1. Java Programming Basics

• Classpath:
  — The JVM is using a class loader to load classes used by an application on an as-needed basis.
  — The CLASSPATH environment variable tells the class loader where to find the classes.
  — Classpath entries can be directories that contain class files for classes, or archive files (such as .zip or .jar files) that contain classes.
  — Classpath entries are colon-separated on Unix-type systems and semicolon-separated on MS Windows systems
  — In a bash/sh environment:
    % CLASSPATH=/home/hpb/Jrms/classes.jar:$CLASSPATH
    % export CLASSPATH
  — For javac/java/... on a UNIX platform:
    % jdkTool -classpath path1:path2...
  — Which classes are used?
    % java -verbose Hello.java
    java -verbose Hello.java
    [Opened /usr/local/j2sdk1.5.0/jre/lib/rt.jar]
    [Opened /usr/local/j2sdk1.5.0/jre/lib/jsse.jar]
    [Opened /usr/local/j2sdk1.5.0/jre/lib/jce.jar]
    [Opened /usr/local/j2sdk1.5.0/jre/lib/charsets.jar]
    [Loaded java.lang.Object from /usr/local/j2sdk1.5.0/jre/lib/rt.jar]
    ...

• Coding Standard: See

1.1. Comment

See also

Java defines three kinds of comments:

/* text */

A traditional comment: all the text from the ASCII characters /* to the ASCII characters */ is ignored (as in C and C++).

// text

A single-line comment: all the text from the ASCII characters // to the end of the line is ignored (as in C++).

/** documentation */

A documentation comment: the text enclosed by the ASCII characters /** and */ can be processed by a separate tool to prepare automatically generated documentation of the following class, interface, constructor, or member (method or field) declaration.
1.2. Identifier

An is an unlimited-length sequence of Java letters and Java digits, the first of which must be a Java letter. An identifier cannot have the same spelling as a keyword, Boolean literal, or the null literal.

1. Can be any length
2. First character must be a letter \{ 0, .. 9 \}
3. Following characters can be letters or digits
4. Are case sensitive: TOM is NOT the same as Tom
5. A Java reserved word CANNOT be used as an identifier, ex. true
6. Must be declared to be of some type.

Identifier should/must be self explained. Please do the following

The words i, k, ith, ... have no real meaning for a human being.

- lessThanTen versus ltt
- thisIsSorted versus t
- mph versus speed
- maximum versus m
The basic syntax for declaring variables is:

typename identifier;
typename identifier = expression;

A variable has the following properties:

• memory location to store the value.
• type of data stored in the memory location.
• name used to refer to the memory location.

Note: You may find the definitive reference for the Java programming language
1.3. Declaration versus Creation

A declaration declares and connects a variable with a type.

```java
int index;
String aString;
```

A creation creates an object.

The following programs have problems:

```java
/**
 * The program will not work.
 * What is the problem?
 */
class WillNotCompile {
    public static void main(String args[])
    {
        String aString;
        aString.length();
    }
}
```

Source Code: Src/4/WillNotCompile.java

```
yps 4 62 javac WillNotCompile.java
WillNotCompile.java:18: Variable aString may not have been initialized.
   aString.length();
```

1 error
class WillDie {
    public static void main(String args[]) {
        String aString = new String();
        aString = null;
        aString.length();
    }
}

Source Code: Src/4/WillDie.java

% javac WillDie.java
% java WillDie
java/lang/NullPointerException
    at WillDie.main(WillDie.java:19)
1.4. String Object/Class

- Strings are constants
- All literals are instances of the Sting class and exist once in a JVM.
- Their value can not be modified - why?

1.5. The First Program

```java
/*
 * Deal with Strings objects.
 * @version $Id$
 * @author hp bischof
 * Revisions: $Log$
 */

class StringThing {
    public static void main(String args[])
    {
        String aString;
        aString = new String("Last Stop Wonderland!");
        System.out.println( aString.length() );
        System.out.println( aString.toUpperCase() );
        System.out.println( aString.toUpperCase() + ".");
        System.out.println( aString.length() + 1 );
        System.out.println( 1 + aString.length() );
        System.out.println( 1 + aString + 1 );
        System.out.println( aString + ( 1 + 1 ) );
    }

Source Code: Src/4/StringThing.java

Result:
20
LAST STOP WONDERLAND
LAST STOP WONDERLAND.

Other Example

```
public static void method(String id, String literal, String aNewString) {
    System.out.println(id + " in method");
    System.out.print("\tliteral= aNewString\n   ");
    System.out.println( literal == aNewString);
}

public static void main( String args[] ) {
    String aString = "abc";
    System.out.print("abc == aString\n   ");
    System.out.println("abc" == aString);

    String newString = new String("abc");
    System.out.print("abc == new String(abc)\n   ");
    System.out.println("abc" == newString);

    method("1", "abc", "abc");
    method("2", "abc", new String("abc") );
    method("3", "abc", "ab" + "c");
    method("4", "abc", "" + "abc");
}

