Introduction: Multiview Programming

Andrew P. Black & Mark P. Jones
Portland State University
Portland, Oregon, USA

Portland State UNIVERSITY

Abstraction Rules, OK?

- As programmers, we know the value of abstraction

Abstraction Rules, OK?

Abstraction Rules, OK?

Representing a date in a computer

Abstraction Rules, OK?

Representing a date in a computer

Tuesday, 26th May 2004

Abstraction Rules, OK?

Representing a date in a computer

Tuesday, 26th May 2004

05.26.04

Abstraction Rules, OK?

Representing a date in a computer

Tuesday, 26th May 2004

05.26.04

May 26, 2004

Abstraction Rules, OK?

Representing a date in a computer

Tuesday, 26th May 2004

05.26.04

May 26, 2004

2004-05-26

Abstraction Rules, OK?

Representing a date in a computer

Tuesday, 26th May 2004

05.26.04

May 26, 2004

2004-05-26

Dienstag, 26 Mai 2004

Abstraction Rules, OK?

Representing a date in a computer

- Tuesday, 26th May 2004
- 05.26.04
- May 26, 2004
- 2004-05-26
- Dienstag, 26 Mai 2004

Abstraction Rules, OK?

Representing a date in a computer

Abstraction Rules, OK?

Representing a date in a computer

732087

Abstraction Rules, OK?

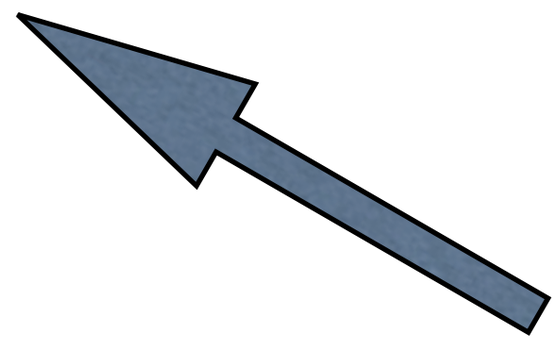
Presenting a date to a human

732087

Abstraction Rules, OK?

Presenting a date to a human

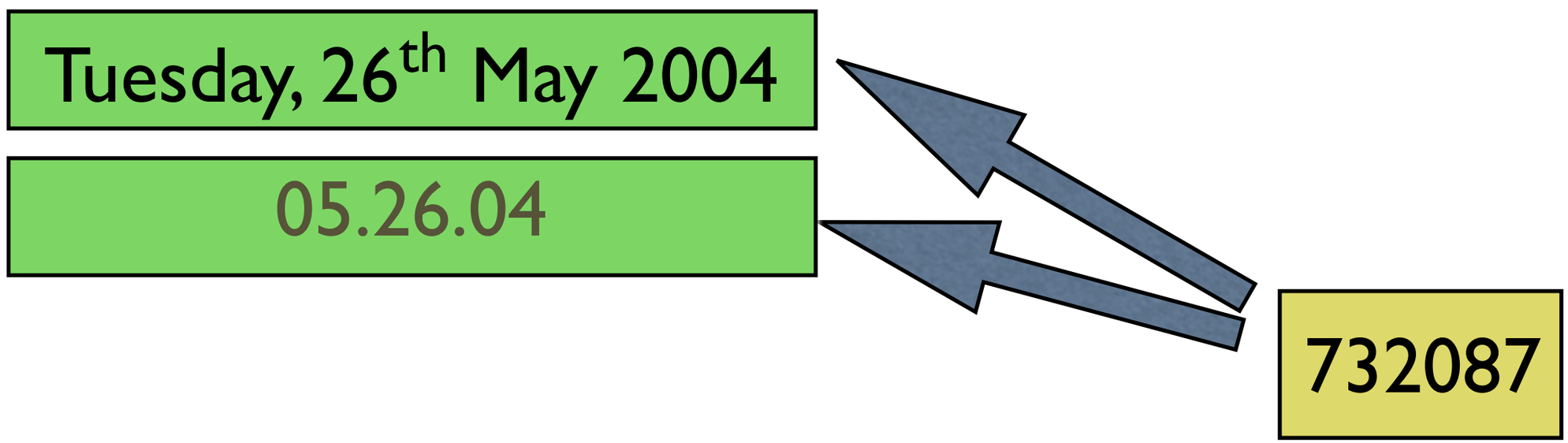
Tuesday, 26th May 2004



732087

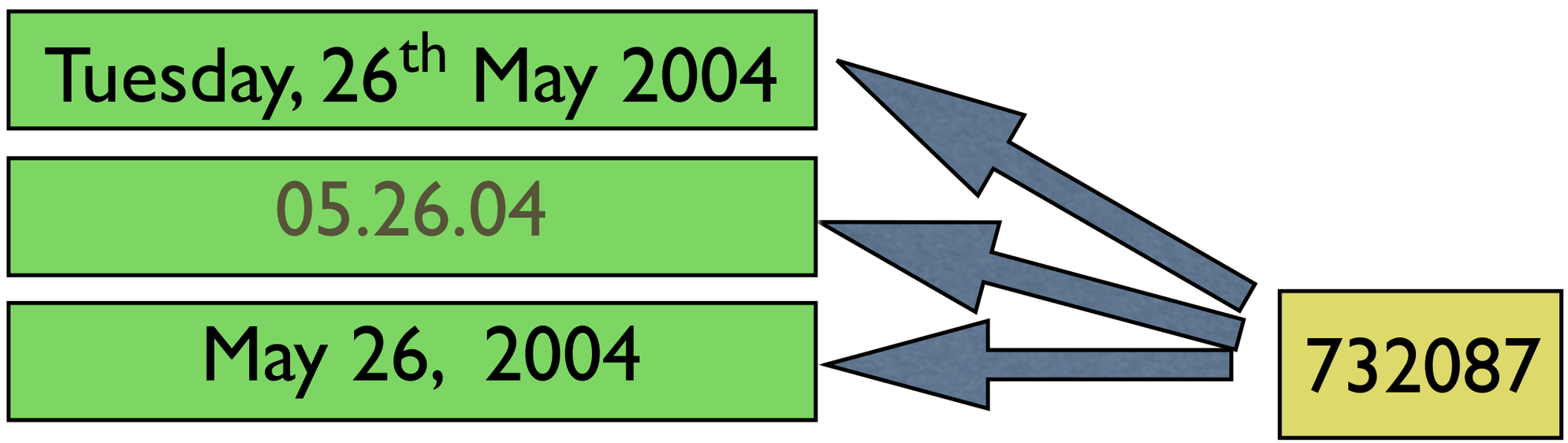
Abstraction Rules, OK?

