Kennismaking met Java

- Drie maal twee uur uitleg en voorbeelden

Java

- 1994: James Gosling - SUN Microsystems
- Designed to program intelligent appliances
- Launched to send programs over the Web
- Object oriented programming style
- Allows multi-threaded execution
- Syntax borrowed from C and C++
- “Cleaner” semantics than C and C++
- Many object libraries widely available

C

- 1972 : Brian Kernighan & Dennis Richie
  - Need to rewrite the UNIX operating system for computer with different assembly language.
  - Other future rewrites likely
- Design of a new High (?) Level Language
  - Minimizing coding work
  - Giving access to all machine details
  - Block structured for detail hiding
  - Easy to compile

C++

- 1987 : Maintenance and debugging of "quick & dirty" C programs is a nightmare !
- Bell Labs design new, object oriented, language, adopting syntax of C for facilitating its adoption by "hard core" programmers.
- Presently, the most used language in professional software development, when reliability is not of prime importance.

To learn programming with Java

Douglas Bell, Mike Parr
Java for students (third edition)
Pearson Education, 1998, 2002
ISBN 0 130 32377 2

Some JAVA References

- Rogers Cadenhead
  Teach yourself Java 1.1 programming in 24 hours
  Sams.net Publishing, 1997
  ISBN : 1-57521-270-6
- H.M. Deitel, P.J. Deitel
  Java - How to program
  Prentice Hall, 1997
  ISBN : 0-13-263401-5

Some JAVA References

- Mark Grand
  Java Language Reference
  ISBN : 1-56592-204-2
- Scott Oaks, Henry Wong
  Java Threads
  ISBN : 1-56592-216-6
- Elliotte Rusty Harold
  Java Network Programming
  ISBN : 1-56592-227-1

O'Reilly, 1997

A First Modula Example

```
(* Sample program to write "Hello World" *)
MODULE  HelloWorld ;
   FROM IO IMPORT WrStr; WrLn;
(* No data declarations in this module *)
BEGIN
   WrStr("Hello World !"); WrLn;
END HelloWorld.
```

A First Java Example

```
/* Sample program to write "Hello World" */
class HelloWorld {
   // No data declarations in this class
   public static void main(String[] arguments) {
      System.out.println("Hello World !");
   }
}
```

Java Comments

```
/* Sample program to write "Hello World" */
class HelloWorld {
   // No data declarations in this class
   public static void main(String[] arguments) {
      System.out.println("Hello World !");
   }
}
```

Three different formats for comments:
- C-style comments, delimited by /* and */
- C++ style single-line comments, starting with //
- *documentation comment, to extract comment into a machine-generated document*

Java blocks

```
/* Sample program to write "Hello
   World" */
class HelloWorld {
   // No data declarations in this class
   public static void main(String[]
   arguments) {
      System.out.println("Hello World !");
   }
}
```

Java programs (just as C and C++ programs) are structured with nested blocks delimited by { and }
For identifiers declared in a block, the same scope rules as those used in Pascal & Modula apply.

Object Oriented Programming

- **Fundamental idea : Abstract Data Type**
  - **What is an ADT ?**
    - **Data declarations**
    - **Procedures acting upon the data**
    - **Only the procedures can access the data.**
  - **Why ADTs ?**
    - **When data formats have to change, only procedures in ADT need to be adapted, no changes outside the ADT.**
  - **Implementation of ADTs ?**
    - **In modula 2, library modules are ADTs.**

Module Stack

```
MODULE Stack;
        EXPORT Push,Pop,Empty;
        CONST       Size = 30;
        TYPE        Task = RECORD Low,High : CARDINAL END;
        VAR         S : ARRAY [0..Size] OF Task;
                    t : [0 .. Size];
        PROCEDURE Push(a,b:CARDINAL);
        BEGIN       t := t + 1;       S[t].Low := a;     S[t].High := b;
        END Push;
        PROCEDURE Pop(VAR a,b:CARDINAL);
        BEGIN       a:=S[t].Low;      b:=S[t].High;      t:=t-1
        END Pop;
        PROCEDURE Empty():BOOLEAN;
        BEGIN       RETURN  t = 0
        END Empty;
BEGIN               t := 0
END Stack;
```

class

```
/* Sample program to write "Hello World" */
class HelloWorld {
    // No data declarations in this class
   public static void main(String[] arguments) {
       System.out.println("Hello World !");
   }
}
```

