Export & CAP files

Introduction

- En raison des contraintes mémoire des périphériques supportant la technologie Java Card, cette technologie a défini son propre format pour l'interopérabilité binaire: le .CAP et le .EXP.
- Celui-ci n'est pourtant pas si différent du format .class comme nous le verrons
- Le *« write once, run anywhere »* est la plus significative des caractéristiques du langage Java.

JAVA

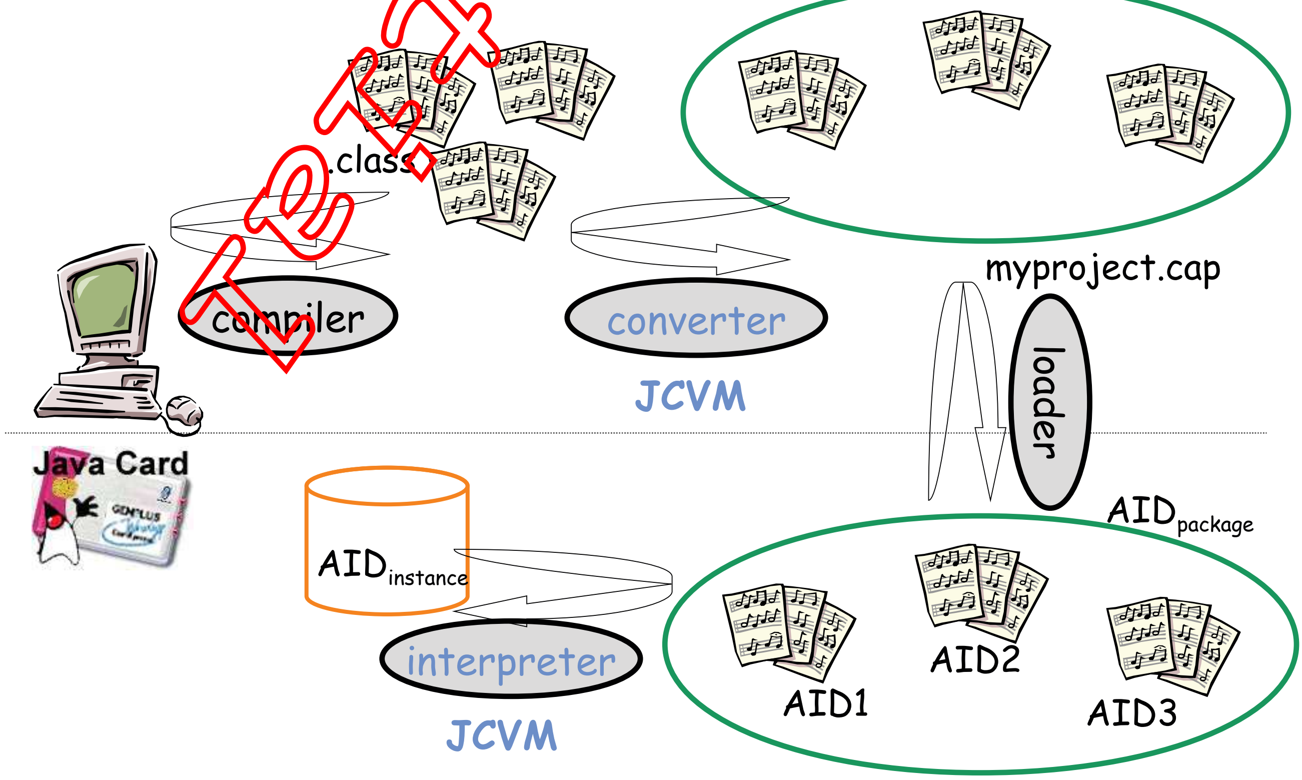
- The *Java Virtual Machine Specification* defines a Java virtual machine as an engine that loads Java `class` files and executes them with a particular set of semantics.
- The `class` file is a central piece of the Java architecture, and it is the standard for the binary compatibility of the Java platform.