Presenting a date to a human



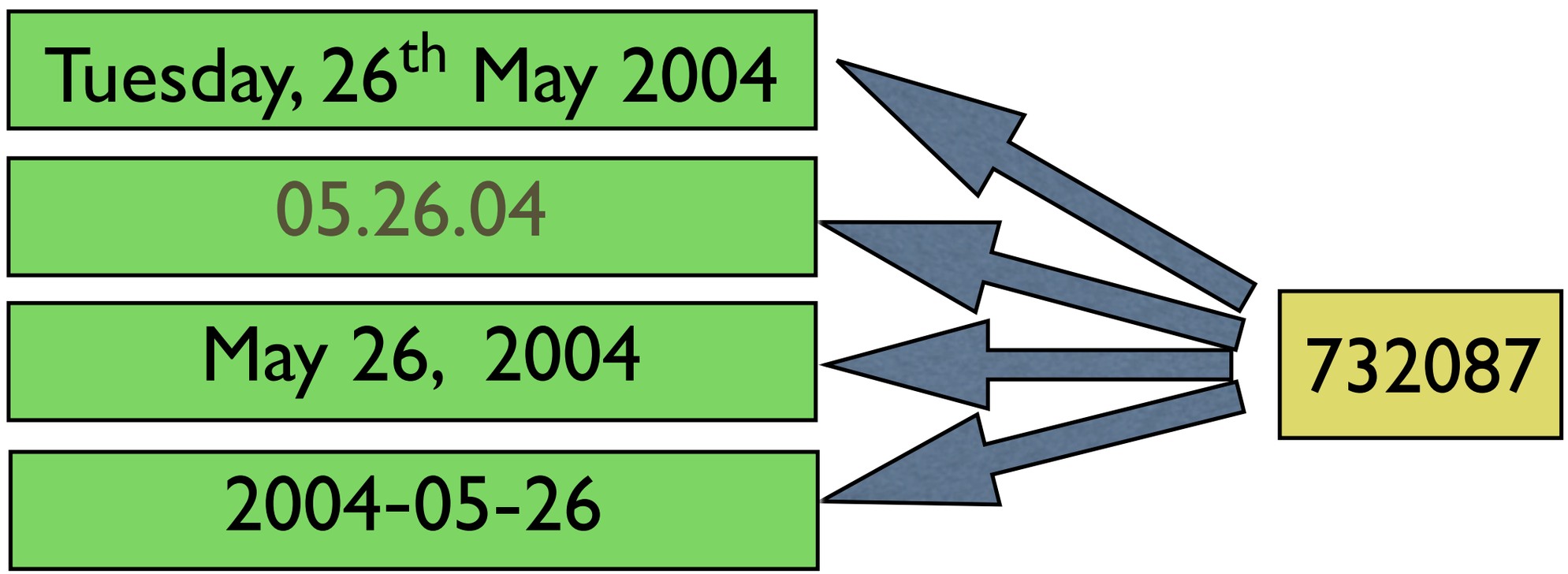
Abstraction Rules, OK?

Presenting a date to a human



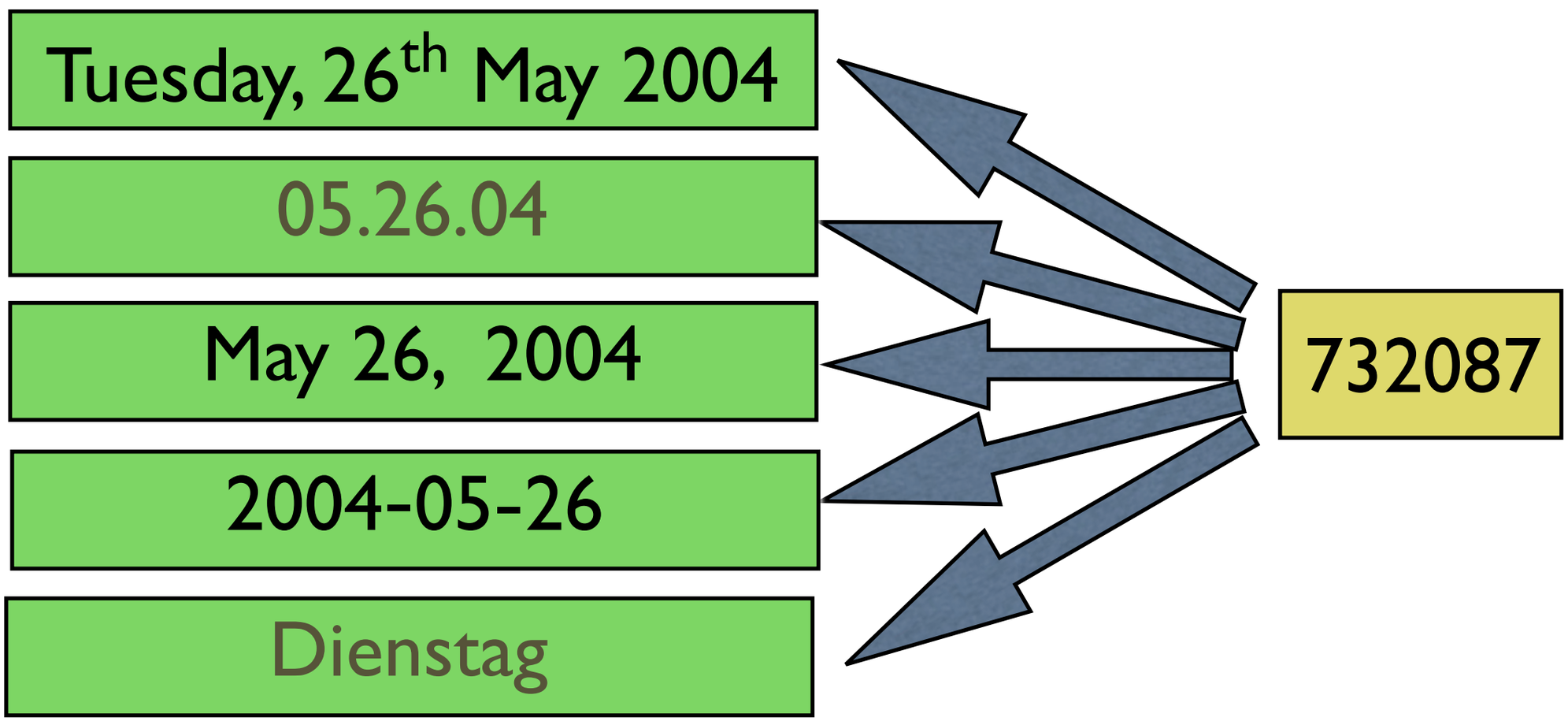
Abstraction Rules, OK?

Presenting a date to a human



Abstraction Rules, OK?

Presenting a date to a human



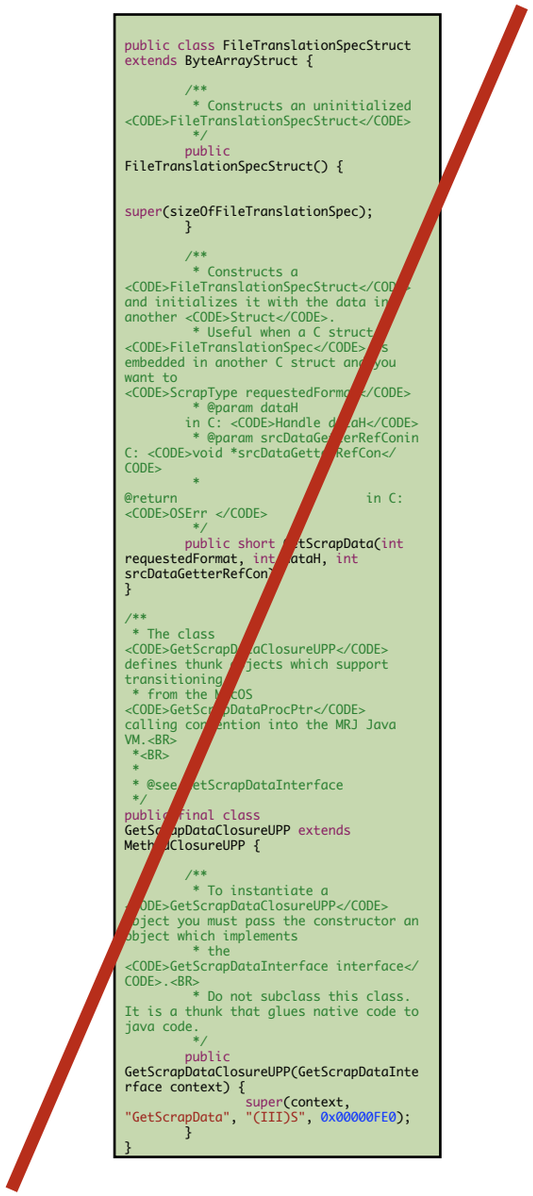
Why don't we Represent *Programs* Abstractly?

```
public class FileTranslationSpecStruct
extends ByteArrayStruct {

        /**
         * Constructs an uninitialized
<CODE>FileTranslationSpecStruct</CODE>
         */
        public
FileTranslationSpecStruct() {


super(sizeOfFileTranslationSpec);
        }

        /**
         * Constructs a
<CODE>FileTranslationSpecStruct</CODE>
and initializes it with the data in
another <CODE>Struct</CODE>.
         * Useful when a C struct
<CODE>FileTranslationSpec</CODE> is
embedded in another C struct and you
want to
<CODE>ScrapType requestedFormat</CODE>
         * @param dataH
        in C: <CODE>Handle dataH</CODE>
         * @param srcDataGetterRefConin
C: <CODE>void *srcDataGetterRefCon</
CODE>
         *
@return                         in C:
<CODE>OSErr </CODE>
         */
        public short GetScrapData(int
requestedFormat, int dataH, int
srcDataGetterRefCon);
}

/**
 * The class
<CODE>GetScrapDataClosureUPP</CODE>
defines thunk objects which support
transitioning
 * from the MacOS
<CODE>GetScrapDataProcPtr</CODE>
calling convention into the MRJ Java
VM.<BR>
 *<BR>
 *
 * @see GetScrapDataInterface
 */
public final class
GetScrapDataClosureUPP extends
MethodClosureUPP {

        /**
         * To instantiate a
<CODE>GetScrapDataClosureUPP</CODE>
object you must pass the constructor an
object which implements
         * the
<CODE>GetScrapDataInterface interface</
CODE>.<BR>
         * Do not subclass this class.
It is a thunk that glues native code to
java code.
         */
        public
GetScrapDataClosureUPP(GetScrapDataInte
rface context) {
                super(context,
"GetScrapData", "(III)S", 0x00000FE0);
        }
}
```

