

DoJa

Application Downsize

Technique Guide

Version1.0

Revision History

Release	Revision	Revised	Date revised:
1.0	First version	N.Koyama	2003/12/11

Index

INDEX	3
INTRODUCTION	5
How to Downsize the DoJa Application Size	5
ALGORITHM OPTIMIZATION	6
Try Not to Use “Else If” as Much as Possible	7
Try Not to Use “Switch” Statement as Much as Possible	8
Optimization of “For” Loop	9
Decrement of “For” Loop	10
Arithmetic Operation	11
Try to Summarize “Try-Catch” Block	12
Try to Use “Existing Value”	13
Try Not to Use the Package	14
METHOD OPTIMIZATION	15
Try to the Decrease Method	15
Try to Use the Static Method	16
VARIABLE OPTIMIZATION	17
Try to Use the Local Variable	17
Try to Declare the Array Variable	18
Try Not to Use the Static Variable	19
Try to Use the Default Variable Value	20
Try to Declare the Local Variables at One Place	21
Try to Use “int” for the Number Value	22
Try to Use a Number Ranged from -1 to 5 Preferentially	23
Try to Declare the Variable Instead of the Constant Variable	24
Try to Use the Specific Value Instead of the Constant Variable	25
CLASS OPTIMIZATION	26
Try to Decrease the Class	26
Try Not to Use the Inner Class	27
NAMING OPTIMIZATION	28

TRY TO SHARE AND SHORTEN THE METHOD AND VARIABLE NAMES.....	28
BINARY OPTIMIZATION	29
OTHERS	エラー! ブックマークが定義されていません。
JPREST(ENGLISH).....	エラー! ブックマークが定義されていません。
RETROGURD(ENGLISH)	エラー! ブックマークが定義されていません。
JAVA BLENDER(JAPANESE).....	エラー! ブックマークが定義されていません。
JARG(JAPANESE).....	エラー! ブックマークが定義されていません。

Introduction

This documentation is aimed to downsize the DoJa application. This documentation describes the tips for reducing the DoJa application size, so this is useful when the application is over the size restriction. The procedure is divided into 6 categories for downsize the Doja application size.

How to Downsize the DoJa application size

This is the recommended steps to optimize the application. This table also describes for the rate percentage of an effect. So according to this , you can decide how much time you can spend on for the optimization of each step. (This result value depends on the application. The rate is approximate.)

Optimization step	Target Optimization	Percentage of the effect (%)
1	Algorithm Optimization	2%
2	Method Optimization	7%
3	Variable Optimization	28%
4	Class Optimization	19%
5	Naming Optimization	43%
6	Binary Optimization	1%

Algorithm Optimization

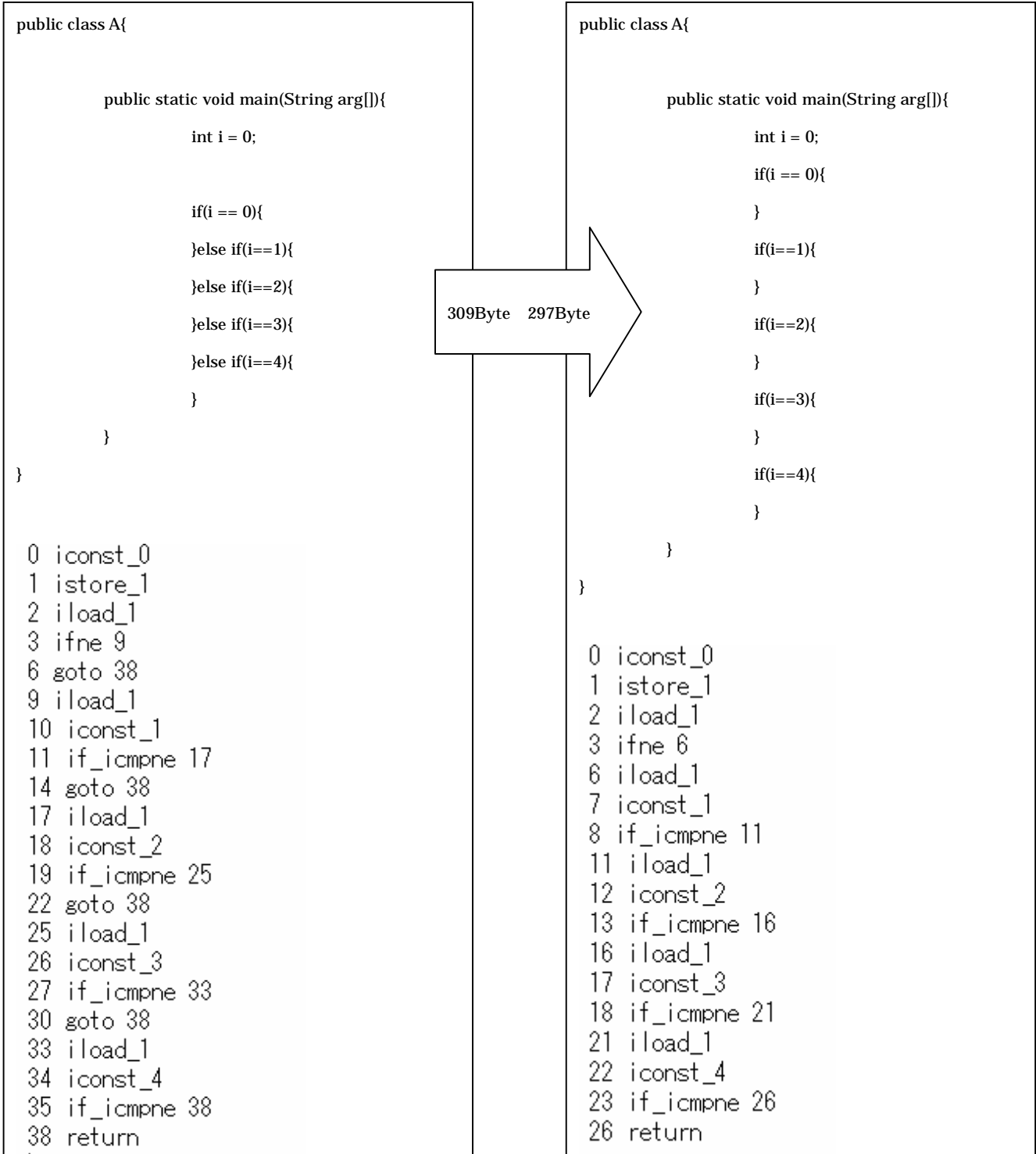
This table describe the machine language commands for VM. According to this table you can see which algorithm is effective to reduce the size.

Below table is the explanation of the function of instruction.

classification	Function	Instruction
Stack	Put the variable on stack	push, ldc, const
	Edit stack	nop, pop, dup, swap
Local variable	Put the variable from local variable on stack	load
	Store the variable from stack to local variable	store
	Others	iinc, wide
Array	Generate the array	newarray, anewarray, multianewarray
	Get the value from the array	aload
	Store the value to the array	astore
	Others	arraylength
Object	Generate new Instance	new
	Access to the field	putfield, getfield, putstatic, getstatic
	Others	checkcast, instanceof
Operation	Arithmetic operation	add, sub, mul, div, rem, neg
	Logic operation	sh, and, or, xor
	Type conversion	2
Moving position	Condition branch	if
	Comparison	cmp
	Unconditional branch and subroutine	goto, jsr, ret
	Table jump	lookupswitch, tableswitch
Method	Call	invokevirtual, invokespecial, invokestatic, invokeinterface
	Return	retrun
Others	Exception	athrow
	Debug	breakpoint
	Monitor	monitorenter, monitorexit

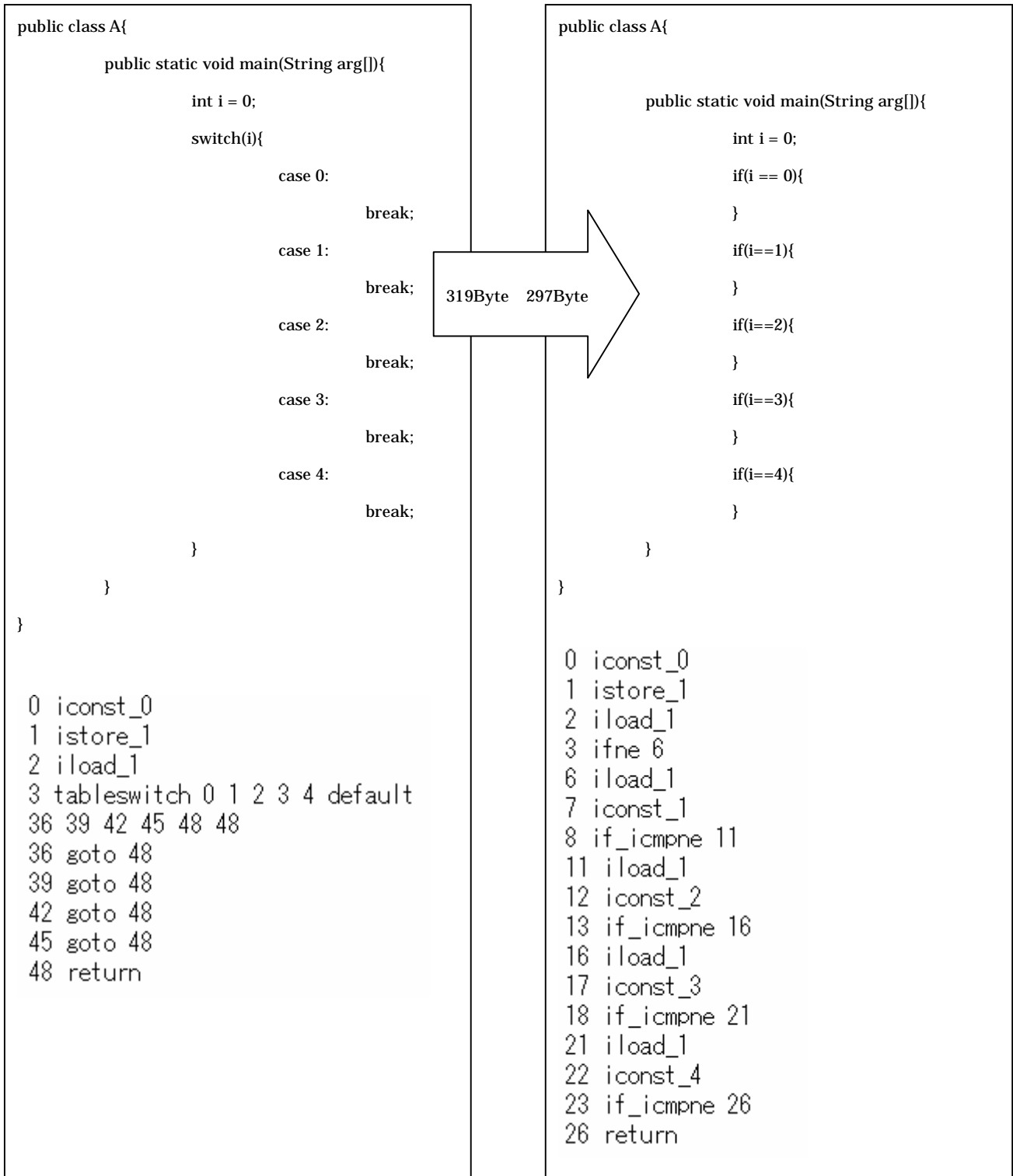
Try not to use “else if” as much as possible

If there is “else if” statement in the source code, consider to change it to plural “if” statement.



