Program, Lexical, Primitive Data Types, Literals, Variables

First JAVA Program

```
class abc
{
        public static void main(String arg[]){
                System.out.print(“Hello Everyone");
        }
}
```

- Save:

For this example, the name of the source file should be **abc.java**.

- Compiling the Program:
- execute the compiler **javac**
- The **javac** compiler creates a file, **abc.class** that contains the bytecode version of the program. As discussed earlier, the Java bytecode is the intermediate representation of your program that contains instructions the Java Virtual Machine will execute.

First JAVA Program

```
class abc
{
        public static void main(String arg[]){
                System.out.print(“Hello Everyone");
        }
}
```

- Execute or Run the program :

you must use the Java application launcher (interpreter), **java**. And pass the class name eg. **Java abc**

- Keep in mind that Java is case-sensitive.
- **main( )** is simply a starting place for your program. A complex program may have dozens of classes, only one of which will need to have a **main()** method to get things started.

Lexical

- Java programs are a collection of whitespace, identifiers, literals, comments, operators, separators, and keywords.

Lexical

➢**Whitespace:** whitespace is a space, tab, or newline.

- Java is a free-form language. This means that you do not need to follow any special indentation rules.
- For instance, the Example program could have been written all on one line or in any other strange way

➢**Identifiers:**

- Identifiers are used to name things, such as classes, variables, and methods.
- An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters.
- They must not begin with a number.

Lexical

- **Literals:** A constant value.
- For instance, Integer 10
- floating-point value 1.414
- character constant ‘A’
- and string “Ram”
- **Comments:**
- there are three types of comments defined by Java.
- single-line //Single line comments
- Multiline /*Multiple line comments*/
- Documentation comment /** documentation comments*/
- Documentation comment is used to produce an HTML file that documents your program. They must not begin with a number.

Lexical

- Javadoc is a tool which comes with JDK and it is used for generating Java code documentation in HTML format from Java file.
- We can include HTML tags inside the javadoc description.

Tag	Description	Syntax
@author	Adds the author of a class.	@author name-text
{@code}	Displays text in code font without interpreting the text as HTML markup or nested javadoc tags.	{@code text}
@deprecated	Adds a comment indicating that this API should no longer be used.	@deprecated deprecatedtext
@exception	Adds a **Throws** subheading to the generated documentation, with the classname and description text.	@exception class-name description
{@link}	Inserts an in-line link with the visible text label that points to the documentation for the specified package, class, or member name of a referenced class.	{@link package.class#member label}
@param	Adds a parameter with the specified parameter-name followed by the specified description to the "Parameters" section.	@param parameter-name description
@return	Adds a "Returns" section with the description text.	@return description
@see	Adds a "See Also" heading with a link or text entry that points to reference.	@see reference
@since	Adds a "Since" heading with the specified since-text to the generated documentation.	@since release
@throws	The @throws and @exception tags are synonyms.	@throws class-name description
@version	Adds a "Version" subheading with the specified version-text to the generated docs when the -version option is used.	@version version-text

Lexical

➢Operators

- There are many types of operators in Java
- Unary Operator ++,-- (prefix and postfix)
- Arithmetic Operator + - * / %
- Shift Operator << >> >>>
- Relational Operator < > <= >= == !=
- Bitwise Operator & ^ |
- Logical Operator && ||
- Ternary Operator ? :
- Assignment Operator = += -= *= /= %= &= ^= |= <<= >>= >>>=

Lexical

➢Separators:

- The most commonly used separator in Java is the semicolon. As you have seen, it is used to terminate statements.

Symbol	Name	Purpose
( )	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a **for** statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.
::	Colons	Used to create a method or constructor reference. (Added by JDK 8.)

Lexical

➢keywords :

- The keywords, combined with the syntax of the operators and separators.