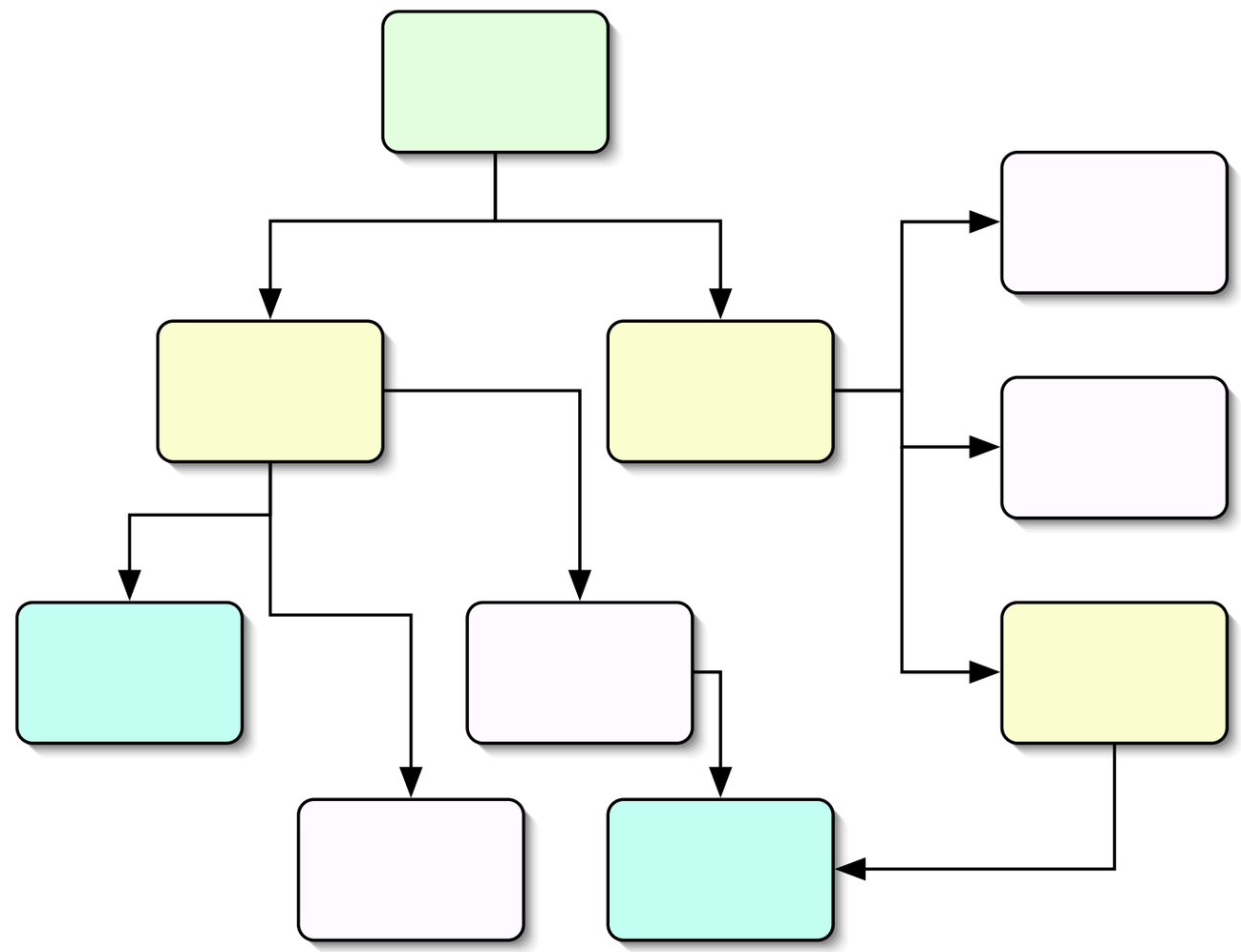
Why don't we Represent *Programs* Abstractly?

```
public class FileTranslationSpecStruct
extends ByteArrayStruct {

        /**
         * Constructs an uninitialized
<CODE>FileTranslationSpecStruct</CODE>
         */
        public
FileTranslationSpecStruct() {


super(sizeOfFileTranslationSpec);
        }

        /**
         * Constructs a
<CODE>FileTranslationSpecStruct</CODE>
and initializes it with the data in
another <CODE>Struct</CODE>.
         * Useful when a C struct
<CODE>FileTranslationSpec</CODE> is
embedded in another C struct and you
want to
<CODE>ScrapType requestedFormat</CODE>
         * @param dataH
        in C: <CODE>Handle dataH</CODE>
         * @param srcDataGetterRefConin
C: <CODE>void *srcDataGetterRefCon</
CODE>
         *
@return                         in C:
<CODE>OSErr </CODE>
         */
        public short GetScrapData(int
requestedFormat, int dataH, int
srcDataGetterRefCon)
}

/**
 * The class
<CODE>GetScrapDataClosureUPP</CODE>
defines thunk objects which support
transitioning
 * from the MacOS
<CODE>GetScrapDataProcPtr</CODE>
calling convention into the MRJ Java
VM.<BR>
 *<BR>
 *
 * @see GetScrapDataInterface
 */
public final class
GetScrapDataClosureUPP extends
MethodClosureUPP {

        /**
         * To instantiate a
<CODE>GetScrapDataClosureUPP</CODE>
object you must pass the constructor an
object which implements
         * the
<CODE>GetScrapDataInterface interface</
CODE>.<BR>
         * Do not subclass this class.
It is a thunk that glues native code to
java code.
         */
        public
GetScrapDataClosureUPP(GetScrapDataInte
rface context) {
                super(context,
"GetScrapData", "(III)S", 0x00000FE0);
        }
}
```

Why don't we Represent *Programs* Abstractly?

