MOTOROLA

Optimizing a Java® ME Application Part 1: Speed

October 1, 2006

Technical Article

Optimizing a Java® ME Application

Part 1: Speed

By

MOTODEV Staff

Introduction and Background

It is well known that the resources of mobile devices are limited, thus making JavaME applications run faster is very important for mobile phone developers. This paper discusses three optimization topics:

- ✧ Speed performance measurement
- ✧ Improving loop performance
- ✧ Object storage speed performance

Memory optimization and additional speed topics will be discussed in future technical articles.

Speed Performance Measurement

Before discussing speed optimization, a simple speed measurement method should be introduced. A very common and simple measure of speed is to use `System.currentTimeMillis()` statement. Example source code is as follows:

```
long starttime = System.currentTimeMillis();

// Here is the section of code to be tested
…

long endtime = System.currentTimeMillis();
long time_consumption = endtime – starttime;
```

You may have concerns that the `System.currentTimeMillis()` call will also take some time to execute which may make the test result inaccurate (a little larger than real value). Actually, this is not of concern – the source code below has been tested on a Motorola V360 device and the results recorded.

```
long timea = System.currentTimeMillis();
long timeb = System.currentTimeMillis();
long timec = System.currentTimeMillis();

System.out.println("Time A: " + timea);
System.out.println("Time B: " + timeb);
System.out.println("Time C: " + timec);
System.out.println("time A-B consumption: " + (timeb - timea));
System.out.println("time B-C consumption: " + (timec - timeb));
System.out.println("average: " + (timec-timea)/2);
```

The output below shows that the elapsed time of the `System.currentTimeMillis()` call can be ignored.

```
======================
Time A: 1157741128296

Time B: 1157741128296

Time C: 1157741128296

time A-B consumption: 0

time B-C consumption: 0

average: 0
======================
```

Improving Loop Performance

The source code below is commonly used when accessing all contained elements of an object:

Code Scenario 1

```
        for(int i=0;i<v.size();i++) {
            //access elements
        }
```

However, the above code may not be good from a performance perspective because the `v.size()` method will be invoked for every loop iteration. To improve the code, another variable should be defined and initialized with the object size. Example source code is below.

Code Scenario 2

```
        for(int i=0, size=v.size();i<size;i++) {
            //access elements
        }
```

The improvement effect is highly dependant on the `size()` method used.
For example, use a vector class as a test object, and initialize the vector in the `startApp()` method of the MIDlet:

Code Scenario 3

```
public class SpeedOptimizationTest extends MIDlet {
    Vector v = new Vector();

    public void startApp() {
        //Initialize vector
        for(int i=0;i<100;i++) {
            v.addElement(String.valueOf(i));
```

```
        }
        …
    }
…
}
```

Code Scenario 4 demonstrates the use of a "`for(int i=0;i<v.size();i++)`" test and Code Scenario 5 demonstrates the use of a "`for(int i=0, size=v.size();i<size;i++)`" test.

Code Scenario 4

```
long start_time = System.currentTimeMillis();
for(int i=0;i<v.size();i++) {
    System.out.print(v.elementAt(i).toString() + " ");
}
long end_time = System.currentTimeMillis();
System.out.println("¥nTime consumption: " +
        (end_time -start_time));
```

Code Scenario 5

```
System.out.println("¥n");
start_time = System.currentTimeMillis();
for(int i=0, size=v.size();i<size;i++) {
    System.out.print(v.elementAt(i).toString() + " ");
}
end_time = System.currentTimeMillis();
System.out.println("¥nTime consumption: " +
```

```
                (end_time -start_time));
```

Tables 1, 2 and 3 show the test results from three iterations of test performed on the Sun WTK default color emulator, Motorola V3i emulator and Motorola V3i handset.

Table 1: Loop improvement test result 1

Code	WTK	Motorola V3i emulator	Motorola V3i handset
Code Scenario 4	60	8	123
Code Scenario 5	60	6	118

Table 2: Loop improvement test result 2

Code	WTK	Motorola V3i emulator	Motorola V3i handset
Code Scenario 4	51	8	121
Code Scenario 5	60	6	119

Table 3: Loop improvement test result 3

Code	WTK	Motorola V3i emulator	Motorola V3i handset
Code Scenario 4	60	10	124
Code Scenario 5	50	10	119

Now, we enlarge the time consumption by inducing the `getSize()` method as code scenario 6. The statement `Thread.sleep(200)` is used to simulate 200ms get-size-time.