Source Code: Src/4/StringL.java

1.6. From the Bridge Exam

import java.util.*;

public class X_s1 {
    private String info;

    public X_s1 (String info) {
        this.info = info;
    }

    private String info() {
        return info;
    }

    public static void main (String args []) {
        X_s1 one = new X_s1("a");
        X_s1 two = new X_s1("a");

        if (one.info() == "a")  // 1 marked
            System.out.println("1. equal");
        if (one.info() == two.info() )  // 2 marked
            System.out.println("2. equal");
        if (one.info().equals(two.info()) ) // 3 marked
            System.out.println("3. equal");
        if (one.info().equals("aa".substring(0,1) ) // 4 marked
            System.out.println("4. equal");
        if (one.info().equals("aa".substring(0,1) ) ) // 5 marked
            System.out.println("5. equal");
    }
}
1.7. Use of the StringThing Class

```java
/**
 * Play with the String class
 *
 * @version $Id$
 * @author Hpb
 */

class String_1 {

   public static void main( String args[] ) {
        String aString = "David";
        String bString = "David Bowie";

        if ( "hello".equals("hello") )
            System.out.println("equal");
        if ( "David" == aString )
            System.out.println("David == aString ");
        System.out.println(aString.length());
        System.out.println(aString.charAt(0));

        System.out.println(aString.indexOf("vid"));

        System.out.println(aString.substring(2,3));
        System.out.println(aString.substring(aString.indexOf("a"),
                                      aString.indexOf("i")
                                      ));

        System.out.println(aString.concat(" Bowie").length());

        String help = bString.substring(0, aString.length());
        System.out.println("-->" + help + "<--");
        if ( "David" == help )
            System.out.println("David == help ");
        if ( "David" == aString )
            System.out.println("David == bString ");
   }
}
```

Source Code: Src/4/String_1.java
Result:

```
equal
David == aString
```

```
1.8. Strings and Numbers
```

```java
1
class StringAndInteger
2 {
3     public static void main(String args[])
4     {
5         System.out.println("Well, 3 + 4 = "+ 7);
6         System.out.println("Well, 3 + 4 = "+ 3 + 4);
7         System.out.println("Well, 3 + 4 = "+(3+4));
8     }
9 }
```

Source Code: Src/4/StringAndInteger.java
class StringAndInteger2
{
    public static void main(String args[])
    {
        System.out.println(3 + 7);
        System.out.println(3 + 7 + "abc");
        System.out.println(3 + 7 + "abc" + 1);
        System.out.println(3 + 7 + "abc" + 1 + 2);
        System.out.println("" + 3 + 7 + "abc" + 1 + 2);
        System.out.println("" + (3 + 7) + "abc" + (1 + 2));
    }
}

Source Code: Src/4/StringAndInteger2.java
1.9. More on Strings

```java
    /**
     * Play with the String class
     * @version $Id$
     * @author Hpb
     * $Log$
     */

    class StringUse {

        public static void compare(String aString, String bString) {
            if (aString.equals(bString))
                System.out.println("\tequal");
            else
                System.out.println("\t! equal");
            if (aString == bString)
                System.out.println("\t==");
            else
                System.out.println("\t!=");
        }

        public static void main(String args[]) {
            String aString = "David";
            String bString = "David";
            compare(aString, bString);

            System.out.println("Using New");
            aString = new String("David");
            bString = new String("David");
            compare(aString, bString);

            System.out.println("Concatenation 1");
            aString = "Da" + "vid";
            bString = "" + "David";
            compare(aString, bString);

            System.out.println("Concatenation 2");
            aString = "Da" + "vid";
            bString = "D" + "a" + "vid";
            compare(aString, bString);
        }
    }

Source Code: Src/4/StringUse.java
```
% java StringUse
  equal ==
Using New
  equal !=
  Concatenation 1
  equal ==
  Concatenation 2
  equal ==
1.10. Confusion about this

```java
/**
 * Use of this!
 */

class UseOfThis
{
    int id;
    UseOfThis(int id) {
        this.id = id;
    }
    private void method_2()
    {
        System.out.println("method_2: " + this);
    }
    private void method_1()
    {
        System.out.println("method_1: " + this);
        this.method_2();
        method_2();
    }
    public String toString() {
        return "" + id;
    }
    public static void main(String args[])
    {
        UseOfThis aUseOfThis = new UseOfThis(1);
        UseOfThis bUseOfThis = new UseOfThis(2);
        System.out.println(aUseOfThis);
        System.out.println(bUseOfThis);
        aUseOfThis.method_1();
        bUseOfThis.method_1();
    }
}
```

Source Code: Src/4/UseOfThis.java
1.11. Primitive Types and Values

A primitive type is predefined by the Java language and named by a reserved keyword. Please see also

Remember:

A variable has the following properties:

- memory location to store the value.
- type of data stored in the memory location.
- name used to refer to the memory location.

Java knows the following types:

<table>
<thead>
<tr>
<th>type</th>
<th>#bits</th>
<th>def. v.</th>
<th>minimum value</th>
<th>maximum value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>0</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>char</td>
<td>16 bits</td>
<td>0</td>
<td>0</td>
<td>65535</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>0</td>
<td>-32768</td>
<td>32767</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>0</td>
<td>-2147483648</td>
<td>2147483647</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>0</td>
<td>-9223372036854775808</td>
<td>9223372036854775807</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>0.0</td>
<td>-3.40282347E38</td>
<td>3.40282347E38</td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>0.0</td>
<td>-1.79E+308</td>
<td>1.79E308</td>
</tr>
</tbody>
</table>

Constants as in ANSI-C, constants consist of a decimal mantissa with a decimal point and an exponent with
prefix e or E and optional sign. Many parts can be omitted, but one of a decimal point, an exponent, or a
suffix must be present.

Constants are float only with the suffix f or F. With or without the suffix d or D they are double.

With the methods Float.intBitsToFloat() and Double.longBitsToDouble() one can change bitwise represen-
tations into floating point values.
Examples:

```java
int index;
int milesPerHour, maximumSpeed;
float pressure, sizeOfme;
double starTrekSpeed;
int picturesTakenSofar = 20;
double probability = 0.789;
```

Conditions can only be expressed with boolean values.

