Java Programming

Topic 1

Introduction to Java

1

Java

- Java is an object-oriented, robust, secure, architecture-neutral language with built in capabilities for distribution and execution via world-wide web browsers

2

Java

- A pure object-oriented language
- Syntax resembles C++
- Structure and design reflects Smalltalk

3

Java and Objects

- Everything in Java is an object
  - unlike C++ which can mix objects and procedural constructs
- Many C and C++ constructs are eliminated
  - pointers and pointer arithmetic
  - structs
  - typedefs
  - preprocessor directives (#define)
  - malloc and free

4

Java is Robust

- extensive type checking
- true arrays with array bounds checking
- null pointer checking
- automatic garbage collection

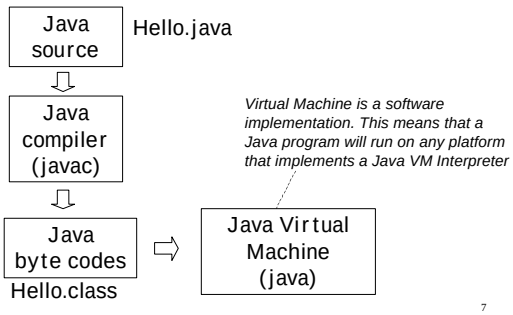
5

Java vs C/C++

- No manual memory allocation
- True arrays; no pointer arithmetic
- if (x = 3) …
  - not allowed
- No multiple inheritance
  - interfaces introduced

6

Java



7

Java Evolution

- Java 1.0
  - OLD - but still around
  - Supported by JDK 1.0.X
- Java 1.1
  - Replaces Java 1.0
  - Uses the new Event Model - very significant
  - Supported by JDK 1.1.X
- Java 1.2
  - known as Java 2; JDK now called SDK

8

What is the JDK/SDK?

- Available from textbook CD ROM or (http://www.javasoft.com)
- The JDK contains:
  - **Java Compiler (javac)**
  - **Java Virtual Machine (java)**
  - **Java Class Libraries**
  - **Java AppletViewer**
  - **Java Debugger**
  - **Javadoc - for documentation**

9

Java Development

Text editors - good for small programs

- Textpad (www.textpad.com)
- Notepad (comes with Windows)

IDEs:

- IBM Visual Age for Java
- Symantec Visual Café
- Imprise: Jbuilder

10

JDK 1.2 Installation (Windows)

- C:\jdk1.2.2 ⇦ Must be on your CLASSPATH
  - docs
  - bin ⇦ Must be on your PATH
  - demo
  - include
  - lib
- javac - the Java compiler
- java - the Java runtime

11

Autoexec.Bat

- Should contain something like the following:

```
set path=%path%;c:\jdk1.2.2\bin
set classpath=
  c:\jdk1.2.2;.;c:\corejava
```

12

Applications vs Applets

- Applications are stand alone
  - compile with javac Welcome.java
  - run with java Welcome
- Applets require a browser or appletviewer
  - compile with javac
  - run via HTML file

13

Applet

This HTML file should
contains a reference to a Java applet

```
<HTML>
<HEAD>
<TITLE> A Simple Program </TITLE>
</HEAD>
<BODY>

<APPLET CODE="HelloWorld.class" WIDTH=150
        HEIGHT=25>
</APPLET>
</BODY>
</HTML>
```

14

Java Programming Fundamentals

15

Overview

- Data types and variables
- Assignment and initialization
- Operators
- Strings
- Class methods
- Arrays

16

Java Applications vs Applets

- Java Applications
  - Any platform with a Java VM interpreter can run a Java program just as one can run a Fortran, C or Cobol program
- Java Applets
  - Designed specifically to be loaded and run BY a Web Browser

17

Java Application

- Requires a main() method
- Cannot have a return statement
- May include System.exit()
  - action taken with value returned is system dependent
  - abruptly terminates the running program including all threads

18

Hello Program 0

```
public class Hello {
  public static void main (String [ ] args ) {
      System.out.println("Hello World");
  }
}
```