Code Scenario 6

```
/**
 * Simulate time consumption
 */
private int getSize() {
    try {
        Thread.sleep(200);
    } catch (Exception e) {
        e.printStackTrace();
```

```
        }
        return v.size();
    }
```

Code Scenario 7 is the modified code with the `getSize()` method which is used for the "`for(int i=0;i<v.size();i++)`" test.
Code Scenario 8 is the modified code with the `getSize()` method which is used for the "`for(int i=0, size=v.size();i<size;i++)`" test.

Code Scenario 7

```
        System.out.println("¥n");
        start_time = System.currentTimeMillis();
        for(int i=0;i<getSize();i++) {
            System.out.print(v.elementAt(i).toString() + " ");
        }
        end_time = System.currentTimeMillis();
        System.out.println("¥nTime consumption: " +
                (end_time -start_time));
```

Code Scenario 8

```
        System.out.println("¥n");
        start_time = System.currentTimeMillis();
        for(int i=0, size=getSize();i<size;i++) {
            System.out.print(v.elementAt(i).toString() + " ");
        }
        end_time = System.currentTimeMillis();
        System.out.println("¥nTime consumption: " +
```

```
                (end_time -start_time));
```

Tables 4, 5 and 6 show the test results from three iterations of test using the `getSize()` method performed on the Sun WTK default color emulator, Motorola V3i emulator and Motorola V3i handset.

Figure 1 provides a comparison of the time consumption for the execution of Code Scenario 7 and Code Scenario 8.

Table 4: Loop improvement test result 4

Code	WTK	Motorola V3i emulator	Motorola V3i handset
Code Scenario 7	20229	20780	20389
Code Scenario 8	281	200	322

Table 5: Loop improvement test result 5

Code	WTK	Motorola V3i emulator	Motorola V3i handset
Code Scenario 7	20589	20484	20415
Code Scenario 8	521	207	321

Table 6: Loop improvement test result 6

Code	WTK	Motorola V3i emulator	Motorola V3i handset
Code Scenario 7	20569	20890	20384
Code Scenario 8	250	200	323

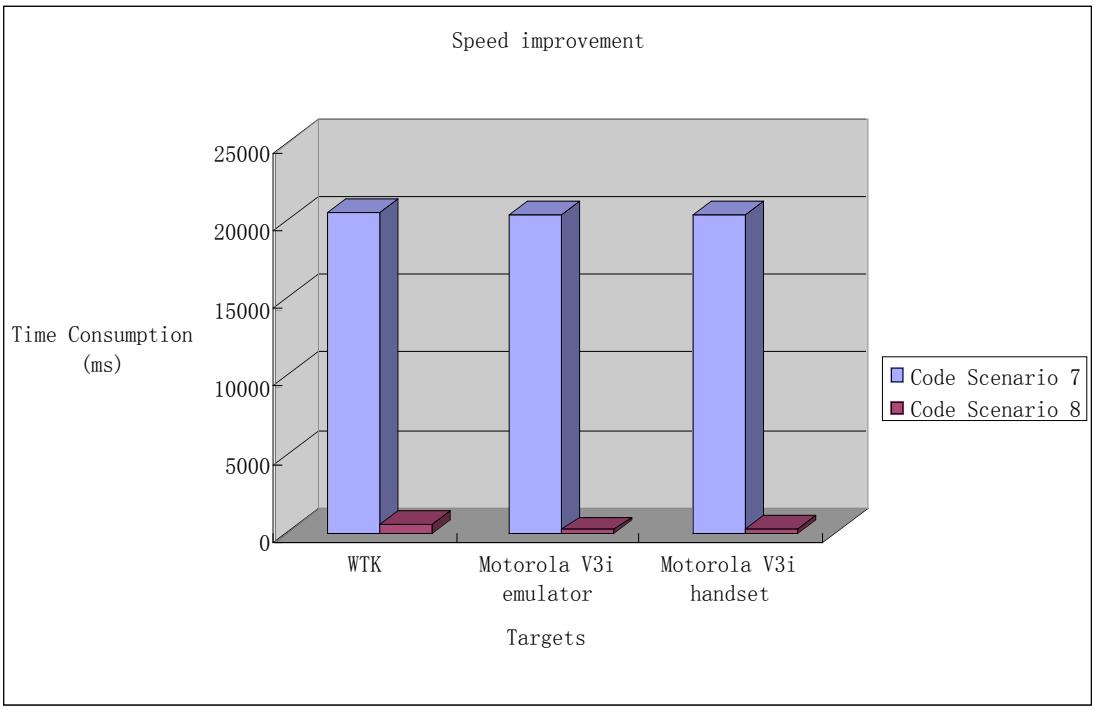

Figure 1: For Loop Speed Improvement

Conclusion: The Code Scenario 2 recommendations will provide a great deal of improvement in speed performance especially if the `size()` method is slow to execute.

