

# Chapter 2 Basic Computation

## Part 1 Identifiers, Assignments, Expressions, and Constants

Dr. Nour Almadhoun Alserr

Week 4

1/10/2023





# Objectives

---

- Write Java statements to **declare variables** and define named constants.
- Write **assignment statements** and **expressions** containing variables and constants.

# Java identifiers

- **Identifier:** the name of something in a Java program, such as a **variable**, **class**, or **method**.
- It **must not** start with a digit and may contain:
  - Letters,
  - Digits 0 through 9,
  - The underscore character (\_)
  - The symbol \$ is also allowed (reserved)
- Since Java is *case sensitive*, **iug**, **Iug**, and **IUG** are different identifiers.

# Java identifiers

```
public class Test
{
    public static void main(String[ ] args)
    {
        int a = 20;
    }
}
```

# Java identifiers

```
public class Test
{
    public static void main(String[] args)
    {
        int a= 20;
    }
}
```

# Java identifiers

```
public class EggBasket
{
    public static void main (String [] args)
    {
        int numberOfBaskets, eggsPerBasket, totalEggs;
        numberOfBaskets = 10;
        eggsPerBasket = 6;
        totalEggs = numberOfBaskets * eggsPerBasket;
        System.out.println ("If you have");
        System.out.println (eggsPerBasket + " eggs per basket and");
        System.out.println (numberOfBaskets + " baskets, then");
        System.out.println ("the total number of eggs is " + totalEggs);
    }
}
```

# Java identifiers

## ■ What are the valid identifiers ?!

miles

public

int

a++

\$4

Four\_Students\$2021

Test

apps

4#R

class

x

radius

Four Students\$2021

--a

#44

y

apps

# Keywords or Reserved Words

- Words such as **if** are called ***keywords*** or ***reserved words*** and have **special, predefined** meanings.
  - Cannot be used as identifiers.

|          |              |           |            |
|----------|--------------|-----------|------------|
| abstract | continue     | for       | new        |
| switch   | assert       | default   | goto       |
| package  | synchronized | boolean   | do         |
| if       | private      | this      | break      |
| double   | implements   | protected | throw      |
| byte     | else         | import    | public     |
| throws   | case         | enum      | instanceof |
| return   | transient    | catch     | extends    |
| int      | short        | try       | char       |
| final    | interface    | static    | void       |
| class    | finally      | long      | strictfp   |
| volatile | const        | float     | native     |
| super    | while        |           |            |

# Where to Declare Variables

- Just **before using it**, or
- At the **beginning** of the section of your program (**main method**) that is enclosed in **{ }**.

```
public static void main(String[] args)
{ /* declare variables here */
    . . .
}
```

# Assignment Statements

- An assignment statement is used to **assign** a value to a variable.

Variable

=

Value

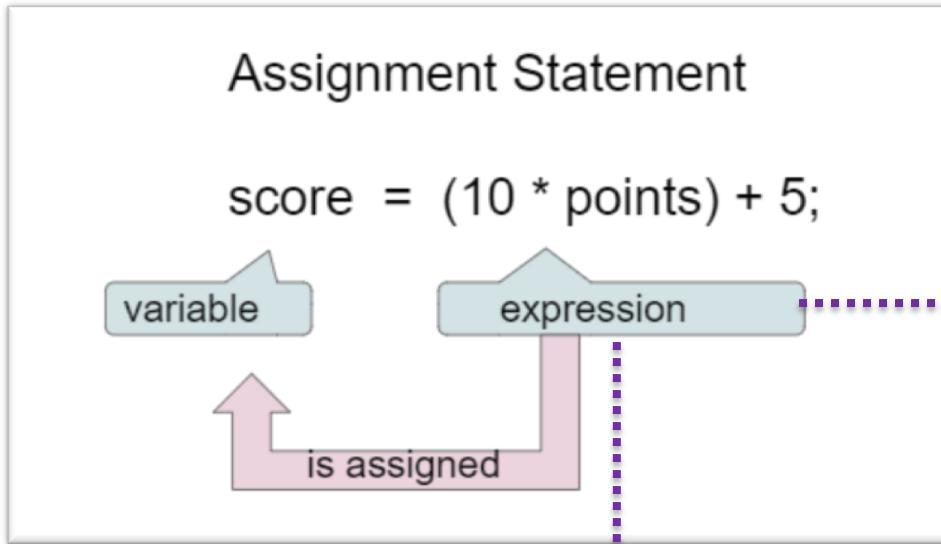
courseGrade = 99



*assignment operator*

# Assignment Statements

## Syntax



- Another **variable**.
- **Literal** or **constant** (e.g., **number**).
- Variables and literals with **operators** (+,-).

