
# CORE JAVA

CONTENTS:

# 1.JAVA:

Java is a popular and widely used programming language known for its versatility, portability, and robustness. It was developed by James Gosling and his team at Sun Microsystems (now owned by Oracle Corporation) and was released to the public in 1995.

Here are some key features and concepts related to Java:

**Object-oriented**: Java is an object-oriented programming language, which means it focuses on creating reusable objects that encapsulate data and behavior.

**Platform independence**: One of Java's major strengths is its "write once, run anywhere" principle. Java programs can run on any device or operating system with a Java Virtual Machine (JVM) installed, making them highly portable.

**Java Virtual Machine** (JVM): The JVM is a crucial component of Java that executes Java bytecode. It acts as an intermediary between the Java code and the underlying hardware or operating system.

**Syntax and structure**: Java's syntax is derived from C and C++, making it familiar to programmers of those languages. It uses curly braces {} to define code blocks and semicolons (;) to terminate statements.

**Strong typing**: Java is statically typed, meaning variable types are explicitly declared and checked at compile time, reducing the chances of runtime errors.

**Automatic memory management**: Java implements automatic memory management through a process called garbage collection. Developers don't need to explicitly free up memory; the JVM takes care of memory allocation and deallocation.

**Standard Library**: Java provides a vast standard library, known as the Java Development Kit (JDK), which includes classes and methods for various tasks, such as input/output, networking, and data manipulation.

**Multi-threading**: Java supports multithreading, allowing developers to create and manage multiple threads of execution within a program. This feature enables concurrent programming and improves application performance.

**Exception handling**: Java provides a robust exception handling mechanism to deal with runtime errors. Developers can catch and handle exceptions gracefully, preventing the program from crashing.

**Community and ecosystem**: Java has a large and active developer community, which has contributed to a vast ecosystem of libraries, frameworks, and tools. This ecosystem enables developers to build a wide range of applications, from desktop software to enterprise systems and mobile apps using frameworks like Spring and Android.

# 2.History of Java :

Java is a programming language that was developed by James Gosling and his team at Sun Microsystems, which was later acquired by Oracle Corporation. The history of Java begins in the early 1990s when Sun Microsystems was looking for a way to develop software for consumer electronic devices.

## Here's a brief history of Java:

Origins: In 1991, James Gosling, along with his team, started working on a project known as "Green" at Sun Microsystems. The project aimed to create a programming language for consumer electronics. Initially, the language was called "Oak" but was later renamed to "Java" due to trademark conflicts.

**Platform Independence:** One of the key goals of Java was to create a language that could run on different platforms without the need for recompilation. The team developed a virtual machine known as the Java Virtual Machine (JVM), which allowed Java programs to be executed on any device or operating system that had a JVM implementation.

**Release of Java 1.0**: The first public release of Java, known as Java 1.0, was made available to the public in 1996. It included features such as the Java Development Kit (JDK), the Java Applet API for creating web applets, and the Swing GUI toolkit.

**Rise of Applets:** Java gained significant popularity with the introduction of Java applets. Applets were small programs that could be embedded in webpages and run within a browser. They allowed for interactive content and helped popularize Java as a language for web development.

**Enterprise Java**: As Java's popularity grew, it began to be used more extensively in enterprise-level applications. Sun Microsystems introduced Java Enterprise Edition (Java EE), a platform for building scalable and robust enterprise applications. Java EE provided libraries, APIs, and tools for developing server-side applications.

**Open Sourcing**: In 2006, Sun Microsystems released the Java platform under the GNU General Public License (GPL) as OpenJDK. This move made the Java platform open source, allowing developers to view, modify, and distribute the source code.

**Java Community Process**: Sun Microsystems established the Java Community Process (JCP) to manage and evolve the Java platform. The JCP allowed developers and organizations to participate in the development of Java by proposing and reviewing new specifications.

