# AP Computer Science A

Java Programming Essentials [Ver. 2.0]

## Unit 1: Elementary Programming

WEEK 2: CHAPTER 2 ELEMENTARY PROGRAMMING – DATA TYPES AND NUMBER SYSTEM

DR. ERIC CHOU IEEE SENIOR MEMBER

## Objectives

**Variables**

- Storage of Data – Type – ID and Value
- Identifier
- Variable Declaration and Initialization
- Naming Convention

## Objectives

**Data Types**

- Primitive Data Versus Reference Data Type
- Constants
- Integer Number
- Real Number

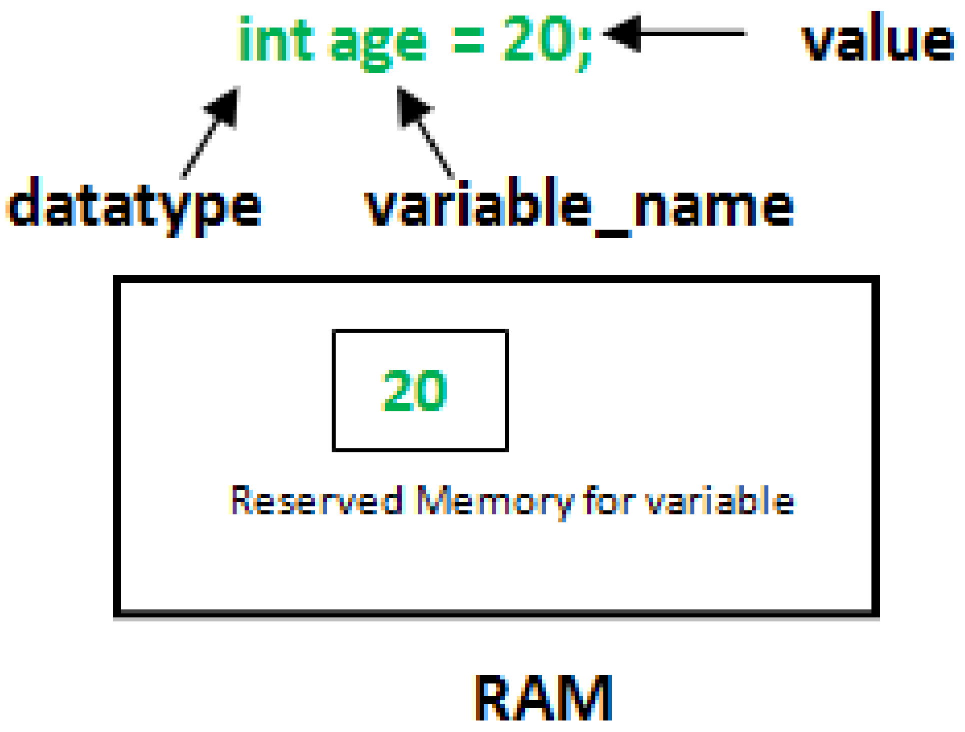
## Objectives

**Reference Data Types – Basic Class and Objects**

- Storage of Data – Type – ID and Value
- Identifier
- Variable Declaration and Initialization
- Naming Convention

# Variables

LECTURE 1

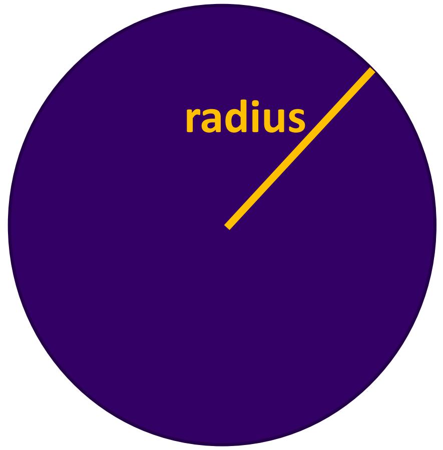

int age = 20;
value
datatype
variable_name
20
Reserved Memory for variable
RAM

# Identifier

LECTURE 1

## Where will see a lot of identifiers

```
public class **Example** {
    public static void main(String[] args){
        // Variable Declaration
        double radius = 5.0;
        // Input part
        Scanner input = new Scanner(System.in);
        radius = input.nextDouble();
        // Processing part
        double area = Math.PI * radius * radius;
        //  Output Part
        System.out.println(area);
    }
}
```



## Identifiers

- Identifiers are the name that identify the elements such as **variables, classes, methods** in a program.
- All identifiers in Java must obey the following rules:
  - An identifier is a sequence of characters that consists of letters, digits underscores (_), and dollar sign ($). An identifier must start with a letter, an underscore (_), or a dollar sign ($). It can not start with a digit (number).
  - An identifier cannot be a reserved word.
  - An identifier cannot be true, false, or null.
  - An identifier can be of any length.

## Variables

Variables are used to represent values that may be changed in the program.

- A variable must be declared before used.
- A variable declaration in syntax: <data type> <variable name>;
- Examples of variable declarations:
    - int count;
    - double radius;
    - double interestRate;

## Variables

Variables are used to represent values that may be changed in the program.

- Primitive data type:

  byte, short, int, long, float, double, char, boolean;

- Reference data type: <class name>
- Multiple instance for variable declaration is allowed in Java:

  datatype variable1, variable2, …, variablen;

- Declaration and assignment in the same statement is also allowed: int count=1;

## Identifiers for Methods

```
public static void main(String[] args){ … }

public double abs(double a){ … }
```

# Variable Declaration

LECTURE 1

## Declaring Variables

```
int x;            // Declare x to be an
                  // integer variable;
double radius;    // Declare radius to
                  // be a double variable;
char a;           // Declare a to be a
                  // character variable;
```

# Assignment Statements

```
x = 1;            // Assign 1 to x;
radius = 1.0;     // Assign 1.0 to radius;
a = 'A';          // Assign 'A' to a;
```

## Variables

```
// Compute the first area
radius = 1.0;
area = radius * radius * 3.14159;
System.out.println("The area is “ + area + " for radius "+radius);
// Compute the second area
radius = 2.0;
area = radius * radius * 3.14159;
System.out.println("The area is “ + area + " for radius "+radius);
```

## Assignment Statement and Assignment Expressions

An assignment statement designates a value for a variable. An assignment statement can be used as an expression in Java.

**Syntax (assignment statement):**

```
<variable> = <expression> ;
```

- An expression represents a computation involving values, variables, and operators that taking them together, evaluates to a value.

```
int y=1;
double radius = 1.0;
int x = 5 * (3 / 2) ;
x = 1;  // correct;
1 = x;  // incorrect;
```

## Assignments

- Evaluate the + sign first, then the assignment.

```
x = 1;
x = x + 1;
```

- If a value is assigned to multiple variables, you can use this syntax:

```
i = j = k = 1;
```

- Which is equivalent to:

```
k = 1;
j = k;
i = j;
```

# Naming Convention

LECTURE 1

## Naming Conventions
(not part of syntax)

- Naming conventions can vary from team to team.  Every programming team may have their own conventions.
- Right here, we are discussing the commonly used naming conventions, but it is not mandatory.

## Naming Conventions

(not part of syntax)

**Variable and Method names:**

- Use lowercase for variables and methods. If a method is longer than one word, the first letter for each word, except the first word, may sometime in uppercase.

```
Example: int aa;
         double women_age;
         int functionName();
         int functionToComputeInterest();
```

**Constant names:**

- Capitalize every letter in a constant and use underscore between words – for example, the constant PI and MAX_VALUE;

## Naming Conventions
(module and package)

**Class names:**

- Capitalize the first letter of each word in the name.
- For example, the class name `ComputeArea`.

**Package names:**

- The whole package name in lower case.

## Java Standard Naming Conventions

**Package Name** - A package should be named in lowercase characters.

**Class Name** - Class names should be nouns in UpperCamelCase.

**Interface Name** - Interface name should start with an uppercase letter and be an adjective.

**Method Name** - Methods should be verbs and in lowerCamelCase.

**Variable Name** - Variable name should in lowerCamelCase.

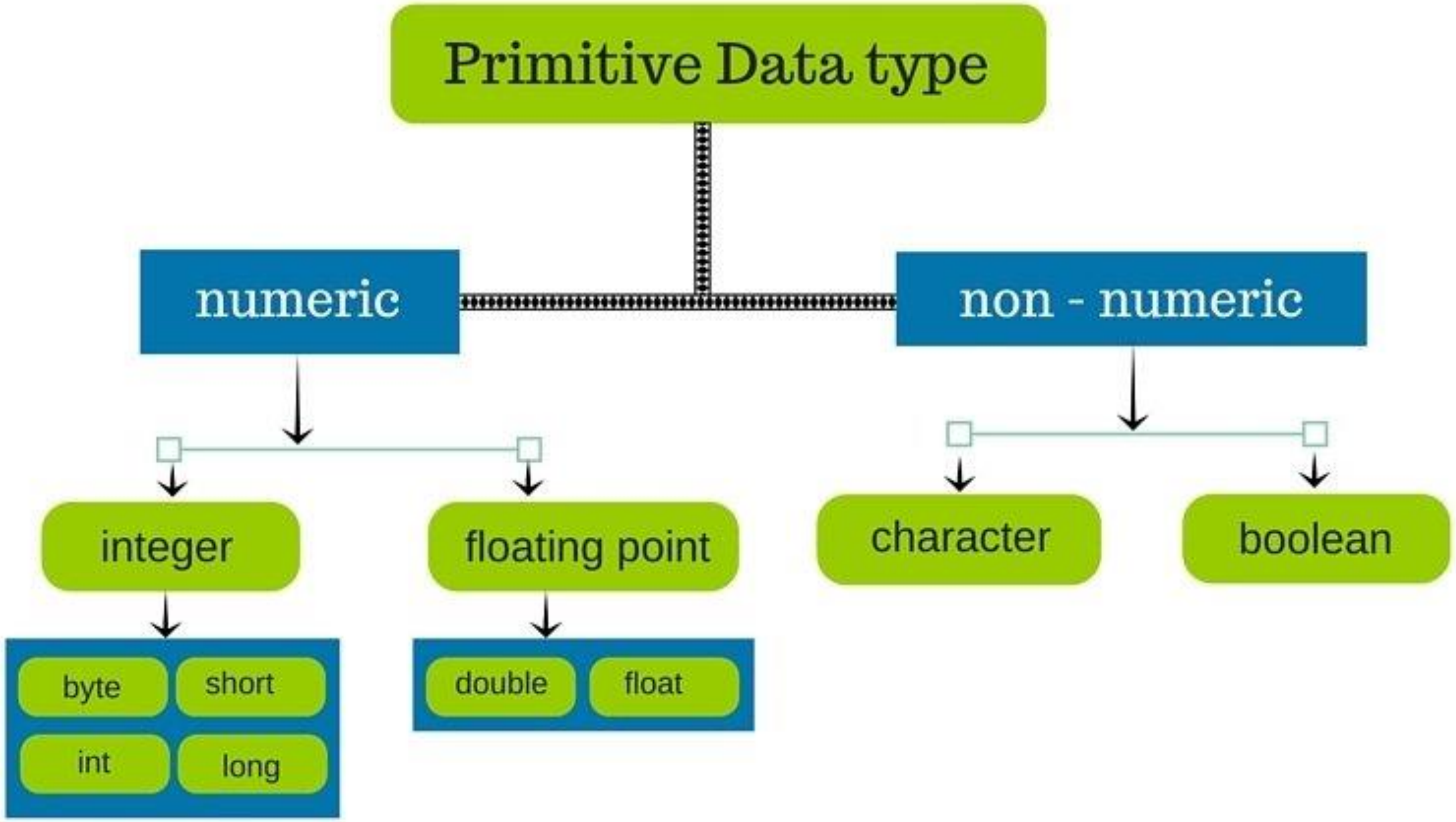
**Constant Variable** - Constant variable names should be written in upper characters separated by underscores.

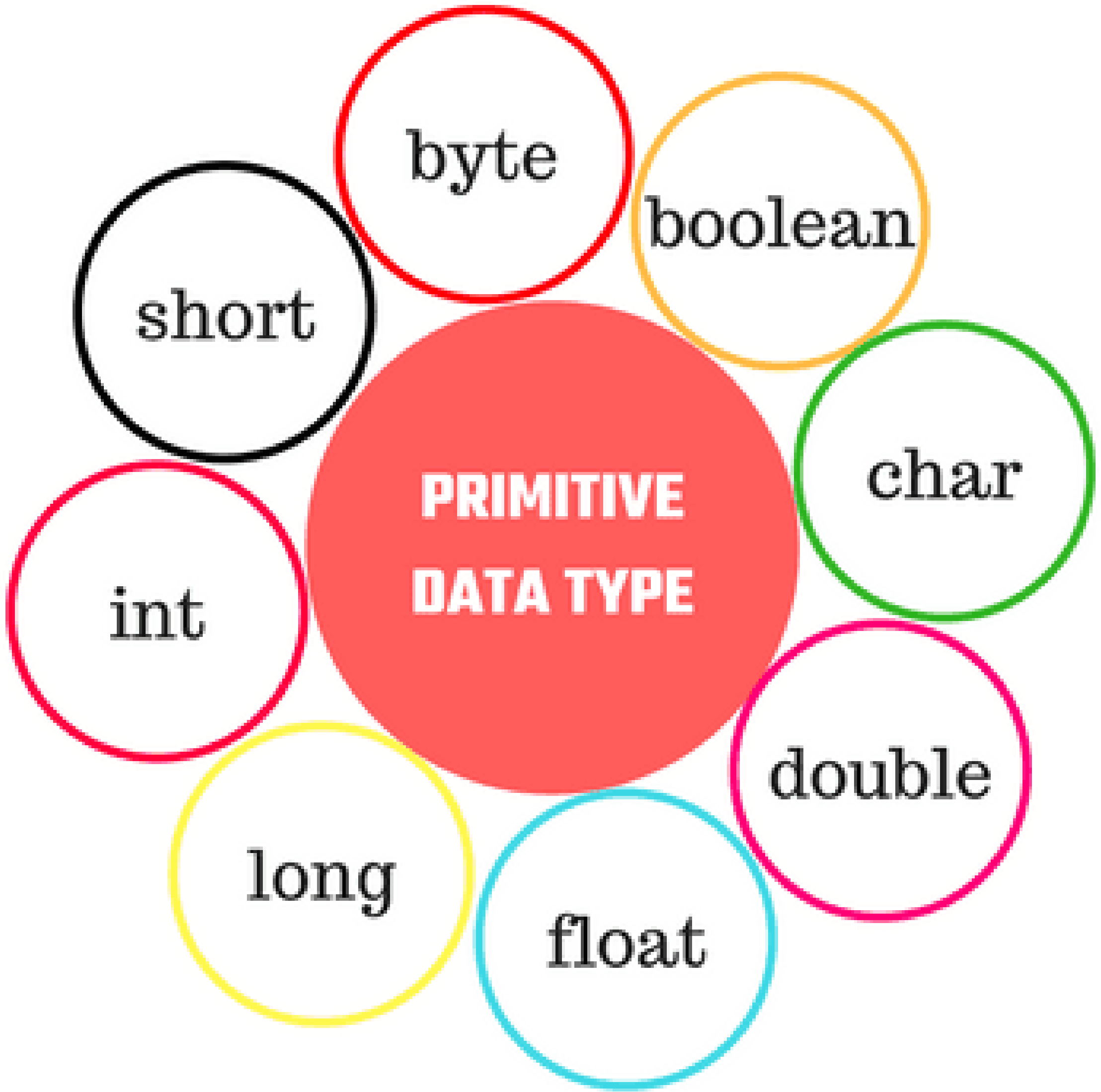
**Abstract Class Name** - Abstract class name must start with Abstract or Base prefix.

**Exception Class Name** - Exception class name must end with Exception suffix

## Data Types

LECTURE 1

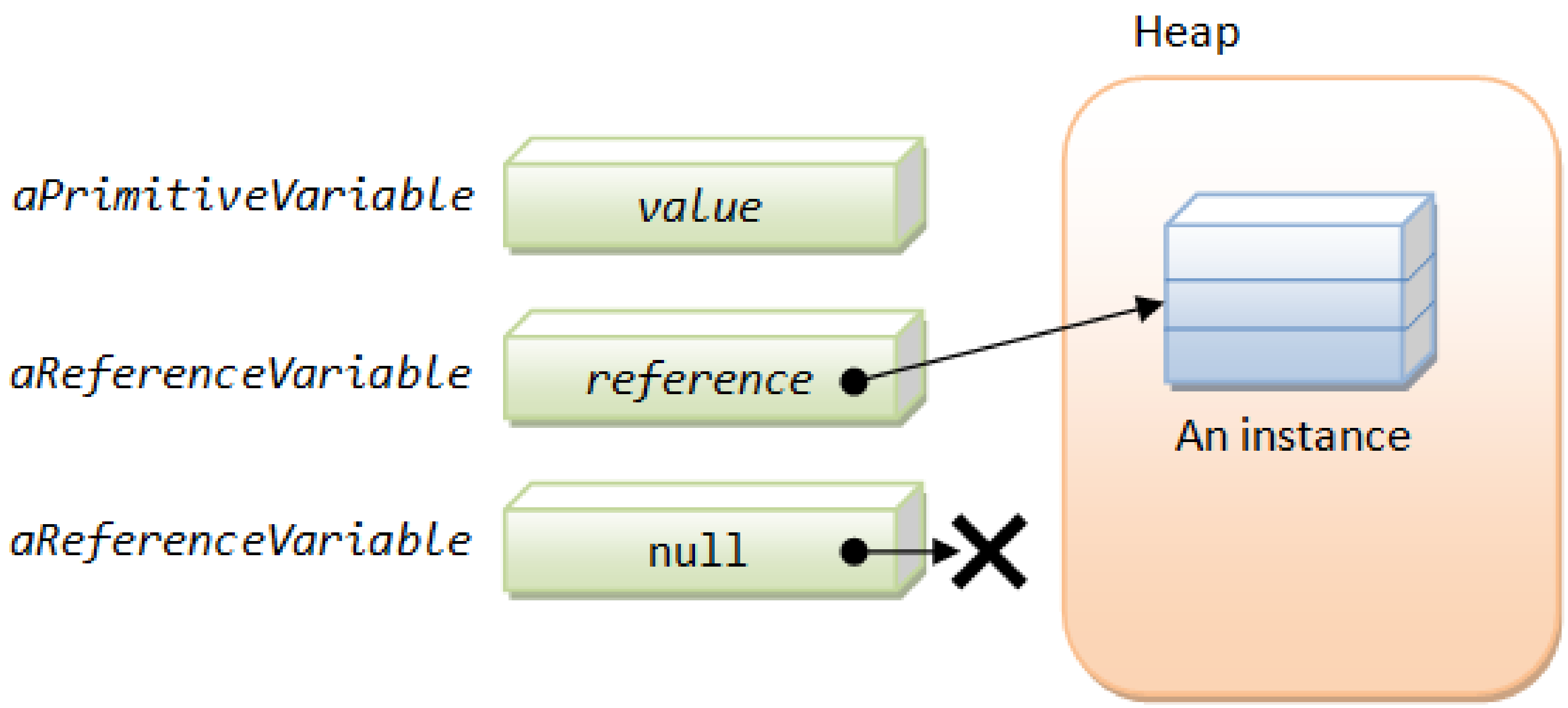

Primitive Data type
numeric
non - numeric
integer
floating point
character
boolean
byte
short
int
long
double
float

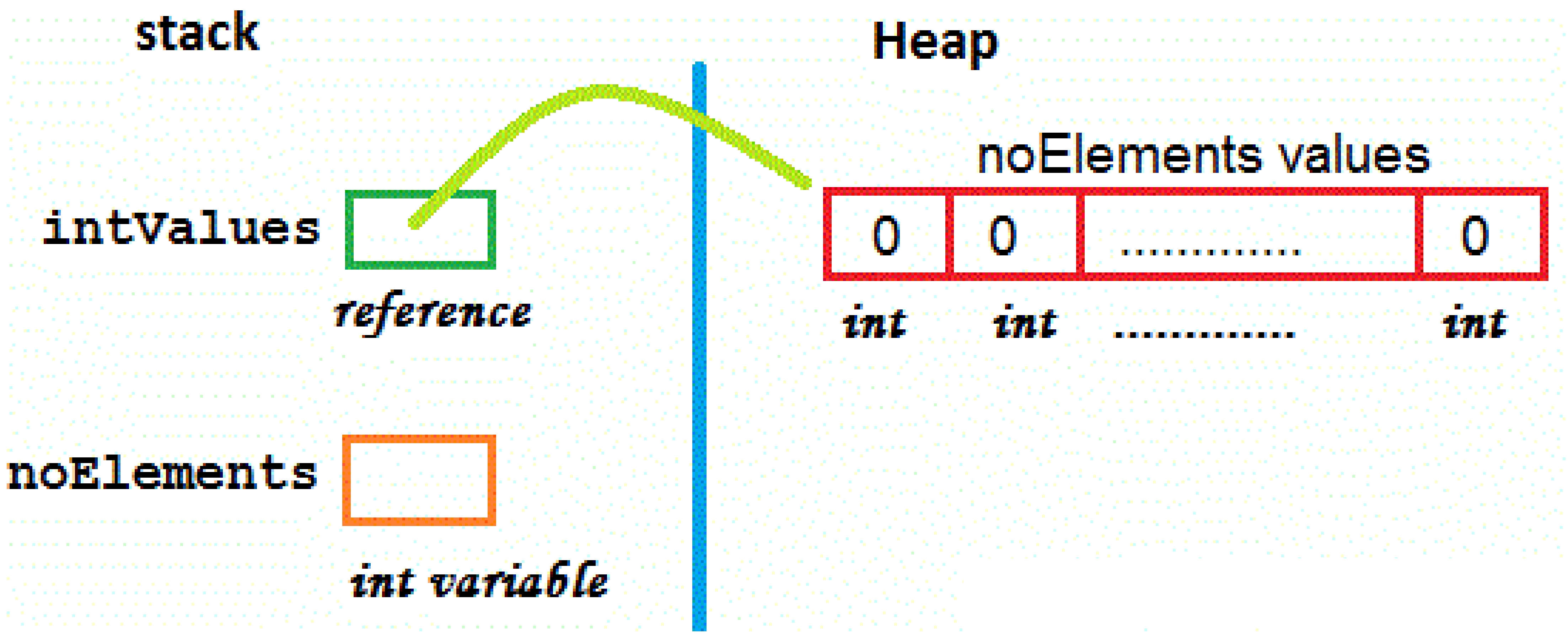

PRIMITIVE DATA TYPE
byte
boolean
char
double
float
long
int
short

# Reference Data Type

- A **reference type** is a data type that's based on a class rather than on one of the primitive types that are built in to the Java language. The class can be a class that's provided as part of the Java API class library or a class that you write yourself.
- **Reference types** are any instantiable class as well as arrays
- String, Scanner, Random, Die, int[], String[], etc.
- **Reference variables** store addresses. (**Address Pointer** is the value of the reference variable.)

# Reference Data Type




stack
Heap
intValues
reference
noElements values
0
0
.............
0
int
int
.............
int
noElements
int variable

## Constants

LECTURE 1

## Declaring constants

- Java does not directly support constants. However, a final variable is *effectively* a constant.
- The **final** modifier causes the variable to be unchangeable
- Java constants are normally declared in ALL CAPS

```
class Math
{
public final double PI=3.14;



}
```

## Named Constants

A named constant is an identifier that represents a permanent value.

Syntax:

```
final <datatype> CONSTANTNAME =  <value> ;
```

The word final is a Java reserved keyword for declaring a constant.

A constant in Java (or most of other language) is usually in all **UPPERCASE**.

**Benefits for using constants:**

(1) you don't have to repeatedly type the same value over over again if it is used multiple times;

(2) if you have to change the constant value, you need to change it only in a single location in the source code; and

(3) a descriptive name for a constant makes the program easier to read.

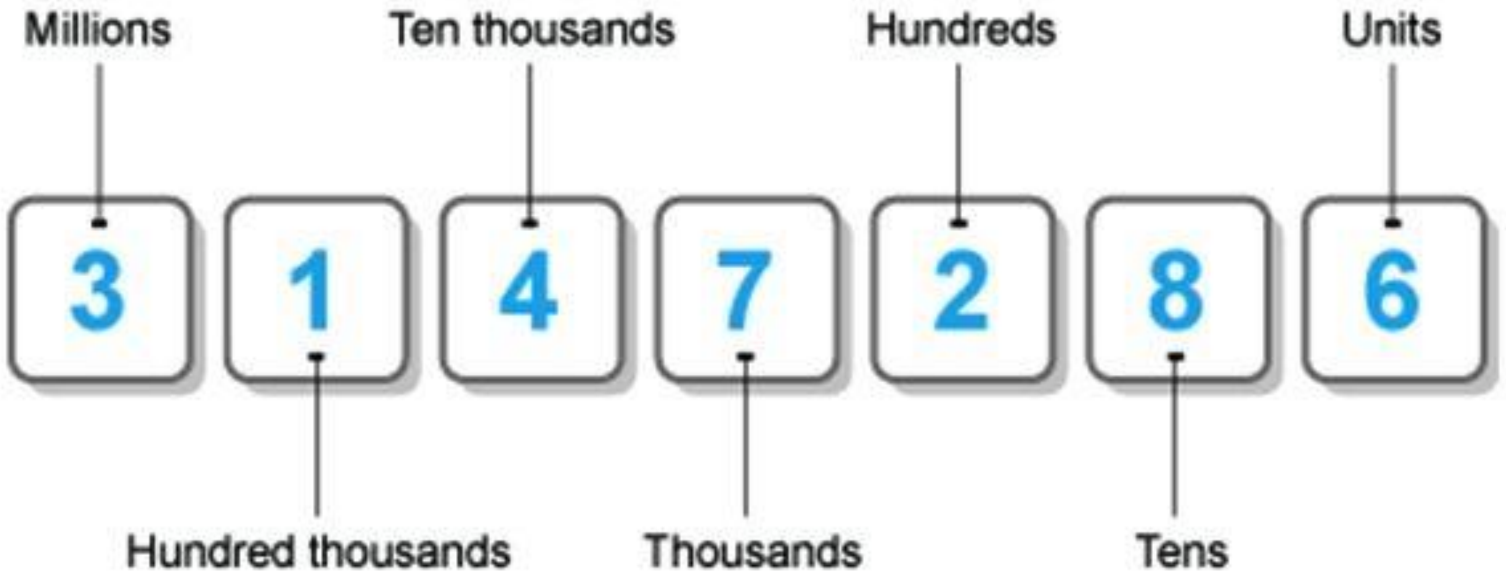
## Named Constants

```
final datatype CONSTANTNAME = VALUE;


final double PI = 3.14159;

final int SIZE = 3;
```

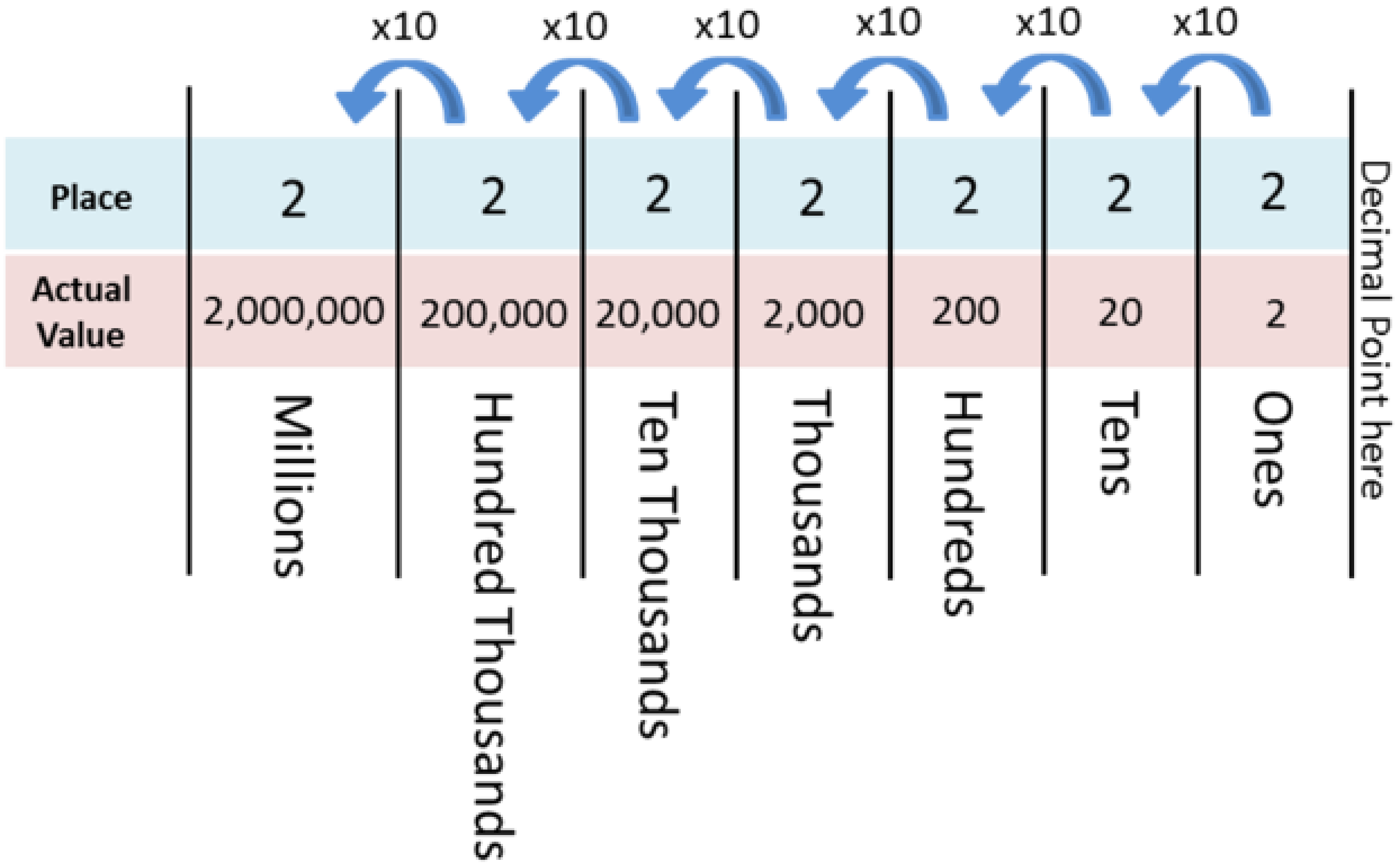
# Number System

LECTURE 1

## Place Value


Millions
Ten thousands
Hundreds
Units
3
1
4
7
2
8
6
Hundred thousands
Thousands
Tens

|  | 3th Place | 2nd Place | 1st Place | Decimal Point | 1st Decimal Place | 2st Decimal Place | 3st Decimal Place | 4st Decimal Place |
|---|---|---|---|---|---|---|---|---|
| Value of place (words) | Hundred | Ten | One | ● | Tenths | Hundredths | Thousandths | Ten Thousandths |
| Value of place (digits) | 100 | 10 | 1 | ● | 1/10 | 1/100 | 1/1 000 | 1/10 000 |


x10
x10
x10
x10
x10
x10
Decimal Point here

| | Millions | Hundred Thousands | Ten Thousands | Thousands | Hundreds | Tens | Ones |
|---|---|---|---|---|---|---|---|
| Place | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| Actual Value | 2,000,000 | 200,000 | 20,000 | 2,000 | 200 | 20 | 2 |

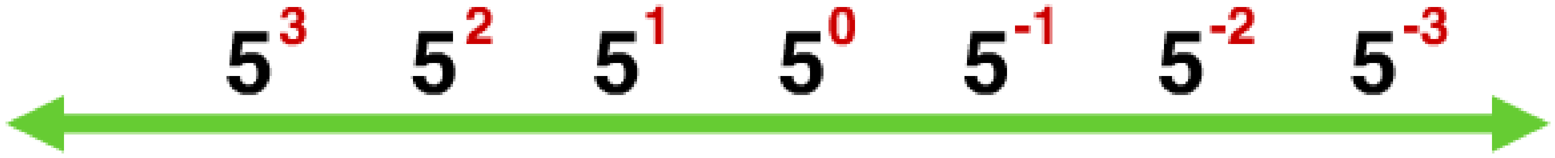
## quinary

A place value Base-5 number system.

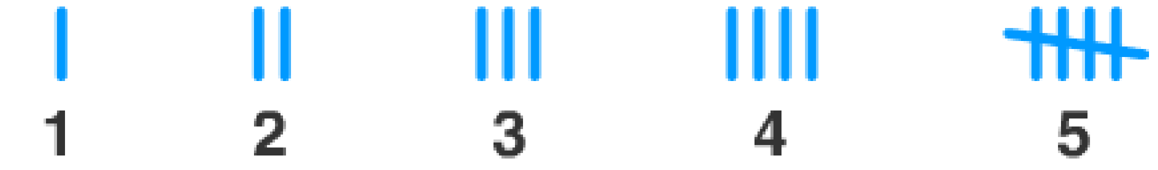
Five digits are used: 0, 1, 2, 3, 4.

## in the quinary system the base number is 5

| $5^5$ | $5^4$ | $5^3$ | $5^2$ | $5^1$ | $5^0$ |
|---|---|---|---|---|---|
| 3125's | 625's | 125's | 25's | 5's | 1's |



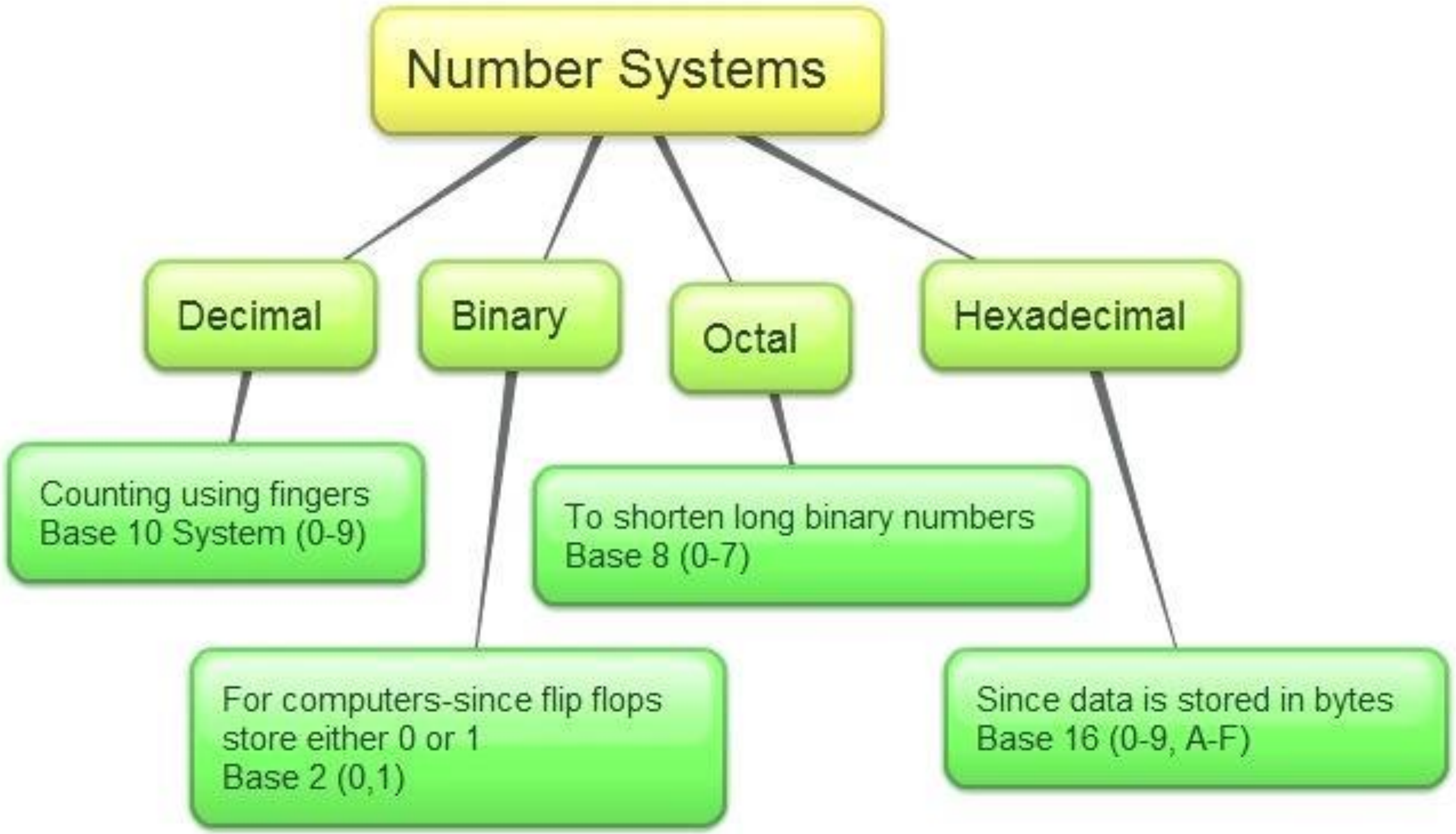
Used in the tally marks system of counting.



$1101001_2$

| Digit | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
|---|---|---|---|---|---|---|---|
| Place value | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |

The place values of the binary number system are powers of 2.

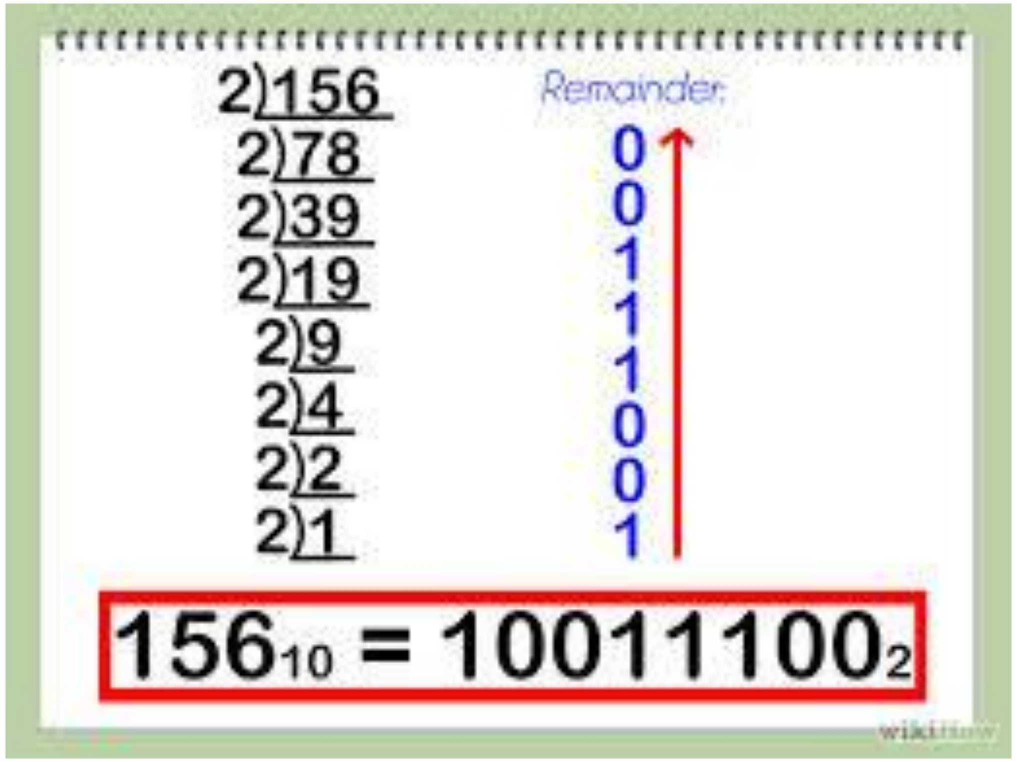

Number Systems
Decimal
Binary
Octal
Hexadecimal
Counting using fingers
Base 10 System (0-9)
To shorten long binary numbers
Base 8 (0-7)
For computers-since flip flops store either 0 or 1
Base 2 (0,1)
Since data is stored in bytes
Base 16 (0-9, A-F)

## Number System Video

# Binary Numbers

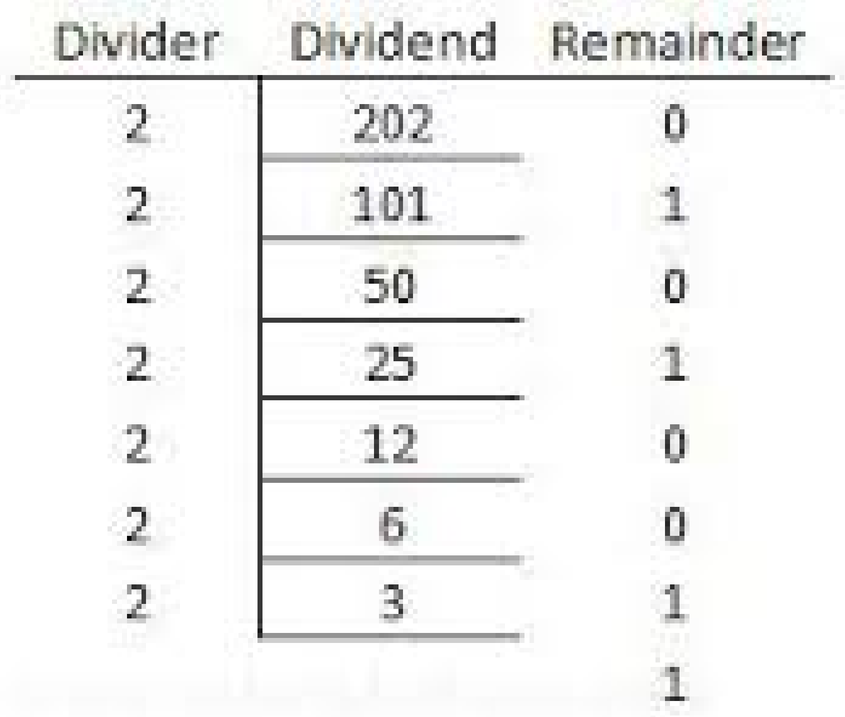
LECTURE 1

# Decimal to Binary



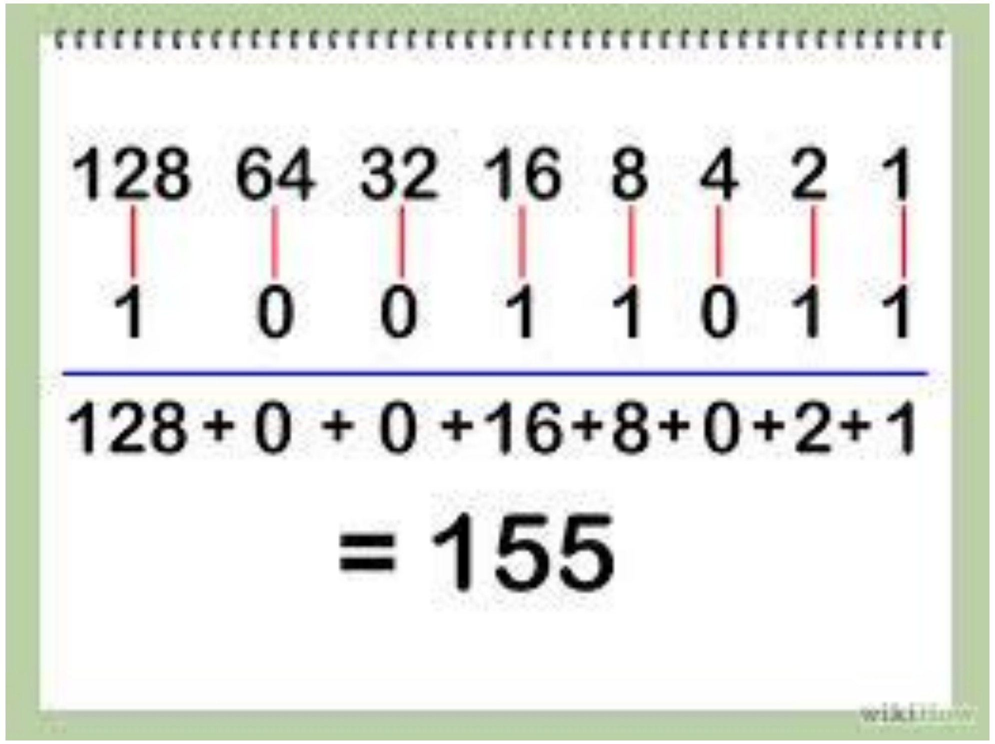
| | Remainder: |
|---|---|
| 2)156 | 0 |
| 2)78 | 0 |
| 2)39 | 1 |
| 2)19 | 1 |
| 2)9 | 1 |
| 2)4 | 0 |
| 2)2 | 0 |
| 2)1 | 1 |

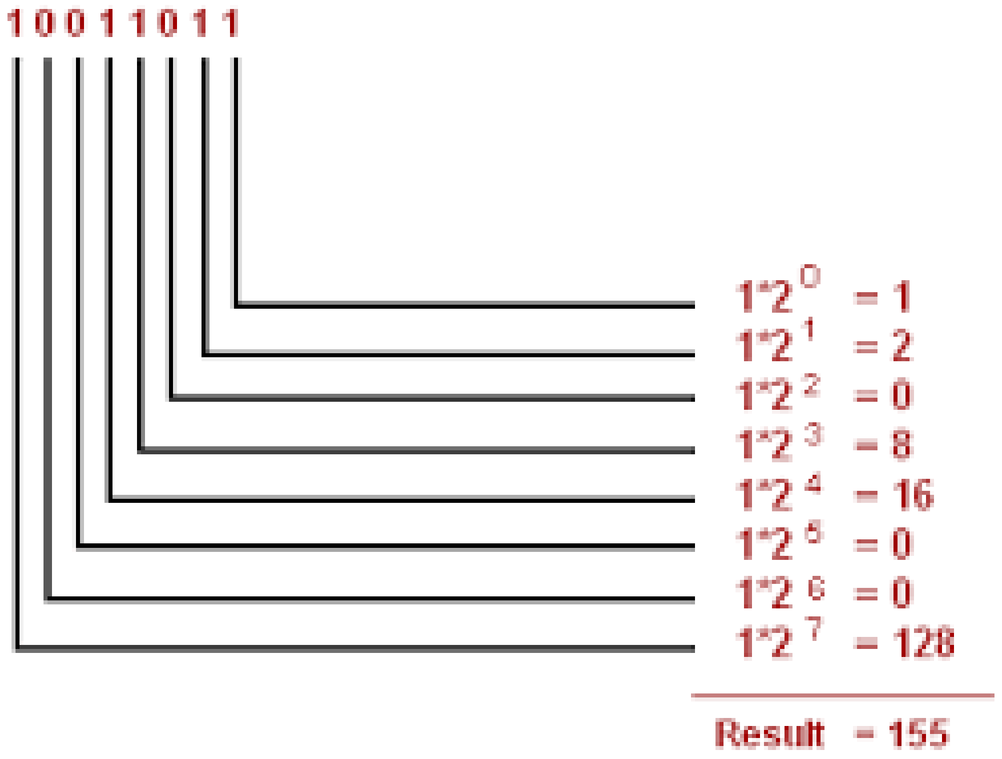
$156_{10} = 10011100_2$



| Divider | Dividend | Remainder |
|---|---|---|
| 2 | 202 | 0 |
| 2 | 101 | 1 |
| 2 | 50 | 0 |
| 2 | 25 | 1 |
| 2 | 12 | 0 |
| 2 | 6 | 0 |
| 2 | 3 | 1 |
| | | 1 |

# Binary/Decimal Conversion





| Binary Value | | | | Decimal Representation | | | | | | | Decimal Value |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 8 | | 4 | | 2 | | 1 | |
| 0 | 0 | 0 | 0 | 0 | + | 0 | + | 0 | + | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | + | 0 | + | 0 | + | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | + | 0 | + | 2 | + | 0 | 2 |
| 0 | 0 | 1 | 1 | 0 | + | 0 | + | 2 | + | 1 | 3 |
| 0 | 1 | 0 | 0 | 0 | + | 4 | + | 0 | + | 0 | 4 |
| 0 | 1 | 0 | 1 | 0 | + | 4 | + | 0 | + | 1 | 5 |
| 0 | 1 | 1 | 0 | 0 | + | 4 | + | 2 | + | 0 | 6 |
| 0 | 1 | 1 | 1 | 0 | + | 4 | + | 2 | + | 1 | 7 |
| 1 | 0 | 0 | 0 | 8 | + | 0 | + | 0 | + | 0 | 8 |
| 1 | 0 | 0 | 1 | 8 | + | 0 | + | 0 | + | 1 | 9 |
| 1 | 0 | 1 | 0 | 8 | + | 0 | + | 2 | + | 0 | 10 |

## Decimal, Binary, Octal, Hexidecimal Values

| Decimal | Binary | Octal | Hexidecimal |
|---|---|---|---|
| 0 | 0000 | 0 | 0 |
| 1 | 0001 | 1 | 1 |
| 2 | 0010 | 2 | 2 |
| 3 | 0011 | 3 | 3 |
| 4 | 0100 | 4 | 4 |
| 5 | 0101 | 5 | 5 |
| 6 | 0110 | 6 | 6 |
| 7 | 0111 | 7 | 7 |
| 8 | 1000 | 10 | 8 |
| 9 | 1001 | 11 | 9 |
| 10 | 1010 | 12 | A |
| 11 | 1011 | 13 | B |
| 12 | 1100 | 14 | C |
| 13 | 1101 | 15 | D |
| 14 | 1110 | 16 | E |
| 15 | 1111 | 17 | F |

```
2 |41  1  •  0.6875
2 |20  0     x      2
2 |10  0     ---------
   2 |5  1   1.3750
   2 |2  0   x    2
       1     ---------
             0.750
             x  2
             ---------
             1.50
             x 2
             ---------
             1.0
```

```
8 |41  1  •  0.6875
    5        x      8
             ---------
             5.5000
             x 8
             -----
             4.0
```

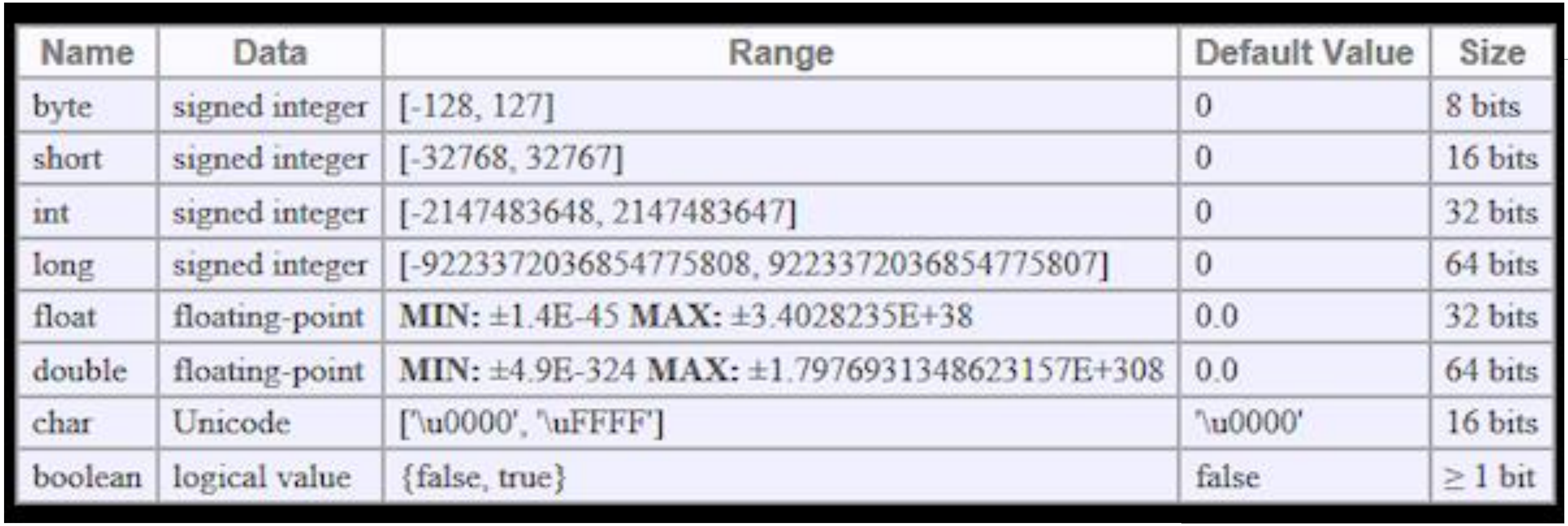
```
16 |41  9    0.6875
     2          x  16
             ---------
                4.1250
             +  6.8750
             ---------
               11.0000
```

# Integer Types

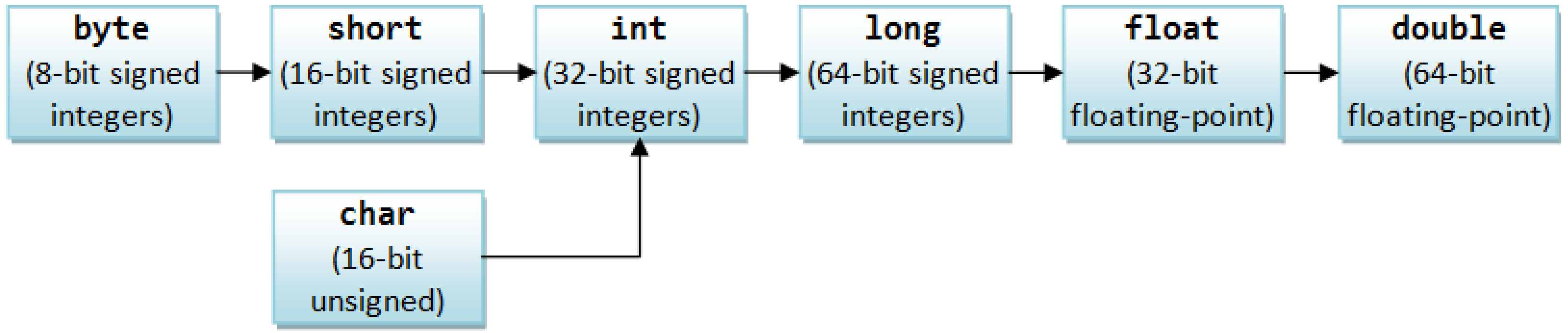
LECTURE 1

# Numeric Data Types and Operations

Java has six numeric types for integers and floating-point numbers with operators +, -, *, . and %



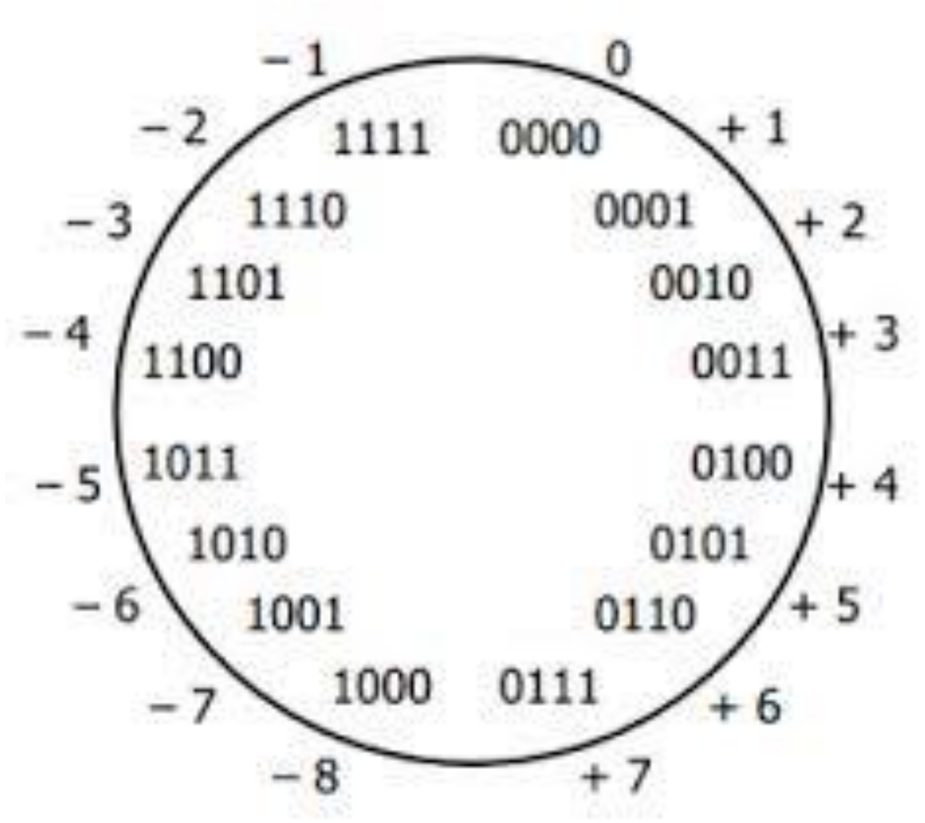
| Name | Data | Range | Default Value | Size |
|---|---|---|---|---|
| byte | signed integer | [-128, 127] | 0 | 8 bits |
| short | signed integer | [-32768, 32767] | 0 | 16 bits |
| int | signed integer | [-2147483648, 2147483647] | 0 | 32 bits |
| long | signed integer | [-9223372036854775808, 9223372036854775807] | 0 | 64 bits |
| float | floating-point | **MIN:** ±1.4E-45 **MAX:** ±3.4028235E+38 | 0.0 | 32 bits |
| double | floating-point | **MIN:** ±4.9E-324 **MAX:** ±1.7976931348623157E+308 | 0.0 | 64 bits |
| char | Unicode | ['\u0000', '\uFFFF'] | '\u0000' | 16 bits |
| boolean | logical value | {false, true} | false | ≥ 1 bit |


byte (8-bit signed integers)
short (16-bit signed integers)
int (32-bit signed integers)
long (64-bit signed integers)
float (32-bit floating-point)
double (64-bit floating-point)
char (16-bit unsigned)

Orders of Implicit Type-Casting for Primitives

Integer.MIN_VALUE
Integer.MAX_VALUE

# Two’s Complement



Negative number is represented as two’s complement.

For byte number’s (8 bits):

-$X = (2^8-1) - X + 1$;

$X + (-X) = X + (2^8-X) = 2^8 = 0$ ;

eg.

A = 0100 -> A’s One’s Complement = 1011 ->

A’s Two’s Complement -> 1100

The number $2^8$ is a overflow for the byte format, because unsigned byte number range

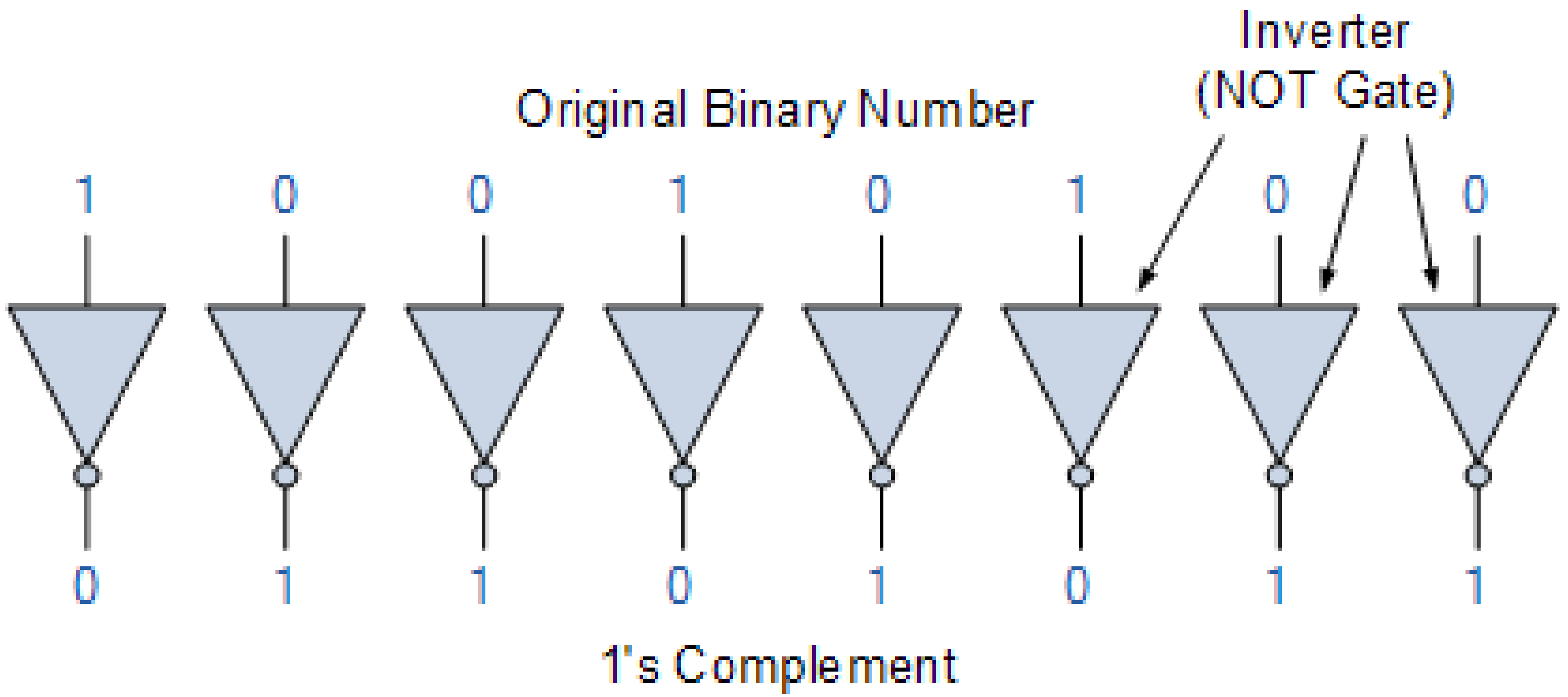
from 0 to $2^8 - 1$ = 11111111.

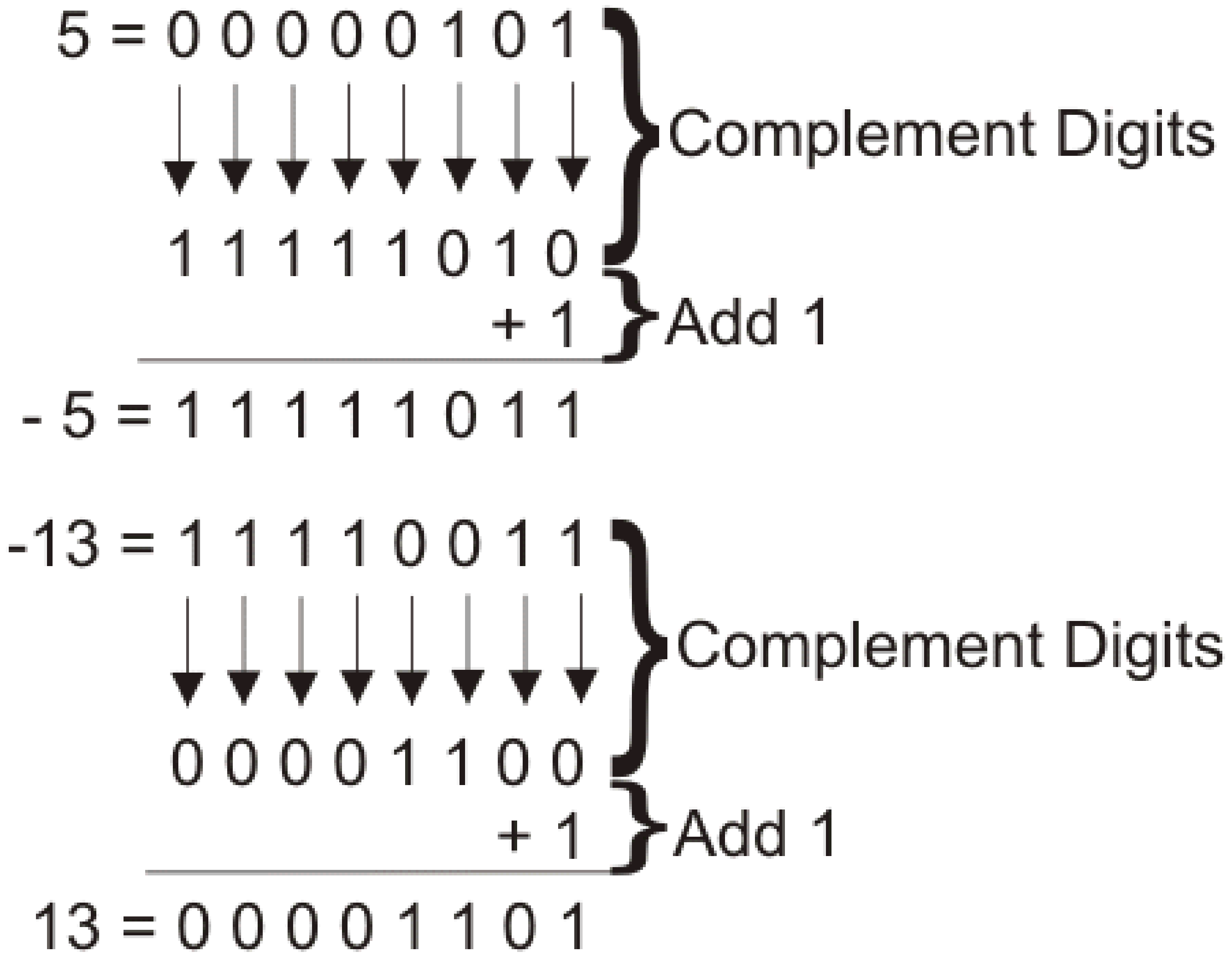
Therefore, this method can work for computer.

## One's Complement

Invert all bits. Each 1 becomes a 0, and each 0 becomes a 1.

| Original Value | | One's Complement |
|---|---|---|
| 0 | → | 1 |
| 1 | | 0 |
| 1010 | → | 0101 |
| 1111 | | 0000 |
| 11110000 | → | 00001111 |
| 10100011 | | 01011100 |
| 11110000 10100101 | → | 00001111 01011010 |


Original Binary Number
Inverter (NOT Gate)
1 0 0 1 0 1 0 0
0 1 1 0 1 0 1 1
1's Complement


5 = 0 0 0 0 0 1 0 1
Complement Digits
1 1 1 1 1 0 1 0
+ 1
Add 1
- 5 = 1 1 1 1 1 0 1 1
-13 = 1 1 1 1 0 0 1 1
Complement Digits
0 0 0 0 1 1 0 0
+ 1
Add 1
13 = 0 0 0 0 1 1 0 1

## Finding 2’s Complement

|  | - 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |  |
|---|---|---|---|---|---|---|---|---|---|
| $X$ | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | **Number : 79 decimal** |
| $(2^8-1)-X$ | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | **Flip the bits** |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | **Add 1** |
| $NegX = (2^8-1)-X+1$ | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | **Number: -79 in 2’s Complement format** |

# Integer Literals

LECTURE 1

# Java Integer literals

- Theoretically Literal means – Any number, text or other information that represents a **value**.
- Different values that can be assigned to an integer variable (Integer data type Literal)

# Literal Types

| Literal Type | Assignment Statement | Explanation |
|---|---|---|
| **Decimal** | int num = 20; | Decimal 20 is assigned to the variable num |
| **Octal** | int num = 020; | "**020**" is octal number , so first octal number is converted into integer and then it is assigned to variable "**num**" |
| **Hexadecimal** | int num = 0x20; | "**0x20**" is hexadecimal number ,<br>It is first converted into Decimal then assigned to variable "**num**" |
| **Binary** | int num = 0b1010; | "**0b1010**" is binary number , assigned to the variable "**num**" after converting it into decimal number |
| **Long** | long num = 563L; | "**563L**" is long number , assigned to the variable "**num**" |

# Java integer literal and Underscore

1. In JDK 7, we can embed one or more underscores in an integer literal.
2. It makes easier to read large integer literals.
3. When the literal is compiled, the underscores are discarded. **int num = 19_90;**
4. Java compiler will discard '_' from the above number and will assign 1990 to variable "num". Thus it is as good as writing – **int num = 1990;**

| **Literal** | **Using Underscore** | **Actual Value** |
|---|---|---|
| Integer Literal | 45_89 | 4589 |
| Octal Literal | 045_23 | Equivalent Octal : 04523 |
| Hexadecimal Literal | 0x56_23 | Equivalent Hex : 0x5623 |
| Binary Literal | 0b1000_1001 | Equivalent Binary : 10001001 |

# Java integer literal and Underscore

**Note : Using Underscore in Integer**

1. Don’t Use Underscore as first and last character.
2. It is used to read long number easily

**Illegal ways of using underscore**

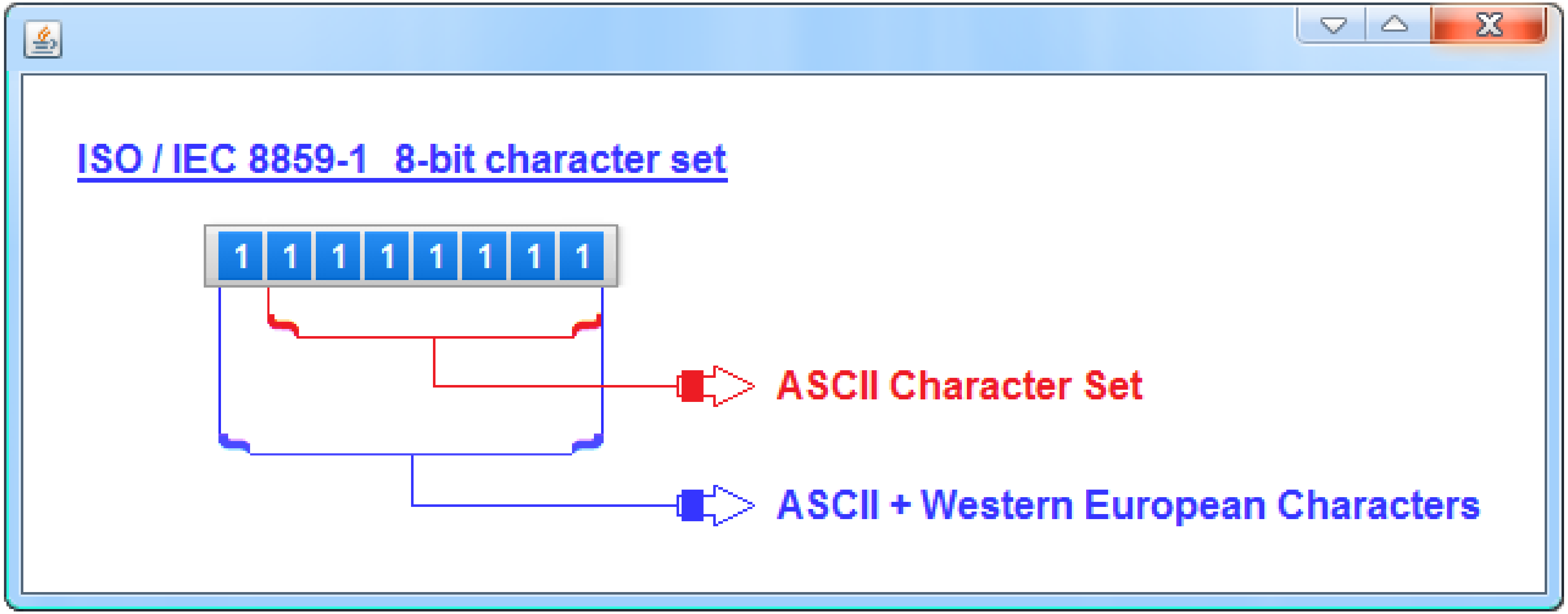
- Below are some places where we cannot put the underscore while using the Java integer literal –
  1. We cannot put underscore at the beginning or end of a number
  2. Underscore should not be placed adjacent to a decimal point in a floating point literal
  3. Use of underscore prior to an F or L suffix is illegal
  4. Underscore should not be used in positions where a string of digits is expected

## Characters

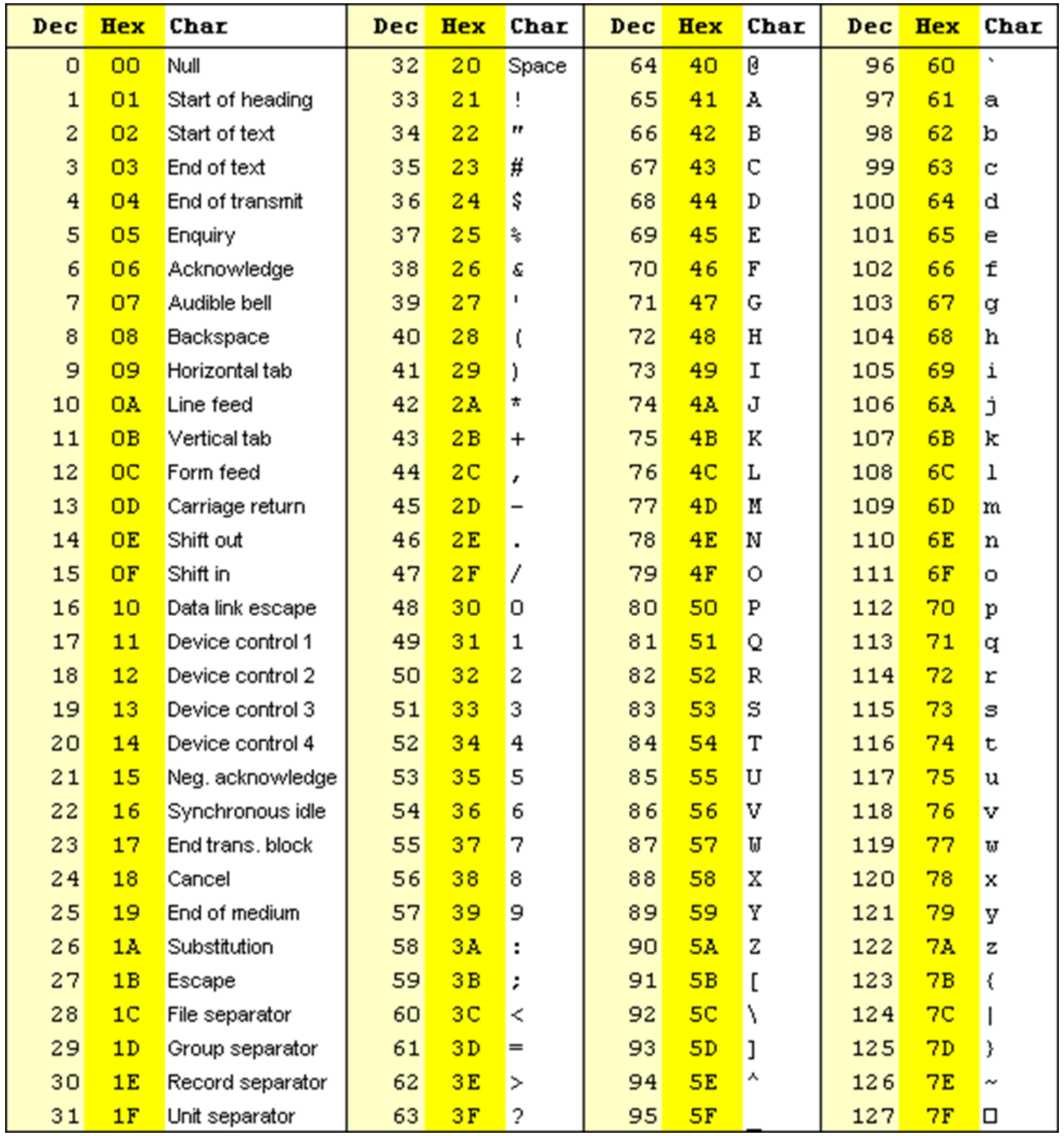
LECTURE 1

## Characters

- The data type **char** represents a single character in Java.
- Character values are written as a symbol: ‘a’, ‘)’, ‘A’, etc.
- A **char** value in Java is really represented as an integer.
  - Each character has an associated integer value. The integer value is the ASCII value or Unicode value (UTF-16)


ISO / IEC 8859-1   8-bit character set
1 1 1 1 1 1 1 1
ASCII Character Set
ASCII + Western European Characters

# ASCII



| Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 00 | Null | 32 | 20 | Space | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 01 | Start of heading | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 02 | Start of text | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 03 | End of text | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 04 | End of transmit | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 05 | Enquiry | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 06 | Acknowledge | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 07 | Audible bell | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 08 | Backspace | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 09 | Horizontal tab | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | 0A | Line feed | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | 0B | Vertical tab | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | 0C | Form feed | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | 0D | Carriage return | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | 0E | Shift out | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | 0F | Shift in | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | Data link escape | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | Device control 1 | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | Device control 2 | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | Device control 3 | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | Device control 4 | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | Neg. acknowledge | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | Synchronous idle | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | End trans. block | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | Cancel | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | End of medium | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | Substitution | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | Escape | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | File separator | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 29 | 1D | Group separator | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | Record separator | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | Unit separator | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | □ |

## Java-Supplementary Characters and UTF-16 Encoding

- The **char** data type are based on the original **Unicode** specification, which defined characters as fixed-width 16-bit entities. The Unicode standard has since been changed to allow for characters whose representation requires more than 16 bits.
- The range of legal code points is now **U+0000** to **U+10FFFF**, known as Unicode scalar value.

# Java-Supplementary Characters and UTF-16 Encoding

- The set of characters from **U+0000** to **U+FFFF** is sometimes referred to as the Basic Multilingual Plane (**BMP**). Characters whose code points are greater than **U+FFFF** are called supplementary characters.
- The Java 2 platform uses the UTF-16 representation in char arrays and in the **String** and **StringBuffer** classes. In this representation, supplementary characters are represented as a pair of char values, the first from the high-surrogates range, (\uD800-\uDBFF), the second from the low-surrogates range (\uDC00-\uDFFF).

# Bit-Level Integer Operations

LECTURE 1

# Binary Addition

## Binary Arithmetic Rules

$$\begin{array}{cccc} 0 & 1 & 0 & 1\\ +\,0 & +\,0 & +\,1 & +_{1}1\\ \hline 0 & 1 & 1 & 0\\ \hline \end{array}$$

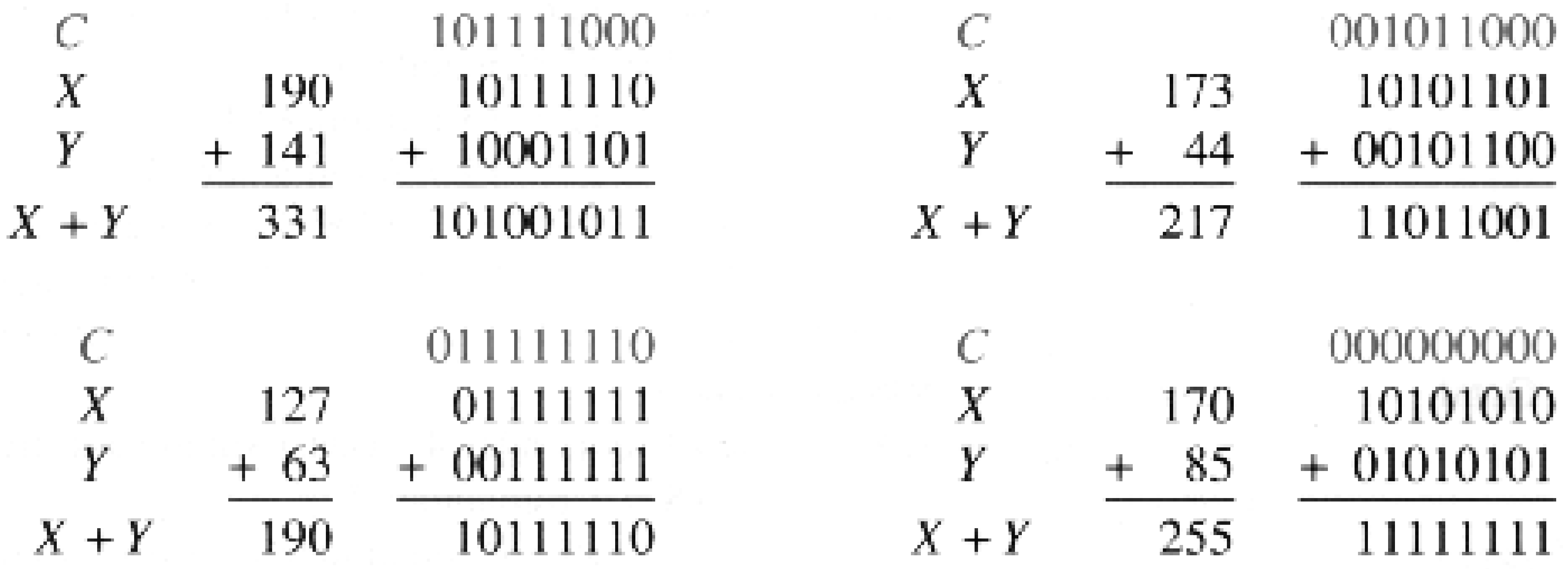
$$\begin{array}{cccc} 0 & 1 & 0 & 1\\ +\ 0_{1} & +_{1}0_{1} & +_{1}1_{1} & +_{1}1_{1}\\ \hline 1 & 0 & 0 & 1\\ \hline \end{array}$$

Key:

Carry Out

Carry in

## Binary Addition



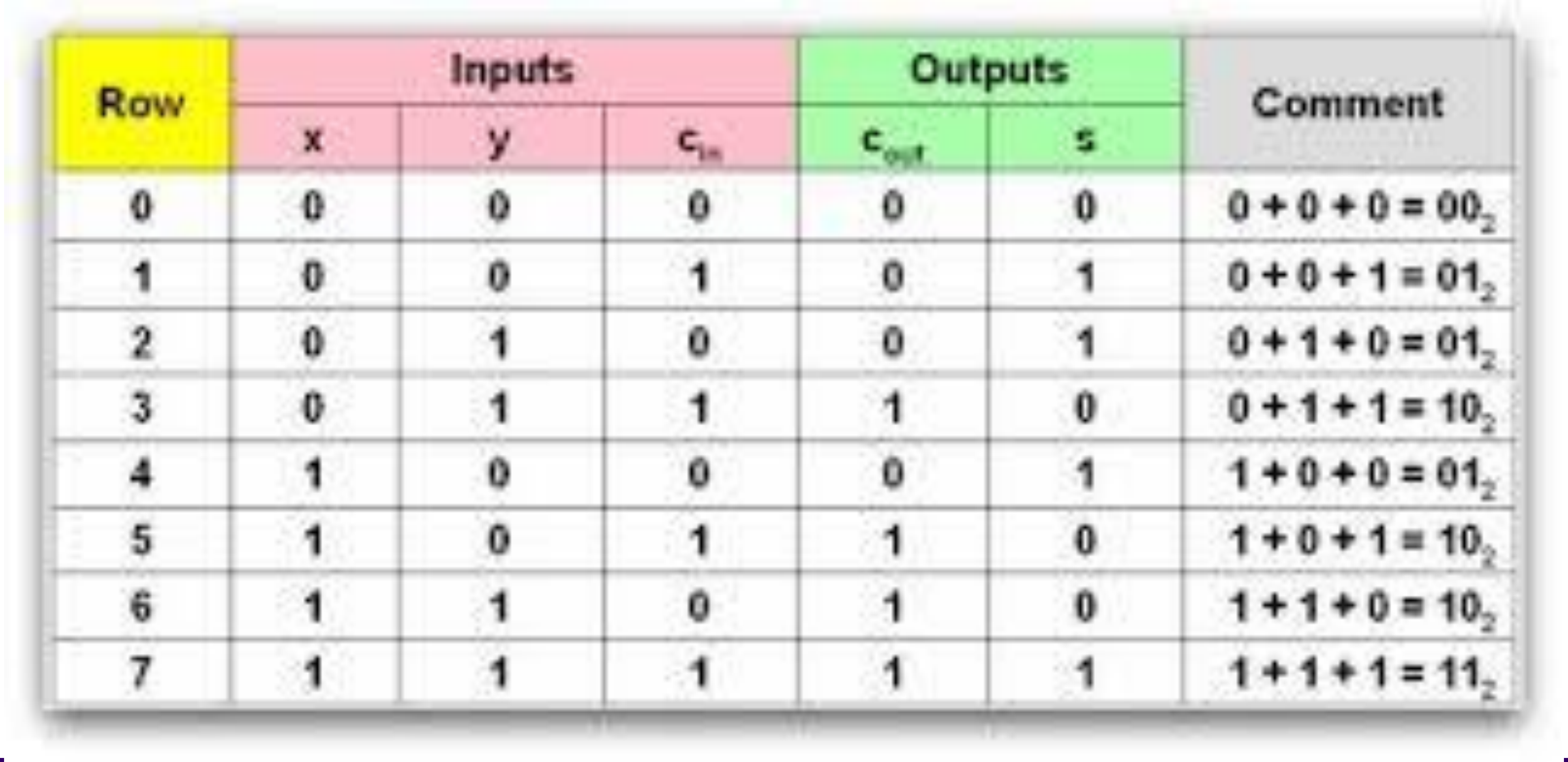
| | | |
|---|---|---|
| *C* | | 101111000 |
| *X* | 190 | 10111110 |
| *Y* | + 141 | + 10001101 |
| *X* + *Y* | 331 | 101001011 |

| | | |
|---|---|---|
| *C* | | 001011000 |
| *X* | 173 | 10101101 |
| *Y* | + 44 | + 00101100 |
| *X* + *Y* | 217 | 11011001 |

| | | |
|---|---|---|
| *C* | | 011111110 |
| *X* | 127 | 01111111 |
| *Y* | + 63 | + 00111111 |
| *X* + *Y* | 190 | 10111110 |

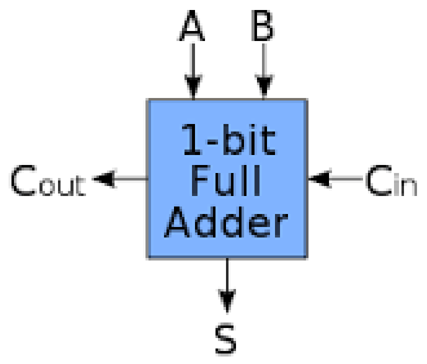
| | | |
|---|---|---|
| *C* | | 000000000 |
| *X* | 170 | 10101010 |
| *Y* | + 85 | + 01010101 |
| *X* + *Y* | 255 | 11111111 |

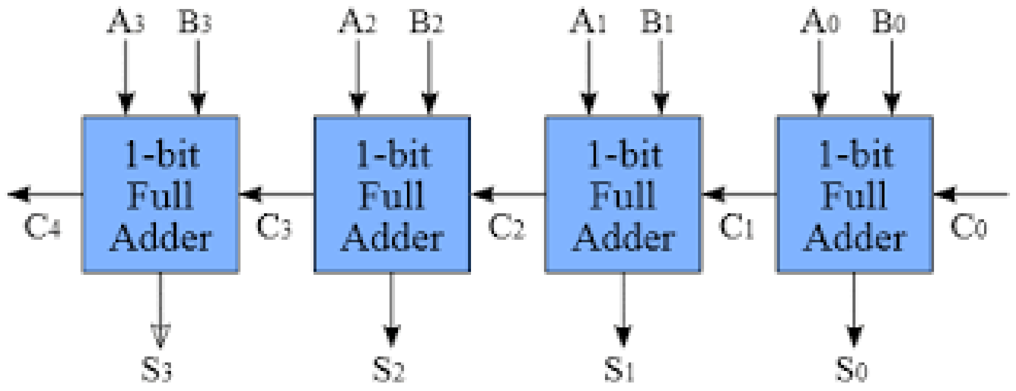
## Hardware Design for a One-bit adder



| Row | Inputs | | | Outputs | | Comment |
|---|---|---|---|---|---|---|
| | $x$ | $y$ | $c_{in}$ | $c_{out}$ | $s$ | |
| 0 | 0 | 0 | 0 | 0 | 0 | $0 + 0 + 0 = 00_2$ |
| 1 | 0 | 0 | 1 | 0 | 1 | $0 + 0 + 1 = 01_2$ |
| 2 | 0 | 1 | 0 | 0 | 1 | $0 + 1 + 0 = 01_2$ |
| 3 | 0 | 1 | 1 | 1 | 0 | $0 + 1 + 1 = 10_2$ |
| 4 | 1 | 0 | 0 | 0 | 1 | $1 + 0 + 0 = 01_2$ |
| 5 | 1 | 0 | 1 | 1 | 0 | $1 + 0 + 1 = 10_2$ |
| 6 | 1 | 1 | 0 | 1 | 0 | $1 + 1 + 0 = 10_2$ |
| 7 | 1 | 1 | 1 | 1 | 1 | $1 + 1 + 1 = 11_2$ |

## Hardware Design for a One-bit adder





# Subtraction (A-B) = (A + -B)

## using Two's compliment addition for subtraction

(using Java int data type example)

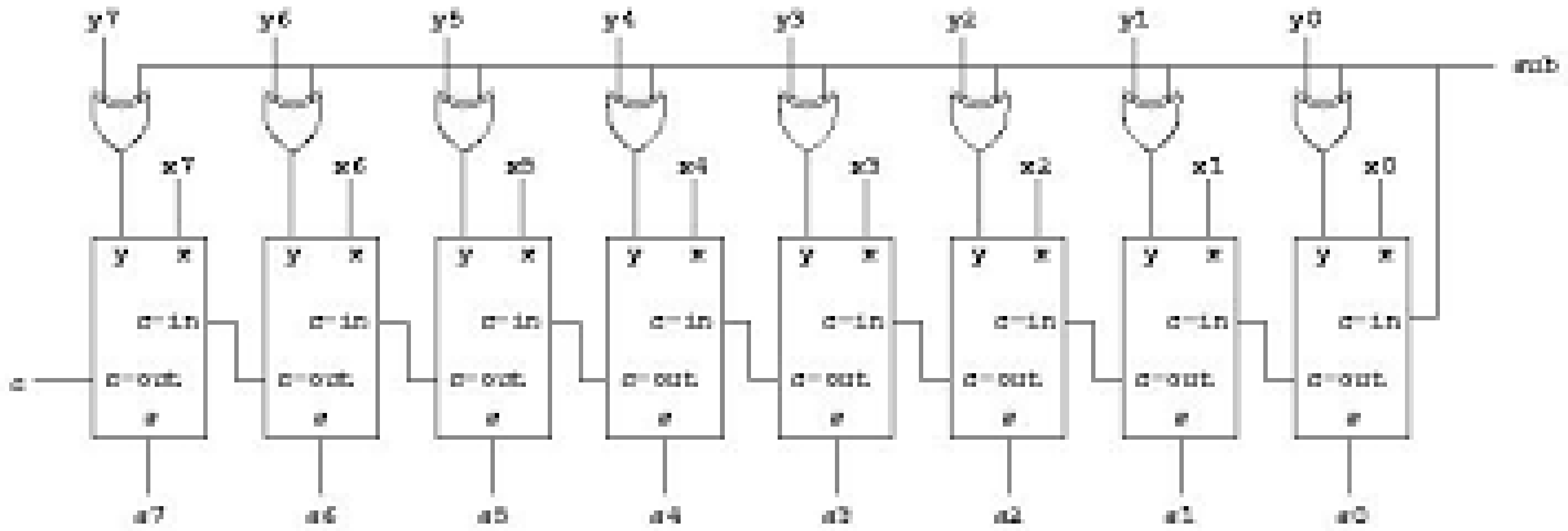
- $12_{ten} - 5_{ten}$

$$
\begin{array}{rl}
 & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1100 \quad (12_{ten}) \\
- & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0101 \quad (\ 5_{ten}) \\
\hline
= & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111 \quad (\ 7_{ten})
\end{array}
$$

- $12_{ten} - 5_{ten} = 12_{ten} + (-5_{ten})$

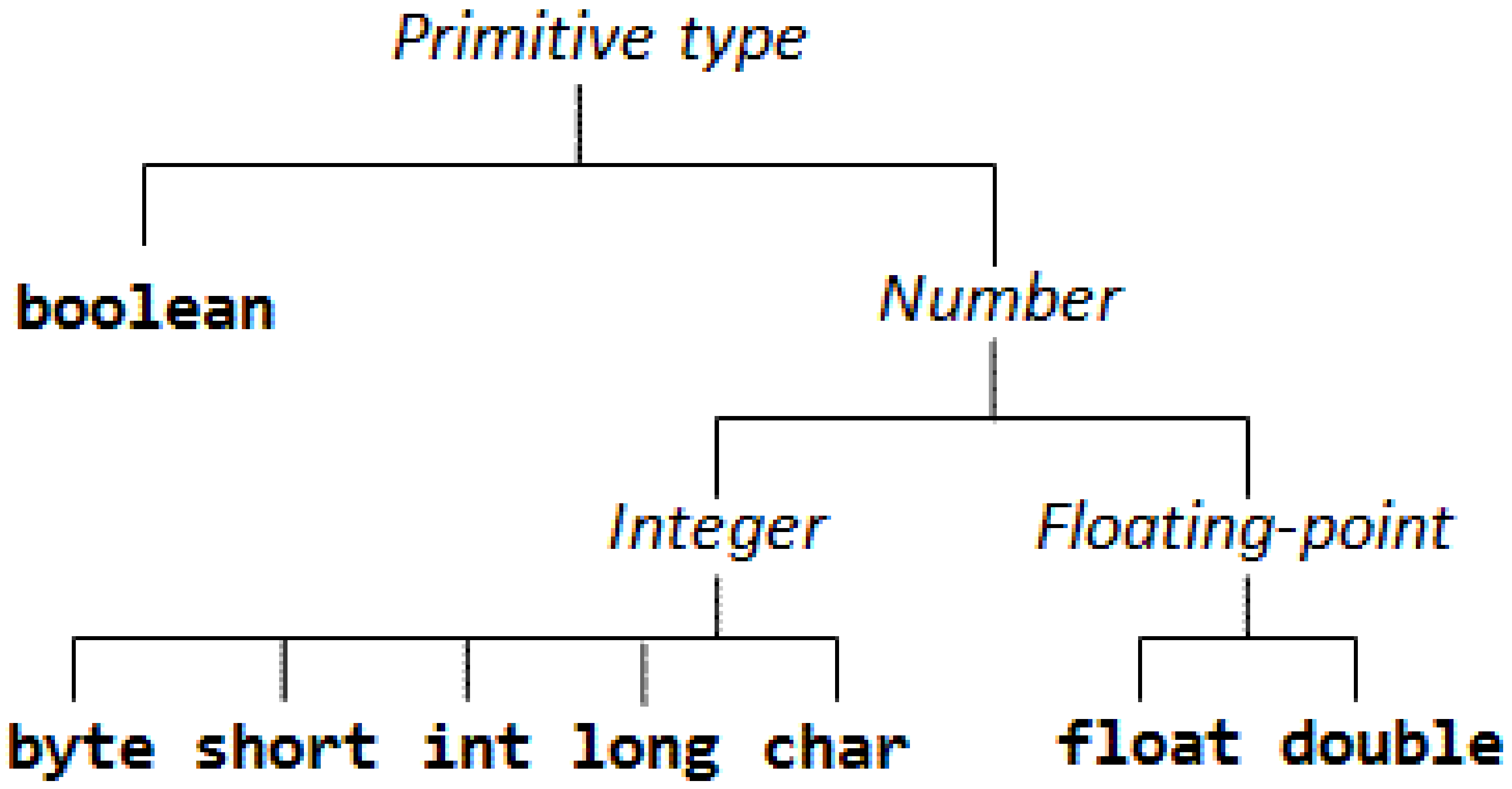
$$
\begin{array}{rl}
 & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1100 \quad (12_{ten}) \\
+ & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1011 \quad (-5_{ten}) \\
\hline
= & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111 \quad (\ 7_{ten})
\end{array}
$$

# Full 8-bit adder/subtractor design



# Floating Point Data Types

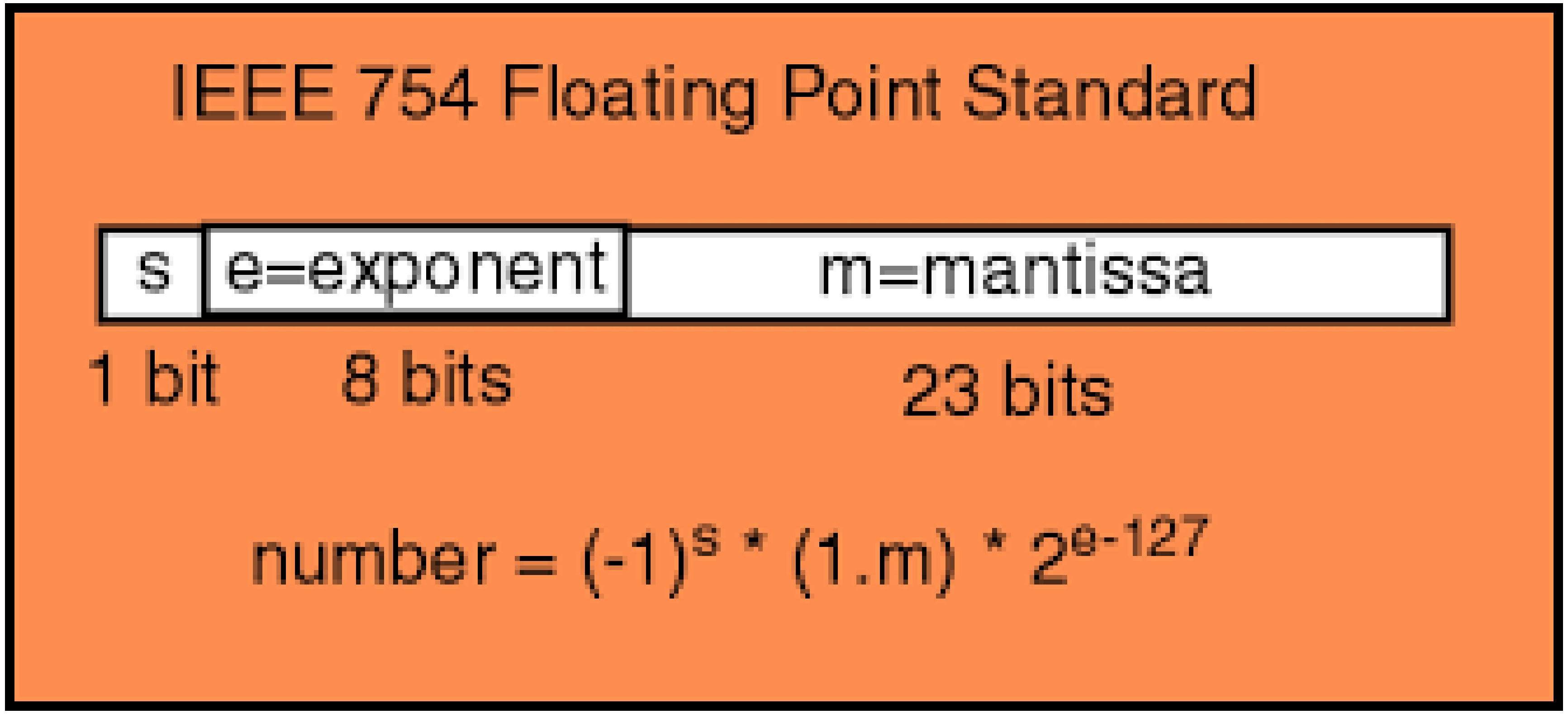
LECTURE 1


Primitive type
boolean
Number
Integer
Floating-point
byte
short
int
long
char
float
double

## Java’s special number rules
(different from other languages)

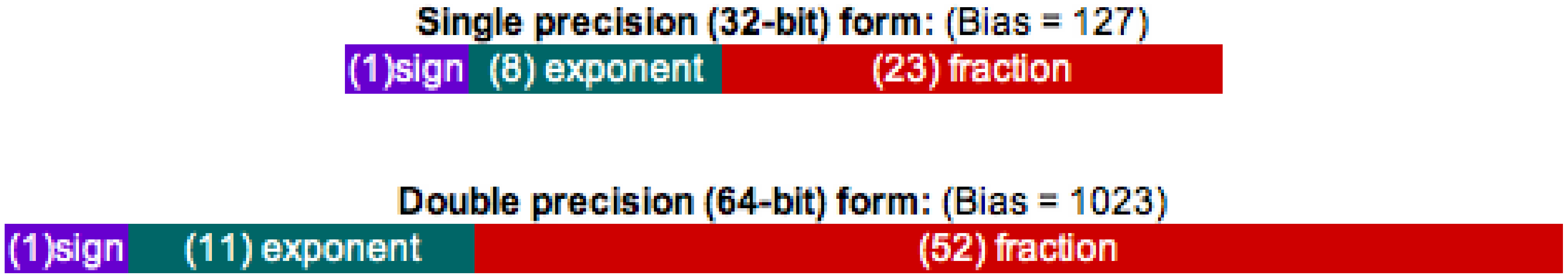
- **Java doesn’t have unsigned number primitives.**
  - unsigned number is seldom used.
  - If you need to use unsigned number, use **char** data type instead. Because char does not follow the number operation rules while **char** can still operate the **bit-wise** operations.
- **Java’s char is 16 bit.** (supporting Unicode: UTF-16)
- **IEEE 754 binary floating point representation. (Java’s Float Standard)**

# Java’s special number rules



$$\text{number} = (-1)^{s} * (1.m) * 2^{e-127}$$

## Java’s special number rules

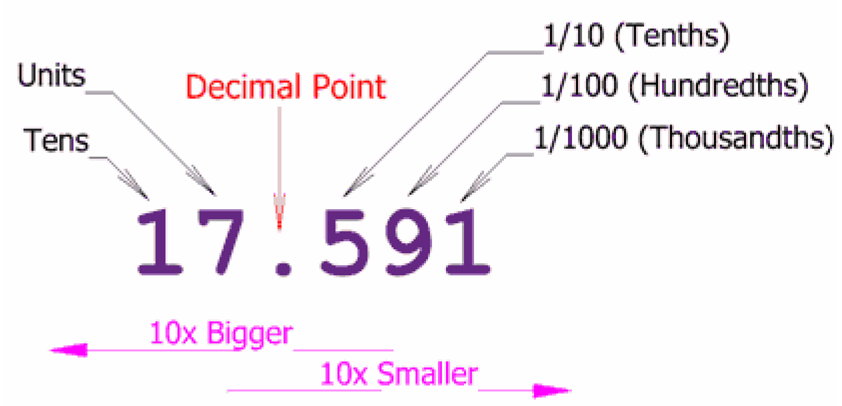

Single precision (32-bit) form: (Bias = 127)
(1)sign
(8) exponent
(23) fraction
Double precision (64-bit) form: (Bias = 1023)
(1)sign
(11) exponent
(52) fraction

## Floating Point Literals

LECTURE 1

## Floating Point Literal : Primitive Data Type in Java Programming

1. Decimal values with a fractional component is called floating point.
2. They can be expressed in either standard or scientific notation.



### Standard Notation

1. Standard notation consists of a whole number component followed by a decimal point followed by a fractional component.
2. For example : 78.0, 3.14159 represent valid standard-notation floating-point numbers.

## Scientific Notation

Scientific notation uses a standard-notation, floating-point number plus a suffix that specifies a power of 10 by which the number is to be multiplied.

The exponent is indicated by an E or e followed by a decimal number, **which can be positive or negative**.

Valid Examples are :

- 6.02E21
- 314159E–05
- 2e+100.

**Floating-point literals in Java default to double precision. 0.333F 0.333D (Same double Format)**

| Literal | Representation | Size | Default |
|---|---|---|---|
| Floating Point Number | F or f | 32 bits | – |
| Double Number | D or d | 64 bits | It is default type |

## Live Example : Assigning Values to Floating Point Literal

```
public static void main(String args[])
    { double d1 = 45.6;
      float  f1 = 32.5;
    }
```

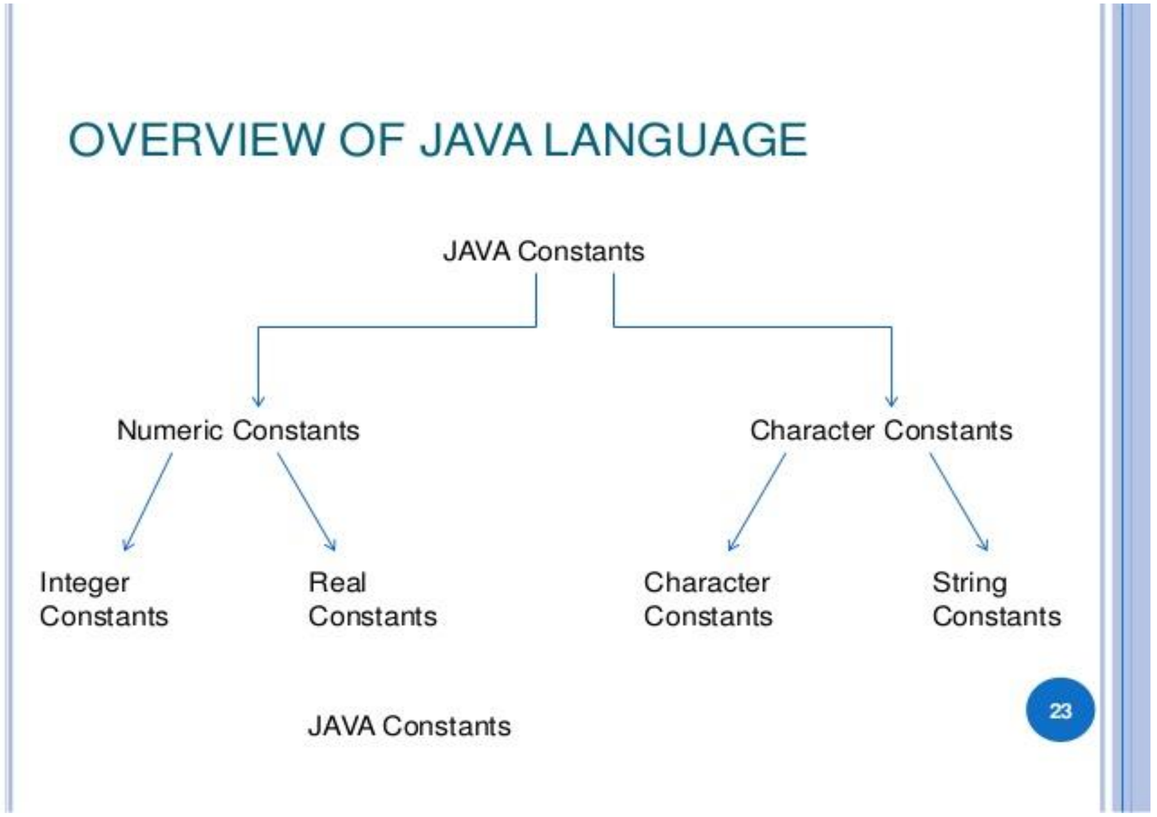
**Short Notes :**

1. Jdk 7 also provides us facility for writing hexadecimal literal but they are rarely used.
2. We can use Underscore inside Literals.

```
double num = 1_567_2_82.0;
```

## Named Constants

```
final datatype CONSTANTNAME = VALUE;


final double PI = 3.14159;
final int SIZE = 3;
```



OVERVIEW OF JAVA LANGUAGE

JAVA Constants

23

# Constants are Immutable Data
Literals are Data Representations

Constants are used to hide coding complexity. (One way of doing abstraction)

Constants for Scientific Calculation:

Math.PI (3.141592….), Math.E

Replacing Long URL or Long Text Message:

```
final String googleURL = “http://www.google.com”
```

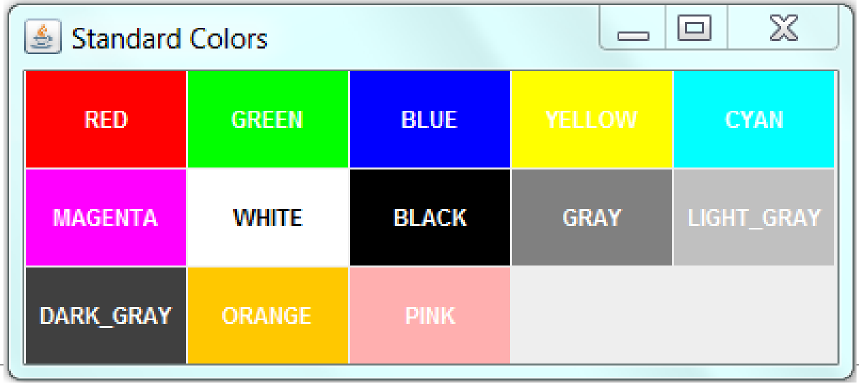
Constants for Program Control: (declare at the head of program and update only once.)

```
final int STEPS = 100;
final int TRIALS = 100000000;
final int MODE = 0;   // 0: debug, 1: development, 2: analysis, 3: production
```

# Color Data Type

LECTURE 1

# Constants: Color



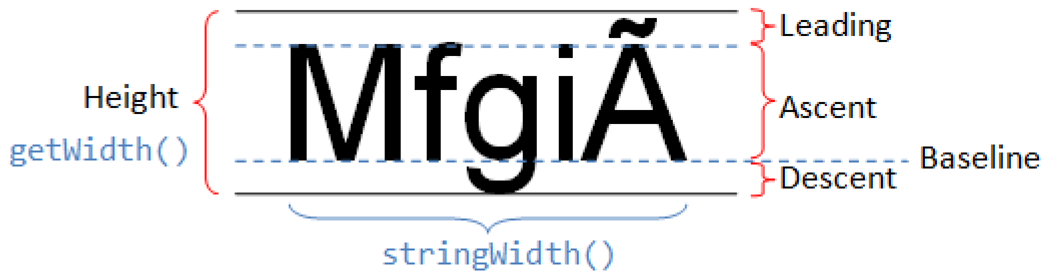
The class **java.awt.Color** provides 13 standard colors as named-constants. They are: **Color.RED, GREEN, BLUE, MAGENTA, CYAN, YELLOW, BLACK, WHITE, GRAY, DARK_GRAY, LIGHT_GRAY, ORANGE,** and **PINK**.

(In JDK 1.1, these constant names are in lowercase, e.g., red. This violates the Java naming convention for constants. In JDK 1.2, the uppercase names are added. The lowercase names were not removed for backward compatibility.)

You can use the toString() to print the RGB values of these color (e.g., System.out.println(**Color.RED**)):

# Font java.awt.Font;

```
Font myFont1 = new Font(Font.MONOSPACED, Font.PLAIN, 12);
Font myFont2 = new Font(Font.SERIF, Font.BOLD | Font.ITALIC, 16);  // bold and italics
JButton btn = new JButton("RESET");
btn.setFont(myFont1);
JLabel lbl = new JLabel("Hello");
lbl.setFont(myFont2);
......
g.drawString("In default Font", 10, 20);     // in default font
Font myFont3 = new Font(Font.SANS_SERIF, Font.ITALIC, 12);
g.setFont(myFont3);
g.drawString("Using the font set", 10, 50);  // in myFont3
```



# Basic Java Program Unit

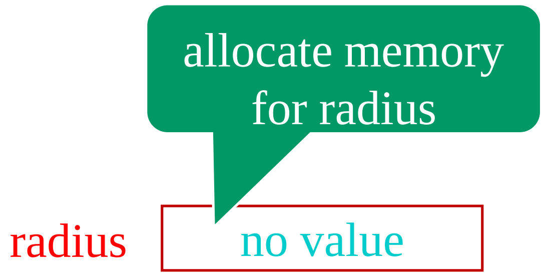
LECTURE 1

## ComputeArea.java

```
public class ComputeArea {
    public static void main(String[] args) {
          double radius;      // Declare radius
          double area;          // Declare area
          // Assign a radius
        radius = 20; // New value is radius
          // Compute area
          area = radius * radius * 3.14159;
          // Display results
          System.out.println("The area for the circle of radius " + radius + " is " + area);
          }
}
```

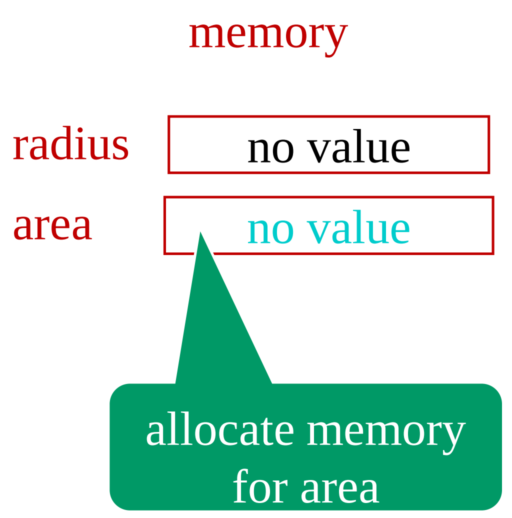
# Trace a Program Execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```



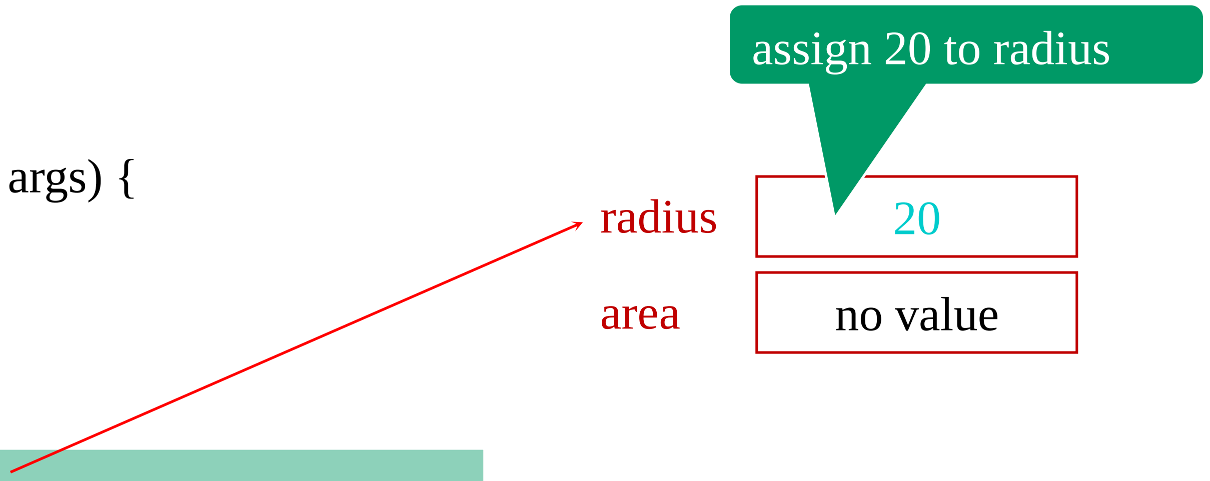
# Trace a Program Execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```



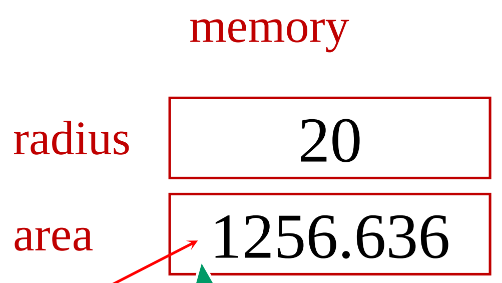
# Trace a Program Execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```



## Trace a Program Execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```



compute area and assign it to variable area

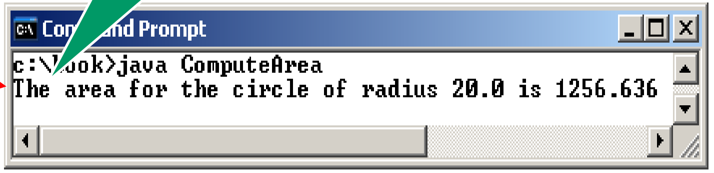
# Trace a Program Execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```

memory

| | |
|---|---|
| radius | 20 |
| area | 1256.636 |

print a message to the console



```
c:\book>java ComputeArea
The area for the circle of radius 20.0 is 1256.636
```

# Input>process>output

## Every system has:

**Input**
(ingredients that are put into the system)

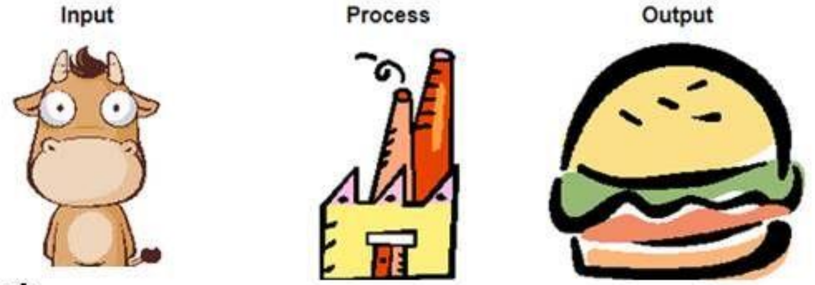
↓

**Process**
(what happens to make the system work)

↓

**Output**
(what the system creates)

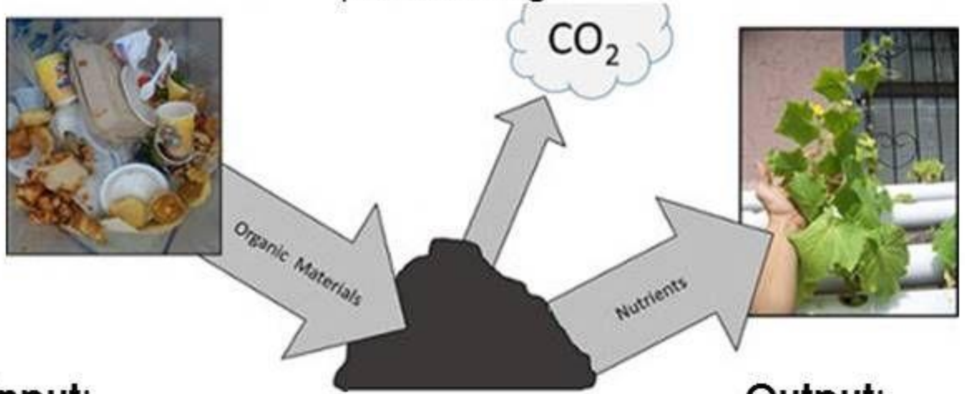
## For example…



**Input**: cow enters processing factory

**Process**: slaughter, butchery, meat processing.

**Output**: Burger to eat!



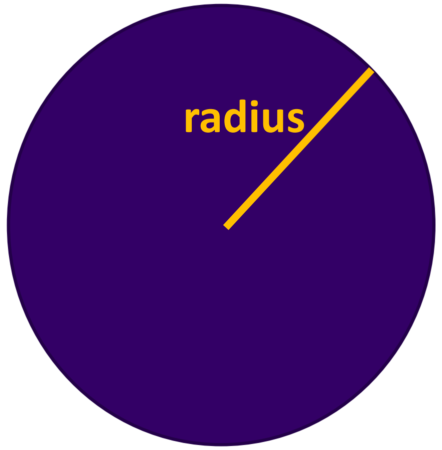
**Input**: leftover food

**Process**: composting and spreading compost on fields

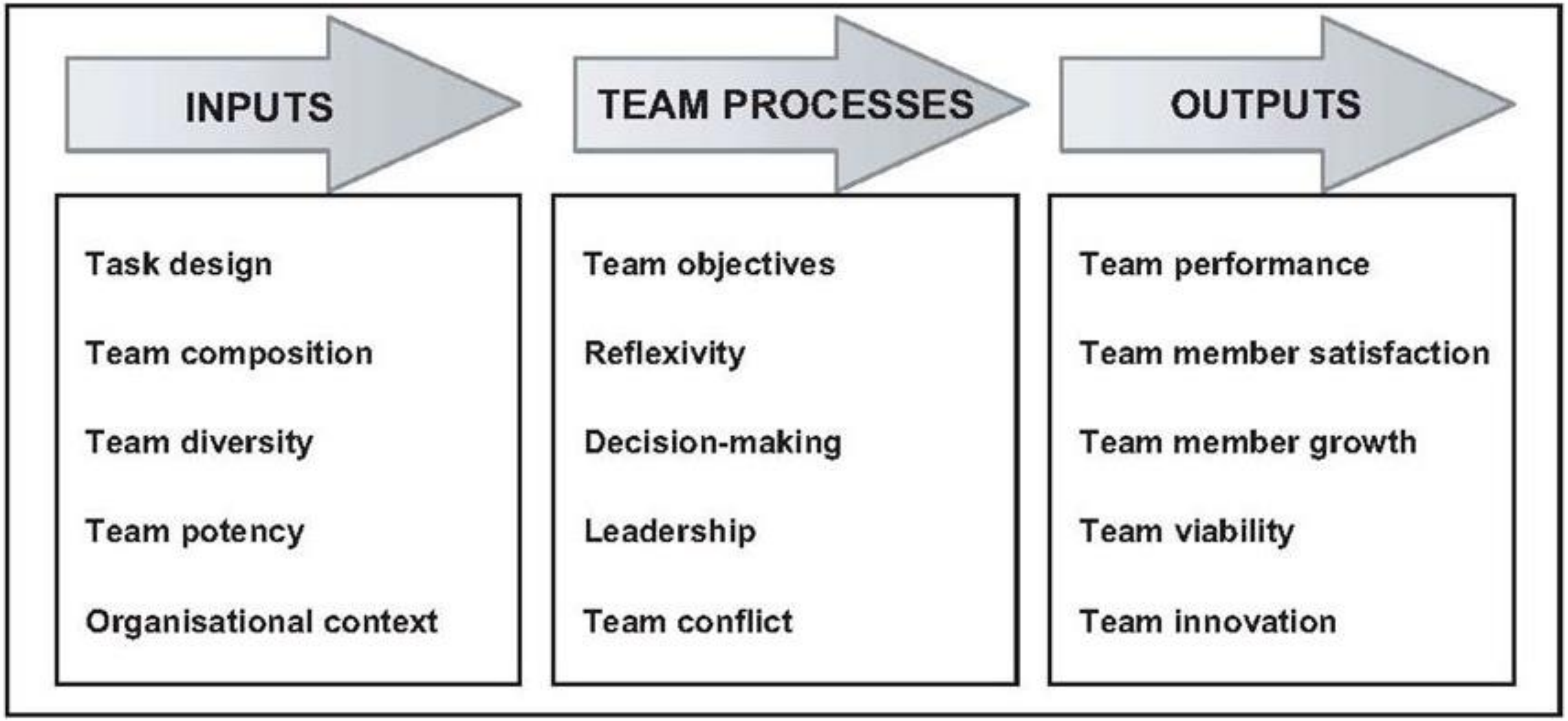
**Output**: New food grown!

## How to input from Console?

```
public class Example {
    public static void main(String[] args){
        // Variable Declaration
        double radius = 5.0;
        // Input part
        Scanner input = new Scanner(System.in);
        radius = input.nextDouble();
        // Processing part
        double area = Math.PI * radius * radius;
        //  Output Part
        System.out.println(area);
    }
}
```



## Input-process-output model of team effectiveness



## Demonstration Program

COMPUTEAREA.JAVA

# Basic Class and Objects

LECTURE 1

## In-Class Demonstration Program

- Constant Pi
- Constructor
- Getter method
- Setter method
- getArea()
- getPerimeter()

CIRCLE.JAVA AND TESTCIRCLE.JAVA

# Scanner Class

LECTURE 1

## Reading Input from the Console

### Demo Program: ComputeAreaWithConsoleInputWithConstant.java

1. Create a Scanner object

```
Scanner input = new Scanner(System.in);
```

2. Use the methods next(), nextByte(), nextShort(), nextInt(), nextLong(), nextFloat(), nextDouble(), or nextBoolean() to obtain to a string, byte, short, int, long, float, double, or boolean value. For example,

```
System.out.print("Enter a double value: ");
Scanner input = new Scanner(System.in);
double d = input.nextDouble();
```

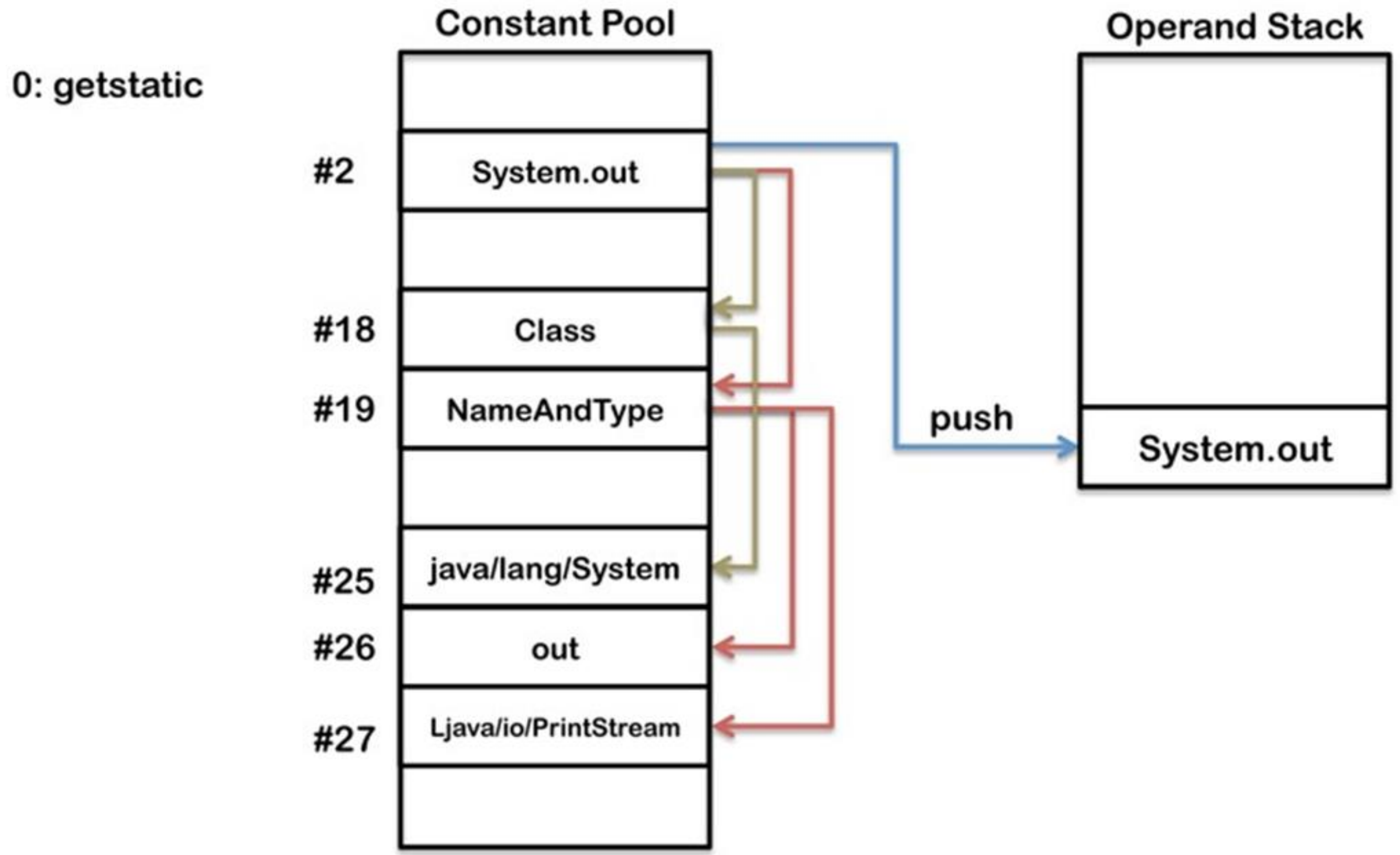
## import

- The **Scanner** class is in the **java.util** package. It is imported on a program for input purpose. There are two types of import statements: ***specific import*** and ***wildcard import***.
- Specific import specifies a single class in the import statement. For example, the following statement imports Scanner from the package **java.util**.

  **import java.util.Scanner;**

- The wildcard import imports all the classes in a package by using the asterisk as the wildcard. (Take much larger space. Push into the Operand Stack or Constant Pool) For example, the following statement imports all the classes from the package java.util.

  **import iava.util.*;**


0: getstatic
Constant Pool
Operand Stack
#2
System.out
#18
Class
#19
NameAndType
#25
java/lang/System
#26
out
#27
Ljava/io/PrintStream
push
System.out

## System Input and Output

- **System** is a class in the java.lang package. out is a static member of the System class, and is an instance of **java.io.PrintStream** . **println** is a method of **java.io.PrintStream** . This method is overloaded to print message to output destination, which is typically a console or file.
- **System.in:** System Input Device
- **System.out:** System Output Device (character stream type)
- Scanner is a class from scan code to ASCII code and then, ASCII String is converted to data of certain data type (**nextDouble(), nextInt()**).
- **println** is a output method

## In-Class Demonstration Program

- HashCode
- Reference data type

HASHCODE.JAVA