# Assignment Examples

---

```
amount = 3.99;  
firstInitial = 'W' ;  
FinalGrade = midterm + FinalExam;  
score = numberOfCards + handicap;  
eggsPerBasket = eggsPerBasket - 2;  
byte = 8;
```

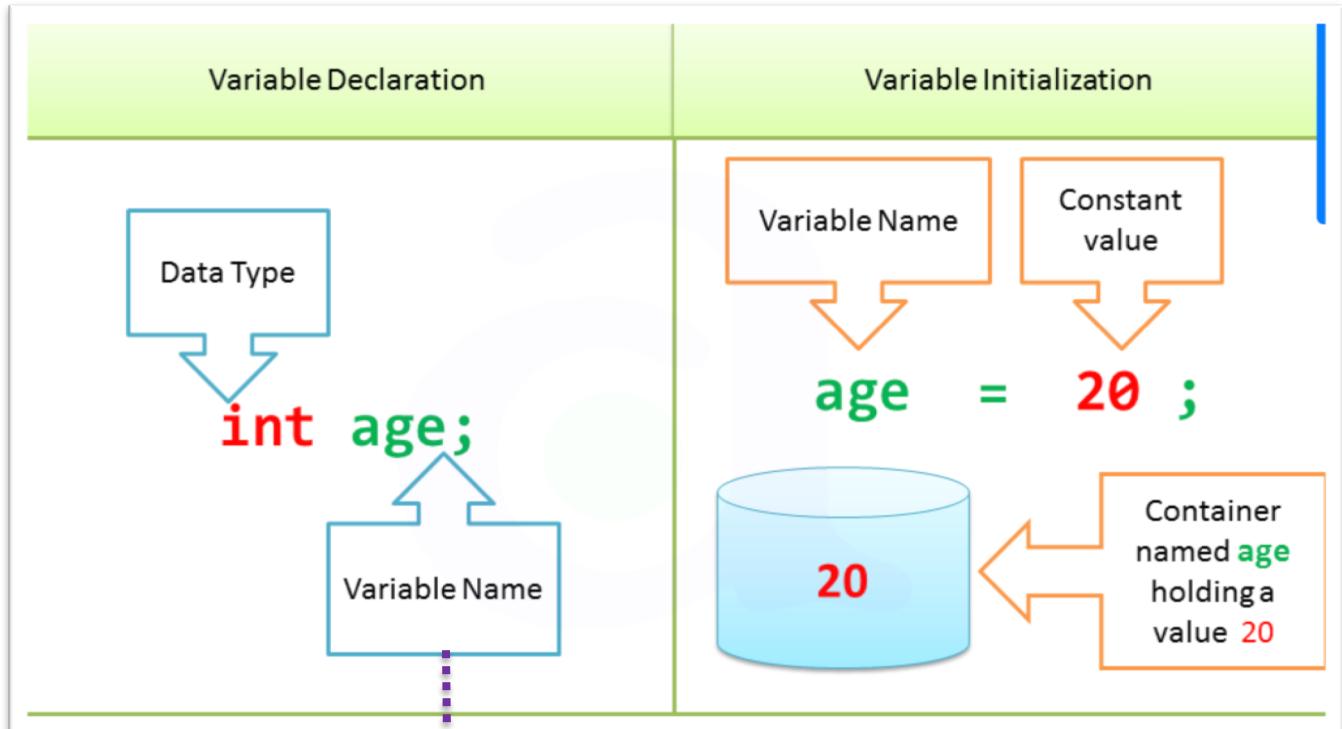
# Assignment Statements

```
1
2 public class VariableAssignment
3 {
4     public static void main(String[] args)
5     {
6         int x = 3;
7         int y = 2;
8         System.out.println(x);
9         System.out.println(y);
10        x = y;
11        System.out.println(x);
12        System.out.println(y);
13        y = 5;
14        System.out.println(x);
15        System.out.println(y);
16    }
17 }
18 }
```

