JAVA: Lesson 2

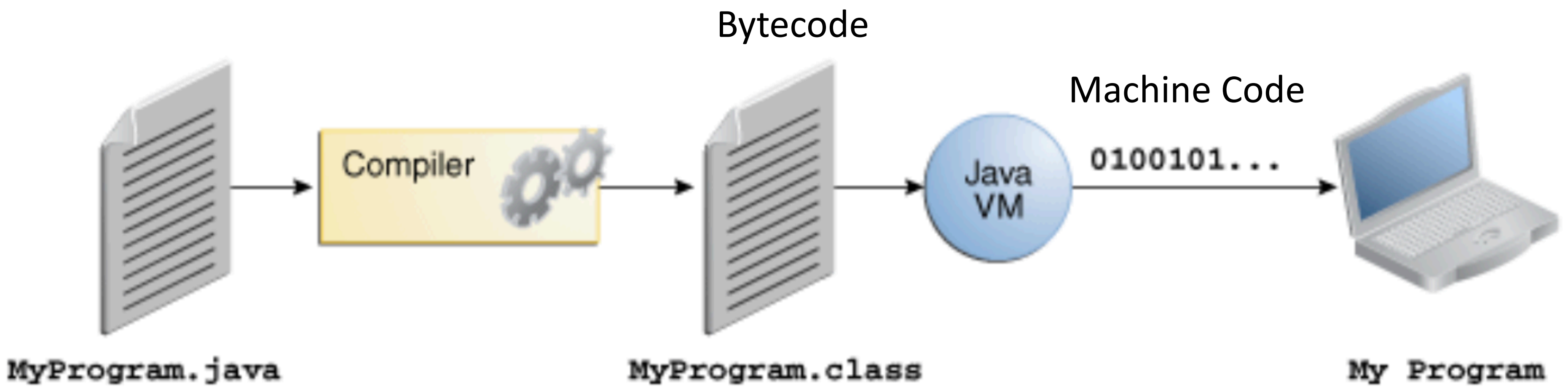
Avanindra Kumar Pandeya

Syllabus

- Week 1: Chapter 1 and 2
  - Creating, Compiling and Executing Java Program
  - Identifiers, Variables and Constants
  - Numeric Data Types and Operations
  - Character Data Types and Operations

Homework discussion

- The slides after the lecture may have additional information.
- Demonstration: Absolute vs relative path
- Ex 1. 1. Bytecode



An overview of the software development process.

Before we begin….

- Software Development requires two skills
  - Problem solving i.e. Algorithms
    - Solve problem correctly
    - Solve problem efficiently
  - Writing code i.e. Programming

Identifiers

- Identifiers are the names that identify the elements such as classes, methods, and variables in a program.
  - An identifier is a sequence of characters that consists of letters, digits, underscores (_), and dollar signs ($). Suggestion: Always begin with small case letter.
  - An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
  - An identifier cannot be a reserved word (keyword).
  - An identifier cannot be true, false, or null.
  - An identifier can be of any length.
- Caution: Since Java is case sensitive, ***area, Area,*** and ***AREA*** are all different identifiers.

Practice : Which ones are valid identifiers?

int

anInt

i

i1

1

thing1

1thing

ONE-HUNDRED

ONE_HUNDRED

something2do

Answer

int

anInt

i

i1

1

thing1

1thing

ONE-HUNDRED

ONE_HUNDRED

something2do

Suggestions for Identifiers

- Never use cryptic identifiers like ***a, b*** …. Use words or abbreviations which are easy to identify.
- When there are more than one words, capitalize the first letter of subsequent letters. e.g. ***numberOfGears***, ***areaOfTriangle*** etc.
- Capitalize the first letter of each word in a class name—for example, ***System, Bicycle***
- Capitalize every letter in a constant, and use underscores between words—for example, the constants ***PI*** and ***MAX_VALUE***.
- Do not name identifiers with the $ character. By convention, the $ character should be used only in mechanically generated source code

