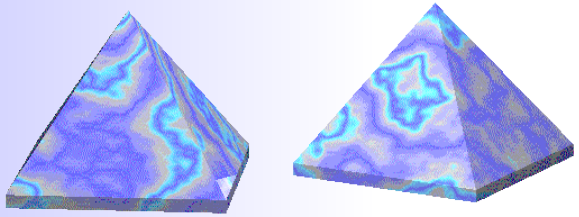
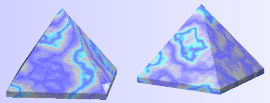
Java™ Introduction

Renaud Florquin
Isabelle Leclercq



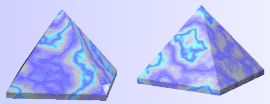
FloConsult SPRL

http://www.floconsult.be
mailto:info@floconsult.be

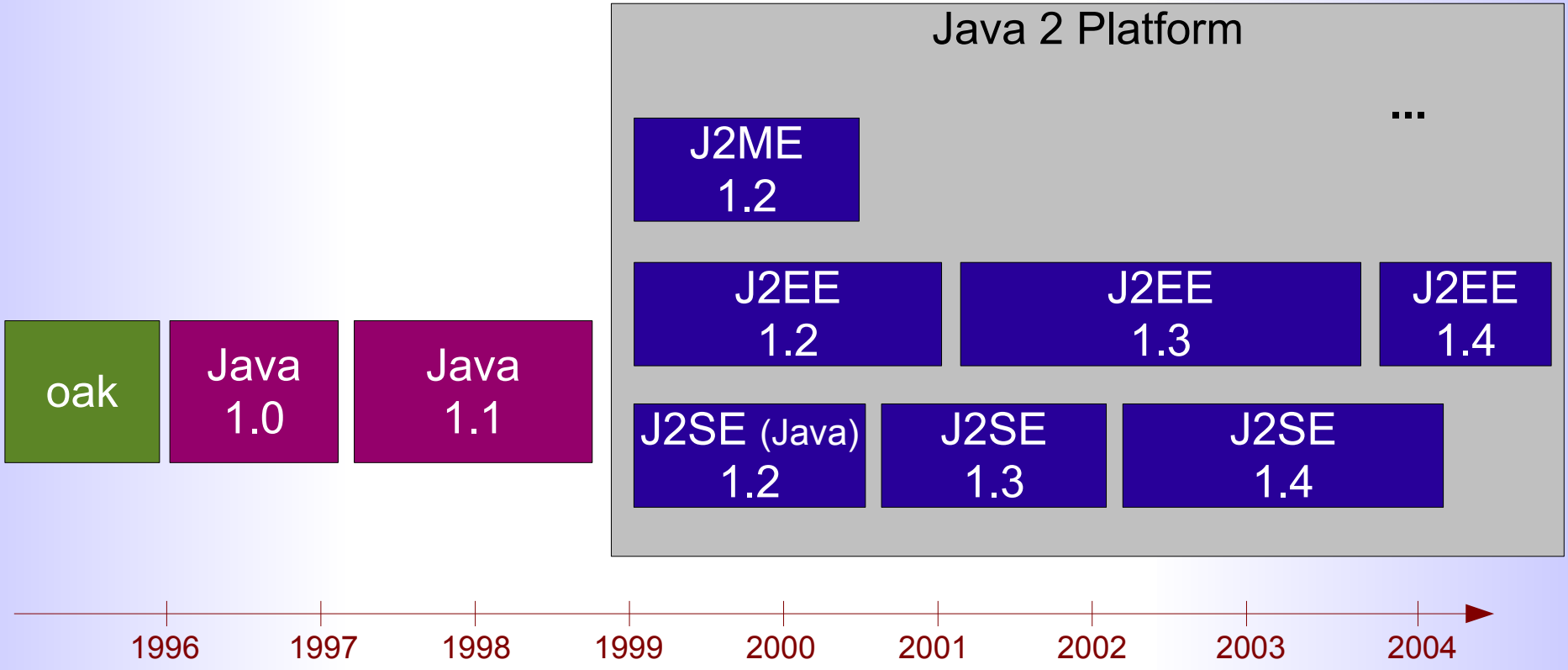


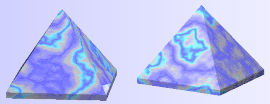
Java Technical Virtues

- "Write once, run anywhere"
- Get started quickly
- Write less code
- Write better code
- Develop programs more quickly
- Avoid platform dependencies with 100% pure Java
- Distribute software more easily



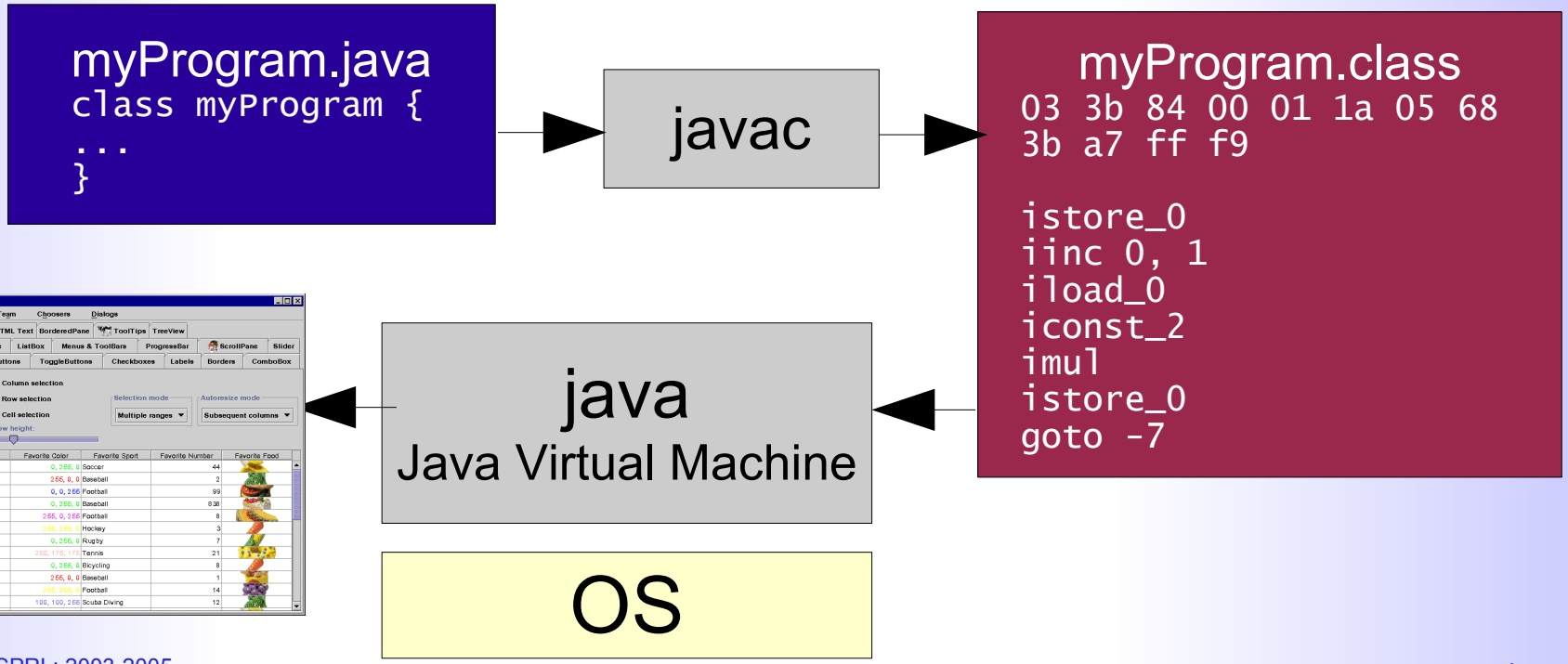
Java: History (1)