class ~ MODULE
Basic mechanism to build ADTs (objects) :
Set of data and procedures acting upon these data

class

- Java program = set of classes
- class =
  - – Data declarations
  - – Methods (= procedures)
- class = template to create similar objects
  - = extension of the notion of type:
    - Set of values the variable can take
    - implicit list of operations defined on that set
  - = Type of variables in object
  - + explicit definition of operations defined on variables
  - + initialization of these variables

class HelloWorld

```
/* Sample program to write "Hello World" */
class HelloWorld {
    // No data declarations in this class
   public static void main(String[] arguments) {
       System.out.println("Hello World !");
   }
}
```

No data declarations
One method : main

Method Definitions

```
/* Sample program to write "Hello World" */
class HelloWorld {
    // No data declarations in this class
   public static void main(String[] arguments) {
       // No data declarations in this method
       System.out.println("Hello World !");
   }
}
```

[protection][linkage][resulttype] name
([parameter declarations])
*{[declarations]*instructions*}*

Method Definitions

public static void main(String argv[])

[protection][linkage][resulttype] name*])* *([parameter declarations])*
*{[declarations]*instructions*}*

- *protection :*
  public : method can be called from anywhere
- *linkage :*
  static : method does not apply to a specific object instantiation
- *resulttype :*
  void : no value is returned by this method
  (in Java, C and C++ all procedures are functions !)
- name :
  main : this method will always be executed first

Method parameters

```
/* Sample program to write "Hello World" */
class HelloWorld {
    // No data declarations in this class
   public static void main(String[] arguments) {
       // No data declarations in this method
       System.out.println("Hello World !");
   }
}
```

Method parameters in C and Java are always passed by VALUE !
In C variable parameters are created by passing pointers by value.
Even when there are no parameters, the () need to be present.
The parameters of main are the arguments of the run command

Instructions

```
/* Sample program to write "Hello World" */
class HelloWorld {
    // No data declarations in this class
   public static void main(String[] arguments) {
       // No data declarations in this method
       System.out.println("Hello World !");
   }
}
```

Most instructions in a Java program are method activations
Method names are preceded by the name of the class they belong to
System.out.println is a method to write its parameter on the screen
Strings are groups of characters between " quotes "

Java Operators

- *Mixing of types in expressions is generally allowed*
- Comparison operators : <, <=, >, >=, ==, !=
- Arithmetic operators : +, -, *, /, %
- Increment, decrement operators : ++, --
- Compound operators : +=, -=, *=, /=, ...
- Boolean operators : && (AND) , || (OR) , ! (NOT)
- Bitwise logical operators : & (AND) , | (OR) , ^ (XOR)
- ...

Increment / Decrement Compound Assignment

- In Modula :

```
a := a + 1;     b := b - 1;
c := c + 5;     d := d / 2;
```

- In Java :

```
a++;            b --;
c += 5;         d /= 2;
```

Data declarations

- In Modula, all variables have to be declared
- In Java, all variables have to be declared
  - Variables can be initialized when declared.
  - Examples:
    - int a = 3;
    - int b = 8 * 5;
    - float pi = 3.0;
    - char key = 'c';
    - String Title = "Data declarations";
    - boolean EndOfList = true;

Constant declarations

- In Modula, named constants can be declared
  CONST pi = 3.1415926;
- In Java, a variable can be initialized to its final value and can not be modified further in the program.

  - Example:

```
final float pi = 3.1415926;
final String Author = "J.Tiberghien";
```

integer variables

- **MODULA integer types : Cardinal and Integer**
- **Modula integer operators : relational, +, -, *, DIV, MOD**
- **Java integer types :**
  - **byte : -128 .. +127; (8 bit)**
  - **short : -32768 .. +32767; (16 bit)**
  - **int : -2147483648 .. +2147483647; (32 bit)**
  - **long : -2^{63} .. $+2^{63} - 1$; (64 bit)**

integer variables

- Java integer operators :
  - + , - , * : same as in Modula.
  - / : same as DIV in Modula when both factors integer.
  - % : same as MOD in Modula.
  - ++ , --: increment and decrement unary operators.
  - *Bitwise logical operators*
  - *Comparison operators*