Object storage speed

In JavaME, there are four ways to store objects: array, Vector, Hashtable and Stack. Regardless of the usage differences, in this article we will just focus on the time consumed by storing and retrieving.

In Code Scenario 9, the `GameItem` class is extended from `StringItem`, which is used to display two opponents in a game.

Code Scenario 9

```
class GameItem extends StringItem {
    private String user1 = null;
    private String user2 = null;
```

```
    int number = 0;


    public GameItem(String label, String text, int appearanceMode) {
        super(label, text, appearanceMode);
        setLayout(Item.LAYOUT_NEWLINE_BEFORE |
                Item.LAYOUT_NEWLINE_AFTER |
                Item.LAYOUT_EXPAND);
    }


    public void setUser(int pos, String user) {
        if (pos == 0) {
            this.user1 = user;
        } else {
            this.user2 = user;
        }
        if (user1 == null && user2 == null) {
            setText("#" + String.valueOf(number+1) +
                    ": ¥n");
        } else {
            setText("#" + String.valueOf(number+1) +
                    ": " + user1 + " - " +
                    user2 + "¥n");
        }
    }


    public String getUser(int pos) {
        if (pos == 0) {
```

```
            return user1;
        } else {
            return user2;
        }
    }
}
```

Writing Speed Test

Code scenario 10 is used to store the GameItem objects into an array.

Code Scenario 10

```
        GameItem[] gt = new GameItem[100];
        System.out.println("¥n");
        start_time = System.currentTimeMillis();
        for(int i=0, size=getSize();i<size;i++) {
            System.out.print(v.elementAt(i).toString() + " ");
        }
        end_time = System.currentTimeMillis();
        System.out.println("¥nTime consumption: " +
                (end_time -start_time));
```

Code Scenario 11 is used to store the GameItem objects into a vector.

Code Scenario 11

```
        Vector v = new Vector();
```

```
start_time = System.currentTimeMillis();
for(int i=0;i<100;i++) {
    GameItem gi = new GameItem("test"+String.valueOf(i),
            "Vector", Item.BUTTON);
    gi.setUser(0, "user"+String.valueOf(0) + "-0");
    gi.setUser(1, "user"+String.valueOf(0) + "-1");

    v.addElement(gi);
}
end_time = System.currentTimeMillis();
System.out.println("Vector put time consumption: " +
        (end_time - start_time));
```

Code Scenario 12 is used to store the GameItem objects into a Hashtable.

Code Scenario 12

```
Hashtable h = new Hashtable();
start_time = System.currentTimeMillis();
for(int i=0;i<100;i++) {
    GameItem gi = new GameItem("test"+String.valueOf(i),
            "Hashtable", Item.BUTTON);
    gi.setUser(0, "user"+String.valueOf(0) + "-0");
    gi.setUser(1, "user"+String.valueOf(0) + "-1");

    h.put(new Integer(i), gi);
}
end_time = System.currentTimeMillis();
System.out.println("Hashtable put time consumption: " +
        (end_time - start_time));
```

Code Scenario 13 is used to store the GameItem objects into a stack.

Code Scenario 13

```
Stack s = new Stack();
start_time = System.currentTimeMillis();
for(int i=0;i<100;i++) {
    GameItem gi = new GameItem("test"+String.valueOf(i),
            "Stack", Item.BUTTON);
    gi.setUser(0, "user"+String.valueOf(0) + "-0");
    gi.setUser(1, "user"+String.valueOf(0) + "-1");

    s.push(gi);
}
end_time = System.currentTimeMillis();
System.out.println("Stack put time consumption: " +
        (end_time - start_time));
```

Tables 7, 8 and 9 show the test results from three iterations of test for putting objects into storage options performed on Sun WTK default color emulator, Motorola V3i emulator and Motorola V3i handset.
Figure 2 compares time consumption for these operations.

