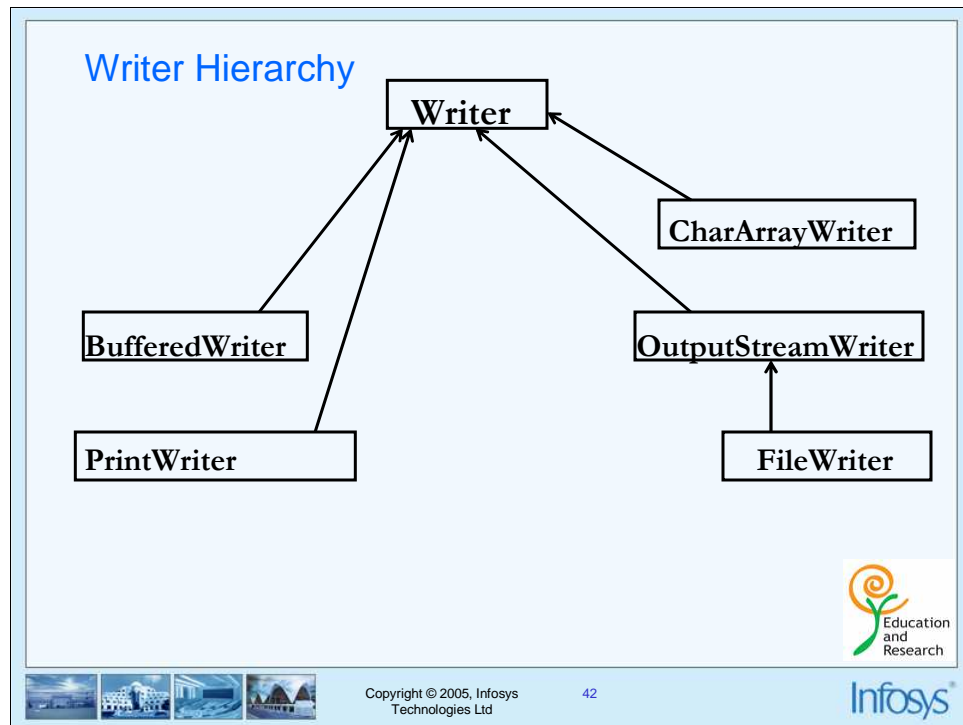
**FilterReader :**Filtered reader

**PushbackReader :** Input stream that allows characters to be returned to the input stream

**CharArrayReader :** Input stream that reads from a character array

**InputStreamReader :** Input stream that translates bytes to characters

**FileReader :**Input stream that reads from a file



**PipedWriter:** Output pipe

**BufferedWriter:** Buffered Output character stream

**OutputStreamWriter:** Output stream that translates characters to bytes

**FilterWriter :**Filtered writer

**CharArrayWriter :** Output stream that writes to a character array

**FileWriter :**Output stream that writes to a file

## InputStream – The read() method (1 of 2)

- The read() method reads a byte from the *input device* and returns an integer
- It returns an integer value -1 when the read fails
  - The read() method of FileInputStream returns -1 when it reaches end of file condition
- In case of a successful read, the returned integer can be typecast to a byte to get the data read from the device



Copyright © 2005, Infosys  
Technologies Ltd

43

Infosys®

## InputStream – The read() method (2 of 2)

- ❏ Why is the byte embedded in an int?
- ❏ Storing the byte into the low end of an integer, makes it possible for the program to distinguish between -1 integer representing end of data condition and -1 byte, a data stored in the device
- ❏ On reading a data of -1 byte successfully from a device, the following integer is returned
  - 00000000 00000000 00000000 11111111
  - The integer value of this is 255
  - This can be typecast to 11111111 which is the real data, -1 byte
- ❏ When a read fails, the following integer is returned
  - 11111111 11111111 11111111 11111111
  - The integer value is -1 indicating end of data



Copyright © 2005, Infosys Technologies Ltd

44

Infosys®

## Device Independent I/O (1 of 2)

- The way in which I/O is performed is not changing with the device
- The following method will write three bytes into **any** output device

```
public void writeBytes(OutputStream outputStream){  
    try{  
        outputStream.write(65);  
        outputStream.write(66);  
        outputStream.write(67);  
    }  
    catch(IOException exception){  
        System.out.println(exception);  
    }  
}
```



Copyright © 2005, Infosys  
Technologies Ltd

45

Infosys®

## Device Independent I/O (2 of 2)

- ❏ `ByteWriter byteWriter = new ByteWriter();`
- ❏ `//Write to the monitor`
- ❏ `byteWriter.writeBytes(System.out);`
- ❏ `//Write to an array`
- ❏ `byteWriter.writeBytes(byteArrayOutputStream);`
- ❏ `//Write to a file`
- ❏ `byteWriter.writeBytes(fileOutputStream);`



Copyright © 2005, Infosys  
Technologies Ltd

46

Infosys®

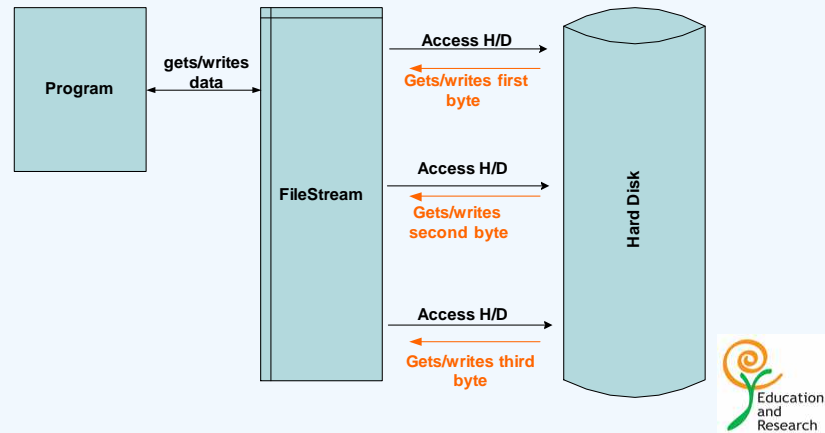
## Filter Classes

- The Java I/O System uses filter classes to build specialized I/O streams that provide particular capabilities as needed.
- Typically an instance of a lower level class is created and then it is wrapped inside more specialized stream instances by passing it to the wrapper via a constructor argument.
- Typical extensions are buffering, character translation, and raw data translation.



## Non-Buffered IO

- Reading and writing a byte at a time can be expensive in terms of processing time



Copyright © 2005, Infosys Technologies Ltd

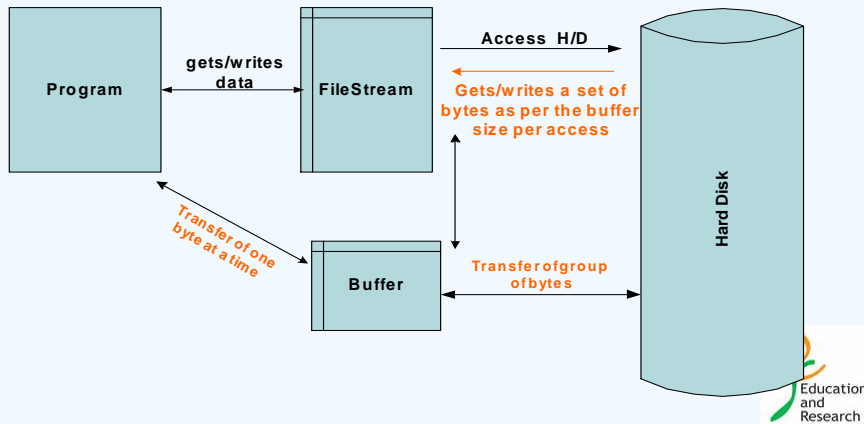
48

Infosys®



## Buffered IO

- Buffering helps java to store an entire block of values into a buffer
- It never has to go back to the source and ask for more, unless it runs out



Copyright © 2005, Infosys Technologies Ltd

49

Infosys<sup>®</sup>

Education  
and  
Research

## Example-Buffering (1 of 2)

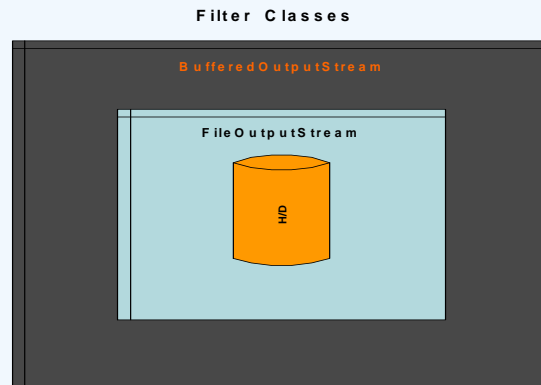
- The following code snippet writes bytes to a file using buffering mechanism

```
FileOutputStream fileOutputStream=new FileOutputStream("Data");  
BufferedOutputStream bufferedOutputStream=new BufferedOutputStream  
    (fileOutputStream);  
  
byte data1 = 65, data2 = 66, data3 = 67;  
  
bufferedOutputStream.write(data1);  
bufferedOutputStream.write(data2);  
bufferedOutputStream.write(data3);  
  
bufferedOutputStream.close();
```



## Example-Buffering (2 of 2)

- The device (Hard Disk) is wrapped in a `FileInputStream` and the `FileInputStream` is in turn wrapped in `BufferedInputStream`



Copyright © 2005, Infosys  
Technologies Ltd

51

Infosys®

Here we first wrap a `FileSteam` with a `BufferedStream`. *Buffered* classes improve the performance of I/O by providing intermediate data storage buffers. The data must fill the buffer to a certain level before it is sent to the next stage, thus performing fewer time-consuming operations. Note that this can require the "flushing" of data near the end of the transmission when the data did not reach the level required for release.

In the above example, we wrap the `FileOutputStream` with a `BufferedOutputStream`.

## DataInputStream and DataOutputStream

- ④ InputStream and OutputStream deal with bytes only
- ④ DataInputStream and DataOutputStream are filter classes that wrap around InputStream and OutputStream objects so that primitive types can be read and written
- ④ DataInputStream has the methods
  - readInt
  - readDouble
  - readBoolean
  - readXXX
- ④ DataOutputStream has the methods
  - writeInt
  - writeDouble
  - writeBoolean
  - writeXXX



Copyright © 2005, Infosys Technologies Ltd

52

Infosys®

The DataInputStream class provides the capability to read arbitrary objects and primitive types from an input stream. The filter provided by this class can be nested with other input filters.