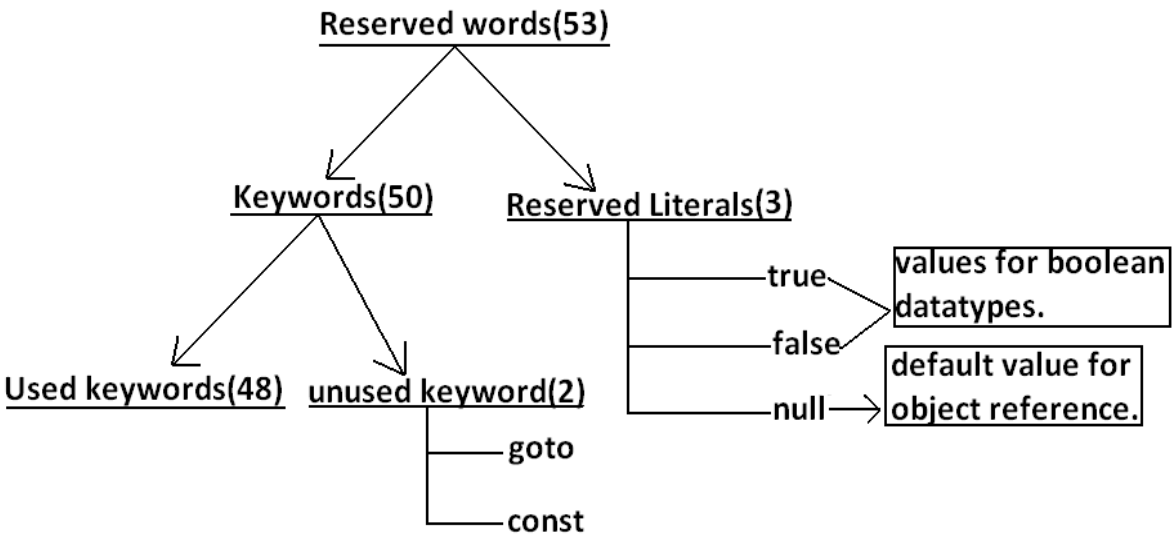
**Acquisition by Oracle**: In 2010, Oracle Corporation acquired Sun Microsystems, including the Java platform. Oracle continued to develop and maintain Java, releasing new versions and updates.

**Java 8 and Beyond**: Java 8, released in 2014, introduced significant changes to the language, including lambda expressions, the Stream API, and default methods in interfaces. Subsequent releases, such as Java 9, 10, and 11, brought additional enhancements and features to the language and the platform.

**Current State**: Java remains one of the most popular and widely used programming languages in the world. It is used for a wide range of applications, including web development, mobile app development (with the Android platform), enterprise software, scientific computing, and more.

Throughout its history, Java has evolved and adapted to the changing needs of developers and technology. Its platform independence, robustness, and extensive ecosystem have contributed to its enduring popularity.

# 3.KEYWORDS




## Reserved words for data types:

```
1)      byte
2)      short
3)      int
4)      long
5)      float
6)      double
7)      char
8)      boolean
```

## Reserved words for flow control:

```
1)      if
2)      else
3)      switch
4)      case
5)      default
6)      for
7)      do
8)      while
9)      break
10)     continue
11)     return
```

## Keywords for modifiers:

```
1)      public
2)      private
3)      protected
4)      static
5)      final
6)      abstract
7)      synchronized
```

8) native
9) strictfp(1.2 version)
10) transient
11) volatile

## Keywords for exception handling:(6)

1) try
2) catch
3) finally
4) throw
5) throws
6) assert(1.4 version)

## Class related keywords:

1) class
2) package
3) import
4) extends
5) implements
6) interface

## Object related keywords:

1) new
2) instanceof
3) super
4) this

## Void return type keyword:

**If a method won't return anything compulsory that method should be declared with thevoid return type in java but it is optional in C++.**

1) **void**

## Unused keywords:

**goto: Create several problems in old languages and hence it is banned in java.Const: Use final instead of this.**
**By mistake if we are using these keywords in our program we will get compile timeerror.**

## Reserved literals:

1) **true values for boolean data type.**
2) **false**
3) **null---------- default value for object reference.**

## Enum:

**This keyword introduced in 1.5v to define a group of named constants**

**Example:**

```
enum Beer

{

KF, RC, KO, FO;

}
```

## Conclusions :

1. **All reserved words in java contain only lowercase alphabet symbols.**
2. **New keywords in java are:**
3. strictfp -------- --1.2v
4. assert..... v
5. enum..... v
6. **In java we have only new keyword but not delete because destruction of uselessobjects is the responsibility of Garbage Collection.**
7. instanceof but not instanceOf
8. strictfp but not strictFp
9. const but not Constant
10. syncronized but not syncronize
11. extends but not extend
12. implements but not implement
13. import but not imports
14. int but not Int
15.

