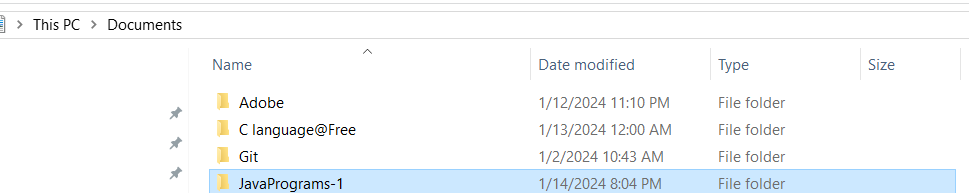
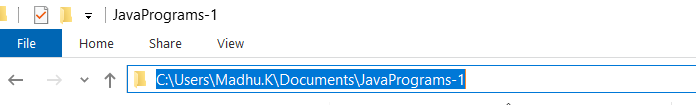
**Steps to write first program**

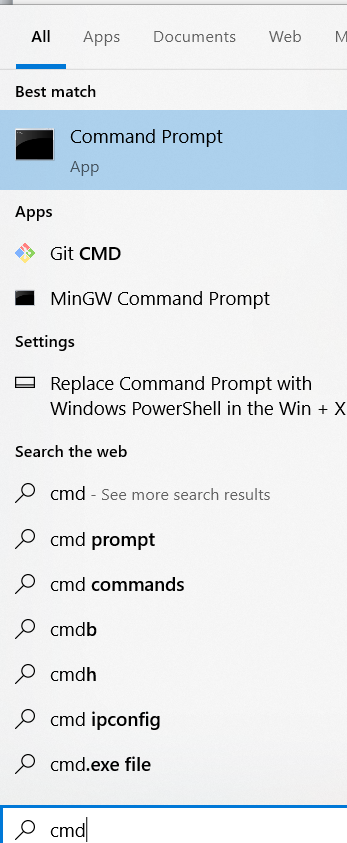
1. Create a separate folder to store (save) the programs.
2. I am creating JavaPrograms-1 folder in documents to store those programs.



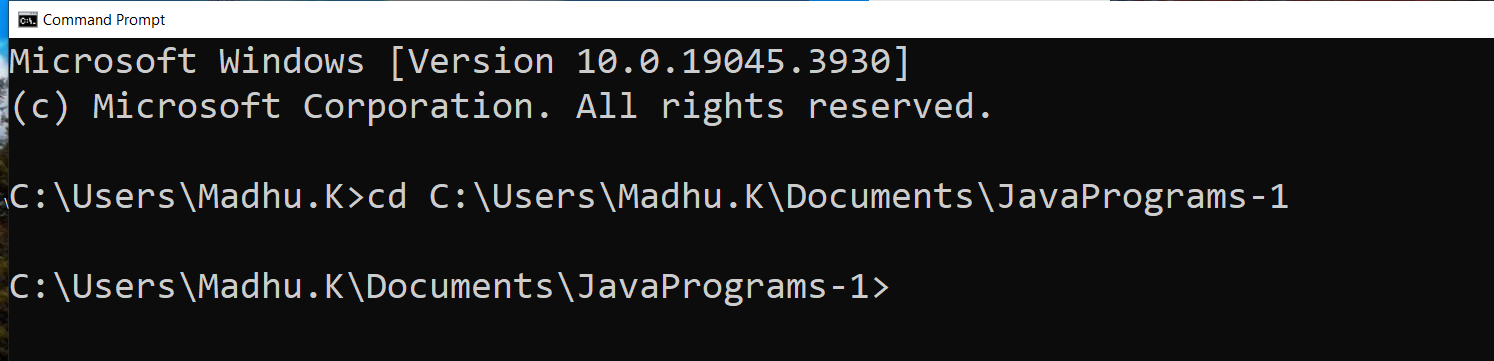
1. Open the folder and copy the path.



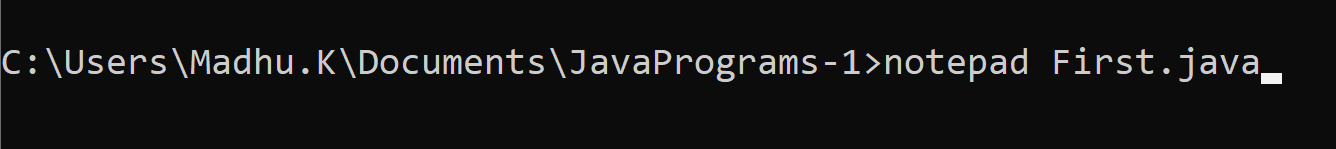
1. Open the command prompt(app).