The DataOutputStream class provides an output complement to DataInputStream. It allows arbitrary objects and primitive data types to be written to an output stream. It also keeps track of the number of bytes written to the output stream. It is an output filter and can be combined with any output-filtering streams.

## PushbackInputStream

- *Pushback* is used on an input stream for reading a byte and then pushing it back to the stream as if it is unread
- The filter class **PushbackInputStream** provides the following methods for pushing back
  - void unread(int *bytevalue*)
  - void unread(byte *buffer*[])
  - void unread(byte *buffer*[], int *offset*, int *numBytes*)



Copyright © 2005, Infosys  
Technologies Ltd

53

Infosys®

When reading an input, you often need to peek at the next byte to see if it is the value that you expect. Java provides `PushbackInputStream` for this purpose.

The first form of **`unread()`** method pushes back the low order byte of *bytevalue*. This will be the next byte returned by call to **`read()`**.

The second form of **`unread()`** returns the bytes in the buffer.

The third form of **`unread()`** pushes back *numBytes* bytes beginning at offset from buffer.

## PrintStream

- This filter class provides functionality to print all data types.
- It provides two methods, `print()` and `println()`, that are overloaded to print any primitive data type or object.
- Objects are printed by first converting them to strings using their `toString()` method inherited from the `Object` class.
- `PrintStream` class has the following constructor
  - `PrintStream(OutputStream outputStream)`
- `System.out` is a `PrintStream` object



Copyright © 2005, Infosys  
Technologies Ltd

54

Infosys®

**PrintStream** objects support the **print()** and **println()** methods for all types.

## Character I/O

- **Reader** and **Writer** classes are to character I/O what **InputStream** and **OutputStream** classes are to byte I/O
- Classes corresponding to byte stream classes are available for character I/O
  - `FileOutputStream` – `FileWriter`
  - `FileInputStream` – `FileReader`
  - `BufferedOutputStream` – `BufferedWriter`
  - `BufferedInputStream` – `BufferedReader`
  - `ByteArrayOutputStream` – `CharArrayWriter`
  - `ByteArrayInputStream` – `CharArrayReader`
  - etc...



## “Bridge” Classes

- Two classes act as bridge between byte streams and characters
  - InputStreamReader
  - OutputStreamWriter
- InputStreamReader wraps around an InputStream object and helps to read characters from an InputStream
- OutputStreamWriter wraps around an OutputStream object and helps to write characters into an OutputStream





## What is Serialization?

### Persistence:

- The capability of an object to exist beyond the execution of the program which created it.
- In other words, saving the state of an object in some permanent storage device, such as file



Copyright © 2005, Infosys  
Technologies Ltd

57

Infosys®

If the object exists beyond the execution of the program that created it then it is known as persistence. Serialization is the key factor for implementing persistence. Serialization allows to store objects in the files, communicate them across networks and use them in distributed applications.

## Serialization Mechanism

- Serializable objects can be converted into stream of bytes
- This stream of bytes can be written into a file
- These bytes can be read back to re-create the object
- Only those objects that implements `java.io.Serializable` interface can be serialized



Copyright © 2005, Infosys  
Technologies Ltd

58

Infosys®

## ObjectOutputStream and ObjectInputStream

- Filter classes that help to write and read Serializable objects
- ObjectOutputStream wraps around any OutputStream and helps to write a Serializable object into the OutputStream
  - ObjectOutputStream(OutputStream outputStream)
- The method writeObject(Object object) will write the object to the underlying OutputStream object
- ObjectInputStream wraps around any InputStream and helps to read a stream of bytes as an Object from the InputStream
  - ObjectInputStream(InputStream inputStream)
- The method readObject() will read a stream of bytes, convert them into an Object and return it



Copyright © 2005, Infosys Technologies Ltd

59

Infosys®

## Security: an issue in serialization

- Serialized objects can be sent over network
- Can be accidentally or deliberately modified
- Also sensitive data can be read

### Solution

- Encrypt the object during serialization using Security API
- Ensure that sensitive objects do not implement Serializable or Externalizable



Copyright © 2005, Infosys  
Technologies Ltd

60

Infosys®

The digital signature of an object can be created using the methods of security API and stored with the object

## Summary

- IO Streams in Java
  - overview of fundamental streams
  - Some advanced streams
- Serialization



Copyright © 2005, Infosys  
Technologies Ltd

61

Infosys®

Infosys®

POWERED BY INTELLECT  
DRIVEN BY VALUES



## Unit -3 JDBC



## JDBC

- The JDBC (Java Database Connectivity) API helps a **Java program to access a database in a standard way**
- JDBC is a **specification** that tells the database vendors how to write a driver program to interface Java programs with their database



Copyright © 2005, Infosys  
Technologies Ltd

63

Infosys®

## JDBC

- A Driver written according to this standard is called the **JDBC Driver**
- All related classes and interfaces are present in the **java.sql** package
- All JDBC Drivers **implement the interfaces** of java.sql



Copyright © 2005, Infosys  
Technologies Ltd

64

Infosys®



## JDBC Drivers

- There are 4 types of drivers --Type1, Type2, Type3, Type4

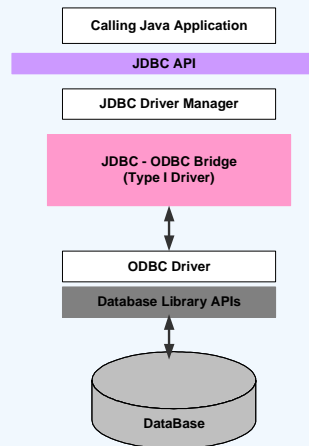


Copyright © 2005, Infosys  
Technologies Ltd

65

Infosys®

## Type1 Driver (JDBC-ODBC bridge driver)



Copyright © 2005, Infosys Technologies Ltd

66

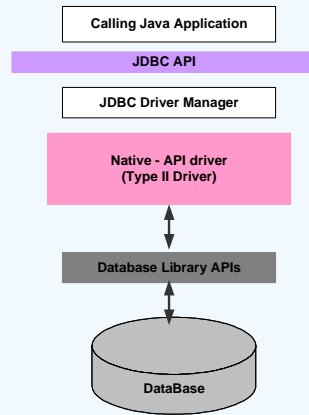
Infosys®

The **JDBC type 1 driver**, also known as the **JDBC-ODBC bridge** is a database driver implementation that uses the ODBC. The driver converts JDBC method calls into ODBC function calls. The bridge is usually used when there is no pure-Java driver available for a particular database.

The driver is implemented by the `sun.jdbc.odbc.JdbcOdbcDriver` class and comes with the Java 2 SDK, Standard Edition.

The driver is platform-dependent as it makes use of ODBC which in turn depends on native libraries of the operating system. Also, using this driver has got other dependencies such as ODBC must be installed on the computer having the driver and the database which is being connected to must support an ODBC driver. Hence the use of this driver is discouraged if the alternative of a pure-Java driver is available.

## Type2 Driver (Native-API driver )



Copyright © 2005, Infosys Technologies Ltd

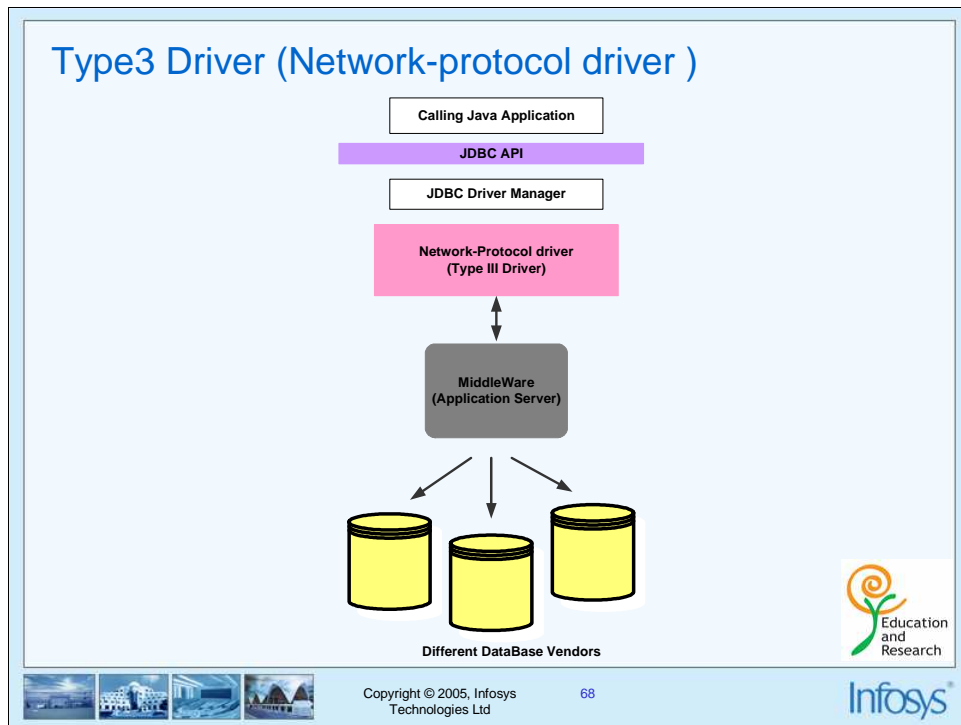
67

Infosys®

The **JDBC type 2 driver**, also known as the **Native-API driver** is a database driver implementation that uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API.

The type 2 driver is not written entirely in Java as it interfaces with non-Java code that makes the final database calls. The driver is compiled for use with the particular operating system. For platform interoperability, the Type 4 driver, being a full-Java implementation, is preferred over this driver.

However the type 2 driver provides more functionality and performance than the type 1 driver as it does not have the overhead of the additional ODBC function calls

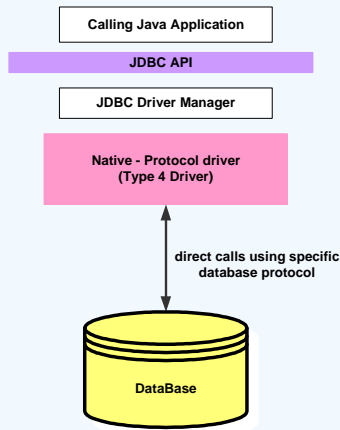


The **JDBC type 3 driver**, also known as the **network-protocol driver** is a database driver implementation which makes use of a middle-tier between the calling program and the database. The middle-tier (application server) converts JDBC calls directly or indirectly into the vendor-specific database protocol.

This differs from the type 4 driver in that the protocol conversion logic resides not at the client but in the middle-tier. However, like type 4 drivers, the type 3 driver is written entirely in Java.