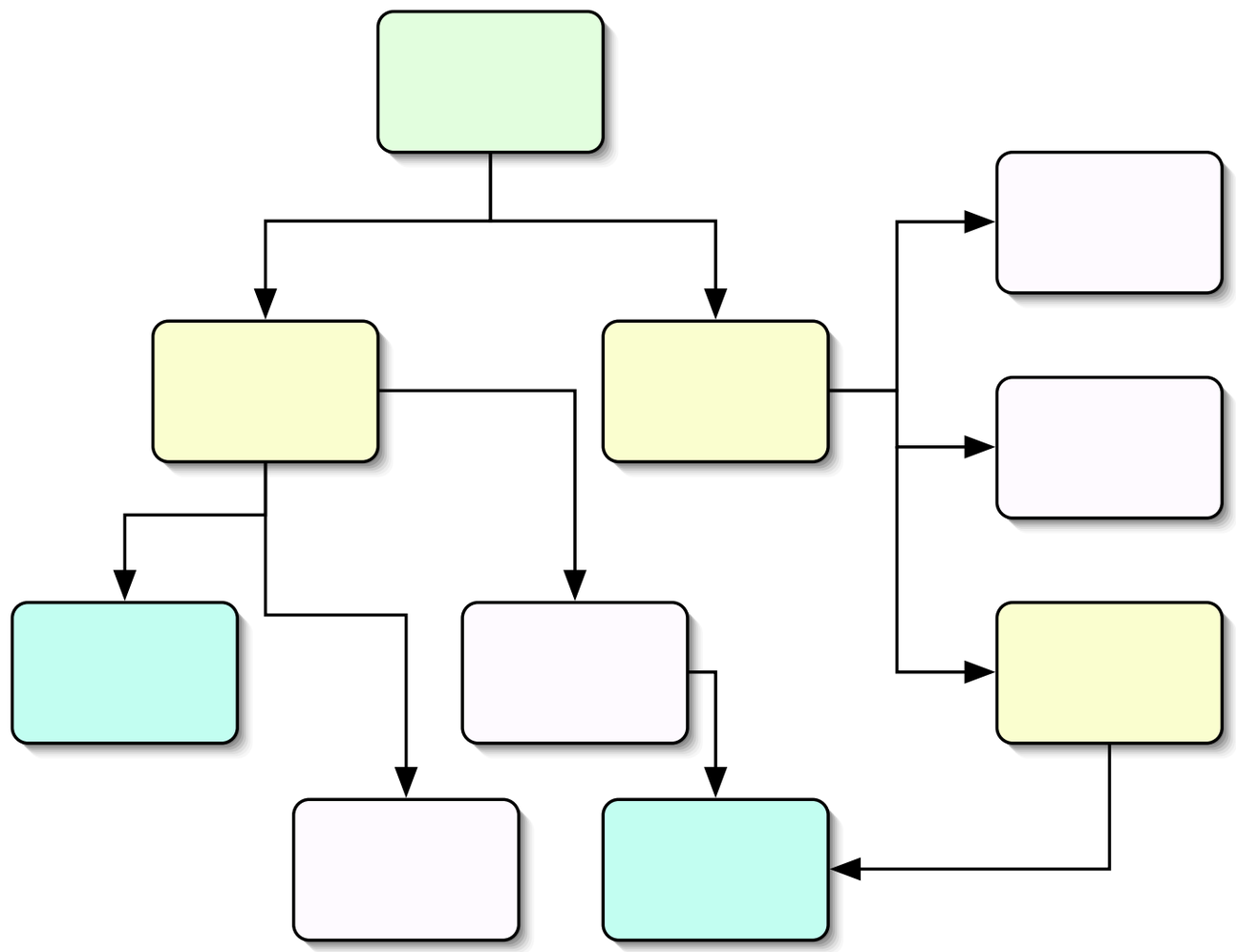


```
public class FileTranslationSpecStruct
extends ByteArrayStruct {

        /**
         * Constructs an uninitialized
<CODE>FileTranslationSpecStruct</CODE>
         */
        public
FileTranslationSpecStruct() {


super(sizeOfFileTranslationSpec);
        }

        /**
         * Constructs a
<CODE>FileTranslationSpecStruct</CODE>
and initializes it with the data in
another <CODE>Struct</CODE>.
         * Useful when a C struct
<CODE>FileTranslationSpec</CODE> is
embedded in another C struct and you
want to
<CODE>ScrapType requestedFormat</CODE>
         * @param dataH
        in C: <CODE>Handle dataH</CODE>
         * @param srcDataGetterRefConin
C: <CODE>void *srcDataGetterRefCon</
CODE>
         *
@return                         in C:
<CODE>OSErr </CODE>
         */
        public short GetScrapData(int
requestedFormat, int dataH, int
srcDataGetterRefCon) {
}

/**
 * The class
<CODE>GetScrapDataClosureUPP</CODE>
defines thunk objects which support
transitioning
 * from the MacOS
<CODE>GetScrapDataProcPtr</CODE>
calling convention into the MRJ Java
VM.<BR>
 *<BR>
 *
 * @see GetScrapDataInterface
 */
public final class
GetScrapDataClosureUPP extends
MethodClosureUPP {

        /**
         * To instantiate a
<CODE>GetScrapDataClosureUPP</CODE>
object you must pass the constructor an
object which implements
         * the
<CODE>GetScrapDataInterface interface</
CODE>.<BR>
         * Do not subclass this class.
It is a thunk that glues native code to
java code.
         */
        public
GetScrapDataClosureUPP(GetScrapDataInte
rface context) {
                super(context,
"GetScrapData", "(III)S", 0x00000FE0);
        }
}
```