# Assignment Statements

```
public class ErrorSwap
{
    public static void main(String[] args)
    {
        int h = 3;
        int w = 5;
        System.out.println(h);    //3
        System.out.println(w);    //5
        h = w;
        w = h;
        System.out.println(h);    //expected 5
        System.out.println(w);    //expected 3
    }
}
```

# Initializing Variables



***Uninitialized***  
(default value)

# Initializing Variables

- To protect against an **uninitialized variable** (and to keep the compiler **happy** 😊) assign a value at the time the variable is declared.
- Examples:

```
int count = 0;
```

```
char grade = 'A' ;
```

```
int a =10, b=20, c=30;
```

# Sample Program

```
import java.util.Scanner;
public class EggBasket2
{
    public static void main (String [] args)
    {
        int numberofBaskets, eggsPerBasket, totalEggs;
        Scanner keyboard = new Scanner (System.in);
        System.out.println ("Enter the number of eggs in each basket:");
        eggsPerBasket = keyboard.nextInt ();
        System.out.println ("Enter the number of baskets:");
        numberofBaskets = keyboard.nextInt ();
        totalEggs = numberofBaskets * eggsPerBasket;
        System.out.println ("If you have");
        System.out.println (eggsPerBasket + " eggs per basket and");
        System.out.println (numberofBaskets + " baskets, then");
        System.out.println ("the total number of eggs is " + totalEggs);
        System.out.println ("Now we take two eggs out of each basket.");
        eggsPerBasket = eggsPerBasket - 2;
        totalEggs = numberofBaskets * eggsPerBasket;
        System.out.println ("You now have");
        System.out.println (eggsPerBasket + " eggs per basket and");
        System.out.println (numberofBaskets + " baskets.");
        System.out.println ("The new total number of eggs is " + totalEggs);
    }
}
```

# Sample Program (Output)

Enter the number of eggs in each basket:

6

Enter the number of baskets:

10

If you have

6 eggs per basket and

10 baskets, then

the total number of eggs is 60

Now we take two eggs out of each basket.

You now have

4 eggs per basket and

10 baskets.

The new total number of eggs is 40

# Constants

- Literal expressions such as 2, 3.7, or 'y' are called *constants*.
- Integer constants can be preceded by a + or – sign (no commas)
  - +2 , +199 , -5147483648 , 2.71828 , 29002 , 0
- Floating-point constants can be written
  - With digits after a decimal point or
  - Using *e notation*.

# e Notation

## Scientific Notation

Move the decimal point to the left.

$$4500 \rightarrow 4.5 \times 10^3$$

positive

$$4.5e3 = 45e3$$

Move the decimal point to the right.

$$0.0677 \rightarrow 6.77 \times 10^{-2}$$

negative

$$6.77e-2$$

# Floating-point

- Floating-point numbers often are only **approximations** since they are stored **with a finite number of bits**.
- $1.0/3.0$  is slightly less than  $1/3$ .
- $1.0/3.0 + 1.0/3.0 + 1.0/3.0$  is less than  $1$ .

## The IEEE-754 floating-point standard

+7.432 x 10<sup>48</sup>

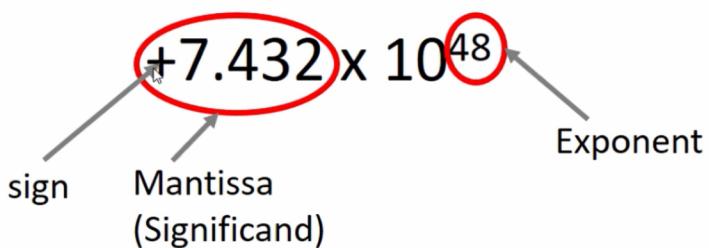
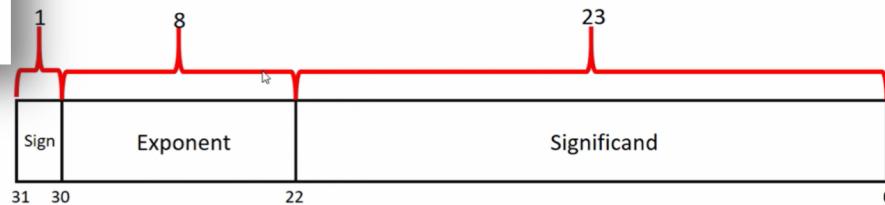


Diagram illustrating the IEEE-754 floating-point representation of the number +7.432 x 10<sup>48</sup>. The number is shown in scientific notation with the sign (+), mantissa (7.432), and exponent (48). Arrows point from the labels 'sign', 'Mantissa (Significand)', and 'Exponent' to their respective parts of the number.