19

Hello Program 0

```
public class Hello {
  public static void main (String [ ] args ) {
      System.out.println("Hello World");
  }
}
```

java is case-sensitive

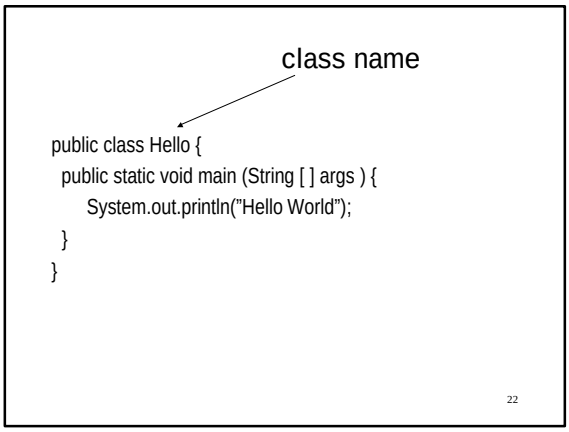
20

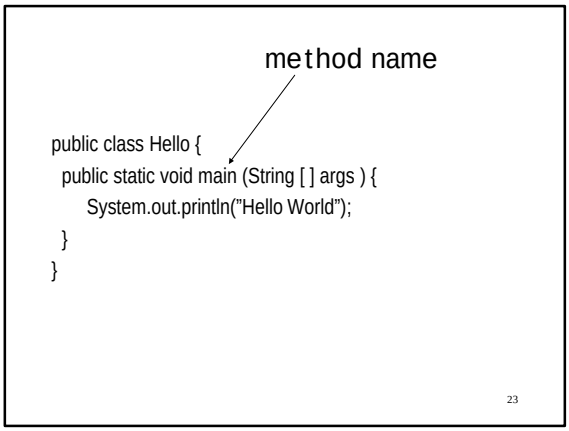
Hello Program 0

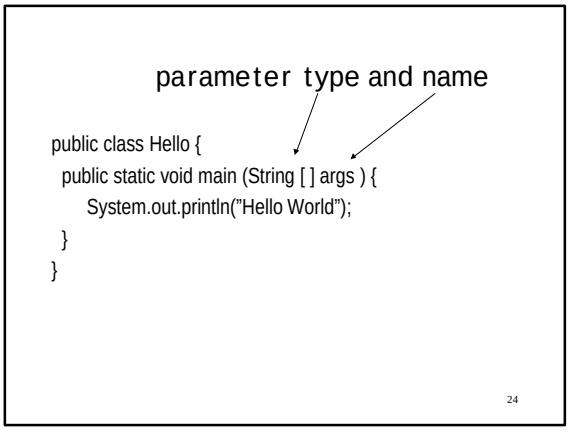
```
public class Hello {
  public static void main (String [ ] args ) {
      System.out.println("Hello World");
  }
}
```

everything in java is inside a class

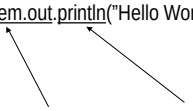
21


class name
public class Hello {
public static void main (String [ ] args) {
System.out.println("Hello World");
}
}
22


method name
public class Hello {
public static void main (String [ ] args) {
System.out.println("Hello World");
}
}
23


parameter type and name
public class Hello {
public static void main (String [ ] args) {
System.out.println("Hello World");
}
}
24

```
public class Hello {
  public static void main (String [ ] args ) {
      System.out.println("Hello World");
  }
}
```



object and method-name
(we ask the System.out object to execute the method println)

25

Comments

- Standard C style
  `/* ….until . ….*/`
- End of line
  `// … until end of line`
- java doc comments
  `/** … until */`

26

Identifiers

- Used to name variables, methods, classes and/or labels
- Must begin with letter, $ or _ (underscore)
- Identifiers are case sensitive

```
foo         //legal
$hello      //legal
_bar        //legal

3k          //illegal
```

27

Java Data Types

28

Java is Strongly Typed

- Every variable must have a declared type
- Java has eight primitive types
  - signed integer types (4)
  - char
  - floating point types (2)
  - boolean
- Classes are also types