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

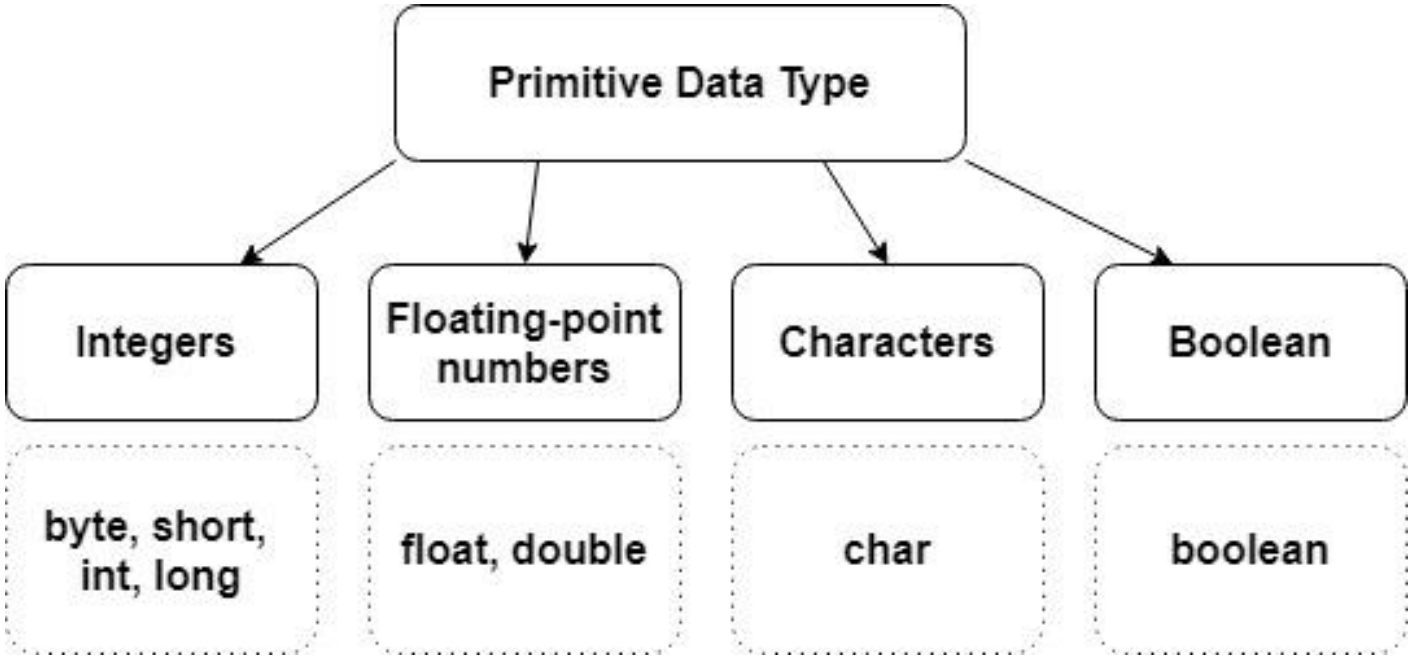
Lexical

➢ **keywords :**

- These keywords cannot be used as identifiers. Thus, they cannot be used as names for a variable, class, or method.
- The keywords **const** and **goto** are reserved but not used.
- In addition to the keywords, Java reserves the following: **true**, **false**, and **null**. These are values defined by Java. You may not use these words for the names of variables, classes, and so on.

Data Types

- Java defines eight primitive types of data: byte, short, int, long, char, float, double, and boolean.
- The primitive types represent single values—not complex objects.



- **Integers:** This group includes **byte, short, int**, and **long**, which are for whole-valued signed numbers.

Data Types: Integers

- All of these are signed, positive and negative values.
- Java does not support unsigned, positive-only integers.
- **Byte:** The smallest integer type is byte. This is a signed 8-bit type that has a range from –128 to 127.
- Byte variables are declared by use of the **byte** keyword.

Name	Width	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

Data Types: Integers

- **short:** short is a signed 16-bit type.
- It has a range from –32,768 to 32,767.
- It is probably the least used Java type. Short variables are declared by use of the **short** keyword.

Name	Width	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

Data Types: Integers

- **int :** The most commonly used integer type is int.
- It is a signed 32-bit type that has a range from –2,147,483,648 to 2,147,483,647.
- In addition to other uses, variables of type **int** are commonly employed to control loops and to index arrays.

Name	Width	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

Data Types: Integers

- Although you might think that using a byte or short would be more efficient than using an int in situations in which the larger range of an int is not needed, this may not be the case.
- The reason is that when byte and short values are used in an expression, they are promoted to int when the expression is evaluated.