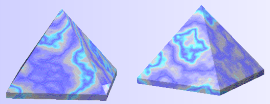




Java: Application production

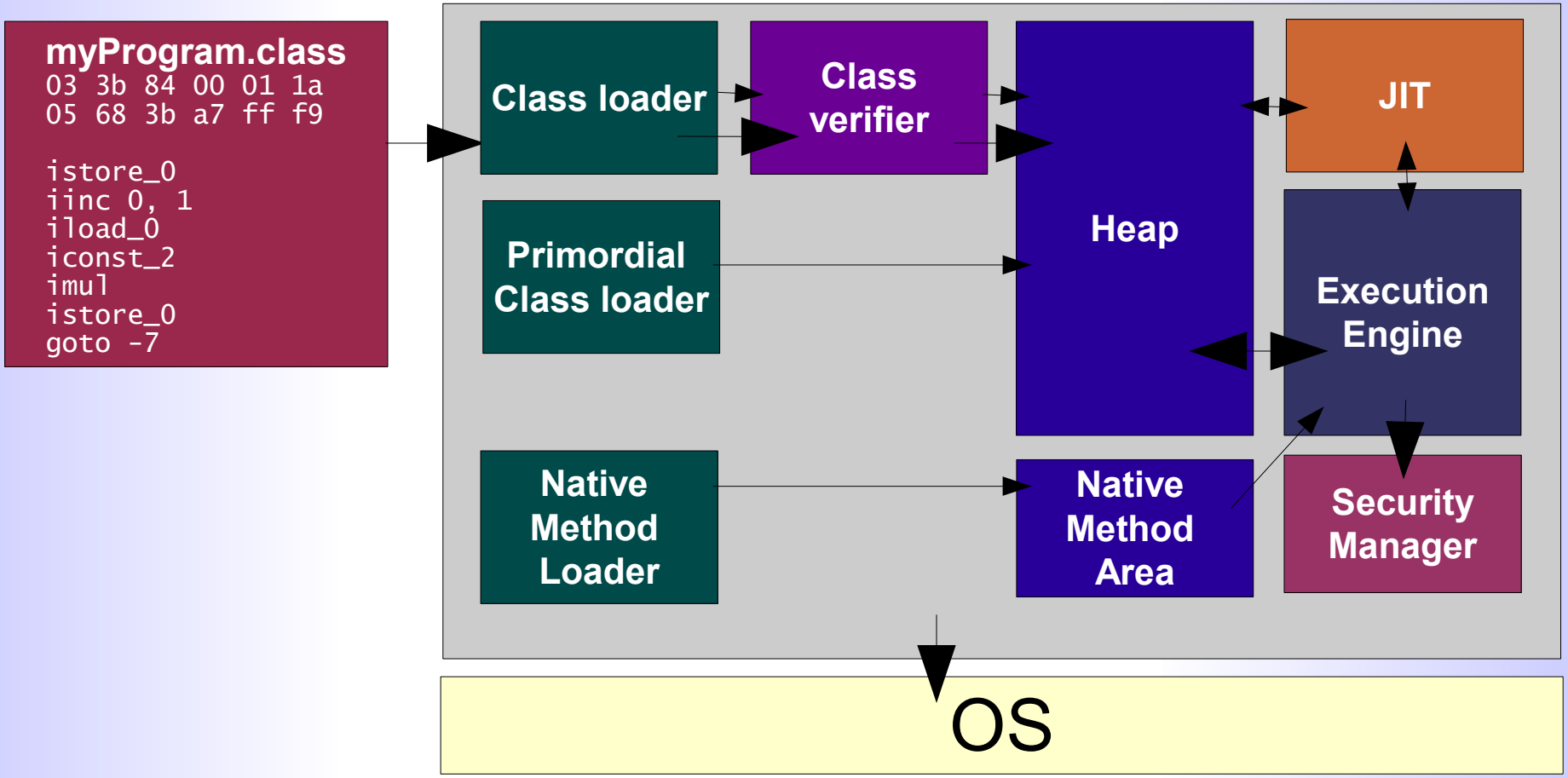
- Compile AND interpret
  - Compiler makes "Java Bytecodes" (*.class)
  - Interpreter (JVM) executes Bytecodes

