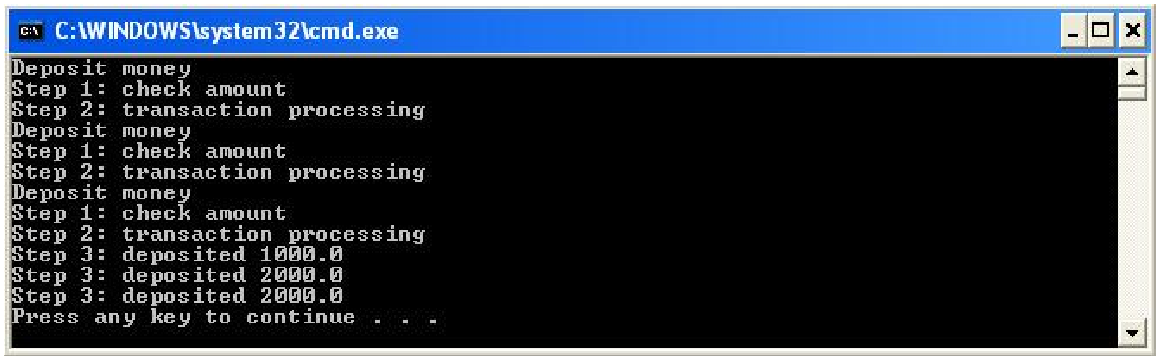
1. Using synchronized mechanism

```
class Account
{
   // The first way: synchronize a method
   /*
   public synchronized void deposit(double amount)
   {
      System.out.println("Step 1: check amount");

      System.out.println("Step 2: transaction processing");
      // Process
      try {
         Thread.sleep(3000);
      } catch(Exception e) {}

      System.out.println("Step 3: deposited " + amount);
   }
   */

   // The second way: not synchronize a method
   public void deposit(double amount)
   {
      System.out.println("Step 1: check amount");

      System.out.println("Step 2: transaction processing");
      // Process
      try {
         Thread.sleep(3000);
      } catch(Exception e) {}

      System.out.println("Step 3: deposited " + amount);
   }

   // No need synchronizing
   public double getBalance()
   {
      return 0;
   }
}


// Deposit transaction
class DepositTransaction implements Runnable
{
   private Account acc;
   private double amount;

   DepositTransaction(Account acc, double amount) {
      this.acc = acc;
```

© 2009 FPT-Aptech

```
        this.amount = amount;
    }

    public void run()
    {
        System.out.println("Deposit money");

        // The first way
        // Do not allow two threads call the deposit() method of
the same object
        acc.deposit(amount);

        // The second way
        // Do not allow two deposit transaction on the same account
        /*
        synchronized(acc) {
            acc.deposit(amount);
        }
        */
    }
}

public class BankTransaction
{
    public static void main(String a[])
    {
        Account acc = new Account();
        Account acc2 = new Account();

        Thread t1 = new Thread(new DepositTransaction(acc, 1000));
        Thread t2 = new Thread(new DepositTransaction(acc2, 2000));
        Thread t3 = new Thread(new DepositTransaction(acc, 2000));
        t1.start();
        t2.start();
        t3.start();
    }
}
```

The result:



```
Deposit money
Step 1: check amount
Step 2: transaction processing
Deposit money
Step 1: check amount
Step 2: transaction processing
Deposit money
Step 1: check amount
Step 2: transaction processing
Step 3: deposited 1000.0
Step 3: deposited 2000.0
Step 3: deposited 2000.0
Press any key to continue . . .
```

2. In this exercise, you are going to run OS specific programs using Runtime class.

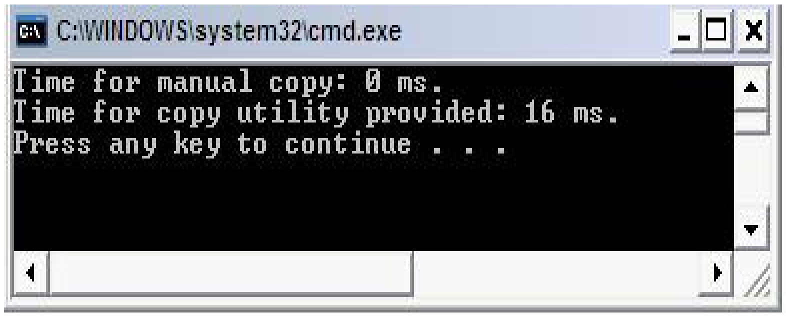
```
class RunOSProgram {

    public static void main(String args[]) {
        Runtime rt = Runtime.getRuntime();

        Process proc;

        try {

            if (System.getProperty("os.name").startsWith("Windows")){
                // Run a OS specific program
                proc = rt.exec("notepad");
            }
            else{
                proc = rt.exec("gedit");
            }

            proc.waitFor();   //try removing this line
        } catch (Exception e) {
            System.out.println("notepad is an unknown command.");
        }
    }

}
```

3. In this exercise, you are going to run various methods of the System class

```
import java.io.*;
class SystemClass {

   public static void main(String args[])
                           throws IOException {
      int arr1[] = new int[1050000];
      int arr2[] = new int[1050000];
      long startTime, endTime;
      /* initialize arr1 */
      for (int i = 0; i < arr1.length; i++) {
         arr1[i] = i + 1;
      }
      /* copying manually */
      startTime = System.currentTimeMillis();
      for (int i = 0; i < arr1.length; i++) {
         arr2[i] = arr1[i];
      }
      endTime = System.currentTimeMillis();
      System.out.println("Time for manual copy: " +
                    (endTime-startTime) + " ms.");
```

```
        /* using the copy utility provided by java */
        startTime = System.currentTimeMillis();
        System.arraycopy(arr1, 0, arr2, 0, arr1.length);
        endTime = System.currentTimeMillis();
        System.out.println("Time for copy utility provided: " +
                          (endTime-startTime) + " ms.");
        System.gc(); //force garbage collector to work
        System.exit(0);
    }
}
```

The result:



```
Time for manual copy: 0 ms.
Time for copy utility provided: 16 ms.
Press any key to continue . . .
```

Do It Yourself

2.1. Do workshop of the module 2, 3

2.2. Using wait and notify mechanism to write an application to demonstrate a withdrawal transaction or a deposit transaction in a bank. This bank includes n customers (this value is entered from user). Information about a customer consists id, name, balance. Building the menu:

```
Menu
--------------------------
   1. Create n accounts
   2. Do transactions
   3.  Exit
   Your choice: _
```

+ Create n accounts: create n accounts with the current balance is 1000.
+ Do transactions: process n transactions (withdrawal or deposit) on the n customers. Amount of the transaction is random but less than 100.

2.3. Write a program named ProgramExecutor. This program loads information about executable programs in the text file programs.txt, display on the screen, and enable users to select and run an outside program.

Sample run:

```
RUN MY FAVOURITE PROGRAMS

   1. Notepad
   2. Paint
   3. Microsoft Word
   4. Microsoft Excel
   5. Calculator
   6. Internet Explorer
   7. Windows Explorer
   8. Exit

Run: _
```

Sample content of the file programs.txt

```
Notepad, notepad.exe
Paint, panit.exe
Microsoft Word, C:\Program Files\Microsoft Office\OFFICE11\Winword.exe
Microsoft Excel, C:\Program Files\Microsoft Office\OFFICE11\Excel.exe
Calculator, calc.exe
Internet Explorer, iexplore.exe
Windows Explorer, explorer.exe
```

References

+ Java tutorials

+ Javadoc

+ Java2s.com

+ Javapassion.com

+ Java almanac
http://www.exampledepot.com