## Which of the following list contains only java reserved words ?

1. **final, finally, finalize (invalid) //here finalize is a method in Object class.**
2. **throw, throws, thrown(invalid) //thrown is not available in java**
3. **break, continue, return, exit(invalid) //exit is not reserved keyword**
4. **goto, constant(invalid) //here constant is not reserved keyword**
5. **byte, short, Integer, long(invalid) //here Integer is a wrapper class**
6. **extends, implements, imports(invalid) //imports keyword is not available in java**
7. **finalize, synchronized(invalid) //finalize is a method in Object class**
8. **instanceof, sizeOf(invalid) //sizeOf is not reserved keyword**
9. **new, delete(invalid) //delete is not a keyword**
10. **None of the above(valid)**

## Which of the following are valid java keywords?

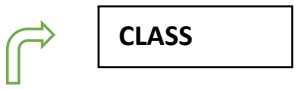
1. **public(valid)**
2. **static(valid)**
3. **void(valid)**
4. **main(invalid)**

5. **String(invalid)**
6. **args(invalid)**

## Class:

In Java, a class is a blueprint or a template for creating objects. It defines the structure and behavior of objects of that particular type. A class encapsulates data (in the form of fields or variables) and methods (functions) that operate on that data.

**Example :**




```
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello, World!");
   }
}
```

In this program, we have a class named HelloWorld. It contains a single method called main, which is the entry point of the program. The main method is declared as public, which means it can be accessed from outside the class.

Inside the main method, we have a single line of code: System.out.println("Hello, World!");. This line uses the System.out object to print the string "Hello, World!" to the console. The println method is used to print the string and move the cursor to the next line.

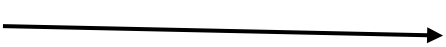
To run this program, save it with a .java extension (e.g., HelloWorld.java), then compile it using the Java compiler. After successful compilation, you can execute the program using the Java Virtual Machine (JVM), and you will see the output "Hello, World!" printed in the console.

## Object:

**In Java, an object is an instance of a class. It represents a specific entity or a real-world concept that is defined by the class. Objects are created from classes using the new keyword and can be assigned to variables or used directly.**

**Example**:

```
public class Car {
   private String brand;
   private String color;
   private int year;

   public Car(String brand, String color, int year) {
      this.brand = brand;
      this.color = color;
      this.year = year;
   }
```



OBJECT:

**Brand**

**Color**

**year**

```
    public void startEngine() {
        System.out.println("Starting the engine of the " + brand + " car...");
    }

    public void drive() {
        System.out.println("Driving the " + brand + " car. Vroom vroom!");
    }

    public static void main(String[] args) {
        Car myCar = new Car("Toyota", "Blue", 2022);

        System.out.println("My car is a " + myCar.getColor() + " " + myCar.getBrand() + " car
made in " + myCar.getYear());
        myCar.startEngine();
        myCar.drive();
    }

    public String getBrand() {
        return brand;
    }

    public String getColor() {
        return color;
    }

    public int getYear() {
        return year;
    }
}
```

This example, we have a Car class that represents a car object. It has three private fields: brand, color, and year. The class also has a constructor to initialize these fields and three methods: startEngine(), drive(), and getter methods for the fields.

In the main method, we create an instance of the Car class named myCar using the new keyword and passing the brand, color, and year as arguments to the constructor. We then access the object's fields and methods using dot notation.

## BLOCK:

In Java, a block refers to a group of zero or more statements enclosed within a pair of curly braces "{ }". Blocks are commonly used to define the scope of variables, control the flow of execution, and organize code into logical units.

Example:

```
public class Example {
    public static void main(String[] args) {
        // Code outside the block
```

```
        {
            // Code inside the block
            int x = 5;
            System.out.println("Inside the block: " + x);
        }

        // Code outside the block

        // The variable 'x' is not accessible here because it was declared inside the block
    }
}
```

In the above example, the block is denoted by the curly braces { }. Inside the block, you can declare variables, define methods, or write any valid Java statements. The code inside the block has its own scope, and variables declared within the block are only accessible within that block or nested blocks. Once the program execution goes outside the block, those variables are no longer accessible.