Class file format

- There are 10 basic sections to the Java Class File structure:
  - Magic Number: this is currently `0xCAFEBABE`
  - Version of Class File Format: the minor and major versions of the class file
  - Constant Pool: Pool of constants for the class
  - Access Flags: for example whether the class is abstract, static, etc.
  - This Class: The name of the current class
  - Super Class: The name of the super class
  - Interfaces: Any interfaces in the class
  - Fields: Any fields in the class
  - Methods: Any methods in the class
  - Attributes: Any attributes of the class (for example the name of the sourcefile, etc.)
    - See `https://docs.oracle.com/javase/specs/jvms/se6/html/VMSpecTOC.doc.html`

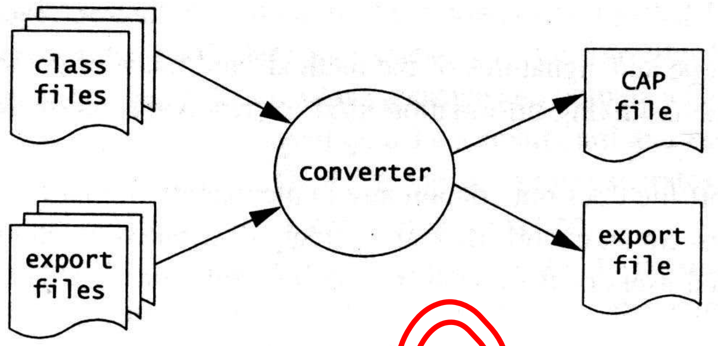
Java Card

Applet development



Converter

- It processes one class at a time.
- Its conversion unit is a package.



- It is in charge to use the export files (classes already loaded on card) and to transform the current class files to a CAP file.
- Often the converter calls the byte code verifier prior to convert the file.
- Enable to obtain a bytecode with a format adapted to the smart card **but** is 'platform dependant' due to export files that make correspondence between converted code and card API implementation

Two specific file formats

- The CAP (Converted Applet) file format
  - Contains all the classes and interfaces from **one package**
  - Semantically, is equivalent to a set of class (`.class`) files
  - Syntactically, differs a lot from class (`.class`) files
    - All "string names" are **replaced by "token identifiers"**
  - Loaded on card
    - *inside the card the format can be proprietary*
    - But in the fact, the CAP file is an "execute in the place" format and is often implemented as is.
- The EXP (Export) file format
  - Maintains the consistency between the originated class (`.class`) files and the resulting CAP file
    - Only for public (exported) data
  - Can be freely distributed, used during pre-linking phase
  - Not loaded into the card

JavaCard VM instruction

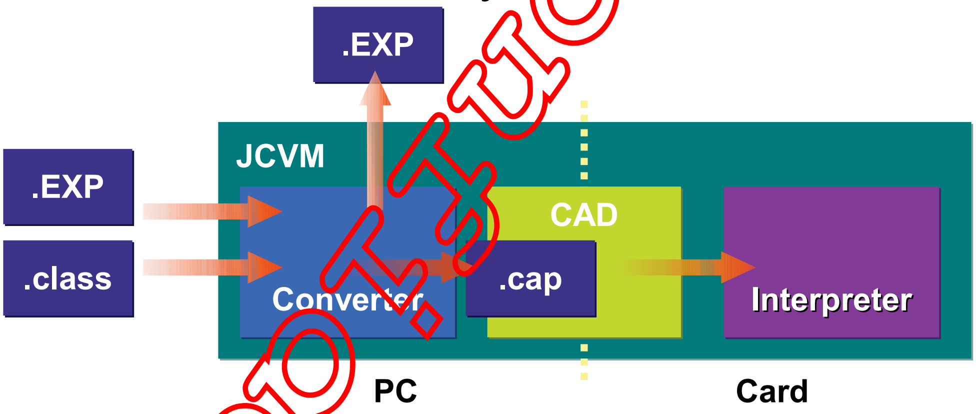
- Everything is based on *token*,
- A token is assigned for each public class, interface, method, fields…
- The export file will contain the conversion table between tokens and source code definition,
- In the card a *Constant Pool Table* will link the information.
- Two kinds of token:
  - Externally visible tokens: token associated with it to enable references from other packages to the item to be resolved on a device.
  - Internally visible items are not described in a package's export file, but some such items use *private tokens* to represent internal references.