The same driver can be used for multiple databases. It depends on the number of databases the middleware has been configured to support. The type 3 driver is platform-independent as the platform-related differences are taken care by the middleware. Also, making use of the middleware provides additional advantages of security and firewall access.

## Type4 Driver (Native-protocol driver )



Copyright © 2005, Infosys Technologies Ltd

69


Infosys®

The **JDBC type 4 driver**, also known as the **native-protocol driver** is a database driver implementation that converts JDBC calls directly into the vendor-specific database protocol.

The type 4 driver is written completely in Java and is hence platform independent. It provides better performance over the type 1 and 2 drivers as it does not have the overhead of conversion of calls into ODBC or database API calls. Unlike the type 1 and 2 drivers, it does not need associated software to work.

As the database protocol is vendor-specific, separate drivers, usually vendor-supplied, need to be used to connect to the database. Performance wise this is the most efficient

## Database interaction

 The steps involved in a database interaction are:

- **Loading** the specific driver
- Making a **connection** to the database
- **Sending** SQL statements to the database
- **Processing** the results



Copyright © 2005, Infosys  
Technologies Ltd

70

Infosys®

## JDBC - classes and interfaces

### **DriverManager class :**

- Manages all the JDBC Drivers that are loaded in the memory
- Helps in dynamic loading of Drivers

### **Data Source :**

- Offer the user considerably more capability than the basic Connection objects that the DriverManager provides.
- It supports connection pooling and distributed transactions



Copyright © 2005, Infosys  
Technologies Ltd

71

Infosys®

The DriverManager class provides methods to obtain a connection to a database through a driver, register and deregister drivers, set up logging, and set login timeouts for database access. When you request a connection it returns an object of the Connection interface. The Connection object is then used to create SQL statements and handle ResultSets.

The additional features of DataSource objects make it a preferred means of getting a Connection to any source of data. This source can be anything from a relational database to a spreadsheet or a file in tabular form.

## JDBC - classes and interfaces

Three types of standard DataSource objects :

- The basic DataSource that produces standard Connection objects just like those that the DriverManager produces
- A PooledDataSource that supports connection pooling. Pooled connections are returned to a pool for reuse by another transaction.
- A DistributedDataSource that supports distributed transactions accessing two or more DBMS servers.



Copyright © 2005, Infosys Technologies Ltd

72

Infosys®

With connection pooling, connection can be used over and over again, avoiding the overhead of creating a new connection for every database access.

Distributed transactions involve tables on more than one database server

A DataSource object is normally registered with a JNDI naming service.



## JDBC - classes and interfaces

### Methods in **DriverManager** class -

- getConnection() : to establish a connection to a database.
  - Connection getConnection(String url, Properties info)
  - Connection getConnection(String url)
  - Connection getConnection(String url, String userID, String password)
- registerDriver(java.sql.Driver)



Copyright © 2005, Infosys  
Technologies Ltd

73

Infosys®

The methods present in 'java.sql.DriverManager' class are:

**public static synchronized Connection getConnection(String url, Properties info) throws SQLException**

This method and other **getConnection()** methods attempts to establish a connection to the given database URL and attempts to select an appropriate driver from the set of registered JDBC drivers. 'info' is a reference to a Properties container object of tag and value pairs, typically username and password.

The example below shows the use of the method.

```
/* initialise the Properties object */
Properties prop = new Properties();
/* create two properties for the Properties object */
prop.put("user", "");
prop.put("password, "");
/* create a DSN by name test for a database and then use the method
getConnection() to get a connection by passing the DSN name 'test'
Properties object prop */
and DriverManager.getConnection("jdbc:oracle:thin:@IP of the
Oracle server:host string", prop);
```

## JDBC - classes and interfaces (Contd...)

Please refer to Notes page for more explanation on previous slide



Copyright © 2005, Infosys  
Technologies Ltd

74

Infosys®

**public static synchronized Connection getConnection(String url  
,String user, String password) throws SQLException**

In this method, the along with the URL, the user Id and password can be passed directly but not a very good way because mutiple changes need to be made when there are changes in userId and password

```
DriverManager.getConnection("jdbc:oracle:thin:@IP of the Oracle server:host  
string", "", "");
```

**public static synchronized void registerDriver(java.sql.Driver  
driver) throws SQLException**

This method is invoked by Drivers to register themselves with DriverManager. This is generally used when you have a custom driver to connect to the database. The following example illustrates the use of the method.

*/\* myDriver is a custom driver written in Java. How to write a driver is beyond the scope of this session \*/*

```
DriverManager.registerDriver(Driver myDriver);  
public static void setLogStream(java.io.PrintStream out) throws  
SQLException
```

## JDBC - classes and interfaces (Contd...)

 **Connection interface** - defines methods for interacting with the database via the established connection.

- A connection object represents a connection with a database.
- A connection session includes the SQL statements that are executed and the results that are returned over that connection.
- A single application can have one or more connections with a single database, or it can have many connections with many different databases.



Copyright © 2005, Infosys  
Technologies Ltd

75

Infosys®

To execute a SQL statement you need to create an object of the Statement (any of the three types) by invoking the appropriate method of the Connection interface. You should then use the method of the Statement interface to execute the SQL query. The method may return a 'ResultSet' object, integer value telling number of lines modified or updated by the statement, etc.

## JDBC - classes and interfaces (Contd...)

 The different methods of Connection interface are:

- **close()** - closes the database connection
- **createStatement()** - creates an SQL Statement object
- **prepareStatement()** - creates an SQL PreparedStatement object.  
(PreparedStatement objects are precompiled SQL statements)
- **prepareCall()** - creates an SQL CallableStatement object using an SQL string.  
(CallableStatement objects are SQL stored procedure call statements)



Copyright © 2005, Infosys  
Technologies Ltd

76

Infosys®

## Statement

A statement object is used to send SQL statements to a database.

Three kinds :

- **Statement**
  - Execute simple SQL without parameters
- **PreparedStatement**
  - Used for pre-compiled SQL statements with or without parameters
- **CallableStatement**
  - Execute a call to a database stored procedure or function



Copyright © 2005, Infosys  
Technologies Ltd

77

Infosys®

SQL is the language used to interact with the database. To retrieve or insert, update and delete data into a database you need to execute SQL statements. Java provides three interfaces, i.e. the Statement interface, the PreparedStatement interface and the CallableStatement interface to achieve this.

**Java Provides three interfaces to execute SQL statements:**

Statement interface

PreparedStatement interface

CallableStatement interface

## JDBC - classes and interfaces (Contd...)

- ④ **Statement interface** - defines methods that are used to interact with database via the execution of SQL statements.
- ④ The different methods are:
  - **executeQuery(String sql)** - executes an SQL statement (SELECT) that queries a database and returns a ResultSet object.
  - **executeUpdate(String sql)** - executes an SQL statement (INSERT, UPDATE, or DELETE) that updates the database and returns an int, the row count associated with the SQL statement
  - **execute(String sql)** - executes an SQL statement that is written as String object
  - **getResultSet()** - used to retrieve the ResultSet object



Copyright © 2005, Infosys Technologies Ltd

78

Infosys®

An example of the usage of the Statement interface and its methods is shown here using the 'Employee' table and 'emp' DSN. The 'Employee' table consists of three fields namely 'empcode', 'empname' and 'empage'.

```
Connection connection = DriverManager.getConnection("jdbc:odbc:emp", "", "");
```

```
/* create statement */
```

```
Statement statement = connection.createStatement();
```

```
/* get all records from the Employee table */
```

```
ResultSet resultSet = statement.executeQuery("select * from Employee");
```

```
/* update the age in empcode 1 and check no of records affected by change */
```

```
String sql = "update Employee set empage = 25 where empcode = 1";
```

```
int recordsAffected = stmt.executeUpdate(sql);
```

```
if(recordsAffected == 0)
```

```
    System.out.println("Update failed");
```

```
/* delete employee record with empcode = 2 */
```

```
String sql = "delete from employee where empcode = 2";
```

```
int recordsAffected = statement.executeUpdate(sql);
```

```
/* we have to commit the transaction once we delete the record, otherwise the record is just
```

```
marked for deletion but not physically deleted. we achieve this by using the commit() method of the Connection interface */
```

```
connection.commit();
```

## JDBC - classes and interfaces (Contd...)

- ④ **ResultSet Interface** - maintains a pointer to a row within the tabular results.

The **next()** method is used to successively step through the rows of the tabular results.

- ④ The different methods are:

- **getBoolean(int)** - Get the value of a column in the current row as a Java boolean.
- **getByte(int)** - Get the value of a column in the current row as a Java byte.
- **getDouble(int)** - Get the value of a column in the current row as a Java double.
- **getInt(int)** - Get the value of a column in the current row as a Java int.



Copyright © 2005, Infosys Technologies Ltd

79

Infosys®

Database query results are retrieved as a ResultSet object

The ResultSet maintains the position of the current row, starting with the first row of data returned.

When a database query is executed, the results of the query are returned as a table of data organised in rows and columns. The 'ResultSet' interface gives access to this tabular format of data. Database query results are retrieved as a ResultSet object. The ResultSet maintains the position of the current row, starting with the first row of data returned.

Every method which you use to access the database would throw an SQL Exception so, you have to include the code in a try and catch block or declare that the method throws SQLException.

The following example illustrates the use of the ResultSet interface and its methods.

```
Connection connection = DriverManager.getConnection("jdbc:odbc:emp", "", "");
Statement statement = connection.createStatement();
ResultSet resultSet = statement.executeQuery("select * from Employee");
/* Employee table has three columns first is a code (number), second is name (String) and
third is age (number)*/
while(resultSet.next()){
    int code = rs.getInt(1);
    String name= rs.getString(2);
    int age = rs.getInt(1);
    System.out.println("Code : "+ code + "Name: " + name + " Age : " + age);
}
resultSet.close();
connection.close();
```

## JDBC - classes and interfaces (Contd...)

### Types of ResultSet Objects

- **ScrollableResultSet**

It supports the ability to move a result set's cursor in either direction and there are methods for getting the cursor position and moving the cursor to a particular row.

- **Updatable ResultSets**

This allows to make the updates to the values in the ResultSet itself, and these changes are reflected in the database.



Copyright © 2005, Infosys  
Technologies Ltd

80

Infosys®

To create a Scrollable Resultset, we need to call the createStatement method with two arguments passed, the first specify the type of the ResultSet object.

The second argument must be one of the two ResultSet constants for specifying whether a result set is read-only or updateable.

To create an UpdatableResultSet object, we need to call the createStatement method with the ResultSet constant CONCUR\_UPDATABLE as the second argument. The Statement object created produces an updatable ResultSet object when it executes a query.