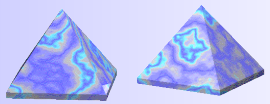




Java Virtual Machine

- Interprets Java Byte Code





Compilation phases

myProgram.java
```
class myProgram {
...
}
```

Parser

Syntax tree

tokens

Scanner

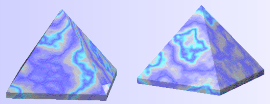
Constrainer

Code Generator

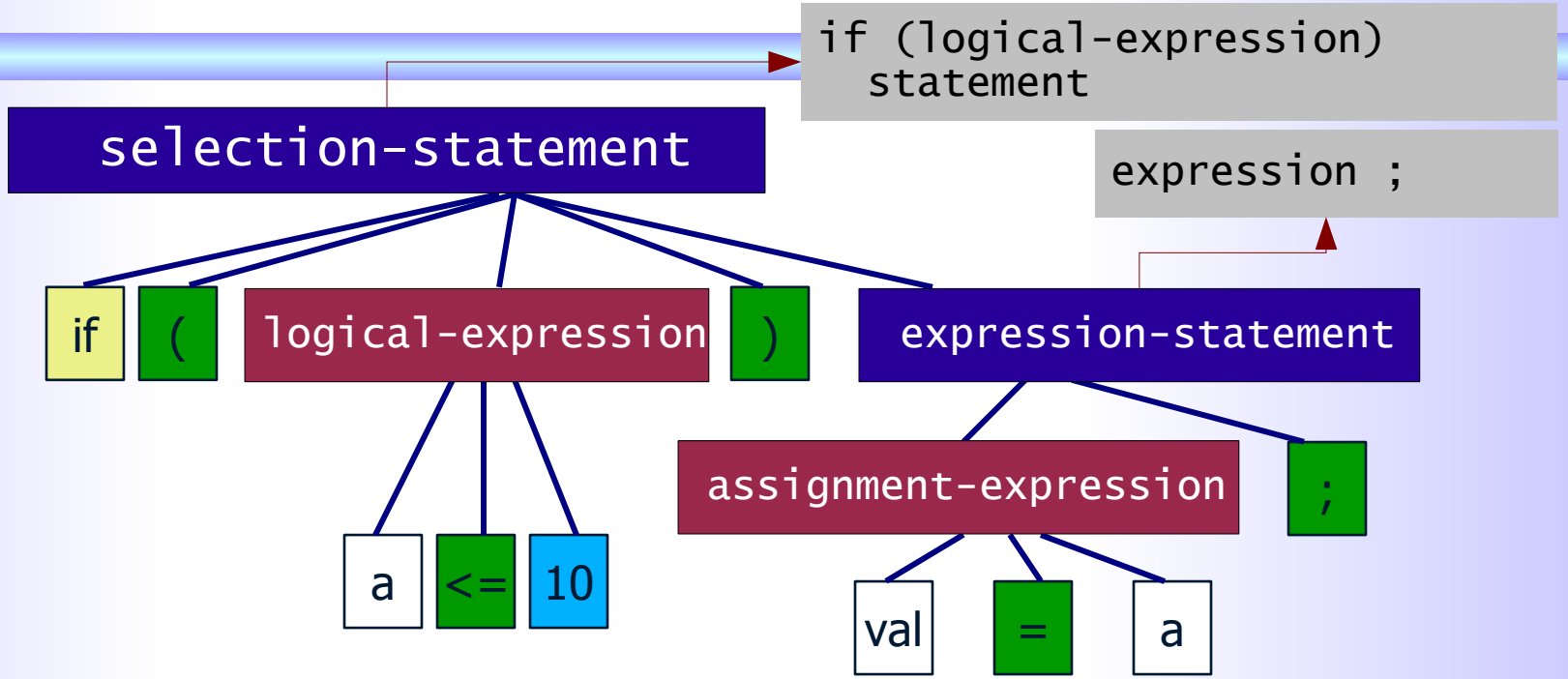
Symbol Table	
position	int
...	...

myProgram.class
```
03 3b 84 00 01 1a 05 68
3b a7 ff f9

istore_0
iinc 0, 1
iload_0
iconst_2
imul
istore_0
goto -7
```

Compiler

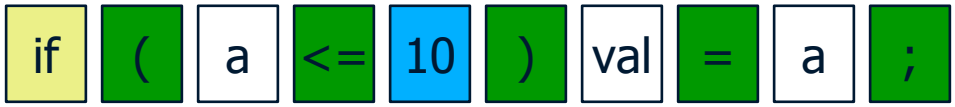


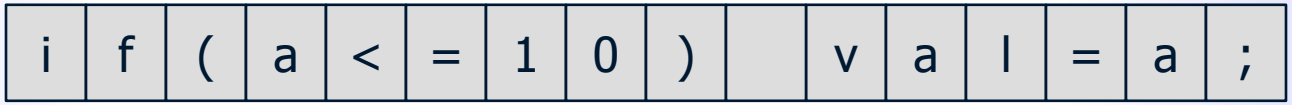
Compilation phases

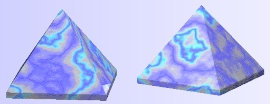


Parser

Scanner

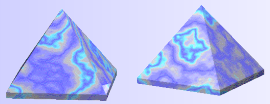






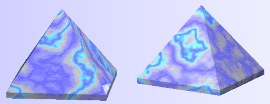
Types

- Java programming language is a *strongly typed* language
  - Everything that has a *value* also has a *type.*
- In Java there are two kinds of types:
  - Primitive Types *(Built-in data types).*
  - Reference Types *(Everything else, including objects)*



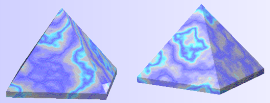
Primitive Types

Primitive	Description	Wrapper class
`boolean`	Either **`true`** or **`false`**	`Boolean`
`char`	16-bit Unicode character ('\u0000' .. '\uffff')	`Character`
`byte`	8-bit signed two's complement integer [-2^7.. $+2^7-1$]	`Byte`
`short`	16-bit signed two's complement integer[-2^{15}.. $+2^{15}-1$]	`Short`
`int`	32-bit signed two's complement integer[-2^{31}.. $+2^{31}-1$]	`Integer`
`long`	64-bit signed two's complement integer[-2^{63}.. $+2^{63}-1$]	`Long`
`float`	32-bit IEEE 754-1985 floating-point number	`Float`
`double`	64-bit IEEE 754-1985 floating-point number	`Double`



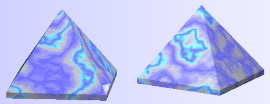
Reference Types

- Everything that is not a primitive type is a reference type
- Three kinds of reference types:
  - *Classes*
    `String`
    `java.lang.Integer`
  - *Interfaces*
    `Serializable`
  - *Arrays*
    `int[]`
    `String[]`



Array of What ?

- You can have an array of anything, but all the elements of an array are of the same type.
  - any primitive type.
  - any reference type.
- You need to declare the type of an array (the type of each element).
- You also need to explicitly create an array
  - just declaring the type is not enough.



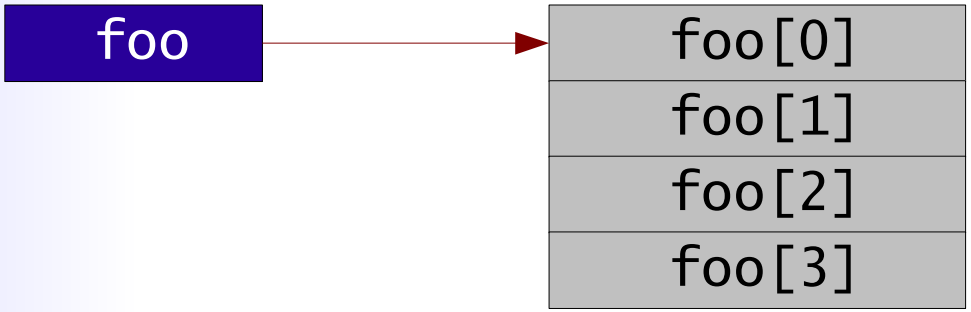
Creating an Array

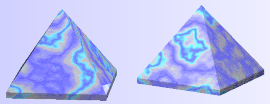
- Use the new operator:

```
foo   = new int[4];
```

- We can combine the declaration and creation

```
int[] foo       = new int[4];
```