REAL variables

- **MODULA REAL type,**
- **Modula real operators : relational , +, -, *, /**
- **Java real types :**
  - **float : $1.4\ 10^{-45}$ .. $3.4\ 10^{+38}$ 32 bit, IEEE 754.**
  - **double : $4.9\ 10^{-324}$ .. $1.8\ 10^{+308}$ 64 bit, IEEE 754.**
- **Java real operators :**
  - **+, -, *, / : same as in Modula**
  - **++ , -- : increment and decrement unary operators.**
  - ***Comparison operators***

BOOLEAN variabeles

- **MODULA BOOLEAN type: *is an ordinal type***
  - **ORD(FALSE) = 0 ; ORD(TRUE) = 1**
- **Modula Boolean operators:**
  - **NOT, AND, OR**
  - **= # > < >= >= IN**
- **Java boolean type: *can not be cast into integer type***
- **Java Boolean operators**
  - **NOT : !**
  - **AND : &&**
  - **OR : ||**

Character variables

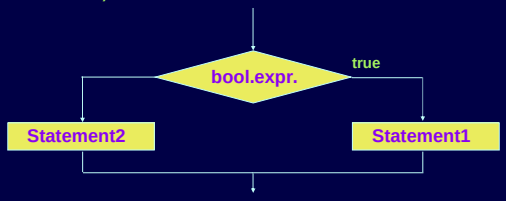
- **In MODULA:**
  - **CHAR type,**
  - **string: ARRAY[0..20] OF CHAR**
- **In Java:**
  - **char:**
    - **uses the 16 bit Unicode,**
    - **is considered as an always positive integer type**
    - **all operations allowed on integers are allowed**
  - **String :**
    - **is a specific class to represent strings.**
    - **a String is not an ARRAY OF CHAR.**
    - **String's can be concatenated by the + operator**

Structured variables

- **In MODULA:**
  - **ARRAY**
    - **A : ARRAY[1..3,-5..+5] OF REAL;**
    - **Matrix : ARRAY['A'..'J',1..10] OF REAL;**
  - **RECORD, SET**
- **In Java: Only arrays, Records are made by classes**
- **Array Declarations:**
  - **int [] a; or int a[]**
  - **int [][] a2;**
  - **int [][][] a3;**
  - **Arrays are created by the new operator.**
  - **Dimensions are defined when array is created**

if statement

If (*boolean expression*)
  statement1
else
  statement2
;



A Second Java Example

```
class NewRoot { // to compute the square root of the first argument
      public static void main (String[] arguments) {
            if (arguments.length > 0)
                  {  int number = 0;
                     number = Integer.parseInt(arguments[0]);
                     System.out.println("The Square root of "
                       + number
                       + " is "
                       + Math.sqrt(number));
                  }
            else  System.out.println("use: 'java NewRoot
   [argument]'");
      }
}
```

A Second Java Example

```
class NewRoot { // to compute the square root of the first argument
      public static void main (String[] arguments) {
            if (arguments.length > 0)
                  {  int number = 0;
                     number = Integer.parseInt(arguments[0]);
                     System.out.println("The Square root of "
                       + number
                       + " is "
                       + Math.sqrt(number));
                  }
            else  System.out.println("use: 'java NewRoot [argument]'");
      }
}
```

The formal parameter of the method main *is a one dimensional array of* String*'s called* arguments *containing the text typed after the name of the program in the run command*

A Second Java Example

```
class NewRoot { // to compute the square root of the first argument
      public static void main (String[] arguments) {
            if (arguments.length > 0)
                  {  int number = 0;
                     number = Integer.parseInt(arguments[0]);
                     System.out.println("The Square root of "
                       + number
                       + " is "
                       + Math.sqrt(number));
                  }
            else  System.out.println("use: 'java NewRoot [argument]'");
      }
}
```

length *is a variable associated with each* array arguments.length *gives the number of elements in the array* arguments

A Second Java Example

```
class NewRoot { // to compute the square root of the first argument
      public static void main (String[] arguments) {
            if (arguments.length > 0)
                  {  int number = 0;
                     number = Integer.parseInt(arguments[0]);
                     System.out.println("The Square root of "
                       + number
                       + " is "
                       + Math.sqrt(number));
                  }
            else  System.out.println("use: 'java NewRoot [argument]'");
      }
}
```

Wherever the syntax specifies a statement, a block, delimited by { and } can be substituted