Once you have an UpdatableResultSet object, you can insert a new row, delete an existing row, or modify one or more column values.



## JDBC - classes and interfaces (Contd...)

### Cursor control methods

- next()
- previous()
- first()
- last()
- beforeFirst()
- afterLast()
- absolute(int rowNum)
- relative(int rowNum)



Copyright © 2005, Infosys  
Technologies Ltd

81

Infosys®

In addition to the `ResultSet.next()` method, which is used to move the cursor forward, one row at a time, scrollable `ResultSets` support the method `previous()` which moves the cursor back one row.

The affect of the `first()`, `last()`, `beforeFirst()` and `afterLast()` is apparent from the method names.

The method `absolute(int number)` moves the cursor to the row number indicated in the argument.

The method `relative(int rowNum)` lets you specify how many rows to move from the current row and in which direction to move. A positive number moves the cursor forward the given number of rows, a negative number moves the cursor backward the given number of rows.

## Using Statement and ResultSet

```
import java.sql.*;
class JDBCtest{
    public static void main(String args[]) {
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection connection =
                DriverManager.getConnection("jdbc:oracle:thin:@DB,
                    IPaddress:port_no:host string","uid","password");
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery("select * from
                Student");

            while(resultSet.next()){
                System.out.println(resultSet.getInt("ClassNo"));
            }
        }
        catch(Exception exception) {
            System.out.println(exception)
        }
    }
}
```

tion  
rch



Copyright © 2005, Infosys  
Technologies Ltd

82

Infosys®

## JDBC - classes and interfaces (Contd...)

- ④ ***PreparedStatement interface*** -- helps us to work with ***precompiled SQL statements***
- ④ Precompiled SQL statements are ***faster*** than normal statements
- ④ So, if a SQL statement is to be repeated, it is better to use PreparedStatement
- ④ Some values of the statement can be represented by a ***?*** character which can be replaced later using ***setXXX*** method



Copyright © 2005, Infosys  
Technologies Ltd

83

Infosys®

## Using PreparedStatement

```
import java.sql.*;

class PreparedStatementTest{

    public static void main(String args[]) throws Exception{

        try{

            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection connection = DriverManager.getConnection("url",
                                                                "UID", "password");

            PreparedStatement preparedStatement =
            connection.prepareStatement("select * from Emp where ename=?");
            preparedStatement.setString(1,str);
            ResultSet resultSet = preparedStatement.executeQuery();
            while(resultSet.next()){
                System.out.println(resultSet.getString("ename"));
            }


        }
        catch(Exception exception){
            System.out.println(exception);
        }

    }

}
```



## JDBC - classes and interfaces (Contd...)

 **CallableStatement interface** -- helps us to call stored procedures and functions

```
CallableStatement callableStatement = connection.prepareCall("execute proc ?");  
callableStatement.setInt(1,50);  
callableStatement.execute();
```



Copyright © 2005, Infosys  
Technologies Ltd

85

Infosys®

## JDBC - classes and interfaces (Contd...)

- The **out** parameters are to be registered

```
callableStatement.registerOutParameter(int parameterIndex, int SQLType);
```

- To get the value stored in the out parameter--

```
callableStatement.getXXX(int parameterIndex);
```



Copyright © 2005, Infosys  
Technologies Ltd

86

Infosys®

## Using CallableStatement

- Example - Calling a stored procedure named GetSalary. The procedure queries on the Employee table and returns the salary of an employee. It has one input parameter that takes the EmpCode and an out parameter that returns the salary

```
CallableStatement callableStatement =  
    connection.prepareCall("begin GetSalary(?,?); end;");  
callableStatement.setInt(1,29418);  
// OUT parameters must be registered.  
callableStatement.registerOutParameter(2, Types.DOUBLE);  
callableStatement.execute();  
System.out.println("Salary : " + callableStatement.getDouble(2));
```



## JDBC - classes and interfaces (Contd...)

⑩ **ResultSetMetaData Interface** - holds information on the types and properties of the columns in a ResultSet. Provides information about the database as a whole. Constructed from the Connection object

⑩ The different methods are:

- **getColumnName()**
- **getColumnType()**
- **getColumnLabel(count)**



Copyright © 2005, Infosys  
Technologies Ltd

88

Infosys®

```
/* To create an object of ResultSetMetaData */
```

```
ResultSetMetaData resultSetMetaData = resultSet.getMetaData();
```

```
/* use methods getColumnCount() and getColumnName() to get the number of  
columns present in the table and display each column name */
```

```
for(int i = 0; i < resultSetMetaData.getColumnCount(); ++i)
```

```
    System.out.println(resultSetMetaData.getColumnName(i);
```

```
resultSet.close();
```

```
connection.close();
```



## Transaction Management using JDBC

### Transactions

- The capability to group SQL statements for execution as a single entity is provided through SQL's transaction mechanism.
- A transaction consists of one or more statements that are executed, completed and either committed or rolled back as a group.
- The commit means that the change is made permanently in the database, and the term rollback means that no change is made in the database.



Copyright © 2005, Infosys  
Technologies Ltd

89

Infosys®

When the method commit or rollback is called, the current transaction ends, and another one begins.

A new JDBC connection is in auto-commit mode by default, meaning that when a statement is completed, the method commit is called on that statement automatically.

The commit occurs when the statement completes or the next execute occurs, whichever comes first.

## Transaction Management using JDBC

- By default, auto commit mode of the connection reference is set to true
- A transaction can be done as follows using methods of the Connection interface

```
...  
connection.setAutoCommit(false);    //by default it is true  
  
try{  
    //Statements  
    connection.commit();  
}  
catch(Exception exception){  
    connection.rollback();  
}
```

Education  
and  
Research



Copyright © 2005, Infosys  
Technologies Ltd

90

Infosys®

If auto-commit mode has been disabled, a transaction will not terminate until either the commit method or the rollback method is called explicitly, so it will include all the statements that have executed since the last invocation of the commit or rollback method.

In this case, all the statements in the transaction are committed or rolled back as a group.

## Transaction Management using JDBC

### Transaction Isolation Levels

This allows to manage simple conflicts arising from events such as a failure to complete a linked series of SQL commands.

- The current level of isolation can be queried using the method :

```
public int getTransactionIsolation()
```

- The control over the isolation level of a connection is provided by this method:

```
setTransactionIsolation( TRANSACTION_ISOLATION_LEVEL_XXX).
```



Copyright © 2005, Infosys  
Technologies Ltd

91

Infosys®

Consider the case of a multiuser application, where one transaction has initiated a transfer between two accounts but has not yet committed it, when a second transaction attempts to access one of the account in question.

If the first transaction is rolled back, the value the second transaction reads will be invalid.

JDBC defines levels of transaction isolation that provides different levels of conflict management. The lowest level specifies that transaction are not supported at all, and the highest specifies that while one transaction is operating on a database, no other transactions may make any changes to the data that transaction reads.

## Transaction Savepoints

- During a transaction, a named savepoint may be inserted between operations to act as a marker, so that the transaction may be rolled back to that marker, leaving all of the operations before the marker in effect.
- Transaction savepoints are JDBC enhancements that offer finer control over transaction commit and rollback.

```
con.setAutoCommit(false);  
Statement stmt = con.createStatement();  
stmt.executeUpdate(update1);  
Savepoint savepoint1 = con.setSavepoint("SavePoint1");  
stmt.executeUpdate(update2);  
stmt.executeUpdate(update3);  
  
con.rollback(savePoint1);
```



Copyright © 2005, Infosys  
Technologies Ltd

92

Infosys®

The above code shows a Savepoint being set after the first update, and the transaction being rolled back to that Savepoint, removing two subsequent updates.

When a SQL statements make changes to a database, the commit method makes those changes permanent, and it releases any locks the transaction holds.

The rollback method, on the other hand, simply discards those changes.

## Summary

### **Java Data Base Connectivity**

- JDBC API
- Different drivers
- Set up a connection to a database from Java
- Create a database application



Copyright © 2005, Infosys  
Technologies Ltd

93

Infosys®



## Unit -4 Java Remote Method Invocation



## Objective

### Remote Method Invocation

- Need for RMI
- Access to Remote Objects
- RMI APIs



Copyright © 2005, Infosys  
Technologies Ltd

95

Infosys®

## Remote Method Invocation

### Examples of Use

- Database access
- Computations
- Any custom protocol
- Not for standard protocols (HTTP, FTP, etc.)



Copyright © 2005, Infosys  
Technologies Ltd

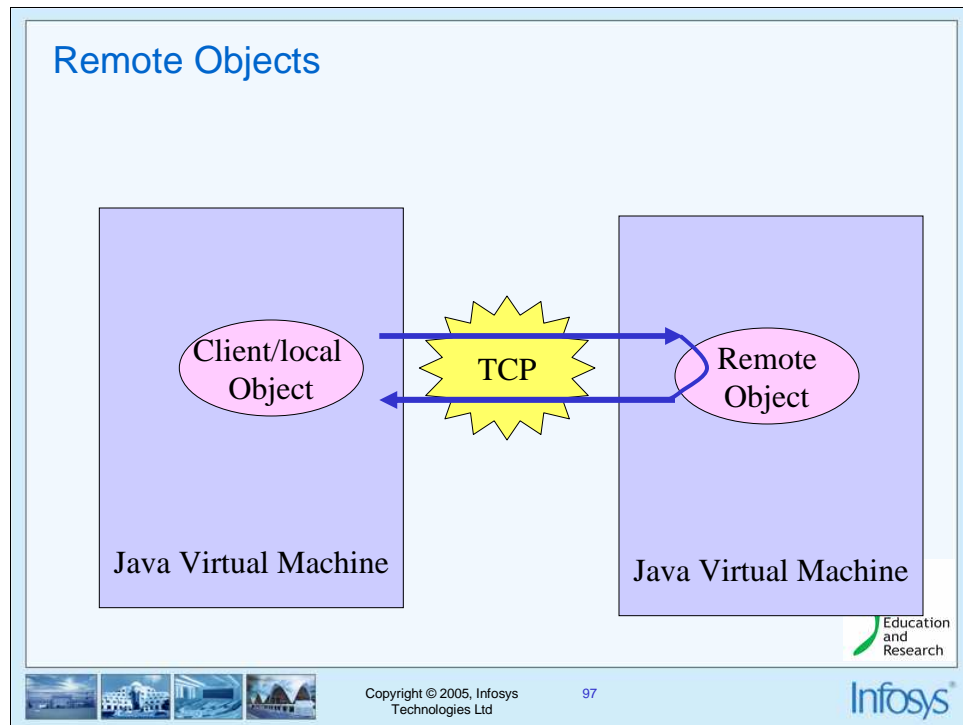
96

Infosys®

When the processing is distributed across the multiple networked computers then it is called as “distributed application”.