Converter

- Réalise les taches que la machine virtuelle Java sur une station de travail doit réaliser au chargement des classes:
  - Vérifie que le chargement des images des classes Java sont bien formées.
  - Contrôle des violations du langage Java Card.
  - Réalise des initialisations de variables static.
  - Optimise le byte code
  - Résout les références symboliques aux classes, méthodes et champs

Split JCVM Architecture

- The converter (off-carte)
  - Class *loading, linking* and name resolution
  - *Verification*
  - Bytecode optimization and *conversion*
- The interpreter (on-card)
  - Bytecode *execution* and security enforcement



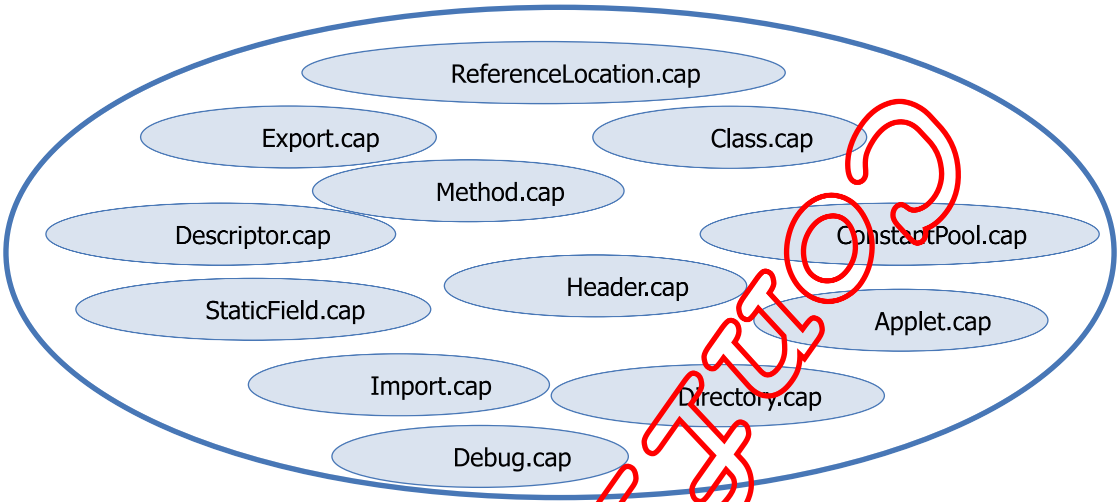
CAP file content

- Contains 12 different components
  - Header
  - Directory
  - Applet (optional)
  - Import
  - Constant Pool
  - Class
  - Method
  - Static Field
  - Reference location
  - Export (optional)
  - Descriptor
  - Debug (optional)

  + Custom component

.CAP is a JAR file

- Contains the different elements of the .CAP, but in separated files



Header Component

- It contains general information about this CAP file and the package it defines.
- Package information (*AID*)
- Example:

```
.header = {
   magic:  decaffed
   minor_version:  1
   major_version:  2
   flags:  4
   pkg_minor_version:  0
   pkg_major_version:  1
   pkg_AID_length:  6
   pkg_AID:  01.02.03.04.05.01
}
```

Directory Component

- It lists the size of each of the components defined in this CAP file:
  - size of components, number of imported packages, number of applets, …
- When an optional component is not included it is represented in the Directory Component with size equal to zero.
- The custom_component_info structure is defined as:

```
custom_component_info {
u1 component_tag
u2 size
u1 AID_length
u1 AID[AID_length]
}
```


Directory Component

```
.DirectoryComponent = {
   component_sizes = {
     Header:   13
     Directory:   31
     Applet:   20
     Import:   11
     ConstantPool:   98
     Class:   24
     Method:   575
     StaticField:   199
     ReferenceLocation: 72
     Export:   0
     Descriptor:   273
   }
   static_field_size_info = {
     image_size:   7
     array_init_count:   3
     array_init_size:   189
   }
   import_count:   1
   applet_count:   1
   custom_count:   0
}
```

Applet Component

- It contains an entry for each of the applets defined in this package.
- Number of Applets, *AIDs*, `install()` method offset
- If no applet is defined, this component must not be present in this CAP file
- Example:

```
.Applets = {
   AID: 01.02.03.04.05.01.01
   install_method_offset: @011c
}
```


Import Component

- It lists the set of packages imported by the classes in this package.
- It does not include an entry for the package defined in this CAP file
- Example:

```
.ImportComponent = {
   count:  1
   package_info = {
      minor_version:  0
      major_version:  1
      AID_length:  7
      AID:  a0.00.00.00.62.01.01
   }
}
```