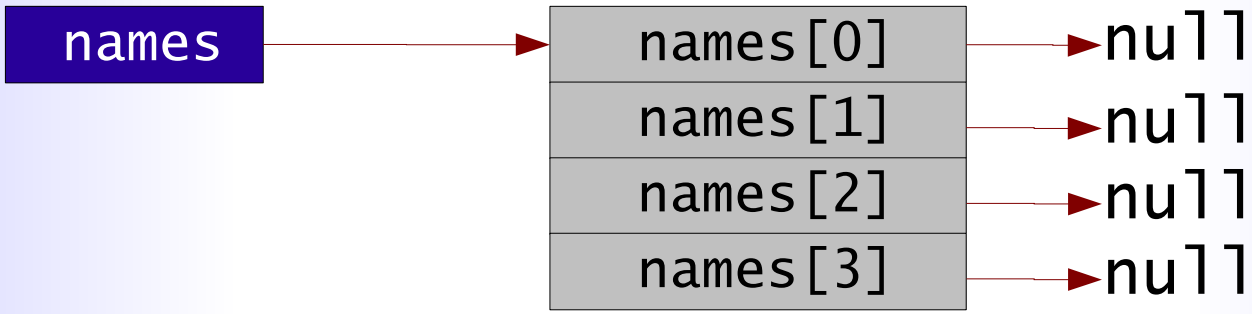


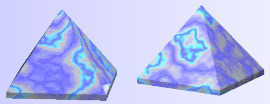
Creating an Array

- Reference type:

```
names = new String[4];
// or
String[] names = new String[4];
```

- How many objects are created ?

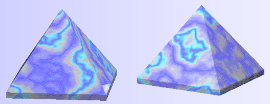




Array Length

- The length of an array is available as a field of the array.

```
byte[] buffer = new byte[100];


for (int i=0; i < buffer.length; i++) {
  buffer[i] = i;
}
```

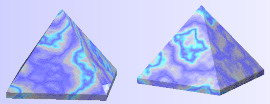


Array Initializers

- You can use an *array literal* to create an array
  - you assign the literal to the array variable.

Example: Array Initialializers

```
int[] foo = {4, 3, 2, 1, 0};

for (int i =0; i < foo.length; i++) {
  System.out.println("Foo[" + i + "]=" + foo[i]);
}

int[][] pascalsTriangle = {
    { 1 },
    { 1 , 1},
    { 1 , 2, 1},
    { 1 , 3, 3, 1},
  };
```

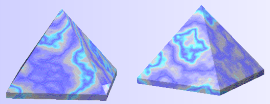


Initial Values of Variables

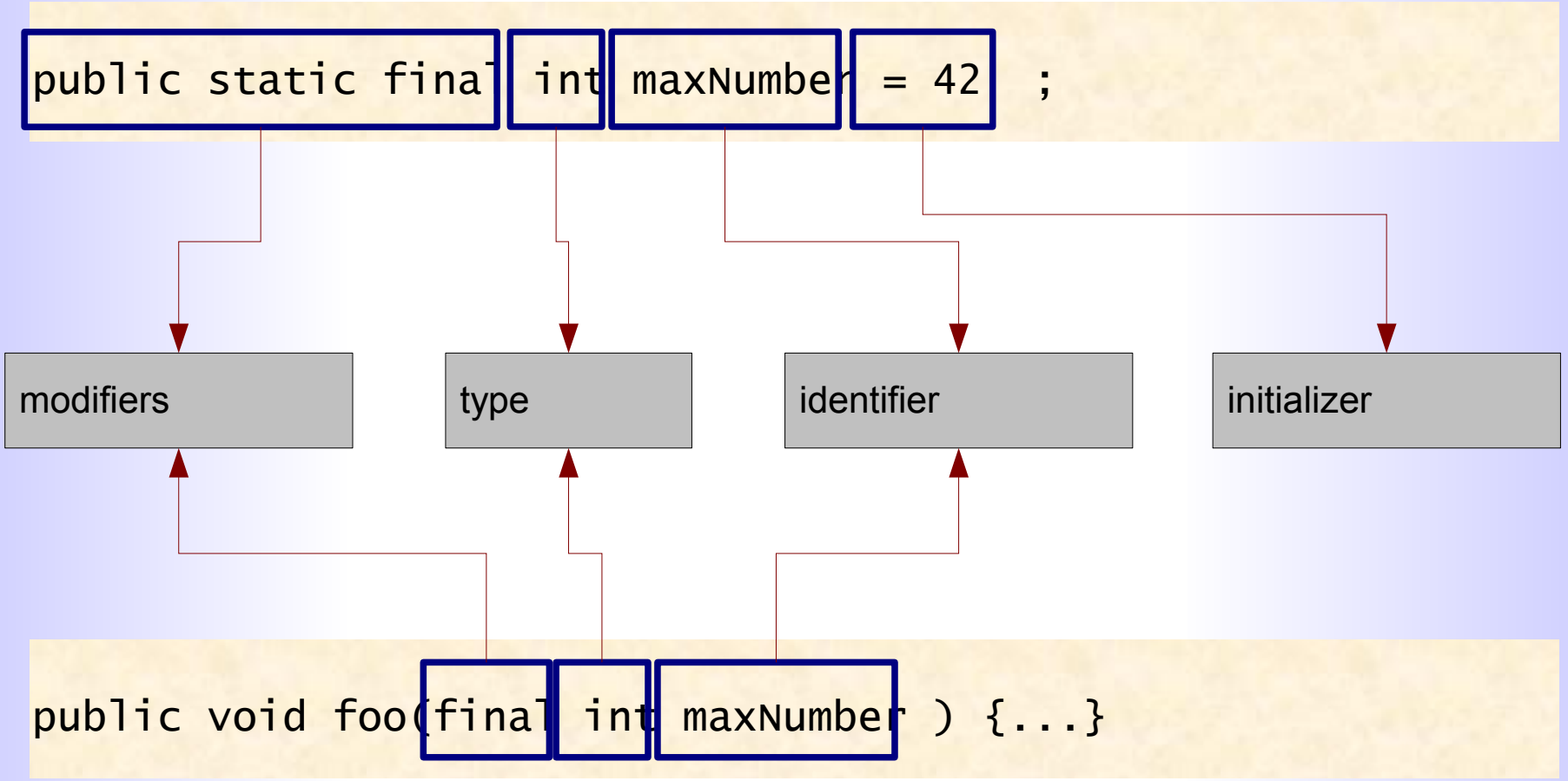
- For the class variables and instance variables:

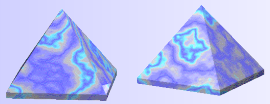
boolean	false
char	'\u0000'
byte, short, int, long	0
float	+0.0f
double	+0.0
Object reference	null

- Parameters
  - Provided by the invoker
- Local Variable
  - Must be explicit (verified by the compiler)



Variables


public static final int maxNumber = 42 ;
modifiers
type
identifier
initializer
public void foo(final int maxNumber) {...}