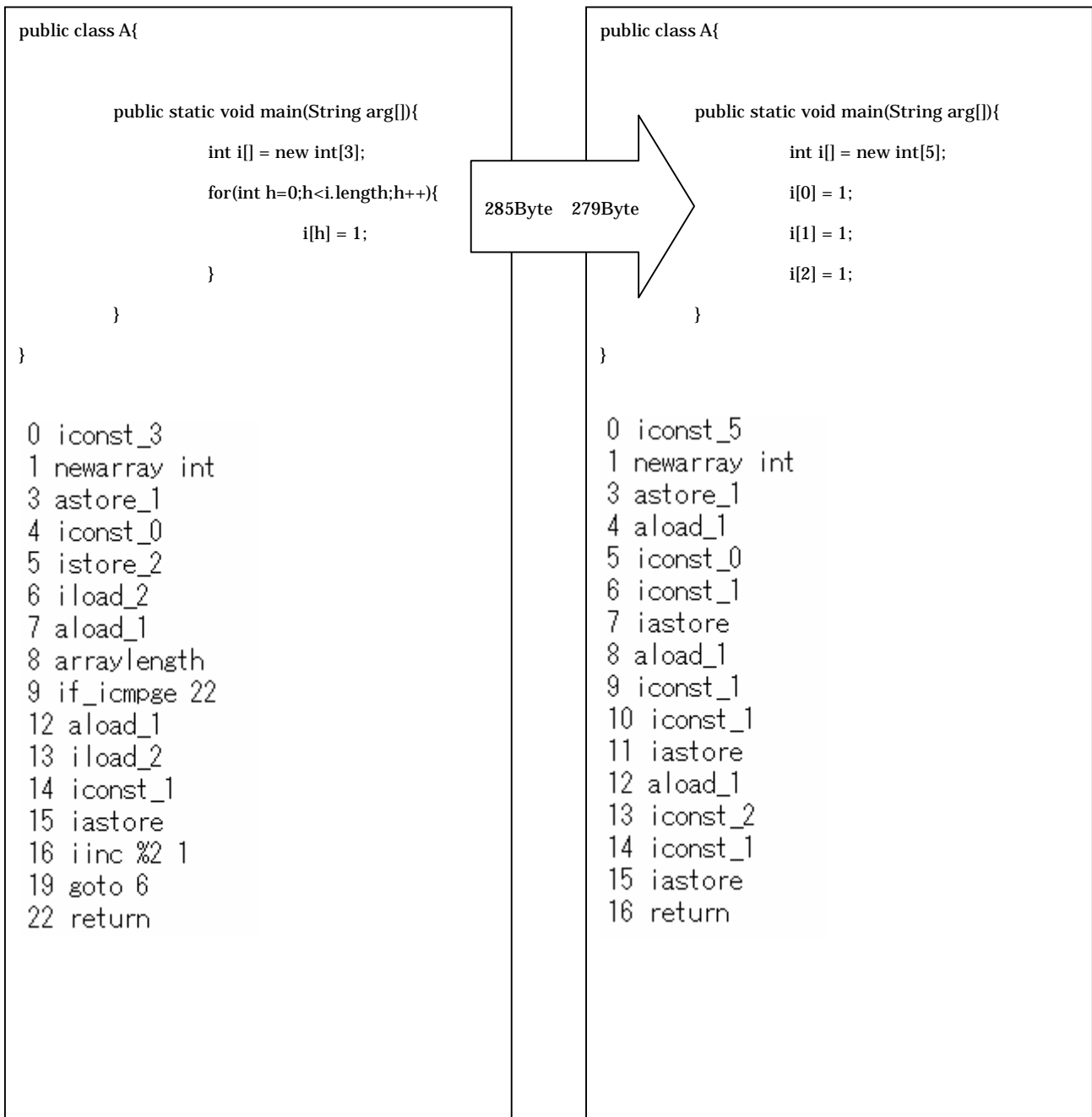
Try not to use “switch” statement as much as possible

If there is “switch” statement in the source code, consider to change it to plural “if” statement.



Optimization of “for” loop

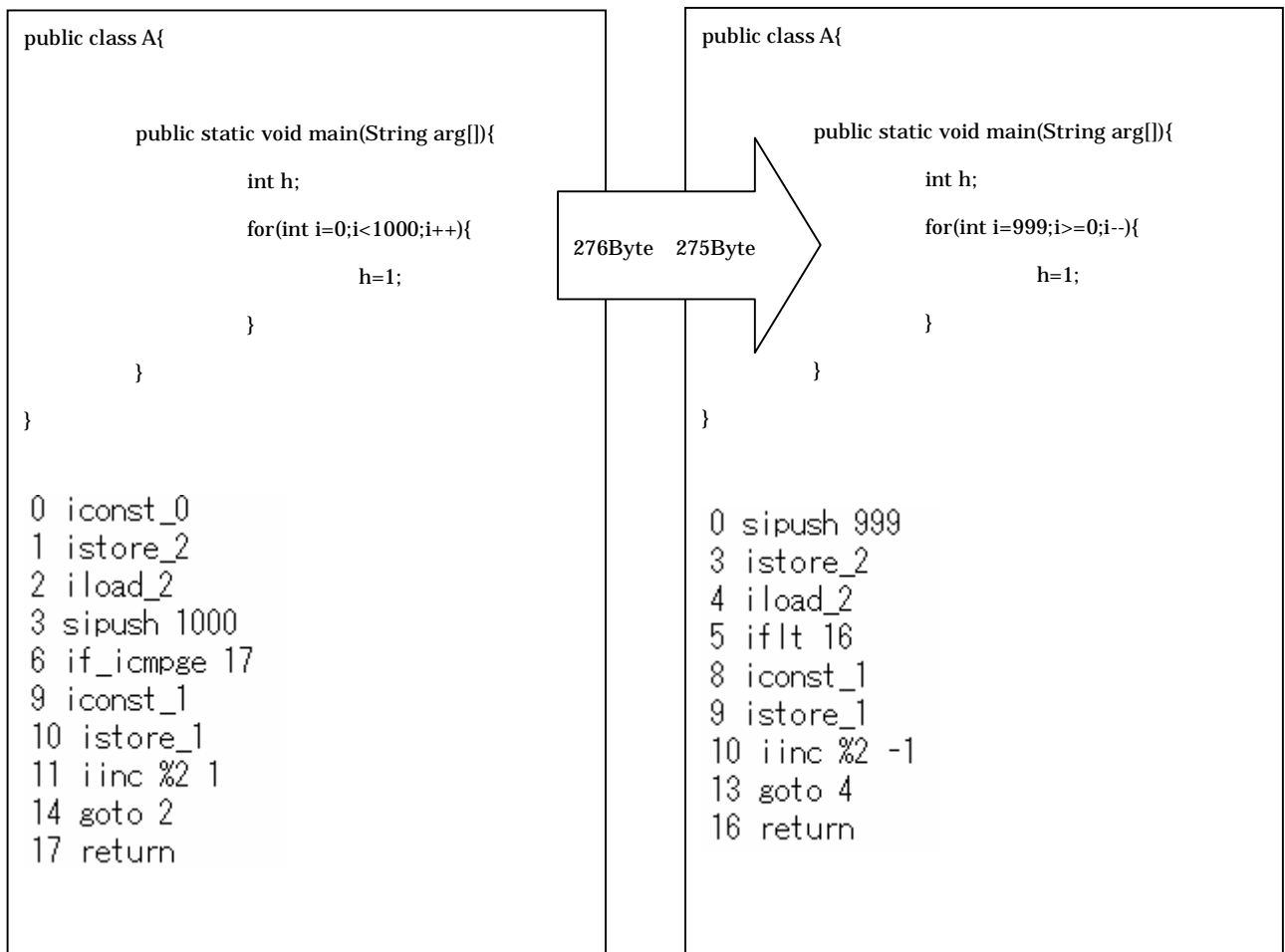
If there is “for” loop statement in the source code, consider to change it not to deploy each instance in the loop statement. Class size might be bigger , but the jar file size will be smaller. It depends on the case ,but generally around 7th loop is the turning point. If you know the times of the loop beforehand, try to write for each statement instead of “for” loop.



Decrement of "for" loop

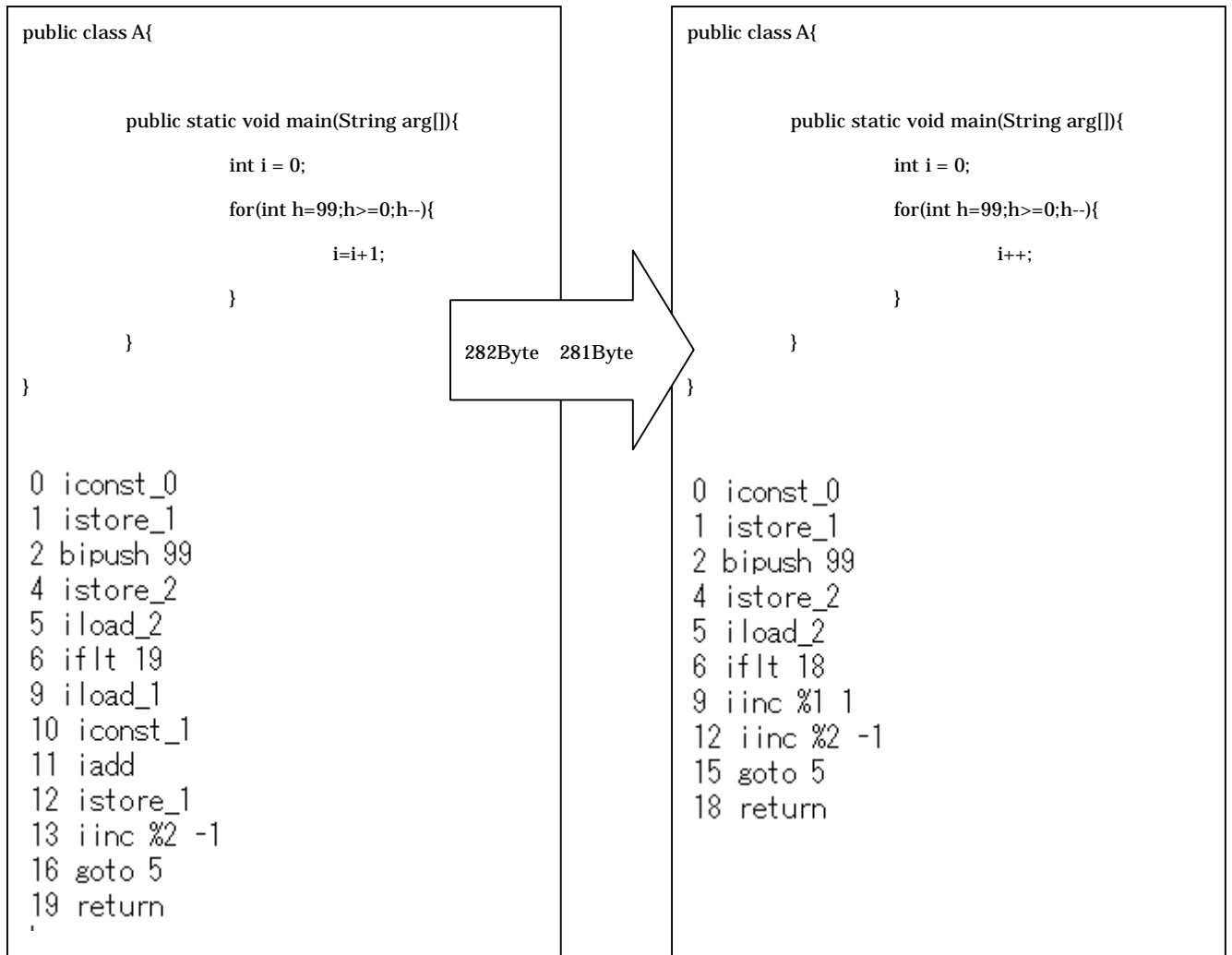
If there is comparison statement in the source code, consider to compare with 0. A command of exclusive use is prepared for bite code level with comparing value with 0. So procedure is faster and also the size would be smaller.