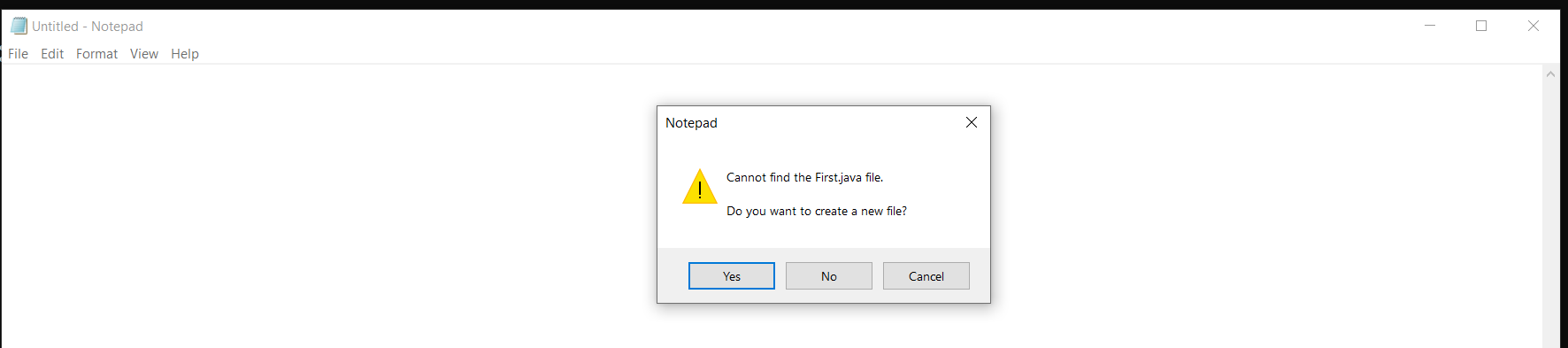
1. Move from user folder to JavaPrograms-1 folder by using (cd(change directory) command). See the below image.



1. Now write a program in notepad (see the below command to open a notepad with a java file name).



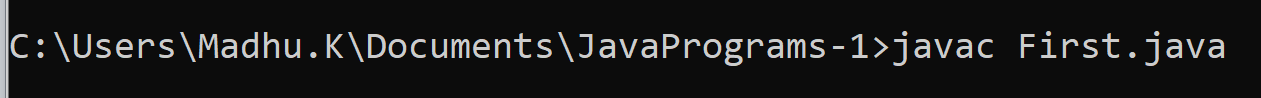
1. Whenever you press the enter notepad asks you about do you want to create new file. Then you have to say yes



1. Write a program in note pad and save it (short cut save is ctrl+s)

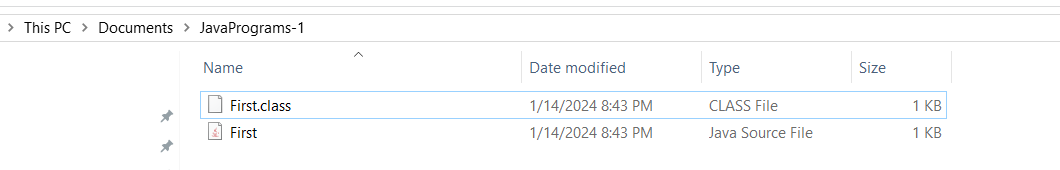
|  |
| --- |
| import java.lang.\*; //import statement  class First  {  public static void main(String args[])  {  System.out.println("Hello World!");  }  } |

1. Compile the program (use tool javac to compile the program). Before compile the code you have to save the written code(ctrl+s in notepad).

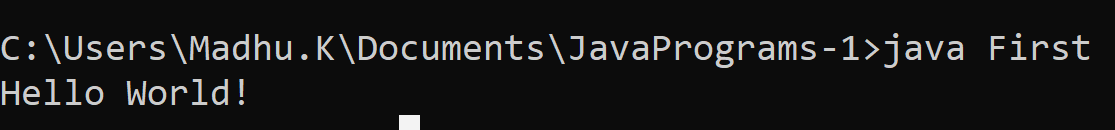


**Note:** when you compile the code, java compiler checks for syntax errors in our program. If it finds any errors it displays them. If no errors are found then compiler translates the source code into byte code and that will be placed in .class file or files.

You can see the First.class file in the below image



1. How to run the byte code.



What is the use of javac tool?

|  |
| --- |
| It is used to invoke and ask the compiler to compile the code/program. |

What is compiler?

|  |
| --- |
| 1. It is a translator, which translates source code into byte code. |

What is byte code?

|  |
| --- |
| 1. It is not a machine code (binary code). 2. It is called as intermediate code which is understandable by a JVM(Java virtual Machine) 3. JVM understandable code |

What is JVM?

|  |
| --- |
| 1. It is a virtual machine installed with JDK (JAVA SOFTWARE), which is used to run java programs. 2. Java programs are executed in JVM |

What is a .class file?

|  |
| --- |
| It is a file which contains byte code. |

Who generates .class file?

|  |
| --- |
| It is generated by a compiler. |

Note: In Java everything you have to write within a class, except import, package statements and comments.