Table 7: Writing objects test result 1

Code	WTK	Motorola V3i emulator	Motorola V3i handset
Code Scenario 10	70	159	5361
Code Scenario 11	80	156	7839
Code Scenario 12	91	171	10407
Code Scenario 13	160	240	13517

Table 8: Writing objects test result 2

Code	WTK	Motorola V3i emulator	Motorola V3i handset
Code Scenario 10	70	159	5369
Code Scenario 11	80	156	7879
Code Scenario 12	91	171	10410

Code Scenario 13	160	240	13562

Table 9: Writing objects test result 3

Code	WTK	Motorola V3i emulator	Motorola V3i handset
Code Scenario 10	70	154	5371
Code Scenario 11	70	154	7841
Code Scenario 12	80	175	10380
Code Scenario 13	90	254	13558

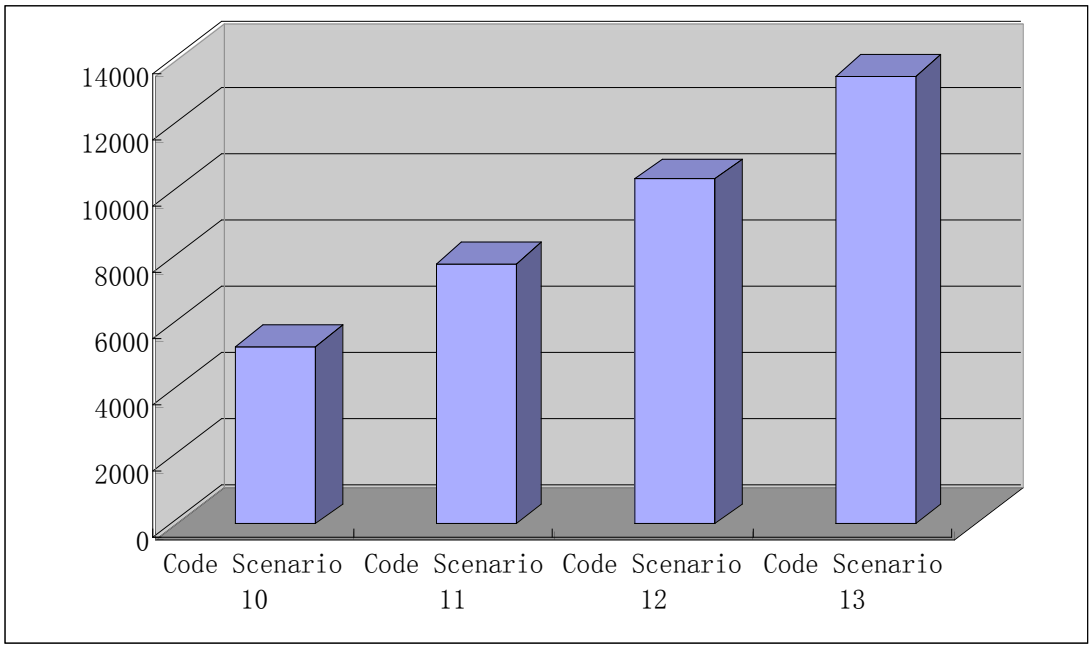

Figure 2: Writing Objects Into Storage

Conclusion:

- ✧ Writing objects to an Array is the fastest option.
- ✧ Writing objects to a Vector is faster than a Hashtable but slower than an array.
- ✧ Writing objects to a Stack is the slowest option.

Reading Speed Test

Code Scenario 14 is used to read the GameItem objects from an array.

Code Scenario 14

```
start_time = System.currentTimeMillis();
for(int i=0;i<gt.length;i++) {
    String user1 = gt[i].getUser(0);
    String user2 = gt[i].getUser(1);
}
end_time = System.currentTimeMillis();
System.out.println("Array reading time consumption: " +
        (end_time - start_time));
```

Code Scenario 15 is used to read the GameItem objects from a vector.

Code Scenario 15

```
start_time = System.currentTimeMillis();
for(int i=0;i<v.size();i++) {
    GameItem gi = (GameItem)v.elementAt(i);
    String user1 = gi.getUser(0);
    String user2 = gi.getUser(1);
}
end_time = System.currentTimeMillis();
System.out.println("Vector reading time consumption: " +
        (end_time - start_time));
```

Code Scenario 16 is used to read the GameItem objects from a Hashtable via the “for” statement.

Code Scenario 16

```
        start_time = System.currentTimeMillis();
        for(int i=0;i<v.size();i++) {
            GameItem gi = (GameItem)h.get(new Integer(i));
            String user1 = gi.getUser(0);
            String user2 = gi.getUser(1);
        }
        end_time = System.currentTimeMillis();
        System.out.println("Hashtable reading time consumption: " +
                (end_time - start_time));
```

Code Scenario 17 is used to read the GameItem objects from a Hashtable via an enumeration.