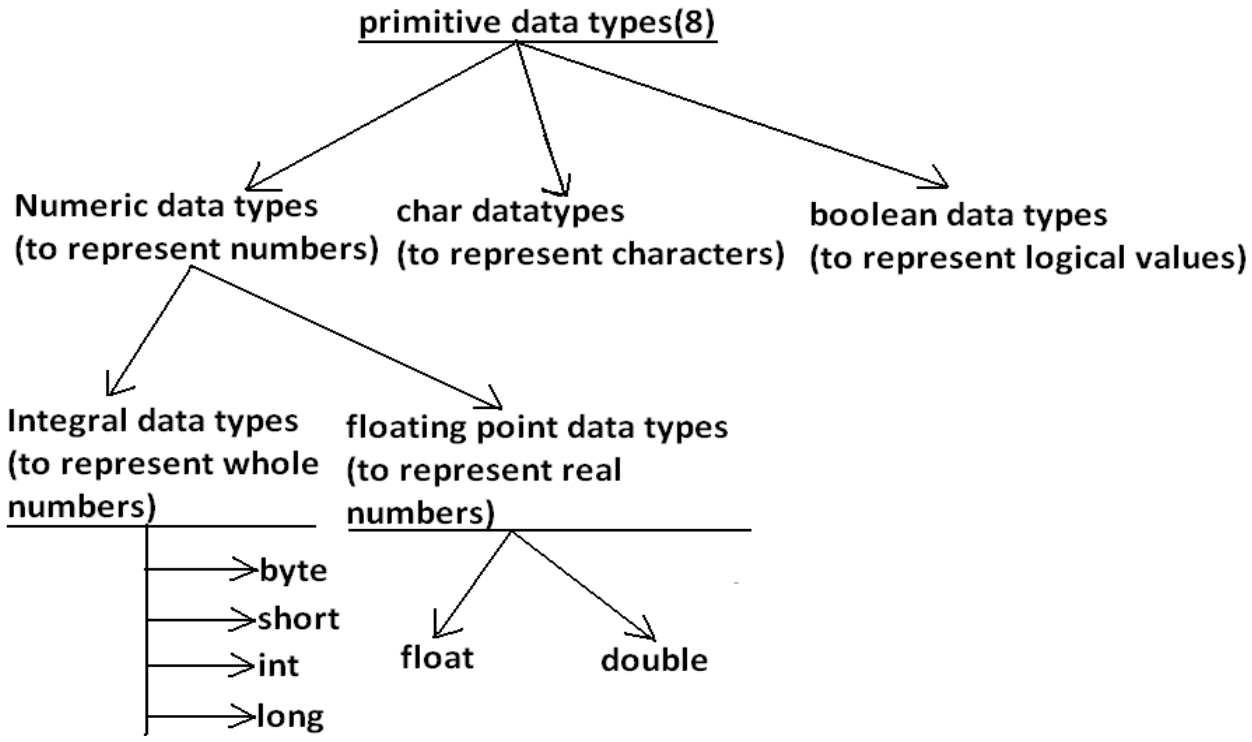
Blocks are commonly used with control flow statements like if, for, while, and switch to group multiple statements together. They are also used to define methods, constructors, and classes.

## Data types:

# Java is pure object oriented programming or not?

**Java is not considered as pure object oriented programming language because several oops features (like multiple inheritance, operator overloading) are not supported by java moreover we are depending on primitive data types which are non objects.**

**Diagram:**




**Except Boolean and char all remaining data types are considered as signed data types because we can represent both "+ve" and"-ve" numbers.**