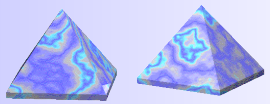


Assignment Expressions

- General form

```
lvalue = expression
```

  - Overwrites the current value of an object (lvalue, l standing for left) with a new value (expression)
  - The type of the expression must match the type of the object (caution: implicit conversion)
  - The type and the value of an Assignment expression are the type and the value of the lvalue



Assignment Expressions

- Compound assignment

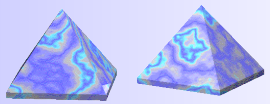
```
lvalue op= expression
```

  - Equivalent to:

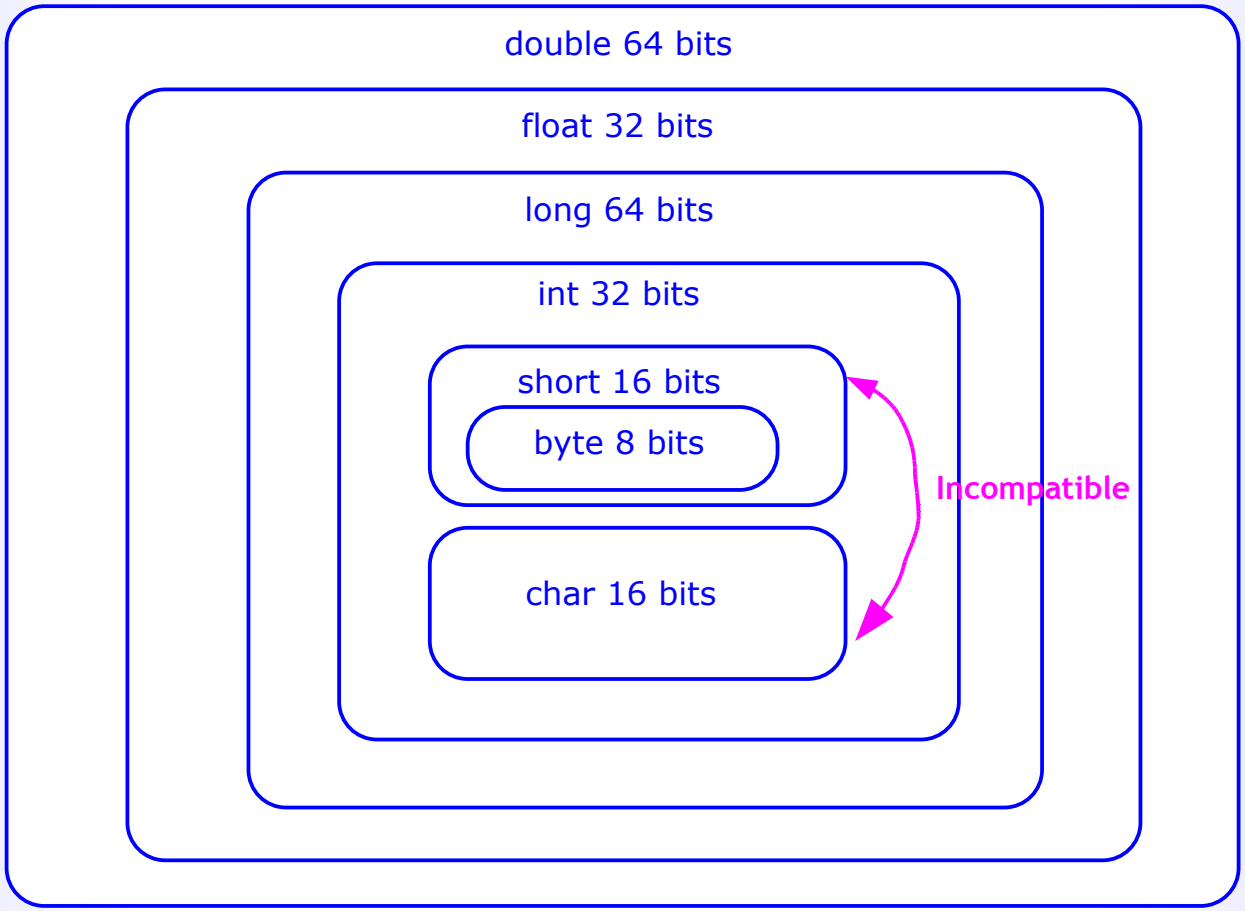
```
lvalue = lvalue op expression
```

- Example:

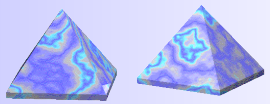
```
int i = 0;

i = i + 3;
i += 2;       // shortcut assignment

a = b = 2;
```



Widening Implicit conversions

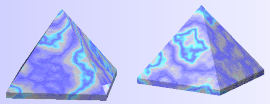


No conversion for booleans

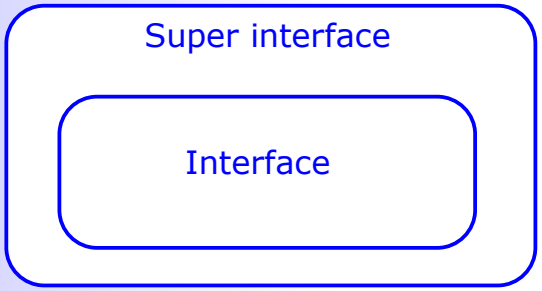


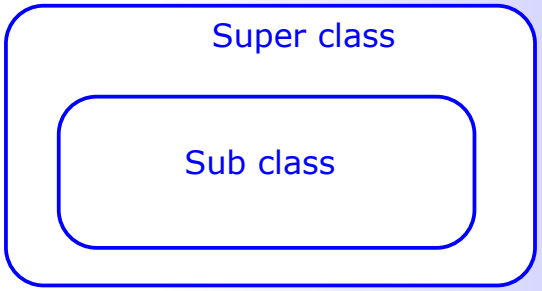
Conversion during numeric ops

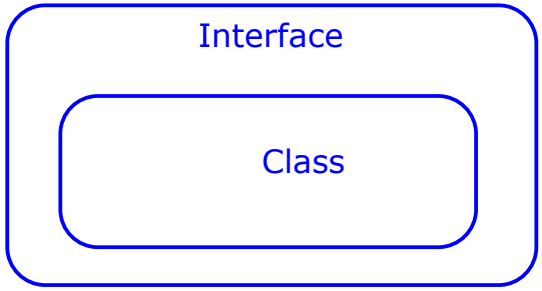
- For unary operators:
  - ++, --: no conversion
  - +, -, ^: byte, short, char promoted to int
- For binary operators:
  - If one operand double: other promoted double before operation
  - Else, if one is float: other promoted float before operation
  - Else, if one is long: other promoted long before operation
  - Else, both promoted int before operation

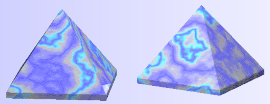


Object ref. implicit conversion



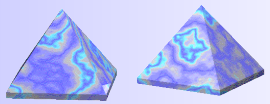




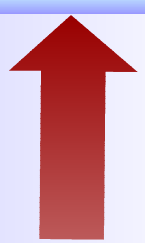


Conversion to String

- All objects may be converted to a String
  - Implicit call of toString()
  - e.g. String s = "" + 2 → s = "2", type String