29

Java Integer Data Types

Signed		min	max
• byte	8-bit	-2^7	$2^7 - 1$
• short	16-bit	-2^{15}	$2^{15} - 1$
• int	32-bit	-2^{31}	$2^{31} - 1$
• long	64-bit	-2^{63}	$2^{63} - 1$
Unsigned			
• char	16-bit	0	$2^{16} - 1$

30

Integral Literals

- 33
- 054
- 0x1c //hex
- 0x1C
- 0X1c
- 5000000000L

int literals must be in the range: -2,147,483,648 to 2,147,483,647

integral literals default to 32 bit values -- append L for 64-bit values

31

char

- Based on the Unicode character scheme: uses 2-byte codes
- When the upper 9 bits are 0, the code is equivalent to 7-bit ASCII
- Assigning char values:

```
char c = 'a';
char c3 = 45;
```

32

char

- For full unicode use: \uXXXX where XXXX is a hex number

```
char c2 = '\u00DE';  // hex value
```

- Special characters:
  - \b backspace \u0008
  - \t tab \u0009
  - \r return \u000d
- Example:

```
char c = '\t';
```

33

Java Floating Point Data Types

- float (IEEE 754-1885) 32-bit floating point
- double (IEEE 754-1885) 64-bit floating point

```
Double.MIN_VALUE = 5e-324;
Double.MAX_VALUE =
 1.7976931348623157e+308;
```

34

Floating Point Literals

- 1.414
- 4.23E+21 or 4.23e+21
- 1.29F or 1.29f
  - floating point literal
- 2.334D or 2.334d
  - double literal
- Default is 64-bit double (e.g. 1.414)

35

Java Boolean Data Type

- Boolean has two values: true and false
- boolean isBig = true;
- boolean isSmall = false;

36

Variables - Types and Naming

```
int   myCounter;
byte  b;  //comment
double grossNatProd;
long  longVar88, myLong;
```

variable names begin with a letter, $ or _ followed by letters or digits or any unicode letter character.

37

Assignment and Initialization

```
double d;  // declare
d = 44.494;// initialize

double d = 44.494;   // combine
```

38

Conversion of Numeric Types

- Java automatically converts a numeric expression the highest precision type of any components
- double x = 12.22;
- float f = 88.987f;
- (x*f) <-- double expression

conversion is automatic when there is no chance of loss of information

39

Casting

- Forcing a conversion from one data type to another
- Java requires casting to affirm you are willing to lose precision
  - double x = 9.9876
  - int k = (int)x; //what is k?
- Or..
  - int m = (int)Math.round(x);
    - round returns a long

40

Auto Casting

- byte -> short -> int -> long -> float -> double
- char -> int

Can always assign a variable of a type on the left to a type on the right without an explicit cast

41

Cast Examples...

```
char c = 'a';
short s = (short)c;
byte b = (byte)c;
```

stores the value 97 (ascii value of 'a')

42

Constants
use keyword final

- A final variable cannot be changed
- A final class cannot be subclassed
- public final class Math {
  public static final double PI = 3.14159…;
  public static final double E = …;
- No C style constants in java

43

Operators

44

Arithmetic Operators

- arithmetic: + - * /
  / does integer division if both operands are integers (truncation not rounding)
  / does floating point division if one or more operands is non-integer
- shortcut:
  k +=4 (same as k = k + 4)
  j *=5 (same as j = j * 5)

45

+

- If one operand is a string, non-string operands are converted to string and the result is the concatenation of the two
- Conversion occurs by invoking the toString() method

```
k = 34.3;
System.out.println("k= " + k);
```

⇩

k= 34.3

46

exponentiation

- No built-in operator
- Use the pow method from class Math
- double z = Math.pow(x, c)
  - computes $z = x^c$
  - both arguments must be double

47

modulo operator (for remainder)

- 15 % 2 --> 1
  - the remainder when 2 divides 15
- 14 % 3 --> 2
- 20.5 % 3 --> 2.5

works with integer and floating point

- j %=2 --> j = j % 2

48

increment and decrement operators

- k++
  - adds 1 to k; same as k = k + 1
- j--
  - subtracts 1 from j; same as j = j - 1
- Also: ++k and --j
- Difference relates to appearance in expressions

49

k++ vs. ++k

- k = 2;
- System.out.println(2*(k++));
  - output: 4 and k is now 3

Avoid using ++ or -- inside expressions. Source of errors.