Second.java

|  |
| --- |
| import java.lang.\*;  class Second  {  public static void main(String args[])  {  int a=10,b=20,c=0;  c=a+b;  System.out.println("a="+a);  System.out.println("b="+b);  System.out.println("c="+c);  }  }  Compile:    Run: |

Note: Java is case sensitive

Naming Conventions

|  |  |
| --- | --- |
| Java element | Conventions followed by java inventors |
| Class name, interface name, enum name, annotations names | Each word first letter is capital  Ex: String, System, StringBuffer, DataInputStream |
| Keywords | Single word and small letters  Ex: int,byte,short,public, static, class, void |
| Package | Small letters  Ex: java.lang |
| Variable names and function names | First word is in small letters after that each word first letter capital  Ex: empName, studentNo;, |
|  |  |

**Keywords**

|  |
| --- |
| 1. A keyword is a pre-defined word provided by the Java inventors to us. 2. Every keyword has a special meaning and used for a specific purpose 3. We use keywords to write programs   In Java we have 50 keywords |

**Keywords:**

**What are keyword?**

|  |
| --- |
| 1. Keywords are pre-defined words, which are comes along with java. And provided by inventors of Java. 2. Every keyword has a special meaning and can be used for a specific purpose. 3. For example, to write a for loop we have to use a keyword called for 4. To write condition in a program we have to use if and else keyword 5. We use int keyword to declare a variable   Etc.. |

**List of keywords**

**List of Java keywords**

1. **\_**

**Added in Java 9, the underscore has become a keyword and cannot be used as a variable name anymore.**

**Yes**

**/tmp/Ngz25IYAAC/HelloWorld.java:7: error: as of release 9, '\_' is a keyword, and may not be used as an identifier**

**int \_=100;**

**^**

**ERROR!**

**Note: in jdk 22 version I am not getting error after usage of \_ as an identifier**

public class EscapeSeq

{

    public static void main(String[] args) {

        int \_=1000;

       // System.err.println(\_);

    }

}

Note: we are not getting errors while compilation of this program

1. **abstract**

**A method with no definition must be declared as abstract and the class containing it must be declared as abstract. Abstract classes cannot be instantiated. Abstract methods must be implemented in the sub classes. The abstract keyword cannot be used with variables or constructors. Note that an abstract class isn't required to have an abstract method at all.**

1. **assert (added in J2SE 1.4)[4]**

**Assert describes a predicate (a true–false statement) placed in a Java program to indicate that the developer thinks that the predicate is always true at that place. If an assertion evaluates to false at run-time, an assertion failure results, which typically causes execution to abort. Assertions are disabled at runtime by default, but can be enabled through a command-line option or programmatically through a method on the class loader.**

1. **boolean**

**Defines a boolean variable for the values "true" or "false" only. By default, the value of boolean primitive type is false. This keyword is also used to declare that a method returns a value of the primitive type boolean.**

1. **break**

**Used to end the execution in the current loop body.**

**Used to break out of a switch block.**

1. **byte**

**The byte keyword is used to declare a field that can hold an 8-bit signed two's complement integer.[5][6] This keyword is also used to declare that a method returns a value of the primitive type byte.[7][8]**

1. **case**

**A statement in the switch block can be labeled with one or more case or default labels. The switch statement evaluates its expression, then executes all statements that follow the matching case label; see switch.[9][10]**

1. **catch**

**Used in conjunction with a try block and an optional finally block. The statements in the catch block specify what to do if a specific type of exception is thrown by the try block.**

1. **char**

**Defines a character variable capable of holding any character of the java source file's character set.**

1. **class**

**A type that defines the implementation of a particular kind of object. A class definition defines instance and class fields, methods, and inner classes as well as specifying the interfaces the class implements and the immediate superclass of the class. If the superclass is not explicitly specified, the superclass is implicitly Object. The class keyword can also be used in the form Class.class to get a Class object without needing an instance of that class. For example, String.class can be used instead of doing new String().getClass().**

1. **continue**

**Used to resume program execution at the end of the current loop body. If followed by a label, continue resumes execution at the end of the enclosing labeled loop body.**

1. **default**

**The default keyword can optionally be used in a switch statement to label a block of statements to be executed if no case matches the specified value; see switch.[9][10] Alternatively, the default keyword can also be used to declare default values in a Java annotation. From Java 8 onwards, the default keyword can be used to allow an interface to provide an implementation of a method.**