Constant Pool Component

- It contains an entry for each of the classes, methods, and fields referenced by elements in the Method Component of this CAP file.
- An external reference to a class is composed of a package token and a class token.
- Together those tokens specify a certain class in a certain package.
- Example: a *Class reference* is composed of *package token plus a Class token*

```
.ConstantPool = {
   /* 0000, 0 */CONSTANT_InstanceFieldRef:  field 2 of class 0x0000
   /* 0020, 8 */CONSTANT_StaticMethodRef:  external: 0x80,0x3,0x0
   /* 0044, 17*/CONSTANT_VirtualMethodRef:  method 8 of class 0x0000
```

```
CONSTANT_StaticMethodref_info {
     u1 tag
     union {
               {
                    u1 padding
                    u2 offset
               } internal_ref
               {
                    u1 package_token
                    u1 class_token
                    u1 token
               } external_ref
     } static_method_ref
}
```

Constant Type	Tag
CONSTANT_Classref	1
CONSTANT_InstanceFieldref	2
CONSTANT_VirtualMethodref	3
CONSTANT_SuperMethodref	4
CONSTANT_StaticFieldref	5
CONSTANT_StaticMethodref	6

Class Component

- It describes each Class/Interface, instance size, methods and package used, ...

```
.class { // @0000
   interface_count: 0
   super_class_ref: external class 0x3 of package 0x80
   declared_instance_size: 8 ; first_reference_index: 0 ;reference_count: 8
   public_method_table_base: 4 ; public_method_table_count: 7
   public_methods = {
       public method @00fd
       public method inherited
   …
   }
   package_method_table_base: 0 ; package_method_table_count: 0
   package_methods = {}
   interfaces = {}
}
```

Method Component

- It describes each of the methods declared in this package, excluding `<clinit>` methods and interface method declarations. Abstract methods defined by classes (not interfaces) are included.
- The exception handlers associated with each method are also described.
- Ressources used, ***byte code***
- The core of the malicious applet modification.

Method Component

```
.method {
  method_info[0] // @0001= {
    // flags:   0
    // max_stack:  5
    // nargs:  4
    // max_locals: 7
    /*0003*/ L0: aload_0
    /*0004*/ invokespecial 8 //  Which is…? How many bytes?
    /*0007*/ aload_0
    /*0008*/ bspush 100
    /*000a*/ newarray byte
    /*000c*/ putfield_a 0
    /*000e*/ aload_0
    …
```

Static Field Component

- It contains all of the information required to create and initialize an image of all of the static fields defined in this package, referred to as the static field image.
- It includes all information required to initialize classes. In the Java virtual machine a class is initialized by executing its `<clinit>` method.
- In the Java Card virtual machine the functionality of `<clinit>` methods is represented in the Static Field Component as array initialization data and non-default values of primitive types data.

Static Field Component

```
.StaticFieldComponent = {
   image_size:  7
   reference_count:  3
   array_init_count:  3
   array_init = {
      type:  byte
      count:  70
      values = {
         bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
         bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
         bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
         bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb
         bb bb bb bb bb bb
      }
…
```

Reference Location Component

- One of the components used to create malicious applet.
- Link between Constant Pool Component and Method Component
- It represents lists of offsets into the `info` item of the Method Component to items that contain indices into the `constant_pool[]` array of the Constant Pool Component
- Some of the constant pool indices are represented in one-byte values while others are represented in two-byte values.
- Should/can be considered as a linker accelerator…

Reference Location Component

```
.ReferenceLocationComponent = {
   offsets_to_byte_indices = {
      @000d @0014 @001b @0022 @0029 @0030 @0037 @003e
      @006b @0077 @0081 @0088 @008f @0096 @00a2 @00ac
      @00b8 @00c2 @00ce @00d8 @00e3 @0175 @01a0 @01b5
      @01c9 @01dd @0206 @021a
   }
   offsets_to_byte2_indices = {
      @0005 @00f4 @00fa @011f @0125 @012c @0131 @0141
      @0151 @0177 @017a @0181 @018a @018d @0190 @0197
      @01a2 @01a5 @01ac @01b7 @01ba @01c1 @01cb @01ce
      @01d5 @01df @01e2 @01e9 @01f1 @01f4 @01f7 @01fe
      @0208 @020b @0212 @021c @021f @0226 @022e @0236
   }
}
```

Other components

- Export Component:
  - List of static elements in this packages usable by other Classes in other packages.
- Descriptor Component (*optional*):
  - Represents each Class and Interface with class token, access condition, fields (token, access condition, type) and methods (token, access condition, bytecode length, exceptions, ...)
  - Provides sufficient information to parse and verify all elements of the CAP file. It references, and therefore describes, elements in the Constant Pool Component, Class Component, Method Component, and Static Field Component. No components in the CAP file reference the Descriptor Component.