A Second Java Example

```
class NewRoot { // to compute the square root of the first argument
      public static void main (String[] arguments) {
            if (arguments.length > 0)
                  {  int number = 0;
                     number = Integer.parseInt(arguments[0]);
                     System.out.println("The Square root of "
                       + number
                       + " is "
                       + Math.sqrt(number));
                  }
            else  System.out.println("use: 'java NewRoot [argument]'");
      }
}
```

Local variables can be declared in any block

A Second Java Example

```
class NewRoot { // to compute the square root of the first argument
      public static void main (String[] arguments) {
            if (arguments.length > 0)
                  {  int number = 0;
                     number = Integer.parseInt(arguments[0]);
                     System.out.println("The Square root of "
                       + number
                       + " is "
                       + Math.sqrt(number));
                  }
            else  System.out.println("use: 'java NewRoot [argument]'");
      }
}
```

ParseInt is a method of the class Integer
which parses a String into an integer value

A Second Java Example

```
class NewRoot { // to compute the square root of the first argument
      public static void main (String[] arguments) {
            if (arguments.length > 0)
                  {  int number = 0;
                     number = Integer.parseInt(arguments[0]);
                     System.out.println("The Square root of "
                       + number
                       + " is "
                       + Math.sqrt(number));
                  }
            else  System.out.println("use: 'java NewRoot [argument]'");
      }
}
```

sqrt is the method of the Math class
to compute square roots

A Second Java Example

```
class NewRoot { // to compute the square root of the first argument
      public static void main (String[] arguments) {
            if (arguments.length > 0)
                  {  int number = 0;
                     number = Integer.parseInt(arguments[0]);
                     System.out.println("The Square root of "
                       + number
                       + " is "
                       + Math.sqrt(number));
                  }
            else  System.out.println("use: 'java NewRoot [argument]'");
      }
}
```

The + operator for String's
Appends one String to another and
converts numerical values into a decimal representation

Iterations

- *While statement ~ Modula 2 WHILE loop*

```
while (Lives>0) { /* play game */ }
```

- *Do statement ~ Modula 2 REPEAT loop*

```
do { /* play game */ } while (Lives>0)
```

- *For statement ~ Modula 2 FOR loop*

```
for (int number =0; number < 100; number ++) {
  if ( number % 3 == 0 )
        System.out.println ("# " + number);
}
```

Exiting a loop

- *In Modula 2 :* EXIT
- *The Java* break *statement*

```
while (Lives>0) {
  /* play game */
  if (Cookiemonster.strikes) break;
  /* play game */
}
```

Skipping an iteration

- *No specific instruction in Modula 2*
- *The Java* continue *statement*

```
for ( int Year = 1600; Year < 2600; Year += 4 ) {
  if (( Year % 100 == 0 ) && ( Year % 400 != 0 ))
        continue;
  System.out.printl ( "The year " + Year
                      + " is a leapyear");
}
```

Strings example

```
class Strings {
    public static void main (String[] arguments) {
        int length = arguments.length;
        if (length > 0) {
            System.out.println(" you entered following words :");
            int max = 0;
            for(int l = 0; l < length; l++) {
                System.out.println(arguments[l].toUpperCase());
                if(arguments[l].length() > max) max = arguments[l].length();
                if(arguments[l].equals("stop")) break;
            }
            System.out.println("Longest word : "+ max + " characters ");
        }
        else  System.out.println("use: 'java Strings [string]'");
    }
}
```

Strings example

```
class Strings {
    public static void main (String[] arguments) {
        int length = arguments.length;
        if (length > 0) {
            System.out.println(" you entered following words :");
            int max = 0;
            for(int l = 0; l < length; l++) {
                System.out.println(arguments[l].toUpperCase());
                if(arguments[l].length() > max) max = arguments[l].length();
                if(arguments[l].equals("stop")) break;
            }
            System.out.println("Longest word : "+ max + " characters ");
        }
        else  System.out.println("use: 'java Strings [string]'");
    }
}
```

length *is a variable associated with each* array
arguments.length *gives the number of elements in the array* arguments

Strings example

```
class Strings {
    public static void main (String[] arguments) {
        int length = arguments.length;
        if (length > 0) {
            System.out.println(" you entered following words :");
            int max = 0;
            for(int l = 0; l < length; l++) {
                System.out.println(arguments[l].toUpperCase());
                if(arguments[l].length() > max) max = arguments[l].length();
                if(arguments[l].equals("stop")) break;
            }
            System.out.println("Longest word : "+ max + " characters ");
        }
        else  System.out.println("use: 'java Strings [string]'");
    }
}
```

toUpperCase *is a method which belongs to the class* String
arguments[l] *is a* String