1. **do**

**The do keyword is used in conjunction with while to create a do-while loop, which executes a block of statements associated with the loop and then tests a boolean expression associated with the while. If the expression evaluates to true, the block is executed again; this continues until the expression evaluates to false.[11][12]**

1. **double**

**The double keyword is used to declare a variable that can hold a 64-bit double precision IEEE 754 floating-point number.[5][6] This keyword is also used to declare that a method returns a value of the primitive type double.[7][8]**

1. **else**

**The else keyword is used in conjunction with if to create an if-else statement, which tests a boolean expression; if the expression evaluates to true, the block of statements associated with the if are evaluated; if it evaluates to false, the block of statements associated with the else are evaluated.[13][14]**

1. **enum (added in J2SE 5.0)[4]**

**A Java keyword used to declare an enumerated type. Enumerations extend the base class Enum.**

1. **extends**

**Used in a class declaration to specify the superclass; used in an interface declaration to specify one or more superinterfaces. Class X extends class Y to add functionality, either by adding fields or methods to class Y, or by overriding methods of class Y. An interface Z extends one or more interfaces by adding methods. Class X is said to be a subclass of class Y; Interface Z is said to be a subinterface of the interfaces it extends.**

**Also used to specify an upper bound on a type parameter in Generics.**

1. **final**

**Define an entity once that cannot be changed nor derived from later. More specifically: a final class cannot be subclassed, a final method cannot be overridden, and a final variable can occur at most once as a left-hand expression on an executed command. All methods in a final class are implicitly final.**

1. **finally**

**Used to define a block of statements for a block defined previously by the try keyword. The finally block is executed after execution exits the try block and any associated catch clauses regardless of whether an exception was thrown or caught, or execution left method in the middle of the try or catch blocks using the return keyword.**

1. **float**

**The float keyword is used to declare a variable that can hold a 32-bit single precision IEEE 754 floating-point number.[5][6] This keyword is also used to declare that a method returns a value of the primitive type float.[7][8]**

1. **for**

**The for keyword is used to create a for loop, which specifies a variable initialization, a boolean expression, and an incrementation. The variable initialization is performed first, and then the boolean expression is evaluated. If the expression evaluates to true, the block of statements associated with the loop are executed, and then the incrementation is performed. The boolean expression is then evaluated again; this continues until the expression evaluates to false.[15]**

**As of J2SE 5.0, the for keyword can also be used to create a so-called "enhanced for loop",[16] which specifies an array or Iterable object; each iteration of the loop executes the associated block of statements using a different element in the array or Iterable.[15]**

1. **if**

**The if keyword is used to create an if statement, which tests a boolean expression; if the expression evaluates to true, the block of statements associated with the if statement is executed. This keyword can also be used to create an if-else statement; see else.[13][14]**

1. **implements**

**Included in a class declaration to specify one or more interfaces that are implemented by the current class. A class inherits the types and abstract methods declared by the interfaces.**

1. **import**

**Used at the beginning of a source file to specify classes or entire Java packages to be referred to later without including their package names in the reference. Since J2SE 5.0, import statements can import static members of a class.**

1. **instanceof**

**A binary operator that takes an object reference as its first operand and a class or interface as its second operand and produces a boolean result. The instanceof operator evaluates to true if and only if the runtime type of the object is assignment compatible with the class or interface.**

1. **int**

**The int keyword is used to declare a variable that can hold a 32-bit signed two's complement integer.[5][6] This keyword is also used to declare that a method returns a value of the primitive type int.[7][8]**

1. **interface**

**Used to declare a special type of class that only contains abstract or default methods, constant (static final) fields and static interfaces. It can later be implemented by classes that declare the interface with the implements keyword. As multiple inheritance is not allowed in Java, interfaces are used to circumvent it. An interface can be defined within another interface.**

1. **long**

**The long keyword is used to declare a variable that can hold a 64-bit signed two's complement integer.[5][6] This keyword is also used to declare that a method returns a value of the primitive type long.[7][8]**

1. **native**

**Used in method declarations to specify that the method is not implemented in the same Java source file, but rather in another language.[8]**

1. **new**

**Used to create an instance of a class or array object. Using keyword for this end is not completely necessary (as exemplified by Scala), though it serves two purposes: it enables the existence of different namespace for methods and class names, it defines statically and locally that a fresh object is indeed created, and of what runtime type it is (arguably introducing dependency into the code).**

1. **package**

**Java package is a group of similar classes and interfaces. Packages are declared with the package keyword.**

1. **private**

**The private keyword is used in the declaration of a method, field, or inner class; private members can only be accessed by other members of their own class.[17]**

