Diagram 4.4-E. Constant pool relations

New items abbreviated Anchor, Linkage are fully spelled
CONSTANT_SpecializationAnchor, CONSTANT_SpecializationLinkage

- Anchor*, Dynamic, InvokeDynamic
- Anchor* reference
- Linkage* reference
- "API Point Name"*
- "API Point Name"* new concept
- "API Point Reference"*
- "API Point Reference"* new concept
- Fieldref, XMethodref
- NameAndType
- Utf8
- "Loadable Constant"
- "Loadable Constant" existing concept
- (see Diagram 4.4-G)
- MethodHandle
- String
- MethodType
- Module, Package
- Class
- name
- descriptor
- name_and_type
- class
- parameter
- bootstrap_arguments
- bootstrap method
- reference
- name
- descriptor
- name
- Utf8
- bootstrap_arguments
- name
- descriptor
- name
- Utf8
- bootstrap_arguments
- name
- descriptor
- name
- Utf8
Diagram 4.4-F(a). API Point relations
Note: A Linkage constant is usable anywhere its reference item may be used.

Diagram showing relationships between ClassFile structure, MethodHandle, "API Point Reference", "API Point Name", Fieldref, XMethodref, Class, "Loadable Constant", and Linkage.*

Key concepts include:
- invokeX, getX, putX, withfield
- ldc, new, defaultvalue, Xnewarray, instanceof, checkcast
- operand (field or method)
- operand (class or interface)
- class
- recursive
- recursive, to class only

(see Diagram 4.4-G)
Diagram 4.4-F(b). “API Point References”, all configurations
(These are also “API Point Names”, except Linkage constants.)
Note: A Linkage constant is only loadable if its reference is loadable, i.e., a Class. Other Linkage constants (e.g., of a Fieldref) are not loadable.
Diagram 4.4-H(a). Example constant pool:
non-parametric client of List<Point>
(resolution states are at right; all are invariant)
public interface List<T>
{
    ...get...
}

ClassFile
this = j/u/List
interface = Collection[R.T]
Parametric = Anchor R

(self species reference, if any)

method_info
name = subList
type = (int,int)List
TypeRestriction = {List[R.T]}
Parametric = Anchor R
Code = none (ACC_ABSTRACT)

method_info
name = get
type = (int)Object
TypeRestriction = {R.T}
Parametric = Anchor R
Code = none (ACC_ABSTRACT)
Diagram 4.4-H(c). Example parametric implementation
class ArrayList<T> implements List<T> { ...get... }

ClassFile
this = j/u/ArrayList
interface = j/u/List[R.T]
Parametric = Anchor R
(self species reference, if any)

method_info
name = subList
type = (int,int)List
TypeRestriction = {List[R.T]}
Parametric = Anchor R
Code = stuff using elements/A

method_info
name = get
type = (int)Object
TypeRestriction = {R.T}
Parametric = Anchor R
Code = stuff using elements/A

field_info
name = elements
type = Object[]
TypeRestriction = {R.T[]}
Parametric = Anchor R
Diagram 4.4-H(d). Example parametric subclass

```java
class MyVector<T> extends ju.Vector<T> { ...get... }
class Vector<T> { ... protected T[] elementData; ... }
```

```
Utf8 j/u/Vector
Class Vector
Utf8 MyVector
Linkage
MyVector[T]
Class MyVector
Utf8 MyVector
Utf8 j/u/Vector
Linkage
Vector[R.T]
Class Vector
Utf8 MyVector
ConDy R.T
Anchor R BSM=...
Fieldref V[T].elm-Data
NameAndType elmData:Obj[]

ClassFile
this = MyVector
super = j/u/Vector[R.T]
Parametric = Anchor R

(method_info
name = get
type = (int)Object
TypeRestriction = {R.T}
Parametric = Anchor R
Code = (invariants, parametrics)

(bytecodes...)

getfield
name = elementData
type = Object[]
class = Vector[R.T]
possible type restriction T[]
iload_1 aaload ...
```
Diagram 4.7-D(a). Parametric attribute relations
Diagram 4.7-D(b). TypeRestriction attribute relations

- field_info
- method_info

TypeRestriction

*new attribute

restriction(s)

Loadable Constant
(recursive)

(see Diagram 4.4-G)
Diagram 4.4-H(d). Example parametric subclass

```java
class MyVector<T> extends ju.Vector<T> { ...get... }

class Vector<T> { ... protected T[] elementData; ... }
```

```
Utf8 j/u/Vector

Class File
this = MyVector
super = j/u/Vector[R.T]
Parametric = Anchor R

(self species reference, if any)

Utf8 MyVector

Class MyVector

Linkage MyVector[T]

Utf8 MyVector

Class MyVector

Linkage Vector[R.T]

Utf8 j/u/Vector

Class Vector

Utf8 j/u/Vector

ConDy R.T

Anchor R BSM=...

Fieldref V[T].el-Data

NameAndType elmData:Obj[]

method_info
name = get
type = (int)Object
TypeRestriction = {R.T}
Parametric = Anchor R
Code = (invariants, parametrics)

(getfield
name = elementData
type = Object[]
class = Vector[R.T]
possible type restriction ⦿ T[]

(seealso...)

iload_1 aaload ...
```
Graph of existing and proposed relations among constant pool structures, to support the “Parametric VM”.

Legend:

• A rectangular box shows one or more constant types. For example, a box labeled `class` represents a `CONSTANT_Class_info` structure.

• An arrow shows how one entity depends directly on another. For example, `String` depends directly on the `Utf8` which specifies its characters.

• Arrows from boxes are labeled to show which item in a given constant pool structure provides the index for the relation indicated by the arrow.

• A circle shows, not a single constant type, but a conceptual group of types, which are (for some uses) interchangeable. The conceptual groups are `Loadable Constant`, `API Point Reference`, and `API Point Name`.

• Box-headed arrows into a group circle show users of all the types in the group, while arrows out of the circle indicate the group’s types. Small dotted arrows show some routes through a group.

• A starred footnote of `new` indicates a proposed new constant type or conceptual group of types. The new types are `SpecializationAnchor` and `SpecializationLinkage`, or `Anchor` and `Linkage` for short. The new conceptual groups are `API Point Reference` and (a subset) `API Point Name`.

• An arrow with a blank head redirects to a different diagram.

• Recursion points are white, and stand in place of the corresponding colored concept or constant type.

• A lozenge shaped box shows one or more instruction types.

• A box with rounded corners shows some other structure, such as the new `Parametric` and `TypeRestriction` attributes.