In case you use "for" statement, compare the loop variable with the value of 0 using a loop variable decrement instead of increment.



Arithmetic Operation

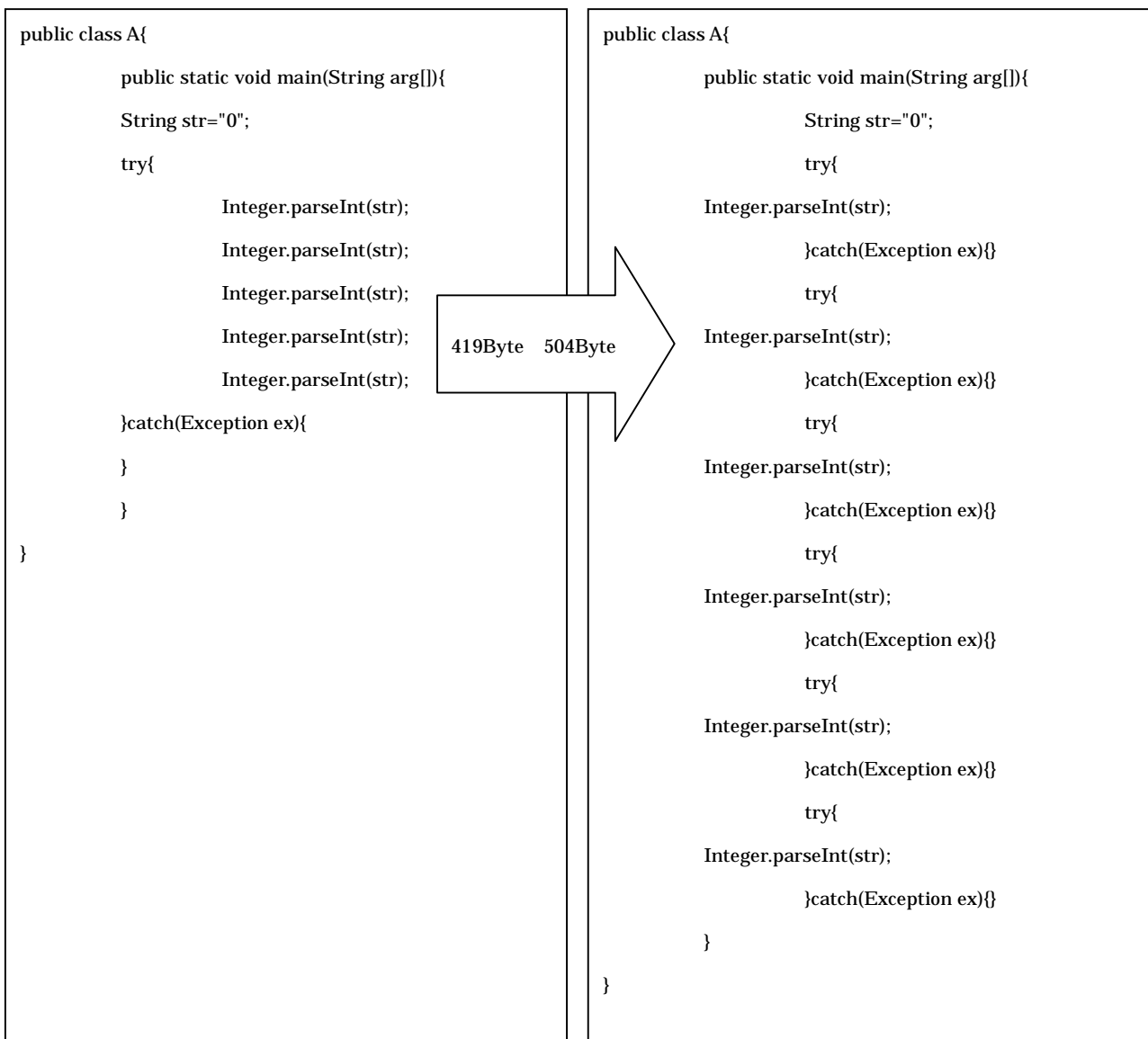
If there is Arithmetic operation like “i=i+1” in the source code, consider to use “i+=1”. If you write “i=i+1”, the variable is deployed 2 times. So it’s better to use “i+=1” to reduce the size of the application.



Try to summarize “try-catch” block

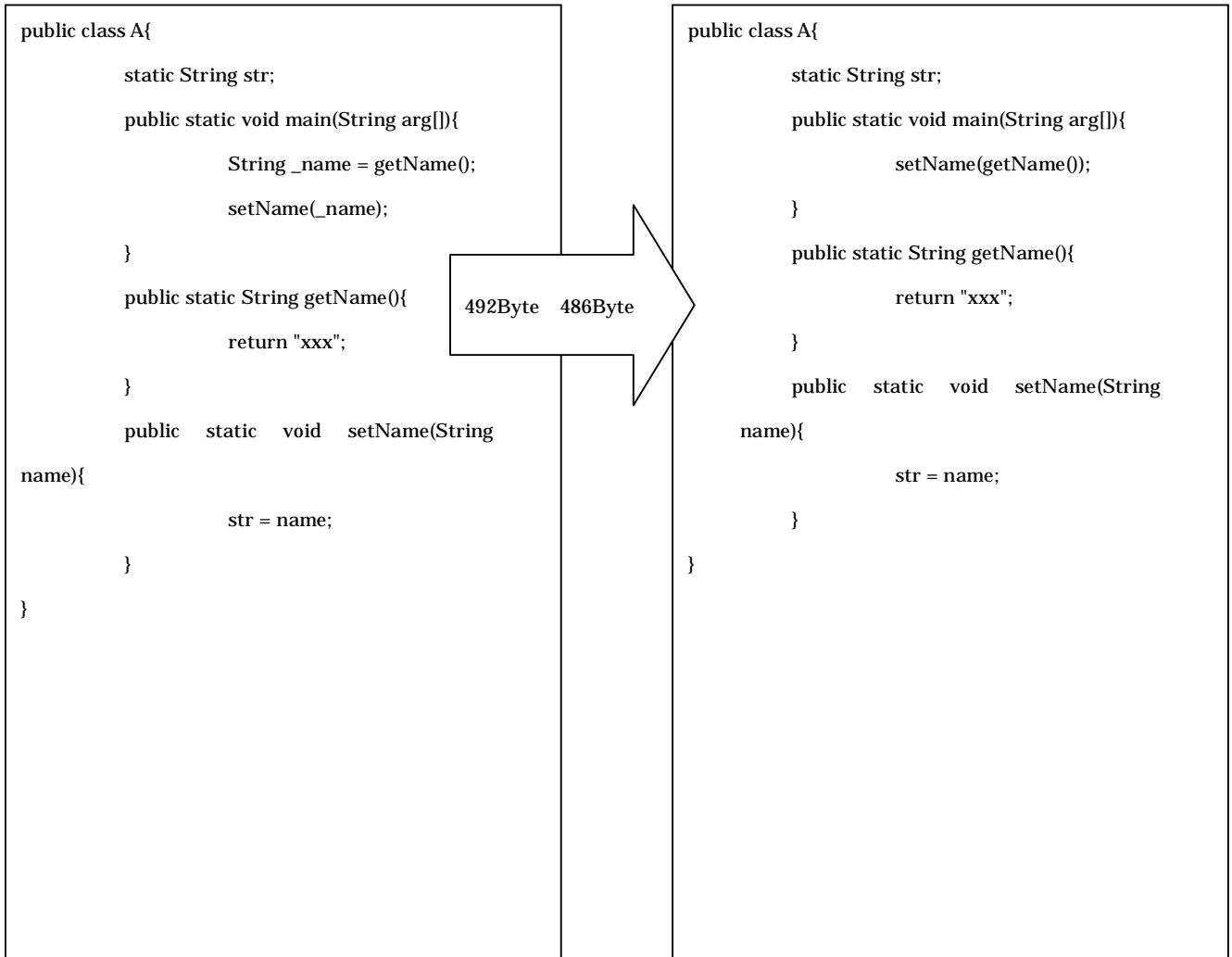
If there is try-catch statement in the source code, consider to use it with the wide scope. Using try-catch statement costs a lot. So try not to use it inside the loop. Try to use it for whole method once or throw the exception to upper method.

Omit the machine language command code for following source code on the relation of the size of this documentation.



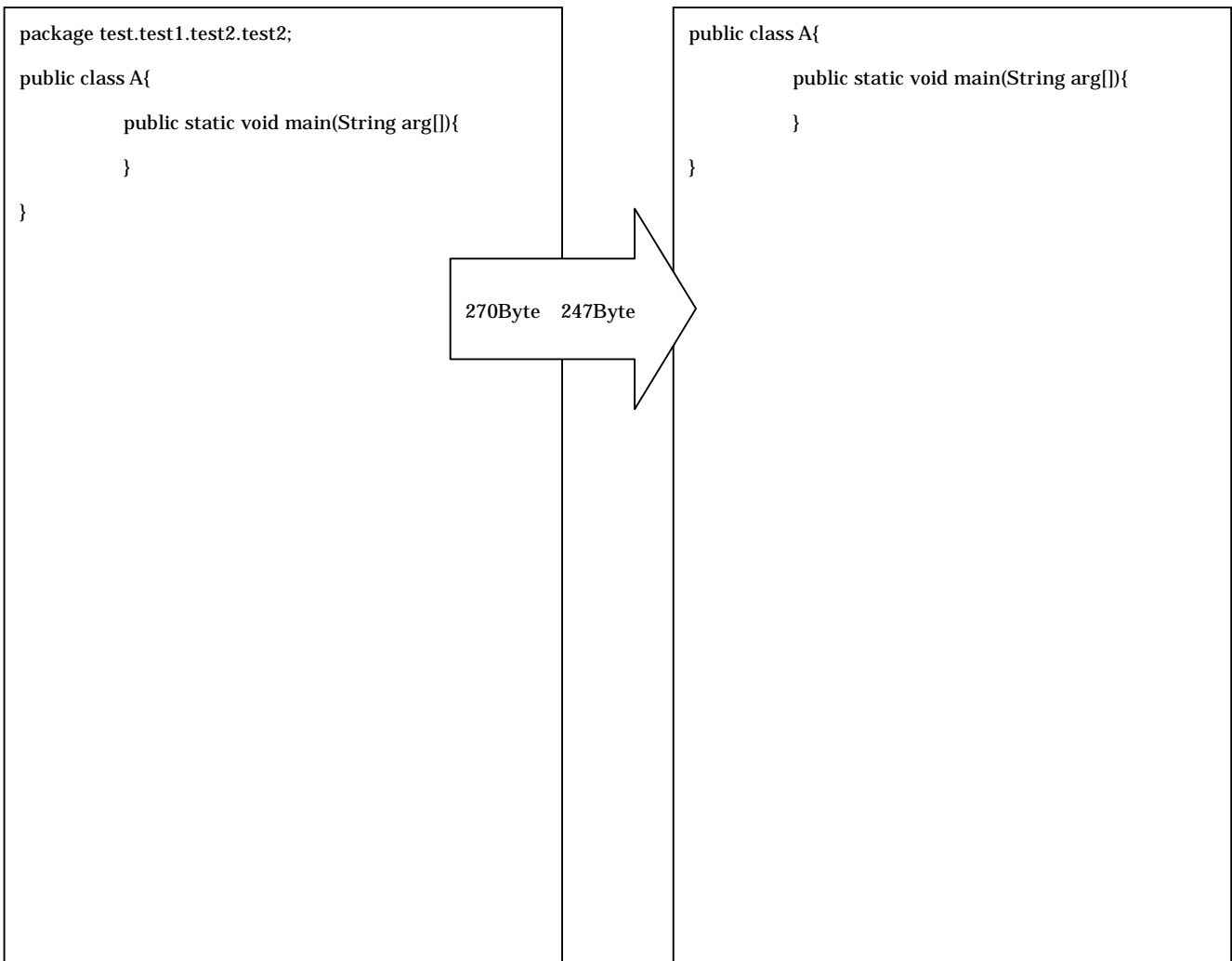
Try to use "Existing value"

If there is method with the argument in the source code, consider to write the argument directly. If you write the argument directly, it wouldn't create the useless instance. So if it is possible to set the argument directly, try not to make the variable to put the method return value.



Try not to use the package

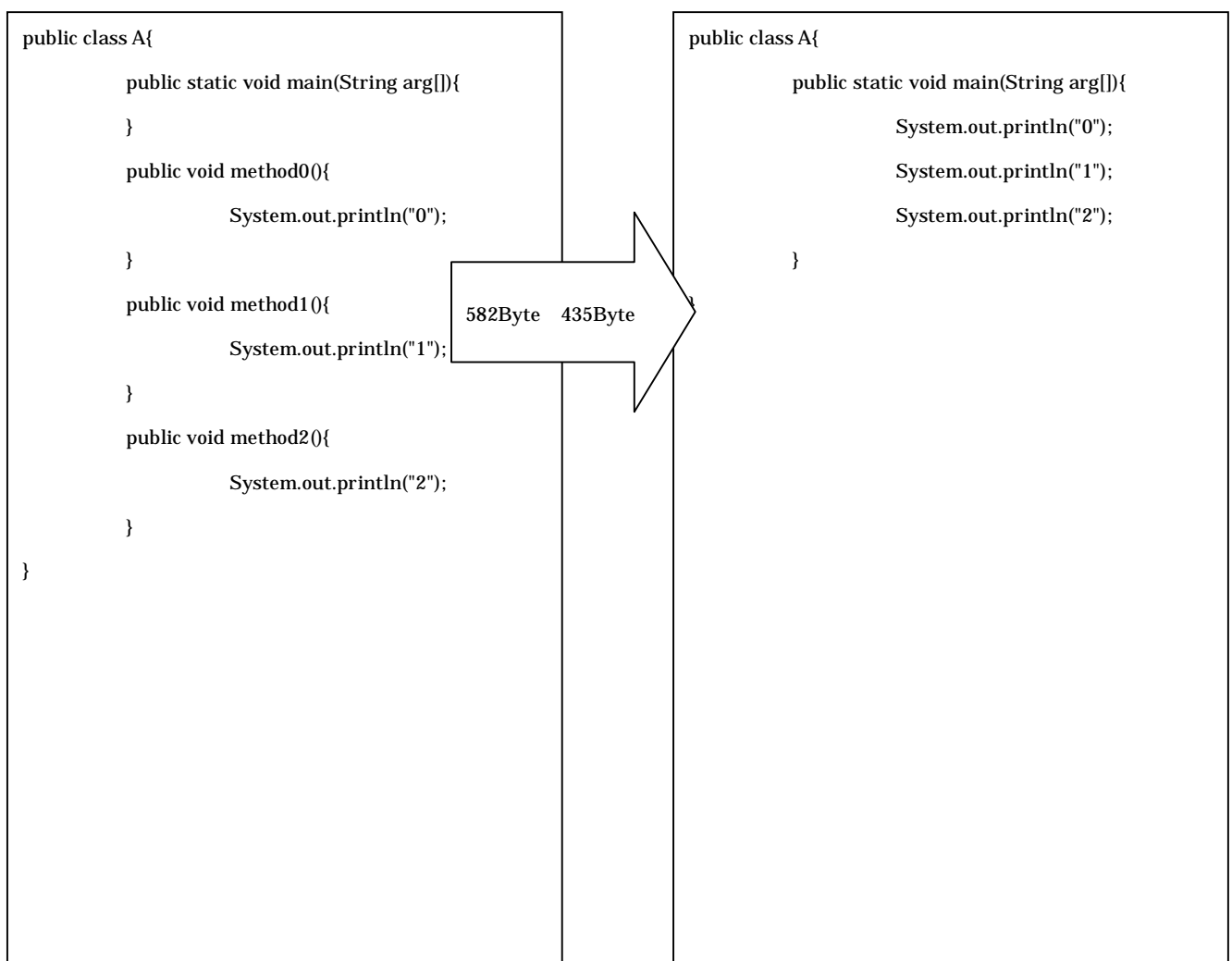
If there is package in the source code, consider not to use package. Package is useful for dynamic loading, but for DoJa the system is not supported. So it only make the constant pool size larger. Then if it's not needed, try not to use the package.



Method Optimization

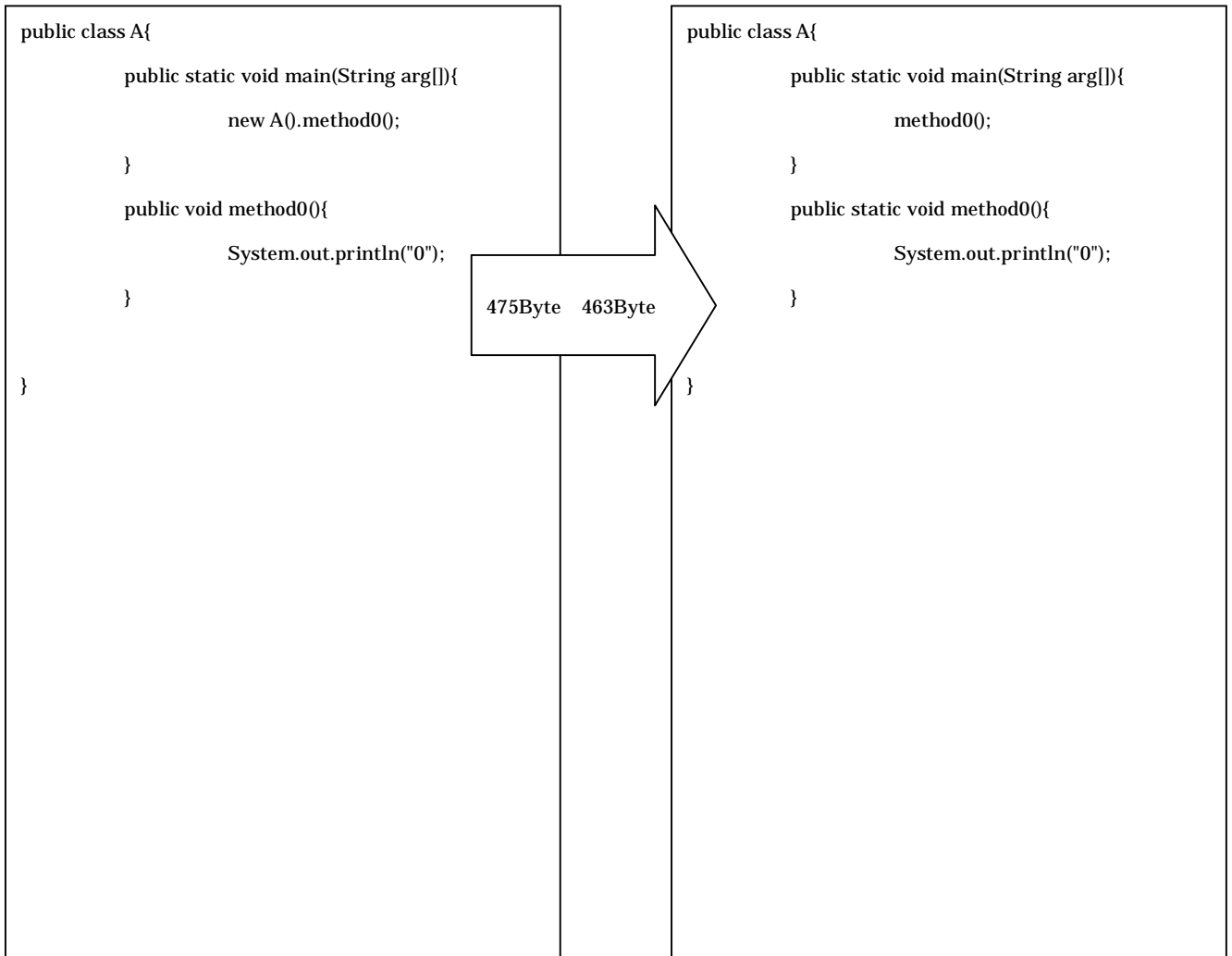
Try to decrease method

If there are many methods in the source code, consider to decrease them. Making method increase the size, so if the method is not called for many times, it's better to use in-line deployment for decreasing the size. But if the method is called many times, making method and reuse them might make the size smaller. It depends on the application, so you can check it each time.



Try to use the static method

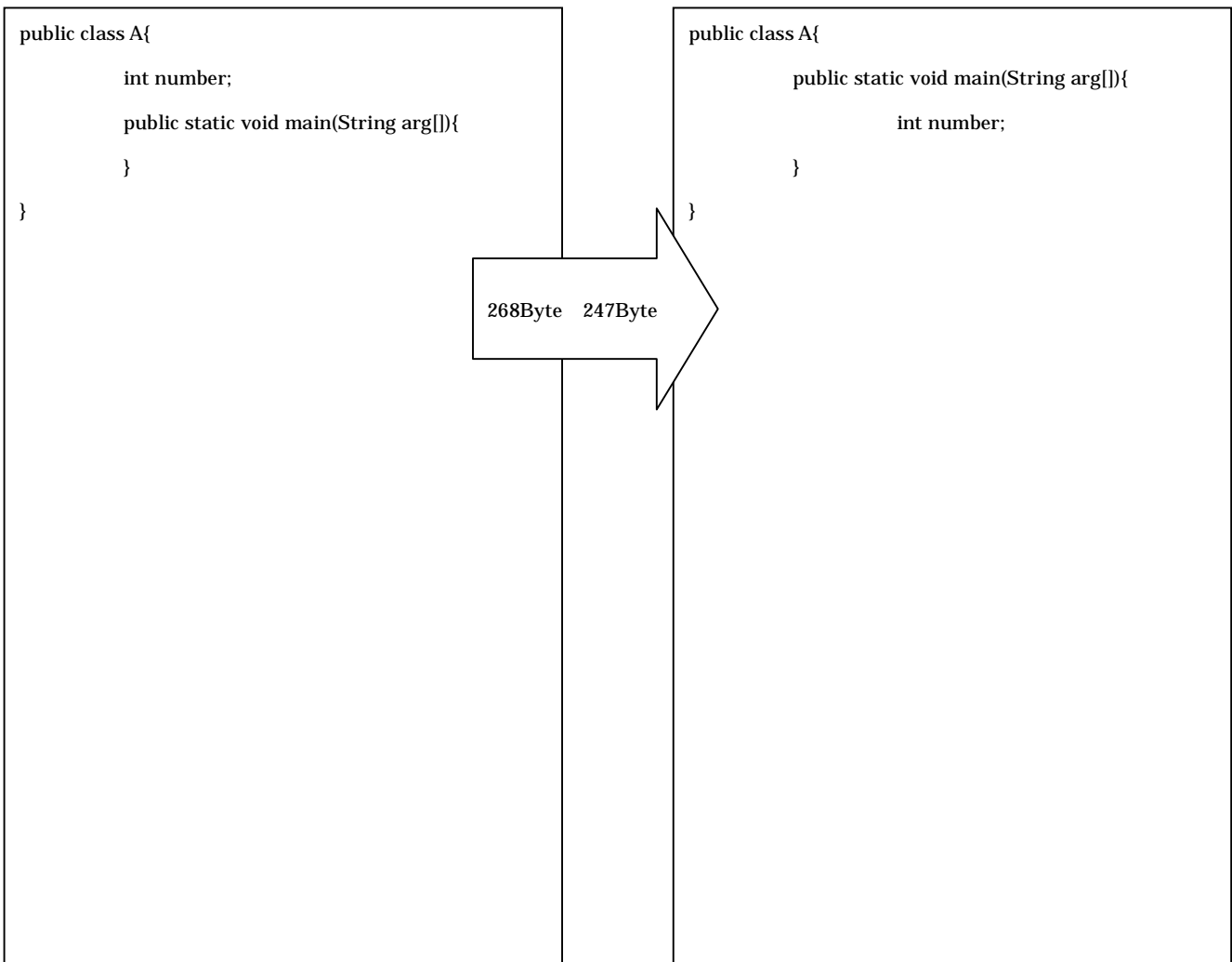
If there are methods in the source code, consider to make them as static method. Static method doesn't include the object information, so it is smaller than the instance method.



Variable Optimization

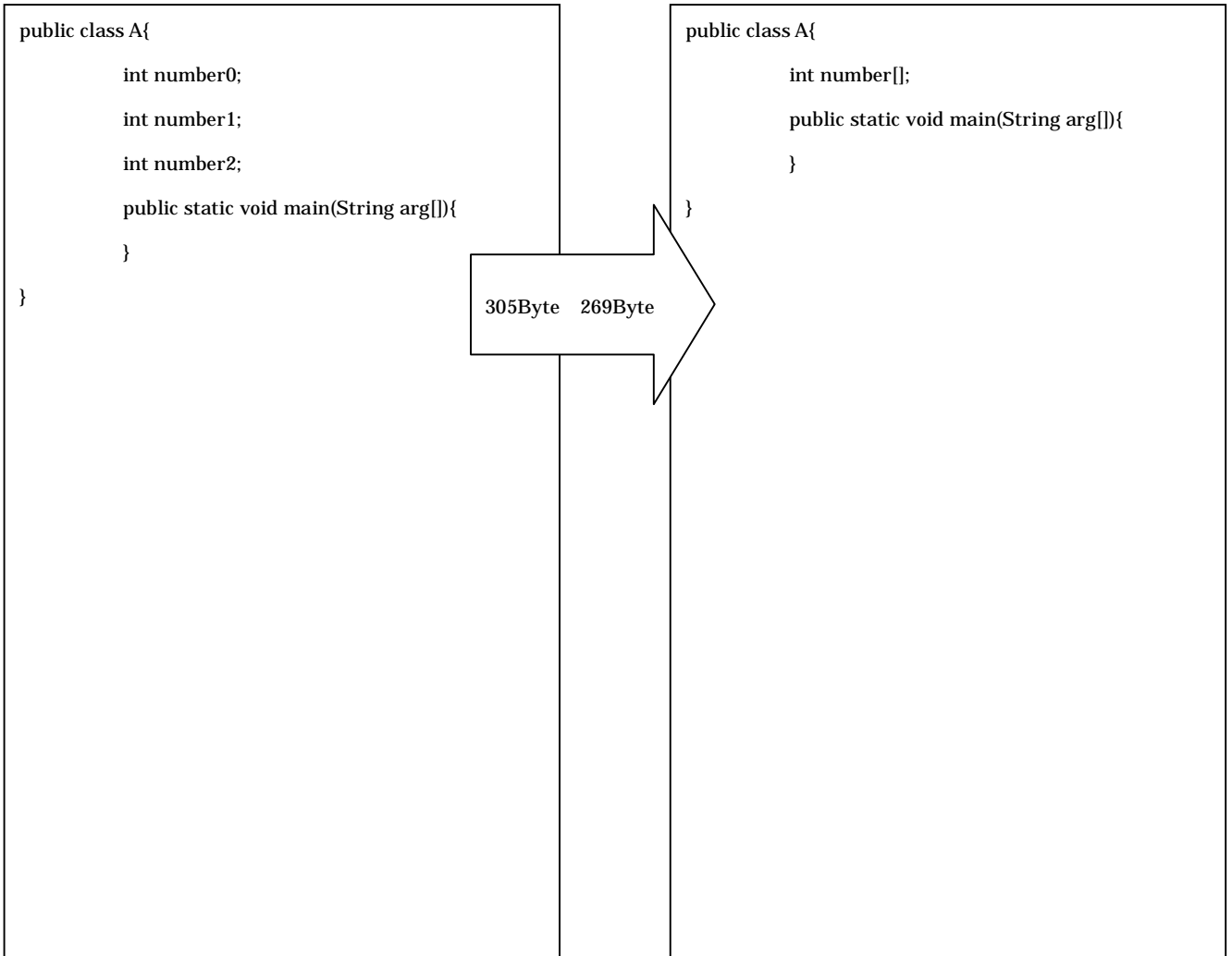
Try to use the local variable

If there are variables in the source code, consider to make them as local variable. The variable declared in the Class takes the memory domain. And the variable declared as a local variable takes the place in the stack area. If you see the code with assembler level, the size is smaller.



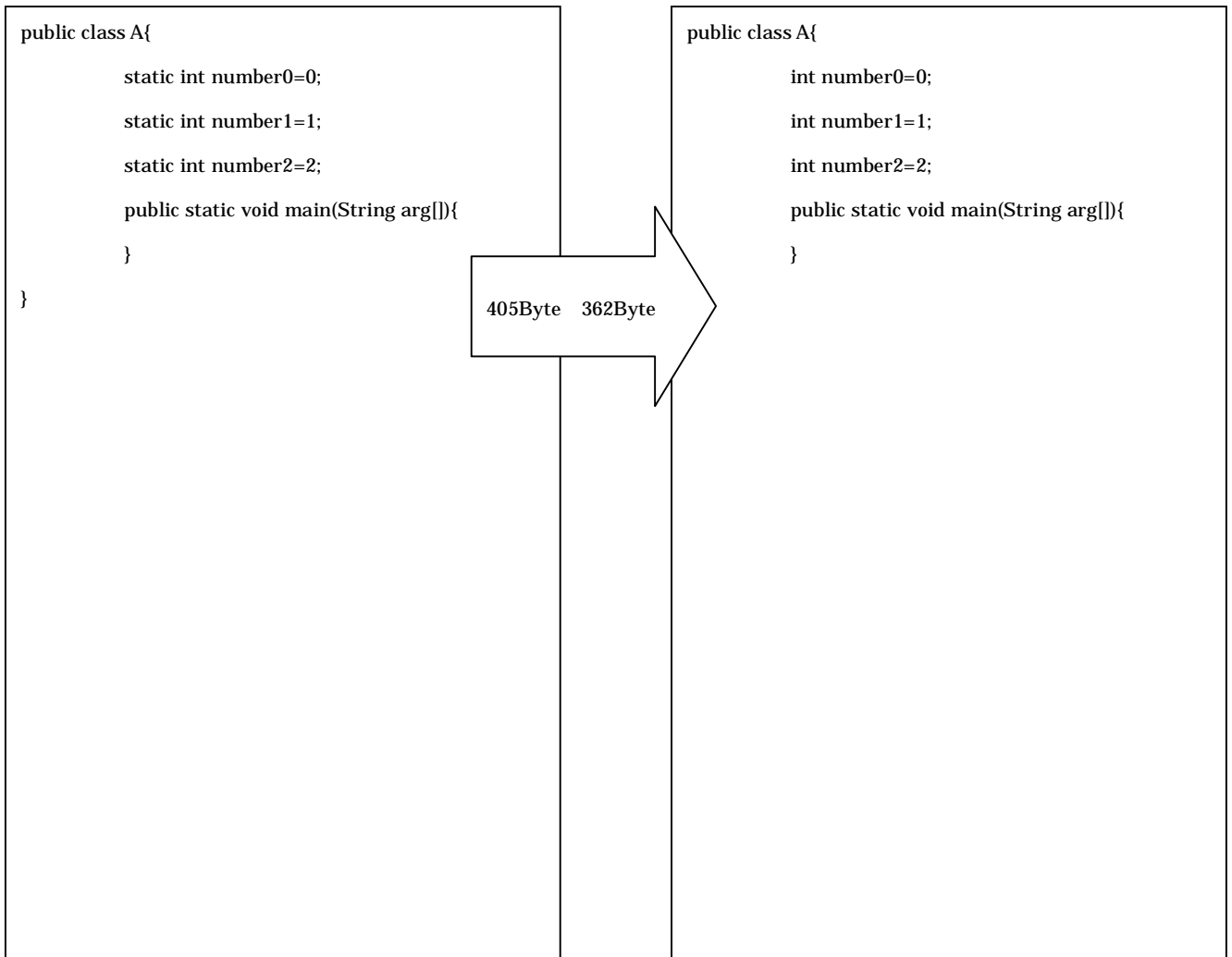
Try to declare the array variable

If there are variables in the source code, consider to use array. Decreasing the size of name data in constant pool area, changing same class type variables into array makes it possible. If there are many variables which cannot change into local variables, try to change them into arrays.