1. **protected**

**The protected keyword is used in the declaration of a method, field, or inner class; protected members can only be accessed by members of their own class, that class's subclasses or classes from the same package.[17]**

1. **public**

**The public keyword is used in the declaration of a class, method, or field; public classes, methods, and fields can be accessed by the members of any class.[17]**

1. **return**

**Used to finish the execution of a method. It can be followed by a value required by the method definition that is returned to the caller**

1. **short**

**The short keyword is used to declare a field that can hold a 16-bit signed two's complement integer.[5][6] This keyword is also used to declare that a method returns a value of the primitive type short.[7][8]**

1. **static**

**Used to declare a field, method, or inner class as a class field. Classes maintain one copy of class fields regardless of how many instances exist of that class. static also is used to define a method as a class method. Class methods are bound to the class instead of to a specific instance, and can only operate on class fields. Classes and interfaces declared as static members of another class or interface are actually top-level classes and are not inner classes.**

1. **super**

**Inheritance basically used to achieve dynamic binding or run-time polymorphism in java. Used to access members of a class inherited by the class in which it appears. Allows a subclass to access overridden methods and hidden members of its superclass. The super keyword is also used to forward a call from a constructor to a constructor in the superclass.**

**Also used to specify a lower bound on a type parameter in Generics.**

1. **switch**

**The switch keyword is used in conjunction with case and default to create a switch statement, which evaluates a variable, matches its value to a specific case, and executes the block of statements associated with that case. If no case matches the value, the optional block labelled by default is executed, if included.[9][10]**

1. **synchronized**

**Used in the declaration of a method or code block to acquire the mutex lock for an object while the current thread executes the code.[8] For static methods, the object locked is the class's Class. Guarantees that at most one thread at a time operating on the same object executes that code. The mutex lock is automatically released when execution exits the synchronized code. Fields, classes and interfaces cannot be declared as synchronized.**

1. **this**

**Used to represent an instance of the class in which it appears. this can be used to access class members and as a reference to the current instance. The this keyword is also used to forward a call from one constructor in a class to another constructor in the same class.**

1. **throw**

**Causes the declared exception instance to be thrown. This causes execution to continue with the first enclosing exception handler declared by the catch keyword to handle an assignment compatible exception type. If no such exception handler is found in the current method, then the method returns and the process is repeated in the calling method. If no exception handler is found in any method call on the stack, then the exception is passed to the thread's uncaught exception handler.**

1. **throws**

**Used in method declarations to specify which exceptions are not handled within the method but rather passed to the next higher level of the program. All uncaught exceptions in a method that are not instances of RuntimeException must be declared using the throws keyword.**

1. **transient**

**Declares that an instance field is not part of the default serialized form of an object. When an object is serialized, only the values of its non-transient instance fields are included in the default serial representation. When an object is deserialized, transient fields are initialized only to their default value. If the default form is not used, e.g. when a serialPersistentFields table is declared in the class hierarchy, all transient keywords are ignored.[18][19]**

1. **try**

**Defines a block of statements that have exception handling. If an exception is thrown inside the try block, an optional catch block can handle declared exception types. Also, an optional finally block can be declared that will be executed when execution exits the try block and catch clauses, regardless of whether an exception is thrown or not. A try block must have at least one catch clause or a finally block.**

1. **void**

**The void keyword is used to declare that a method does not return any value.[7]**

1. **volatile**

**Used in field declarations to guarantee visibility of changes to variables across threads. Every read of a volatile variable will be read from main memory, and not from the CPU cache, and that every write to a volatile variable will be written to main memory, and not just to the CPU cache.[20] Methods, classes and interfaces thus cannot be declared volatile, nor can local variables or parameters.**

1. **while**

**The while keyword is used to create a while loop, which tests a boolean expression and executes the block of statements associated with the loop if the expression evaluates to true; this continues until the expression evaluates to false. This keyword can also be used to create a do-while loop; see do.**

**Not used keywords**

1. **const**
2. **goto**
3. **strictfp**

**Conclusion:**

|  |
| --- |
| Now in jdk22 version we are haing only 50 keywords because of \_ is not using as a keword in jdk22 version. |

**Reserved Identifiers or contextual keywords**

**The following identifiers are contextual keywords, and are only restricted in some contexts:**

1. exports
2. module
3. non-sealed
4. open
5. opens
6. permits
7. provides
8. record
9. requires
10. sealed
11. to
12. transitive
13. uses
14. var
15. when
16. with
17. yield

