Java Building Elements

Dr. H. B. Prajapati

Associate Professor
Department of Information Technology
Dharmsinh Desai University

30 June '20

Core Java Technology

Table of contents

Outline of Presentation

Console Input and Output

- Any program takes some input and produces some output
- Java associates System.out with display and System.in with keyboard
- Earlier, we have seen that we can generate output in Java program using System.out.println(). We will also learn formated output.
- Java program can take input in two ways:
  - Command-line arguments (before the program starts execution)
  - Input at runtime (while program is executing)

Console Input using Command-line Arguments

Java program: Greetings.java

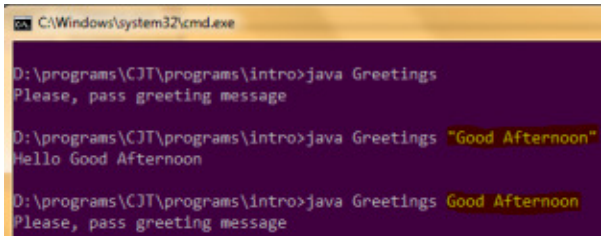
```
1 class Greetings{
2    public static void main(String[] args){
3       if(args.length!=1)
4          System.out.println("Please, pass
               greeting message");
5       else{
6          System.out.println("Hello "+args[0]);
7       }
8    }
9 }
```


Console Input using Command-line Arguments

Java program: Greetings.java

We test our program with three cases

1. No command-line arguments
2. One command-line argument
3. More than one command-line arguments



Console Input at Runtime

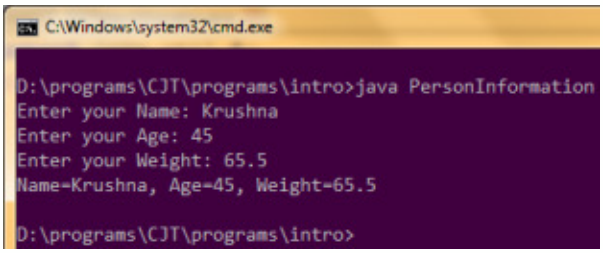
Java program: PersonInformation.java

```
1  import java.util.*;
2  class PersonInformation{
3     public static void main(String[] args){
4        String name;
5        int age;
6        float weight;
7        Scanner input = new Scanner(System.in);
8        System.out.print("Enter your Name: ");
9        name = input.next();
10       System.out.print("Enter your Age: ");
11       age = input.nextInt();
12       System.out.print("Enter your Weight: ");
13       weight = input.nextFloat();
14       System.out.println("Name="+name+",
            Age="+age+", Weight="+weight);
15    }
16 }
```


Console Input at Runtime, Running the Program

Java program: PersonInformation.java

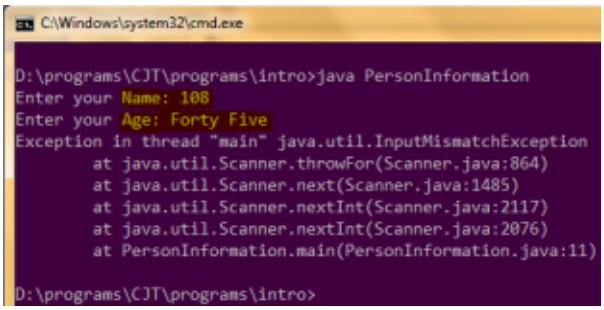
We run the program and input all correct values.



Console Input at Runtime, Running the Program

Java program: PersonInformation.java

We run the program and input values in wrong data type.



Console Input Methods

```
1 Scanner input = new Scanner(System.in);
2 int intValue = input.nextInt();
3 long longValue = input.nextLong();
4 double doubleValue = input.nextDouble();
5 float floatValue = input.nextFloat();
6 String string = input.next();
```

- In earlier version of Java, Scanner class was not available. We had to use java.io package for keyboard input (Will be discussed in Input/Output)
- Java has added Scanner utility class for console input. Available in java.util package
- What is a package?
  - Package is a grouping concepts, in which we can place related classes under one group.

Console Output: Formatted Output

- Sometimes, we need display/rendering of data in a desired format.
- Examples:
  - Number as right justified to make digits on multiple lines aligned
  - Amount in Rs. (float or double value) has only two digits after decimal point
  - Percentage value has only two digits after decimal point
- In C language (procedural language), this formating is provided by printf() function
- In Java printf() is available as System.out.printf() (as 100% object oriented, the printf() method is not standalone)
- The printf() in Java is similar to the printf() in C.

Method for Formatted Output

- The syntax of printf() function is as follows:

```
1 System.out.printf(format, item1, item2, ...,
     itemK);
```

- Where the format is a string that may have substrings and format specifiers
- A format specifier specifies how an item should be displayed.
- An item could be
  - a numeric (int, long, float, or double)
  - a Boolean value
  - a character
  - a string

Format Specifiers

- A simple specifier has percentage sign (%) followed by one letter character code for the type of data.

Specifier	Output	Example
%b	a Boolean value	true or false
%c	a character	a
%d	a decimal integer	108
%e	a number in scientific notation	3.2320000e+01
%f	a floating point number	32.320000
%s	a string	Java is powerful

Program on Formatting output

```
1 class FormattedOutput{
2     public static void main(String[] args){
3         int rollNo = 1;
4         float percent = 78.333333f;
5         System.out.printf("Roll No. %d has %f
              percentage",rollNo,percent);
6     }
7 }
```

- The items must match the specifiers in order, in number, and in type.
- By default, floating-point number is displayed with six digits after decimal point.

```
D:\programs\CJT\programs\intro>javac FormattedOutput.java

D:\programs\CJT\programs\intro>java FormattedOutput
Roll No. 1 has 78.333336 percentage
```


Specifying Width and Precision

- We can specify width (number of characters to be displayed) for an item to be displayed
- For floating-point number, we can also specify precision (number of digits after decimal point).
- In general, width indicates at least how many characters to be displayed:
  - If number of characters in the item is less than the width, extra whites-space characters are added at the front
  - Otherwise, the width is increased to accommodate the characters of the item.
- Examples:
  - %5c: character with total width 5, i.e., add four spaces
  - %6b: boolean with total width 6, i.e., add one space for false and two spaces for true.
  - %12s: string of at least 12 characters
  - %10.2f: total width of floating-point at least 10 and precision 2

Program on Formatting output: width and precision

```
1  class FormattedOutputWidthPrecision{
2      public static void main(String[] args){
3          String name = "Kisan";
4          char gender = 'M';
5          int age = 45;
6          float amount = 48000.9900999f;
7          boolean futureVisit = false;
8          System.out.println("# # # # # # # # # #
               # # #");
9          System.out.printf("Name:%10s\n",name);
10         System.out.printf("Gender:%3c
               Age:%3d\n",gender,age);
11         System.out.printf("Paid
               Amount:%10.2f\n",amount);
12         System.out.printf("To visit in
               future:%6b\n",futureVisit);
13     }
14 }
```


Program on Formatting output: width and precision

```
D:\programs\CJT\programs\intro>javac FormattedOutputWidthPrecision.java

D:\programs\CJT\programs\intro>java FormattedOutputWidthPrecision
# # # # # # # # # # # #
Name:     Kisan
Gender:  M   Age: 45
Paid Amount:  48000.99
To visit in future: false
```


Outline of Presentation

Identifiers

- Every entity in the real world is identifiable with a name.

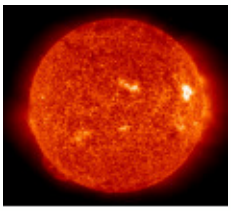


Figure : Sun

Figure : Krushna

- In program also we have to choose names for things/entities that we want to refer.
- Programming languages use special symbols, called identifiers, for naming programming entities.

Identifiers

- We assign names to the following programming entities:
  - variables
  - constants
  - methods
  - classes
  - interfaces
  - packages

Rules for Naming Identifiers

There are rules for naming identifiers, like we have rules for naming persons or objects

- An identifier must start with a letter (a-z,A-Z), an underscore (_), or a dollar sign ($).
- An identifier cannot contain operators, such as +, -, *, etc.
- An identifier cannot be a reserved word
- An identifier cannot be true, false, or null
- An identifier can be of any length

Valid and Invalid Identifiers

Valid identifiers

- $2, _age, newAge
- Char, α (unicode letter allowed)
- X, x (both are different, Java is case sensitive)
- ____2_w, _$
- is_umbrella_open_during_rain

Invalid identifiers

- -d, :b, e#
- 1rank, .com
- new, float, class, private, public, protected
- is#umbrella#open#during#rain

Variables

- Variables are used to store data for input, output, and intermediate result.
- Declaring a variable
  - Before using a variable, we have to declare it
  - We need to specify both name and data type for the variable
  - The syntax is dataType variableName;
  - Example: int x;
- Assignment Statements
  - After a variable is declared, we can assign some value to it
  - The syntax is variableName = value; or variableName = expression;, where expression could be any valid expression or a method call
  - Example: area = radius*radius*3.14;
- Combining declaration and initialization
  - We can combine variable declaration and value assignment in a single step
  - Example: int age = 50;

Constants

- The value of a variable can change during the execution of the program.
- However, the value of a constant, cannot change once assigned
- Example: final int ENTRY_AGE = 5;
- Constants declared as class members are made static.
- Example: static final int ENTRY_AGE = 5;

Numerical Data Types

- Each data type has a domain(range) of valid values
- The compiler allocates memory to a variable or constant based on its data type
- Java provides several primitive data types for numerical values, characters, and boolean values

Data type	Domain	Storage size
byte	-2^7 to 2^7+1	8-bit signed
short	-2^{15} to $2^{15}+1$	16-bit signed
int	-2^{31} to $2^{31}+1$	32-bit signed
long	-2^{63} to $2^{63}+1$	64-bit signed
float	$-3.4E38$ to $+3.4E38$ (6 to 7 significant digits of accuracy)	32-bit IEEE 754
double	$-1.7E308$ to $+1.7E308$ (14 to 15 significant digits of accuracy)	64-bit IEEE 754

Use of Numeric Data Types

```
1 int i1 = 22 + 1;          // i1 becomes 23
2 float d1 = 22.0f - 0.1f;  // d1 becomes 21.9
3 long i2 = 300*30 ;        // i2 becomes 9000
4 double d2 = 1.0/2.0;      // d2 becomes 0.5
5 int i3 = 1/2;             // i3 becomes 0
6 byte i4 = 21%2;           // i4 becomes 1
```


Numerical Literals

- A literal is a primitive type value that directly appears in the program.
- For example, 23, 1000000, and 10.0 are literals

```
1    int i = 23;
2    long l = 1000000;
3    double d = 10.0;
```

- By default floating point number (with decimal point) is considered as a double number.
- We can indicate a number literal as float by appending f or F, and double by appending d or D.

```
1    float ff = 23.0f;
2    double dd = 10.0D;
```


Numeric Type Conversion

- In an expression $a+b$, a and b are called operands and $+$ is called operator.
- It is possible that operands are of different data types.
- Java automatically converts different types into unifying type.
- Java chooses unifying type in the following order: double, float, long, int, short, and byte.
- For example, if one of operands is of type double, the unifying type will be double.

```
1    byte bVar = 108;
2    long lVar = i * 3 + 4;
3    double dVar = i * 3.1 + 1/2;
```

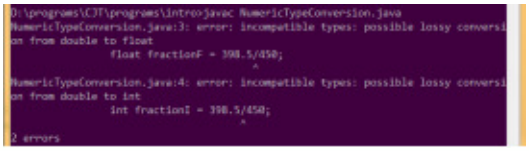

Numeric Type Conversion, Slide - I

- Sometimes, we may want to convert the result of the expression into some desired data type.
- For example, converting double result into float or float result into integer.

```
1 class NumericTypeConversion{
2    public static void main(String[] args){
3       float fractionF = 398.5/450;
4       int fractionI = 398.5/450;
5       System.out.println("Fraction in Integer
           "+fractionI);
6       System.out.println("Fraction in Float
           "+fractionF);
7    }
8 }
```


Numeric Type Conversion, Slide - II

- The program does not compile, and generate the following errors:



Numeric Type Conversion, Slide - III

- We use type casting for converting result of the expression into desired/target data type.
- Syntax $(target - datatype)expression$.

```
1 class NumericTypeConversion{
2    public static void main(String[] args){
3       float fractionF = (float) 398.5/450;
4       int fractionI = (int) 398.5/450;
5       System.out.println("Fraction in Integer
           "+fractionI);
6       System.out.println("Fraction in Float
           "+fractionF);
7    }
8 }
```


Numeric Type Conversion, Slide - IV

- The program now compiles successfully and generates the following output:

```
D:\programs\CJT\programs\intro>javac NumericTypeConversion.java

D:\programs\CJT\programs\intro>java NumericTypeConversion
Fraction in Integer 0
Fraction in Float 0.88555557
```

- The result of the integer fraction is 0 as the fractional part from the result is dropped.

Character Data Type, Slide - I

- The character data type, char, is used to represent a single character.
- A character literal is indicated by enclosing a character in single quotes.

```
1 char grade='A';
2 char position = '3';
```


Character Data Type, Slide - II

- An integer value should not be assigned to a character data type.

```
1 class CharDataType{
2    public static void main(String[] args){
3       char position = 65;
4       System.out.println("Position =
           "+position);
5    }
6 }
```

- The output would not be as expected:

```
D:\programs\CJT\programs\intro>javac CharDataType.java

D:\programs\CJT\programs\intro>java CharDataType
Position = A
```


Unicode Characters, Slide - I

- Java uses Unicode characters
- It is 16-bit encoding scheme established by Unicode Consortium
- Unicode can support interchange, processing, and display of texts of the world languages.
- As Unicode occupies 2 bytes, Java's character occupies 2 bytes
- Unicode literal is indicated using 2-byte hex numbers with \u as prefix.
- Unicode range is '\u0000' to '\uFFFF', in which ASCII code is included in the range '\u0000' to '\u00FF'.
- Unicode can be found at `http://www.unicode.org`

Unicode Characters, Slide - II

```
1 class Unicode{
2    public static void main(String[] args){
3       int h = 72;
4       char charH = (char) h ;
5       char uniH = '\u0048';
6       System.out.println("charH = "+charH+"
           uniH = "+uniH);
7    }
8 }
```

```
D:\programs\CJT\programs\intro>javac Unicode.java

D:\programs\CJT\programs\intro>java Unicode
charH = H uniH = H
```

Boolean Data Type

- The boolean data type is from Boolean algebra
- The domain of boolean type has only two values: true and false, which are used as literals while assigning value.

```
1 boolean isWorking = true;
```


Outline of Presentation

Shortcut Operators, Slide - I

- Many times, we want to update value of a variable by some amount.

```
1    i = i + 20;
```

- We can perform same expression using less characters.

```
1    i += 20;
```

- Here, += is called shortcut operator.

Operator	Example	Equivalent
+=	i+=5;	i = i+5;
-=	f-=5.0;	f = f+5.0;
*=	i*=5;	i = i*5;
/=	i/=5;	i = i/5;
% =	i%=5;	i = i%5;

Shortcut Operators, Slide - II

- There are two more shortcut operators for incrementing or decrementing by 1.
  - ++ is for incrementing by 1
  - -- is for decrementing by 1
- Depending upon where (before or after) they are placed along with variable name, there are two types of operation:
  1. Prefix: ++x
  2. Postfix x++

Example	Equivalent
x++	x = x+1;
++x	x = x+1;
x--	x = x-1;
--x	x = x-1;

Shortcut Operators, Slide - III

- The prefix ++x and suffix or postfix x++ are different when they are in expression.
- The prefix is evaluated before other operators in an expression
- The postfix or suffix is evaluated after entire expression is evaluated.

```
1  class PrefixPostfix{
2     public static void main(String[] args){
3        int i=0, j=0;
4        int valPre, valPost;
5        valPre = ++i + ++i + ++i + ++i;
6        valPost = j++ + j++ + j++ + j++;
7        System.out.println("Values: i = "+i+"
            valPre = "+valPre);
8        System.out.println("Values: j = "+j+"
            valPost = "+valPost);
9     }
10 }
```


Shortcut Operators, Slide - IV

- When we run the program, we get the following output

```
D:\programs\CJT\programs\intro>javac PrefixPostfix.java

D:\programs\CJT\programs\intro>java PrefixPostfix
Values: i = 4 valPre = 10
Values: j = 4 valPost = 6
```

- It can be observed that after evaluation of expressions, values of both i and j become 4; however, values of valPre and valPost are different.

Comparison Operators

- Comparison operators evaluate to boolean value

Operator	Name	Example	Answer
$<$	less than	$21 < 23$	true
$<=$	less than or equal to	$21 <= 21$	true
$>$	greater than	$21 > 23$	false
$>=$	greater than or equal to	$21 >= 21$	true
$==$	equal to	$21 == 23$	false
$!=$	not equal to	$21 != 23$	true

Boolean Operators, Slide - I

- Boolean operators operate on boolean values and result in a new boolean value.

Operator	Name	Description
!	not	logical negation
&&	and	logical conjunction
||	or	logical disjunction
^	exclusive or	logical exclusive

- The not operator (!) is unary operator, whereas other three are binary operators.

Boolean Operators, Slide - II

Table : Truth values for Operator !

Operand	!Operand
true	false
false	true

Table : Truth values for Operator &&

Operand-1	Operand-1	Operand-1 && Operand-2
false	false	false
false	true	false
true	false	false
true	true	true

Boolean Operators, Slide - III

Table : Truth values for Operator ||

Operand-1	Operand-1	Operand-1 || Operand-2
false	false	false
false	true	true
true	false	true
true	true	true

Table : Truth values for Operator ^

Operand-1	Operand-1	Operand-1 ^ Operand-2
false	false	false
false	true	true
true	false	true
true	true	false

Boolean Operators, Slide - IV

- When evaluating b1 && b2, Java first evaluates b1
  - If b1 is false, b2 is not evaluated
  - If b1 is true, then b2 is evaluated
- When evaluating b1 || b2, Java first evaluates b1
  - If b1 is true, b2 is not evaluated
  - If b1 is false, then b2 is evaluated
- Java provides other operators & and |
  - These operators are similar to && and ||
  - The difference is in & and |, second operand is also evaluated irrespective of value of the first operand.

Boolean Operators, Slide - V

```
1  class BooleanOperators{
2     public static void main(String[] args){
3        boolean hungry = true;
4        int eatDish = 0;
5        if(hungry && ++eatDish >= 2)
6           hungry = false;
7        if(hungry && ++eatDish >= 2)
8           hungry = false;
9        if(hungry && ++eatDish >= 2)
10          hungry = false;
11       System.out.println("Now hungry =
            "+hungry+", Consumed dish =
            "+eatDish);
12    }
13 }
```

Boolean Operators, Slide - VI

```
D:\programs\CJT\programs\intro>javac BooleanOperators.java

D:\programs\CJT\programs\intro>java BooleanOperators
Now hungry = false, Consumed dish = 2
```


Boolean Operators, Slide - VII

```
1  class BooleanOperators{
2     public static void main(String[] args){
3        boolean hungry = false;
4        int eatDish = 0;
5        if(hungry & ++eatDish >= 2)
6           hungry = false;
7        if(hungry & ++eatDish >= 2)
8           hungry = false;
9        if(hungry & ++eatDish >= 2)
10          hungry = false;
11       System.out.println("Now hungry =
            "+hungry+", Consumed dish =
            "+eatDish);
12    }
13 }
```


Boolean Operators, Slide - VIII

```
D:\programs\CJT\programs\intro>javac BooleanOperators.java

D:\programs\CJT\programs\intro>java BooleanOperators
Now hungry = false, Consumed dish = 3
```

- We should avoid using & and | operators

Operator Precedence, Slide - I

- It is possible to write an arbitrary long expression, containing different operators
- Operator precedence determines the order in which expressions are evaluated.
- Parenthesis has highest priority. In nested parenthesis, the innermost parenthesis has highest priority.

Operator Precedence, Slide - II

Table : Operator precedence from highest to lowest

() parenthesis, () function call, [] array subscript, . member access
++,--, + unary plus, - unary minus, ! unary logical negation, (type) unary casting, new create object
*, /, %
+, -
<, <=, >, >=, instanceof
==, !=
&&
||
?: ternary condition
=, +=, -=, *=, /=, %=

Operator Precedence, Slide - III

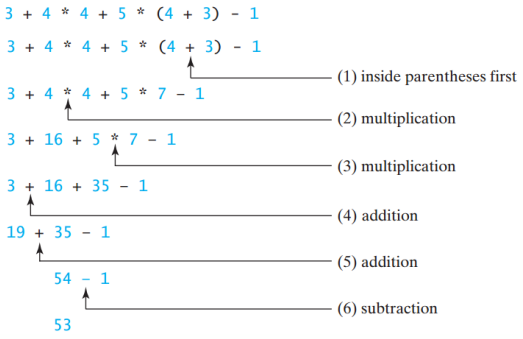


Figure : Example of Precedence [1]

[1] Source: An Introduction to Java Programming, Y. Daniel Liang, Eigth Edition, Prentice Hall

Outline of Presentation

Errors in Program

- Even experienced programmer might make mistakes.
- The errors can be categorized into two:
  1. Compilation Errors
  2. Runtime Errors
- Compilation errors are reported by compiler
- Runtime errors occur while program is running

Compilation Errors, Slide - I

- Errors that occur during compilation are called compilation errors or syntax errors
- These are easy to detect as the compiler tells the location and reason for errors.
- Consider the following program:

```
1 class CompilationError{
2    public static void main(String[] args){
3       i = 100;
4       System.out.println(i+8);
5    }
6 }
```


Compilation Errors, Slide - II

```
D:\programs\CJT\programs\intro>javac CompilationError.java
CompilationError.java:3: error: cannot find symbol
                i = 100;
                ^
  symbol:   variable i
  location: class CompilationError
CompilationError.java:4: error: cannot find symbol
                System.out.println(i+8);
                                   ^
  symbol:   variable i
  location: class CompilationError
2 errors
```

- We can see that the compiler shows errors at two places, where we are accessing variable i.
- The reason for errors is that we have not declared variable i. It should be int i;
- It is possible that for a single error in our program, the compiler shows multiple errors.
- Therefore, we should start troubleshooting from the first error.

Runtime Errors

- Runtime errors occur while program is running
- These errors generally cause programs to terminate abnormally
- There are three main reasons for runtime errors:
  1. Input errors
  2. System software or hardware errors
  3. Logical errors

Runtime Errors

Input Errors

- An input error occurs when the user enters an unexpected input value and the program cannot handle it.
- For example, in our earlier program on PersonInformation, when we enter unexpected value for age, we get runtime error

```
C:\Windows\system32\cmd.exe

D:\programs\CJT\programs\intro>java PersonInformation
Enter your Name: 100
Enter your Age: Forty Five
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:864)
        at java.util.Scanner.next(Scanner.java:1485)
        at java.util.Scanner.nextInt(Scanner.java:2117)
        at java.util.Scanner.nextInt(Scanner.java:2076)
        at PersonInformation.main(PersonInformation.java:11)

D:\programs\CJT\programs\intro>
```

- We should provide informative message to the user. For example, "Enter your age (integer value)".

Runtime Errors

System Software or Hardware Errors

- System errors are rarely encountered.
- However, sometimes, unreliable system software and hardware malfunctions can cause a program to abort.
- System errors are beyond programmer's control
- For mission-critical applications, system reliability should be achieved using fault tolerance system

Runtime Errors

Logical Errors

- Logical errors either generate incorrect results or terminate the program abnormally.
- These errors are called bugs.
- The process of finding logical errors (bugs) is called debugging.
- Finding logical errors is a difficult task
- IDEs are equipped with debugger, which can be used for step-by-step execution and watching/inspecting values of variables.
- A common approach is to locate code region generating logical errors and then to reach at the precise cause point.

Outline of Presentation

Programming Style and Documentation

- Programming style deals with the appearance of the program.
- It is possible to write the whole program on just one line.
- However, it is difficult to read and understand; therefore, it is a bad programming style.
- Programming Style and Documentation are as important as coding
- There are some guidelines for Java programming style and documentation.

Appropriate Comments

- We should place summary at the beginning of the program.
- The summary can include major information about
  - key features
  - supporting data structures
  - unique/special logic or algorithm it uses
- We should write comments before each major step in a long program
- We should write comments for each complicated or complex part or logic
- The comments should be concise and easy to read.

Proper Indentation I

- An unindented program is difficult to read
- Using indentation, we can show structural relationships among the program's components or statements.

```
1 class Greetings{
2 public static void main(String[] args){
3 if(args.length!=1)
4 System.out.println("Please, pass greeting
    message");
5 else{
6 System.out.println("Hello "+args[0]);
7 }
8 }
9 }
```

Proper Indentation II

- The following is the same program, but properly indented
- The program is very easy to read and understand

```
1 class Greetings{
2    public static void main(String[] args){
3       if(args.length!=1)
4          System.out.println("Please, pass
               greeting message");
5       else{
6          System.out.println("Hello "+args[0]);
7       }
8    }
9 }
```


Naming Conventions - I

- Classes and interfaces
  - The first letter should be capitalized
  - If several words are linked together to form the name, the first letter of the inner words should be uppercase
  - This format is sometimes called camelCase
  - Examples:
    - Person
    - Account
    - PrintWriter
- Methods:
  - The first letter should be lowercase
  - Then normal camelCase rules should be used
  - the names should typically be verb-noun pairs
  - Examples:
    - getBalance

Naming Conventions - II

    - doCalculation
    - setCustomerName
- Variables:
  - similar to methods, the camelCase format should be used (the first letter is lower case)
  - Short and meaningful names are recommended
  - Examples:
    - buttonWidth
    - accountBalance
    - myString
- Constants:
  - Java constants are created by marking variables static and final
  - They should be named using uppercase letters with underscore characters as separators
  - Examples:
    - MIN_HEIGHT
    - MAX_MARKS

Summary of key terms

- Scanner (Console input), Command-line arguments
- Identifiers, variables, constants, numerical data types, character data type, Unicode, Boolean data type
- Operators: shortcut, comparison, boolean (logical), precedence
- Error handling: compilation, runtime
- Indentation, naming convention, comments

References

- An Introduction to Java Programming, Y. Daniel Liang, PHI
- An Introduction to Java Programming, Y. Daniel Liang, Eigth Edition, Prentice Hall