Strings example

```
class Strings {
    public static void main (String[] arguments) {
        int length = arguments.length;
        if (length > 0) {
            System.out.println(" you entered following words :");
            int max = 0;
            for(int l = 0; l < length; l++) {
                System.out.println(arguments[l].toUpperCase());
                if(arguments[l].length() > max) max = arguments[l].length();
                if(arguments[l].equals("stop")) break;
            }
            System.out.println("Longest word : "+ max + " characters ");
        }
        else  System.out.println("use: 'java Strings [string]'");
    }
}
```

length *is a method which belongs to the class* String
arguments[l] *is a* String
arguments[l].length *gives the number of characters in the string* arguments[l]

Strings example

```
class Strings {
    public static void main (String[] arguments) {
        int length = arguments.length;
        if (length > 0) {
            System.out.println(" you entered following words :");
            int max = 0;
            for(int l = 0; l < length; l++) {
                System.out.println(arguments[l].toUpperCase());
                if(arguments[l].length() > max) max = arguments[l].length();
                if(arguments[l].equals("stop")) break;
            }
            System.out.println("Longest word : "+ max + " characters ");
        }
        else  System.out.println("use: 'java Strings [string]'");
    }
}
```

equals *is a method which belongs to the class* String
arguments[l].equals("stop") *is true if*
arguments[l] *has the value* "stop"

Strings example

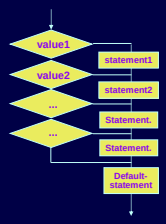
```
class Strings {
    public static void main (String[] arguments) {
        int length = arguments.length;
        if (length > 0) {
            System.out.println(" you entered following words :");
            int max = 0;
            for(int l = 0; l < length; l++) {
                System.out.println(arguments[l].toUpperCase());
                if(arguments[l].length() > max) max = arguments[l].length();
                if(arguments[l].equals("stop")) break;
            }
            System.out.println("Longest word : "+ max + " characters ");
        }
        else  System.out.println("use: 'java Strings [string]'");
    }
}
```

This break *statement causes immediate termination of the* for *loop*

The switch Statement

- Similar to the Modula 2 CASE statement, but with fundamentally different semantics

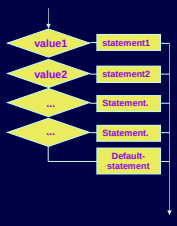
- switch(*selector*) {
  case *value1* : *statement1*;
  case *value2* : *statement2*;
  ...
  default : *defaultstatement*;
  }



The switch Statement

- To make it equivalent to the Modula 2 CASE statement, break statements are required

- switch(*selector*) {
  case *value1* : {*statement1*;break}
  case *value2* : {*statement2*;break}
  ...
  default : *defaultstatement*;
  }



Objects in Modula 2

- One object = One module
- Objects have
  - Attributes
    - thinghs that describe the object
    - things that show how it is different from others
    - The variables
  - Behaviour
    - what an object does
    - The PROCEDURES
- Objects are created at compile time
- Objects tend to be large, complex and few

Objects in Java

- Derived from a class (=template for a certain class of objects)
- Objects have
  - Attributes
    - thinghs that describe the object
    - things that show how it is different from others
    - The variables
  - Behaviour
    - what an object does
    - The methods
- Objects are created by the new constructor
- Objects tend to be small, simple and many

Static vs. Dynamic Variables

Modula 2	Java
• Static variables	• Static Variables
– All ordinal types	– byte, short, int, long
– Reals	– float, double
– Arrays	– boolean
– Records	– char
– Sets	= *Built-in types*
– Procedures	
– Pointers	
• Dynamic variables	• Dynamic objects
– Defined by TYPE definition (Record + Pointer)	– Defined by Class definition (any class)
– Created by NEW	– Created by new
– Accessed via pointers	– Accessed via methods
– Deleted by DISPOSE	– Deleted by garbage collector