Example on contextual keyword var

|  |
| --- |
| public class Sixteen  {      public static void main(String[] args)      {          var a=100;          var b=200.50f;          var c=200.00;          var name="madhu";          System.out.println("a:\t"+a);          System.out.println("b:\t"+b);          System.out.println("c:\t"+c);          System.out.println("name:\t"+name);      }  }  Output:  a: 100  b: 200.5  c: 200.0  name: madhu |

Example on usage of contextual keyword as an identifier

|  |
| --- |
| public class Sixteen  {      public static void main(String[] args)      {          int var=100; //we are using contextual keyword as an identifier          System.out.println("var:\t"+var);      }  }  **Output:**  var: 100 |

**Identifier**

|  |
| --- |
| 1. Identifiers are names given to functions, classes, memory locations(variables) etc…   Rules to follow at the time of writing an identifier   1. We can use alphabets, digits, under score(\_) and dollar sign 2. It should not start with a digit 3. We can’t use space in an identifier 4. We can’t use keywords as identifiers   Example:  class First  {  public static void main(String args[])  {  int \_eno=1;  System.out.print(\_eno);  float $sal=200000.00f;  System.out.print($sal);  //int 1e=100; error  //String emp name="Madhu.K"; error  //int float=20.300f; error  }  } |

What is a variable?

|  |
| --- |
| 1. Variable is changeable 2. Variable is a container where we can store a value |

Note: in any language you should give space after keyword.

How to declare a variable?

|  |
| --- |
| In Java we can declare a variable by using a data type or by using a contextual keyword called var.  Syntax: <data-type or var> <var-name>;  Example: int n; //variable declaration |

Why we declare a variable?

|  |
| --- |
| To allocate memory in RAM.  If the statement “int n;” executes then 4 bytes memory will be allocated in RAM. |

What is the use of data type?

|  |
| --- |
| 1. It is used to declare variables 2. By using data type we can tell the JVM how much memory should allocated and what type of data should be stored in that memory. |

Example on variable declaration and initialization

|  |
| --- |
| class First  {  public static void main(String args[])  {  int n; //variable declaration  //int: keyword as well as data type  //n is a name given to memory location  //; is separator    n=100; //variable initialization  System.out.println(n);  n=1000; //assigning a value    }  } |

Data types

What is a data type?

|  |
| --- |
| A data type is a keyword or class name or interface name or enum name or annotation name or combination of any data type plus one or more subscripts ,which is used to tell the JVM that how much memory should be allocated for a variable and what type of value that variable can store.  (or)  In Java, a **data type** specifies the size of the memory which is allocated for variable and also tells that what type of value a variable can store. |

|  |
| --- |
| 1. Data types are used to declare variables 2. By using data-type we can tell the JVM how much memory should allocated and what type of data should be stored in that memory.   **There are 2 types of data types available in Java**   1. Primary (primitive) data types 2. Reference (object) data types (or) non-primitive data types |

Primary data types

|  |  |  |
| --- | --- | --- |
| **Data type** | **Size** | **Default value** |
| Byte | 1 byte | 0 |
| Short | 2 bytes | 0 |
| Int | 4 bytes | 0 |
| Long | 8 bytes | 0 |
|  |  |  |
| Float | 4 bytes | 0.0f |
| Double | 8 bytes | 0.0 |
|  |  |  |
| Char | 2 bytes | ‘\u0000’ |
|  |  |  |
| Boolean | 1 bit | False |