Expression evaluation



Operator Precedence

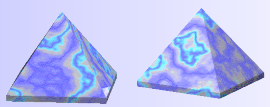
[]	.	()	expr++	expr--	
++expr	--expr	+expr	-expr	~	!
new					
*	/	%			

=	+=	...

est priority

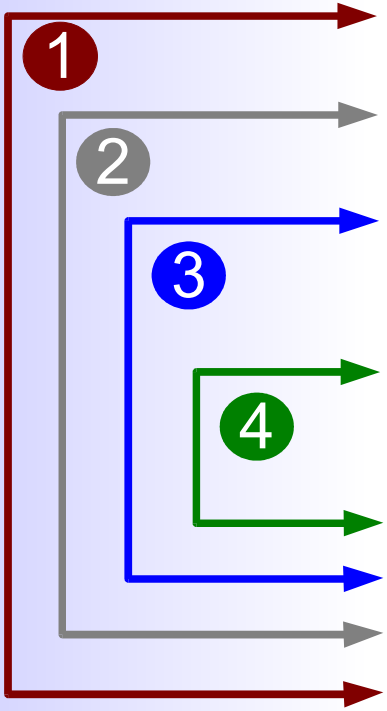
est priority

```
3 + 4/2 * 10 = ?
(3 + 4/2) * 10 = ?
```

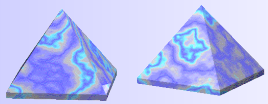


Compound Statements

Example

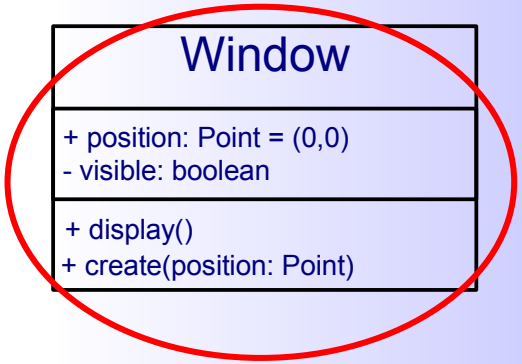


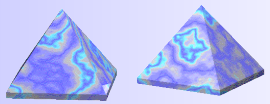

```
public class Test
{
  public void aMethod()
  {
    int j = 2;   // j is defined for level 2-3-4
    {
      int i = 0; // i is defined for level 3-4
      while (i < 10)
      {
        System.out.println("Value: " + i*j);
        i++;
      }
    }
  }
}
```



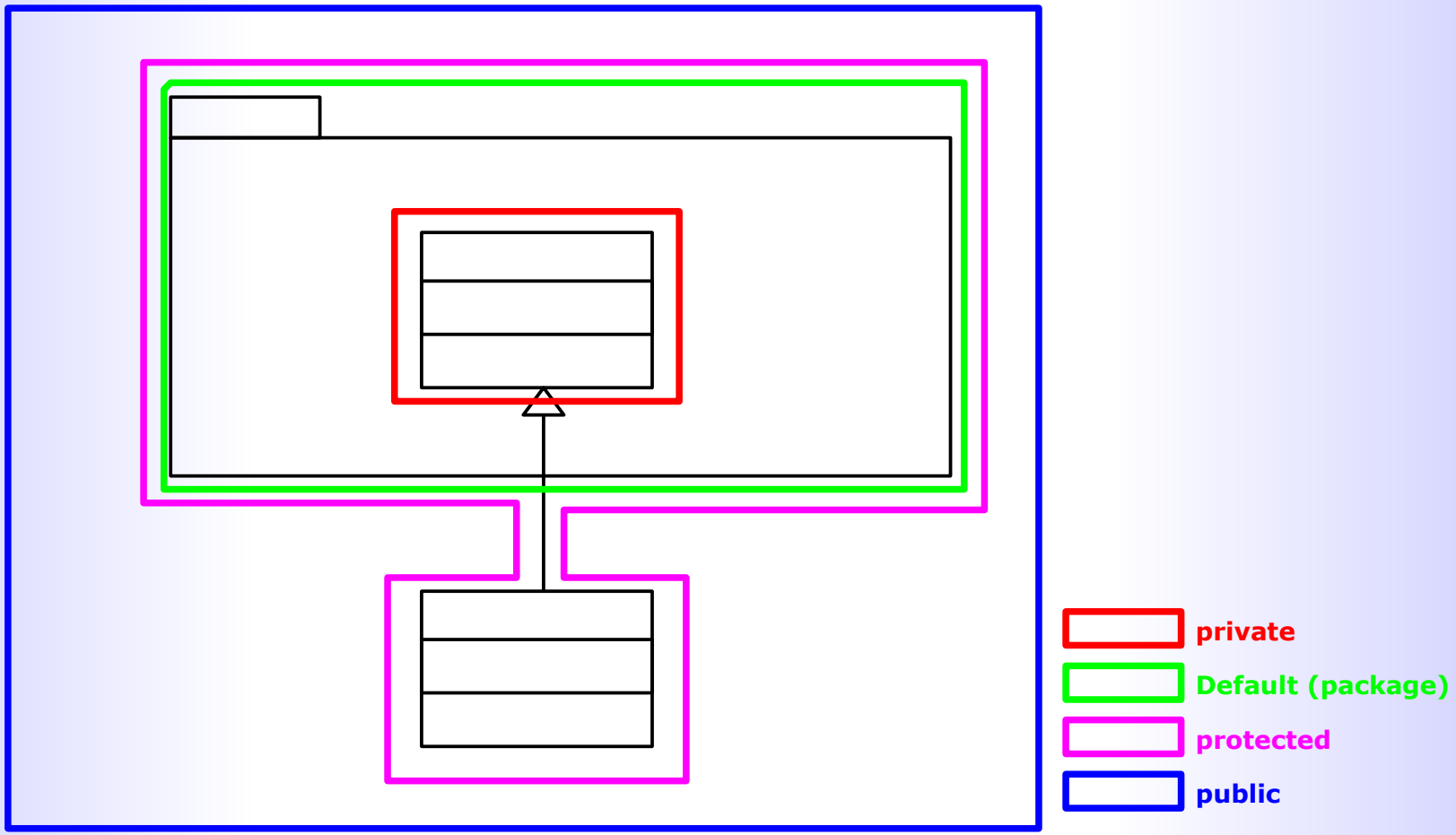
Class Definition

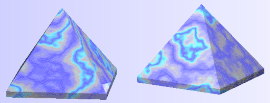
- Class Members
  - Fields (= attributes)
  - Methods
  - Nested classes and interfaces
- Class Modifiers
  - public: publicly accessible
    - otherwise only accessible within its own package
  - abstract: incomplete class (no instance)
  - final: cannot be subclassed





Access control (2)

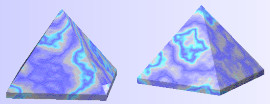




Modifiers

Modifiers	Class	Package	Subclass	World
`private`	✔	✘	✘	✘
No specifier	✔	✔	✘	✘
`protected`	✔	✔	✔	✘
`public`	✔	✔	✔	✔

`static`	Class variable (shared by every instance)
`final`	The variable's value cannot be changed
`transient`	The variable is not part of the persistent state (should not be serialized).
`volatile`	The variable can be changed asynchronously.

