Java Expressions & Flow Control

Rui S. Moreira
@ FTC/UFP

Expression Separators: . [ ] ( ) , ;

- Dot used as decimal separator or to access attributes and methods

```
double d = 2.6;
Ponto ponto = new Ponto(2.0f, 3.0f);
float i = ponto.x;
float j = ponto.getY();
```

- Array definition uses [ ]

```
int[] arrayInts = new int[10]; //Array of primitive int
Point arrayPts[] = new Point[8]; //Array of references
```

- Parenthesis change order of evaluation of expressions

```
float f=5*(4+8.9);
```

- Comma used to separate elements of an expression (on a single line)

```
long x1=18L, x2=28L, x3=38L;
```

- Semicolon is a terminator (used to terminate expressions)

```
double fact (int number){
   double f=1;
   for(int i=2; i<=number; i++){ f *= i; }
   return f;
}
```

Operators

- **Attribution and Arithmetic:** **= ++ -- * / % + -** (can be combined)

```
int i=+1, k=15, j=-20, p=10;
i++; k--; ++j; p+=10;
d = 2.6+5*(x-x%2)/10;
System.out.print("Hello " + "World!"); // + used to concatenate strings
```

- **Comparison:** **== != > >= < <= instanceof**

```
boolean bool1 = 1==10; // bool1 gets false
boolean bool2 = 1!=10; // bool2 gets true
boolean bool3 = 1>10; // bool3 gets false
boolean bool4 = 1<=10; // bool4 gets true
boolean bool5 = ponto instanceof Ponto; // bool5 gets true
```

- **Logic:** **&& || !** (used only with **boolean**)

```
bool6 = bool1 && bool2; // bool6 gets false because bool1 is false (AND)
bool6 = bool1 || bool2; // bool6 gets true because bool1 is true (OR)
bool6 = !bool1; // bool6 gets true, i.e, negation of bool1 (NOT)
if (x<10 && x>5 || !(y==18)) { // NB: short-circuit operators
   truth = !rect.isInside(ponto);
}
```


Operators

- **Bitwise:** **& ^ | ~** (used with **int/short/long/char/byte**)

```
byte b1 = 63 & 252; // Does bit-by-bit AND - 60(10)=0..00111100(2)
byte b2 = 63 | 252; // Does bit-by-bit OR - 255(10)=0..11111111(2)
byte b3 = 63 ^ 252; // Does bit-by-bit XOR - 195(10)=0..11000011(2)
byte b4 = 7; // 7(10)=00000111(2)
byte b5 = ~b4; // Does bit-by-bit COMPLEMENT/NOT -8(10)=11111000(2)
```

Examples:

```
// 63 & 252 (AND example) - true if both bits are true
63(10)   = 00000000 00000000 00000000 00111111
252(10)  = 00000000 00000000 00000000 11111100
----------------------------------------
60(10)   = 00000000 00000000 00000000 00111100

// 63 ^ 252 (XOR example) - true if both bits are equal
63(10)   = 00000000 00000000 00000000 00111111
252(10)  = 00000000 00000000 00000000 11111100
----------------------------------------
195(10)  = 00000000 00000000 00000000 11000011
```

Operators

- **Bitwise Arithmetic Shift:** **>> <<** (left-shift and right-shift)

```
// only used with int/short/long/char/byte
// with int the shift-value is always between 0..31
// with long the shift-value is always between 0..63
byte b1 = 128>>1; // b1 gets 64 (divide by 2) - inserts left 0's
b1 = 32<<4; // b1 gets 256 (multiply by 4)
b1 = -256>>4; // b1 gets -32 (divide by 4) - copies signal-bit
```

Examples:

// 16 << 5 right-shift example – inserts 0's on the rigth

16_{10} = 00000000 00000000 00000000 00010000
512_{10} = 00000000 00000000 00000010 00000000

// 16 >> 2 right-shift example – signal bit 0 is preserved

16_{10} = 00000000 00000000 00000000 00010000
4_{10} = 00000000 00000000 00000000 00000100