Code Scenario 17

```
        start_time = System.currentTimeMillis();
        Enumeration e = h.elements();
        while(e.hasMoreElements()) {
            GameItem gi = (GameItem)e.nextElement();
            String user1 = gi.getUser(0);
            String user2 = gi.getUser(1);
        }
        end_time = System.currentTimeMillis();
        System.out.println("Hashtable   enumeration   reading   time
consumption: " +
                (end_time - start_time));
```

Code scenario 18 is used to read the GameItem objects from a stack.

Code Scenario 18

```
start_time = System.currentTimeMillis();
while(!s.isEmpty()) {
    GameItem gi = (GameItem)s.pop();
    String user1 = gi.getUser(0);
    String user2 = gi.getUser(1);
}
end_time = System.currentTimeMillis();
System.out.println("Stack reading time consumption: " +
        (end_time - start_time));
```

Tables 10, 11 and 12 show the test results from three iterations of tests for retrieving objects from storage performed on the Sun WTK default color emulator, Motorola V3i emulator and Motorola V3i handset.
Figure 3 Compares time consumption for these operations.

Table 10: Reading objects test result 1

Code	WTK	Motorola V3i emulator	Motorola V3i handset
Code Scenario 14	0	0	4
Code Scenario 15	10	0	7
Code Scenario 16	20	0	65
Code Scenario 17	0	0	12
Code Scenario 18	10	1	67

Table 11: Reading objects test result 2

Code	WTK	Motorola V3i emulator	Motorola V3i handset
Code Scenario 14	10	0	5
Code Scenario 15	10	0	7
Code Scenario 16	10	0	62
Code Scenario 17	0	0	11
Code Scenario 18	0	0	60

Table 12: Reading objects test result 3

Code	WTK	Motorola V3i emulator	Motorola V3i handset
Code Scenario 14	0	0	4
Code Scenario 15	10	1	8
Code Scenario 16	10	0	65
Code Scenario 17	0	0	11
Code Scenario 18	0	0	67

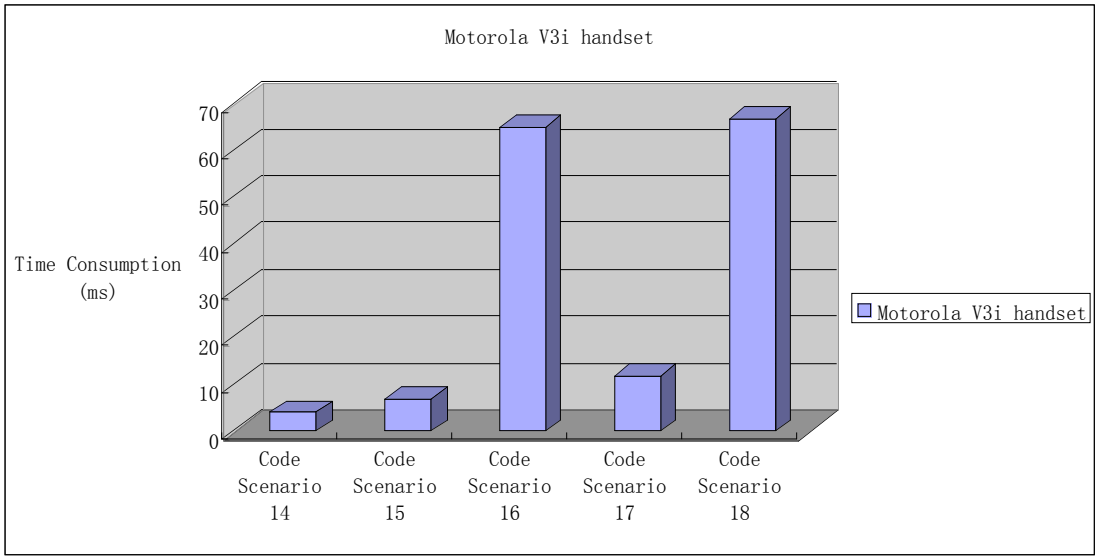

Figure 3: Reading objects from storage

Conclusion:

- ✧ Reading objects from an Array is the fastest option.
- ✧ Reading objects from a Vector is faster than a Hashtable but slower than an array.
- ✧ Reading objects from a Hashtable, enumeration is faster than for a loop.
- ✧ Reading objects from a Stack is the slowest option.