- k = 2;
- System.out.println(2*(++k));
  - output: 6 and k is now 3

50

Relational and Boolean Operators

- Equality Test with ==
  (4 == 2) is false
- Inequality
  (4 != 2) is true
- Relational
  > < >= <=
  (4 < 2) is false

51

Gotcha

- double d = 0.0 / 0.0;
- **if (d == Double.NaN)** ⇨ false
  System.out.println("NaN");

USE:

- **if (Double.isNaN(d))** ⇨ true
  System.out.println("NaN");

52

Logical Operators

- Logical AND: &&
  ((x != 0) && ((y/x) > 2))

  short circuit evaluation -- if one expression is false, the others not evaluated
- Logical OR: ||
  (x <= y) || (x <= z)

53

Bitwise Operators

For integer types

& and
| or
^ xor
~ not

```
int foo = 12
int zz = (foo & 4) / 4

zz is 1 if foo has a 1 in third bit
position from the right

foo = 000…1100  (binary 12)
4   = 000…0100  (binary 4)
      -----------
      000…0100  (foo & 4)
```

54

Bitshift Operators <<, >>, and >>>

- For integer types
- >> *signed* bitshift right
  zeros OR ones added on the left depending on sign
- << bitshift left
  zeros added on the right
- >>> bitshift right
  zeros added on the right

55

<<

- int k = 21;
- int j = k << 2;
- k =

00000000 00000000 00000000 00010101 = 21

- j =

00000000 00000000 00000000 01010100 = 84

56

>>

- int k = 21;
- int j = k >> 2;
- k =

00000000 00000000 00000000 00010101 = 21

- j =

00000000 00000000 00000000 00000101 = 5
zeros moved in from right

57

>>

- int k = -192;
- int j = k >> 1;
- k =

11111111 11111111 11111111 01000000 = -191

- j =

⇨ 11111111 11111111 11111111 00100000 = -96
ones moved in from right when number is negative

58

>>>

- int k = -192;
- int j = k >>> 1;
- k =

11111111 11111111 11111111 01000000 = -191

- j =

⇨ 01111111 11111111 11111111 00100000 = 2147483552
zero always moved in from right

59

Logical And &

```
  1001 1010
& 0110 1001
-----------
  0000 1000
```

c = a & b;

both bits must be ONE for the result to be ONE

60

Logical Or

|

```
  1001 1010
| 0110 1001
-------------
  1111 1011
```

c = a | b;

if either bit is ONE, the result is ONE

61

Logical XOr

^

```
  1001 1010
^ 0110 1001
-------------
  1111 0011
```

c = a ^ b;

if either bit is ONE, the result is ONE

62

Logical NOT

~

```
1001 1010
    ⇩
0110 0101
```

c = ~a;

ONES go to ZEROS; ZEROS go to ONE

63

Strings (and StringBuffer)

64

Strings

- Not primitive but treated special
- "hello" is a String constant:

  System.out.println("hello" + " world");

  + is string concatentation
- String(capital 'S') is a class

```
String s = "hello world";
String s2 = "hello " + "folks";
```

65

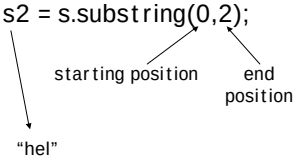
Strings

- NOT an array of characters (like C/C++)
- Strings are immutable
  - "hello world"
    - creates a String instance object

66

Substrings

- Use substring method of String class
- String s = “hello”;
- String s2 = s.substring(0,2);



