# Code Optimization - Java ME

Mobile devices are resource-constrained. This article contains a compilation of code optimization techniques which may be helpful to reduce jar size and memory and CPU usage in your Java ME applications.

## Optimisation techniques

1. Use an obfuscator like the free Proguard or Retroguard. They can remove unused code and perform other optimizations to decrease MIDlet size (JAR file size).
2. Use minimum number of classes, even a mostly-empty class uses space.
3. Avoid unnecessary object creation.
4. Reuse objects when possible.
5. For saving memory, explicitly set unused object to null.
6. Beware of calling garbage collection, calling system.gc to free memory may cause slowdowns because in certain phones it is a blocking statement.
7. Use only minimum images in the application to save space.
8. If not using an obfuscator
    1. All the variable, method & class name should be small
    2. Remove any unwanted code
    3. The package structure should be as minimum as possible, if possible avoid packages
9. Avoid creating many Threads and reuse Thread objects when needed.

## Avoid unnecessary object creation

In order to avoid unnecessary objection creation, define your classes in such a way that you can reuse their objects. Avoid creating object types that do not allow the modification of their contents (e.g. String), forcing the creation of new object instances in order to treat new values. The goal is to minimize memory allocation by reusing objects that have already been created.

For instance, take a look at the next example. Imagine adding the values of two *int*s (stored in a int[]) and printing its value. The first *for* step shows how **NOT** to reuse objects. The second one shows how this code can be optimized.

```
public class MyObject {

    public int plus(int a, int b) {
        return a+b;
    }

}

//Not reusing objects
int[] values = new int[]{... ... ..};
for (int i=0; ... ...) {
     obj = new MyObject();
     int value = obj.plus(i, i+1);
     System.out.println(value);
}

//Reusing objects
int[] values = new int[]{... ... ..};
obj = new MyObject();
int value;
for (int i=0; ... ...) {
     value = obj.plus(i, i+1);
     System.out.println(value);
}
```

## Allocate memory when needed (Lazy Loading)

Avoid creating objects (or other primitive variables) if during the execution of the code this (variable) MAY NOT be used.

```
public class MyObject {
    private Image photo;

    public Image getPhoto(String file) {
        if(photo == null)
           photo = Image.createImage(file); //Create object only when needed.

        return photo;
    }
}
```

## Singleton Design Pattern

The singleton design pattern can be one of the important features in order to optimize your code, avoiding unnecessary object creation. Classes that treat requirements such as log messages, configurations, persistence, among others, are good candidates to be singleton.

## Loop definition

Be careful when defining loop steps, such as *while* and *for*. The following example shows how to *NOT* build your loops, consuming time processing.

```
//array.size() will be called in every loop from i=0 to size.
for (int i=0; i<array.size(); i++) {
   ...
}

//On the other hand, keep the length of the array object, and use it
int size = array.size();
for (int i=0; i<size; i++) {
   ...
}
```