# Integral data types :

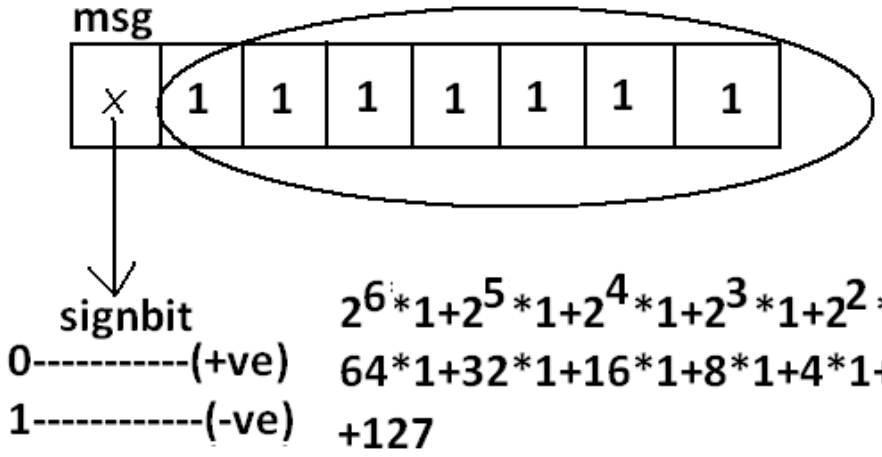
**Byte:**

**Size: 1byte (8bits)**

**Minvalue: -128**

**Maxvalue: +127**

**Range:-128to 127[$-2^7$ to $2^7-1$]**




- **The most significant bit acts as sign bit. "0" means "+ve" number and "1"means "–ve" number.**
- **"+ve" numbers will be represented directly in the memory whereas "–ve"numbers will be represented in 2's complement form.**

**Example:**

```
byte b=10;

byte b2=130;//C.E:possible loss of precision

             found : int

             required : byte

byte b=10.5;// C.E:possible loss of precisionbyte
b=true;//C.E:incompatible types

byte b="ashok";//C.E:incompatible types

               found:java.lang.String

               required : byte
```

**Byte data type is best suitable if we are handling data in terms of streams either fromthe file or from the network.**

**Short: The most rarely used data type in java is short.**

**Size: 2 bytes**

**Range: -32768 to 32767($-2^{15}$ to $2^{15}-1$)**

**Example:**

```
short s=130;

short s=32768;//C.E:possible
loss of precisionshort
s=true;//C.E:incompatible types
```

**Short data type is best suitable for 16 bit processors like 8086 but these**

**processors are completely outdated and hence the corresponding short data type is also out data type.**

## Int:

**This is most commonly used data type in java.**

```
Size: 4 bytes

Range:-2147483648 to 2147483647 (-2^31 to 2^31-1)
```

**Example:**

```
int i=130;

int i=10.5;//C.E:possible
loss of precisionint
i=true;//C.E:incompatible
types
```

## Long:

**Whenever int is not enough to hold big values then we should go for long data type.Example:**
**To hold the no. Of characters present in a big file int may not enough hence the returntype of length() method is long.**

```
long
l=f.length();//f
is a fileSize: 8
bytes

Range:-2^63  to 2^63-1
```

**Note: All the above data types (byte, short, int and long) can be used to represent wholenumbers. If we want to represent real numbers then we should go for floating point data types.**

## Floating Point Data types:

| Float | double |
|---|---|
| If we want to 5 to 6 decimal places of accuracy then we should go for float. | If we want to 14 to 15 decimal places of accuracy then we should go for double. |
| Size:4 bytes. | Size:8 bytes. |
| Range:-3.4e38 to 3.4e38. | -1.7e308 to1.7e308. |
| float follows single precision. | double follows double precision. |