67

String Editing

- String length: use the length method

```
String s = “tomato”;
s.length();          // returns 6
```

- You cannot change characters inside a string - you can only create a new string constant and adjust the reference

```
String s = “Java Server”;
s = s.substring(0,8) + “let”;
```

68

String Equality

- Use the string equals method NOT ==
- String s = “Zorro”;
- s.equals(“Zorro”); // true
- “Zorro”.equals(s); // true
- “zorro”.equalsIgnoreCase(s); // true

69

String Equality beware ==

- s = “Zorro”;
- (“Zorro” == s)
  - true when the two identical strings are stored in same memory location -- usually true
- s.substring(0,2) == “Zor”
  - probably false

Don’t use == for string comparison

70

String (instance) Methods

- int length()
- char charAt(int index)
- boolean equals(Object obj)
- boolean equalsIgnoreCase(String s)
- int indexOf(char c)
- String subString(int beginIdx, int endIdx)

71

String static methods

- String valueOf(Object obj)
- String valueOf(char [] data)
- String valueOf(char c)
- String valueOf(int i)
- String valueOf(long l)
- String valueOf(float f)
- String valueOf(double d)

72

StringBuffer

- Contents can be modified
- Grows in length as needed
- Can modify in place with:
  - setCharAt()
  - append()
  - insert()
- Convert to string with
  - toString()

73

Control Flow

74

Blocks {...}

- Java Block - statement(s) inside curly braces {..}
- Blocks define the scope of variables

```
{ int k;
    k = 3;
  ..
}
// k not defined outside block
```

75

Blocks...

- Can't define same variable in nested blocks

```
{int k;
  { int j;
    int k;   // illegal
  }
}
```

76

Conditional Statements

- if (*condition*) *statement;*
- if (*condition*) *block;*

```
if (x <= y) {
  j = 23;
  y = x % 10;
}
```

77

if..then..else

```
if (x <= y) {
  j = 23;
  y = x % 10;
}
else {
  j = 100;
  y = x % 4;
}
```

78

if..then..else

```
if (x <= y) {
  j = 23;
  y = x % 10;
}
else if (x < 20) {
  j = 100;
  y = x % 4;
}
else if (x < 40) {
  j += 20;
  y %= 4;
}
```

79

ternary operator

- condition ? e1 : e2
- if condition is true, evaluates to e1, else e2

```
int x, y = 2, z = 3;
x = (y > z) ? 1 : 2;  // x is 2
```

80

while loop

- while (condition) { block }

```
while ( x < target ) {
  System.out.println(x, target);
  x++;
}
```

81

do - while loop

- do { block }
  while (condition)

```
do {
  System.out.println(x, target);
  x++;
} while ( x < target )
```

82

for loop

for (initialization; termination; increment)
{block}

```
for (int i=0; i< 10; i++) {
  System.out.println("number is " + i);
}
```

note that the variable i only has scope inside the loop

83

switch statement

```
switch (month) {
 case 1:  System.out.println("January");
          break;
 case 2:  System.out.println("February");
          break;
 case 3:  System.out.println("March");
          break;
 default: System.out.println("other");
}
```

- can only switch on char or integers (except long)
- execution stops at break or end of switch
- default is optional

84

unlabeled breaks

- used to break out of for, while, do or switch statements

```
while (x < y) {
  z = z + x;
  if (z > max) break;
  x++;
}
```

85

labeled breaks

```
test:
for (int j=0; j<10; j++) {
  for (int k=0; k<10; k++) {
    System.out.println(i, k);
    if ((i==4) && (k==7)) break test;
  }
}
```

breaks out of statement associated with label

86

return

Exits from the current method

```
return x;
```

The value returned by return must match the type of method's declared return value.