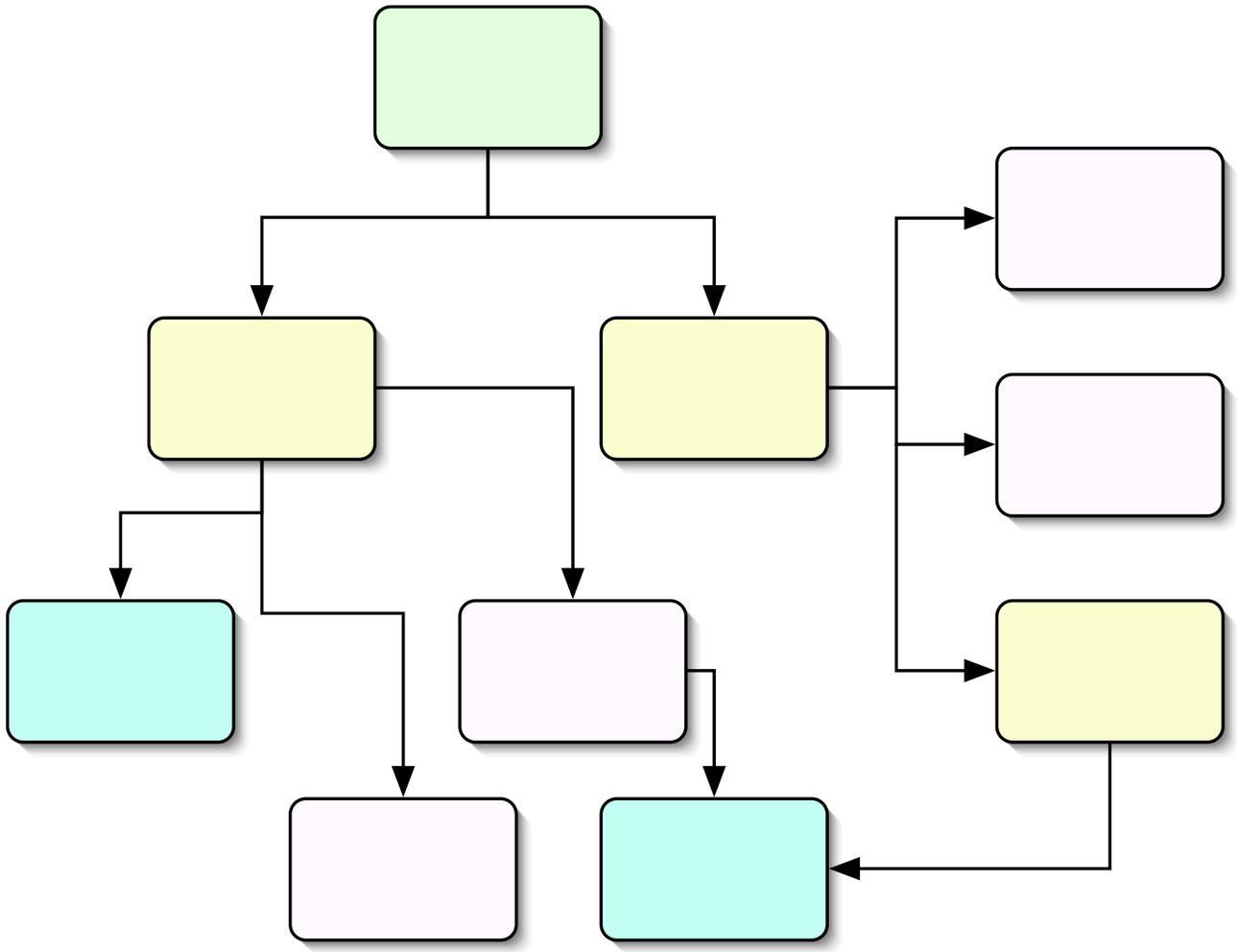


Why don't we Represent *Programs* Abstractly?

```
public class FileTranslationSpecStruct
extends ByteArrayStruct {

        /**
         * Constructs an uninitialized
<CODE>FileTranslationSpecStruct</CODE>
         */
        public
FileTranslationSpecStruct() {


super(sizeOfFileTranslationSpec);
        }

        /**
         * Constructs a
<CODE>FileTranslationSpecStruct</CODE>
and initializes it with the data in
another <CODE>Struct</CODE>.
         * Useful when a C struct
<CODE>FileTranslationSpec</CODE> is
embedded in another C struct and you
want to
<CODE>ScrapType requestedFormat</CODE>
         * @param dataH
        in C: <CODE>Handle dataH</CODE>
         * @param srcDataGetterRefConin
C: <CODE>void *srcDataGetterRefCon</
CODE>
         *
@return                         in C:
<CODE>OSErr </CODE>
         */
        public short GetScrapData(int
requestedFormat, int dataH, int
srcDataGetterRefCon);
}

/**
 * The class
<CODE>GetScrapDataClosureUPP</CODE>
defines thunk objects which support
transitioning
 * from the MacOS
<CODE>GetScrapDataProcPtr</CODE>
calling convention into the MRJ Java
VM.<BR>
 *<BR>
 *
 * @see GetScrapDataInterface
 */
public final class
GetScrapDataClosureUPP extends
MethodClosureUPP {

        /**
         * To instantiate a
<CODE>GetScrapDataClosureUPP</CODE>
object you must pass the constructor an
object which implements
         * the
<CODE>GetScrapDataInterface interface</
CODE>.<BR>
         * Do not subclass this class.
It is a thunk that glues native code to
java code.
         */
        public
GetScrapDataClosureUPP(GetScrapDataInte
rface context) {
                super(context,
"GetScrapData", "(III)S", 0x00000FE0);
        }
}
```

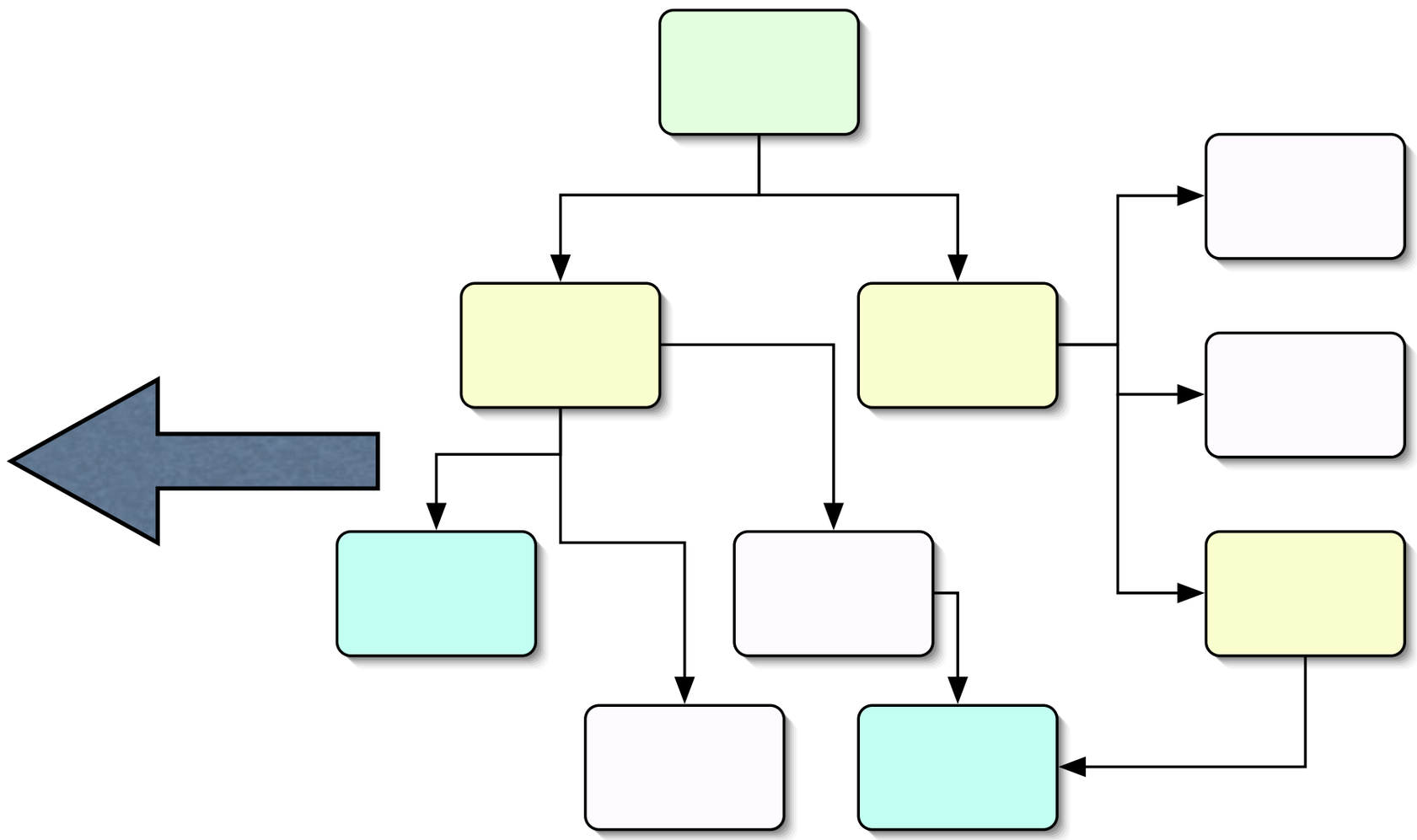


Why don't we Represent *Programs* Abstractly?