## Boolean data type:

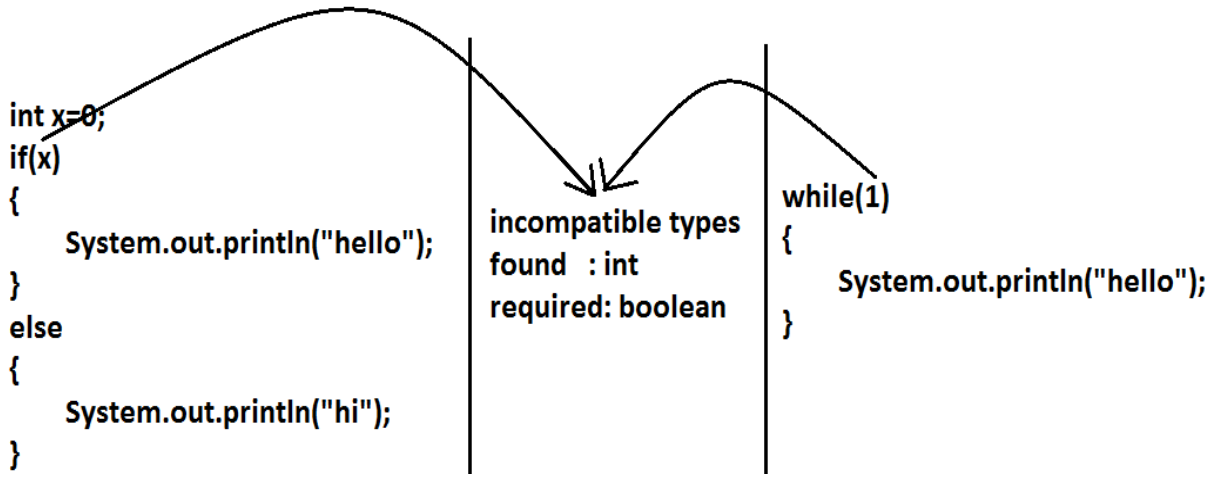
```
Size: Not applicable (virtual machine dependent)

Range: Not applicable but allowed values are true or false.
```

## Which of the following boolean declarations are valid?

**Example 1:**

```
boolean b=true;

boolean b=True;//C.E:cannot
find symbol boolean
b="True";//C.E:incompatible
typesboolean
b=0;//C.E:incompatible
types
```

**Example 2:**




## Char data type:

**In old languages like C & C++ are ASCII code based the no.Of ASCII code charactersare < 256 to represent these 256 characters 8 - bits enough hence char size in old languages 1 byte.**

**In java we are allowed to use any worldwide alphabets character and java is Unicodebased and no.Of unicode characters are > 256 and <= 65536 to represent all these characters one byte is not enough compulsory we should go for 2 bytes.**

```
Size: 2 bytes

Range: 0 to 65535
```

**Example:**

```
char ch1=97;

char ch2=65536;//C.E:possible loss of precision
```

## Summary of java primitive data type:

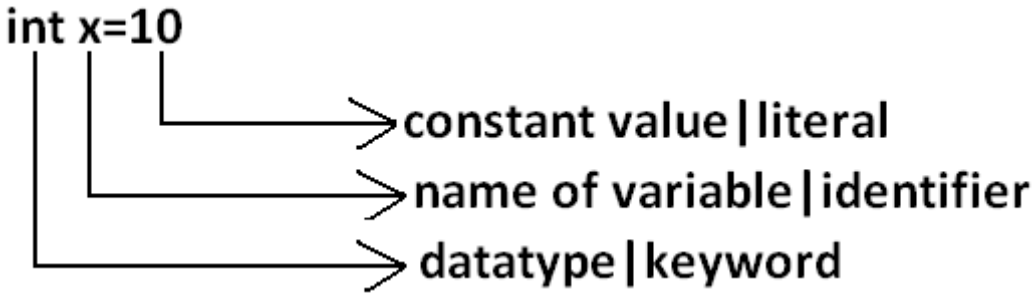
| data type | Size | Range | Corresponding Wrapper class | Default value |
|---|---|---|---|---|
| byte | 1 byte | $-2^7$ to $2^7-1$(-128 to 127) | Byte | 0 |
| short | 2 bytes | $-2^{15}$ to $2^{15}-1$ (-32768 to | Short | 0 |

| | | | | |
|---|---|---|---|---|
| | | 32767) | | |
| int | 4 bytes | $-2^{31}$ to $2^{31}-1$ (-2147483648 to 2147483647) | Integer | 0 |
| long | 8 bytes | $-2^{63}$ to $2^{63}-1$ | Long | 0 |
| float | 4 bytes | -3.4e38 to 3.4e38 | Float | 0.0 |
| double | 8 bytes | -1.7e308 to 1.7e308 | Double | 0.0 |
| boolean | Not applicable | Not applicable(but allowed values true\|false) | Boolean | false |
| char | 2 bytes | 0 to 65535 | Character | 0(represents blank space) |