Name	Width	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

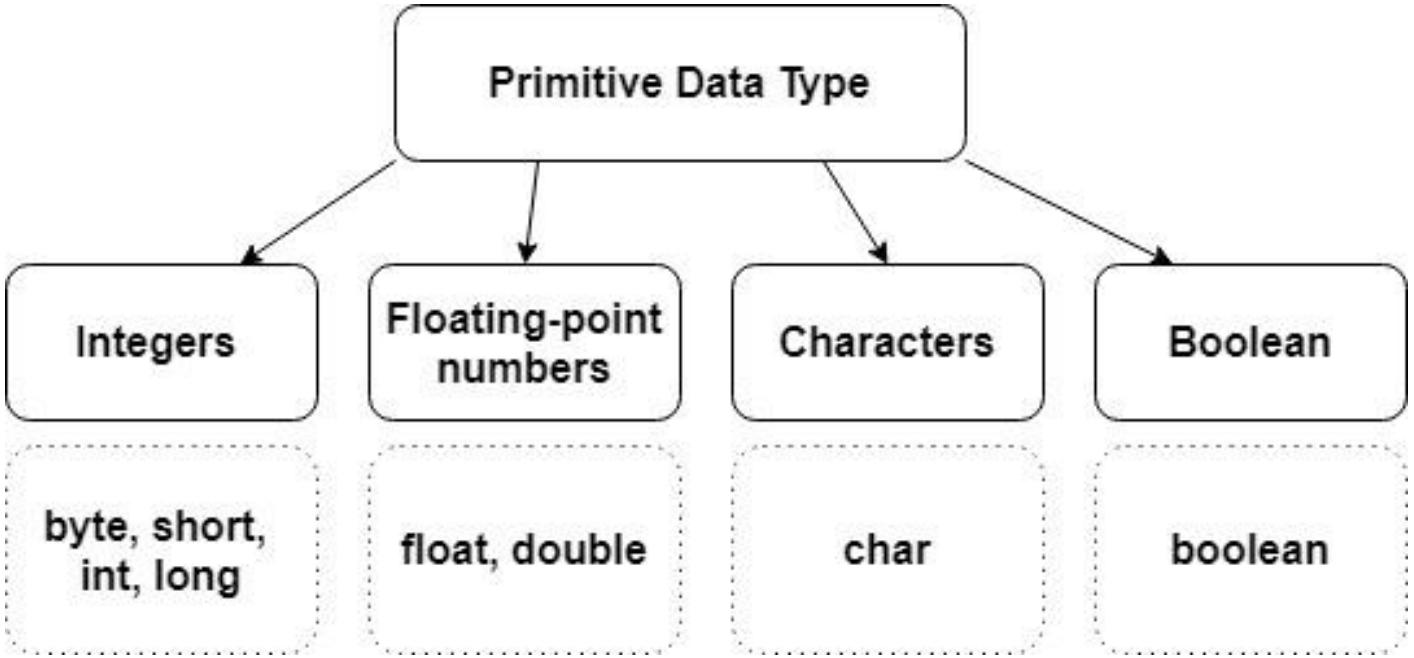
Data Types: Integers

- **long:** long is a signed 64-bit type and is useful for those occasions where an int type is not large enough to hold the desired value.
- The range of a long is quite large.

Name	Width	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

Data Types

- Java defines eight primitive types of data: byte, short, int, long, char, float, double, and boolean.



- **Floating-point numbers:** This group includes **float** and **double**, which represent numbers with fractional precision.

Data Types: Floating-Point Types

- **floating-point numbers:** Floating-point numbers, also known as real numbers, are used when evaluating expressions that require fractional precision.
- There are two kinds of floating-point types, float and double, which represent single- and double-precision numbers, respectively.

Data Types: Floating-Point Types

- **float:** The type float specifies a single-precision value that uses 32 bits of storage.
- Float variables are declared by use of the **float** keyword.