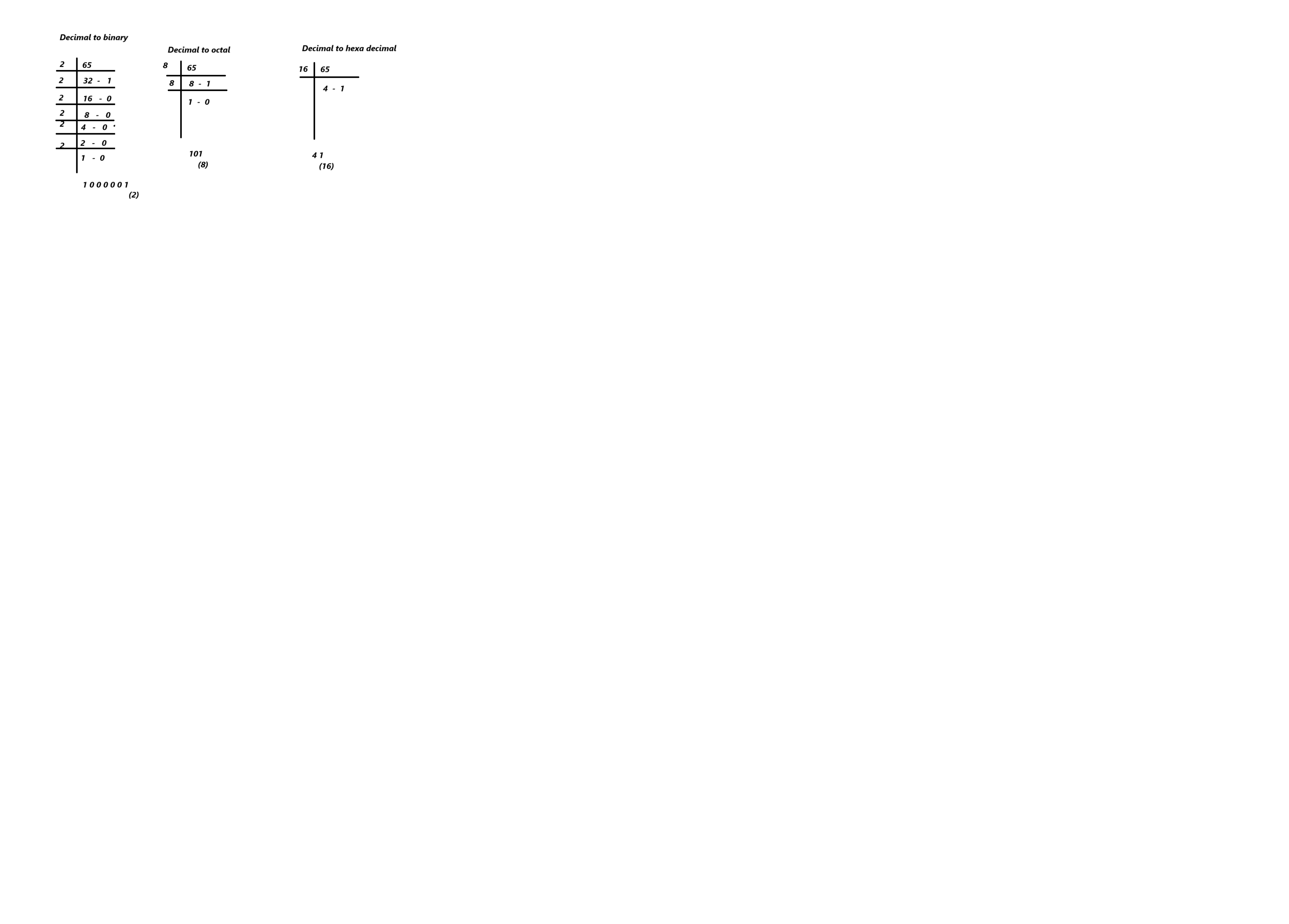
Primitive data types are divided into 4 categories

|  |
| --- |
| 1. Integer or integral data types: by using these data types if we declare a variable in that we can store only int values 2. Floating point data types: by using these data types if we declare a variable in that we can store only float values 3. Character data types: by using these data types if we declare a variable in that we can store only a character 4. Boolean data types: by using these data types if we declare a variable in that we can store only boolean values (either true or false) |

**Literals**

|  |
| --- |
| The values(constants) used in out program is called as a literal  Types of literals supported by Java   1. Integer literals: Ex: 100, -100, 100l, 100L 2. Floating point literals: Ex: 10.50f, -10.50f,10.50F, 10.00, 10.00d,10.00D, 3. Character literals: Ex: ‘A’, ‘9’, ‘\u0041’ 4. String literals: “sambar” 5. Special literal: null 6. Boolean literals (true or false) |

**How to convert decimal to binary , decimal to octal and decimal to hexa decimal**



Char ch1=’\u1947’ ch1 contains ‘᥇’

|  |
| --- |
| class Second{      public static void main(String args[])      {          char ch1='\u1947';          System.out.printf("%c",ch1);      }  }  Output:  ? //jvm is not able to display the ‘᥇’ character |

Example on literals

|  |
| --- |
| class Second{      public static void main(String args[])      {          //literals demo          byte b=100;          short s=100;          int i=(int)100.45;          long l=100;          float f=(float)10.00;          double d1=100.00;          double d2=100.00d;          double d3=100.00D;          char ch1='a';          char ch2=97;          char ch3='\u0061';          System.err.printf("ch1=%c\n",ch1);          System.err.printf("ch2=%c\n",ch2);          System.err.printf("ch3=%c\n",ch3);          boolean b1=true;          boolean b2=false;          System.out.printf("b1=%s\n",b1);          System.out.printf("b2=%s\n",b2);      }  }  Output:  ch1=a  ch2=a  ch3=a  b1=true  b2=false |

