

# CS-259

## Data Structures with Java

Basic Syntax and Control Flow



Instructor: **Joel Castellanos**

e-mail: [joel@unm.edu](mailto:joel@unm.edu)

web: <http://cs.unm.edu/~joel/>

Course website:

<http://cs.unm.edu/~joel/cs259/>

```
public class DivisibleBy7
{ public static void main()
  { for (int i=1; i<100; i++)
    { if (i % 7 == 0)
      { System.out.println(i);
      }
    }
  }
}
```

9/30/2009

## A Very Simple Java Program

```
public class HelloWorld
{
  public static void main(String[] args)
  {
    int a = 5;
    int b = 10;

    a = a + b;

    System.out.println(a);
  }
}
```

In Java, all code is **inside** some **class**

**Execution** starts in main()

**Allocate memory** for an **int** variable named **a**, and **Initialize** that memory to the value 5.

- 1) **Gets** the value stored in memory named **a**.
- 2) **Gets** the value stored in memory named **b**.
- 3) **Adds** these two values.
- 4) **Overwrites** whatever value was in **a** with the newly calculated sum: **a+b**.

**Output** current value of **a**

2

## Java's Primitive Types

- byte:** 8-bit, [-128, 127].
- short:** 16-bit, [-32,768, 32,767] .
- int:** 32-bit, [-2,147,483,648, 2,147,483,647].
- long:** 64-bit, [-9,223,372,036,854,775,808, 9,223,372,036,854,775,807].
- float:** 32-bit, [1.4x10<sup>-45</sup>, 3.4028235x10<sup>38</sup>]
- double:** 64-bit, [4.9x10<sup>-324</sup>, 1.7976931348623157x10<sup>308</sup>]
- boolean:** Only two possible values: **true** and **false**.
- char:** 16-bit, ['\u0000' (0), '\uffff' (65,535) ].

3

## Basic Syntax: Part 1

```
public class HelloWorld
{ public static void main(String[] args)
  {
    final double PROB_WIN = 18/38;
    final double PROB_LOSE = 20.0/38.0;
    System.out.println("Probabilities: Win=" +
      PROB_WIN + ", Lose=" + PROB_LOSE);
  }
}
```

The System class is a package or a library that contains many useful methods and fields

method call: `println( )`

block structure: { } with correct indenting

Statements end with a ;

4

## Basic Syntax: Part 2

```
public class HelloWorld
{ public static void main(String[] args)
  {
    final double PROB_WIN = 18/38;
    final double PROB_LOSE = 20.0/38.0;

    System.out.println("Probabilities: Win=" +
      PROB_WIN + ", Lose=" + PROB_LOSE);
  }
}
```

Annotations:

- int literals (pointing to 18 and 38 in the first line)
- double literals (pointing to 20.0 and 38.0 in the second line)
- String literal (pointing to "Probabilities: Win=" in the print statement)

5

## Basic Syntax: Part 3

```
public class HelloWorld
{ public static void main(String[] args)
  {
    final double PROB_WIN = 18/38;
    final double PROB_LOSE = 20.0/38.0;

    System.out.println("Probabilities: Win=" +
      PROB_WIN + ", Lose=" + PROB_LOSE);
  }
}
```

Annotations:

- arithmetic operator (pointing to / in the first line)
- concatenation operator  
The + operator is *overloaded* (pointing to + in the print statement)

6

## What is the Output?

```
public class HelloWorld
{ public static void main(String[] args)
  {
    final double PROB_WIN = 18/38;
    final double PROB_LOSE = 20.0/38.0;

    System.out.println(
      "Probabilities: Win=" +
      PROB_WIN + ", Lose=" + PROB_LOSE);
  }
}
```

```
Probabilities: Win=0.0, Lose=0.5263157894736842
```

7

## Short-Cut Operators: ++, +=

```
public class HelloWorld
{ public static void main(String[] args)
  { int a = 5;
    int b = 3;
    int c = a+b;
    a++; //same as a=a+1;
    b += 10; //same as b=b+10;
    System.out.println("a="+a);
    System.out.println("b="+b);
    System.out.println("c="+c);
  }
}
```

Output: a=6  
b=13  
c=8

Notice that c is the sum of what a and b **were** when the calculation was preformed.

8

## Program State

- The State of a single threaded program is
  - The value of the execution pointer, and
  - The value of all memory accessible by the program at that execution moment.

```
1. public class HelloWorld
2. { public static void main(String[] args)
3.   { int x = 5; int a = 0;
4.     if (x < 10) a=1;
5.     if (x < 6) a=2;
6.     System.out.println(a);
7.   }
8. }
```

| line | x | a |
|------|---|---|
| 5)   | 5 | 1 |

9

## Control Flow and Program State: if

```
1. public class HelloWorld
2. { public static void main(String[] args)
3.   { int x = 5;
4.     int a = 0;
5.     if (x < 10) a=1;
6.     if (x < 6) a=2;
7.     if (x < 1) a=3;
8.     System.out.println(a);
9.   }
10. }
```

| line | x | a |
|------|---|---|
| 3)   |   |   |
| 4)   | 5 |   |
| 5)   | 5 | 0 |
| 6)   | 5 | 1 |
| 7)   | 5 | 2 |
| 8)   | 5 | 2 |

Table of **program state** at the **start** of each line in the order of execution.