In Java RMI feature allows us to invoke a method on an object which resides in another JVM.





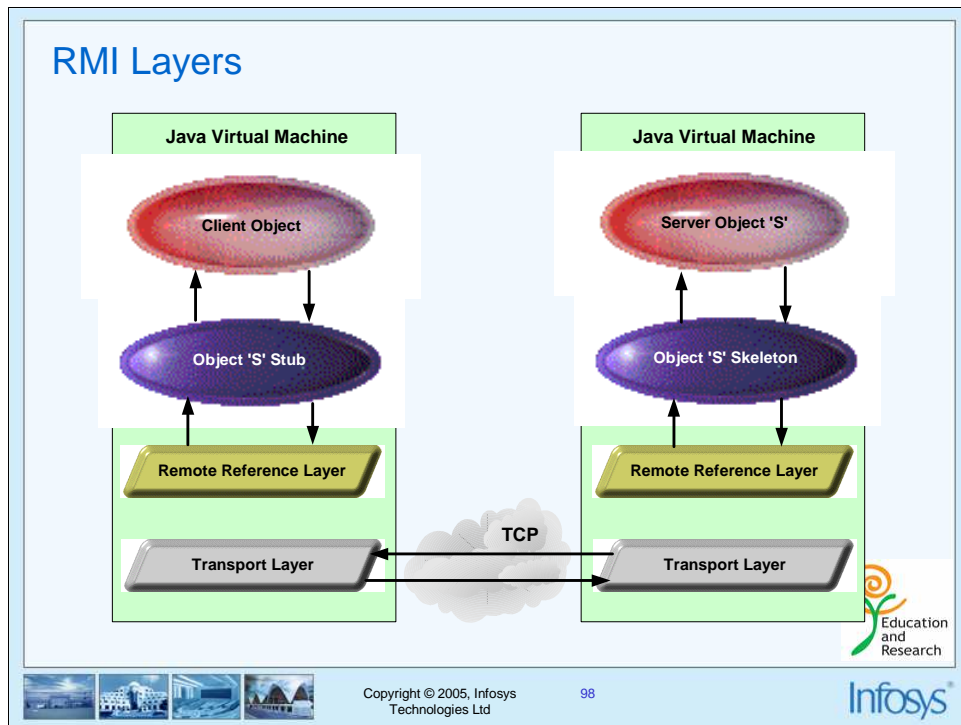
The [RMI \(Java Remote Method Invocation\)](#) is a mechanism that enables an object on one JVM to invoke methods on an object in another JVM.

A '**distributed application**' is an application whose processing is distributed across multiple networked computers. Distributed architecture is based on the three-tier application development, where the front end, namely the user interface is on one network and the business rules and the database on a different network.

The models for developing distributed applications that exist in the industry today are DCOM, RMI and CORBA.

All these three methods of developing distributed applications allow objects on one host to invoke methods of objects on other computers or even computers on a different network. DCOM and CORBA are standards of developing distributed applications and so can be developed in any language, whereas RMI is specific to developing distributed applications in Java.

The distributed object model used by Java allows objects in one JVM to invoke methods of objects in a separate JVM and this is known as Remote Method Invocation (RMI). These separate JVMs may execute as a different process in the same computer or on a remote computer.



### Remote Reference Layer

A client invoking a method on a remote server object actually uses a stub or proxy as a conduit to the remote object. A client-held reference to a remote object is a reference to a local stub, which is an implementation of the remote interfaces of the object and which forwards invocation requests to it via the remote reference layer. Stubs are generated using the `rmic` compiler.

The remote reference layer in the RMI system separates out the specific remote reference behavior from the client stub. Any call initiated by the stub is done directly through the reference layer, enabling appropriate reference semantics to be carried out. For example, if a remote reference had the behavior of connection retry, it would attempt to establish the retry connection on behalf of the stub. When the connection initiation succeeded, the reference layer would forward the marshaling of arguments and send the RMI call to the underlying transport for that reference type.

The remote reference layer transmits data to the transport layer via the abstraction of a stream-oriented *connection*. The transport takes care of the implementation details of connections. Although connections present a streams-based interface, a connectionless transport can be implemented beneath the abstraction.

### Transport Layer

The transport layer is responsible for connection setup, connection management, and keeping track of and dispatching to remote objects (the targets of remote calls) residing in the transport's address space.

In order to dispatch to a remote object, the transport forwards the remote call up to the remote reference layer. The remote reference layer handles any server-side behavior that needs to occur before handing off the request to the server-side skeleton. The skeleton for a remote object makes an up call to the remote object implementation which carries out the actual method call.

The return value of a call is sent back through the skeleton, remote reference layer, and transport on the server side, and then up through the transport, remote reference layer, and stub on the client side.

In general, the transport layer of the RMI system is responsible for:

- Setting up connections to remote address spaces.

- Managing connections.

- Monitoring connection "liveness."

- Listening for incoming calls.

- Maintaining a table of remote objects that reside in the address space.

- Setting up a connection for an incoming call.

- Locating the dispatcher for the target of the remote call and passing the connection to this dispatcher.

## General RMI architecture

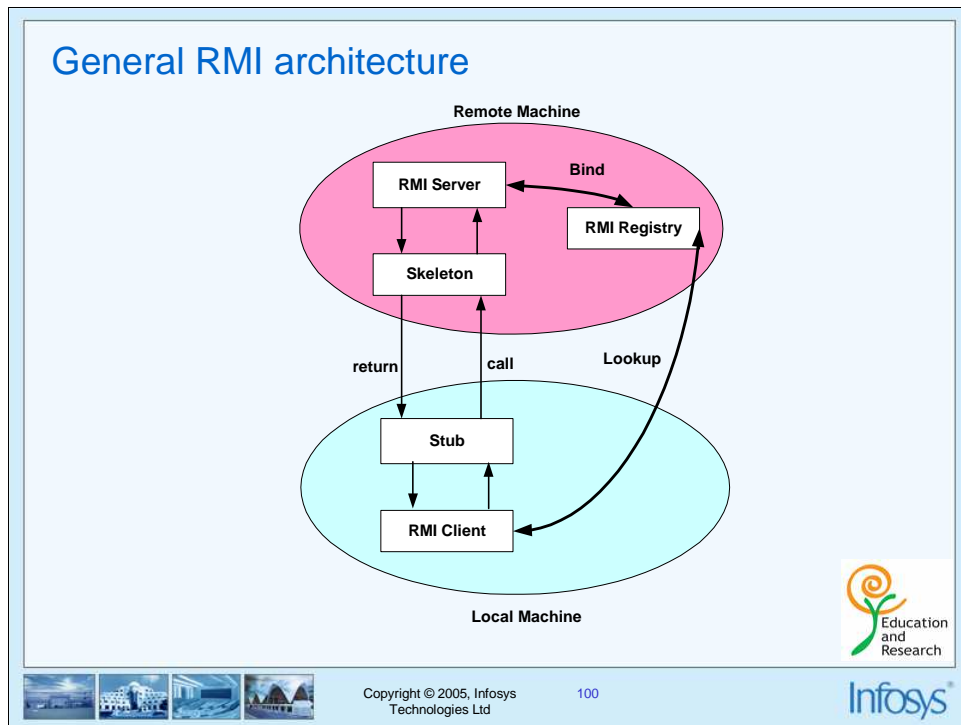
- ④ The server must first bind its name to the **registry**
- ④ The client **lookup** the server name in the registry to establish remote references.
- ④ The **Stub** serializes the parameters to **skeleton**, the skeleton invokes the remote method and serializes the result back to the stub.



Copyright © 2005, Infosys  
Technologies Ltd

99

Infosys®



### RMI Registry

The registry is a simple server that enables an application to look up objects that are being exported for remote method invocation.

The registry simply keeps track of the addresses of remote objects that are being exported by their applications

All objects are assigned unique names that are used to identify them.

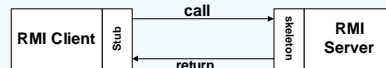
A server registers one of its objects with a registry by calling a `bind()` or `rebind()` method on a registry instance, passing it a `String` that uniquely identifies the object and a reference to an instance of the object that is being exported.

Since all objects must have a unique name, a `bind()` call to a registry that contains a name `String` that is already registered will result in an exception being thrown.

Alternatively, `rebind()` replaces an old object with a given name with a new object.

## Stub and skeleton

- ❶ A client invokes a remote method, *the call is first forwarded to stub.*
- ❷ The stub is responsible for sending the remote call over to the server-side skeleton
- ❸ The stub opening a socket to the remote server, marshaling the object parameters and forwarding the data stream to the skeleton.
- ❹ A skeleton contains a method that receives the remote calls, unmarshals the parameters, and invokes the actual remote object implementation.



Copyright © 2005, Infosys Technologies Ltd

101

Infosys®

### STUB

Is the client side proxy for the remote object.

Applications interface to the remote object

Responsible for initiating a call

Marshaling arguments to a marshal stream (marshal stream is used to transport parameters, exceptions and errors needed for method dispatch and returning the results)

pretends to be remote object

Informing the remote reference layer that the call should be invoked.

Upon return:

Unmarshaling the return value or exception from a marshal stream.

Informing the remote reference layer that the call is complete.

### SKELETON

lives on server

receives requests from stub

talks to true remote object

delivers response to stub

## Steps to develop a RMI system

- 1. Define the remote interface
- 2. Develop the remote object (Server object) by implementing the remote interface.
- 3. Develop the client program.
- 4. Compile the Java source files.
- 5. Generate the client stubs and server skeletons.
- 6. Start the RMI registry.
- 7. Start the remote server objects.
- 8. Run the client



## Step1: Defining remote interface

- ☛ To create a RMI application, the first step is defining of a **remote interface** between the client and server objects.

```
/* InterfaceRMI.java */  
  
import java.rmi.*;  
  
public interface InterfaceRMI extends Remote{  
  
    int cube(int x) throws RemoteException;  
  
}
```



Copyright © 2005, Infosys  
Technologies Ltd

103

Infosys®

The Remote interface must have the following properties:

The interface must be public.

The interface must extend the interface java.rmi.Remote.

Every method in the interface must declare that it throws java.rmi.RemoteException.

Other exceptions may also be thrown

## Step2: Develop the server object

☉ The server is a simple unicast remote server.