The Dog Class

```
Public class Dog {
   String Color;                 //This is an attribute
   public void speak() {         //This is behaviour
      System.out.println(" Wouw!,Wouw! ");
   }
   public void sleep() {         //This is more behaviour
      System.out.println(" zzzzzzzzzzzzzzzz");
   }
}
// Constructing two Dogs :
Dog Bobby = new Dog();
Dog Pluto = new Dog();
```

The Cat Class

```
Public class Cat {
    String Color;                   //This is an attribute
    public void speak() {           //This is behavoiur
        System.out.println(" Miauw!,Miauw! ");
    }
    public void sleep() {           //This is more behaviour
        System.out.println(" zzzzzzzzzzzzzzzz");
    }
}
// Constructing two Cats :
Cat Tom = new Cat();
Cat Garfield = new Cat();
```

The Pet class

```
Public class Pet {
    String Color;                   //This is a common attribute
    public void sleep() {           //This is a common behaviour
        System.out.println(" zzzzzzzzzzzzzzzz");
    }
}
Public class Dog extends Pet{
    public void speak() {           //This is a Dog's specific behaviour
        System.out.println(" Wouw!,Wouw! ");
    }
}
Public class Cat extends Pet{
    public void speak() {           //This is a Cat's specific behaviour
        System.out.println(" Miauw!,Miauw! ");
    }
}
```

Playing with Cats and Dogs

```
Garfield.Color = "Brown tiger";
Tom.Color = "Grey";
Bobby.Color = "White";
Pluto.Color = "Orange";
…
Bobby.sleep(); Tom.speak(); Pluto.speak(); Garfield.sleep();
```

zzzzzzzzzzzzzzzz
Miauw!,Miauw!
Wouw!,Wouw!
zzzzzzzzzzzzzzzz

Inheritance

- Avoids redundant work when creating related classes
- Subclass: Cat and Dog are subclasses of the class Pet
- Superclass: Pet is the superclass of classes Cat and Dog
- The whole Java language is a tree of classes
- Multiple inheritance is NOT allowed in Java

The Die class

```
public class Die{
    public int value;
    // constructor of class Die
    public Die(){ value = 0; }
    // executed when an object of class Die is constructed
    public void RollDie() {
        double tempvalue = Math.random() * 6;
        value = (int) Math.floor(tempvalue) + 1;
    }
}
```

The Die module

```
MODULE  Die;
  FROM Lib IMPORT RANDOM;
  EXPORT Value, RollDie;
  VAR value : CARDINAL;
  PROCEDURE RollDie;
  BEGIN
    Value = RANDOM(5)+1
  END RollDie;
BEGIN
  Value := 0
END Die;
```

The Die module

```
MODULE  Die1;
  FROM Lib IMPORT RANDOM;
  EXPORT Value1, RollDie;
  VAR value1 : CARDINAL;
  PROCEDURE RollDie1;
  BEGIN
    Value1 = RANDOM(5)+1
  END RollDie1;
BEGIN
  Value1 := 0
END Die1;
```

The Die module

```
MODULE  Die2;
  FROM Lib IMPORT RANDOM;
  EXPORT Value2, RollDie;
  VAR value2 : CARDINAL;
  PROCEDURE RollDie2;
  BEGIN
    Value2 = RANDOM(5)+1
  END RollDie2;
BEGIN
  Value2 := 0
END Die2;
```

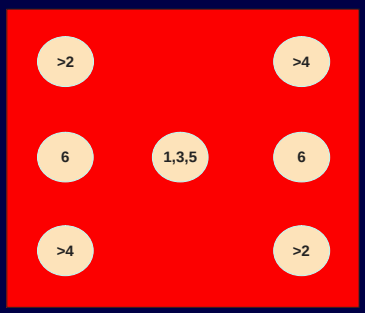
Test the Die

```
class TestDie {
        public static void main (String[] arguments) {
                Die Die1 = new Die();
                Die Die2 = new Die();
                System.out.println("Initially,        first die:  "
                  + Die1.Value +", Second die: " + Die2.Value);
                Die1.RollDie();
                Die2.RollDie();
                System.out.println("After rolling, first die:  "
                  + Die1.Value +", Second die: " + Die2.Value);
        }
}
```

The graphical Die class (1)

```
import java.awt.*;
public class DieG extends Die{
      public void DrawDie(Graphics screen, int x, int y) {
        // Draw a red die with black border
                screen.setColor(Color.red);
                screen.fillRoundRect(x, y, 100, 100, 20, 20);
                screen.setColor(Color.black);
                screen.drawRoundRect(x, y, 100, 100, 20, 20);
        // Draw white dots according to value
                (see next 2 slides)
      }
}
```