// -16 >> 2 right-shift example – signal bit 1 is preserved

-16_{10} = 11111111 11111111 11111111 11110000
-4_{10} = 11111111 11111111 11111111 11111100

Operators

- **Bitwise Logic Shift:** **>>>** (unsigned right-shift)

```
// only used with int/short/long/char/byte
// The operator >>> inserts 0's in the most significant bit
int b2 = -12; // this byte is -12(10)=1..00001100(2)
b2 = b2>>2; // b2 gets -3(10)=11000011(2)
b2 = b2>>>2; // b2 gets +3(10)=0..00000011(2)
```

Examples:

// 16 >>> 2 (unsigned right-shift example) – signal bit is the same

16_{10} = 00000000 00000000 00000000 00010000
4_{10} = 00000000 00000000 00000000 00000100

// -16 >> 2 (unsigned right-shift example) – signal bit is NOT preserved

-16_{10} = 11111111 11111111 11111111 11110000
$1.073.741.820_{10}$ = 00111111 11111111 11111111 11111100

Operators

- **Creator:** **`new <ClassConstructor>()`**

```
Ponto p1 = new Ponto(2, 3),
      p2 = new Ponto(5, 8);
Rectangle r1 = new Rectangle(p1, p2);
```

- **Cast:** **`(<ClassName>)`**

```
Object o = new Ponto(1.0f, 4.0f); //o can reference any object
Ponto p2 = (Ponto)o; //Casts (forces type conformance)

ArrayList a = new ArrayList(); //ArrayList is a growable array
a.add(p1); //Stores Point p1 into pos 0
a.add(r1); //Stores Rectangle r1 into pos 1
Ponto p = (Ponto)a.get(0); //Must Cast to Point
Rectangle r = (Rectangle)a.get(1); //Mst Cast to Rectangle
```


Control Flow

- Conditional execution:
  - `if (i<10) { ...; } else { ...; }`
  - `switch (i) { case 1: ...; break; default: ...; }`
- Loops:
  - `for(int i; i<limit; i++){ ...; }`
  - `while (i<limit) { ...; i++; }`
  - `do { ...; i++; } while (i<limit);`
- Special controls:
  - `break`
  - `continue`
  - `label1: while(...) { ...; if (...) break label1}`
  - `label2: for(...) { ...; if (...) continue label2}`

if (<*boolean-expression*>) { } else {}

- **Syntax alternatives:**

  - `if (<boolean-expression>) { ...; }`

  - ```
    if (<boolean-expression>) { ...;
    } else { ...;}
    ```

  - ```
    if (<boolean-expression>) { ...;
    } else if (<boolean-expression>) { ...;}
    ```

  - ```
    if (<boolean-expression>) { ...;
    } else if (<boolean-expression>) { ...;
    } else { ...; }
    ```

- **Example** (NB: short-circuit versus non short-circuit operands):

```
if (i<10 && truth){ //Do not tests 2nd operand if 1st fails
   i++;
} else if (i<20 & truth){ //Tests all operands
   i+=20;
}
```


switch (<*int-expression*>) { ... }

- **Syntax:**

```
switch (<int-expression or string>) {// Does not allow float or double
     case <int-value>: ...;
                       break;
     case <int-value>: ...;
                       break;
     default: ...;
}
```

- **Example:**

```
switch (option) { // If option is char/byte/short it is promoted to int
     case 1:  callOpenFile();
              break;
     case 2:  callSaveFile();
              break;
     default: callExitProgram();
}
```
```

for (<*init*>; <*test*>; <*update*>) { ... }

- **Syntax:**

```
for (<initialisation>; <test>; <actualization>) {...}
for (<classtype> variable: <collection>) {...}
```