## The IEEE-754 floating-point standard

- Single precision → 32 bits representation



# Named Constants

- Java provides mechanism to ...
  - Define a **variable**
  - **Initialize** it
  - **Fix** the **value** so it cannot be changed

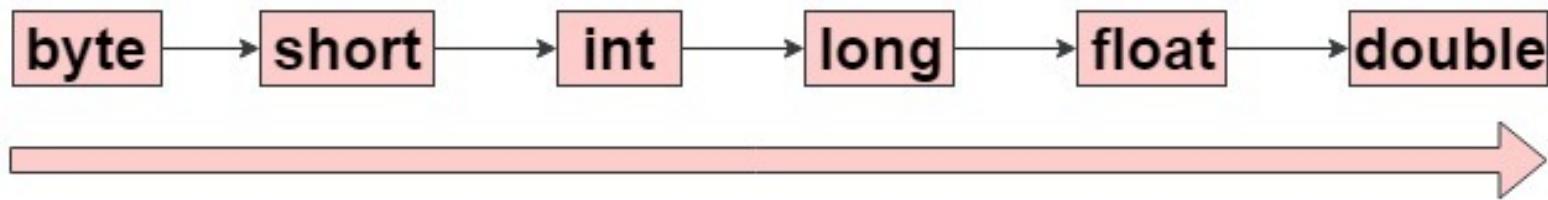
```
public static final Type Variable = Constant;
```

- Example:

```
public static final double PI = 3.14159;
```

# Type Casting (1/2)

## Automatic Type Conversion (Widening - implicit)



- When we assign value of a **smaller** data type to a **bigger** data type.

# Type Casting (1/2)

```
class Test
{
    public static void main(String[] args)
    {
        int i = 100;

        //automatic type conversion
        long l = i;

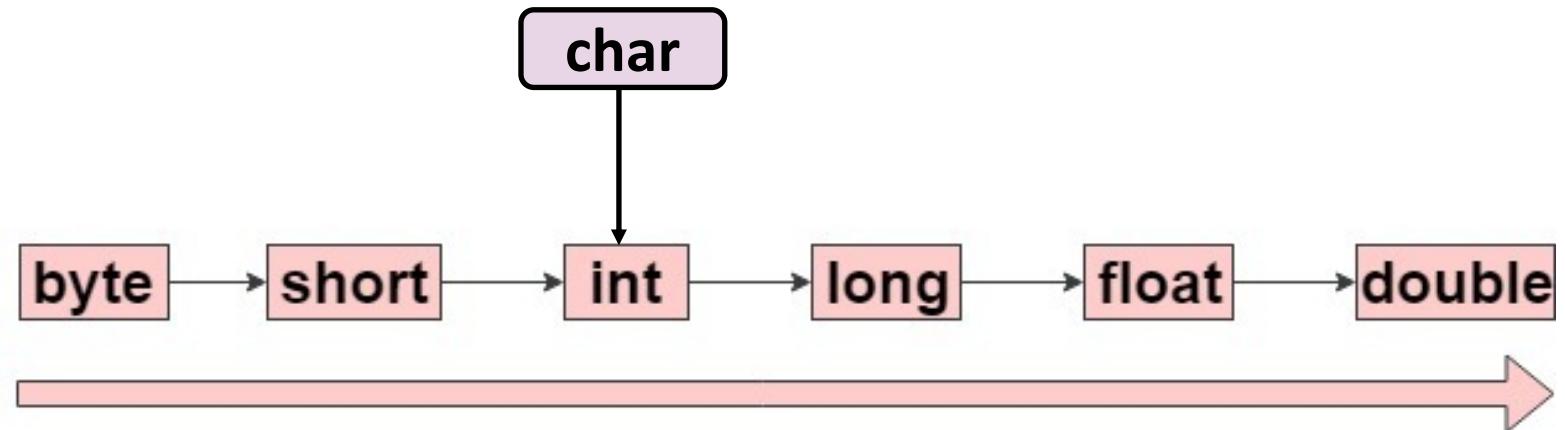
        //automatic type conversion
        float f = l;
        System.out.println("Int value "+i);
        System.out.println("Long value "+l);
        System.out.println("Float value "+f);
    }
}
```

## Output

Int value 100  
Long value 100  
Float value 100.0

# Type Casting (1/2)

- You can assign a value of type **char** to a variable of type **int**.