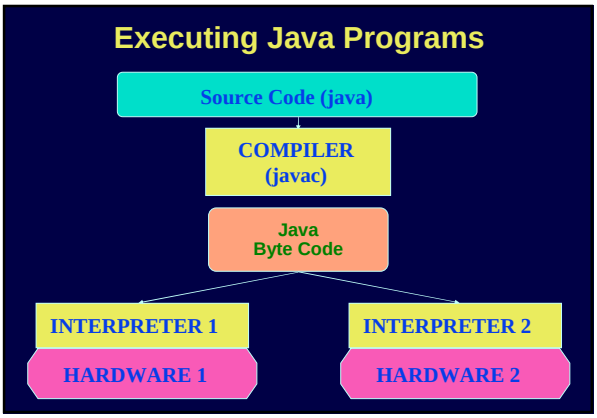
The graphical Die class (2)



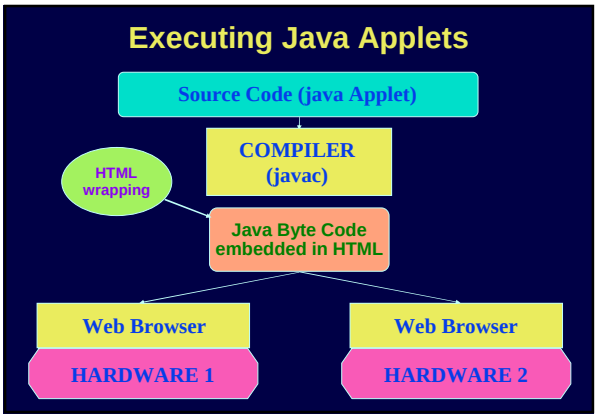
The graphical Die class (3)

```
// Draw white dots according to value
screen.setColor(Color.white);
if (Value > 1)          {screen.fillOval(x+5, y+5, 20, 20);
                          screen.fillOval(x+75, y+75, 20, 20); }
if (Value > 3)          {screen.fillOval(x+75, y+5, 20, 20);
                          screen.fillOval(x+5, y+75, 20, 20);   }
if (Value == 6)         {screen.fillOval(x+5, y+40, 20, 20);
                    screen.fillOval(x+75, y+40, 20, 20); }
if (Value % 2 == 1)   {screen.fillOval(x+40, y+40, 20, 20);}
```

Executing Java Programs



Executing Java Applets



DieApplet, version 0

```
import java.awt.*;
public class DieApplet0 extends java.applet.Applet {
    // create variables
      DieG Die1 = new DieG(); DieG Die2 = new DieG();
    // Initialize program
      public void init() {
         setBackground(Color.green);
      }
    // Display results
      public void paint(Graphics Screen) {
         Die1.DrawDie(Screen,5,50);
         Die2.DrawDie(Screen,175,50);
      }
}
```

Calling an applet in HTML

```
<html>
  <head>
    <title> Dies applet </title>
  </head/
  <body>
    <applet code = "DieApplet0.class"  width=285 height=250>
    </applet>
  </body>
</html>
```

DieApplet, version 1

```
import java.awt.*;
public class DieApplet1 extends DieApplet0 {
    // create variables : done in superclass

    // Initialize program : done in superclass

    // Display results :
        extends the paint method of the superclass
      public void paint(Graphics Screen) {
         Die1.RollDie(); Die2.RollDie();
         super.paint(Screen);
      }
}
```

DieApplet, version 2

```
import java.awt.*;
public class DieApplet2 extends DieApplet0 {
    // create variables : done in superclass

    // Initialize program :
        extends the paint method of the superclass
        public void init() {
            Die1.RollDie(); Die2.RollDie();
            super.init();
        }

    // Display results : done in superclass
}
```

Interactive DieApplet

```
import java.awt.*;
import java.awt.event.*;
public class DieApplet3 extends DieApplet implements ActionListener {
      Button rollButton = new Button("Roll Dice");
      public void init() {
         super.init();
         rollButton.addActionListener(this);
         add(rollButton);
      }
      public void actionPerformed(ActionEvent event) {
         Die1.RollDie(); Die2.RollDie();
         repaint();
      }
}
```