**The default value for the object references is "null".**

## Literals:

**Any constant value which can be assigned to the variable is called literal.Example:**




## Integral Literals:

**For the integral data types (byte, short, int and long) we can specify literal value in thefollowing ways.**

**1) Decimal literals: Allowed digits are 0 to 9.**
**2) Example: int x=10;**

**3) Octal literals: Allowed digits are 0 to 7.**

**4) Literal value should be prefixed with zero.Example: int x=010;**

**5) Hexa Decimal literals:**

- **The allowed digits are 0 to 9, A to Z.**
- **For the extra digits we can use both upper case and lower case characters.**
- **This is one of very few areas where java is not case sensitive.**
- **Literal value should be prefixed**

**with ox(or)oX.Example: int x=0x10;**

**These are the only possible ways to specify integral literal.**

**Which of the following are valid declarations?**

1. **int x=0777; //(valid)**
2. **int x=0786; //C.E:integer number too large: 0786(invalid)**
3. **int x=0xFACE; (valid)**
4. **int x=0xbeef; (valid)**
5. **int x=0xBeer; //C.E:';' expected(invalid) //:int x=0xBeer; ^// ^**

**Example:**
**int x=10; int y=010; int z=0x10;**

```
System.out.println(x+"----"+y+"----"+z); //10----8 --- 16
```

**By default every integral literal is int type but we can specify explicitly as long type by suffixing with small "l" (or) capital "L".**
**Example:**
**int x=10;(valid) long l=10L;(valid)long l=10;(valid)**

```
int x=10l;//C.E:possible loss of precision(invalid)

              found : long

              required : int
```

**There is no direct way to specify byte and short literals explicitly. But whenever we are assigning integral literal to the byte variables and its value within the range of byte compiler automatically treats as byte literal. Similarly short literal also.**

**Example:**

```
byte b=127;(valid)

byte b=130;//C.E:possible loss of precision(invalid)

short s=32767;(valid)
```

## Floating Point Literals:

**Floating point literal is by default double type but we can specify explicitly as float typeby suffixing with f or F.**

**Example:**

```
float f=123.456;//C.E:possible loss of precision(invalid)float
f=123.456f;(valid)

double d=123.456;(valid)
```

**We can specify explicitly floating point literal as double type by suffixing with d or D.**

**Example:**

```
double d=123.456D;
```

**We can specify floating point literal only in decimal form and we can't specify in**

**octaland hexadecimal forms.**

**Example:**

```
double d=123.456;(valid)

double d=0123.456;(valid) //it is treated as decimal value
but not octaldouble d=0x123.456;//C.E:malformed floating
point literal(invalid)
```

**Which of the following floating point declarations are valid?**

1. **float f=123.456; //C.E:possible loss of precision(invalid)**
2. **float f=123.456D; //C.E:possible loss of precision(invalid)**
3. **double d=0x123.456; //C.E:malformed floating point literal(invalid)**
4. **double d=0xFace; (valid)**
5. **double d=0xBeef; (valid)**

**We can assign integral literal directly to the floating point data types and that integralliteral can be specified in decimal , octal and Hexa decimal form also.**

```
double d=0xBeef;
System.out.printl
n(d);//48879.0
```