(Primitive) Data Types in Java

Data Type	Size	Description
byte	1 byte (8 bits) signed	Stores whole numbers from -128 (-2^7) to 127 $(2^7 - 1)$
short	2 bytes signed	Stores whole numbers from $-32{,}768$ (-2^{15}) to $32{,}767$ $(2^{15} - 1)$
int	4 bytes signed	Stores whole numbers from $-2{,}147{,}483{,}648$ (-2^{31}) to $2{,}147{,}483{,}647$ $(2^{31} - 1)$
long	8 bytes signed	Stores whole numbers from (-2^{63}) to $(-2^{63} - 1)$
float	4 bytes IEEE 754	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes IEEE 754	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values
String	-	Stores strings i.e. sequence of characters (*Not a primitive data type*)

Demonstration 1

- Data Types
- Numeric Literals
- Overflow
- Invalid Underscore

Placing Underscore

- You can place underscores only between digits; you cannot place underscores in the following places:
  - At the beginning or end of a number
  - Adjacent to a decimal point in a floating point literal
  - Prior to an F or L suffix
  - In positions where a string of digits is expected

Demonstration 2

- Compute area of a circle

printf

Conversion-Characters:

d :	decimal integer [byte, short, int, long]	
f :	floating-point number [float, double]	
c :	character	Capital C will uppercase the letter
s :	String	Capital S will uppercase all the letters in the string
h :	hashcode	A hashcode is like an address. This is useful for printing a reference
n :	newline	Platform specific newline character- use %n instead of \n for greater compatibility

Examples:

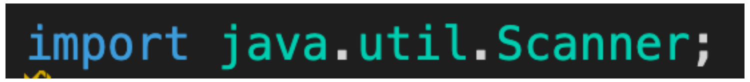
```
System.out.printf("Total is: $%,.2f%n", dblTotal);
System.out.printf("Total: %-10.2f: ", dblTotal);
System.out.printf("% 4d", intValue);
System.out.printf("%20.10s\n", stringVal);

String s = "Hello World";
System.out.printf("The String object %s is at hash code %h%n", s, s);
```

Reference

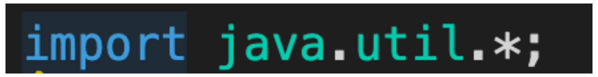
How to read a Console Input?

Step 1: Import Scanner class from java.utils.Scanner as follows:



```
import java.util.Scanner;
```

or

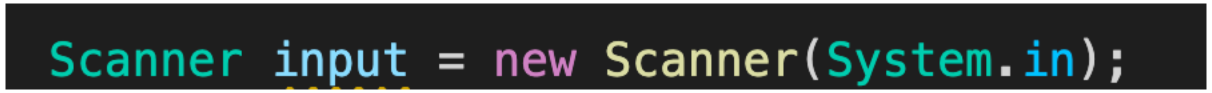


```
import java.util.*;
```

Specific Import

Wildcard Import

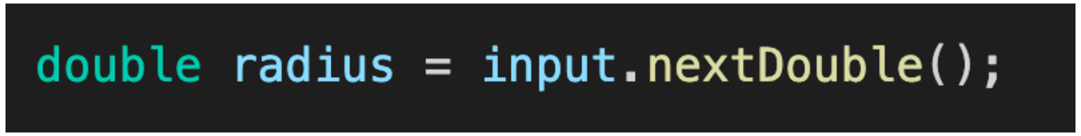
Step 2: Initialize and Create a new scanner type of Object



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

input: an identifier of your choice

Step3: To read console input call one of the methods of scanner object



```
double radius = input.nextDouble();
```