Other components

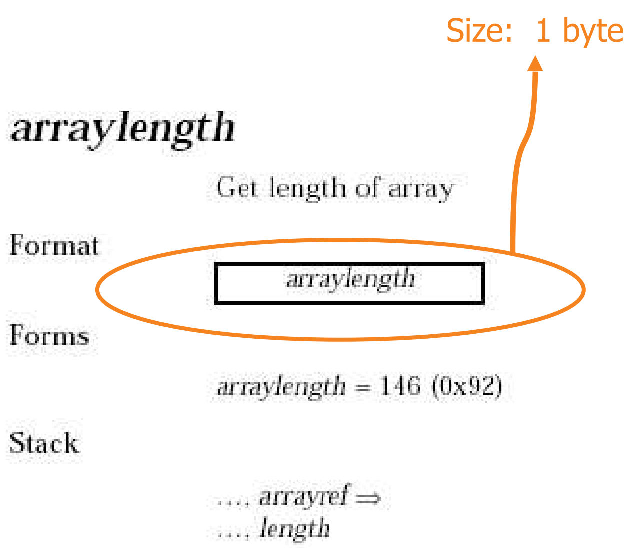
- Custom Component
  - Java Card CAP files are permitted to contain new, or custom, components.
  - Java Card virtual machines must be able to accept CAP files that do not contain new components
  - Implementations are required to silently ignore components they do not recognize.

Structure of the Virtual Machine

- *Bytecode* will describe the method behaviour, based on set of instructions,
- Individual instruction consists of *one-byte opcode* and 0 to more *operand*.

Bytecode example

- Opcode and Operand
- Example 1



Bytecode example

- Opcode and Operand
- Example2:

Size: 4 bytes

checkcast

Check whether object is of given type

Format

checkcast
atype
indexbyte1
indexbyte2

Forms

checkcast = 148 (0x94)

Stack

..., *objectref* –>
..., *objectref*

Liste des bytecodes

dec	hex	mnemonic	dec	hex	mnemonic
0	00	nop	47	2F	sstore_0
1	01	aconst_null	48	30	sstore_1
2	02	sconst_m1	49	31	sstore_2
3	03	sconst_0	50	32	sstore_3
4	04	sconst_1	51	33	istore_0
5	05	sconst_2	52	34	istore_1
6	06	sconst_3	53	35	istore_2
7	07	sconst_4	54	36	istore_3
8	08	sconst_5	55	37	aastore
9	09	iconst_m1	56	38	bastore
10	0A	iconst_0	57	39	sastore
11	0B	iconst_1	58	3A	iastore
12	0C	iconst_2	59	3B	pop
13	0D	iconst_3	60	3C	pop2
14	0E	iconst_4	61	3D	dup
15	0F	iconst_5	62	3E	dup2
16	10	bspush	63	3F	dup_x
17	11	sspush	64	40	swap_x
18	12	bipush	65	41	sadd
19	13	sipush	66	42	iadd

dec	hex	mnemonic	dec	hex	mnemonic
20	14	iipush	67	43	ssub
21	15	aload	68	44	isub
22	16	sload	69	45	smul
23	17	iload	70	46	imul
24	18	aload_0	71	47	sdiv
25	19	aload_1	72	48	idiv
26	1A	aload_2	73	49	srem
27	1B	aload_3	74	4A	irem
28	1C	sload_0	75	4B	sneg
29	1D	sload_1	76	4C	ineg
30	1E	sload_2	77	4D	sshl
31	1F	sload_3	78	4E	ishl
32	20	iload_0	79	4F	sshr
33	21	iload_1	80	50	ishr
34	22	iload_2	81	51	sushr
35	23	iload_3	82	52	iushr
36	24	aaload	83	53	sand
37	25	baload	84	54	iand
38	26	saload	85	55	sor
39	27	iaload	86	56	ior
40	28	astore	87	57	sxor
41	29	sstore	88	58	ixor
42	2A	istore	89	59	sinc
43	2B	astore_0	90	5A	iinc
44	2C	astore_1	91	5B	s2b
45	2D	astore_2	92	5C	s2i
46	2E	astore_3	93	5D	i2b

