Mobile and Wireless Systems Programming

Programming methodology

- Small embedded systems
  - Low effort for maintability
  - Low memory and performance
  - Power consumption
  - Small screens
- Mobile not Desktop !

- Profile
- Configuration
- Screen size
- Available Memory
- Available APIs
- Available media formats

NOK

```
double x = d * (lim / max) * sx;
double y = d * (lim / max) * sy;
```

NOK

```
double x = d * (lim / max) * sx;
double y = d * (lim / max) * sy;
```

OK

```
double xy = d * (lim / max);
double x = xy * sx;
double y = xy * sy;
```

NOK

```
double x = d * (lim / max) * sx;
double y = d * (lim / max) * sy;
```

OK

```
double xy = d * (lim / max);
double x = xy * sx;
double y = xy * sy;
```

NOK

```
int MAX = 1000000;
int[] Tab = new int[1000];

for (int i = 0; i < Tab.length; i++){
    Tab[0] = MAX;
    ...
}
```

NOK

```
int MAX = 1000000;
int[] Tab = new int[1000];

for (int i = 0; i < Tab.length; i++){
    Tab[0] = MAX;
    ...
}
```

OK

```
int MAX = 1000000;
int[] Tab = new int[1000];

for (int i = 0; i < Tab.length; i++){
    if(i==0)
        Tab[0] = MAX;
    ...
}
```

NOK

```
int MAX = 1000000;
int[] Tab = new int[1000];

for (int i = 0; i < Tab.length; i++){
    Tab[0] = MAX;
    ...
}
```

OK

```
int MAX = 1000000;
int[] Tab = new int[1000];

for (int i = 0; i < Tab.length; i++){
    if(i==0)
        Tab[0] = MAX;
    ...
}
```

NOK

```
for (int i = 0; i < x.length; i++)
    x[i] *= Math.PI * Math.cos(y);
```

NOK

```
for (int i = 0; i < x.length; i++)
    x[i] *= Math.PI * Math.cos(y);
```

OK

```
double picosy = Math.PI * Math.cos(y);

for (int i = 0; i < x.length; i++)
    x[i] *= picosy;
```

NOK

```
for (int i = 0; i < x.length; i++)
    x[i] *= Math.PI * Math.cos(y);
```

OK

```
double picosy = Math.PI * Math.cos(y);

for (int i = 0; i < x.length; i++)
    x[i] *= picosy;
```

NOK

```
StringBuffer strBuf = new StringBuffer();
for(int i = 0; i < taille; i++) {
    if(t[i] >='0' && t[i] < '5') {
        strBuf += t[i];
    }
}
```

NOK

```
StringBuffer strBuf = new StringBuffer();
for(int i = 0; i < taille; i++) {
    if(t[i] >='0' && t[i] < '5') {
        strBuf += t[i];
    }
}
```

OK

```
StringBuffer strBuf = new StringBuffer();
char cBuf;
for(int i = 0; i < taille; i++) {
    cBuf = t[i];
    if(cBuf >='0' && cBuf <'5') {
        strBuf += cBuf;
    }
}
```

NOK

```
StringBuffer strBuf = new StringBuffer();
for(int i = 0; i < taille; i++) {
    if(t[i] >='0' && t[i] < '5') {
        strBuf += t[i];
    }
}
```

OK

```
StringBuffer strBuf = new StringBuffer();
char cBuf;
for(int i = 0; i < taille; i++) {
    cBuf = t[i];
    if(cBuf >='0' && cBuf <'5') {
        strBuf += cBuf;
    }
}
```

NOK

```
int sum = 0;
for(int i = 0; i < v.size(); i++) {
    sum += v.elementAt(i);
}
```

NOK

```
int sum = 0;
for(int i = 0; i < v.size(); i++) {
    sum += v.elementAt(i);
}
```

OK

```
int sum = 0;
int size = v.size();
for(int i = 0; i < size; i++) {
    sum += v.elementAt(i);
}
```

NOK

```
int sum = 0;
for(int i = 0; i < v.size(); i++) {
    sum += v.elementAt(i);
}
```

OK

```
int sum = 0;
int size = v.size();
for(int i = 0; i < size; i++) {
    sum += v.elementAt(i);
}
```

Forward loop

```
long start1 = System.currentTimeMillis();
for (i = 0; i <= 2000; i++) {
    for (j = 0; j <= 10; j++) {
    }
}
long stop1 = System.currentTimeMillis();
```

Forward loop

```
long start1 = System.currentTimeMillis();
for (i = 0; i <= 2000; i++) {
    for (j = 0; j <= 10; j++) {
    }
}
long stop1 = System.currentTimeMillis();
```

Backward loop

```
long start2 = System.currentTimeMillis();
for (i = 2000; i >= 0; i--) {
    for (j = 10; j >= 0; j--) {
    }
}
long stop2 = System.currentTimeMillis();
```

Forward loop

```
long start1 = System.currentTimeMillis();
for (i = 0; i <= 2000; i++) {
    for (j = 0; j <= 10; j++) {
    }
}
long stop1 = System.currentTimeMillis();
```

Backward loop

```
long start2 = System.currentTimeMillis();
for (i = 2000; i >= 0; i--) {
    for (j = 10; j >= 0; j--) {
    }
}
long stop2 = System.currentTimeMillis();
```

NOK

```
if(state == 1){
...
}else if(state == 2){
...
}else if (state == 3){
...
}
```

NOK

```
if(state == 1){
...
}else if(state == 2){
...
}else if (state == 3){
...
}
```

OK

```
public static final int MAIN_STATE = 1;
public static final int OPTIONS_STATE = 2;
public static final int PAUSE_STATE = 3;

if(state == MAIN_STATE){
...
}else if(state == OPTIONS_STATE){
...
}else if (state == PAUSE_STATE){
...
}
```

NOK

```
if(state == 1){
...
}else if(state == 2){
...
}else if (state == 3){
...
}
```

OK

```
public static final int MAIN_STATE = 1;
public static final int OPTIONS_STATE = 2;
public static final int PAUSE_STATE = 3;

if(state == MAIN_STATE){
...
}else if(state == OPTIONS_STATE){
...
}else if (state == PAUSE_STATE){
...
}
```

NOK

```
Pos myPos = getMyPos();
Pos monsterPos = getMonsterPos();
int dist = getDistance(myPos, monsterPos);
```

NOK

```
Pos myPos = getMyPos();
Pos monsterPos = getMonsterPos();
int dist = getDistance(myPos, monsterPos);
```

OK

```
dist = getDistance(getMyPos(),getMonsterPos());
```

NOK

```
Pos myPos = getMyPos();
Pos monsterPos = getMonsterPos();
int dist = getDistance(myPos, monsterPos);
```

OK

```
dist = getDistance(getMyPos(),getMonsterPos());
```

NOK

```
String s0 = "abc";
String s1 = "def";
s0 = new String(s0.concat(s1));
```

NOK

```
String s0 = "abc";
String s1 = "def";
s0 = new String(s0.concat(s1));
```

OK

```
StringBuffer s0 = "abc";
StringBuffer s1 = "def";
s0.append(s1);
```

NOK

```
String s0 = "abc";
String s1 = "def";
s0 = new String(s0.concat(s1));
```

OK

```
StringBuffer s0 = "abc";
StringBuffer s1 = "def";
s0.append(s1);
```

- Arrays are faster than Collection objects, so use arrays when possible
- Convert 2D arrays to 1D arrays :
  - less array bounds checks
  - less array.length

NOK

```
boolean[][] enemyMap = new boolean[5][12];
if (enemyMap[myX+1][myY+1] ||
    enemyMap[myX-1][myY+1] ||
    enemyMap[myX+1][myY-1] ||
    enemyMap[myX-1][myY-1] ) {
    ...
}
```

NOK

```
boolean[][] enemyMap = new boolean[5][12];
if (enemyMap[myX+1][myY+1] ||
    enemyMap[myX-1][myY+1] ||
    enemyMap[myX+1][myY-1] ||
    enemyMap[myX-1][myY-1] } {
    . . .
}
```

OK

```
boolean[] enemyMap = new boolean[5*12];
int myLoc = myX*12 + myY;
if (enemyMap[myLoc+1] ||
    enemyMap[myLoc-1] ||
    enemyMap[myLoc+12] ||
    enemyMap[myLoc-12] } {
    . . .
}
```

NOK

```
boolean[][] enemyMap = new boolean[5][12];
if (enemyMap[myX+1][myY+1] ||
    enemyMap[myX-1][myY+1] ||
    enemyMap[myX+1][myY-1] ||
    enemyMap[myX-1][myY-1] } {
    . . .
}
```

OK

```
boolean[] enemyMap = new boolean[5*12];
int myLoc = myX*12 + myY;
if (enemyMap[myLoc+1] ||
    enemyMap[myLoc-1] ||
    enemyMap[myLoc+12] ||
    enemyMap[myLoc-12] } {
    . . .
}
```

- When possible, make the size of the resource available to your ../code/metho
- InputStream.available() works but... sometimes
- Otherwise store sizes in your own structure
- Close streams when finished

- Use clipping to update region

```
Canvas.repaint(int x, int y, int width, int height)
```

- Don't load unused resources
- Avoid loading the same image into memory more than once,
- Adapt resources to mobile phone

NOK

```
void paint(Graphics g){
    int screenWidth = getWidth() ;
    int screenHeight = getHeight() ;
    g.drawString("Bad example"...);
}
```

NOK

```
void paint(Graphics g){
    int screenWidth = getWidth() ;
    int screenHeight = getHeight() ;
    g.drawString("Bad example"...);
}
```

OK

```
int screenWidth = getWidth() ;
int screenHeight = getHeight() ;

void paint(Graphics g){
    g.drawString("Good example"...);
}
```

NOK

```
void paint(Graphics g){
    int screenWidth = getWidth() ;
    int screenHeight = getHeight() ;
    g.drawString("Bad example"...);
}
```

OK

```
int screenWidth = getWidth() ;
int screenHeight = getHeight() ;

void paint(Graphics g){
    g.drawString("Good example"...);
}
```

NOK

```
public void paint(Graphics g) {
    updateScreen();
}

public void keyPressed(int key) {
    repaint();
}
```

NOK

```
public void paint(Graphics g) {
    updateScreen();
}

public void keyPressed(int key) {
    repaint();
}
```

OK

```
public void paint(Graphics g) {
    g.*();
    ...
}

public void keyPressed(int key) {
    updateScreen();
    repaint();
}
```

NOK

```
public void paint(Graphics g) {
    updateScreen();
}

public void keyPressed(int key) {
    repaint();
}
```

OK

```
public void paint(Graphics g) {
    g.*();
    ...
}

public void keyPressed(int key) {
    updateScreen();
    repaint();
}
```

- use finally() to clean
- Avoid Exception

```
int sum = 0;
try {
    int index = 0;
    while (true) { //No bounds check
        sum += array[index++];
    }
} catch (ArrayIndexOutOfBoundsException e) {}
return sum;
```

- Affect to null unsused references
- Remove debug information
- Use bitwise operators when possible
- Not advisable to call System.gc()
- Avoid recursion
- Limit the use of inner classes
- Avoid unnecessary re-initialization of variables that are automatically set to 0 or null by the VM
- Don't create new ../code/metho when ../code/metho already exists in the platform to do close to the same thing. For example, don't create a class to maintain a Vector of record IDs, use RecordEnumerator instead
- Delay all possible work at start

- Rename class, methods and fields
- Create more compact ../code/metho
- Smaller ../code/metho archives
- Smaller memory footprints
- Lists dead ../code/metho
- ...

- Removes unnecessary information in PNG files
- Makes PNG data more compressible
- ...

Use WTK profiler to identify bottlenecks

- methods monitor
- memory monitor

```
long startTime = System.currentTimeMillis();
doSomething();
long timeTaken = System.currentTimeMillis() - startTime;
```

- Think usability
- Links :
  - http ://-wiki.forum.nokia.com/index.php/J2ME_Best_Practices
  - http ://www.javaperformancetuning.com/tips/j2me.shtml