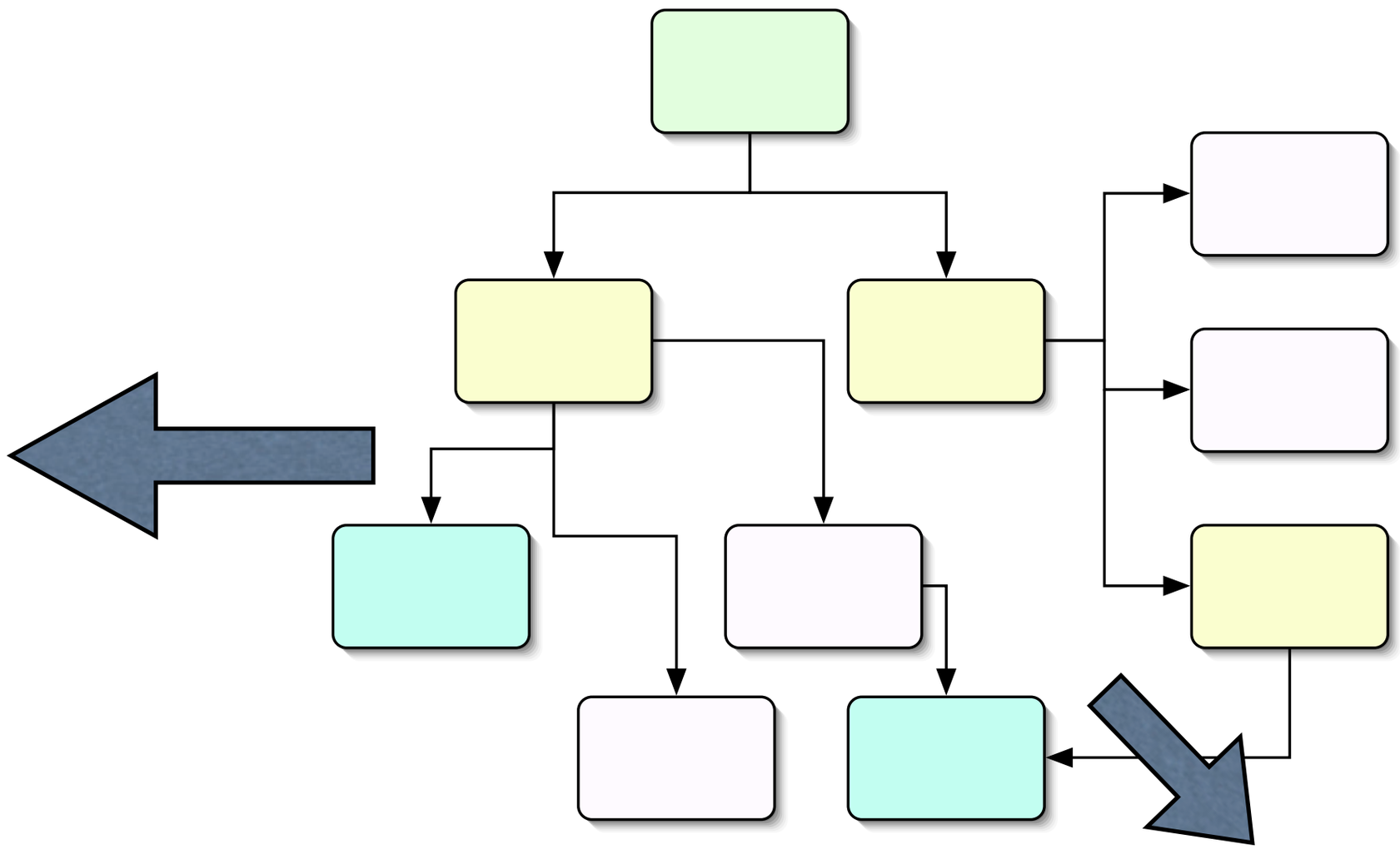
Why don't we Represent *Programs* Abstractly?

```
public class FileTranslationSpecStruct
extends ByteArrayStruct {

        /**
         * Constructs an uninitialized
<CODE>FileTranslationSpecStruct</CODE>
         */
        public
FileTranslationSpecStruct() {


super(sizeOfFileTranslationSpec);
        }

        /**
         * Constructs a
<CODE>FileTranslationSpecStruct</CODE>
and initializes it with the data in
another <CODE>Struct</CODE>.
         * Useful when a C struct
<CODE>FileTranslationSpec</CODE> is
embedded in another C struct and you
want to
<CODE>ScrapType requestedFormat</CODE>
         * @param dataH
        in C: <CODE>Handle dataH</CODE>
         * @param srcDataGetterRefConin
C: <CODE>void *srcDataGetterRefCon</
CODE>
         *
@return                         in C:
<CODE>OSErr </CODE>
         */
        public short GetScrapData(int
requestedFormat, int dataH, int
srcDataGetterRefCon);
}

/**
 * The class
<CODE>GetScrapDataClosureUPP</CODE>
defines thunk objects which support
transitioning
 * from the MacOS
<CODE>GetScrapDataProcPtr</CODE>
calling convention into the MRJ Java
VM.<BR>
 *<BR>
 *
 * @see GetScrapDataInterface
 */
public final class
GetScrapDataClosureUPP extends
MethodClosureUPP {

        /**
         * To instantiate a
<CODE>GetScrapDataClosureUPP</CODE>
object you must pass the constructor an
object which implements
         * the
<CODE>GetScrapDataInterface interface</
CODE>.<BR>
         * Do not subclass this class.
It is a thunk that glues native code to
java code.
         */
        public
GetScrapDataClosureUPP(GetScrapDataInte
rface context) {
                super(context,
"GetScrapData", "(III)S", 0x00000FE0);
        }
}
```



Why don't we Represent *Programs* Abstractly?

```
public class FileTranslationSpecStruct
extends ByteArrayStruct {

        /**
         * Constructs an uninitialized
<CODE>FileTranslationSpecStruct</CODE>
         */
        public
FileTranslationSpecStruct() {


super(sizeOfFileTranslationSpec);
        }

        /**
         * Constructs a
<CODE>FileTranslationSpecStruct</CODE>
and initializes it with the data in
another <CODE>Struct</CODE>.
         * Useful when a C struct
<CODE>FileTranslationSpec</CODE> is
embedded in another C struct and you
want to
<CODE>ScrapType requestedFormat</CODE>
         * @param dataH
        in C: <CODE>Handle dataH</CODE>
         * @param srcDataGetterRefConin
C: <CODE>void *srcDataGetterRefCon</
CODE>
         *
@return                         in C:
<CODE>OSErr </CODE>
         */
        public short GetScrapData(int
requestedFormat, int dataH, int
srcDataGetterRefCon);
}

/**
 * The class
<CODE>GetScrapDataClosureUPP</CODE>
defines thunk objects which support
transitioning
 * from the MacOS
<CODE>GetScrapDataProcPtr</CODE>
calling convention into the MRJ Java
VM.<BR>
 *<BR>
 *
 * @see GetScrapDataInterface
 */
public final class
GetScrapDataClosureUPP extends
MethodClosureUPP {

        /**
         * To instantiate a
<CODE>GetScrapDataClosureUPP</CODE>
object you must pass the constructor an
object which implements
         * the
<CODE>GetScrapDataInterface interface</
CODE>.<BR>
         * Do not subclass this class.
It is a thunk that glues native code to
java code.
         */
        public
GetScrapDataClosureUPP(GetScrapDataInte
rface context) {
                super(context,
"GetScrapData", "(III)S", 0x00000FE0);
        }
}
```