☉ Create server by extending

`java.rmi.server.UnicastRemoteObject`

```
/*ServerRMI.java*/
import java.rmi.*;
import java.rmi.server.*;
public class ServerRMI extends UnicastRemoteObject implements InterfaceRMI {

    public ServerRMI() throws RemoteException {
        super();
    }
    ...
}
```

Education  
and  
Research



Copyright © 2005, Infosys  
Technologies Ltd

104

Infosys®

***UnicastRemoteObject.exportObject(new ServerRMI,0);***

***this can be used in case the server already extends some other class and we want that class to be the server be the server class.***

### ***The java.rmi.server Package***

This package implements several interfaces and classes that support both client and server aspects of *RMI* like a class implementation of *Remote* interface, client *stub*, server *skeleton* and so on. The explanation for each is given below

The '***RemoteObject***' class implements the *Remote* interface. So generally all objects that need to implement the *Remote* interface extend this class.

The '***RemoteServer***' class extends the *RemoteObject* and provides general functions of a server like ***setLog()***, ***getLog()***, and ***getClientHost()***. ***setLog()*** and ***getLog()*** are used to log information about *RMI* accesses and ***getClientHost()*** is used to get the host name of the client.

'***UnicastRemoteObject***' extends the *RemoteServer* to provide default remote object implementation. Classes that implement the remote object usually extend this class.

The '***RemoteStub***' class extends the *RemoteObject* and provides a general implementation of client side stubs. It provides a static method '***setRef()***' which is used to associate a client with its corresponding remote object.

Remote ***skeletons*** implement the '*Skeleton*' interface. This interface provides methods for *skeletons* to access remote methods.

The '***RMIClassLoader***' class supports the loading of classes remotely. The location of the class is specified as an URL. The static ***loadClass()*** method is used to load the remote class.

The ***RMI SocketFactory*** class is used to specify a socket implementation for transporting information between clients and servers. The '***setSocketFactory()***' method can be used to specify a custom socket implementation.



## Step2 contd...

- ☛ The server class needs to implement the remote methods

```
public int cube(int x) throws RemoteException{  
    return x*x*x;  
}
```



Copyright © 2005, Infosys  
Technologies Ltd

105

Infosys®

## Step2 contd...

- ④ The server must **bind its name** to the registry, the client will look up the server name.
- ④ Use **java.rmi.Naming** class to bind the server name to **registry**. In this example the name by which it is registered is "MyServer".

```
...
public static void main (String a[]){
    try {
        ServerRMI remoteServer = new ServerRMI();
        Naming.rebind("MyServer", remoteServer );
        System.out.println("System is ready");
    }catch(Exception e){
        System.out.println(e);
    }
}
}
```



Copyright © 2005, Infosys  
Technologies Ltd

106

Infosys®

### Create a Class that Implements the Remote Interface

The classes that implements the remote interface generally extends the 'UnicastRemoteObject' class because RMI presently supports only unicast objects.

The implementation class will have a constructor, which will create and initialise the remote object. It also implements all the methods present in the remote interface. It has a 'main()' function so that remote client objects can invoke the class.

The method 'main()' sets the object to be used as the remote object's security manager by using the method 'setSecurityManager()' of the 'System' class. The example below illustrates using the *setSecurityManager()* method.

**System.setSecurityManager(new RMISecurityManager());**

The class is registered with the *Remote registry* using the 'rebind()' method of the 'Naming' class. The example below illustrates the class registered with the *Remote registry* using the *rebind()* method.

**ServerExImpl instance = new ServerExImpl();**

**Naming.rebind("//"+hostName+"/ServerExample", instance);**

where:

**ServerExImpl:** Is the class name of the program implementing the remote interface.

**HostName:** Is the name of the host machine containing the class.

**ServerExample:** Is the name used by the client to call the remote object.

## Step 3: Develop the client program

- 1 In order for the client object to invoke methods on the server, it must first **look up** the name of server object in the registry.
- 2 Use the **`java.rmi.Naming`** class to lookup the server name.
- 3 The server name is specified as URL in the form( `rmi://host:port/name` )
- 4 Default RMI port is **1099**.



Copyright © 2005, Infosys  
Technologies Ltd

107

Infosys®

### Step 3 contd...

- The name specified in the URL must exactly match the name with which the server object has bound to the registry. In this example, the name is **"MyServer"**
- The remote method invocation is programmed using the remote interface ***reference\_name.function\_name***



### Step 3 contd...

```
/*ClientRMI.java*/
import java.rmi.*;

public class ClientRMI {
    public static void main(String a[]) {
        try{
            InterfacRMI obj=(InterfacRMI)
                               Naming.lookup("//localhost/MyServer");
            int no=obj.cube(4);
            System.out.println("The value is : " + no);
        }
        catch(Exception e){
            System.out.println(e);
        }
    }
}
```

Education  
Research



Copyright © 2005, Infosys  
Technologies Ltd

109

Infosys®

## Step 4: compiling all the source files

Assume that the code being compiled and run from C:\ in the command prompt

Compile all the 3 source codes using

**C:\javac \*.java**



Copyright © 2005, Infosys  
Technologies Ltd

110

Infosys®

## Step 5: generating stubs and skeleton

Now the stub and skeleton need to be generated using *rmic* (rmi compiler)

```
C:\> rmic ServerRMI
```



Copyright © 2005, Infosys  
Technologies Ltd

111

Infosys®

## Step 6: starting the RMI registry

• The rmiregistry must be started using

C:\ **start rmiregistry**

• This step needs to be done before the server program starts as the server object has to be registered



Copyright © 2005, Infosys  
Technologies Ltd

112

Infosys®



## Step 7: start the server program

- Once the Registry is started, the server can be started and will be able to store itself in the Registry
- Start the server program

C:\ ***java ServerRMI***



Copyright © 2005, Infosys  
Technologies Ltd

113

Infosys®

## Step 8: start the client program

• The client program can now access the server program

C:\ ***java ClientRMI***

• For the client it seems as if the function call is on the same machine



Copyright © 2005, Infosys  
Technologies Ltd

114

Infosys®

## Summary

### Remote Method Invocation

- Need for RMI
- Access to Remote Objects
- RMI APIs
- Client-Server Demo



Copyright © 2005, Infosys  
Technologies Ltd

115

Infosys®

Infosys®

POWERED BY INTELLECT  
DRIVEN BY VALUES



## Unit 5

### Java Beans



## Objective

- Reflection API
- Introducing Java Beans
- Simple Java Bean
- Why Java Bean
- Java Beans in User Interface
- Naming conventions



Copyright © 2005, Infosys  
Technologies Ltd

117

Infosys®

## The Reflection API

- Reflection is the ability of software to analyze itself
- The `java.lang.reflect` package provides reflection capability to a Java program
- The `getClass()` method of the class `Object` returns a `Class` object
- The `getConstructors()`, `getFields()` and `getMethods()` are used to analyze the class object



Copyright © 2005, Infosys  
Technologies Ltd

118

Infosys®

Reflection is an important capability, needed when using components called Java Beans. It allows you to analyze a software component and describe its capabilities dynamically, at run time rather than at compile time.

For example, by using reflection, you can determine what methods, constructors, and fields a class supports.

## Class Objects

- A Class object can be retrieved in several ways.
- If an instance of the class is available, `Object.getClass` is invoked.
- The following line of code gets the Class object for an object named `mystery`:  
`Class c = mystery.getClass();`
- If we need to retrieve the Class object for the superclass that another Class object reflects, invoke the `getSuperclass` method.
- If the name of the class is known at compile time, its Class object can be retrieved by appending `.class` to its name.
- If the class name is unknown at compile time, but available at runtime, you can use the `forName` method



Copyright © 2005, Infosys Technologies Ltd

119

Infosys®

The `getClass` method is useful when you want to examine an object but you don't know its class.

In the following example, `getSuperclass` returns the Class object associated with the `TextComponent` class, because `TextComponent` is the superclass of `TextField`:

```
TextField t = new TextField();  
Class c = t.getClass();  
Class s = c.getSuperclass();
```

In the following example, the Class object that represents the `Button` class is retrieved by appending `.class` to its name:

```
Class c = java.awt.Button.class;
```

In the following example, if the String named `strg` is set to `"java.awt.Button"` then `forName` returns the Class object associated with the `Button` class

```
Class c = Class.forName(strg);
```

## Class Constructors(1/2)

- The information about a class's constructors could be retrieved by invoking the `getConstructors` method, which returns an array of `Constructor` objects



Copyright © 2005, Infosys  
Technologies Ltd

120

Infosys®



## Class Constructors ( 2/2)

```
import java.lang.reflect.*;
import java.awt.*;

class SampleConstructor
{
    public static void main(String[] args)
    {
        Rectangle r = new Rectangle(); showConstructors(r);
        static void showConstructors(Object o)
        {
            Class c = o.getClass();
            Constructor[] theConstructors = c.getConstructors();
            for (int i = 0; i < theConstructors.length; i++)
            {
                System.out.print(" ");
                Class[] parameterTypes = theConstructors[i].getParameterTypes();
                for (int k = 0; k < parameterTypes.length; k++)
                {
                    String parameterString = parameterTypes[k].getName();
                    System.out.print(parameterString + " ");
                }
                System.out.println("");
            }
        }
    }
}
```



Copyright © 2005, Infosys  
Technologies Ltd

121

Infosys®

The above program prints out the parameter types for each constructor in the Rectangle class.

The program performs the following steps:

1. It retrieves an array of Constructor objects from the Class object by calling getConstructors.
2. For every element in the Constructor array, it creates an array of Class objects by invoking getParameterTypes. The Class objects in the array represent the parameters of the constructor.

The program calls getName to fetch the class name for every parameter in the Class array created in the preceding step

## Identifying Class Fields (1/2)

- A class's fields can be identified by invoking the `getFields` method on a `Class` object.
- The `getFields` method returns an array of `Field` objects containing one object per accessible public field.
- A public field is accessible if it is a member of either:
  - this class
  - a superclass of this class
  - an interface implemented by this class
  - an interface extended from an interface implemented by this class



## Identifying Class Fields (2/2)

```
import java.lang.reflect.*;
import java.awt.*;
class SampleField
{ public static void main(String[] args)
  { GridBagConstraints g = new GridBagConstraints();
    printFieldNames(g); }
  static void printFieldNames(Object o)
  { Class c = o.getClass();
    Field[] publicFields = c.getFields();
    for (int i = 0; i < publicFields.length; i++)
    { String fieldName = publicFields[i].getName();
      Class typeClass = publicFields[i].getType();
      String fieldType = typeClass.getName();
      System.out.println("Name: " + fieldName + ", Type: " + fieldType);
    }
  }
}
```



Copyright © 2005, Infosys  
Technologies Ltd

123

Infosys®

The above program prints the names and types of fields belonging to the GridBagConstraints class.