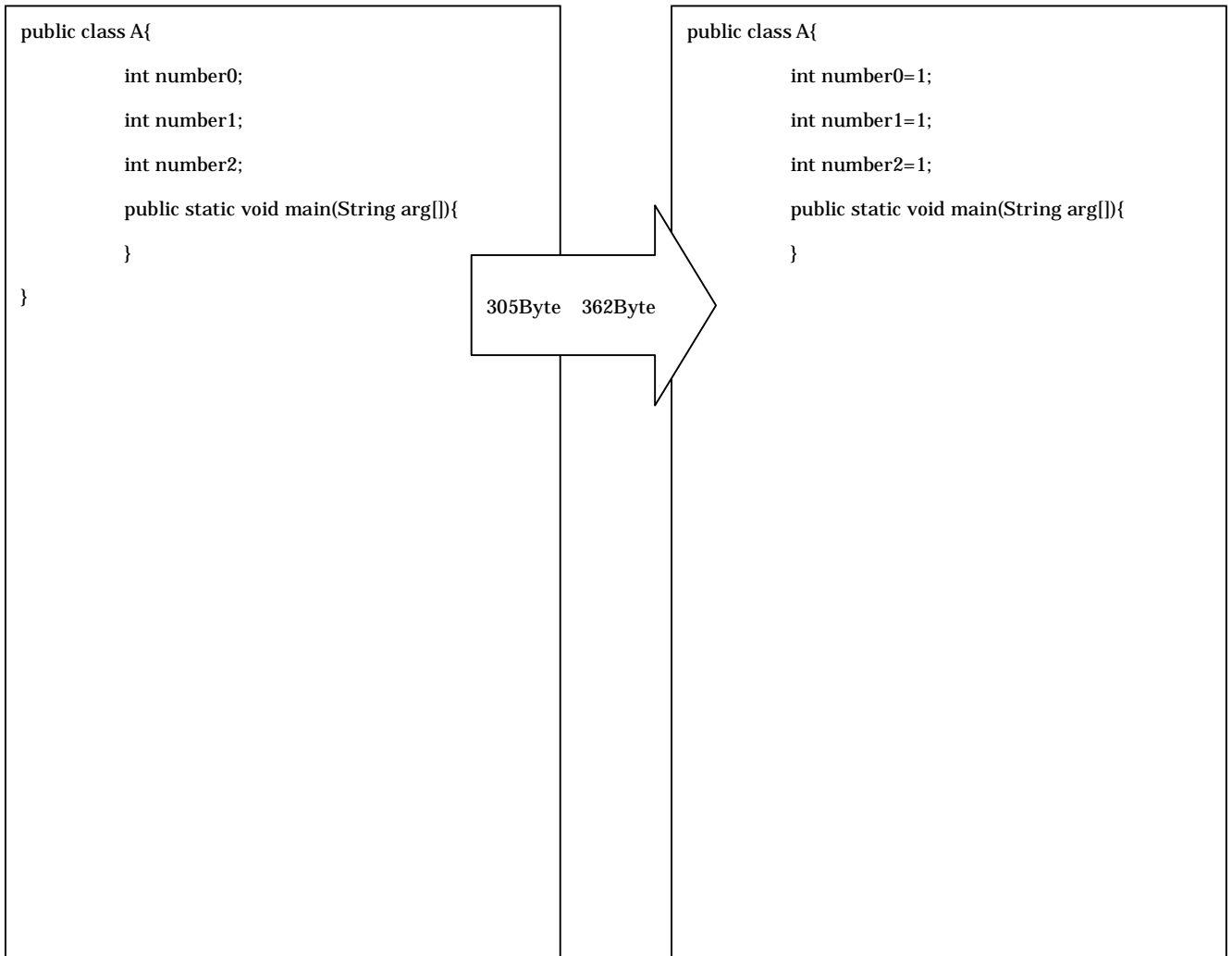
Try not to use the static variable

If there is static variable in the source code, try to change it to non-static variable. The variable name is stored in the constant pool, but the static variable reserve the new memory area. So it is better not to use static variable.



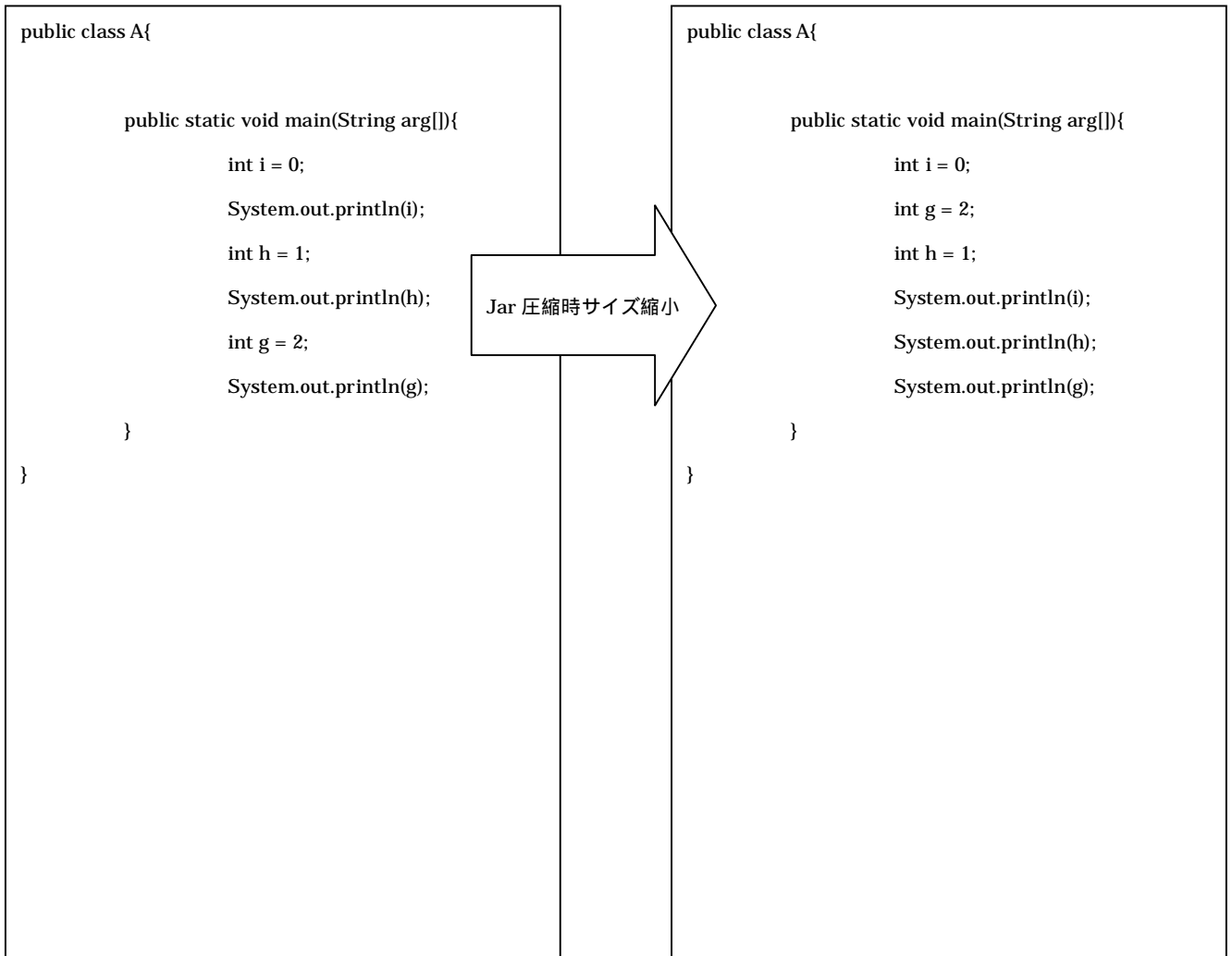
Try to use the default variable value

If there is variable in the source code, try to use the default variable value. The variable value is initialized when it is declared. So try not to initialize with the value which is not default.



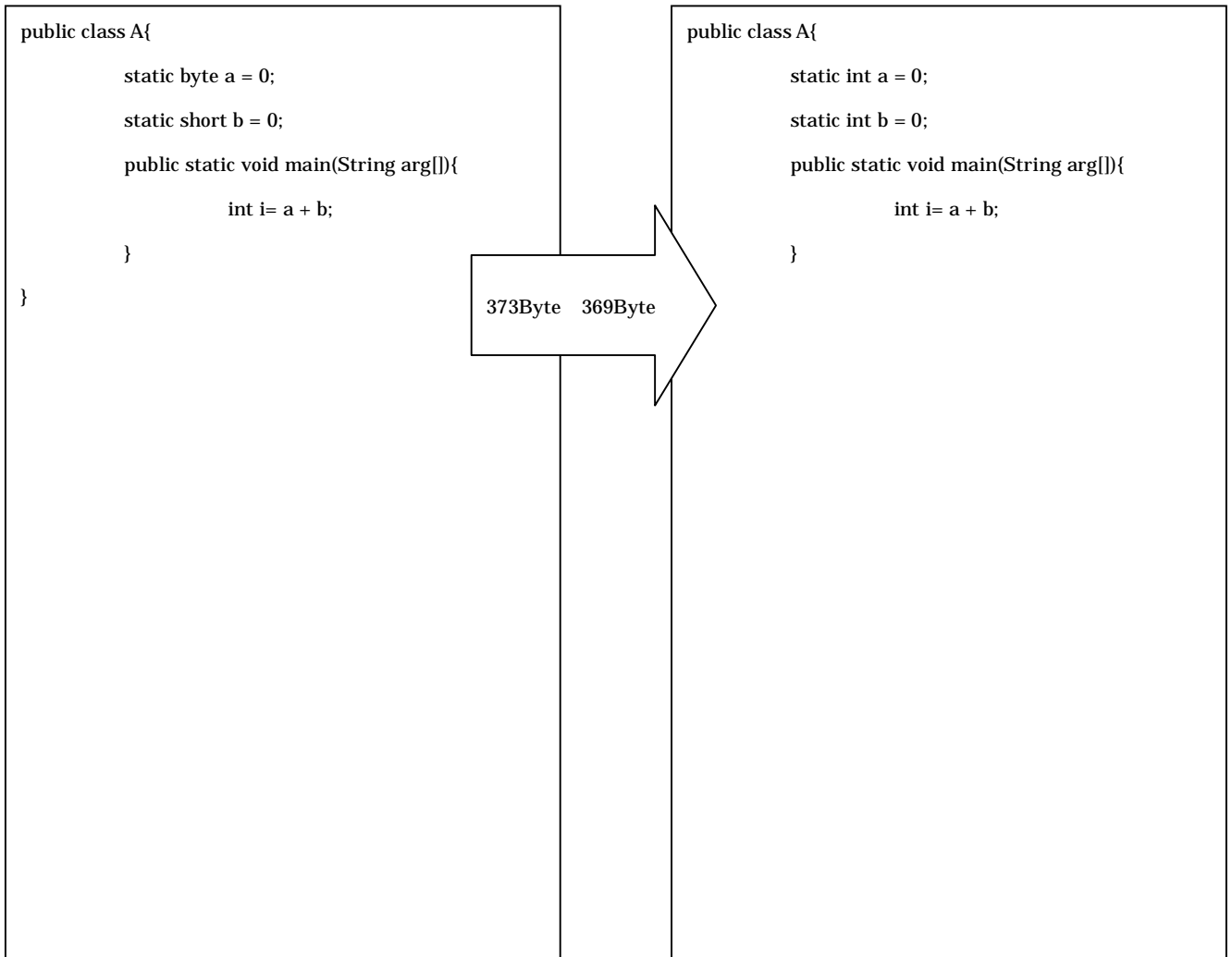
Try to declare the local variables at one place

If there are local variables in the source code, try to declare them at one place. According to Jar pressing, it is better to declare the local variables at one place to decrease the size.