Liste des bytecodes

dec	hex	mnemonic	dec	hex	mnemonic
94	5E	i2s	141	8D	invokestatic
95	5F	icmp	142	8E	invokeinterface
96	60	ifeq	143	8F	new
97	61	ifne	144	90	newarray
98	62	iflt	145	91	anewarray
99	63	ifge	146	92	arraylength
100	64	ifgt	147	93	athrow
101	65	ifle	148	94	checkcast
102	66	ifnull	149	95	instanceof
103	67	ifnonnull	150	96	sinc_w
104	68	if_acmpeq	151	97	iinc_w
105	69	if_acmpne	152	98	ifeq_w
106	6A	if_scmpeq	153	99	ifne_w
107	6B	if_scmpne	154	9A	iflt_w
108	6C	if_scmplt	155	9B	ifge_w
109	6D	if_scmpge	156	9C	ifgt_w
110	6E	if_scmpgt	157	9D	ifle_w
111	6F	if_scmple	158	9E	ifnull_w
112	70	goto	159	9F	ifnonnull_w
113	71	jsr	160	A0	if_acmpeq_w
114	72	ret	161	A1	if_acmpne_w
115	73	stableswitch	162	A2	if_scmpeq_w
116	74	itableswitch	163	A3	if_scmpne_w
117	75	slookupswitch	164	A4	if_scmplt_w
118	76	ilookupswitch	165	A5	if_scmpge_w
119	77	areturn	166	A6	if_scmpgt_w
120	78	sreturn	167	A7	if_scmple_w
121	79	ireturn	168	A8	goto_w
122	7A	return	169	A9	getfield_a_w
123	7B	getstatic_a	170	AA	getfield_b_w
124	7C	getstatic_b	171	AB	getfield_s_w

dec	hex	mnemonic	dec	hex	mnemonic
125	7D	getstatic_s	172	AC	getfield_i_w
126	7E	getstatic_i	173	AD	getfield_a_this
127	7F	putstatic_a	174	AE	getfield_b_this
128	80	putstatic_b	175	AF	getfield_s_this
129	81	putstatic_s	176	B0	getfield_i_this
130	82	putstatic_i	177	B1	putfield_a_w
131	83	getfield_a	178	B2	putfield_b_w
132	84	getfield_b	179	B3	putfield_s_w
133	85	getfield_s	180	B4	putfield_i_w
134	86	getfield_i	181	B5	putfield_a_this
135	87	putfield_a	182	B6	putfield_b_this
136	88	putfield_b	183	B7	putfield_s_this
137	89	putfield_s	184	B8	putfield_i_this
138	8A	putfield_i			…
139	8B	invokevirtual	254	FE	impdep1
140	8C	invokespecial	255	FF	impdep2

A vérifier dans les implémentations

opcode	byte	short	int	reference
Tspush	bspush	sspush		
Tipush	bipush	sipush	iipush	
Tconst		sconst	iconst	aconst
Tload		sload	iload	aload
Tstore		sstore	istore	astore
Tinc		sinc	iinc	
Taload	baload	saload	iaload	aaload
Tastore	bastore	sastore	iastore	aastore
Tadd		sadd	iadd	
Tsub		ssub	isub	
Tmul		smul	imul	
Tdiv		sdiv	idiv	
Trem		srem	irem	
Tneg		sneg	ineg	
Tshl		sshl	ishl	
Tshr		sshr	ishr	
Tushr		sushr	iushr	
Tand		sand	iand	
Tor		sor	ior	

opcode	byte	short	int	reference
Txor		sxor	ixor	
s2T	s2b		s2i	
i2T	i2b	i2s		
Tcmp			icmp	
if_TcmpOP		if_scmpOP		if_acmpOP
Tlookupswitch		slookupswitch	ilookupswitch	
Ttableswitch		stableswitch	itableswitch	
Treturn		sreturn	ireturn	areturn
getstatic_T	getstatic_b	getstatic_s	getstatic_i	getstatic_a
putstatic_T	putstatic_b	putstatic_s	putstatic_i	putstatic_a
getfield_T	getfield_b	getfield_s	getfield_i	getfield_a
putfield_T	putfield_b	putfield_s	putfield_i	putfield_a

Java (Storage) Type	Size in Bits	Computational Type
byte	8	short
short	16	short
int	32	int

Manipulating CAP file

- It is the basis for malicious applet,
- Need to maintain the coherence between all the component of the CAP file,
- Prototype available: CapMap
  - A library to manipulate the CAP (a kind of Byte Code Engineering Library - BCEL)
  - Two versions:
    - A reader will be publicly available,
    - A modifier/compiler restricted distribution.
  - A GUI to use it for educational purpose.
- There exists a more powerful tool to make easy consistent modifications: JCatools *(but it is not yet public)*