When a method is declared **void** use the form of return that doesn't return a value:

```
return;
```

87

Class Methods

88

User defined functions

- Functions or subroutines are known as methods in OOP
- All methods must be defined as part of some class
- In Java methods are either:
  - class methods (with keyword static)
  - instances methods

89

the class Math

(and its static methods)

90

```
public class Math {
 //class static constants
 public static final double E;
 public static final double PI;

// class static methods
 public static int abs (int a);
 public static int abs (float a);
 public static double sin (double a);
 public static double cos (double a);

 public static double sqrt (double a);
 public static double pow
              (double a, double b);
 public static double random ( );
…
```

91

Using static methods and constants

- Must use the class name:

```
area   = Math.PI * radius * radius;
stdDev = Math.sqrt(variance);
side   = Math.cos(x) * hypotenuse;
```

92

Math.random( )

- **public static double random()**
  Returns a random number (double) between 0.0 and 1.0.
  Actually pseudorandom - the numbers repeat over time.
- For random integer:
  multiply and cast

```
int  n  =  (int)(Math.random() * 10);
```

  – yields an integer between 0 ..9 inclusive
  – (int) double --> rounds toward zero

```
(int)9.99 is 9;  (int)9.1 is 9;
(int)-9.99 is -9
```

93

Arrays

94

Arrays

- Arrays are Java objects
- You must
  - Declare
  - Construct (Allocate)
  - Initialize
- Cannot be allocated in place as in C/C++

95

Array Declaration

96

Array Declaration

- Declaration tells the compiler the name and type of the array
- int k[];
- double x[];
- Point loc[];
- float twoDim[] [];

primitive

object

primitive - two dimensional

97

Array Construction / Allocation

- Array sizing is done at run-time

```
int k[];
k = new int[35];

float xBog[];
int size = 200;
xBog = new float[size];
```

variables OK

98

Array Idioms

Declare & Allocate

- `int [] scores = new int[20];`
- `float [] ff = new float[100];`
- `boolean bob[] = new boolean[10];`

Arrays automatically initialized when created:

integer arrays: 0
floating point: 0.0
boolean: false

99

Array Initializing

Declare & Allocate & Initialize

```
int [ ] scores = {1, 2, 3+5, 7};
```

Is this legal?

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

100

Array Initializing

Declare & Allocate & Initialize

```
int [ ] scores = {1, 2, 3+5, 7};
```

Is this legal?

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

No! Cannot specify dimension in declaration

101

Arrays

- Arrays begin at index 0
- Arrays are always checked for bounds correctness
  - ArrayIndexOutOfBounds exception will be thrown
  - **`scores[j] = 34;`**

program will abort if j is not valid index

102

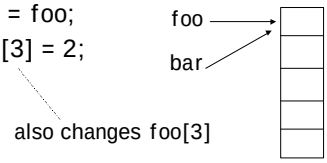
Looping and Arrays

```
for (int i=0; i< scores.length; i++) {
  System.out.print(scores[i] );
}
```

103

copying arrays

- int foo [] = new int [5];
- int bar [];
- bar = foo;
- bar[3] = 2;

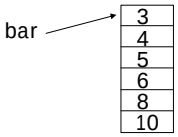


104

True Array Copy

- System.arraycopy(from, fromIdx,
  to, toIdx, count);

```
int foo [] = {1,2,3,4,5};
int bar [] = {0,2,4,6,8,10};
System.arraycopy(foo, 2, bar, 0, 3);
```

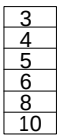


105

Arrays as parameters

- Passing an array as parameter has potential to change the array values
- Arrays class has sort method

```
int foo [] = {10,8,6,5,3,4};
Arrays.sort(foo);
```



106

Array as return value

- Useful when you want to return several values

```
public static int [] drawing
             (int high, int count);
```

107

Multidimensional Arrays

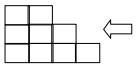
- An array of array

```
double table [] [];
table = new double [10] [5];
…
table [2][3] = 12.22;  // assignment
```

108

Multidimensional Arrays

- Initialize
- int foo [] [] = { {2,3}, {100, 200} };
- int bar [] [] = { {1,2}, {4,5,6}, {2,1,2,1}};



can create asymmetrical arrays

109

Hello Program 1

```
public class Hello {
  public static void main (String [ ] args ) {

      System.out.println("Hello World");
      System.exit(0); // not required
  }
}
```