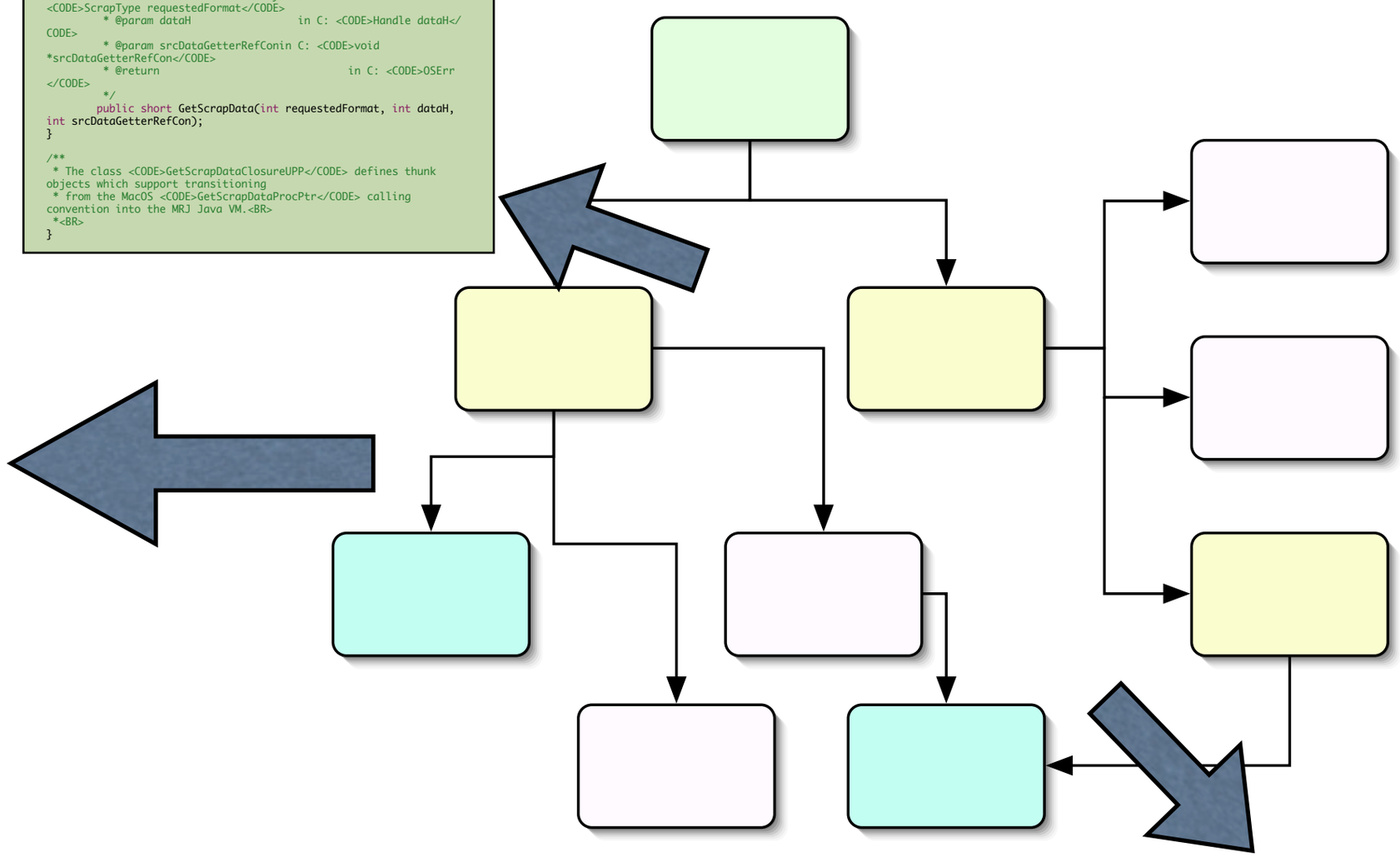


```
public class FileTranslationSpecStruct
extends ByteArrayStruct {

        /**
         * Constructs an uninitialized
<CODE>FileTranslationSpecStruct</CODE>
         */
        public
FileTranslationSpecStruct() {


super(sizeOfFileTranslationSpec);
        }

        /**
```

Why don't we Represent *Programs* Abstractly?

```
public class FileTranslationSpecStruct
extends ByteArrayStruct {

        /**
         * Constructs an uninitialized
<CODE>FileTranslationSpecStruct</CODE>
         */
        public
FileTranslationSpecStruct() {


super(sizeOfFileTranslationSpec);
        }

        /**
         * Constructs a
<CODE>FileTranslationSpecStruct</CODE>
and initializes it with the data in
another <CODE>Struct</CODE>.
         * Useful when a C struct
<CODE>FileTranslationSpec</CODE> is
embedded in another C struct and you
want to
<CODE>ScrapType requestedFormat</CODE>
         * @param dataH
        in C: <CODE>Handle dataH</CODE>
         * @param srcDataGetterRefConin
C: <CODE>void *srcDataGetterRefCon</
CODE>
         *
@return                          in C:
<CODE>OSErr </CODE>
         */
        public short GetScrapData(int
requestedFormat, int dataH, int
srcDataGetterRefCon);
}

/**
 * The class
<CODE>GetScrapDataClosureUPP</CODE>
defines thunk objects which support
transitioning
 * from the MacOS
<CODE>GetScrapDataProcPtr</CODE>
calling convention into the MRJ Java
VM.<BR>
 *<BR>
 *
 * @see GetScrapDataInterface
 */
public final class
GetScrapDataClosureUPP extends
MethodClosureUPP {

        /**
         * To instantiate a
<CODE>GetScrapDataClosureUPP</CODE>
object you must pass the constructor an
object which implements
         * the
<CODE>GetScrapDataInterface interface</
CODE>.<BR>
         * Do not subclass this class.
It is a thunk that glues native code to
java code.
         */
        public
GetScrapDataClosureUPP(GetScrapDataInte
rface context) {
                super(context,
"GetScrapData", "(III)S", 0x00000FE0);
        }
}
```

```
         * Constructs a <CODE>FileTranslationSpecStruct</CODE> and
initializes it with the data in another <CODE>Struct</CODE>.
         * Useful when a C struct <CODE>FileTranslationSpec</CODE>
is embedded in another C struct and you want to
<CODE>ScrapType requestedFormat</CODE>
         * @param dataH                in C: <CODE>Handle dataH</
CODE>
         * @param srcDataGetterRefConin C: <CODE>void
*srcDataGetterRefCon</CODE>
         * @return                            in C: <CODE>OSErr
</CODE>
         */
        public short GetScrapData(int requestedFormat, int dataH,
int srcDataGetterRefCon);
}

/**
 * The class <CODE>GetScrapDataClosureUPP</CODE> defines thunk
objects which support transitioning
 * from the MacOS <CODE>GetScrapDataProcPtr</CODE> calling
convention into the MRJ Java VM.<BR>
 *<BR>
}
```



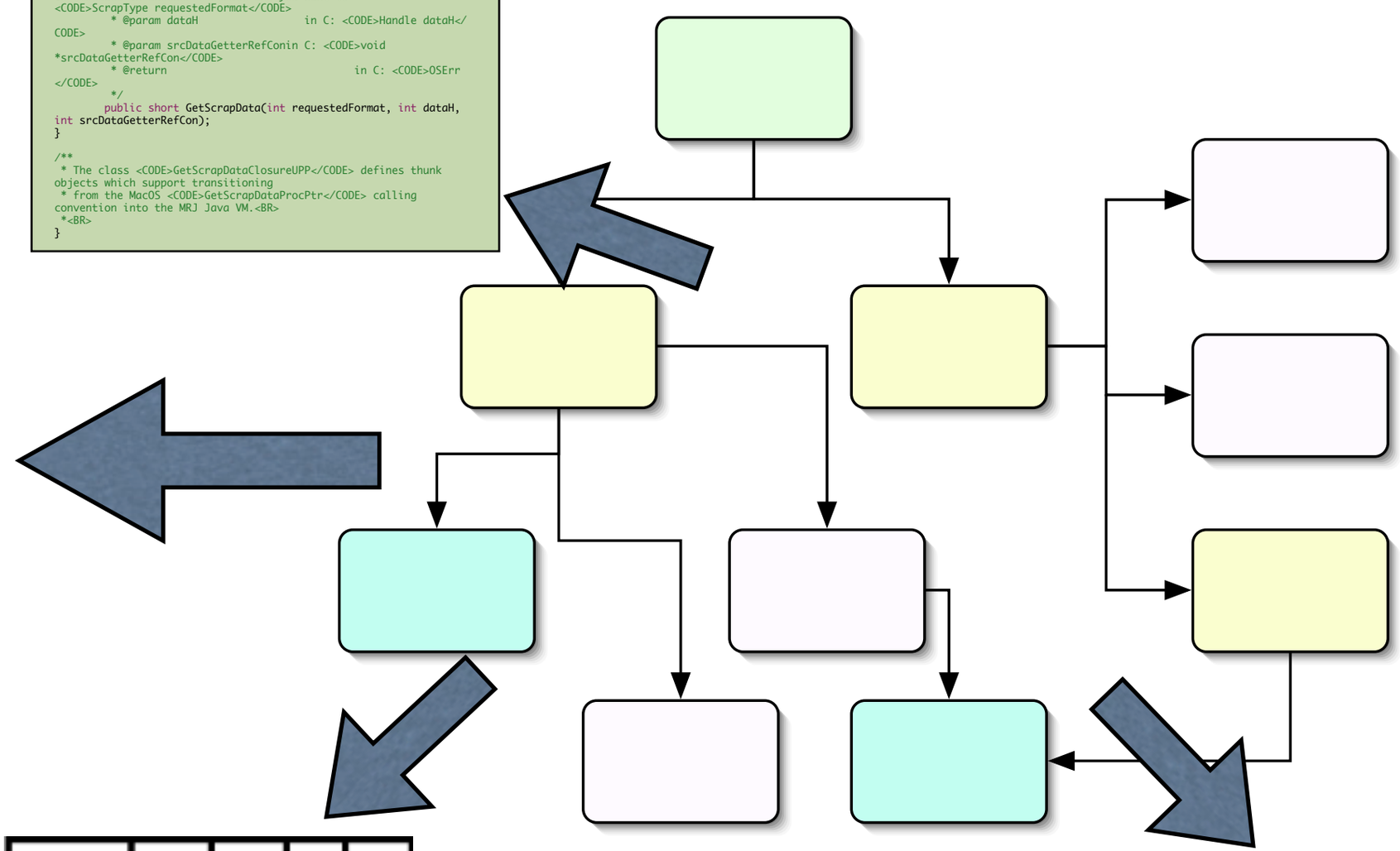
```
public class FileTranslationSpecStruct
extends ByteArrayStruct {

        /**
         * Constructs an uninitialized
<CODE>FileTranslationSpecStruct</CODE>
         */
        public
FileTranslationSpecStruct() {


super(sizeOfFileTranslationSpec);
        }

        /**
```

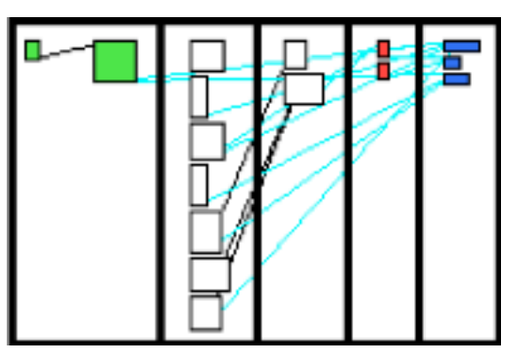
Portland State UNIVERSITY

Why don't we Represent *Programs* Abstractly?

```
public class FileTranslationSpecStruct
extends ByteArrayStruct {

        /**
         * Constructs an uninitialized
<CODE>FileTranslationSpecStruct</CODE>
         */
        public
FileTranslationSpecStruct() {


super(sizeOfFileTranslationSpec);
        }

        /**
         * Constructs a
<CODE>FileTranslationSpecStruct</CODE>
and initializes it with the data in
another <CODE>Struct</CODE>.
         * Useful when a C struct
<CODE>FileTranslationSpec</CODE> is
embedded in another C struct and you
want to
<CODE>ScrapType requestedFormat</CODE>
         * @param dataH
        in C: <CODE>Handle dataH</CODE>
         * @param srcDataGetterRefConin
C: <CODE>void *srcDataGetterRefCon</
CODE>
         *
@return                          in C:
<CODE>OSErr </CODE>
         */
        public short GetScrapData(int
requestedFormat, int dataH, int
srcDataGetterRefCon);
}

/**
 * The class
<CODE>GetScrapDataClosureUPP</CODE>
defines thunk objects which support
transitioning
 * from the MacOS
<CODE>GetScrapDataProcPtr</CODE>
calling convention into the MRJ Java
VM.<BR>
 *<BR>
 *
 * @see GetScrapDataInterface
 */
public final class
GetScrapDataClosureUPP extends
MethodClosureUPP {

        /**
         * To instantiate a
<CODE>GetScrapDataClosureUPP</CODE>
object you must pass the constructor an
object which implements
         * the
<CODE>GetScrapDataInterface interface</
CODE>.<BR>
         * Do not subclass this class.
It is a thunk that glues native code to
java code.
         */
        public
GetScrapDataClosureUPP(GetScrapDataInte
rface context) {
                super(context,
"GetScrapData", "(III)S", 0x00000FE0);
        }
}
```

```
         * Constructs a <CODE>FileTranslationSpecStruct</CODE> and
initializes it with the data in another <CODE>Struct</CODE>.
         * Useful when a C struct <CODE>FileTranslationSpec</CODE>
is embedded in another C struct and you want to
<CODE>ScrapType requestedFormat</CODE>
         * @param dataH                in C: <CODE>Handle dataH</
CODE>
         * @param srcDataGetterRefConin C: <CODE>void
*srcDataGetterRefCon</CODE>
         * @return                            in C: <CODE>OSErr
</CODE>
         */
        public short GetScrapData(int requestedFormat, int dataH,
int srcDataGetterRefCon);
}

/**
 * The class <CODE>GetScrapDataClosureUPP</CODE> defines thunk
objects which support transitioning
 * from the MacOS <CODE>GetScrapDataProcPtr</CODE> calling
convention into the MRJ Java VM.<BR>
 *<BR>
}
```





```
public class FileTranslationSpecStruct
extends ByteArrayStruct {

        /**
         * Constructs an uninitialized
<CODE>FileTranslationSpecStruct</CODE>
         */
        public
FileTranslationSpecStruct() {


super(sizeOfFileTranslationSpec);
        }

        /**
```

I mistook a clear vision

for a short distance

Portland State
UNIVERSITY

What Views are Interesting?

What Views are Interesting?

- Every Refactoring defines an equivalence relation
  - the 2 source forms are views on the same abstraction

What Views are Interesting?

- Every Refactoring defines an equivalence relation
  - the 2 source forms are views on the same abstraction
- Rows *vs.* Columns: classes *vs.* generic operations
  - equally easy to add classes or operations

What Views are Interesting?

- Every Refactoring defines an equivalence relation
  - the 2 source forms are views on the same abstraction
- Rows *vs.* Columns: classes *vs.* generic operations
  - equally easy to add classes or operations
- Program dependencies, *e.g.*,
  - all methods (or statements) that touch a field

What Views are Interesting?

- We need new language features to express new views
  - generic operations (columns) view cannot be expressed in Java — hence Sweet, AspectJ
- Different component-wise decompositions
  - trait view and flat view of a class
- type-annotated view of an implicitly typed language
  - or untyped view of an explicitly-typed language

Consequences

- Many language design questions can be un-asked:
  - keywords *vs.* symbols *vs.* layout
  - type inference *vs.* declaration
- Programming language subsumed by environment

Issues

- Most views are partial
  - *e.g.* extract method refactoring demands a name for the extracted method
  - *e.g.* view of all statements touching a given field
- How does one edit in a partial view?
  - allow the edit only if it is unambiguous?
  - ask the user for guidance?

What views should we choose?

Two approaches:

1. Start with the tools that we have
   - Program slicing, PDGs, structure-folding, etc.
   - Sweet: rows into columns
     - ADT-like view of an OO program
2. Start with a programming task
   - Preferably one that programmers are struggling with!
   - Ask: what views might help?

How to Evaluate Success?

- Evaluations must be *task-centered:*
  - Pick a particular (difficult) programming task
  - See if the multi-view environment helps programmers to do a better job
- Our chosen tasks: program refactorings