Note that the program first retrieves the Field objects for the class by calling getFields, and then invokes the getName and getType methods on each of these Field objects.

## What is a Java Bean?

### **Component**

- An artifact that is one of the individual parts of which makes a composite entity
- Usually a component is a part that can be separated from or attached to a system or module in a system

### **A Java bean is a reusable software component**

- Java Beans is the de-facto component model in Java world
- **Java Bean is NOT AN Enterprise Java Bean (EJB)**
  - EJB is covered in Introduction to J2EE

### **Various Frameworks and Tools can use Java Bean components to build something bigger**

- Example 1: An address component holds address of a person in a program
- Example 2: A text box is a reusable component of a screen



Copyright © 2005, Infosys  
Technologies Ltd

124

Infosys®

### **Java Beans:**

**Definition:** A Java Bean is a reusable software component that can be visually manipulated in builder tools. The JavaBeans API makes it possible to write component software in the Java programming language.

A Java Bean bears a strong resemblance to a well-written Java application. But since it is a component, a Bean's scope is usually smaller than that of an entire application, making it easier to develop.

## What is Java Beans? (Continued...)

④ Java Beans API provides a framework for defining software components that are

- **Reusable:** Component which is capable of being used again and again
- **Embeddable:** Component which is expected to function without human intervention.
- **Modular:** A solution made of several discrete pieces, so that any piece can be replaced without rebuilding the whole solution

④ Allows creation of reusable components

- Library of reusable components can be built
- Third Party library of components can be used to build an application

④ Frameworks and Builder tools which use Java Beans components use Java Reflection API to handle components



Copyright © 2005, Infosys Technologies Ltd

125

Infosys®

### Advantages of Java Beans:

The main purposes of Java Beans are closely related to the main advantages of Java technology: The advantages includes

- Platform independence
- Mobility in a networked environment.
- Expose accessor methods (e.g. `getValue()` and `setValue()`) to allow retrieval and changing of attribute values by external sources;
- Allow for easy mixing and matching via GUI development tools;
- Generate or respond to appropriate events;
- Save their state using the Java Serialization mechanism.
- Compact and Easy
- Leverages the Strengths of the Java Platform
- Flexible Build-Time Component Editors

## Simple Java Bean

### Employee Class

- Has all data members private
- Does not have a constructor
- A pair of public Get/Set methods for all or most member variables
- Names of Get/Set methods should match the variable names

### Results in an Employee Bean

- Has properties
  - employeeNumber
  - employeeName

```
class Employee {  
    private int employeeNumber;  
    private String employeeName;  
    ...  
    public void setEmployeeNumber  
        (int employeeNumber) {  
        this.employeeNumber=  
            employeeNumber;  
    }  
    public int getEmployeeNumber() {  
        return employeeNumber;  
    }  
    public void setEmployeeName  
        (String employeeName) {  
        this.employeeName=  
            employeeName;  
    }  
    public String getEmployeeName() {  
        return employeeName;  
    }  
    ...  
}
```



Copy  
Technologies Ltd

image

## Why Java Beans?

- Many real life situations require components to work with various frameworks and libraries
- Data in text files, XML files or other textual forms have to be mapped programmatically in many applications

- Names in text form can be mapped to variable names of a class
- Reflection API in Java is used to achieve this

- Example:

```
<Employee>
  <employeeNumber>29853</employeeNumber>
  <employeeName>Tom</employeeName>
  ...
</Employee>
```

- This is one of the reasons why Java Beans is used extensively in both UI frameworks and Server side code

- **Use of Java beans is more in Server side code than the GUI code**



## Properties

- Properties are Discrete attributes of a Bean that are referenced by name
  - Example: "EmployeeNumber" is a property of EmployeeBean
- Value of a property can be read programmatically by using a name



Copyright © 2005, Infosys  
Technologies Ltd

128

Infosys®

### Properties:

Properties are attributes of a Bean that are referenced by name. These properties are usually read and written by calling methods on the Bean specifically created for that purpose. A property of the thermostat component mentioned earlier in the chapter could be the comfort temperature. A programmer would set or get the value of this property through method calls, while an application developer using a visual development tool would manipulate the value of this property using a visual property editor.



## Where do we use Java Beans?

- Data handling and data intensive applications for mapping textual data to variables
- Server side applications which use component based design
- Web technologies and related frameworks like JSP, Struts etc
  - JSP Tag beans (covered in Introduction to J2EE course)
  - Struts is a framework for presentation layer of Java based web applications
- GUI applications based on Java Beans components
  - Rarely used because of limited scope of Java in GUI applications



Copyright © 2005, Infosys  
Technologies Ltd

129

Infosys®

## Java Beans in User Interface

• **BDK (Bean Development Kit)** was a specification from Sun for creating and using reusable UI based Java Beans

• **Difference in UI Java Beans compared to Simple Java Beans**

- Methods to draw on screen are provided
- Methods to persist data and settings
  - Persistence: storing data of an object in a non-volatile manner and retrieving it when required
- Event Handling methods to respond to user action on UI components
- Customization of component's features which are visible graphically
- Application Builder or similar GUI frameworks can understand and use these components to build GUI Screens



Copyright © 2005, Infosys Technologies Ltd

130

Infosys®

## Java Beans in User Interface (Continued...)

### • **Application Builder tool and other GUI Frameworks for using Java Beans**

- Use Reflection API and display all the properties and events offered by a Java Bean in a toolbar in the development environment
- Developer can manipulate properties and write event handler code in development environment
- Properties can also be manipulated programmatically



Copyright © 2005, Infosys  
Technologies Ltd

131

Infosys®

## Application Builder Tool

Java Bean components  
(Classified in tabs)

The screenshot shows the Mojo Designer application builder tool. The main window is titled "Client" and displays a design area with the text "Java Beans" in green. Below the text are three small icons: a blue cube, a red sphere, and a black triangle. To the right of the design area is a "Selected Components" panel showing a list of properties for the selected component, "MPanel1". The properties listed are: Top (120), Left (36), Height (147), Width (419), ForegroundColor (Color.black), BackgroundColor (Color.lightGray), FontName (Courier), FontSize (10), and FontStyle (Font.PLAIN). Below the properties list are two tabs: "Properties" and "Actions". The "Properties" tab is currently selected. At the bottom of the window, there is a status bar with the text "Screen1" and "ComplexBeanComponent Component".

Properties Window – Displays properties of selected bean

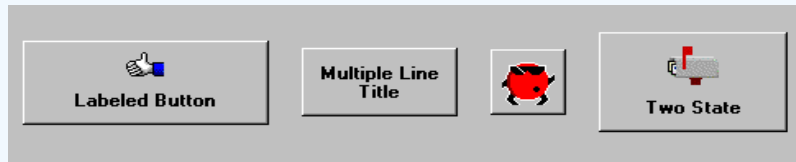
List of events supported

Copyright © 2005, Infosys Technologies Ltd

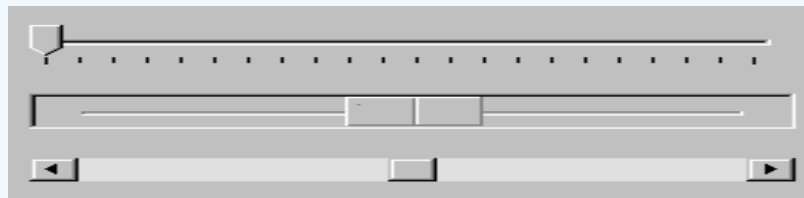
132

Infosys

## Sample Java Beans used in User Interface



**Button Bean**



**Slider Bean**



Copyright © 2005, Infosys  
Technologies Ltd

133

Infosys®

## Naming Conventions

### • Bean

- Class name: Any user defined name
- Should not have a constructor

### • Properties

- What makes up a property?
- Example:
  - A private data member called 'employeeNumber'
  - A public get method by name 'getEmployeeNumber'
  - A public set method by name 'setEmployeeNumber'
  - Together these three form a property of a bean called 'EmployeeNumber'
  - **Note:** Method names and variables names are case sensitive



Copyright © 2005, Infosys  
Technologies Ltd

134

Infosys®

The source code of a Java Bean is subject to certain naming conventions.

These naming conventions make it easier for beans to be reused, replaced and connected.

The naming conventions are:

- Every java bean class should implement `java.io.Serializable` interface
- It should have non parametric constructor
- Its properties should be accessed using get and set methods
- It should contain the required event handling methods

## Bean Serialization (1/2)

- ④ A bean has the property of persistence when its properties, fields, and state information are saved to and retrieved from storage.
- ④ The beans support serialization by implementing either the
  - `java.io.Serializable` interface, or the
  - `java.io.Externalizable` interface
- ④ These interfaces offer the choices of automatic serialization and customized serialization.
- ④ If any class in a class's inheritance hierarchy implements `Serializable` or `Externalizable`, then that class is serializable
- ④ Examples of serializable classes include `Component`, `String`, `Date`, `Vector`, and `Hashtable`



Copyright © 2005, Infosys  
Technologies Ltd

135

Infosys®

## Bean Serialization (2/2)

- The Serializable interface provides automatic serialization by using the Java Object Serialization tools.
- Serializable declares no methods; it acts as a marker, telling the Object Serialization tools that your bean class is serializable
- To exclude fields from serialization in a Serializable object from serialization, mark the fields with the transient modifier.

`transient int status;`

Default serialization will not serialize transient and static fields.



Copyright © 2005, Infosys  
Technologies Ltd

136

Infosys®



Infosys®

POWERED BY INTELLECT  
DRIVEN BY VALUES



## Unit 6

# Java Naming & Directory Interface



## Introduction

- Directory and Naming Services is used to organize information hierarchically to map human understanding of names and directory objects
- JNDI is an extension of Java
- JNDI provides Java application with a unified interface to multiple naming and directory services in the enterprise



Copyright © 2005, Infosys  
Technologies Ltd

138

Infosys®

In a distributed environment where the enterprise applications largely depends upon the services provided by other applications, locating such services is a difficult issue. JNDI provides common interface to many existing naming services like DNS, CORBA, NIS etc.

The Java Naming and Directory Interface (JNDI) is an application programming interface (API) for accessing different kinds of naming and directory services. JNDI is not specific to a particular naming or directory service, it can be used to access many different kinds of systems including file systems; distributed objects systems like CORBA, Java RMI, and EJB; and directory services like LDAP, Novell NetWare, and NIS+.