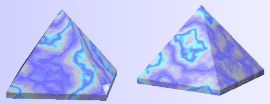


static Fields

- One instance of a field shared by all objects of a class (class variable)
- How to access a static field?
  - Via the class name
  - Via a reference to an object

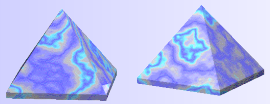
Example

```
public class Window {
  public static nbrWindow;
...
}

System.out.println(Window.nbrWindow);
Window w = new Window();
System.out.println(w.nbrWindow);
```



final Fields

- Immutable property of a class (static+final)
  - Initialization
    - Via Initializer
    - Via static initialization block (see further)
- Immutable property of an object (final).
  - Initialization
    - Via Initializer
    - Constructor
    - Via initialization block (see further)



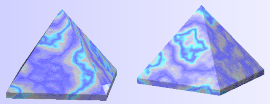
Inheritance - Example (1)

Example

```
class Node {
  private final String name;
  private Object value = null;
  public Node(String name, Object value) { ...}
  public String getName() { return name; }
  ...
}
class ColorNode extends Node {
  private  Color c;
  public ColorNode(String name, Object value, Color c) {
    super(name, value);
    this.c = c;
  }
  public Color getColor() { return c; }
  ...
}
```

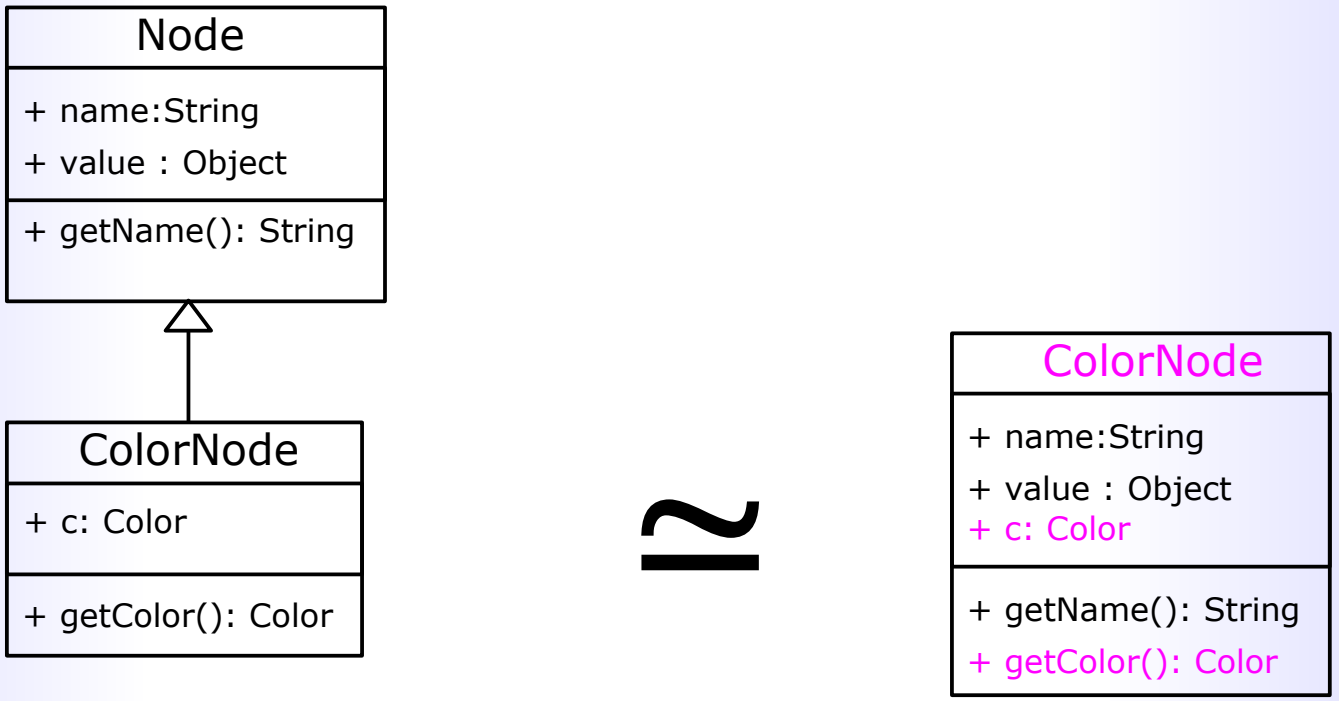
Inheritance

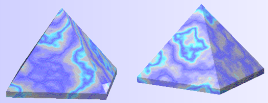
Superclass constructor invocation



Inheritance - Example (2)

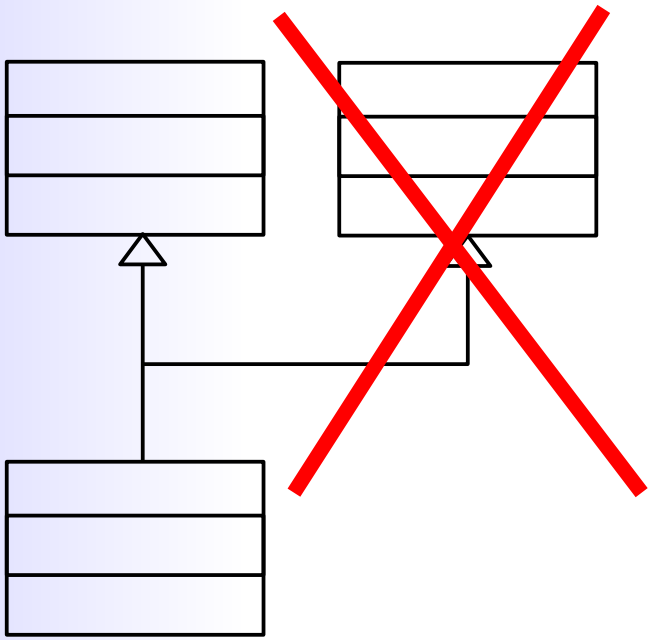
- ColorNode adds attributes and methods to Node:

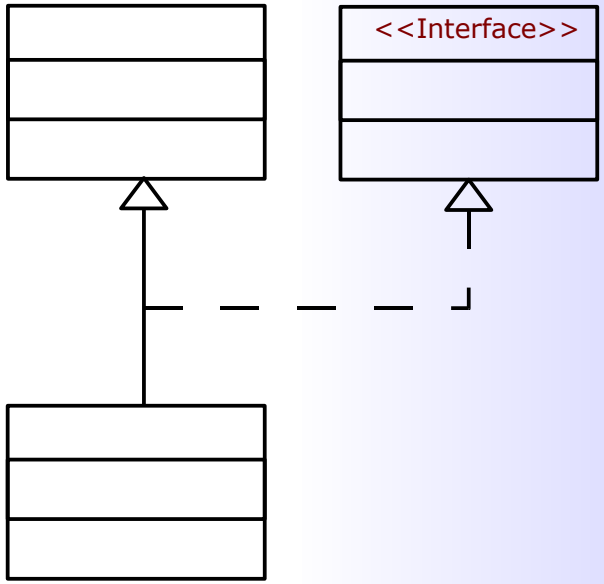


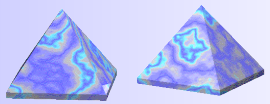


No multiple inheritance

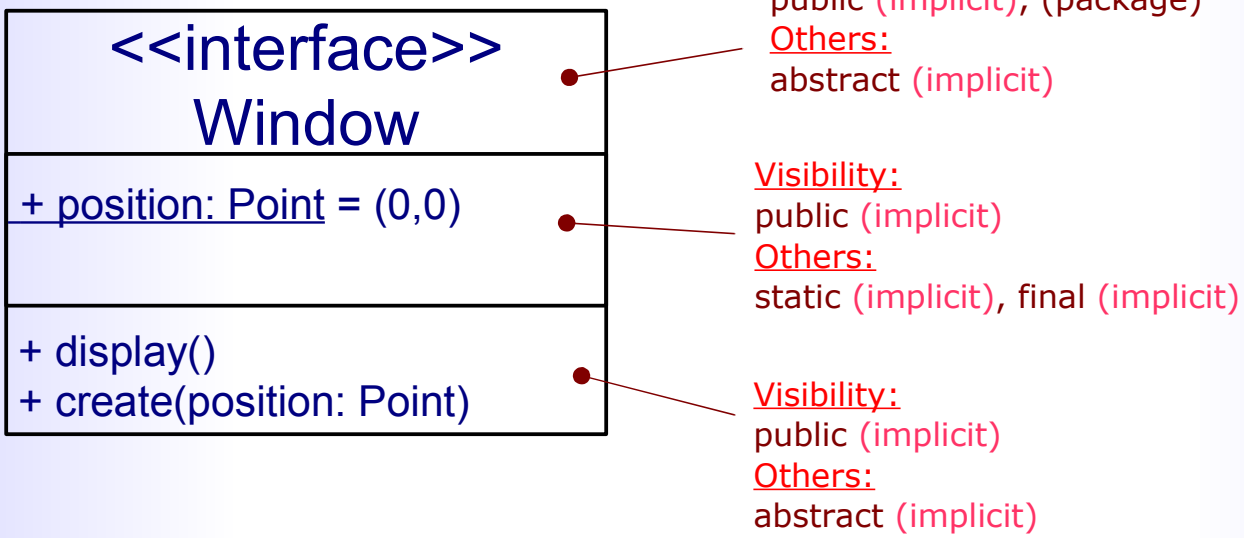
- No multiple inheritance of classes in Java
- But: multiple inheritance of interfaces allowed

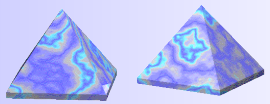






Interface Keywords summary

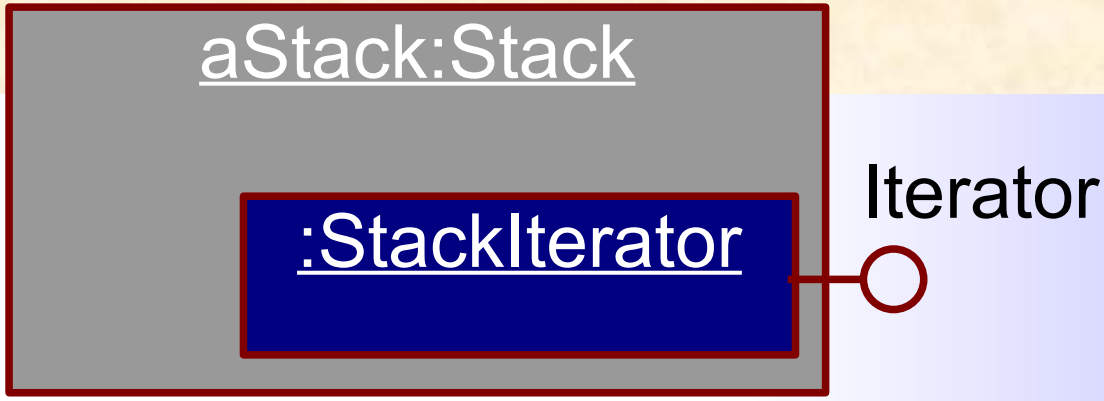


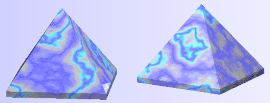


Inner Classes

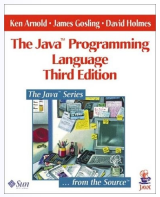
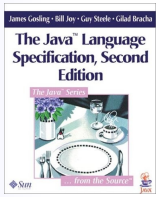
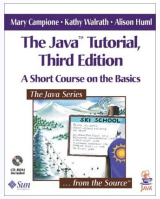
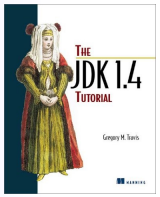
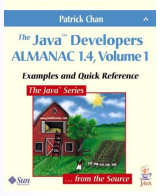
Example

```
public class Stack {
  private Vector items;
  ...
  public Iterator getIterator() {
    return new StackIterator();
  }
  class StackIterator implements Iterator {
    int currentItem = items.size() - 1;
    public boolean hasNext() { ...}
    ...
  }
}
```





Bibliography

	**The Java Programming Language 3rd Edition** by Ken Arnold, James Gosling, David Holmes; 705 p, June 2000, Addison-Wesley, ISBN 0-20170-433-1
	**Java(TM) Language Specification 2d Edition** by Bill Joy, Guy Steele, James Gosling, Gilad Bracha; 544 p, June 2000, Addison-Wesley, ISBN 0-20131-008-2
	**The Java Tutorial 3d Edition** by Mary Campione and Kathy Walrath; 580 p, January 2000, Addison-Wesley, ISBN 0-20170-393-9
	**The JDK 1.4 Tutorial** by Gregory Travis; 408 p, March 2002, Manning publications Company, ISBN 1930110456
	**The Java(TM) Developers Almanach 4th Edition, volume 1** by Patrick Chan; 1024 p, March 2002, Addison-Wesley, ISBN 0-20175-280-8