# Type Casting (1/2)

- You can assign a value of type **char** to a variable of type **int**.

```
class CharToInt {
    public static void main(String[] args) {
        char x = '8';
        int z = x;
        System.out.println(x);
    }
}
```

**Output:**

```
class CharToInt {
    public static void main(String[] args) {
        char x = '88';
        int z = x;
        System.out.println(x);
    }
}
```

**Output:**

```
class CharToInt {
    public static void main(String[] args) {
        char x = 'A';
        int z = x;
        System.out.println(x);
    }
}
```

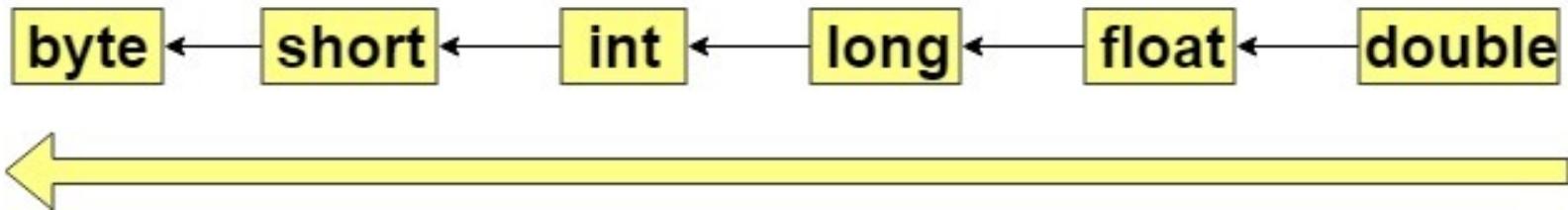
**Output:**

```
class CharToInt {
    public static void main(String[] args) {
        char x = '8';
        int z = (int)x;
        System.out.println(x);
    }
}
```

**Output:**

# Type Casting (2/2)

## Narrowing (explicit)



- If we want to assign a value of **larger** data type to a **smaller** data type.

# Type Casting (2/2)

```
//Java program to illustrate explicit type conversion
class Test
{
    public static void main(String[] args)
    {
        double d = 100.04;

        //explicit type casting
        long l = (long)d;

        //explicit type casting
        int i = (int)l;
        System.out.println("Double value "+d);

        //fractional part lost
        System.out.println("Long value "+l);

        //fractional part lost
        System.out.println("Int value "+i);
    }
}
```

## Output

```
Double value 100.04
Long value 100
Int value 100
```

# Type Casting (2/2)

```
class PrepBytes {  
    public static void main(String args[])  
    {  
        double data = 52.6345;  
        System.out.println("Double - " + data);  
  
        int value = (int) data;  
  
        System.out.println("Integer - " + value);  
    }  
}
```

## Output:

```
Double - 52.6345  
Integer - 52
```

- Any nonzero value to the right of the decimal point is ***truncated*** rather than ***rounded***.

# Type Casting

```
class Int.ToDouble {
    public static void main(String[] args) {
        int x = 11;
        int y = 3;
        int z = x/y;
        System.out.println(z);
    }
}
```

Output:

```
class Int.ToDouble {
    public static void main(String[] args) {
        int x = 11;
        int y = 3;
        double| z = x/y;
        System.out.println(z);
    }
}
```

Output:

# Type Casting

```
class Int.ToDouble {
    public static void main(String[] args) {
        double x = 11.0;
        double y = 3.0;
        double z = x/y;
        System.out.println(z);
    }
}
```

Output:

```
class DoubleToInt {
    public static void main(String[] args) {
        double x = 11.0;
        double y = 3.0;
        int z = (int)(x/y);
        System.out.println(z);
    }
}
```

Output: