Single-line Comments

Single-line comments start with two forward slashes (//).
Any text between // and the end of the line is ignored by Java (will not be executed).
This example uses a single-line comment before a line of code:

Java Comments

Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

```
public class Main {
  public static void main(String[] args) {
    // This is a comment
    System.out.println("Hello World");
  }
}
```

Java Multi-line Comments
Multi-line comments start with /* and ends with */.

Any text between /* and */ will be ignored by Java.

This example uses a multi-line comment (a comment block) to explain the code:

```
public class Main {
  public static void main(String[] args) {
    /* The code below will print the words Hello World
    to the screen, and it is amazing */
    System.out.println("Hello World");
  }
}
```

Java Variables

Java Variables

Variables are containers for storing data values.

In Java, there are different types of variables, for example:

String - stores text, such as "Hello". String values are surrounded by double quotes

int - stores integers (whole numbers), without decimals, such as 123 or -123

float - stores floating point numbers, with decimals, such as 19.99 or -19.99

char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes

boolean - stores values with two states: true or false

Declaring (Creating) Variables

To create a variable, you must specify the type and assign it a value:

```
public class Main {
  public static void main(String[] args) {
    String name = "John";
    System.out.println(name);
  }
}
```

Example

Create a variable called myNum of type int and assign it the value 15:

```
int myNum = 15;
System.out.println(myNum);
```

Java Declare Multiple Variables

```
public class Main {
  public static void main(String[] args)
{
    int x = 5, y = 6, z = 50;
    System.out.println(x + y + z);
  }
}
```

Identifiers

```
public class Main {
  public static void main(String[] args) {
    // Good
    int minutesPerHour = 60;

    // OK, but not so easy to understand what m actually is
    int m = 60;

    System.out.println(minutesPerHour);
    System.out.println(m);
  }
}
```

Java Identifiers

Identifiers
All Java variables must be identified with unique names.

These unique names are called identifiers.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

Note: It is recommended to use descriptive names in order to create understandable and maintainable code:

The general rules for naming variables are:

Names can contain letters, digits, underscores, and dollar signs
Names must begin with a letter
Names should start with a lowercase letter and it cannot contain whitespace
Names can also begin with $ and _ (but we will not use it in this tutorial)
Names are case sensitive ("myVar" and "myvar" are different variables)
Reserved words (like Java keywords, such as int or boolean) cannot be used as names

Java Data Types

```
public class Main {
  public static void main(String[] args) {
    int myNum = 5;               // integer (whole number)
    float myFloatNum = 5.99f;    // floating point number
    char myLetter = 'D';         // character
    boolean myBool = true;       // boolean
    String myText = "Hello";     // String
    System.out.println(myNum);
    System.out.println(myFloatNum);
    System.out.println(myLetter);
    System.out.println(myBool);
    System.out.println(myText);
  }
}
```

Data types are divided into two groups:

Primitive data types - includes byte, short, int, long, float, double, boolean and char
Non-primitive data types - such as String, Arrays and Classes (you will learn more about these in a later chapter)

Primitive Data Types

A primitive data type specifies the size and type of variable values, and it has no additional methods.

There are eight primitive data types in Java:

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Non-Primitive Data Types

Non-primitive data types are called reference types because they refer to objects.

The main difference between primitive and non-primitive data types are:

Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
A primitive type has always a value, while non-primitive types can be null.
A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.
Examples of non-primitive types are Strings, Arrays, Classes, Interface, etc. You will learn more about these in a later chapter.

Java Type Casting

Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

Widening Casting (automatically) - converting a smaller type to a larger type size
byte -> short -> char -> int -> long -> float -> double

Narrowing Casting (manually) - converting a larger type to a smaller size type
double -> float -> long -> int -> char -> short -> byte

Widening Casting

Widening casting is done automatically when passing a smaller size type to a larger size type:

```
public class Main {
  public static void main(String[] args) {
    int myInt = 9;
    double myDouble = myInt; // Automatic
casting: int to double

    System.out.println(myInt);      // Outputs 9
    System.out.println(myDouble);   // Outputs
9.0
  }
}
```

Narrowing Casting

Narrowing casting must be done manually by placing the type in parentheses in front of the value:

Example

```
public class Main {
  public static void main(String[] args) {
    double myDouble = 9.78d;
    int myInt = (int) myDouble; // Manual casting: double to int

    System.out.println(myDouble);   // Outputs 9.78
    System.out.println(myInt);      // Outputs 9
  }
}
```

Java Operators

Java Operators
Operators are used to perform operations on variables and values.

Java divides the operators into the following groups:

Arithmetic operators
Assignment operators
Comparison operators
Logical operators
Bitwise operators

Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.

Operator	Name	Description	Example	Try it
+	Addition	Adds together two values	x + y	
-	Subtraction	Subtracts one value from another	x - y	
*	Multiplication	Multiplies two values	x * y	
/	Division	Divides one value by another	x / y	
%	Modulus	Returns the division remainder	x % y	
++	Increment	Increases the value of a variable by 1	++x	
--	Decrement	Decreases the value of a variable by 1	--x	

Java Assignment Operators

Assignment operators are used to assign values to variables.

In the example below, we use the assignment operator (=) to assign the value 10 to a variable called x:

A list of all assignment operators:

Operator	Example	Same As	Try it
=	x = 5	x = 5	
+=	x += 3	x = x + 3	
-=	x -= 3	x = x - 3	
*=	x *= 3	x = x * 3	
/=	x /= 3	x = x / 3	
%=	x %= 3	x = x % 3	
&=	x &= 3	x = x & 3	
|=	x |= 3	x = x | 3	
^=	x ^= 3	x = x ^ 3	
>>=	x >>= 3	x = x >> 3	
<<=	x <<= 3	x = x << 3	

Java Comparison Operators

Comparison operators are used to compare two values (or variables). This is important in programming, because it helps us to find answers and make decisions.

The return value of a comparison is either true or false. These values are known as Boolean values, and you will learn more about them in the Booleans and If..Else chapter.

In the following example, we use the greater than operator (>) to find out if 5 is greater than 3:

Operator	Name	Example	Try it
==	Equal to	x == y	
!=	Not equal	x != y	
>	Greater than	x > y	
<	Less than	x < y	
>=	Greater than or equal to	x >= y	
<=	Less than or equal to	x <= y	

Java Logical Operators

You can also test for true or false values with logical operators.

Logical operators are used to determine the logic between variables or values:

Operator	Name	Description	Example	Try it
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10	
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4	
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)	

Java Strings
Strings are used for storing text.

A String variable contains a collection of characters surrounded by double quotes:

```
public class Main {
  public static void main(String[] args) {
    String greeting = "Hello";
    System.out.println(greeting);
  }
}
```

String Length

A String in Java is actually an object, which contain methods that can perform certain operations on strings. For example, the length of a string can be found with the length() method:

```
public class Main {
  public static void main(String[] args) {
    String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    System.out.println("The length of the txt string is: " +
txt.length());
  }
}
```