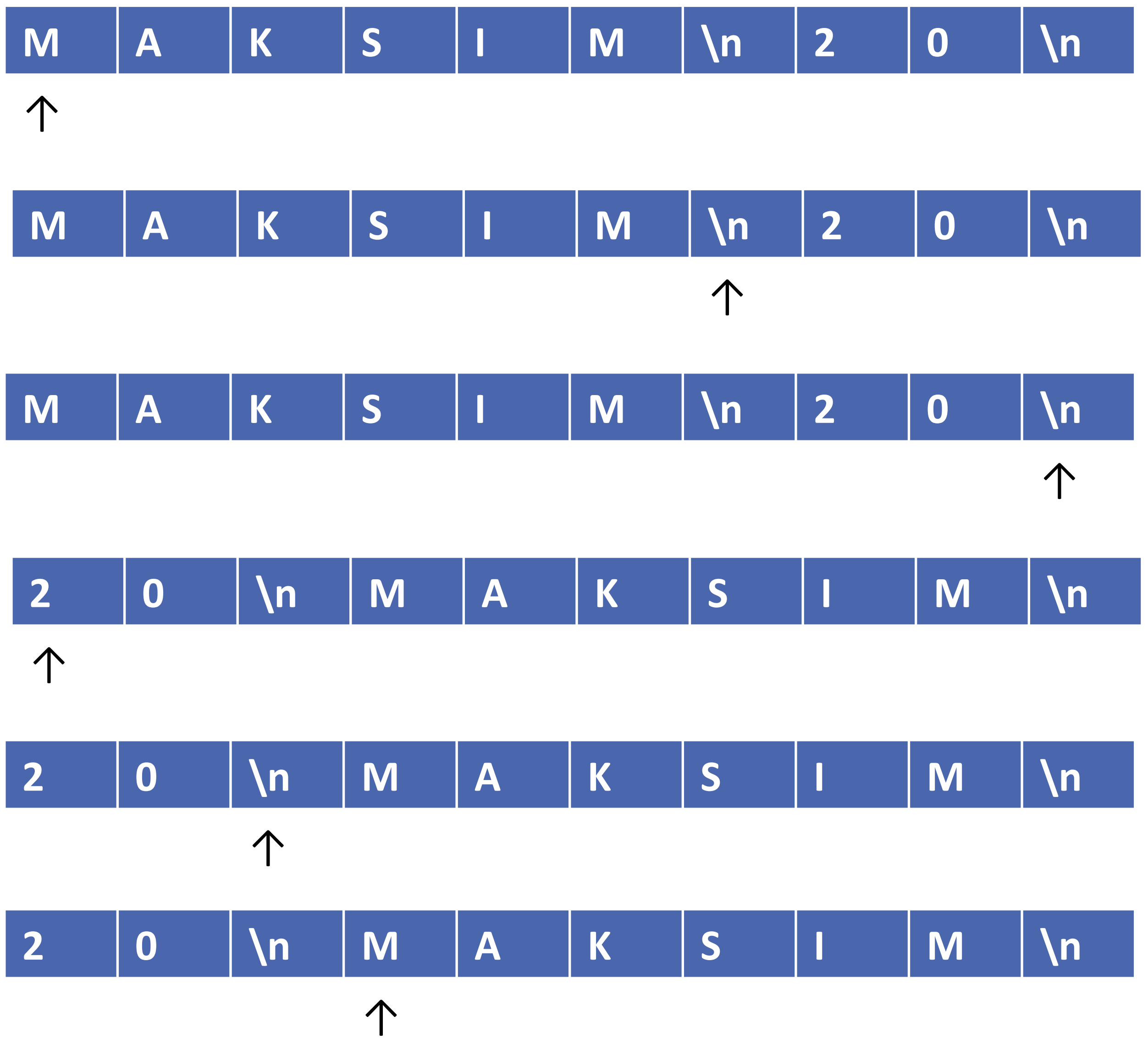
Scanner Methods for different numeric inputs

Method	Description
$nextByte()$	Reads an integer of $byte$ type
$nextShort()$	Reads an integer of $short$ type
$nextInt()$	Reads an integer of int type
$nextLong()$	Reads an integer of $long$ type
$nextFloat()$	Reads a number of $float$ type
$nextDouble()$	Reads a number of $double$ type
$nextLine()$	Reads till $\backslash n$ is encountered

Demonstration 3

- Compute area of Circle with Console Input
- ScannerDemo
- NumericInput – For your reference only
- IntegerBeforeString