**But we can't assign floating point literal directly to the integral types.**

**Example:**

```
int x=10.0;//C.E:possible loss of precision
```

**We can specify floating point literal even in exponential form also(significant notation).**

**Example:**

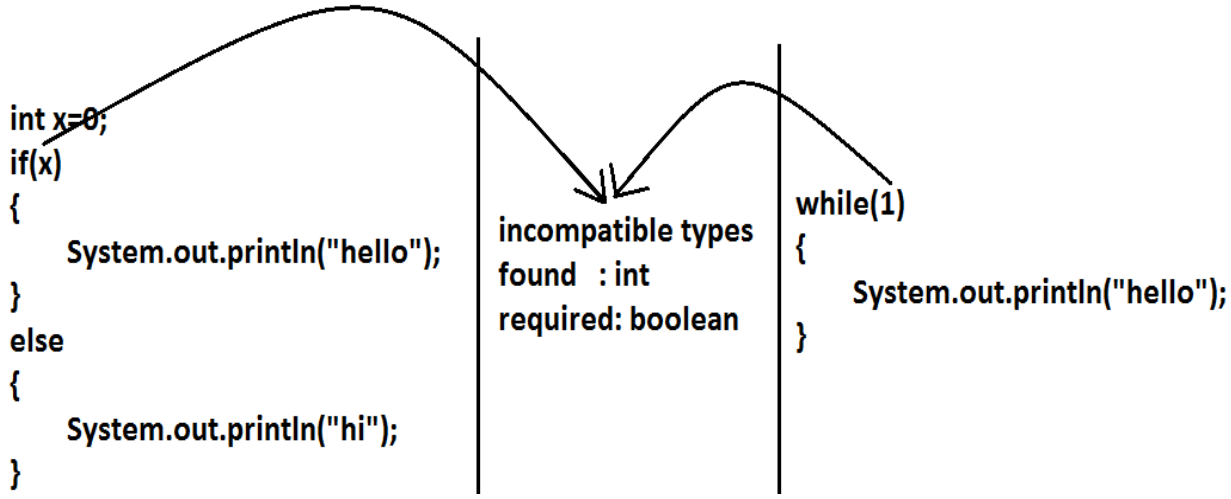
```
double d=10e2;//==>10*10²(valid)

System.out.println(d);//1000.0

float f=10e2;//C.E:possible loss of
precision(invalid)float f=10e2F;(valid)
```

## Boolean literals:

**The only allowed values for the boolean type are true (or) false where case is important.i.e., lower case**

**Example:**

1. **boolean b=true;(valid)**
2. **boolean b=0;//C.E:incompatible types(invalid)**
3. **boolean b=True;//C.E:cannot find symbol(invalid)**
4. **boolean b="true";//C.E:incompatible types(invalid)**




## Char literals:

**1) A char literal can be represented as single character within single quotes.**

**Example:**

1. **char ch='a';(valid)**
2. **char ch=a;//C.E:cannot find symbol(invalid)**
3. **char ch="a";//C.E:incompatible types(invalid)**
4. **char ch='ab';//C.E:unclosed character literal(invalid)**

**2) We can specify a char literal as integral literal which represents Unicode of that character.**
**We can specify that integral literal either in decimal or octal or hexadecimal form butallowed values range is 0 to 65535.**

**Example:**

**char ch=97; (valid)**
char ch=0xFace; (valid) System.out.println(ch); //?

1. **char ch=65536; //C.E: possible loss of precision(invalid)**

**3) We can represent a char literal by Unicode representation which is nothing but'\uxxxx' (4 digit hexa-decimal number) .**

**Example:**

char ch='\ubeef'; 2. char ch1='\u0061';

3. **char ch2=\u0062; //C.E:cannot find symbol**

4. **char ch3='\iface'; //C.E:illegal escape character**
5. **Every escape character in java acts as a char literal.**

**Example:**

```
1) char ch='\n';  //(valid)
2) char ch='\l';  //C.E:illegal escape character(invalid)
```

| Escape Character | Description |
|---|---|
| \n | New line |
| \t | Horizontal tab |
| \r | Carriage return |
| \f | Form feed |

| | |
|---|---|
| \b | Back space character |
| \' | Single quote |
| \" | Double quote |
| \\ | Back space |

**Which of the following char declarations are valid?**

1. **char ch=a; //C.E:cannot find symbol(invalid)**
2. **char ch='ab'; //C.E:unclosed character literal(invalid)**
3. **char ch=65536; //C.E:possible loss of precision(invalid)**
4. **char ch=\uface; //C.E:illegal character: \64206(invalid)**
5. **char ch='/n'; //C.E:unclosed character literal(invalid)**
6. **none of the above. (valid)**

## String literals:

**Any sequence of characters with in double quotes is treated as**

**String literal.Example:**
**String s="Ashok"; (valid)**

## 1.7 Version enhansements with respect to Literals :

**The following 2 are enhansements**

1. **Binary Literals**
2. **Usage of '_' in Numeric Literals**

## Binary Literals :

**For the integral data types untill 1.6v we can specified literal value in the following ways**

1. **Decimal**
2. **Octal**
3. **Hexa decimal**

**But from 1.7v onwards we can specified literal value in binary form also.The allowed digits are 0 to 1.**
**Literal value should be prefixed with Ob or OB .**

```
int x = 0b111;
System.out.pri
ntln(x);  // 7
```

# Diagram: