



Java Fundamentals

The Parts of a Java program

Table of contents for this presentation

- 1 [Parts of a Java program](#)
- 2 [The print and println methods](#)
- 3 [Variables and Literals](#)
- 4 [Displaying Multiple Items with the + Operator](#)
- 5 [Identifiers](#)
- 6 [Variable and class names](#)
- 7 [Primitive Data Types](#)
- 8 [Arithmetic Operators](#)
- 9 [The Math Class](#)
- 10 [Combined Assignment Operators](#)
- 11 [Conversion between Primitive Data Types](#)
- 12 [Mixed Integer Operations](#)
- 13 [Creating Named Constants with final](#)
- 14 [The String Class](#)
- 15 [The Scope of a variable](#)
- 16 [Comments](#)
- 17 [The Scanner class](#)
- 18 [The JOptionPane Class](#)
- 19 [Converting String Input to Numbers](#)

Parts of a Java program

```
1 // A simple Java program
2
3 public class Parts
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("I love programming!");
8     }
9 }
```

Program output
I love programming!

Parts of a Java program

“//” marks the beginning of a comment



```
1 // A simple Java program
2
3 public class Parts
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("I love programming!");
8     }
9 }
```

Program output
I love programming!

Parts of a Java program

Insert blank lines in programs to make them easier to read

```
1 // A simple Java program
2
3 public class Parts
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("I love programming!");
8     }
9 }
```

Program output
I love programming!

Parts of a Java program

Class header marks the beginning of a class definition.
A class serves as a container for an application.

```
1 // A simple Java program
2
3 public class Parts
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("I love programming!");
8     }
9 }
```

Program output
I love programming!

Parts of a Java program

Is a java key word. It must be written in lowercase letters.

Known as an access specifier. The public specifier means access to the class is unrestricted (“Open to the public”).

```
1 // A simple Java program
2
3 public class Parts
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("I love programming!");
8     }
9 }
```

Program output
I love programming!

Parts of a Java program

Is a java key word. It must be written in lowercase letters.

Indicates the beginning of a class definition

```
1 // A simple Java program
2
3 public class Parts
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("I love programming!");
8     }
9 }
```

Program output
I love programming!

Parts of a Java program

The name of the class made up by the programmer.

```
1 // A simple Java program
2
3 public class Parts
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("I love programming!");
8     }
9 }
```

Program output
I love programming!

Parts of a Java program

In short: This line tells the compiler that a publicly accessible class named “Parts” is being defined.

```
1 // A simple Java program
2
3 public class Parts
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("I love programming!");
8     }
9 }
```

Program output
I love programming!

Parts of a Java program

Left brace or opening brace associated with the beginning of the class definition.

```
1 // A simple Java program
2
3 public class Parts
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("I love programming!");
8     }
9 }
```

Right brace or closing brace. Everything between the two braces is the body of the class named Parts.

Parts of a Java program

The method header marks the beginning of a method. The name of this method is “main”. The rest of the words are required for the method to be properly defined.

```
1 // A simple Java program
2
3 public class Parts
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("I love programming!");
8     }
9 }
```

Every Java application must have a method named “main”. This is the starting point of an application.

Parts of a Java program

Left brace or opening brace associated with the beginning of the main method.

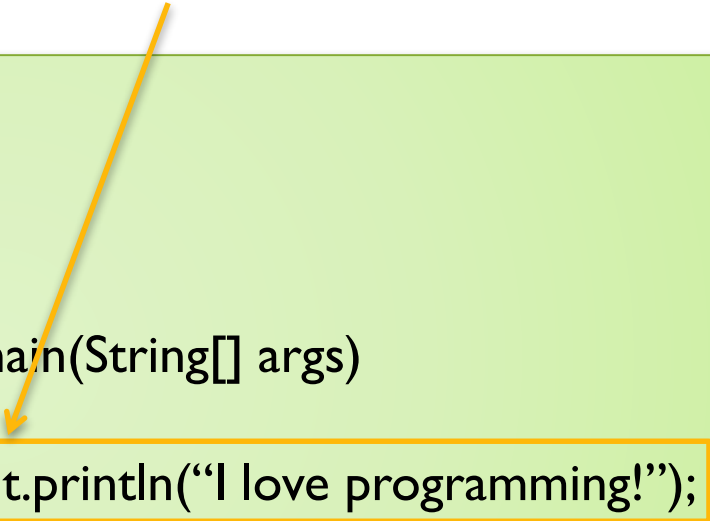
```
1 // A simple Java program
2
3 public class Parts
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("I love programming!");
8     }
9 }
```

Right brace or closing brace. Everything between the two braces is the body of the method main.

Parts of a Java program

In short: this line displays a message on the screen.
The message “I love programming!” is printed on the screen without any quotation marks.


```
1 // A simple Java program
2
3 public class Parts
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("I love programming!");
8     }
9 }
```



Parts of a Java program

At the end of the line is a semicolon. A semicolon marks the end of a statement in Java. Not every line of code ends with a semicolon.

```
1 // A simple Java program
2
3 public class Parts
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("I love programming!");
8     }
9 }
```





Java Fundamentals

The *print* and *println* methods

These methods are used to display text output. They are part of the Java *Application Programmer Interface* (API), which is a collection of prewritten classes and methods for performing specific operations in Java and are available to all Java programs.

The *print* and *println* methods

```
1 // A simple Java program
2
3 public class Parts
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("I love programming!");
8     }
9 }
```

The *print* and *println* methods

```
1 // A simple Java program
2
3 public class Parts
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("I love programming!");
8     }
9 }
```

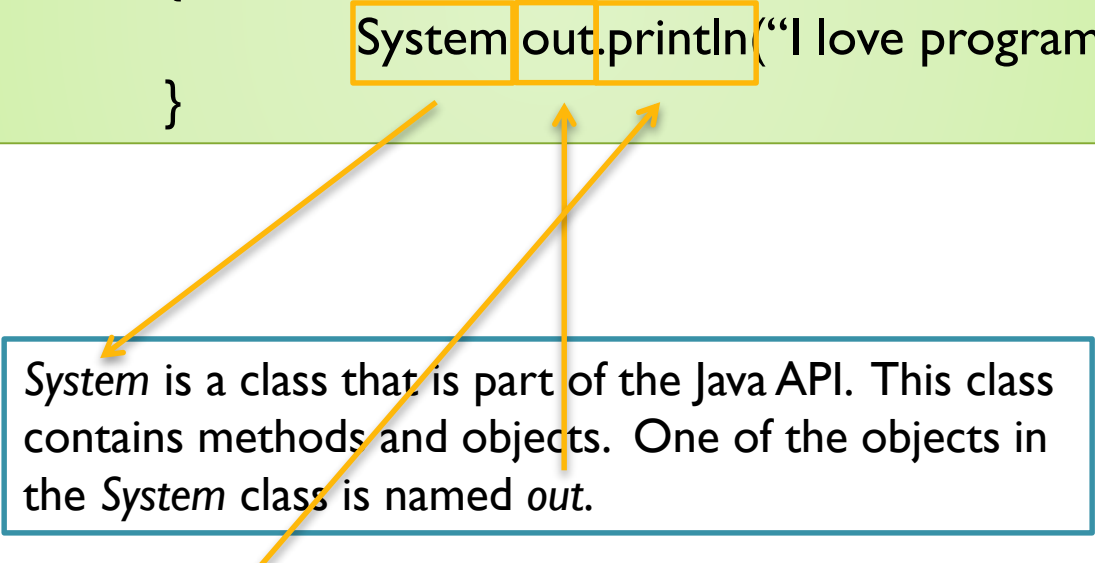
The *print* and *println* methods

```
5      public static void main(String[] args)
6      {
7          System out.println("I love programming!");
8      }
```

System is a class that is part of the Java API. This class contains methods and objects. One of the objects in the *System* class is named *out*.

The *print* and *println* methods

```
5      public static void main(String[] args)
6      {
7          System.out.println("I love programming!");
8      }
```



System is a class that is part of the Java API. This class contains methods and objects. One of the objects in the *System* class is named *out*.

The *out* object also has methods which include *print* and *println*.

The *print* and *println* methods

System class

Object *out*

Methods

print

println(" ");

• •

The *print* method

```
5      public static void main(String[] args)
6      {
7          System.out.print("I love programming!");
8      }
```

An important thing to know about the print method is that it does not advance the cursor to the beginning of the next line after displaying the message:

Printed on the computer screen when application runs:

I love programming!

The *print* method

```
5      public static void main(String[] args)
6      {
7          System.out.print("I love programming!");
8      }
```

An important thing to know about the print method is that it does not advance the cursor to the beginning of the next line after displaying the message:

Printed on the computer screen when application runs:

I love programming!|

The *println* method

```
5      public static void main(String[] args)
6      {
7          System.out.println("I love programming!");
8      }
```

An important thing to know about the *println* method is that it advances the cursor to the beginning of the next line after displaying the message:

Printed on the computer screen when application runs:

I love programming!

The *println* method

```
5      public static void main(String[] args)
6      {
7          System.out.println("I love programming!");
8      }
```

An important thing to know about the *println* method is that it advances the cursor to the beginning of the next line after displaying the message:

Printed on the computer screen when application runs:

```
I love programming!
|
```

The *print* and *println* methods

```
public static void main(String[] args)
{
    System.out.print("I love the following:");
    System.out.print("Sweets");
    System.out.print("Chips");
    System.out.print("Coffee");
}
```

Printed on the computer screen when application runs:

I love the following:SweetsChipsCoffee

The *print* and *println* methods

```
public static void main(String[] args)
{
    System.out.println("I love the following:");
    System.out.println("Sweets");
    System.out.println ("Chips");
    System.out.print("Coffee");
}
```

Printed on the computer screen when application runs:

```
I love the following:SweetsChipsCoffee
Sweets
Chips
Coffee
```

The *print* and *println* methods

```
public static void main(String[] args)
{
    System.out.println("I love the following:");
    System.out.println("Sweets");
    System.out.println ("Chips");
    System.out.print("Coffee");
}
```

Printed on the computer screen when application runs:

```
I love the following:
Sweets
Chips
Coffee|
```

The escape sequence

The escape sequence starts with a backslash character “\” followed by control characters (in this case the letter “n”)

```
System.out.print("I love the following:\n");  
System.out.print("Sweets\n");  
System.out.print ("Chips\n");  
System.out.print("Coffee\n");  
  
}
```

Printed on the computer screen when application runs:

```
I love the following:  
Sweets  
Chips  
Coffee
```

The escape sequence

The escape sequence starts with a backslash character “\” followed by control characters (in this case the letter “n”)

```
System.out.print("I love the following:\n");  
System.out.print("Sweets\n");  
System.out.print ("Chips\n");  
System.out.print("Coffee\n");  
  
}
```

Printed on the computer screen when application runs:

```
I love the following:  
Sweets  
Chips  
Coffee  
|
```



Java Fundamentals

Variables and Literals

A variable is a named storage location in the computer's memory. A literal is a value written into the code of a program.

Variables and Literals

```
1 // A program that uses variables
2
3 public class Variables
4 {
5     public static void main(String[] args)
6     {
7         int value;
8
9         value = 10;
10        System.out.print("The value is ");
11        System.out.println(value);
12
13    }
14 }
```

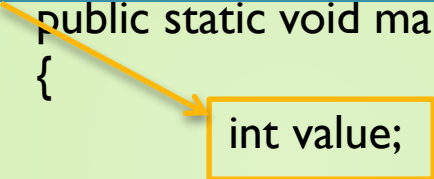
Printed on the computer screen when application runs:

The value is 10

Variables and Literals

We call this line of code a variable declaration. A variable must first be declared before it can be used. A variable declaration tells the compiler the variable's name and type of data it will hold.

```
5      public static void main(String[] args)
6      {
7          int value;
8
9          value = 10;
10         System.out.print("The value is ");
11         System.out.println(value);
12
13     }
14 }
```



Printed on the computer screen when application runs:

The value is 10

Variables and Literals

```
1 // A program that uses variables
2
3 public class Variables
4 {
5     public static void main(String[] args)
6     {
7         int value;
8
9         value = 10;
10        System.out.print("The value is ");
11        System.out.println(value);
12
13    }
14 }
```

The word *int* stands for integer which is the data type of the declaration.

Printed on the computer screen when application runs:

The value is 10

Variables and Literals

```
1 // A program that uses variables
2
3 public class Variables
4 {
5     public static void main(String[] args)
6     {
7         int value;
8
9         value = 10;
10        System.out.print("The value is ");
11        System.out.println(value);
12
13    }
14 }
```

The word *value* is a word the programmer chooses to hold integer numbers.

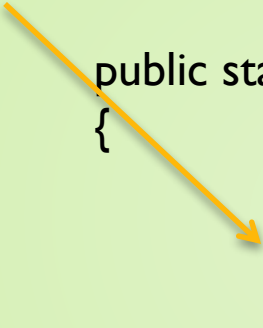
Printed on the computer screen when application runs:

The value is 10

Variables and Literals

This is called an assignment statement. The equal sign is an operator that stores the value on its right (10) into the variable on its left (value).

```
4      {
5          public static void main(String[] args)
6      {
7          int value;
8
9          value = 10;
10         System.out.print("The value is ");
11         System.out.println(value);
12
13     }
14 }
```



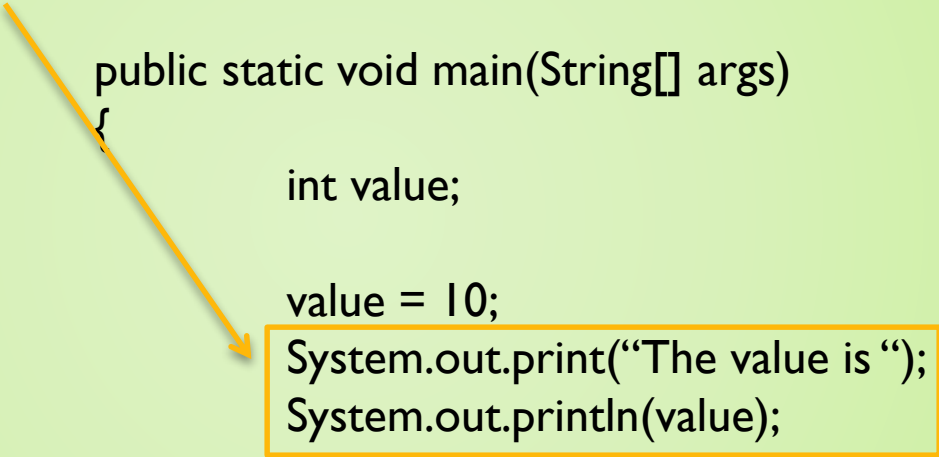
Printed on the computer screen when application runs:

The value is 10

Variables and Literals

The first line here sends the string literal “The value is “ to the *print* method. The second line sends the name of the *value* variable to the *println* method.

```
4      {
5          public static void main(String[] args)
6      {
7          int value;
8
9          value = 10;
10         System.out.print("The value is ");
11         System.out.println(value);
12
13     }
14 }
```



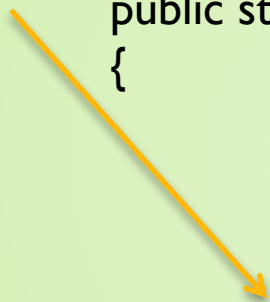
Printed on the computer screen when application runs:

The value is 10

Variables and Literals

When you send a variable to the *print* or *println* methods, the value of that variable will be displayed. Important: there are no quotation marks around the variable *value*!

```
5      public static void main(String[] args)
6      {
7          int value;
8
9          value = 10;
10     System.out.print("The value is ");
11     System.out.println(value);
12
13     }
14 }
```



Printed on the computer screen when application runs:

The value is 10

Displaying Multiple Items with the + Operator

When the + operator is used with strings ,we call it a *string concatenation operator*. To concatenate means to append. The string concatenation operator appends one string to another.

```
System.out.println("Today is " + "a good day!");
```

The + operator produces a string that is a combination of the 2 strings both sides of the operator.

Printed on the computer screen when application runs:

Today is a good day!

Displaying Multiple Items with the + Operator

We can also use the + operator to concatenate the contents of a variable to a string.

```
number = 857623;  
System.out.println("Today's lotto number is: " + number);
```

The + operator is used to concatenate the contents of the **number** variable with the string "**Today's lotto number is:**". The + operator converts the **number** variable's value from an integer to a string and then appends the new value.

Printed on the computer screen when application runs:

```
Today's lotto number is: 857623
```


Identifiers

Variable names and class names are examples of identifiers (represents some element of a program)

You should always choose names for your variables that give an indication of what they are used for:

```
int y;
```

This gives us no clue as to what the purpose of the variable is.

```
int numberOfCows;
```

```
int numberofcows;
```

numberOfCows gives anyone reading the program an idea of what the variable is used for.

Identifiers

The following rules must be followed with all identifiers:



The first character must be one of the letters **a-z**, **A-Z**, underscore “_”, or the dollar sign “\$”



After the first character, you may use the letters **a-z**, **A-Z**, underscore “_”, the dollar sign “\$”, and the digits 0-9



No spaces



Uppercase and lowercase characters are distinct.
This means that `numberOfCows` is not the same as `numberofcows`.

Variable and Class names

Variable

Start variable names with a lowercase letter

Each subsequent word's first letter must be capitalised

Example:

numberOfCows

Class

Start class names with an uppercase letter

Each subsequent word's first letter must be capitalised

Example:

FarmCows

Primitive Data Types

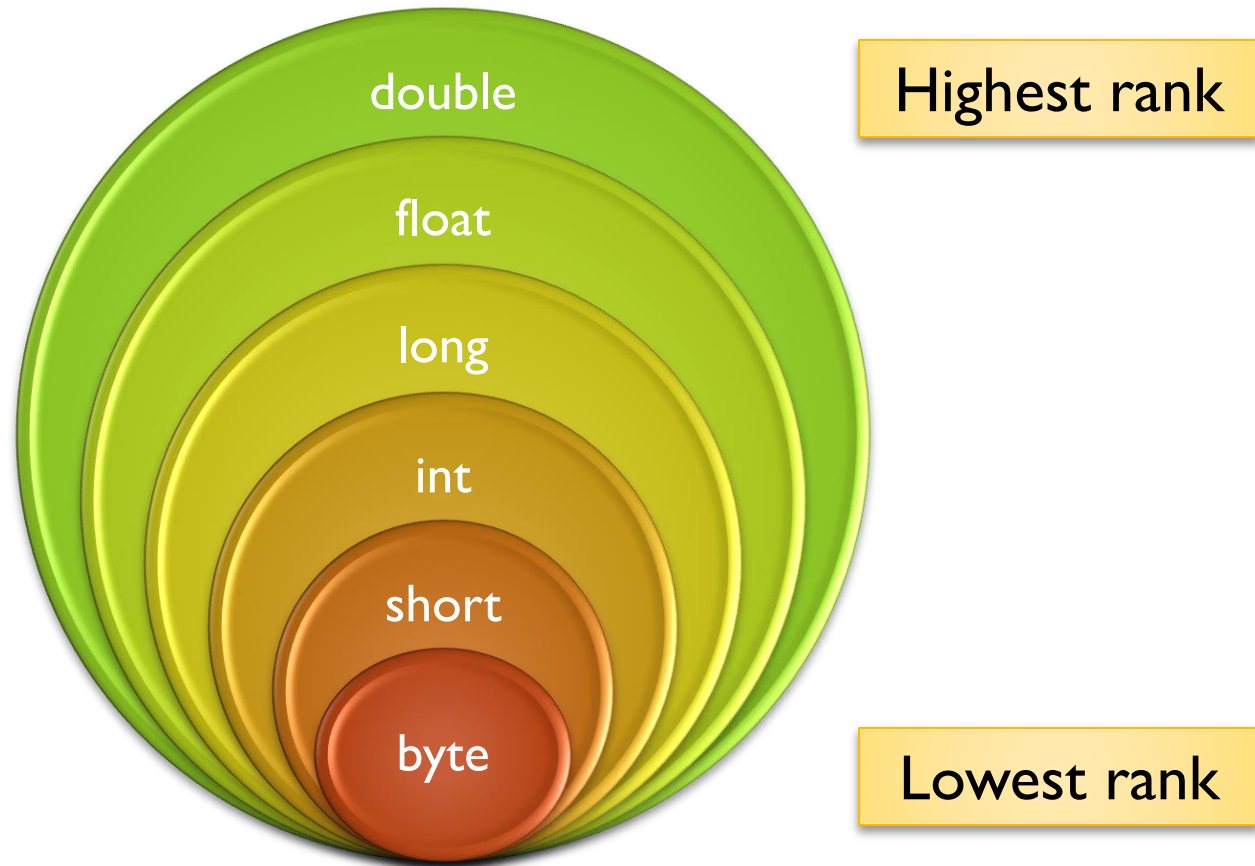
There are many different types of data. Variables are classified according to their data type. The data type determines the kind of data that may be stored in them.

The data type also determines the amount of memory the variable uses, and the way the variable formats and stores data.

Primitive Data Types

Data Type	Size	Range
byte	1 byte	Integers (-128 to +127)
short	2 bytes	Integers (-32,768 to +32,767)
int	4 bytes	Integers (-2,147,483,648 to +2,147,483,647)
long	8 bytes	Integers (-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807)
float	4 bytes	Floating-point numbers ($\pm 3.4E-38$ to $\pm 3.4E38$, 7 digits of accuracy)
double	8 bytes	Floating-point numbers ($\pm 1.7E-308$ to $\pm 1.7E308$, 15 digits of accuracy)

Primitive Data Types ranking



The Integer Data Types

The integer data types include ***byte***, ***short***, ***int*** and ***long***.

When you write an integer literal in your program code, Java assumes it to be of the ***int*** type. You can force an integer literal to be treated as a ***long*** by suffixing it with the letter **L**. Example: **14L** would be treated as a ***long***.

```
long numberOfCows;  
numberOfCows = 14;
```

Java will assume an ***int*** type

```
long numberOfCows;  
numberOfCows = 14L;
```

Forced to be of type ***long***

The Integer Data Types

```
1 public class IntegerVariables
2 {
3     public static void main(String[] args)
4     {
5         byte miles;
6         short minutes;
7         int temperatureLondon;
8         long days;
9
10        miles = 120;
11        minutes = 100;
12        temperatureLondon = -15;
13        days = 182500;
14
15        System.out.println("We have made a journey of " + miles +
16                            " miles in only " + minutes + " minutes.");
17        System.out.println("The temperature in London is " +
18                            temperatureLondon + " degrees.");
19        System.out.println("There are " + days + " days in 500 years.");
20    }
21 }
```


The Integer Data Types

```
1 public class IntegerVariables
2 {
3     public static void main(String[] args)
4     {
5         byte miles;
6         short minutes;
7         int temperatureLondon;
8         long days;
9
10        miles = 120;
11        minutes = 100;
12        temperatureLondon = -15;
13        days = 182500;
14
15        System.out.println("We have made a journey of " + miles +
16                            " miles in only " + minutes + " minutes.");
17        System.out.println("The temperature in London is " +
18                            temperatureLondon + " degrees.");
19        System.out.println("There are " + days + " days in 500 years.");
20    }
21 }
```

Variables declared as a certain type

Values assigned to variables

The Integer Data Types

Printed on the computer screen when application runs:

We have made a journey of 120 miles in only 100 minutes.

String literal

```
miles = 120;
minutes = 100;
temperatureLondon = -15;
days = 182500;

System.out.println("We have made a journey of " + miles +
    " miles in only " + minutes + " minutes.");
System.out.println("The temperature in London is " +
    temperatureLondon + " degrees.");
System.out.println("There are " + days + " days in 500 years.");
```

```
}
```

```
}
```

The Integer Data Types

Printed on the computer screen when application runs:

We have made a journey of 120 miles in only 100 minutes.

Concatenation (+) operator

```
10      miles = 120;
11      minutes = 100;
12      temperatureLondon = -15;
13      days = 182500;
14
15      System.out.println("We have made a journey of " + miles +
16                          " miles in only " + minutes + " minutes.");
17      System.out.println("The temperature in London is " +
18                          temperatureLondon + " degrees.");
19      System.out.println("There are " + days + " days in 500 years.");
20      }
21 }
```

The Integer Data Types

Printed on the computer screen when application runs:

We have made a journey of 120 miles in only 100 minutes.

Value of
variable *miles*

```
miles = 120;
minutes = 100;
temperatureLondon = -15;
days = 182500;

System.out.println("We have made a journey of " + miles +
    " miles in only " + minutes + " minutes.");
System.out.println("The temperature in London is " +
    temperatureLondon + " degrees.");
System.out.println("There are " + days + " days in 500 years.");
}
```

The Integer Data Types

Printed on the computer screen when application runs:

We have made a journey of 120 miles in only 100 minutes.

Concatenation (+) operator

```
10      miles = 120;
11      minutes = 100;
12      temperatureLondon = -15;
13      days = 182500;
14
15      System.out.println("We have made a journey of " + miles +
16                          " miles in only " + minutes + " minutes.");
17      System.out.println("The temperature in London is " +
18                          temperatureLondon + " degrees.");
19      System.out.println("There are " + days + " days in 500 years.");
20      }
21 }
```

The Integer Data Types

Printed on the computer screen when application runs:

We have made a journey of 120 miles in only 100 minutes.

String literal

```
miles = 120;
minutes = 100;
temperatureLondon = -15;
days = 182500;

System.out.println("We have made a journey of " + miles +
    " miles in only " + minutes + " minutes.");
System.out.println("The temperature in London is " +
    temperatureLondon + " degrees.");
System.out.println("There are " + days + " days in 500 years.");
```

The Integer Data Types

Printed on the computer screen when application runs:

We have made a journey of 120 miles in only 100 minutes.

Value of variable
minutes

```
10 miles = 120;
11 minutes = 100;
12 temperatureLondon = -15;
13 days = 182500;
14
15 System.out.println("We have made a journey of " + miles +
16                    " miles in only " + minutes + " minutes.");
17 System.out.println("The temperature in London is " +
18                    temperatureLondon + " degrees.");
19 System.out.println("There are " + days + " days in 500 years.");
20
21 }
```

The Integer Data Types

Printed on the computer screen when application runs:

We have made a journey of 120 miles in only 100 minutes.

String literal

```
10 miles = 120;
11 minutes = 100;
12 temperatureLondon = -15;
13 days = 182500;
14
15 System.out.println("We have made a journey of " + miles +
16                     " miles in only " + minutes + " minutes.");
17 System.out.println("The temperature in London is " +
18                     temperatureLondon + " degrees.");
19 System.out.println("There are " + days + " days in 500 years.");
20
21 }
```


The Integer Data Types

Printed on the computer screen when application runs:

We have made a journey of 120 miles in only 100 minutes.

The temperature in London is -15 degrees.

There are 182500 days in 500 years.

```
10     miles = 120;
```

```
11     minutes = 100;
```

```
12     temperatureLondon = -15;
```

```
13     days = 182500;
```

```
15     System.out.println("We have made a journey of " + miles +  
16                          "miles in only " + minutes + " minutes.");
```

```
17     System.out.println("The temperature in London is " +  
18                          temperatureLondon + " degrees.");
```

```
19     System.out.println("There are " + days + " days in 500 years.");
```

```
20     }
```

```
21 }
```

The Floating-Point Data Types

The floating-point data types include *float* and *double*.

When you write a floating-point literal in your program code, Java assumes it to be of the *double* data type. You can force a floating-point literal to be treated as a *float* by suffixing it with the letter **F**. Example: 14.0**F** would be treated as a *float*.

```
float pay;  
pay = 1800.99;
```

This statement will give an error message (1800.99 seen as a *double*)

```
float pay;  
pay = 1800.99F;
```

Forced to be of type *float*

Scientific and E notation

Floating-point literals can be represented in scientific notation.

The number 3872.38 can be represented as 3.87238×10^3

Java uses E notation to represent values in scientific notation:

In E notation, the number 3.87238×10^3 would be 3.87238E3

The boolean Data Type

The ***boolean*** data type allows you to create variables that may hold one of two possible values: ***true*** or ***false***.

boolean variables are useful for evaluating conditions that are either ***true*** or ***false***.

The boolean Data Type

```
1    public class booleanData
2    {
3        public static void main(String[] args)
4        {
5            boolean value;
6
7            value = true;
8            System.out.println(value);
9            value = false;
10           System.out.println(value);
11        }
12    }
```

Printed on the computer screen when application runs:

true

false

The char Data Type

The ***char*** data type is used to store characters. This data type can hold only one character at a time.

Character literals are enclosed in ***single quotation marks***.

The char Data Type

```
1    public class Characters
2    {
3        public static void main(String[] args)
4        {
5            char letter;
6
7            letter = 'A';
8            System.out.println(letter);
9            letter = 'B';
10           System.out.println(letter);
11        }
12    }
```

Printed on the computer screen when application runs:

A

B

The char Data Type

Unicode

Characters are internally represented by numbers. Each character is assigned a unique number.

Java uses Unicode, which is a set of numbers that are used as codes for representing characters. Each Unicode number requires two bytes of memory, so ***char*** variables occupy ***two bytes***.

The char Data Type

```
1    public class Characters
2    {
3        public static void main(String[] args)
4        {
5            char letter;
6
7            letter = 65;
8            System.out.println(letter);
9            letter = 66;
10           System.out.println(letter);
11        }
12    }
```

Printed on the computer screen when application runs:

A

B

Arithmetic Operators

Java offers a multitude of operators for manipulating data. There are 3 types of operators: unary, binary, ternary. (Please read page 79 for information on these 3 types)

Arithmetic Operators

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

Arithmetic Operators

Addition (+)

The addition operator returns the sum of its two operands.

```
answer = 5 + 4;    // assigns 9 to variable answer
```

```
pay = salary + bonus; // assigns the value of salary + bonus to variable pay
```

```
number = number + 1; // assigns number + 1 to variable number
```

Arithmetic Operators

Subtraction (-)

The subtraction operator returns the value of its right operand subtracted from its left operand.

```
answer = 5 - 4;    // assigns 1 to variable answer
```

```
pay = salary - tax; // assigns the value of salary - tax to variable pay
```

```
number = number - 1; // assigns number - 1 to variable number
```

Arithmetic Operators

Multiplication (*)

The multiplication operator returns the product of its two operands.

```
answer = 5 * 4;    // assigns 20 to variable answer
```

```
pay = hours * rate // assigns the value of (hours * rate) to variable pay
```

```
students = students * 2; // assigns (students * 2) to variable students
```

Arithmetic Operators

Division (/)

The division operator returns the quotient of its left operand divided by its right operand.

```
answer = 20 / 4;    // assigns 5 to variable answer
```

```
average = marks / number; // assigns the value of (marks / number) to variable  
//average
```

```
half = number / 2; // assigns (number / 2) to variable half
```

Arithmetic Operators

Modulus (%)

The modulus operator returns the remainder of a division operation involving two integers.

```
leftOver = 22 / 4;    // assigns 2 to variable leftOver
```


Arithmetic Operators

Printed on the computer screen when application runs:

My gross salary for this month is R8750.75

```
7      double pension,  
8      double medical;  
9  
10     basicSalary = 19000.00;  
11     tax = basicSalary * 0.35;  
12     pension = 2399.50;  
13     medical = pension / 2;  
14     grossSalary = basicSalary - tax - pension - medical;  
15  
16     System.out.println("My gross salary for this month is R" +  
17         grossSalary);  
18     }  
19 }
```

Integer division

When both operands of a division statement are integers, the statement will result in ***integer division***. This means that the result of the division will be an integer as well. If there is a remainder, it will be discarded.

```
double parts;
```

```
parts = 22 / 4; // will assign the value of 5.0
```

In order for a division operation to return a floating-point value, one of the operands must be of a floating-point data type.

```
double parts;
```

```
parts = 22.0 / 4; // will assign the value of 5.5
```

Operator Precedence

Please read page 81 and 82

Grouping with parentheses

Please read page 83 (top)

Calculating Percentages and Discounts

Please read pages 83 to 85

$$50\% = \frac{50}{100} = 0.5$$

In Java we only use the 0.5 to represent 50% in calculations.

The Math Class

The Java API provides a class named `Math`, which contains numerous methods that are useful for performing complex mathematical operations.

The methods *`pow`*, *`sqrt`* and constant *`PI`* are part of the *`Math`* class.

The Math Class

The *Math.pow* Method

The *Math.pow* method raises a number to a power.

```
result = Math.pow(3.0, 2.0);
```

This method takes two **double** arguments. It raises the first argument to the power of the second argument, and returns the result as a **double**.

$$result = 3^2$$

$$result = 9.0$$

The Math Class

The *Math.sqrt* Method

The *Math.sqrt* method accepts a double value as its argument and returns the square root of the value.

```
result = Math.sqrt(4.0);
```

$$result = \sqrt{4}$$

$$result = 2.0$$

The Math Class

The *Math.PI* predefined constant

The *Math.PI* constant is a constant assigned with a value of 3.14159265358979323846, which is an approximation of the mathematical value pi.

```
double area, radius;
```

```
radius = 5.5;
```

```
area = Math.PI * radius * radius;
```

$$area = \pi \times 5.5 \times 5.5$$

$$area = 95.03317777$$

Combined Assignment Operators

`x = x + 1;`

Faster way:

`x += 1;`

On the right of the assignment operator, 1 is added to x. The result is then assigned to x, replacing the previous value. Effectively, this statement adds 1 to x.

`y = y - 1;`

Faster way:

`y -= 1;`

On the right of the assignment operator, 1 is subtracted from y. The result is then assigned to y, replacing the previous value. Effectively, this statement subtracts 1 from y.

Combined Assignment Operators

```
z = z*10;
```

Faster way:

```
z *= 10;
```

On the right of the assignment operator, 10 is multiplied by z. The result is then assigned to z, replacing the previous value. Effectively, this statement multiplies z with 10.

```
a = a / b;
```

Faster way:

```
a /= b;
```

On the right of the assignment operator, a is divided by b. The result is then assigned to a, replacing the previous value. Effectively, this statement divides a by b.

Combined Assignment Operators

```
x = x % 4;
```

Faster way:

```
x %= 4;
```

On the right of the assignment operator, the remainder of x divided by 4 is calculated. The result is then assigned to x , replacing the previous value. Effectively, this statement assigns the remainder of $x/4$ to x .

Conversion between Primitive Data Types

Before a value can be stored in a variable, the ***value's data type*** must be compatible with the ***variable's data type***. Java performs some conversions between data types automatically, but does not automatically perform any conversion that can result in the loss of data.

Conversion between Primitive Data Types

```
int x;  
double y = 2.8;  
x = y;
```

This statement is attempting to store a **double** value (2.8) in an **int** variable. This will give an error message. A **double** can store fractional numbers and can hold values much larger than an **int** can hold. If this were permitted, a loss of data would be likely.

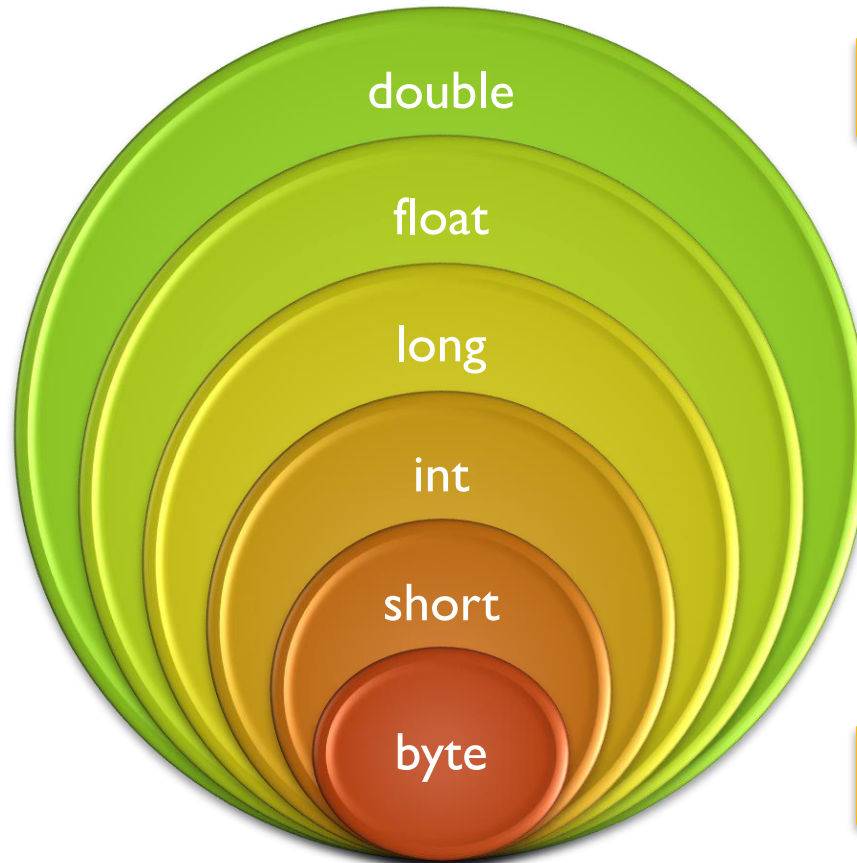
Conversion between Primitive Data Types

```
int x;  
short y = 2;  
x = y;
```

This statement is attempting to store a ***short*** value (2) in an ***int*** variable. **This will work with no problems.**

Conversion between Primitive Data Types

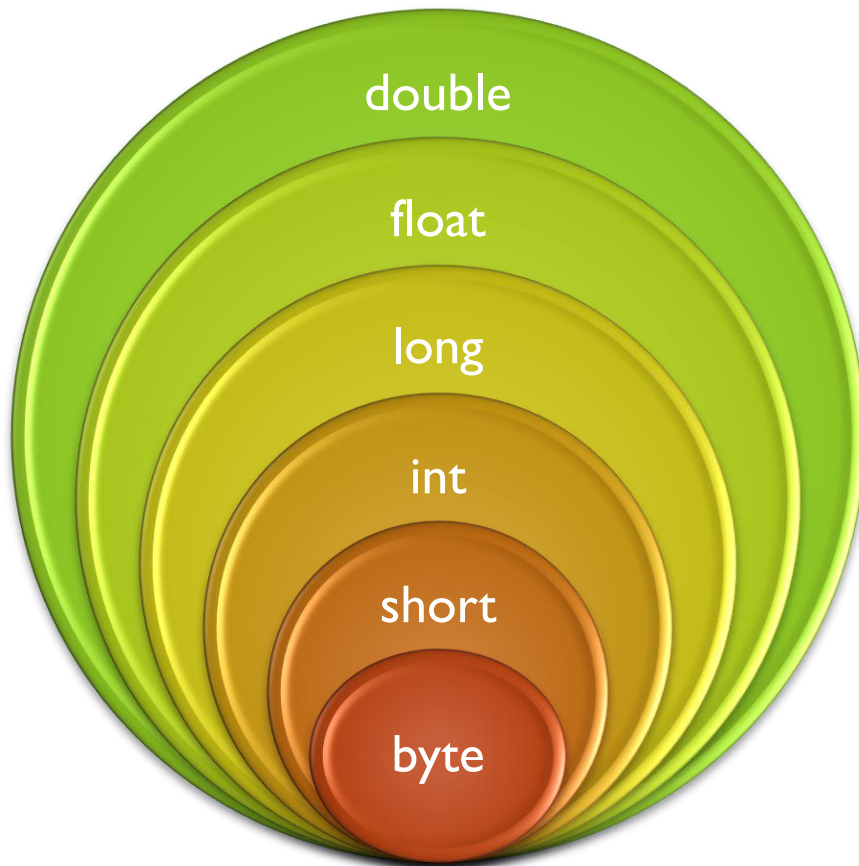
Primitive data type ranking



Highest rank

Lowest rank

Conversion between Primitive Data Types

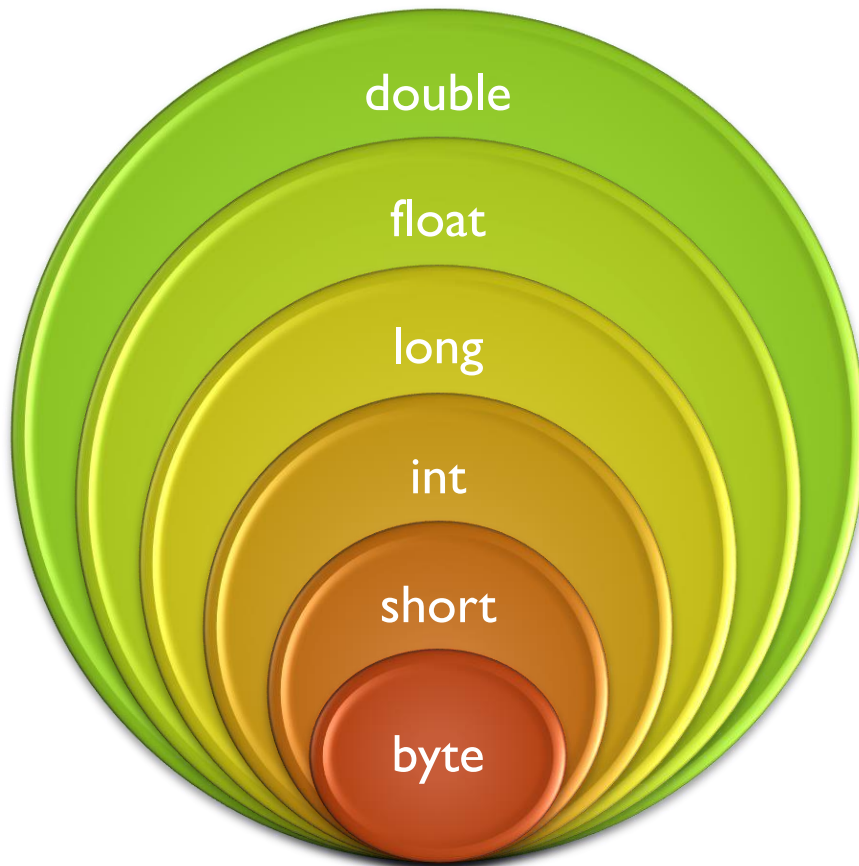


In assignment statements where values of **lower-ranked data types** are stored in variables of **higher-ranked data types**, Java automatically converts the lower-ranked value to the higher-ranked type.

```
double x;  
int y = 2;  
x = y;
```

We call this a widening conversion

Conversion between Primitive Data Types



In assignment statements where values of **higher-ranked data types** are stored in variables of **lower-ranked data types**, Java does not automatically perform the conversion because of possible data loss.

```
int x;  
double y = 2.0;  
x = y;
```

Error!

We call this a narrowing conversion

Conversion between Primitive Data Types

(**Cast operators**)

The cast operator lets you manually convert a value, even if it means that a ***narrowing conversion*** will take place.

```
int x;  
double y = 2.5;  
x = y;
```

Error!

Cast
operator

```
int x;  
double y = 2.5;  
x = (int) y;
```

No problem!

Conversion between Primitive Data Types

(Cast operators)

Java compiles the code with the cast operator with no problems. In this case variable *y* has a value of 2.5 (floating-point value) which must be converted to an integer.

Cast
operator

```
int x;  
double y = 2.5;  
x = (int) y;
```

No problem!

The value that is returned and stored in variable *x* would be truncated, which means the fractional part of the number is lost to accommodate the *integer* data type.

Thus: $x = 2$

The value of variable *y* is not changed at all: $y = 2.5$

Mixed Integer Operations

One of the nuances of the Java language is the way it handles arithmetic operations on **int**, **byte** and **short**.

When values of the **byte** or **short** data types are used in arithmetic expressions, they are temporarily converted to **int** values.

```
short x = 10, y = 20, z;  
z = x + y;
```

Error!

How can we rectify this error?

Mixed Integer Operations

```
short x = 10, y = 20, z;  
z = x + y;
```

Error!

The error results from the fact that **z** is a short. The expression **x + y** results in an *int* value.

This can be corrected if **z** is declared as an *int*, or if a cast operator is used.

```
short x = 10, y = 20;  
int z;  
z = x + y;
```

No problem!

```
short x = 10, y = 20, z;  
z = (short) (x + y);
```

No problem!

Other Mixed Mathematical Expressions

Please read pages 91 and 92.

Creating Named Constants with *final*

The `final` key word can be used in a variable declaration to make the variable a named constant. Named constants are initialized with a value, and that value cannot change during the execution of the program.

```
amount = balance * 0.072;
```

The 1st problem that arises is that it is not clear to anyone but the original programmer as to what the 0.072 is.

The 2nd problem occurs if this number is used in other calculations throughout the program and must be changed periodically.

Creating Named Constants with *final*

We can change the code to use the *final* key word to create a constant value.

```
amount = balance * 0.072;
```

Old code

```
final double INTEREST_RATE = 0.072;
```

```
amount = balance * INTEREST_RATE;
```

New code

Now anyone who reads the code will understand it. When we want to change the interest rate, we change it only once at the declaration.

The *String* Class

The ***String*** class allows you to create objects for holding strings. It also has various methods that allow you to work with strings.

A string is a ***sequence of characters***. It can be used to present any type of data that contains ***text***. String literals are enclosed in ***double quotation marks***.

Java does not have a primitive data type for storing strings in memory. The Java API provides a ***class*** for handling strings. You use this class to create ***objects*** that are capable of storing strings and performing operations on them.

The *String* Class

Earlier we introduced you to **objects** as software entities that can contain **attributes** and **methods**. An object's attributes are data values that are stored in the object. An object's methods are procedures that perform operations on the object's attributes.

Think of a **class** as a blueprint that **objects** may be created from. So a class is not an object, but a **description of an object**.

When the program is running, it can use the class to create, in memory, as many objects as needed.

The *String* Class

Creating a String Object

Any time you write a string literal in your program, Java will create a ***String object*** in ***memory*** to hold it. You can create a String object in memory and store its ***address*** in a String variable with a simple assignment statement:

```
String name = "Jet Li";
```

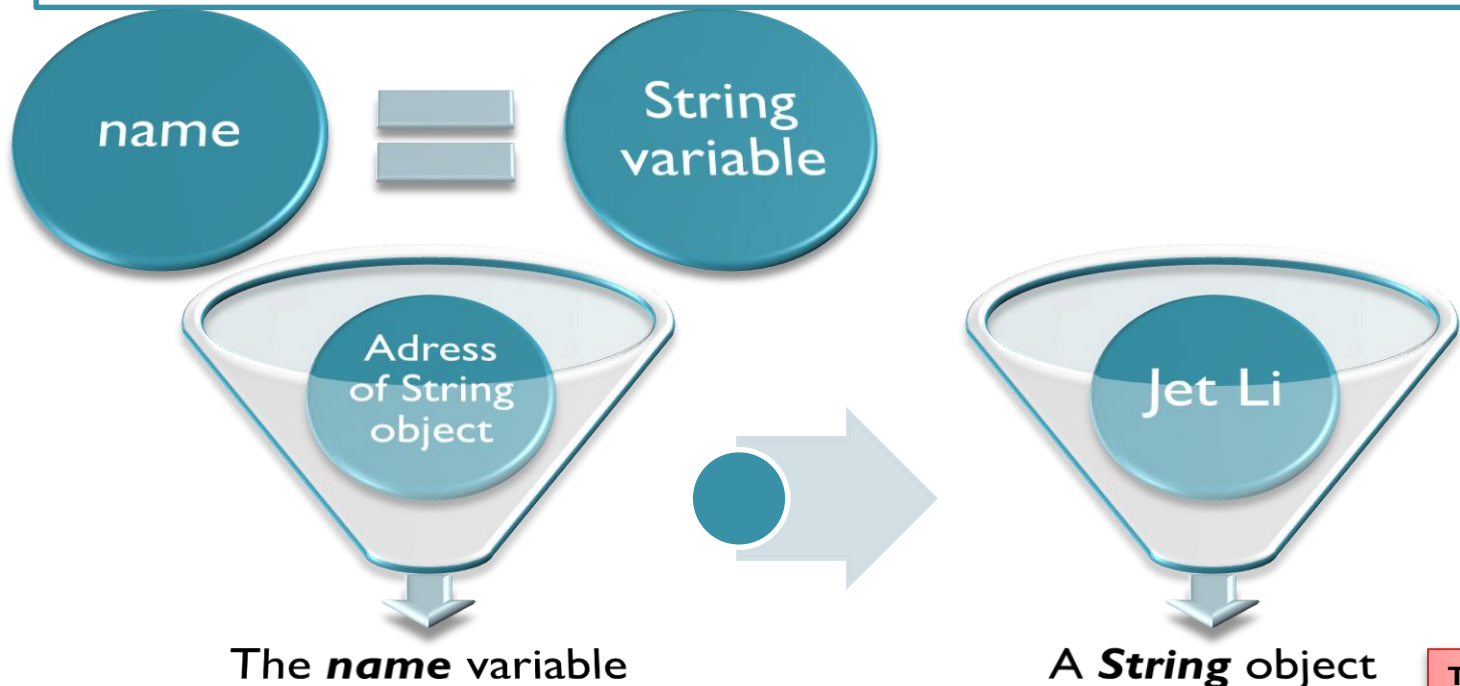
This statement declares ***name*** as a ***String*** variable, creates a ***String object*** with the value "***Jet Li***" stored in it, and assigns the object's memory address to the ***name*** variable.

The *String* Class

Creating a String Object

```
String name = "Jet Li";
```

This statement declares ***name*** as a ***String*** variable, creates a ***String object*** with the value "***Jet Li***" stored in it, and assigns the object's memory address to the ***name*** variable.



The *String* Class

```
1 public class StringExample
2 {
3     public static void main(String[] args)
4     {
5         String greeting = "Good morning ";
6         String name = "Jet Li!";
7
8         System.out.println(greeting + name);
9     }
10 }
```

Printed on the computer screen when application runs:

Good morning Jet Li!

The *String* Class

String Methods

Because the *String* type is a **class** instead of a **primitive data type**, it provides numerous **methods** for working with strings.

The *String* Class

charAt() Method

This method returns the ***character*** at the specified ***position***.

```
char letter;  
String name = "Arnold";  
letter = name.charAt(2);
```

0	1	2	3	4	5
<i>A</i>	<i>r</i>	<i>n</i>	<i>o</i>	<i>l</i>	<i>d</i>

After this code executes, the variable ***letter*** will hold the character ***'n'***.

The *String* Class

length() Method

This method returns the ***number of characters*** in a string.

```
int stringSize;  
String name = "Arnold";  
stringSize = name.length();
```

0	1	2	3	4	5
<i>A</i>	<i>r</i>	<i>n</i>	<i>o</i>	<i>l</i>	<i>d</i>

After this code executes, the variable ***stringSize*** will hold the value **6**.

The *String* Class

toLowerCase() Method

This method returns a new string that is the **lowercase** equivalent of the string contained in the calling object.

```
String bigName = "ARNOLD";  
String littleName = bigName.toLowerCase();
```

After this code executes, the variable **littleName** will hold the string "**arnold**".

The *String* Class

toUpperCase() Method

This method returns a new string that is the ***uppercase*** equivalent of the string contained in the calling object.

```
String littleName = "arnold";  
String bigName = littleName.toUpperCase();
```

After this code executes, the variable ***bigName*** will hold the string ***"ARNOLD"***.

The *Scope* of a variable

A variable's **scope** is the part of the program that has access to the variable. A variable is only visible to statements inside the variable's **scope**.

Variables that are declared inside a method (like the main method) are called **local variables**.

A local variable's scope begins at the **variable's declaration** and ends at the **end of the method** in which the variable is declared.

A local variable cannot be accessed by code that is **outside the method**, or inside the method but **before the variable's declaration**.

The *Scope* of a variable

```
1 public class VariableScope
2 {
3     public static void main(String[] args)
4     {
5         System.out.println(value);
6
7         double value = 5.7;
8     }
9 }
```

ERROR! This program attempts to send the contents of variable *value* to *println* before the variable is declared.

The *Scope* of a variable

```
1 public class VariableScope
2 {
3     public static void main(String[] args)
4     {
5         double value = 5.7;
6
7         System.out.println(value);
8     }
9 }
```

No problem!

The *Scope* of a variable

Another rule you must remember about local variables is that you cannot have ***two local variables*** with the ***same name*** in the ***same scope***.

The *Scope* of a variable

```
1 public class VariableScope
2 {
3     public static void main(String[] args)
4     {
5         int number = 5;
6         System.out.println(number);
7
8         int number = 7;
9         System.out.println(number);
10    }
11 }
```

ERROR! The variable *number* is declared twice within one method.

The *Scope* of a variable

```
1 public class VariableScope
2 {
3     public static void main(String[] args)
4     {
5         int number = 5;
6         System.out.println(number);
7
8         number = 7;
9         System.out.println(number);
10    }
11 }
```

No problem!

Comments

Comments are *notes of explanation* that document lines or sections of a program. Comments are part of the program, but the **compiler ignores them**. They are intended for people who may be reading the source code.

Comments

Three ways to comment in Java

Single-Line comments (//)

Multi-line comments (/*..... */)

Documentation comments (/**..... */)

Comments

Single-Line comment

You simply place two forward slashes (`//`) where you want the comment to begin. The compiler ignores everything from that point to the end of the line.

```
1 // This is a single-line comment
2
3 public class ...
4 {
5     ....
6 }
```

Comments

Multi-Line comment

Multi-Line comments start with a forward slash followed by an asterisk (`/*`) and end with an asterisk followed by a forward slash (`*/`). Everything between these markers is ignored.

```
1  /*
2      This is a
3      Multi-Line comment
4  */
5  public class ...
6  {
7      ...
8  }
```

Comments

Documentation Comments

Documentation comments starts with `/**` and ends with `*/`. Normally you write a documentation comment just before ***class*** and ***method headers***, giving a brief description of the ***class*** or ***method***.

These comments can be read and processed by a program named ***javadoc***, which comes with the Sun JDK. The purpose of the ***javadoc*** program is to read Java source code files and generate attractively formatted HTML files that document the source code.

Please read pages 103, 104, 105

Comments

Documentation Comments

```
1  /**
2      This class creates a program that calculates
3      company payroll
4  */
5  public class Comment
6  {
7      /**
8          The main method is the program's starting point
9      */
10     public static void main(String[] args)
11     {
12         ...
13     }
14 }
```

Programming Style

Please read pages 106 and 107

Reading Keyboard Input

The Scanner class

Objects of the **Scanner** class can be used to read input from the keyboard.

The Java API has an object **System.in** which refers to the standard input device (normally the keyboard). The **System.in** object reads input only as byte values which isn't very useful. To work around this, we use the **System.in** object in conjunction with an object of the **Scanner** class.

The **Scanner** class is designed to read input from a source (**System.in**) and provides methods that you can use to retrieve the input formatted as **primitive values** or **strings**.

Reading Keyboard Input

The Scanner class

First, you create a **Scanner** object and connect it to the **System.in** object:

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

Declares a variable named **keyboard**. The data type of the variable is **Scanner**.

Because **Scanner** is a **class**, the **keyboard** variable is a **class type variable**.

Remember that a class type variable holds the **memory address** of an object. Therefore, the **keyboard** variable will be used to hold the address of a **Scanner object**

Reading Keyboard Input

The Scanner class

First, you create a **Scanner** object and connect it to the **System.in** object:

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

The assignment operator will assign something to the **keyboard** variable

Reading Keyboard Input

The Scanner class

First, you create a **Scanner** object and connect it to the **System.in** object:

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

new is a Java key word. It is used to create an object in memory. The type of object that will be created is listed after the **new** key word.

Reading Keyboard Input

The Scanner class

First, you create a **Scanner** object and connect it to the **System.in** object:

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

This specifies that a **Scanner** object should be created, and it should be connected to the **System.in** object as source for input.

The memory address of this object is assigned to the variable **keyboard**.

After this statement executes, the **keyboard** variable will reference the **Scanner object** in memory.

Reading Keyboard Input

Scanner class methods

The Scanner class has methods for reading **strings**, **bytes**, **integers**, **long** integers, **short** integers, **floats** and **doubles**.

Reading Keyboard Input

Scanner class methods : `nextByte`

Returns input as a **byte**

```
1 byte x;  
2 Scanner keyboard = new Scanner(System.in);  
3 System.out.println("Enter a byte value: ");  
4 x = keyboard.nextByte();
```

The **nextByte** method formats the input that was entered at the keyboard as a **byte**, and then returns it.

Reading Keyboard Input

Scanner class methods : `nextDouble`

Returns input as a ***double***

```
1 double number;  
2 Scanner keyboard = new Scanner(System.in);  
3 System.out.println("Enter a double value: ");  
4 number = keyboard.nextDouble();
```

The ***nextDouble*** method formats the input that was entered at the keyboard as a ***double***, and then returns it.

Reading Keyboard Input

Scanner class methods : `nextFloat`

Returns input as a ***float***

```
1 float number;  
2 Scanner keyboard = new Scanner(System.in);  
3 System.out.println("Enter a float value: ");  
4 number = keyboard.nextFloat();
```

The ***nextFloat*** method formats the input that was entered at the keyboard as a ***float***, and then returns it.

Reading Keyboard Input

Scanner class methods : `nextInt`

Returns input as an *int*

```
1 int number;  
2 Scanner keyboard = new Scanner(System.in);  
3 System.out.println("Enter a integer value: ");  
4 number = keyboard.nextInt();
```

The ***nextInt*** method formats the input that was entered at the keyboard as an *int*, and then returns it.

Reading Keyboard Input

Scanner class methods : `nextLine`

Returns input as a ***String***

```
1 String name;  
2 Scanner keyboard = new Scanner(System.in);  
3 System.out.println("Enter your name: ");  
4 name = keyboard.nextLine();
```

The ***nextLine*** method formats the input that was entered at the keyboard as a ***String***, and then returns it.

Reading Keyboard Input

Scanner class methods : `nextLong`

Returns input as a ***long***

```
1 long number;  
2 Scanner keyboard = new Scanner(System.in);  
3 System.out.println("Enter a long value: ");  
4 number = keyboard.nextLong();
```

The ***nextLong*** method formats the input that was entered at the keyboard as a ***long***, and then returns it.

Reading Keyboard Input

Scanner class methods : `nextShort`

Returns input as a ***short***

```
1 short number;  
2 Scanner keyboard = new Scanner(System.in);  
3 System.out.println("Enter a short value: ");  
4 number = keyboard.nextShort();
```

The ***nextShort*** method formats the input that was entered at the keyboard as a ***short***, and then returns it.

Reading Keyboard Input

Scanner class : *import* statement

The **Scanner** class is not automatically available to your Java programs. Any program that uses the **Scanner** class should have the following statement near the beginning of the file, before any class definition:

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

This statement tells the Java compiler where in the Java library to find the **Scanner** class, and makes it available to your program.

Reading Keyboard Input

```
1 import java.util.Scanner; // Needed for the Scanner class
2
3 public class InputProblem
4 {
5     public static void main(String[] args)
6     {
7         String name;
8         int age;
9         double income;
10
11         Scanner keyboard = new Scanner(System.in);
12
13         System.out.print("What is your age? ");
14         age = keyboard.nextInt();
15
16         System.out.print("What is your annual income? ");
17         income = keyboard.nextDouble();
18
19         System.out.print("What is your name? ");
20         name = keyboard.nextLine();
21
22         System.out.println("Hello " + name + ". Your age is " +
23             age + " and your income is R" + income);
24     }
25 }
```

Variable declarations

Create Scanner object to read input

Get user's age

Get user's income

Get user's name

Display information back to the user

Reading Keyboard Input

Printed on the computer screen when application runs:

What is your age? **25 [enter]**

What is your annual income? **80000 [enter]**

What is your name? Hello .Your age is 25 and your income is R80000.00

The program does not give the user time to enter his/her name

Problem!!

```
11 Scanner keyboard = new Scanner(System.in);
12
13 System.out.print("What is your age? ");
14 keyboard.nextInt();
15
16 System.out.print("What is your annual income? ");
17 income = keyboard.nextDouble();
18
19 System.out.print("What is your name? ");
20 name = keyboard.nextLine();
21
22 System.out.println("Hello " + name + ". Your age is " +
23 age + " and your income is R" + income);
24 }
25 }
```


Reading Keyboard Input

When the user types keystrokes at the keyboard, those keystrokes are stored in an area of memory called the **keyboard buffer**.

Pressing the “**Enter**” key causes a **new-line** character to be stored in the keyboard buffer.

Reading Keyboard Input

keyboard buffer

25

/n

nextInt() method

age =

Stops when it sees the new-line character

keyboard buffer, and then stopped when it encountered the **newline** character. The newline character was not read and remained in the keyboard buffer.

```
13      System.out.print("What is your age? ");
14      age = keyboard.nextInt();
15
16      System.out.print("What is your annual income? ");
17      income = keyboard.nextDouble();
18
19      System.out.print("What is your name? ");
20      name = keyboard.nextLine();
```

Reading Keyboard Input

keyboard buffer

nextDouble() method

Skips newline character

/n

income =

80000

Stops when it sees the newline character

/n

newline characters it encounters. It skips the newline character, reads the value 80000.00 and stops reading when it encounters the newline character which is then left in the keyboard buffer.

```
16 System.out.print("What is your annual income? ");
17 income = keyboard.nextDouble();
18
19 System.out.print("What is your name? ");
20 name = keyboard.nextLine();
```

Reading Keyboard Input

keyboard buffer

Next the user was asked to enter his/her ***name***

In line 20 the **`nextLine()`** method is called.

The **`nextLine()`** method, however, is not designed to skip over an initial newline character. If a newline character is the first character that **`nextLine()`** method encounters, then nothing will be read. It will immediately terminate and the user will not be given a chance to enter his or her name.



/n

```
19  
20
```

```
System.out.print("What is your name? ");  
name = keyboard.nextLine();
```

Reading Keyboard Input

```
1 import java.util.Scanner; // Needed for the Scanner class
2
3 public class InputProblem
4 {
5     public static void main(String[] args)
6     {
7         Scanner keyboard = new Scanner(System.in);
8
9         System.out.print("What is your age? ");
10        age = keyboard.nextInt();
11
12        System.out.print("What is your annual income? ");
13        income = keyboard.nextDouble();
14        keyboard.nextLine();
15        System.out.print("What is your name? ");
16        name = keyboard.nextLine();
17
18        System.out.println("Hello " + name + ". Your age is " +
19            age + " and your income is R" + income);
20    }
21 }
22
23
24
25 }
```

The purpose of this call is to consume, or remove, the newline character that remains in the keyboard buffer. We do not need to keep the method's return value so we do not assign the method's return value to any variable

Dialog Boxes

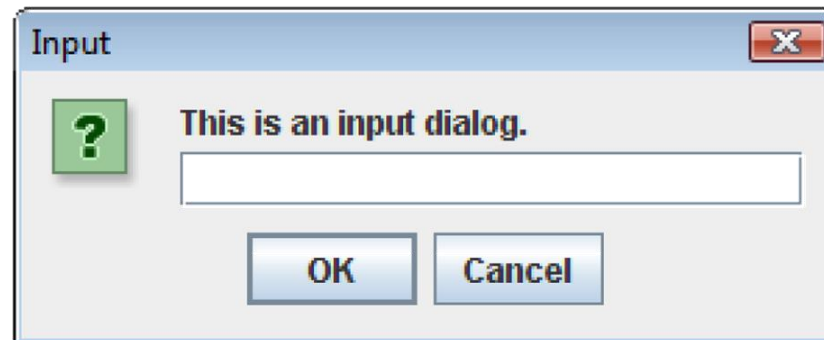
The JOptionPane Class

The `JOptionPane` class provides methods to display each type of **dialog box**.

Message dialog



Input dialog



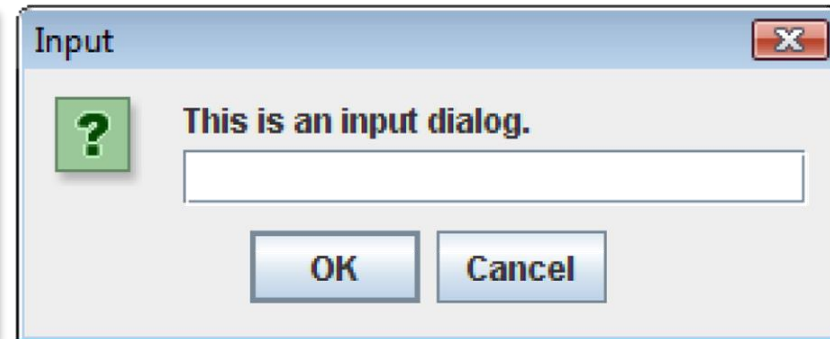
Dialog Boxes

The JOptionPane Class

A dialog box that displays a message and an OK button



A dialog box that prompts the user for input and provides a text field where input is typed. An OK and Cancel button are displayed.



Dialog Boxes

The `JOptionPane` Class

The ***JOptionPane*** class is not automatically available to your Java programs. The class must be imported:

```
import javax.swing.JOptionPane;
```

This statement tells the compiler where to find the ***JOptionPane*** class and makes it available to your program.

Dialog Boxes

Displaying Message Dialog Boxes

The *showMessageDialog* method is used to display a message dialog.

```
JOptionPane.showMessageDialog(null, "Today is a great day");
```

This argument is only important in programs that displays **other graphical windows**. You will use **null** as first argument. This causes the dialog box to be displayed in the **center of the screen**.

Dialog Boxes

Displaying Message Dialog Boxes

The ***showMessageDialog*** method is used to display a message dialog.

```
JOptionPane.showMessageDialog(null, "Today is a great day");
```

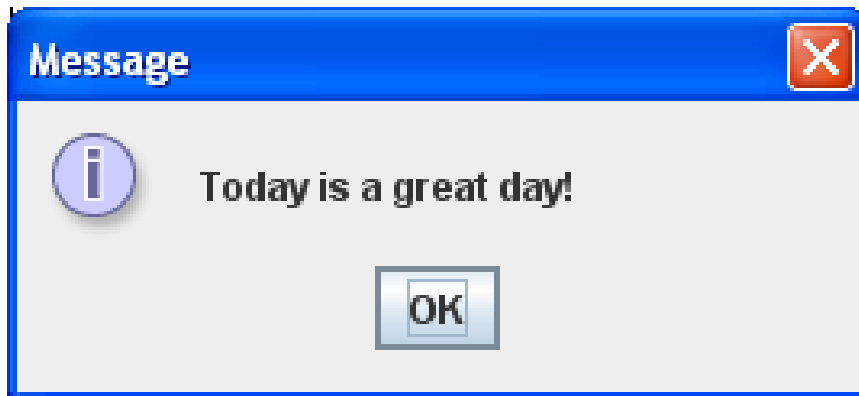
This argument is the ***message*** we wish to display in the dialog box.

Dialog Boxes

Displaying Message Dialog Boxes

The *showMessageDialog* method is used to display a message dialog.

```
JOptionPane.showMessageDialog(null, "Today is a great day");
```



When the user clicks on the **OK button**, the dialog box will **close**.

Dialog Boxes

Displaying Input Dialog Boxes

The ***showInputDialog*** method is used to display an input dialog.

String name;

```
name = JOptionPane.showInputDialog("Enter your name:");
```

This argument is the ***message*** we wish to display in the dialog box.

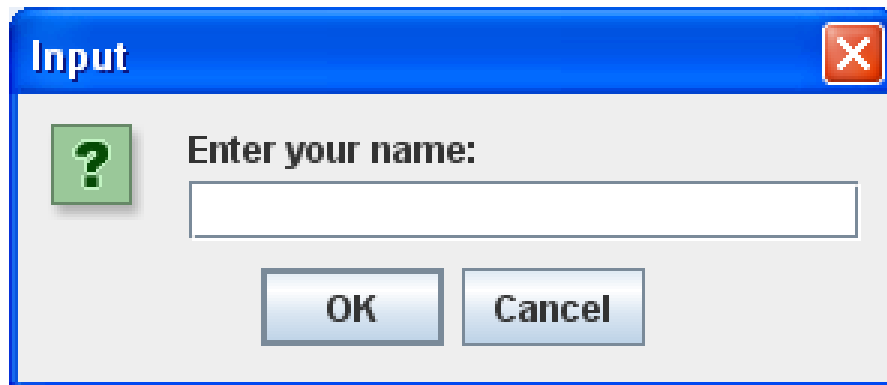
Dialog Boxes

Displaying Input Dialog Boxes

The *showInputDialog* method is used to display an input dialog.

String name;

```
name = JOptionPane.showInputDialog("Enter your name:");
```



When the user clicks on the **OK button**, variable *name* will reference the string value entered by the user into the **text field**.

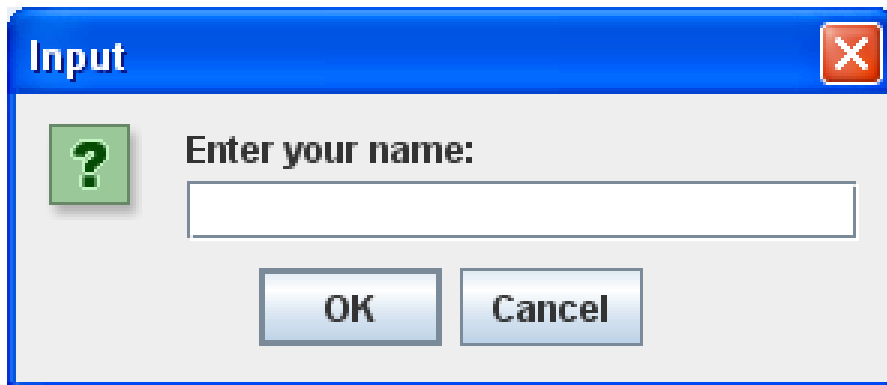
Dialog Boxes

Displaying Input Dialog Boxes

The ***showInputDialog*** method is used to display an input dialog.

String name;

```
name = JOptionPane.showInputDialog("Enter your name:");
```



If the user clicks the ***Cancel*** button, variable name will reference the special value ***null***.

Dialog Boxes

```
1 import javax.swing.JOptionPane;
```

import the *JOptionPane* for use in the program

```
2  
3 public class NamesDialog
```

```
4 {
```

```
5     public static void main(String[] args)
```

```
6     {
```

```
7         String firstName, middleName, lastName;
```

String declarations

```
8  
9         firstName = JOptionPane.showInputDialog("What is your first name?");
```

```
10  
11        middleName = JOptionPane.showInputDialog("What is your middle " +  
12                                                    "name");
```

```
13  
14        lastName = JOptionPane.showInputDialog("What is your last name?");
```

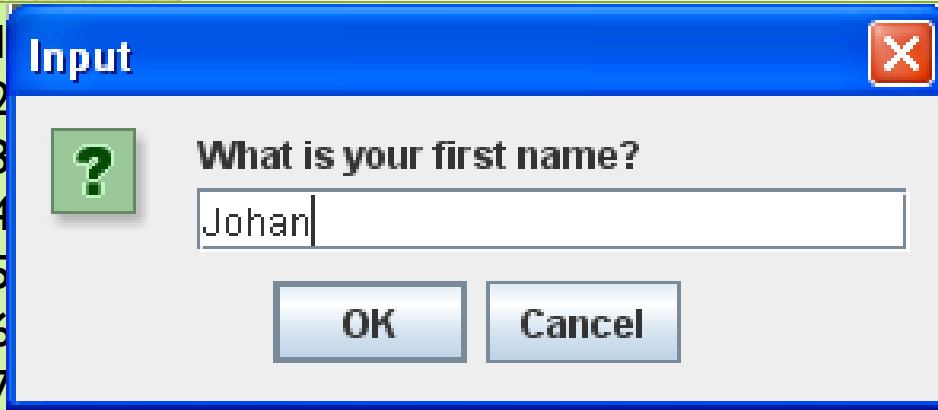
```
15  
16        JOptionPane.showMessageDialog(null, "Hello " + firstName + " " +  
17                                                    middleName + " " + lastName);
```

```
18        System.exit(0);
```

```
19    }
```

```
20 }
```

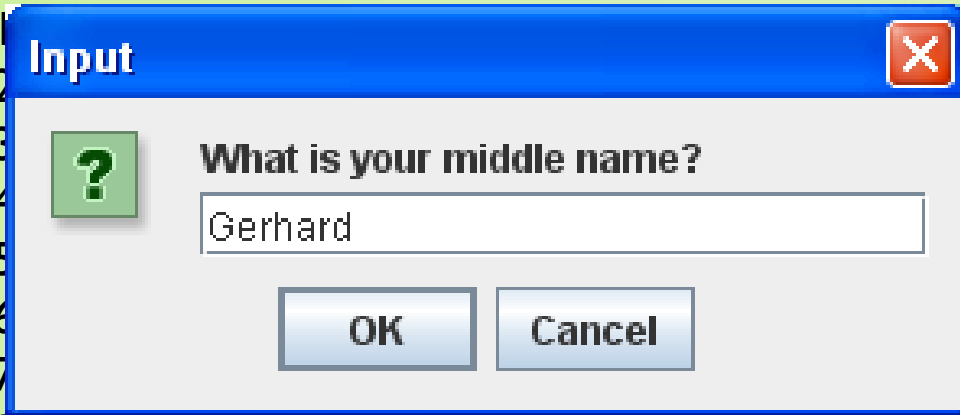
Dialog Boxes



Opens the *input dialog box* and asks the question: “**What is your first name?**” The user then enters his/her *first name* and clicks on **OK**.

```
1  firstName = JOptionPane.showInputDialog("What is your first name?");
2
3
4
5
6
7
8
9
10 middleName = JOptionPane.showInputDialog("What is your middle " +
11     "name");
12
13 lastName = JOptionPane.showInputDialog("What is your last name?");
14
15 JOptionPane.showMessageDialog(null, "Hello " + firstName + " " +
16     middleName + " " + lastName);
17
18 System.exit(0);
19 }
```


Dialog Boxes



Opens the *input dialog box* and asks the question: “**What is your middle name?**” The user then enters his/her **middle name** and clicks on **OK**.

```
8  
9     firstName = JOptionPane.showInputDialog("What is your first name.");
```

```
10  
11     middleName = JOptionPane.showInputDialog("What is your middle " +  
12                                     "name");
```

```
13     lastName = JOptionPane.showInputDialog("What is your last name?");
```

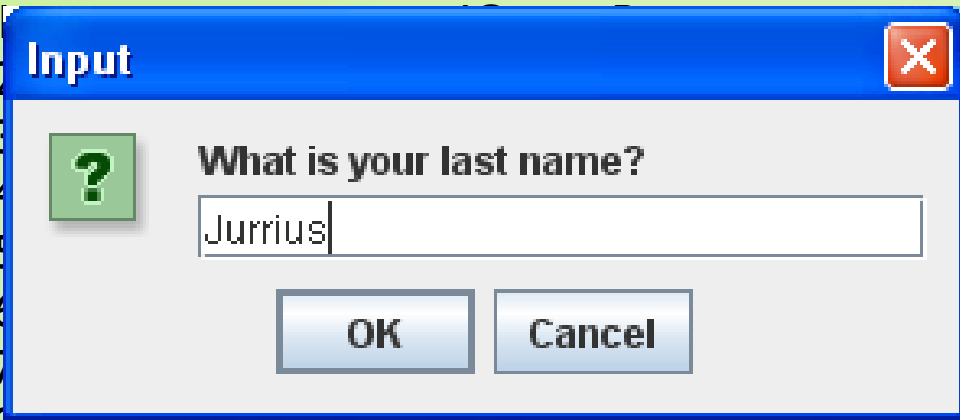
```
14  
15     JOptionPane.showMessageDialog(null, "Hello " + firstName + " " +  
16                                     middleName + " " + lastName);
```

```
17     System.exit(0);
```

```
18 }
```

```
19 }
```

Dialog Boxes



Opens the *input dialog box* and asks the question: “**What is your last name?**” The user then enters his/her *last name* and clicks on **OK**.

```
9      firstName = JOptionPane.showInputDialog("What is your first name?");
10
11     middleName = JOptionPane.showInputDialog("What is your middle " +
12                                           "name");
13     lastName = JOptionPane.showInputDialog("What is your last name?");
14
15     JOptionPane.showMessageDialog(null, "Hello " + firstName + " " +
16                                   middleName + " " + lastName);
17
18     System.exit(0);
19 }
```

Dialog Boxes



Opens the *message dialog box* and displays the *first name, middle name and last name* the user typed in the *input dialog boxes*. The user clicks on the **OK button** and the program **exits**.

```
9     firstName = JOptionPane.showInputDialog("What is your first name?");
```

```
10    middleName = JOptionPane.showInputDialog("What is your middle name?");
```

```
13    lastName = JOptionPane.showInputDialog("What is your last name?");
```

```
15    JOptionPane.showMessageDialog(null, "Hello " + firstName + " " +  
16        middleName + " " + lastName);
```

```
17    System.exit(0);
```

```
18 }
```

```
19 }
```

Dialog Boxes

```
1 import javax.swing.JOptionPane;
2
3 public class NamesDialog
4 {
5     public static void main(String[] args)
6     {
7         String firstName, middleName, lastName;
8
9         firstName = JOptionPane.showInputDialog("What is your first name?");
10
11        middleName = JOptionPane.showInputDialog("What is your middle " +
12                                                "name");
13
14        lastName = JOptionPane.showInputDialog("What is your last name?");
15
16        JOptionPane.showMessageDialog(null, "First name: " +
17        firstName + " Middle name: " + middleName + " Last name: " +
18        lastName);
19        System.exit(0);
20    }
21 }
```

This statement causes the program to **end**, and is **required** if you use the ***JOptionPane class*** to display dialog boxes.

Converting String Input to Numbers

The *JOptionPane class* does not have different methods for reading values of **different data types** as input. The *showInputDialog* method always returns the user's input as a **String**, even if the user enters **numeric data**.

Converting String Input to Numbers

The *Byte.parseByte* method

This method converts a ***string*** to a ***byte***.

```
byte number;
```

```
String input;
```

```
input = JOptionPane.showInputDialog("Enter a number:");
```

```
number = Byte.parseByte(input);
```

The ***string*** value in variable ***input*** is ***converted*** to a ***byte*** and then stored in variable ***number***.

Converting String Input to Numbers

The *Double.parseDouble* method

This method converts a *string* to a *double*.

```
double number;
```

```
String input;
```

```
input = JOptionPane.showInputDialog("Enter a price:");
```

```
number = Double.parseDouble(input);
```

The *string* value in variable *input* is *converted* to a *double* and then stored in variable *number*.