Reading Integer Before String



Stream of Characters

$name = input.nextLine()$

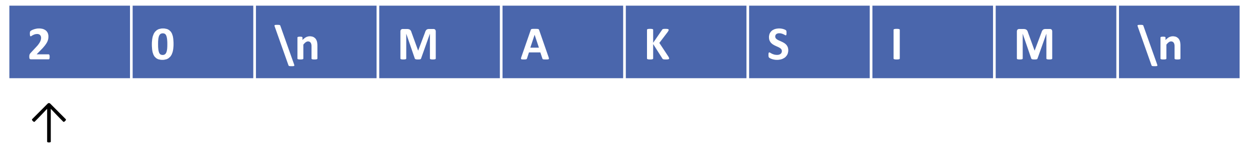
$age = input.nextInt()$

Stream of Characters

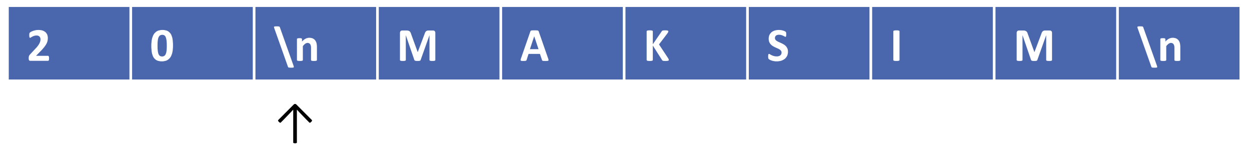
$age = input.nextInt()$

$name = input.nextLine()$

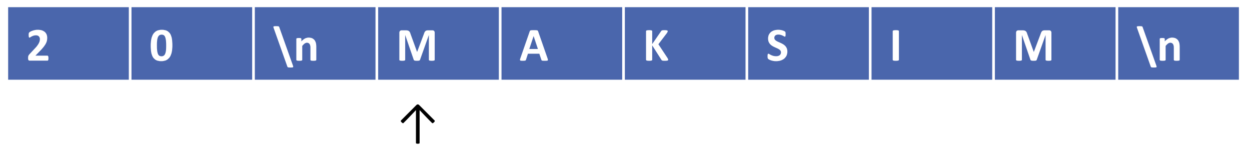
Reading Integer Before String



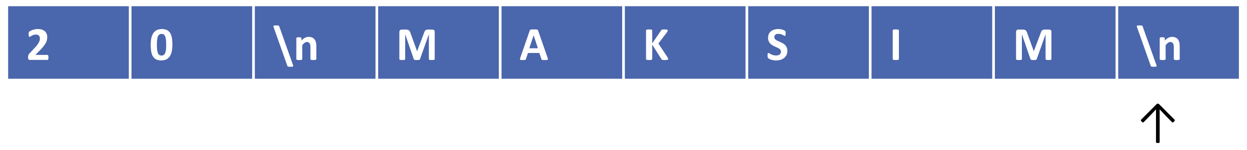
Stream of Characters



$age = input.nextInt()$



$input.nextLine()$



$name = input.nextLine()$

Default Value of Data Types

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

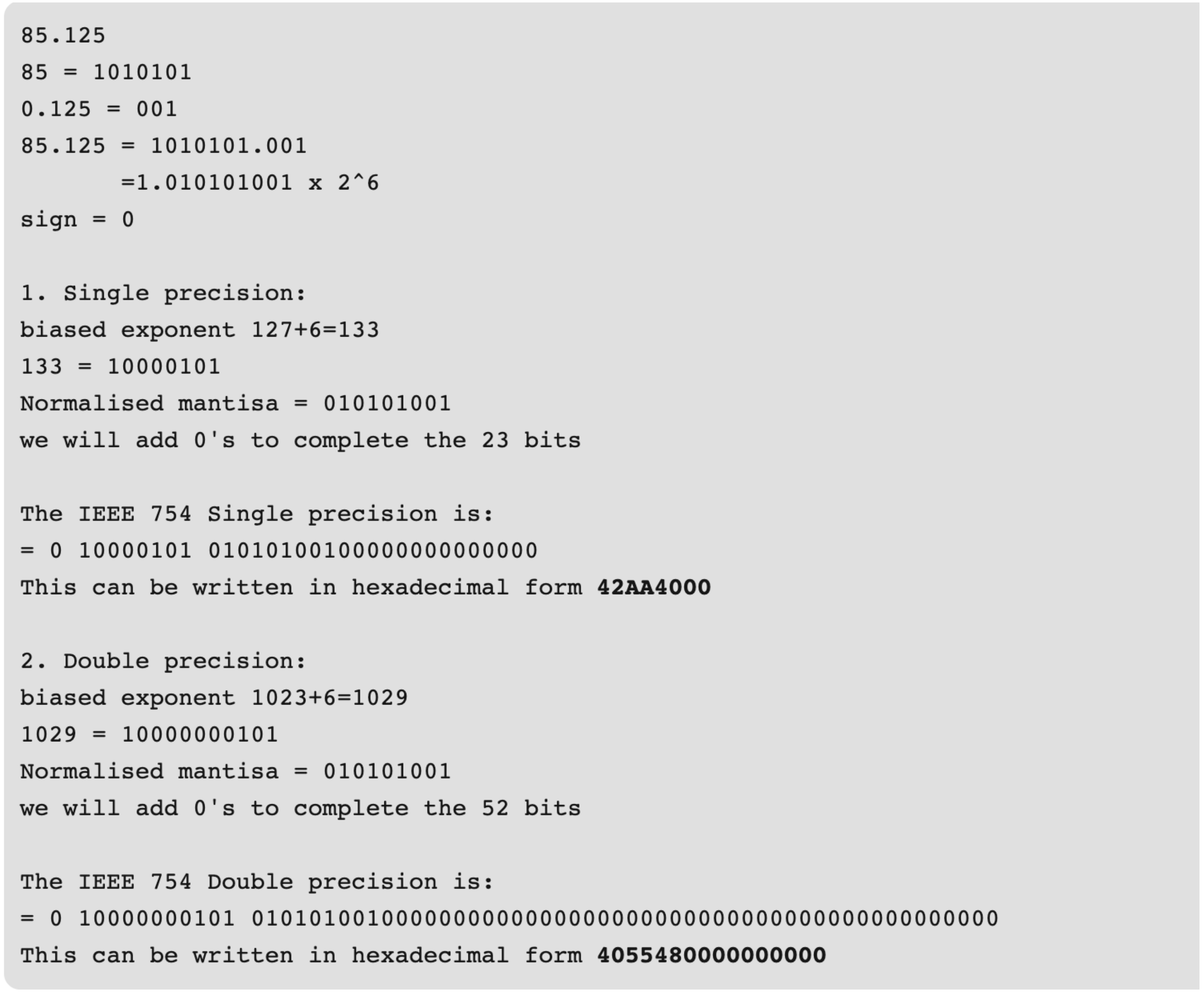
Never rely on these values! Always assign values with declaration. If that is not possible, assign before using them, otherwise it will result in compilation error.

Demonstration 4: UsedBeforeAssign.jav

IEEE 754 Floating Point Standard

- **The Sign of Mantissa**
  This is as simple as the name. 0 represents a positive number while 1 represents a negative number.
- **The Biased exponent –**
  The exponent field needs to represent both positive and negative exponents. A bias is added to the actual exponent in order to get the stored exponent.
- **The Normalised Mantissa –**
  The mantissa is part of a number in scientific notation or a floating-point number, consisting of its significant digits. Here we have only 2 digits, i.e. O and 1. So a normalised mantissa is one with only one 1 to the left of the decimal.

IEEE 754 Floating Point Standard



```
85.125
85 = 1010101
0.125 = 001
85.125 = 1010101.001
       =1.010101001 x 2^6
sign = 0

1. Single precision:
biased exponent 127+6=133
133 = 10000101
Normalised mantisa = 010101001
we will add 0's to complete the 23 bits

The IEEE 754 Single precision is:
= 0 10000101 01010100100000000000000
This can be written in hexadecimal form 42AA4000

2. Double precision:
biased exponent 1023+6=1029
1029 = 10000000101
Normalised mantisa = 010101001
we will add 0's to complete the 52 bits

The IEEE 754 Double precision is:
= 0 10000000101 0101010010000000000000000000000000000000000000000000
This can be written in hexadecimal form 4055480000000000
```