- **Example:**

```
for (int i=0, j=20; i<18 && j>5; i++, j++) {
   // We can use commas to separate expressions
   temp += i*j;
}

int arrayInts[] = new int[10];
for (int n: arrayInts) {//For each n int inside arrayInts
   System.out.println(i);
}
```


while (<*boolean-expression*>) { ... }

- **Syntax:**

```
while (<boolean-expression>) {
   // This cycle will not execute if expression is false
   // We must include code to control stop/break the cycle
}
```

- **Examples:**

```
while (i<100) {
   // ...
   i++; // Sometime it will reach 100
}

while (keepdoing) {
   // ...
   if (temp==value) keepdoing=false; // Stop cycle
}
```

do { ... } while (<*boolean-expression*>);

- **Syntax:**

```
do {
   // This cycle executes at least one time – e.g., txt-based menu
   // We must include code to stop/break the cycle
} while (<boolean-expression>);
```

- **Examples:**

```
do {
   // ...
   i++; // Sometime it will reach 100
} while (i<100);
```

```
do {
   // ...
   if (temp==value) keepdoing=false; // Stop cycle
} while (keepdoing);
```


Exercises

- **Class MyMath:**
  - Create static methods, for calculating factorial of a number, using different cycles provided by Java:
    - double fact_for(int n)
    - double fact_while(int n)
    - double fact_do_hile(int n)
    - double fact_recursive(int n)
  - Create static methods, for calculating absolute value of a number:
    - int abs(int n)
    - long abs(long n)
    - float abs(float n)
    - double abs(double n)
  - Create static method, for calculating power of a base:
    - double pow_iterative(double base, double exponent)
    - double pow_recursive(double base, double exponent)
  - Create static method, for calculating exponential (Nepper: Math.E):
    - double exp_iterative(long exponent)
    - double exp_recursive(double exponent)

Exercises

- Class `MyInstanceofDemo`:
  - Create simple hierarchy of classes

```
class A { int a;}
class B extends A { int b; }
class C extends A { int c; }
```

  - Use `instanceof` to avoid `ClassCastException`

```
A a1 = new A();
if (/.../){a1 = new B();} else {a1 = new C();}
if (a1 instanceof B) {
   //Must cast to B for accessing attribute a1.b
   System.out.println(((B)a1).b);
} else if (a1 instanceof C) {
  //Must cast to C for accessing attribute a1.c
  System.out.println(((B)a1).b);
}
```


Exercises

- Class `MyArraysDemo`:
  - Create two static methods to demo the use of arrays
    - primitive arrays:

```
int[] arrayInts = new int[3];
float[] arrayFloats = new float[2];
```

    - object arrays:

```
Point[] arrayPoints = new Point[3];
Rectangle[] arrayRects = new Rectangle[2];
```

- Class `MyArrayListsDemo`:
  - Create two static methods to demo the use of ArrayLists
    - Raw ArraysList (storing any type of Object):

```
ArrayList arrayListObj = new ArrayList();
```

    - Generified ArraysList (storing Strings):

```
ArrayList<String> arrayListStr  = new ArrayList();
```

Exercises

- **Class Date:**
  - Create a class Date for storing date objects (day, month, year)
  - Ensure that constructors and sets receive valid parameters (e.g., months are between 1..12, month 2 in [1..28|29], months 11, 4, 9 in [1..30], others [1..31])
  - Include *diferenceDays(Date d)* that returns difference of days between this object and another Date d
  - Include *diferenceMonths(Date d)* that returns difference of months between this object and Date d
  - Include *diferenceYear(Date d)* that returns difference of years between this object and Date d
  - ...

Exercises

- **Class Point:**
  - Create constructor to init coordinates of point
  - Create method move() that moves the point coordinates by dx and dy
  - Create methods distX(), distY() and dist(), for calculating distances between this point and another
  - ...
- **Class Rectangle:**
  - Create constructor to init coordinates of rectangle points (check non-colinear)
  - Create method move() that moves rectangle by dx and dy
  - Create method *isInside(Ponto p)* to checks if point is inside Rectangle
  - ...