*boolean* has the predefined constants

- true
- false.

The names are not reserved; however, they are only recognized as boolean constants.
1.12. Unicode

Skip

See also

It is necessary for a modern environment to handle, uniformly and comfortably, the textual representation of all the major written languages.

Unicode Standard defines a uniform 16-bit code based on the principle of unification:

two characters are the same if they look the same even though they are from different languages.

This principle, called Han unification, allows the large Japanese, Chinese, and Korean character sets to be packed comfortably into a 16-bit representation.

The UTF encoding of the Unicode Standard is backward compatible with ASCII.

Letting numbers be binary, a rune \( c \) is converted to a multibyte UTF sequence as follows: (See also

1. \( c \) in \([00000000.0bbbbbbb]\) 0bbbbbbb
2. \( c \) in \([00000bbb.bbbbbbbb]\) 110bbbbb, 10bbbbbb
3. \( c \) in \([bbbbbbbb.bbbbbbbb]\) 1110bbbb, 10bbbbbb, 10bbbbbb

A byte in the ASCII range 0-127 represents itself in UTF.

Thus UTF is backward compatible with ASCII.
1.13. String to int

```java
/**
 * Converts a String to an int - this is one way out of many
 */

class StringToInt
{
    public static void main(String args[])
    {
        int i;
        Integer aInt = new Integer("4");
        i = aInt.intValue();
        i = Integer.parseInt("4");
    }
}
```

Source Code: Src/4/StringToInt.java

See also:
1.14. Arithmetic Expressions

- Exercises:

1.15. Arithmetic Operators

- the table below shows some of the arithmetic operators that Java provides, in the order of their precedence.
- parentheses can be used to change the order of evaluation
- an arithmetic expression returns (calculates) a value when executed.

<table>
<thead>
<tr>
<th>Binary Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>/</td>
<td>integer division, if both operands are integer; real division otherwise</td>
</tr>
<tr>
<td>%</td>
<td>remainder</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>bit shift left</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>bit shift right</td>
</tr>
<tr>
<td>&gt;&gt;&gt;</td>
<td>unsigned right shift</td>
</tr>
<tr>
<td>&amp;</td>
<td>bitwise and</td>
</tr>
<tr>
<td>^</td>
<td>bitwise xor</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
class BitPrint {
    private void printBytes (String what, int value) {
        System.out.print(what + "\t\t" + value + "\t\t");
        for ( int index = 31; index >= 0 ; index --) {
            if ( ( 1 << index ) & value ) == ( 1 << index ) )
                System.out.print("1");
            else
                System.out.print("0");
        }
        System.out.println();
    }

    public static void main (String args []) {
        BitPrint aBitPrint = new BitPrint();
        aBitPrint.printBytes("3   ", 3);
        aBitPrint.printBytes("4   ", 4);
        aBitPrint.printBytes("7   ", 7);
        aBitPrint.printBytes("-3  ", -3);
        aBitPrint.printBytes("-4  ", -4);
        aBitPrint.printBytes("-7  ", -7);
        aBitPrint.printBytes("5   ", 5);
        aBitPrint.printBytes("5 >> 1 ", (5 >> 1));
        aBitPrint.printBytes("-5  ", -5);
        aBitPrint.printBytes("-5 >> 1 ", (-5 >> 1));
        aBitPrint.printBytes("-5 >>> 1", (-5 >>> 1));
    }
}

Source Code: Src/4/BitPrint.java
Result:

```
javac BitPrint.java && java BitPrint
3  =  3  = 00000000000000000000000000000011
4  =  4  = 00000000000000000000000000000100
7  =  7  = 00000000000000000000000000000111
-3 = -3  = 11111111111111111111111111111011
-4 = -4  = 11111111111111111111111111111100
-7 = -7  = 11111111111111111111111111111101
5  = 5  = 00000000000000000000000000000001
5 >> 1 = 2  = 000000000000000000000000000000010
-5 = -5  = 11111111111111111111111111111101
-5 >> 1 = -3 = 111111111111111111111111111111101
-5 >>> 1 = 2147483645 = 011111111111111111111111111111101
```

1.16. Unary/Binary Operator

<table>
<thead>
<tr>
<th>unary Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>increment by 1</td>
</tr>
<tr>
<td>--</td>
<td>decrement by 1</td>
</tr>
<tr>
<td>!</td>
<td>not</td>
</tr>
</tbody>
</table>

Example:

```java
int left;
int right;
int result;

left = 4;
right = 3;
result = 4 * 3;
result = 4 + 3;
result = 4 / 3;
result = 4 % 3;

left ++;
result = left++;
result = ++left;

right ++;
result = right++;
result = ++right;
```
1.17. Mixed Mode Arithmetic and Casting

When an expression contains more than one arithmetic type all are converted to the heaviest.

\[
\text{byte} \rightarrow \text{char} \rightarrow \text{short} \rightarrow \text{int} \rightarrow \text{long} \rightarrow \text{float} \rightarrow \text{double}
\]

For example, \(2 + 3.3\) is interpreted as \(2.0 + 3.3\).

Java is strongly typed. However, the type of the results of evaluating an expression may be changed using casting. To cast, place the target type in parentheses before the operand or expression to be converted.

For example, if we really wanted the results of \(2 + 3.3\) to be integer, we could use

```java
int index;

index = 2 + (int) 3.3;
index = (int) (2 + 3.3);
```
Example:

```java
/**
 * The program deals with operators.
 * Comment not included.
 * @version $Id$
 * @author hp bischof
 * Revisions:
 * $Log$
 */

class OpEx {
    public static void main(String args[])
    {
        char aChar = 'b';
        byte aByte = 2;

        int intVar_1 = 1;
        int intVar_2 = 2;
        int intRes = 3;
        double doubleVar_1 = 3.8;
        double doubleVar_2 = 4.8;
        double doubleRes = doubleVar_1 - doubleVar_2;

        System.out.println("1. "+aChar); // man ascii decimal set
        System.out.println("2. "+aByte);
        System.out.println("3. "+aByte+aChar);
        System.out.println("4. "+aByte+0);
        System.out.println("5. "+aChar+0);

        intRes = 5 / 3; System.out.println("6. "+intRes);
        intRes = 5 % 3; System.out.println("7. "+intRes);
        intRes = (int)(5 / doubleVar_2); System.out.println("8. "+intRes);

        doubleRes = 5 / doubleVar_2; System.out.println("9. "+doubleRes);
        doubleRes = 5.0 / doubleVar_2; System.out.println("10. "+doubleRes);
    }
}

Source Code: Src/4/OpEx.java

% javac OpEx.java
% java OpEx
1. b
2. 2
3. 2b
4. 20
5. b0
6. 1
7. 2
8. 1
9. 1.0416666666666667
10. 1.0416666666666667
1.18. Assignment Operators

There are 12 assignment operators; all are syntactically right-associative (they group right-to-left). See also

\[
\begin{align*}
&= \ast= \div= \%= += -= \shortleftarrow= \shortrightarrow= \shortdownarrow= \shortuparrow= \\
&\ll= \lll= &\llll= \lllll=
\end{align*}
\]

\begin{itemize}
  \item \ll = bit shift left
  \item \lll = bit shift right
  \item \llll = unsigned right shift
  \item \& = and
  \item ^ = xor
  \item | = or
\end{itemize}

The syntax for an assignment is:

Assignment:
  \[ \text{LeftHandSide AssignmentOperator AssignmentExpression} \]

<table>
<thead>
<tr>
<th>LeftHandSide</th>
<th>must be a variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssignmentOperator</td>
<td>must be one of (= \ast= \div= %= += -= \shortleftarrow= \shortrightarrow= \shortdownarrow= \shortuparrow=)</td>
</tr>
<tr>
<td>AssignmentExpression</td>
<td>must be ConditionalExpression or Assignment</td>
</tr>
</tbody>
</table>

Note: A variable that is declared final cannot be assigned to.
1.19. Playing with the Basic Types

```java
/**
 * Ranges demonstrates integer value manipulation, bit operations, and conversions.
 * Comment not included.
 * @version $Id$
 * @author Axel T. Schreiner
 * @author hp bischof
 *
 * Revisions:
 * $Log$
 */

class Ranges {
  /** uses complement operations */
  short x;
  void intRange () {
    System.out.println("int\t" + (~0 >>> 1) + "\t" + (~(~0 >>> 1)));
    System.out.println("int\t" + (~(~0 >>> 1)));
    printBytes("(~0 >>> 1)\t”, (~(~0 >>> 1)));
    }
  /** maximum and minimum long value */
  static final long maxLong = ~0L >>> 1;
  static final long minLong = ~(~0L >>> 1);
  
  private static void printBytes (String what, int value) {
    System.out.print(what + "\t" + value + "\t=\t");
    for (int index = 31; index >= 0; index --) {
      if ((1 << index) & value) System.out.print("1");
      else System.out.print("0");
    }
    System.out.println();
  }
  }
  void longRange () {
    System.out.println("long\t" + maxLong + "\t" + minLong);
  }
  /** uses casts and literals */
  void shortRange () {
    System.out.println("short\t" + Short.MIN_VALUE + "\t" + Short.MAX_VALUE);
    System.out.println("short\t" + (short)077777 + "\t" + (short)0x8000);
  }
  /** shifts ones until no further changes occur */
  void byteRange () {
```

byte i, j = 1;

do {
    i = j; j = (byte)(i << 1 | 1);
} while (j > i);
System.out.print("byte\t" + i);

do {
    i = j; j = (byte)(i << 1);
} while (j < i);
System.out.println("\t" + i);

public static void main (String args []) {
    Ranges aRange = new Ranges();
    aRange.byteRange();
    aRange.shortRange();
    aRange.intRange();
    aRange.longRange();
}

Source Code: Src/4/Ranges.java
**java Ranges**

<table>
<thead>
<tr>
<th>Type</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>127</td>
<td>-128</td>
</tr>
<tr>
<td>short</td>
<td>32767</td>
<td>-32768</td>
</tr>
<tr>
<td>int</td>
<td>2147483647</td>
<td>-2147483648</td>
</tr>
<tr>
<td>long</td>
<td>9223372036854775807</td>
<td>-9223372036854775808</td>
</tr>
</tbody>
</table>

**intRange()**
- produces a maximal bit pattern by complementing 0 and shifting without propagating the sign, and a minimal bit pattern by complementing once more.
- concatenates strings; integer operands are converted to short strings. java/gtext has facilities for explicit formatting.

**static**
- defines `maxLong` and `minLong` as class variables,

**final**
- permits exactly one assignment. Initialization is done with a long literal and thus long arithmetic.

**shortRange()**
- uses int literals which must be explicitly cast to short. The following expression produces a minimal value in C but not in Java (why?):
  \[
  (\text{short}) \sim ((\text{unsigned short}) \sim 0 \gg 1)
  \]

**byteRange()**
- uses do-while loops. byte can be initialized with int, but for assignment an explicit cast is required.
1.20. Flow Control

1.21. Conditions

- Conditions can only be expressed with boolean values; unlike C, an implicit comparison to zero is not acceptable.
- Boolean has the predefined constants true and false. The names are not reserved; however, they are only recognized as boolean constants.

1.22. Relational Operators

- Simple boolean expressions consist of comparing things using relational operators. There are two types of relational operators: equality and comparison.

Equality operators are defined for all objects and primitive types.

```plaintext
==   equal
!=   not equal
```

All comparisons produce true or false.
1.23. Logical Operators

These operators take boolean arguments and return boolean results. They are used to construct complex boolean expressions from simple ones consisting of boolean values and relational expressions.

- `&` bitwise and
- `&&` conditional and (short circuits)
- `|` bitwise or
- `||` conditional or (short circuits)
- `^` bitwise xor
- `!` not (unary operator)/boolean complement

- `x && y` y will be evaluated only if x is true
- `x || y` y will be evaluated only if x is false
class ShortCut
{
    private boolean testIt(double n) {
        return n != 0.0;
    }
    private double oneDn(double n) {
        return 1.0 / n;
    }
    public static void main(String args[]) {
        double n;
        ShortCut aS = new ShortCut();
        n = 0.5;
        if (aS.testIt(n) && (aS.oneDn(n) > 1))
            System.out.println("1: 1 / n" + 1 / n);
        n = 0;
        if (aS.testIt(n) && (aS.oneDn(n) > 1))
            System.out.println("2: 1 / n" + 1 / n);
        System.out.println("3: 1.0 / 0 = " + 1.0 / 0);
        if ((n = 0) || (n == 1))
            System.out.println("4: ( n == 0 ) || ( n == 1 )");
        System.out.println("5. true || false && true: " + (true || false && true));
        if (4 == (n = 4))
            System.out.println("6. 4 == ( n = 4 )");
    }
}

Source Code: Src/4/ShortCut.java

Result:
%
javac ShortCut.java && java ShortCut
1: 1 / n 2.0
3: 1.0 / 0 = Infinity
4: ( n == 0 ) || ( n == 1 )
5. true || false && true: true
6. 4 == ( n = 4 )
The precedence for the arithmetic, relational, boolean operators, and assignment from highest to lowest is:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Symbol</th>
<th>Precedence</th>
<th>Association</th>
</tr>
</thead>
<tbody>
<tr>
<td>grouping</td>
<td>( )</td>
<td>1</td>
<td>left to right</td>
</tr>
<tr>
<td>unary</td>
<td>+ -</td>
<td>2</td>
<td>right to left</td>
</tr>
<tr>
<td>multiplication</td>
<td>*</td>
<td>3</td>
<td>left to right</td>
</tr>
<tr>
<td>division</td>
<td>/</td>
<td>3</td>
<td>left to right</td>
</tr>
<tr>
<td>remainder</td>
<td>%</td>
<td>3</td>
<td>left to right</td>
</tr>
<tr>
<td>addition</td>
<td>+</td>
<td>4</td>
<td>left to right</td>
</tr>
<tr>
<td>subtraction</td>
<td>-</td>
<td>4</td>
<td>left to right</td>
</tr>
<tr>
<td>less than</td>
<td>&lt;</td>
<td>5</td>
<td>left to right</td>
</tr>
<tr>
<td>less than or equal</td>
<td>&lt;=</td>
<td>5</td>
<td>left to right</td>
</tr>
<tr>
<td>greater than</td>
<td>&gt;</td>
<td>5</td>
<td>left to right</td>
</tr>
<tr>
<td>greater than or equal</td>
<td>&gt;=</td>
<td>5</td>
<td>left to right</td>
</tr>
<tr>
<td>equal</td>
<td>==</td>
<td>6</td>
<td>left to right</td>
</tr>
<tr>
<td>not equal</td>
<td>!=</td>
<td>6</td>
<td>left to right</td>
</tr>
<tr>
<td>bit and</td>
<td>&amp;</td>
<td>7</td>
<td>left to right</td>
</tr>
<tr>
<td>xor</td>
<td>^</td>
<td>8</td>
<td>left to right</td>
</tr>
<tr>
<td>bit or</td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>conditional and</td>
<td>&amp;&amp;</td>
<td>10</td>
<td>left to right</td>
</tr>
<tr>
<td>conditional or</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>assignment</td>
<td>=, +=, *= ...</td>
<td>12</td>
<td>N.A.</td>
</tr>
</tbody>
</table>
1.24. if Statement

See also

The if statement allows conditional execution of a statement or a conditional choice of two statements, executing one or the other but not both.

IfThenStatement:

\[
\text{if ( Expression )} \\
\text{Statement}
\]

Example:

\[
x = 3; \\
y = 4; \\
\text{if ( x > y )} \\
\text{z = x;}
\]

IfThenElseStatement:

\[
\text{if ( Expression )} \\
\text{Statement} \\
\text{else} \\
\text{Statement}
\]

Example:

\[
x = 3; \\
y = 4; \\
\text{if ( x > y )} \\
\text{z = x;} \\
\text{else} \\
\text{z = y;}
\]

class If
{
    public static void main(String args[]) {
        int index = 2;
        System.out.println("1. " + index);
        if ( ++index == 2 )
            System.out.println("2. " + index);
        else if ( index++ == 3 )
            System.out.println("2. " + index);
        else
            System.out.println("2. " + index);
    }
}

Source Code: Src/4/If.java

% java If
1. 2
2. 4
1.25. Find the Maximum of two Numbers I