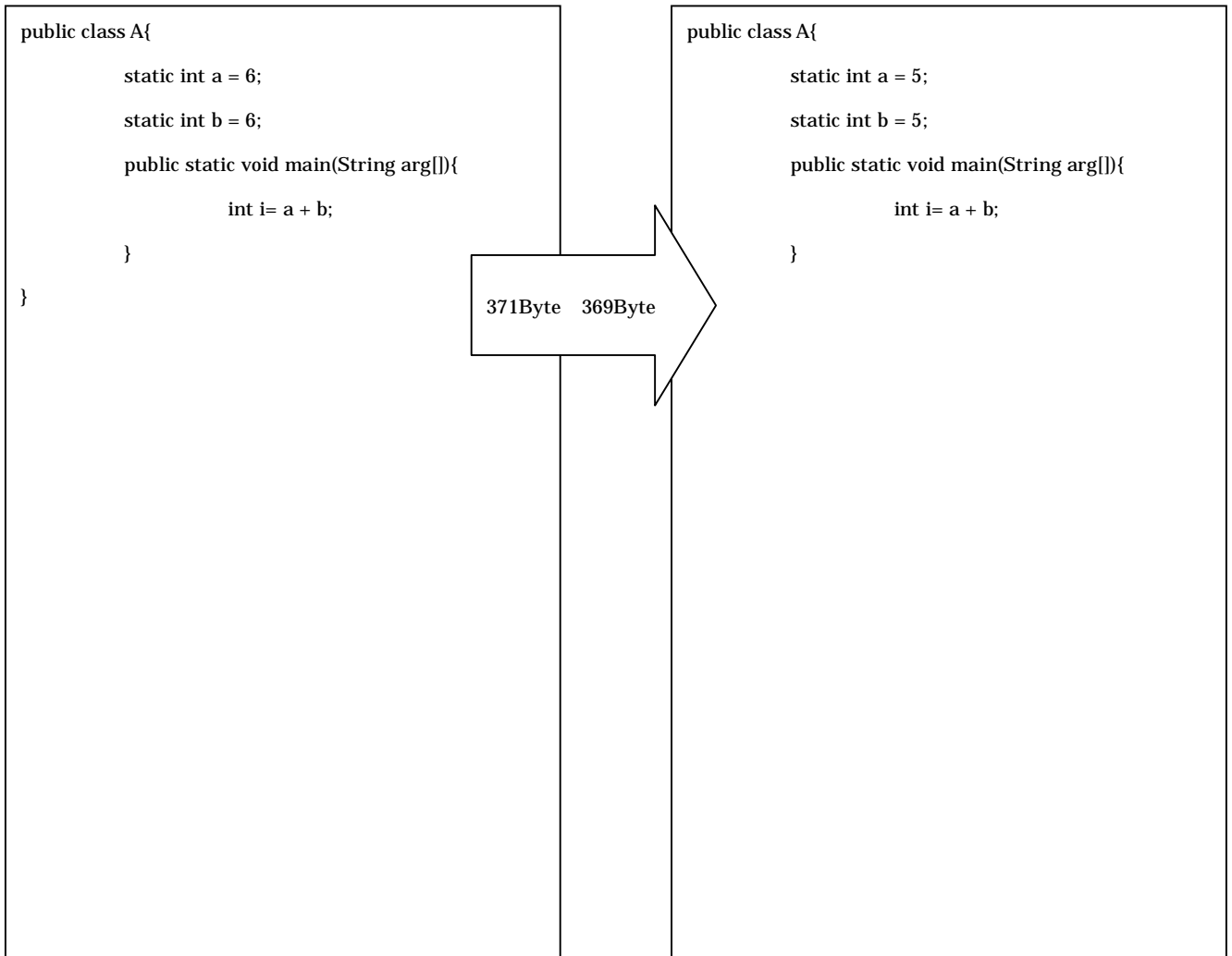
Try to use “int” for the number value

If there are number value in the source code, try to use “int”. Data size of “byte” or “short” are smaller than “int”, but actually Class size become larger because internally they are converted into “int”.



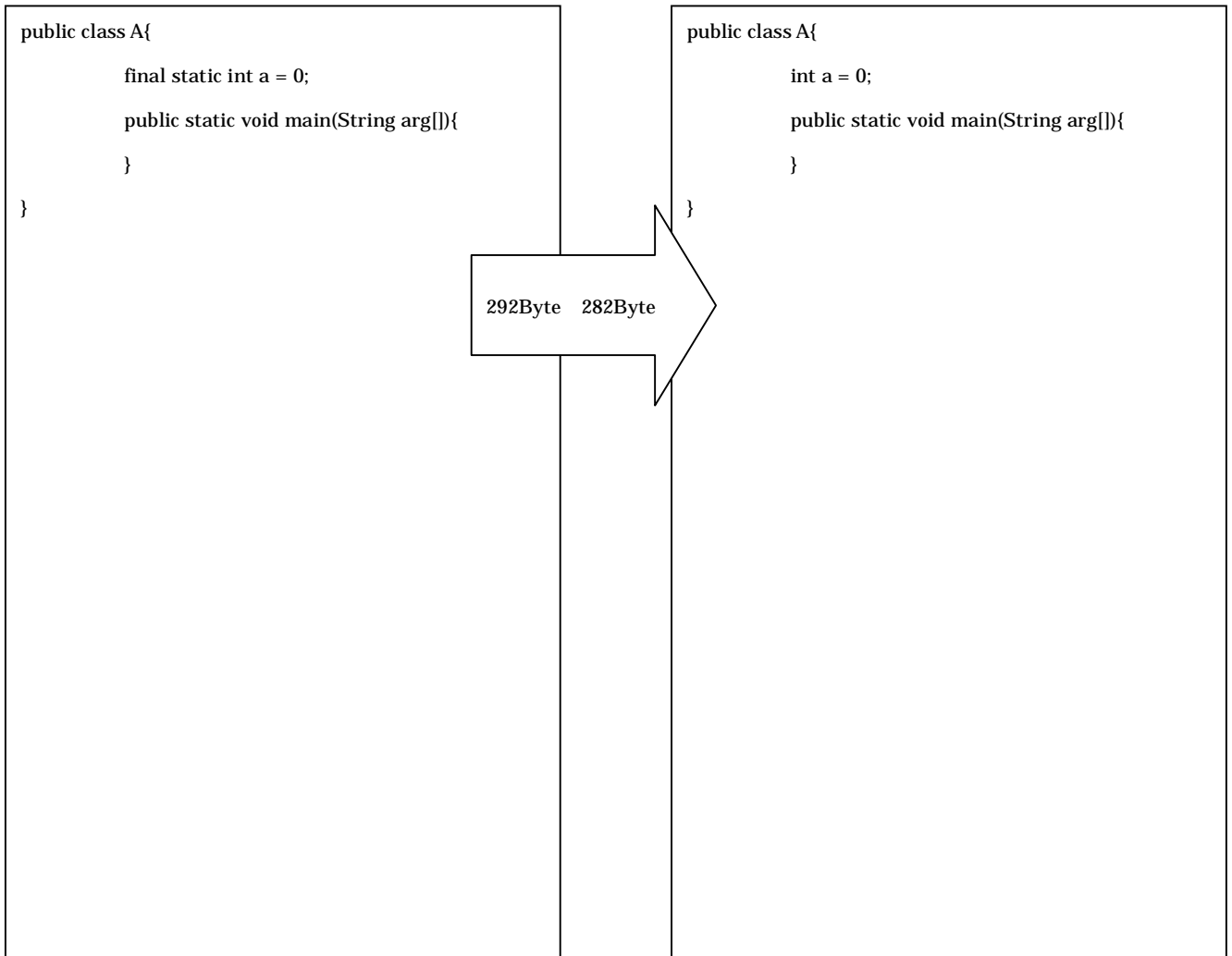
Try to use a number ranged from -1 to 5 preferentially

If there are number value in the source code, try to use a number ranged from -1 to 5. Those number range is treated specially in bite code level. So if it is possible try to those number value preferentially.



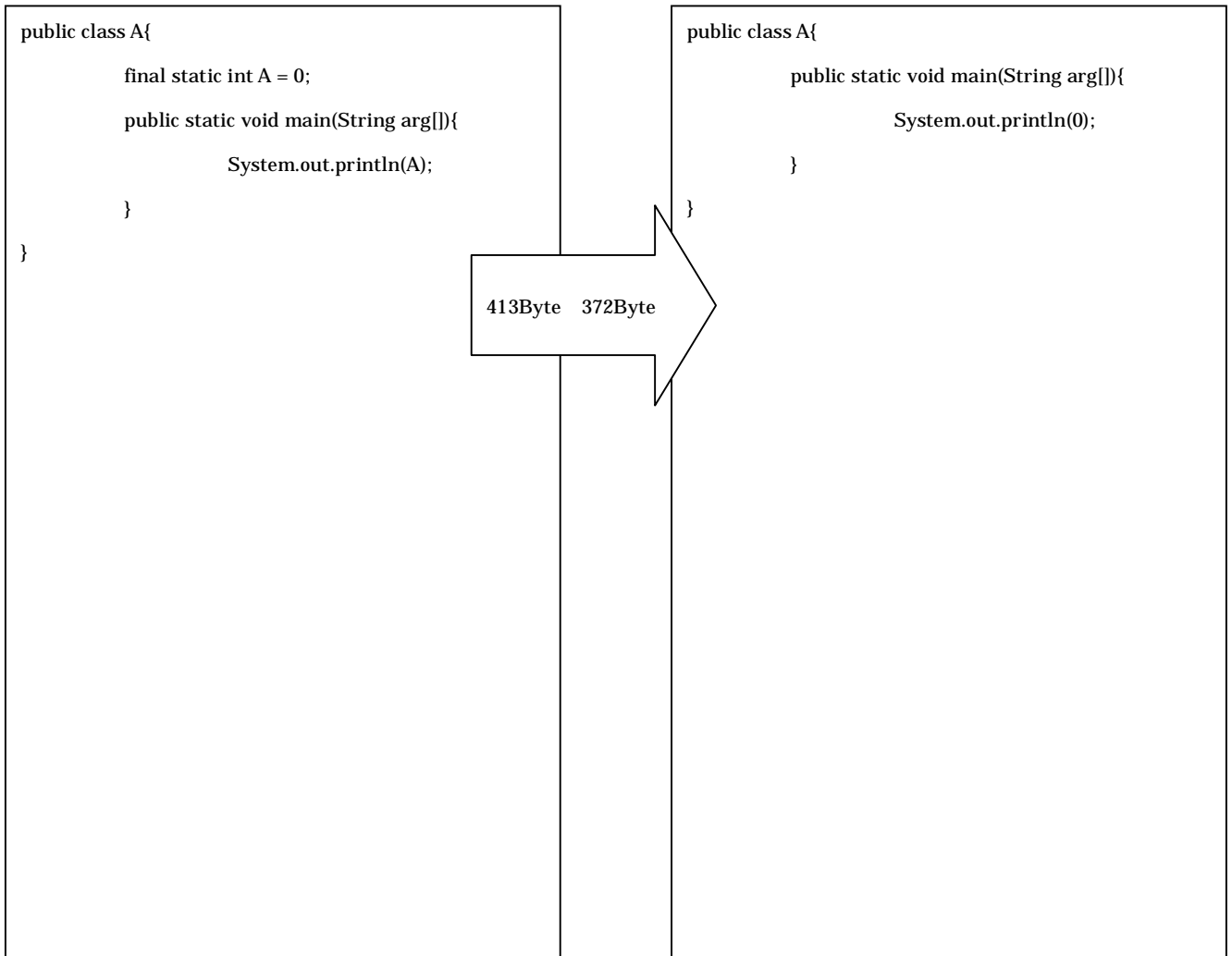
Try to declare the variable instead of the constant variable

If there are the constant variable in the source code, try to use the variable.
The constant variable such as “final static” is larger than the non-identifier variable.



Try to use the specific value instead of the constant variable

If there is the constant variable in the source code, try to use the specific variable directly. The size tends to be smaller if the specific value is set directly in stead of the constant variable such as “final static”.