As part of the Java Enterprise API set, JNDI enables seamless connectivity to heterogeneous enterprise naming and directory services. Developers can now build powerful and portable directory-enabled applications using this industry standard.

## Naming Service

- Associates names with Objects
- Allows to lookup an Object by its name
- Binding : association of name with the object
- Context : a set of name-to-object bindings
- E.g.:
  - Phone book associates people's name with phone number and addresses
  - DNS maps human friendly URLs with IP addresses



Copyright © 2005, Infosys Technologies Ltd

139

Infosys®

More examples:

- when you use an electronic mail system, you must provide the name of the recipient to whom you want to send mail.
- To access a file in the computer, you must supply its name (A file system maps a filename (for example, c:\bin\autoexec.bat) to a file handle that a program can use to access the contents of the file)
- The Internet Domain Name System(DNS) maps machine names (such as www.sun.com) to IP addresses (such as 192.9.48.5).

A **naming service** allows you to look up an object given its name.

Following is the list of Naming Service Providers

**COS (Common Object Services) Naming:** The naming service for CORBA applications; allows applications to store and access references to CORBA objects.

**DNS (Domain Name System):** The Internet's naming service; maps people-friendly names (such as [www.etcee.com](http://www.etcee.com)) into computer-friendly IP (Internet Protocol) addresses in dotted-quad notation (207.69.175.36). Interestingly, DNS is a *distributed* naming service, meaning that the service and its underlying database is spread across many hosts on the Internet.

**LDAP (Lightweight Directory Access Protocol):** Developed by the University of Michigan; as its name implies, it is a lightweight version of DAP (Directory Access Protocol), which in turn is part of X.500, a standard for network directory services. Currently, over 40 companies endorse LDAP.

**NIS (Network Information System) and NIS+:** Network naming services developed

## Directory Service (1 of 2)

- ④ Extended naming service
- ④ Not only allows name to be associated with objects as in Naming service but also allows objects to have attributes associated with them
- ④ Directory Service = Naming Service + attributes of objects
- ④ E.g. Yellow Pages Directory system :
  - Associates Customer name with the actual telephone number/mobile number
  - Also provides extra information like address, occupation of customer as attributes
- ④ Directory object represents an object in the computing environment



Copyright © 2005, Infosys Technologies Ltd

140

Infosys®

A directory service can be viewed as a type of naming service augmented with the capability to perform more sophisticated searches for objects.

Directory service searching can be based on attributes of that object with search filter criteria and search controls. Directory services also allow for the modification of object attributes.

For example searching for a person object with name as “John” who lives in “NY Street”. Where John is the name of the object, and “NY Street” is associated attribute value.

## Directory Services (2 of 2)

- A set of connected objects forms a directory
- Directory service provides operation for creating, adding, removing, and modifying the attributes associated with objects in a directory
- Arrange the namespaces created in the Naming Services in a hierarchy
- Like the DOS file system; where the hierarchy starts from the root directory then the subdirectories and then the files.
- It also has attributes like the date, size of the file which gives us additional information.



Copyright © 2005, Infosys Technologies Ltd

141

Infosys®

Directory services typically have a hierarchical structure in which a directory object has a context in some directory structure.

Thus each context will contain zero or more sub contexts and zero or more named directory objects.

The directory object itself is manifested in a directory service as a collection of attributes describing this object.

Directory services are used in many distributed enterprise applications in which a set of distributed objects or information useful to other distributed objects may be registered, unregistered, and queried based on a set of descriptive attributes.

A directory service search for a particular user may first involve formulating a query given for a particular last name and geographical location such as city, state, and country.

The search result from this query may yield one or more directory objects matching this criteria, each with a collection of attributes.

## JNDI Architecture (1 of 2)

- 10 Java Naming and Directory Interface (JNDI) is an API that provides directory and naming functionality to Java applications
- 10 The JNDI architecture consists of an API and a service provider interface (SPI)
- 10 JNDI API allows Java applications to access a variety of naming and directory services
- 10 The SPI enables a variety of naming and directory services to be plugged in transparently, thereby allowing the Java application using the JNDI API to access their services



Copyright © 2005, Infosys  
Technologies Ltd

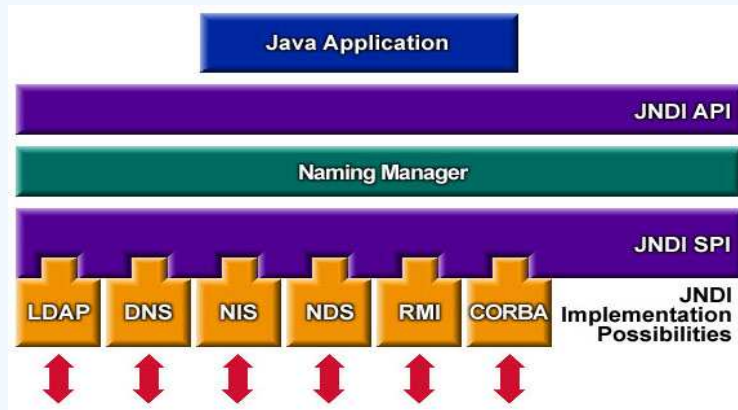
142

Infosys®

Although the Java LDAP API is a standard and tends to offer you support for the latest and greatest in LDAP API features, there exist a JNDI available for use with LDAP.

The `javax.naming.ldap` package contains a set of JNDI API classes and interfaces that support some of the LDAP extended set of operations and controls.

## JNDI Architecture (2 of 2)



Courtesy [www.java.sun.com](http://www.java.sun.com)



Copyright © 2005, Infosys Technologies Ltd

143

Infosys®

Not only is there a JNDI SPI for LDAP, but there also exists a special JNDI `javax.directory ldap` package for supporting some of the more sophisticated LDAP features.

## Service Providers

- Service providers offers Naming and directory services for various platforms
- *Service provider's* software maps the JNDI API to actual calls to the naming or directory server
- Important Service providers that comes along with JDK
  - Light Weight Directory Protocol (LDAP)
  - CORBA Common Object Services naming (COS naming)
  - RMI registry
  - Domain Name Service (DNS)
- JNDI client may invoke the JNDI API calls on these service provider's software to access the underline resources in unified manner



Copyright © 2005, Infosys  
Technologies Ltd

144

Infosys®



## JNDI Packages

- ④ javax.naming : contains classes and interfaces for accessing naming services
- ④ javax.naming.directory : extends the core javax.naming package to provide access to directories
- ④ javax.naming.spi : provide support for service provider interface
- ④ javax.naming.event : contains classes and interfaces for supporting event notification in naming and directory services
- ④ javax.naming.ldap : contains classes and interfaces for supporting LDAP v3 extensions and controls



Copyright © 2005, Infosys  
Technologies Ltd

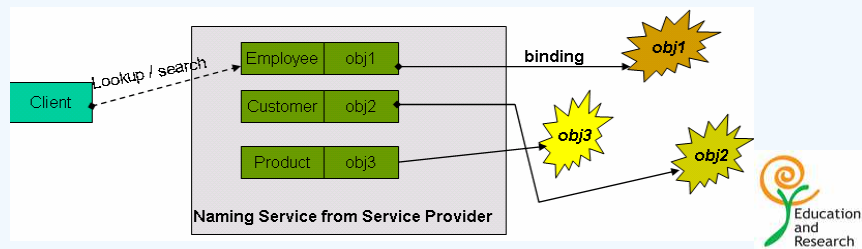
145

Infosys®

## javax.naming Package

### Some important APIs

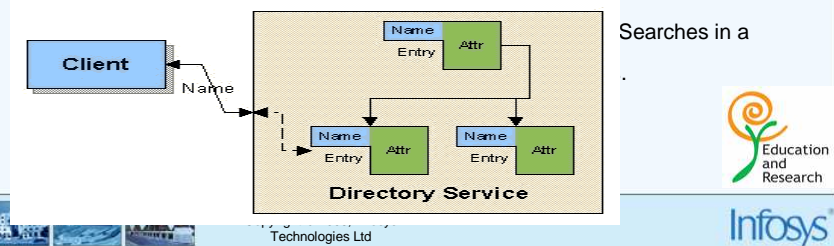
- Context.bind(String name, Object obj) : Binds the object **obj** with **name**
- Context.lookup(String name) : Retrieves the **named** object from the list of bindings
- Context.listBindings(String name): Returns NamingEnumeration of the names bound in the **named** context
- Context.unbind(String name) : Unbinds the named object



## javax.naming.directory Package

### Some important APIs

- `BasicAttribute.BasicAttribute(String id)` : Constructs a new instance of an unordered attribute
- `BasicAttribute.add(Object attrVal)` : Adds a new value to this attribute
- `BasicAttribute.get()` : Retrieves one of this attributes' value
- `BasicAttribute.getAll()` : Retrieves enumeration of this attributes' value
- `DirContext.bind(String name, Object obj, Attributes attrs)` : Binds a name to an object, along with associated attributes



## JNDI Naming & Lookup Example

```
import java.rmi.*;
import java.rmi.server.*;

public class ServerRMI extends UnicastRemoteObject implements InterfaceRMI
{
    public ServerRMI() throws RemoteException {
        super();
    }
    public int cube(int x) throws RemoteException {
        return x*x*x;
    }
    public static void main(String a[]) {
        try{
            ServerRMI rs = new ServerRMI();
            //binding the server object with the RMIRegistry
            Naming.rebind("MyServer",rs);
            System.out.println("System is ready");
        }catch(Exception e){System.out.println(e);}
    }
}
```



Copyright © 2005, Infosys  
Technologies Ltd

148

Infosys®

In the above example we are creating the object of ServerRMI. This object is bound to the RMI registry (running on the local machine) by invoking Naming.rebind() method. Here the service provider for RMI registry is Sun.

List of all binding in the registry can be obtained by invoking a call for **list()**.

**Naming.list("//localhost")** : Returns all the available bindings in the registry

To search a particular object with its name, invoke **lookup()**

**Naming.lookup("//localhost/MyServer")** : Searches the RMI registry for object named as MyServer

## Summary

- ❶ RMI Interface can be used to invoke a method of an object on different machine or different JVM on same machine
- ❷ Java security model is critical as mostly this language is being used in Internet scenario
- ❸ JNDI API are used to interface Java Programs with directory services to simplify enterprise networking



Copyright © 2005, Infosys  
Technologies Ltd

149

Infosys®

# Thank You!



Copyright © 2005, Infosys  
Technologies Ltd

150

Infosys®