Output:  
2

10

## Control Flow and Program State: if

```
1. public static void main(String[] args)
2. { int x = 5;
3.   int a = 0;
4.   if (x < 10)
5.     { a=1;
6.     }
7.   if (x < 6)
8.     { a=2;
9.     }
10.  if (x < 1)
11.    { a=3;
12.    }
13.  System.out.println(a);
14. }
```

| line | x | a |
|------|---|---|
| 2)   |   |   |
| 3)   | 5 |   |
| 4)   | 5 | 0 |
| 5)   | 5 | 0 |
| 7)   | 5 | 1 |
| 8)   | 5 | 1 |
| 10)  | 5 | 2 |
| 13)  | 5 | 2 |

Output:  
2

11

## Control Flow: if and else if

```
1. public static void main(String[] args)
2. { int x = 5;
3.   int a = 0;
4.   if (x < 10)
5.     { a=1;
6.     }
7.   else if (x < 6)
8.     { a=2;
9.     }
10.  else if (x < 1)
11.    { a=3;
12.    }
13.  System.out.println(a);
14. }
```

| line | x | a |
|------|---|---|
| 2)   |   |   |
| 3)   | 5 |   |
| 4)   | 5 | 0 |
| 5)   | 5 | 0 |
| 13)  | 5 | 1 |

Output:  
1

12

## What is the Output?

```
public class HelloWorld
{ public static void main(String[] args)
  { boolean cool = false;
    if (cool = true)
    { System.out.print("BAD: ["+cool);
    }
    if (cool = false)
    { System.out.print("XOX: ["+cool);
    }
    System.out.println("] -> END: ["+cool +"]");
  }
}
```

BAD: [true] -> END: [false]

¿What is going on?

13

## Integers Divisible by 7 – Explicit Prints

```
public class IntegersDivisibleBy7v1
{ public static void main(String[] args)
  { //Print the integers less than 100 that are divisible by 7.
    System.out.println("7");
    System.out.println("14");
    System.out.println("21");
    System.out.println("28");
    System.out.println("35");
    ...
  }
}
```

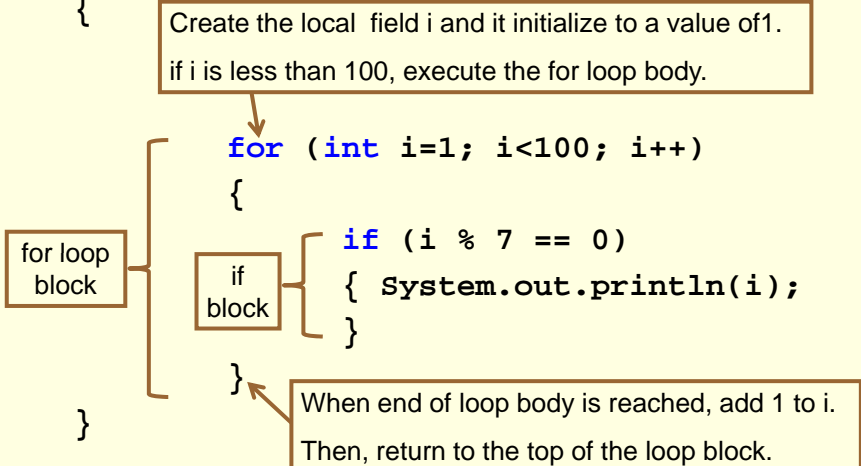
14

## Properties of Explicit Print Solution

- The code is long and repetitive.
- The programmer must know all parts of the solution.
- Difficult to modify the functionality.
  - For example, to change the program from printing all integers less than 100 that are divisible by 7, to all integers less than 100 divisible by 11, the programmer would need to change almost every line of the code.
- Not a scalable solution.

15

## Integers Divisible by 7 – for loop & if

```
public class IntegersDivisibleBy7v2
{
    public static void main(String[] args)
    {
        
        for (int i=1; i<100; i++)
        {
            if (i % 7 == 0)
            {
                System.out.println(i);
            }
        }
    }
}
```

16

## Integers Divisible by 7 – for loop & if

```
public class IntegersDivisibleBy7v2
{
    public static void main(String[] args)
    {
        for (int i=1; i<100; i++)
        {
            if (i % 7 == 0)
            {
                System.out.println(i);
            }
        }
    }
}
```

for loop block

if block

logical expression

This block is executed if the logical expression evaluates to true.

17

## Integers Divisible by 7 – for loop & if

```
public class IntegersDivisibleBy7v2
{
    public static void main(String[] args)
    {
        for (int i=1; i<100; i++)
        {
            if (i % 7 == 0)
            {
                System.out.println(i);
            }
        }
    }
}
```

1. The code is short.
2. The programmer need not know all parts of the solution.
3. Easy to modify.
4. Scalable.
5. However, i walks through many unneeded values (values that are not divisible by 7).

Compile and Run!

18

## Integers Divisible by 7 – Leverage Pattern

```
public class IntegersDivisibleBy7v3
{ public static void main(String[] args)
  { for (int i=7; i<100; i=i+7)
    { System.out.println(i);
    }
  }
}
```

Compile  
and Run!

1. The code is even shorter.
2. The programmer need not know all parts of the solution.
3. Easy to modify.
4. Scalable.
5. Leverages a pattern to skip values of i that are not needed.

19