Converting String Input to Numbers

The *Float.parseFloat* method

This method converts a *string* to a *float*.

```
float number;
```

```
String input;
```

```
input = JOptionPane.showInputDialog("Enter a price:");
```

```
number = Float.parseFloat(input);
```

The *string* value in variable *input* is *converted* to a *float* and then stored in variable *number*.

Converting String Input to Numbers

The *Integer.parseInt* method

This method converts a **string** to an **int**.

```
int number;
```

```
String input;
```

```
input = JOptionPane.showInputDialog("Enter a number:");
```

```
number = Integer.parseInt(input);
```

The **string** value in variable **input** is **converted** to an **int** and then stored in variable **number**.

Converting String Input to Numbers

The *Long.parseLong* method

This method converts a ***string*** to a ***long***.

```
int number;
```

```
String input;
```

```
input = JOptionPane.showInputDialog("Enter a number:");
```

```
number = Long.parseLong(input);
```

The ***string*** value in variable ***input*** is ***converted*** to a ***long*** and then stored in variable ***number***.

Converting String Input to Numbers

The *Short.parseShort* method

This method converts a **string** to a **short**.

```
int number;
```

```
String input;
```

```
input = JOptionPane.showInputDialog("Enter a number:");
```

```
number = Short.parseShort(input);
```

The **string** value in variable **input** is **converted** to a **short** and then stored in variable **number**.

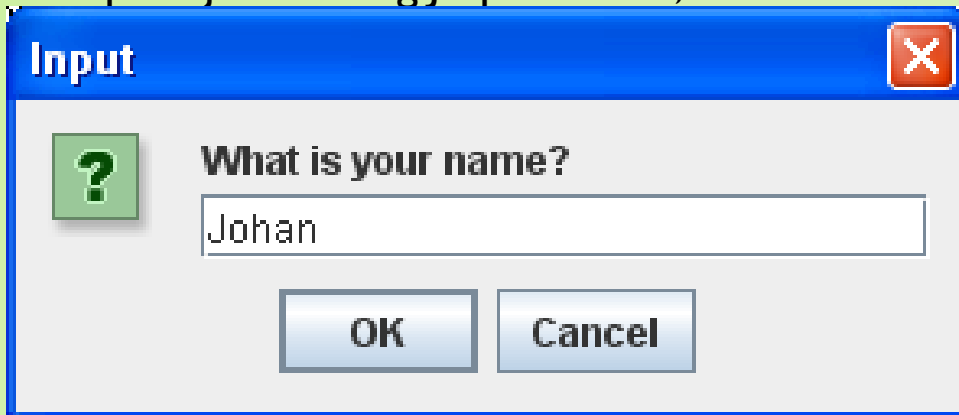
```
1 import javax.swing.JOptionPane;
2
3 public class DialogExample
4 {
5     public static void main(String[] args)
6     {
7         String input, name;
8         int hours;
9         double payRate, grossPay;
10
11         name = JOptionPane.showInputDialog("What is your name?");
12
13         input = JOptionPane.showInputDialog("How many hours did you work?");
14         hours = Integer.parseInt(input);
15
16         input = JOptionPane.showInputDialog("What is your hourly pay rate?");
17         payRate = Double.parseDouble(input);
18
19         grossPay = hours * payRate;
20
21         JOptionPane.showMessageDialog(null, "Hello " + name +
22                                     "! Your gross pay is: R" + grossPay);
23         System.exit(0);
24     }
25 }
```

```
1 import javax.swing.JOptionPane;
2
3 public class DialogExample
4 {
5     public static void main(String[] args)
6     {
7         String input, name;
8         int hours;
9         double payRate, grossPay;
10
11         name = JOptionPane.showInputDialog("What is your name?");
12
13         input = JOptionPane.showInputDialog("How many hours did you work?");
14         hours = Integer.parseInt(input);
15
16         input = JOptionPane.showInputDialog("What is your hourly pay rate?");
17         payRate = Double.parseDouble(input);
18
19         grossPay = hours * payRate;
20
21         JOptionPane.showMessageDialog(null, "Hello " + name +
22                                     "! Your gross pay is: R" + grossPay);
23         System.exit(0);
24     }
25 }
```

import **JOptionPane** to be used in program

variable declarations

```
1 import javax.swing.JOptionPane;
```



Input dialog box.
Input saved as normal
string value.

```
9     double payRate, grossPay;
```

```
11    name = JOptionPane.showInputDialog("What is your name?");
```

```
13    input = JOptionPane.showInputDialog("How many hours did you work?");  
14    hours = Integer.parseInt(input);
```

```
16    input = JOptionPane.showInputDialog("What is your hourly pay rate?");  
17    payRate = Double.parseDouble(input);
```

```
19    grossPay = hours * payRate;
```

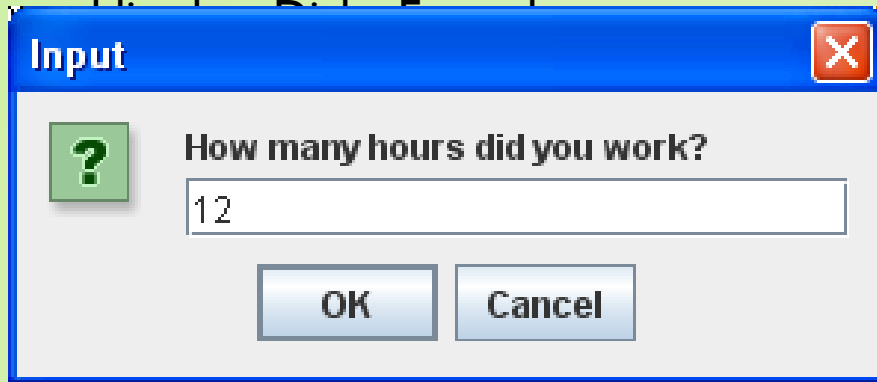
```
21    JOptionPane.showMessageDialog(null, "Hello " + name +  
22                                "! Your gross pay is: R" + grossPay);
```

```
23    System.exit(0);
```

```
24 }
```

```
25 }
```

```
1 import javax.swing.JOptionPane;
```



Input dialog box.
Input saved as **string**
and **converted to int**.

```
10  
11     name = JOptionPane.showInputDialog("What is your name?");
```

```
12  
13     input = JOptionPane.showInputDialog("How many hours did you work?");  
14     hours = Integer.parseInt(input);
```

```
15  
16     input = JOptionPane.showInputDialog("What is your hourly pay rate?");  
17     payRate = Double.parseDouble(input);
```

```
18  
19     grossPay = hours * payRate;
```

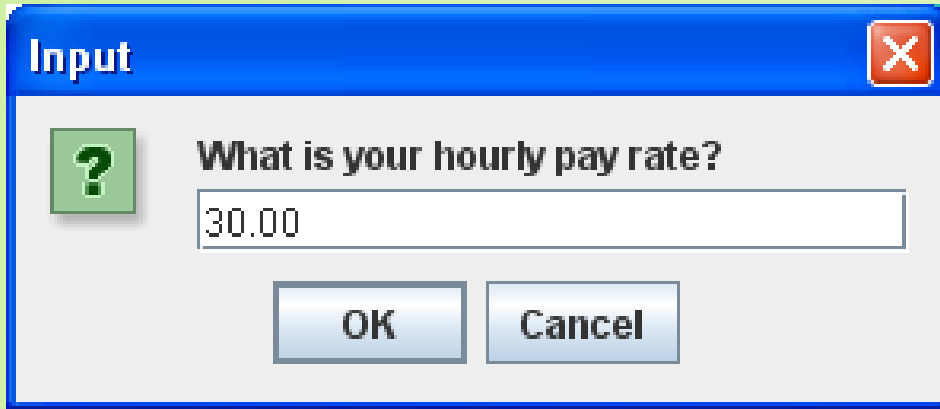
```
20  
21     JOptionPane.showMessageDialog(null, "Hello " + name +  
22         "! Your gross pay is: R" + grossPay);
```

```
23     System.exit(0);
```

```
24 }
```

```
25 }
```

```
1 import javax.swing.JOptionPane;
```



Input dialog box.
Input saved as **string**
and **converted** to
double.

```
10  
11     name = JOptionPane.showInputDialog("What is your name?");
```

```
12  
13     input = JOptionPane.showInputDialog("How many hours did you work?");  
14     hours = Integer.parseInt(input);
```

```
15  
16     input = JOptionPane.showInputDialog("What is your hourly pay rate?");  
17     payRate = Double.parseDouble(input);
```

```
18  
19     grossPay = hours * payRate;
```

```
20  
21     JOptionPane.showMessageDialog(null, "Hello " + name +  
22         "! Your gross pay is: R" + grossPay);
```

```
23     System.exit(0);
```

```
24 }
```

```
25 }
```



```
1 import javax.swing.JOptionPane;
2
3 public class DialogExample
4 {
5     public static void main(String[] args)
6     {
7         String input, name;
8         int hours;
9         double payRate, grossPay;
10
11         name = JOptionPane.showInputDialog("What is your name?");
12
13         input = JOptionPane.showInputDialog("How many hours did you work?");
14         hours = Integer.parseInt(input);
15
16         input = JOptionPane.showInputDialog("What is your hourly pay rate?");
17         payRate = Double.parseDouble(input);
18
19         grossPay = hours * payRate;
20
21         JOptionPane.showMessageDialog(null, "Hello " + name +
22                                     "! Your gross pay is: R" + grossPay);
23         System.exit(0);
24     }
25 }
```

Calculates the user's
gross pay by
multiplying *hours* with
payRate.

```
1 import javax.swing.JOptionPane;
```

```
2  
3  
4 Message
```



```
5 Hello Johan! Your gross pay is: R360.0
```

```
6  
7  
8 OK  
9
```

Message dialog box
displays the **gross pay**
of the user.

```
10  
11 name = JOptionPane.showInputDialog("What is your name?");
```

```
12  
13 input = JOptionPane.showInputDialog("How many hours did you work?");  
14 hours = Integer.parseInt(input);
```

```
15  
16 input = JOptionPane.showInputDialog("What is your hourly pay rate?");  
17 payRate = Double.parseDouble(input);
```

```
18  
19 grossPay = hours * payRate;
```

```
20  
21 JOptionPane.showMessageDialog(null, "Hello " + name +  
22 " ! Your gross pay is: R" + grossPay);
```


```
23 System.exit(0);
```

```
24 }
```

```
25 }
```

```
1 import javax.swing.JOptionPane;
2
3 public class DialogExample
4 {
5     public static void main(String[] args)
6     {
7         String input, name;
8         int hours;
9         double payRate, grossPay;
10
11         name = JOptionPane.showInputDialog("What is your name?");
12
13         input = JOptionPane.showInputDialog("How many hours did you work?");
14         hours = Integer.parseInt(input);
15
16         input = JOptionPane.showInputDialog("What is your hourly pay rate?");
17         payRate = Double.parseDouble(input);
18
19         grossPay = hours * payRate;
20
21         JOptionPane.showMessageDialog(null, "Hello " + name +
22                                     "! Your gross pay is: R" + grossPay);
23         System.exit(0);
24     }
25 }
```

This statement causes the program to **end**, and is **required** if you use the **JOptionPane class** to display dialog boxes.



Please read pages 123
and 124 on ***common
errors to avoid*** when
programming in Java.