```java
    /**
     * Find the maximum of two numbers.
     */
    @version $Id$
    @author Hpb
    *
    * Revisions:
    * $Log$
    */

class Maximum_2 {
    public static double maximum(double _first, double _second ) {
        double max;
        // find maximum
        if ( _first > _second )
            max = _first;
        else
            max = _second;
        return max;
    }

    private double minimum(double _first, double _second ) {
        double minimum;
        // find minimum
        if ( _first < _second )
            minimum = _first;
        else
            minimum = _second;
        return minimum;
    }

    public static void main( String args[] ) {
        Maximum_2 aMaximum_2 = new Maximum_2();
        double firstN   = 42.0;
        double secondN = 666.0;
        System.out.println("Maximum(\" + firstN + ", \" + secondN + ") = \" +
            aMaximum_2.maximum( firstN, secondN) );
        System.out.println("Minimum(\" + firstN + ", \" + secondN + ") = \" +
            aMaximum_2.minimum( firstN, secondN) );
    }
}

Source Code: Src/4/Maximum_2.java

As a Nassi Shneidermann diagram:

```public static double maximum(double _first, double _second )
if if ( _first > _second ) then Ye s: return _first else No
return _second }```
public static double minimum(double _first, double _second )
if (_first < _second ) then Yes : return _first else No
return _second 

1.26. The conditional operator

?:
uses the boolean value of one expression to decide which of two other expressions should be evaluated.
The conditional operator is syntactically right-associative (it groups right-to-left), so that

\[ a \ ? \ b : c \ ? \ d : e \ ? \ f : g \]

means the same as

\[ a \ ? \ b : (c \ ? \ d : (e \ ? \ f : g)) \].

The conditional operator may be used to choose between second and third operands of numeric type, or
second and third operands of type boolean, or second and third operands that are each of either reference
type or the null type. All other cases result in a compile-time error.

See also:

```java
1 2 class QuestionM
3 { 4    public static void main(String args[]) {
5        int value = 2 > 3 ? 2 : 3;
6        String aString = "2 > 3 ? 2 : 3";
7        System.out.println(aString + " = " + value);
8
9        value = (1 > 2 ? 3 : (4 < 5 ? 6 : 7));
10       aString = "1 > 2 ? 3 : (4 < 5 ? 6 : 7)";
11      System.out.println(aString + " = " + value);
12
13        value = 1 > 2 ? 3 : 4 > 5 ? 8 : 7;
14        aString = "1 > 2 ? 3 : 4 > 5 ? 8 : 7";
15      System.out.println(aString + " = " + value);
16    }
17 }

Source Code: Src/4/QuestionM.java
```

% java QuestionM
2 > 3 ? 2 : 3 = 3
( 1 > 2 ? 3 : ( 4 < 5 ? 6 : 7 )) = 6
1 > 2 ? 3 : 4 > 5 ? 8 : 7 = 7
class Maximum_3 {
    private double maximum(double _first, double _second ) {
        return _first > _second ? _first : _second;
    }

    public static double minimum(double _first, double _second ) {
        return _first < _second ? _first : _second;
    }

    public static void main( String args[] ) {
        Maximum_3 aMax = new Maximum_3();
        double firstN = 42.0;
        double secondN = 7.0;

        System.out.println("Maximum(" + firstN + ", " + secondN + ") = " + aMax.maximum( firstN, secondN ) );
        System.out.println("Minimum(" + firstN + ", " + secondN + ") = " + aMax.minimum( firstN, secondN ) );
    }
}

Source Code: Src/4/Maximum_3.java
1.27. while Statement

See also

The while statement executes an Expression and a Statement repeatedly until the value of the Expression is false.

WhileStatement:

```
while ( Expression )
    Statement
```

Example:

```
x = 1;
while ( x < 10 ){
    print x
    x += 2;
}
```

```java
1
2
class While
3 {
  public static void main(String args[]) {
    int index = 1;
    while ( index > 0 ? ( index == 7 ) : (index == 8 ) ) {
      System.out.println("index = " + index );
    }
    System.out.println("index ="+index );
  }
}
Source Code: Src/4/While.java
```

```java
% java While
index = 1
```

```java
1
2
class While_1
3 {
  public static void main(String args[]) {
    int index = 1;
    while ( ++index++ < 4 ) {
      System.out.println("index = " + index );
    }
    System.out.println("index = " + index );
  }
}
Source Code: Src/4/While_1.java
```
class While_2
{
    public static void main(String args[])
    {
        int index = 1;
        while ( ++index < 4 )
        {
            System.out.println("index = " + index);
        }
        System.out.println("index = " + index);
    }

Source Code: Src/4/While_2.java

% java While_2
index = 2
index = 3
index = 4
The Expression must have type boolean, or a compile-time error occurs.

A while statement is executed by first evaluating the Expression. If evaluation of the Expression completes abruptly for some reason, the while statement completes abruptly for the same reason. Otherwise, execution continues by making a choice based on the resulting value:

If the value is true, then the contained Statement is executed. Then there is a choice:

If execution of the Statement completes normally, then the entire while statement is executed again, beginning by re-evaluating the Expression.

If the value of the Expression is false, no further action is taken and the while statement completes normally.

If the value of the Expression is false the first time it is evaluated, then the Statement is not executed.
1.28. Calculate Sqrt(2) without the MathClass

Algorithm:

\[
\begin{align*}
\text{n1} &= 1.0 : \quad \text{n2} = 2.0 : \quad \text{index} = 1 \ 	ext{do} \quad \text{while} \ n2 \times n2 - n1 \times n1 > 0.0001 : \quad \text{x} = n1 + ( n2 - n1 ) \times 0.5 \\
\text{if} \ ( x \times x > 2 ) \ \text{then YES} : \quad n2 = x \ \text{else NO} : \quad n1 = x 
\end{align*}
\]
class Sqrt {

    private double calculateSqrt_2() {
        double n1 = 1.0;
        double n2 = 2.0;
        while ((n2 * n2 - n1 * n1) > 0.0001) {
            double x = (n2 + n1) * 0.5;
            if (x * x > 2.0 )
                n2 = x;
            else
                n1 = x;
        }
        return n1;
    }

    public static void main( String args[] ) {
        System.out.println("sqrt(2) = " + new Sqrt().calculateSqrt_2() + " +- 0.0001 ");
    }
}

Source Code: Src/4/Sqrt.java
1.29. **Continue**

- The `continue` statement may occur only in a `while`, `do`, or `for` statement;
- The `continue` statement with no label attempts to transfer control to the innermost enclosing `while`, `do`, or `for` statement;
- This statement, which is called the `continue` target, then immediately ends the current iteration and begins a new one.
- If no `while`, `do`, or `for` statement encloses the `continue` statement, a compile-time error occurs.

**Example 1:**