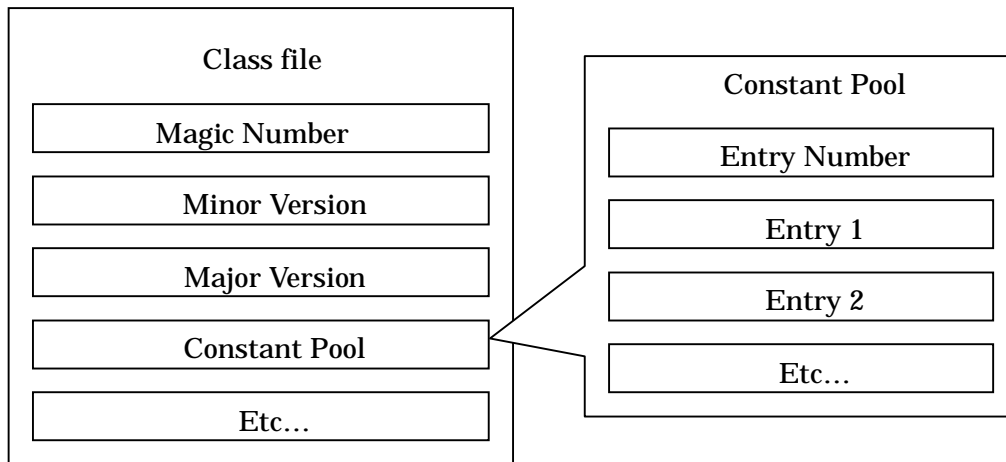


Class Optimization

Try to decrease the Class

If there are many Class in the source code, try to reduce them.

Class file is expressed as a bite sequence. The structure is as follows.



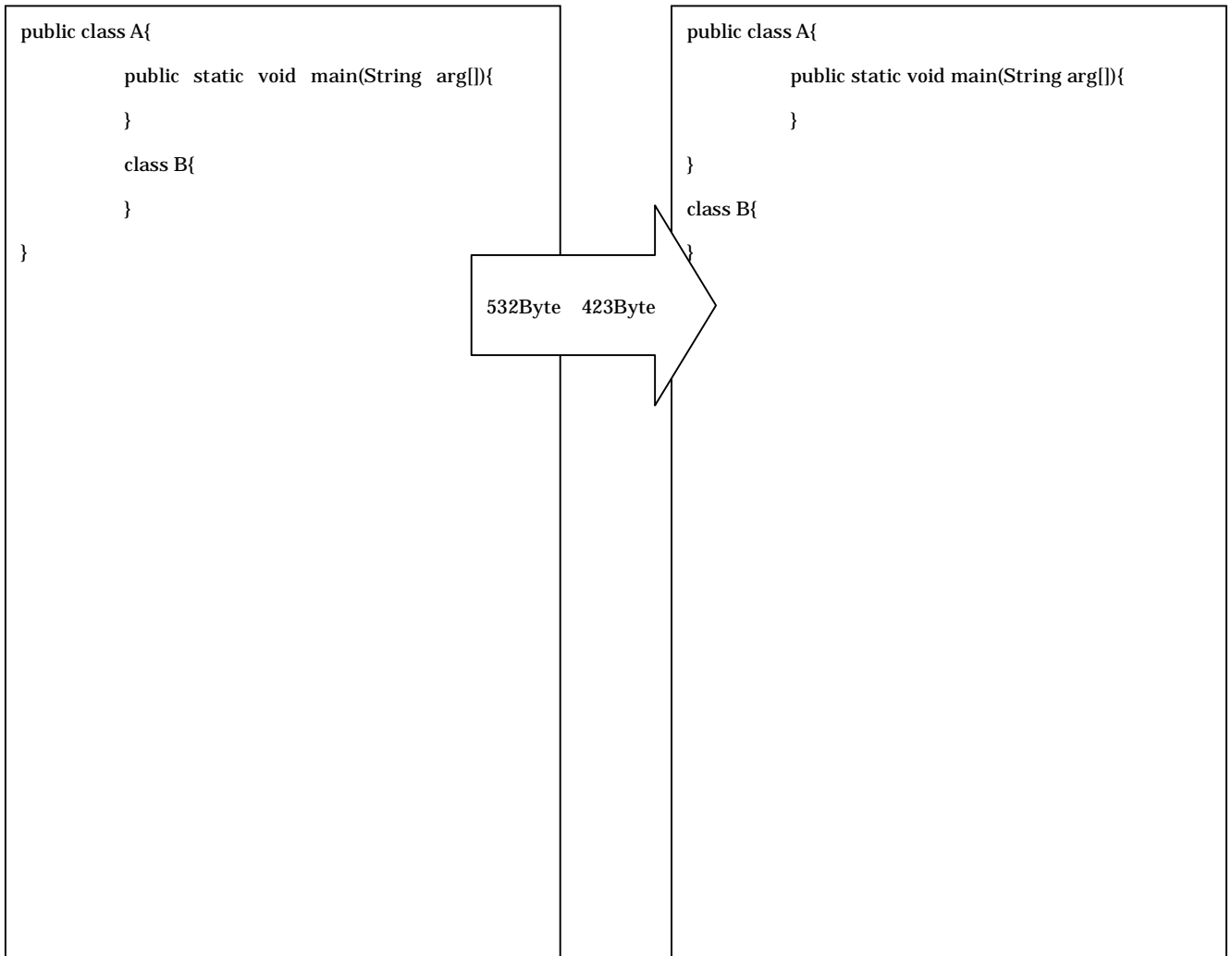
Constant pool is the array of the constant pool entry. And it occupy the 30% - 50% of the Class file capacity.

The variable which is used in one Class appear once in constant pool even if the variable is refered many times. But the variable shared among many Class takes place in each Class data in the constant pool. So Class costs a lot for the memory size.

If you use Panel Class, only make one Class extends Iapplication. If you use Canvas Class, make only 2 Class , Class extends IApplication and Class extends Canvas if it's possible.

Try not to use the inner Class

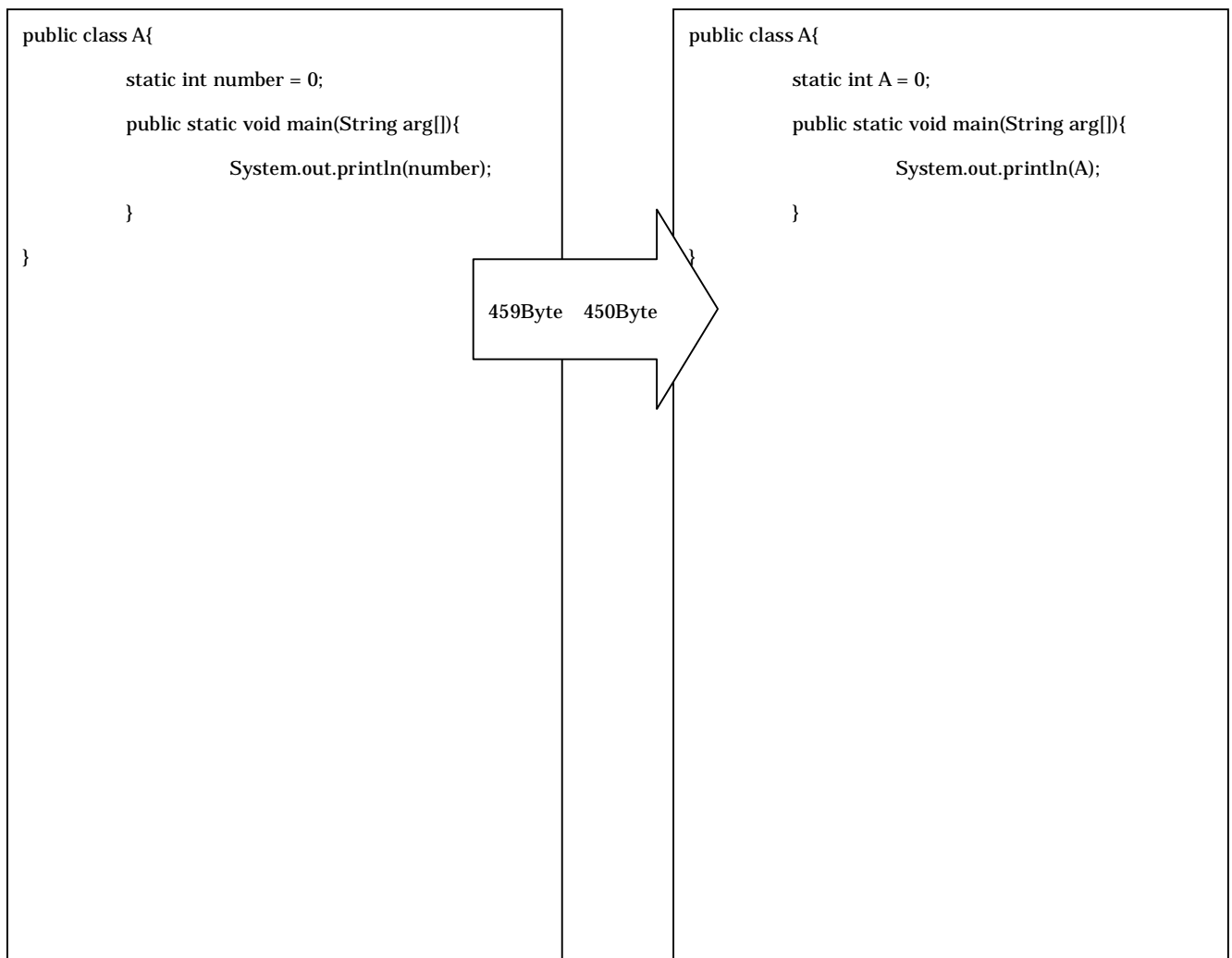
If there is inner Class in the source code, try to make non-inner Class. Inner Class has some data about itself, so it's better to make another Class instead of them to reduce the size.



Naming Optimization

Try to share and shorten the method and variable names

If there are variables in the source code, consider to make the same name and make their names as short as they can. Class, method and variable names store their name data in the constant pool. So make them short reduce the size. Also using same name for the variable and method share the same name data in constant pool then it is effective to reduce the size of the application.



Binary Optimization

If you try to optimize the application in binary level. Jakarta Project is developing the library called BCEL (Byte code engineering library). So you can use them.

The Byte Code Engineering Library is intended to give users a convenient possibility to analyze, create, and manipulate (binary) Java class files (those ending with .class). Classes are represented by objects which contain all the symbolic information of the given class: methods, fields and byte code instructions, in particular.

Such objects can be read from an existing file, be transformed by a program (e.g. a class loader at run-time) and dumped to a file again. An even more interesting application is the creation of classes from scratch at run-time. The Byte Code Engineering Library (BCEL) may be also useful if you want to learn about the Java Virtual Machine (JVM) and the format of Java .class files.

You can download the library from : <http://jakarta.apache.org/bcel/>

The screenshot shows a Java class file viewer with the following content:

1 CONSTANT_Methodref
void [Object.<init> \(\)V\(\)](#)
• [Class index\(4\)](#)
• [NameAndType index\(21\)](#)

2 CONSTANT_Fieldref
[a I](#)
• [Class\(3\)](#)
• [NameAndType\(22\)](#)

3 CONSTANT_Class
[Test](#)
• [Name index\(23\)](#)

4 CONSTANT_Class
[Object](#)

void <init>()
Attributes
• [Code](#)
 o [LineNumberTable](#)

Byte Instruction Argument

Byte offset	Instruction	Argument
0	aload_0	
1	invokespecial	Object.<init> ()V:void
4	aload_0	
5	iconst_0	
6	putfield	a I
9	return	

public static void main(String[])
Attributes
• [Code](#)
 o [LineNumberTable](#)

Access flags Type Field name

Access flags	Type	Field name	Value
final	int	a	= 0
static final	int	b	= 0
static final	int	c	= 0
static final	int	d	= 0
static final	int	e	= 0

Image: HTML binary data