Name	Width in Bits	Approximate Range
double	64	4.9e–324 to 1.8e+308
float	32	1.4e–045 to 3.4e+038

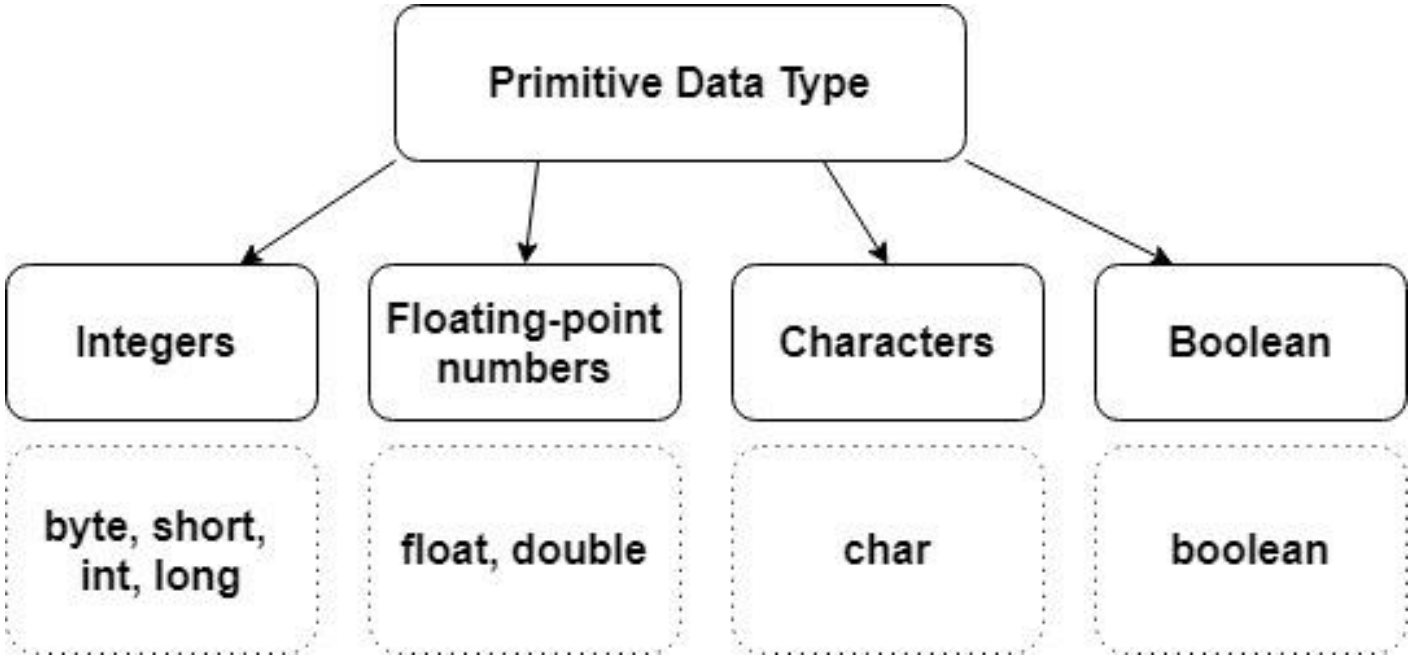
Data Types: Floating-Point Types

- **double:** Double precision, as denoted by the **double** keyword, uses 64 bits to store a value.
- All transcendental math functions, such as sin( ), cos( ), and sqrt( ), return double values.

Name	Width in Bits	Approximate Range
double	64	4.9e–324 to 1.8e+308
float	32	1.4e–045 to 3.4e+038

Data Types

- Java defines eight primitive types of data: byte, short, int, long, char, float, double, and boolean.