```java
class Continue_1 {
    public static void main( String args[] ) {
        int n = 0;
        label1:
        while ( n < 6 ) {
            System.out.println("1. n == " + n);
            while ( n < 4 ) {
                n++;
                System.out.println(" 2. n == " + n);
                if ( n > 2 )
                    continue label1;
                System.out.println(" 3. n == " + n + "--------");
            }
            n++;
        }
    }
}
```

Source Code: Src/4/Continue_1.java

```
% java Continue_1
1. n == 0
  2. n == 1
  3. n == 1--------
  2. n == 2
  3. n == 2--------
  2. n == 3
  1. n == 3
    2. n == 4
  1. n == 4
  1. n == 5
```

1.30. **Break**

- A break statement transfers control out of an enclosing iteration or switch statement.
- Labeled breaks are used to jump out of nested loops
• break statement with no label attempts to transfer control to the innermost enclosing switch, while, do, or for statement;
• If no switch, while, do, or for statement encloses the break statement, a compile-time error occurs.

Example 1:

```java
class Break_1 {
    public static void main( String args[] ) {
        int n = 0;
        here:
            while ( true ) {
                System.out.println("1. n == " + n);
                while (n<100) { // while ( true ) --> which problem
                    n++;
                    System.out.println("2. n == " + n);
                    if (n>2)
                        break here;
                }
                System.out.println("3. n == " + n);
            }
        System.out.println("after here ");
    }
}
```

Source Code: Src/4/Break_1.java

Example 2:

```java
class Break_2 {
    public static void main( String args[] ) {
        int n = 0;
        System.out.println("start");
        while ( n < 100 ) {
            if (n>4)
                System.exit(1);
            while (n<100) { // while ( true ) --> which problem
                n++;
                System.out.println("inner while here n == " + n);
                if (n>2)
                    break;
            }
            System.out.println("outer while here n == " + n);
        }
        System.out.println("after here ");
    }
}
```
1.31. Return

A return statement returns control to the invoker of a method or constructor.

```java
/* How can we set the exit code? */

class Return {

    public static void main( String args[] ) {
        int x = 0;
        return x;
    }
}

Source Code: Src/4/Return.java
```

```java
/* How can we set the exit code? */

class Return_1 {

    public static int method() {
        System.exit(2);
        return 0;
    }

    public static void main( String args[] ) {
        method();
        System.out.println("xxx");
    }
}

Source Code: Src/4/Return_1.java
```
1.32. Return vs. Continue vs. Break

- What are the differences?

1.33. Abrupt Completion

Abrupt completion of the contained Statement is handled in the following manner:

An abrupt completion always has an associated reason, which is one of the following: (from

- A break with no label
- A break with a given label
- A continue with no label
- A continue with a given label
- A return with no value
- A return with a given value
- A throw with a given value, including exceptions thrown by the Java virtual machine
class Break {
    public static void main( String args[] ) {
        int n = 0;

        here: {
            while ( true ) {
                System.out.println("a: uter while here n == " + n);
                if ( n > 4 )
                    System.exit(1);

                while ( true ) {
                    System.out.println(" inner while here n == " + n);
                    if ( ++n == 0 )
                        System.out.println("n == 0");
                    else if ( n++ == 1 ) {
                        System.out.println(" n= =1");
                        System.out.println(" break");
                        break;
                    } else if ( n++ == 2 )
                        System.out.println(" n == 2");
                    else
                        System.out.println(" n= =3");

                    System.out.println(" executing break here");
                    System.out.println(" n is " + n);
                    break here;
                    }
                    System.out.println("b: outer while here n == " + n);
                } // unreachable statement ...System.out.println("here ");
            }
        }

        Source Code: Src/4/Break.java
1.34. do Statement

See also

The do statement executes a Statement and an Expression repeatedly until the value of the Expression is false.

do Statement while ( Expression ) ;
1.35. for Statement

See also

The for statement executes some initialization code, then executes an Expression, a Statement, and some update code repeatedly until the value of the Expression is false.

ForStatement:

```
for ( ForInit; Expression; ForUpdate)
    Statement
```

Example:

```java
class For_1 {
    public static void main( String args[] ) {
        int index = 0;
        for ( index = 0 ; index < 1; index ++ ) {
            System.out.println("1. index = " + index );
        }
        System.out.println("2. index = " + index );
    }
}
```

Source Code: Src/4/For_1.java

```java
class For_2 {
    public static void main( String args[] ) {
        int index = 0;
        for ( index = 0 ; index < 1; index ++ ) {
            index = -1;
            System.out.println("1. index = " + index );
            break;
        }
        System.out.println("2. index = " + index );
    }
}
```

Source Code: Src/4/For_2.java

```java
class For_3 {
    public static void main( String args[] ) {
        for ( int index = 0 ; index < 1; index ++ ) {
            System.out.println("1. index = " + index );
            break;
        }
        System.out.println("2. index = " + index );
    }
}
```

Source Code: Src/4/For_3.java
1.36. Find all Prime Numbers in \([ 2 \ldots 100 ]\)

\(\text{isPrime(n): do } \text{for index = 2 to n - 1: if ( index % n == 0 ) return false } \text{ return true } \)

\(\text{findAllPrimeN(): do } \text{for index = 1 to 100: if ( isPrime(n) ) print index } \)
/**
 * Find all prime numbers in the range between 1 and 10
 * @version $Id$
 * @author Hpb
 * Revisions:
 * $Log$
 */

class Prime_1 {
    private boolean isPrime(int n) {
        for (int index = 2; index < n; index++) {
            if (n % index == 0)
                return false;
        }
        return true;
    }
    public static void main(String args[])
    {
        Prime_1 aPrime = new Prime_1();
        for (int index = 2; index <= 10; index++)
        {
            if (aPrime.isPrime(index))
                System.out.println(index + " ");
        }
    }

    Source Code: Src/4/Prime_1.java
1.37. Switch Statement

See also

The switch statement transfers control to one of several statements depending on the value of an expression.

The type of the switch expression can be

- char
- byte
- short
- int
- strings
- enum type See

Switch Statement:

```java
switch ( Expression ) {
    case ConstantExpression_1 : action_1;
    case ConstantExpression_2 : action_2;
    ...
    default: action_d
```
Example:

```java
class Switch {

    static void method(int k) {
        switch (k) {
            case 1: System.out.println("with break: 1 ");
             break;
            case 2: System.out.println("with break: 2 ");
             break;
            default: System.out.println("with break: default");
        }
    }

    static void methodWithoutDefault(int k) {
        switch (k) {
            case 1: System.out.println(" without break: 1 ");
            break;
            case 2: System.out.println(" without break: 2 ");
            break;
        }
    }

    public static void main(String[] args) {
        new Switch().method(3);
        new Switch().methodWithoutDefault(2);
        new Switch().methodWithoutDefault(3);
    }
}
```

Source Code: Src/4/Switch.java

1.38. Partial Lowercase → Uppercase

```java
/**
 * Test of the switch statement.
 *
 * @version $Id$
 * @author hpb
 *
 * Revisions:
 * $Log$
 */

class Switch_1 {
    private String itWasA(char c) {
        switch( c ) {
            case 'a': return("A"); // break??
            case 'b': return("B"); // break??
        }
    }
```
18 case 'c':  return("C");  // break??
19 case 100: return("D");  // break??
20 case 101: return("E");  // break??
21 default: return("no clue, but not an [a-e]");
22 // What happens if
23 // we delete this line?
24 } 
25 } 
26 
27 public static void main( String args[] ) { 
28 char theChar; 
29 theChar = 'd'; 
30 System.out.println("You typed in an "+
31 new Switch_1().itWasA(theChar) + ""); 
32 System.exit(0);  // beware of ...
33 
34 } 
35 
36 Source Code: Src/4/Switch_1.java 
37

Characters can be safely converted to integers (Unicode), but should be avoided at all times.

1.39. Questions
• Which variable names are valid:

```
1               class X_1
2               {
3                  public static void main(String args[])
4                  { 
5                     int aInt;
6                     int countUpTo5;
7                     int 5IsA_niceNumber;
8                     int ooo\";
9                     int ntoTooMany:areAllowed;
10                  }
11               }
12               
13 Source Code: Src/4/X_1.java
```

• What is the output of the following program:

```
1               class X_2
2               {
3                  public static void main(String args[])
4                  { 
5                     System.out.println("I like to play " + 6 + 2");
6                     System.out.println("I like to play " + 6 * 2);
7                     System.out.println("I like to play " + (6 + 2));
```
Will the following program compile?

```java
class X_3
{
    public static void main(String args[])
    {
        int i = 0;
        i += 63;
        System.out.println("1. " + (i++ >> 2));
        System.out.println("2. " + (1 > 2 ? 3 : 6));
    }
}
```

What is the output of the following program:

```java
class X_4
{
    public static void main(String args[])
    {
        int i = 0;
        i += 63;
        System.out.println("1. " + (i++ >> 2));
        System.out.println("2. " + (1 > 2 ? 3 : 6));
        System.out.println("3. " +
                        (1 > 2 ? 3 : (4 < 5 ? 6 : 9 < 10 ? 7 : 8)));
        System.out.println("4. " +
                         (1 > 2 ? 3 : (4 < 5 ? 6 : (9 < 10 ? 7 : 8))));
    }
}
```
class X_5 {
public static void main( String args[] ) {
    int n = 0;
    while ( true ) {
        System.out.println("xx");
        if ( n++ == 0 ) {
            System.out.println("n == 0");
        } else if ( n++ == 1 ) {
            System.out.println("n == 1");
        } else if ( n-- == 2 )
            System.out.println("n == 2");
    }
}

Source Code: Src/4/X_5.java