Interpretation left up to the Operating System

110

Hello Program 2

```
public class Hello {
  public static void main (String [ ] args ) {
    for (int i=0; i<args.length; i++) {
      System.out.println( args[i] );
    }
  }
}
```

111

Hello Program 3

```
public class Hello {
  public static void main (String [ ] args ) {
    for (int i=0; i<args.length; i++) {
      System.out.println("args[" + i + "] = " + args[i]);
    }
  }
}
```

112

import

- To use the classes of any package (except Java.lang) you must import the packages
- Option:
  - import java.net.Socket;
- OR
  - import java.net.*;

113

Classpath

- Java knows where to look to find system classes
- The Classpath variable is used to tell java where to look for user classes

```
set CLASSPATH=.;C:\joe\apps;D:\myjava
```


current directory

114

String Conversion

115

Data Conversion String to Primitive

- Each primitive data type has a corresponding Wrapper class

Classes

- Integer
- Float
- Double
- Byte
- Long
- Short
- Char
- Boolean

⇦ each class has a parse utility method

116

Strings to Primitives (Numbers)

- byte b = Byte. parseByte(“21”);
- double d = Double.parseDouble(“21.3”);
- int k = Integer.parseInt(“344”);

trying to convert an improper string will throw a NumberFormatException -- your program will abort

```
int t = Integer.parseInt(“44.4”); // oops!
```

117

Data Conversion Primitive to String

- String class - static methods
  - String.valueOf(int)
  - String.valueOf(double)

also in String class:
valueOf(boolean)
valueOf(char)
valueOf(char [])
valueOf(long)

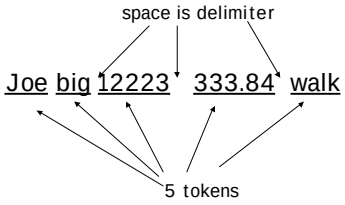
```
String s = String.valueOf(22);
String t = String.valueOf(44.4);
```

118

String Tokenizer

119

StringTokenizer class



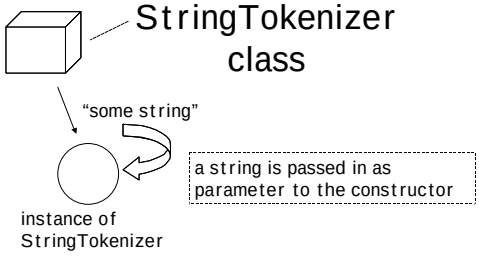
120

StringTokenizer class

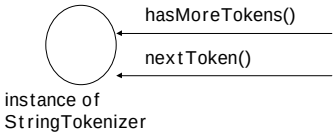
- Breaks a string into "tokens" based on delimiters which can be:
  - defined at object creation time
  - defined on a per token basis
- The "tokens" are returned as strings
- Optionally the delimiter can be returned

121

StringTokenizer class



122



123

StringTokenizer Example

```
StringTokenizer st =
 new StringTokenizer("this is  test");

while (st.hasMoreTokens() ) {
  System.out.println(st.nextToken() );
}
```

Output:
this
is
a
test

124

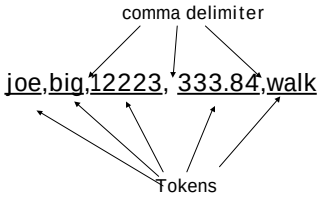
CONSTRUCTOR

public StringTokenizer (String str)

- Constructs a tokenizer for str
- The default delimiter characters are:
  - "\t\n\r"
  - space
  - tab
  - newline
  - carriage return

125

other delimiters?



126

public StringTokenizer
(String str,
String delim, → ","
boolean returnTokens)

CONSTRUCTOR

- Constructs a tokenizer for str
- The delimiter characters are in delim
- The delimiters are returned (as string of length 1) if returnTokens is true

127

public StringTokenizer
(String str,
String delim)

CONSTRUCTOR

- Constructs a tokenizer for str
- The delimiter characters are in delim
- The delimiters are not returned

128

comma delimiter

```
String s = "hello, there joe";
StringTokenizer ss =
     new StringTokenizer(s, ",");

while (ss.hasMoreTokens() ) {
 System.out.println(
             ss.nextToken() );
}
```

⇨ hello
there joe

129

METHOD

public boolean hasMoreTokens ()

- Returns true if more tokens remain

130

METHOD

public String nextToken ()

- Returns next token from the StringTokenizer
- Throws
  - NoSuchElementException if no more tokens

131

METHOD

public String nextToken (String delim)

- Returns next token from the StringTokenizer after switching to new delimter set
- New delimiter set remains in effect
- Throws
  - NoSuchElementException if no more tokens

132

example

```
String s = "hello zonko?misto?miles davis?";
StringTokenizer st = new StringTokenizer(s);
System.out.println(st.nextToken() );

System.out.println(st.nextToken("?") );
while (st.hasMoreTokens() )
   System.out.println(st.nextToken() );
```

```
hello
 zonko
misto
miles davis
```

133

METHOD

public boolean hasMoreElements ()

- Returns same value as hasMoreTokens
- Exists so that this class can implement the Enumeration interface

134

METHOD

public Object nextElement ()

- Returns same value nextToken but returns an Object
- Exists so that this class can implement the Enumeration interface

135

METHOD

public int countTokens()

- Returns the number of times that nextToken() can be called before it generates an exception

136

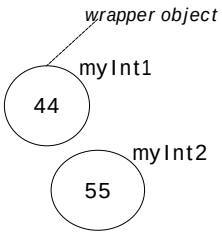
Wrapper Classes

137

Wrapper Classes

Classes

- Integer
- Float
- Double
- Byte
- Long
- Short
- Char
- Boolean

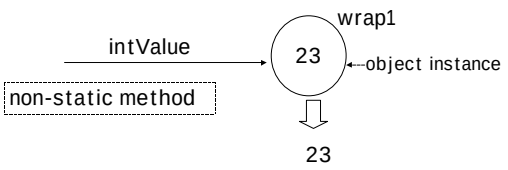



```
Integer myInt1 = new Integer("44");
Integer myInt2 = new Integer(55);
```

creates instances of Integer class

138

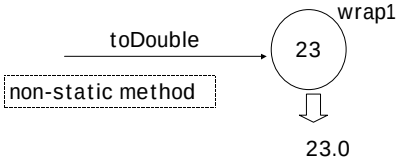
Integer Wrapper



int k = wrap1.intValue();

139

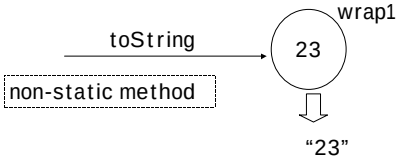
Integer Wrapper



double z = wrap1. toDouble();

140

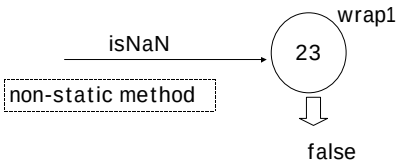
Integer Wrapper



String s = wrap1. toString();

141

Integer Wrapper




```
boolean b = wrap1. isNaN();
```

142

the corejava package

(written by textbook authors Cornell and Horstmann)

143

corejava package

Includes utility classes

- Assertion
- Day
- Format
- Console
- RandomIntGenerator

144

jar files

145

Jar files

- JAR stands for Java Archive
  - - used for packing many files into one
- Designed for quick applet download of classes to a browser in a single HTTP transaction
- JAR uses compression (based on ZIP)
- For class assignments, applications and all their associated files (including .java files) should be delivered as a Jar file

146

Create a jar file

```
%jar cvf a1jonesk.jar *.java *.class
```

name of jar file

files to include

options

c = create new archive

v = verbose

f = look for files to jar at end of command line

147

Extract from a jar file

```
%jar xf a5jonesk.jar
```

extracts all files into current directory

148

END

149