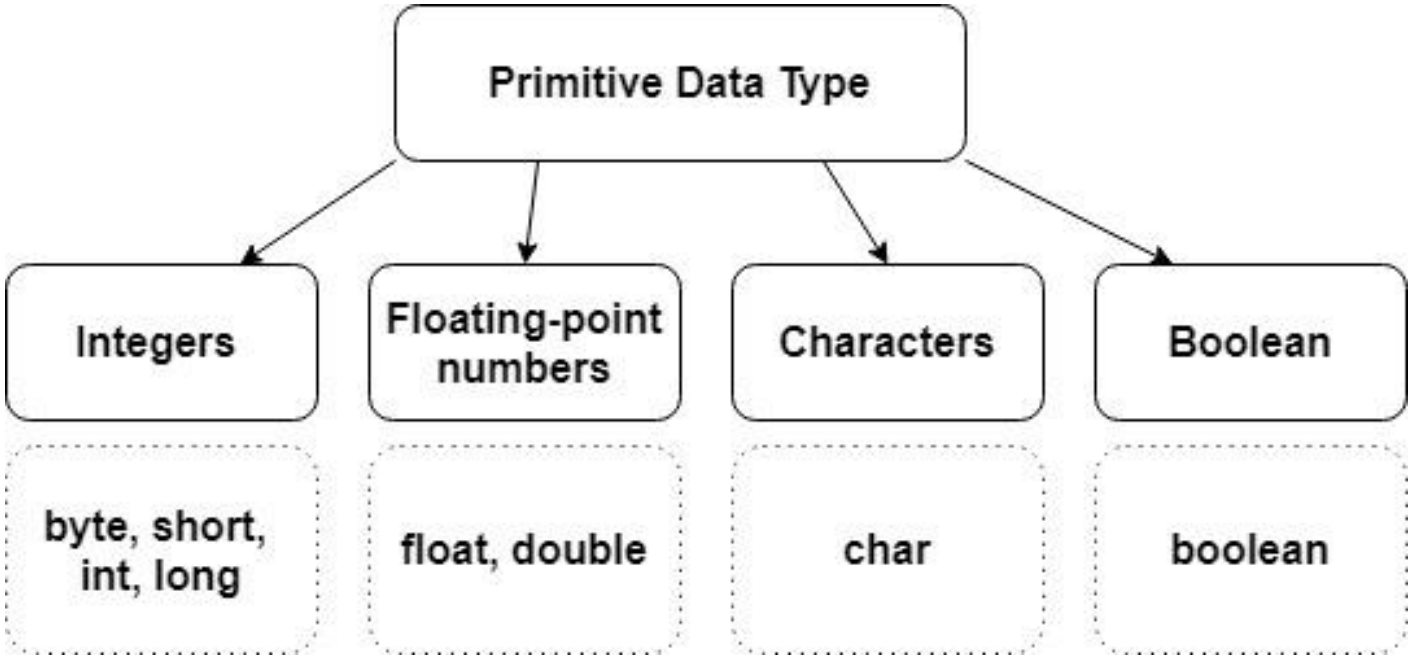
- **Characters:** This group includes **char**, which represents symbols in a character set, like letters and numbers.

Data Types: Characters

- **Characters:** the data type used to store characters is char.
- Java uses Unicode to represent characters.
- in Java char is a 16-bit type.
- It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).
- The range of a char is 0 to 65,536.
- There are no negative chars.
- The standard set of characters known as ASCII still ranges from 0 to 127 as always, and the extended 8-bit character set, ISO-Latin-1, ranges from 0 to 255..

Data Types

- Java defines eight primitive types of data: byte, short, int, long, char, float, double, and boolean.



- **Boolean:** This group includes **boolean**, which is a special type for representing true/false values.

Data Types: Booleans

- **Booleans:** Java has a primitive type, called **boolean**, for logical values.
- It can have only one of two possible values, **true** or **false**.
- This is the type returned by all relational operators.

Data Types: Default Values

- It's not always necessary to assign a value when a field is declared.
- Fields that are declared but not initialized will be set to a reasonable default by the compiler.
- Generally speaking, this default will be zero or null, depending on the data type.

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

Data Types: Default Values

- Local variables are slightly different.
- The compiler never assigns a default value to an uninitialized local variable.
- If you cannot initialize your local variable where it is declared, make sure to assign it a value before you attempt to use it.
- Accessing an uninitialized local variable will result in a compile-time error.

Literals

- Literals can be assigned to any primitive type data.
- Here are different types of literals corresponding to the primitive data types in Java.
- **Integer Literals:**
- Any whole number value is an integer literal. Examples are 1, 2, 3, and 42. These are all decimal values, meaning they are describing a base 10 number.
- There are 4 types of integer literals in Java:

1) binary (base 2)
2) decimal (base 10)
3) octal (base 8)
4) hexadecimal (base 16)

Literals

1) binary (base 2)

- binary literals starts with **0b** or **0B**. (defined in JDK 7)

2) decimal (base 10)

3) octal (base 8)

- Octal values are denoted in Java by a leading zero (**0**).

4) hexadecimal

- hexadecimal constant values are denoted by leading zero-x, (**0x** or 0X).
- In JDK 7, you can embed one or more underscores in an integer literal.
- Doing so makes it easier to read large integer literals. When the literal is compiled, the underscores are discarded.

Literals

```
public class abc
{
   public static void main(String arg[]) {
     abc obj=new abc();
     int hexVal = 0x1a;
     int octVal = 032;
     int binVal = 0b11010;
   System.out.print("hexa  Value="+hexVal+"  Octal
   Value="+octVal+" Binary Value="+binVal);
   }
}
```

Literals

- **Floating-Point Literals:**
- Floating-point numbers can be expressed in either standard or scientific notation.
- Standard notation consists decimal point followed by a fractional component. For example, 2.0, 3.14159, and 0.6667.
- Scientific notation uses a standard-notation, floating-point number plus a suffix that specifies a **power of 10** by which the number is to be multiplied.
- The exponent is indicated by an **E** or **e** followed by a decimal number, which can be positive or negative. Examples include 6.022E23, 314159E–05, and 2e+100.

Literals

- **Floating-Point Literals:**
- Floating-point literals in Java default to double precision.
- To specify a **float** literal, you must append an **F** or **f** to the constant.
- You can also explicitly specify a double literal by appending a **D** or **d**.
- You can embed one or more underscores in a floating-point literal. This feature works the same as it does for integer literals.
- It is also permissible to use underscores in the fractional portion of the number.

Literals

- **Boolean Literals:**
- Boolean literals are simple.
- There are only two logical values that a boolean value can have, true and false.
- The values of **true** and **false** do not convert into any numerical representation.
- Boolean literals can only be assigned to variables declared as boolean or used in expressions with Boolean operators

Literals

- **Character Literals:**
- Characters in Java are indices into the Unicode character set.
- They are 16-bit values that can be converted into integers and manipulated with the integer operators, such as the addition and subtraction operators.
- A literal character is represented inside a pair of single quotes.
- For characters that are impossible to enter directly, there are several **escape sequences** that allow you to enter the character you need, such as '\'' for the single-quote character itself and '\n' for the newline character.

Literals

- **Character Literals:**

Escape Sequence	Description
\ddd	Octal character (ddd)
\uxxxx	Hexadecimal Unicode character (xxxx)
\'	Single quote
\"	Double quote
\\	Backslash
\r	Carriage return
\n	New line (also known as line feed)
\f	Form feed
\t	Tab
\b	Backspace

- There is also a mechanism for directly entering the value of a character in octal or hexadecimal.
- For octal notation, use the backslash followed by the three-digit number. For example, ' \141' is the letter 'a'.
- For hexadecimal, you enter a backslash-**u** (**\u**), then exactly four hexadecimal digits. For example, '\u0061' is the letter 'a‘.

Literals

- **String Literals:**
- Strings are actually object types in Java.
- String literals are specified like they are in most other languages—by enclosing a sequence of characters between a pair of double quotes.
- The **escape sequences** and **octal/hexadecimal notations** that were defined for character literals work the same way inside of string literals.
- One important thing to note about Java strings is that they must begin and end on the same line.
- There is no line-continuation escape sequence as there is in some other languages

Variables

- The variable is the basic unit of storage in a Java program.
- A variable is defined by the combination of an **identifier**, a **type**, and an **optional initializer**.

type identifier = value

- ***type*** is one of Java's atomic types, or the name of a class or interface.
- The ***identifier*** is the name of the variable.
- You can **initialize** the variable by specifying an *equal sign* and a *value*.
- For instance, int a=10;
- In addition, all variables have a **scope**, which defines their **visibility**, and a lifetime.
- All variables must be declared before they can be used.
- To declare more than one variable of the specified type, use a comma-separated list. For instance, int a, b=5,c=10;

Variables

Variables Initialization:

- Although the preceding examples have used only constants as initializers
- In Java variables can be initialized dynamically, using any expression valid at the time the variable is declared.

For instance,

```
int a=10, b=20,c;
c=a+b;
```