**Value type variables and reference variables**

**What is a value type variable?**

|  |
| --- |
| 1. It is a variable where we can store a value not address. 2. We can declare a value type variable by using primitive data types   Ex: byte b=100; // defining a variable (declaration and initialization in single line)  Ex: char ch1=’B’;  Ex: int a=-100; |

What is a reference variable?

|  |
| --- |
| 1. It is a variable where we can store either address or null 2. We can declare a reference variable by using reference data type |

Reference data type

|  |
| --- |
| 1. It is a data type which Is used to declare a reference variable   We can use the following as reference data types   1. Class name 2. Interface name 3. Enum name 4. Annotation name 5. Combination of Any data type with one or more subscripts   Ex: String s1=”madhu”; //now s contains address  Ex: Second s=null;  Ex: int[] arr;  Example on how to declare a reference variable  class Second{      public static void main(String args[])      {          String s1=null;          Second s2=null;          System.out.println("s1:\t"+s1);          System.out.println("s2:\t"+s2);      }  }  Output:  s1: null  s2: null |

Another example on reference variables declaration

|  |
| --- |
| class Second{      public static void main(String args[])      {          String s1=null;          Second s2=null;          int[] arr1=null;          Second[][] arr2=null;          System.out.println("s1:\t"+s1);          System.out.println("s2:\t"+s2);          System.out.println("arr1:\t"+arr1);          System.out.println("arr2:\t"+arr2);      }  }  Output:  s1: null  s2: null  arr1: null  arr2: null |

What is a string?

|  |
| --- |
| A collection of characters represented by pair of double quotations  Ex: “hello”  Ex: “123456” |

Example on value type and reference variables

|  |
| --- |
| public class Sixteen  {      public static void main(String[] args)      {          int var=100; //we are using contextual keyword as an identifier          System.out.println("var:\t"+var);          float f=10.40f;          System.out.println("f:\t"+f);          String name="Madhu.K";          System.out.println("name:\t"+name);      }  }  **Output:**  *var: 100*  *f: 10.4*  *name: Madhu.K* |

**Concatenation**

|  |
| --- |
| In java by using concatenation operator we can append anything to the string  Ex: “madhu”+420 => “madhu420”  Ex: “madhu”+”babu” 🡺 “madhubabu” |

|  |
| --- |
| class Second{      public static void main(String[] args)      {          int a=100,b=20;          String s1="a="+a;  //after concatenation we will get string as a result          //String s1="a=100"          System.out.println(s1);          String s2="tokkaley="+b;          System.out.println(s2);          int c=a+b;          System.out.println(  "orey anniyya resulteyntantey...."+120);      }  }  **Output:**  a=100  tokkaley=20  orey anniyya resulteyntantey....120 |

**Another example on concatenation operator usage**

|  |
| --- |
| class Third{  public static void main(String[] args)  { int a=10,b=5;  int c=a+b;  System.out.println(a+"+"+b+"="+c);  }  }  **Output:** |

Example on Byte range

|  |
| --- |
| public class Fourth  {  public static void main(String[] args)  {  byte b=128;  System.out.println("b:\t"+b);  }  }  /\*  1. What is the result?  a. compile time error  b. -128  c. 128  d. b: 128  Ans) a    \*/ |

Another example on byte range

|  |
| --- |
| public class Fourth  {  public static void main(String[] args)  {  byte b=(byte)128;  System.out.println("b:\t"+b);  }  }  /\*  1. What is the result?  a. compile time error  b. b: -128  c. 128  d. b: 128  Ans) b  \*/ |

Another example on byte operations

|  |
| --- |
| public class Fourth  {  public static void main(String[] args)  {  byte b1=10;  byte b2=20;  byte b3=b1+b2;  System.out.println("b3:\t"+b3);  }  }  /\*  1. What is the result?  a. compile time error  b. b3: 30  c. runtime error  d. none of the above  Ans) A  \*/  Output: |

|  |  |  |
| --- | --- | --- |
| Type | Resultant will be stored in the memory of | description |
| byte + byte | Int |  |
| byte+short | Int |  |
| Short+byte | Int |  |
| Short+short | Int |  |
| Int+int | Int |  |
| Int+byte | Int |  |
| Int+short | Int |  |
| Long+byte | Long |  |
| Long+short | Long |  |
| Long+int | Long |  |
| Long+long | Long |  |
| Float+long | Float |  |
| Float+int | Float |  |
| Float+byte | Float |  |
| Float+float | Float |  |
| Double+any\_type\_of\_value | Double |  |

What is the range of byte type variable?

|  |
| --- |
| -128 to 127 |

What is the range of short type variable?

|  |
| --- |
| -32768 to 32767 |

How many public classes we can write in a program?

|  |
| --- |
| 1. We can write only one public class in a program 2. If you have written public class then the class name and file name should be same it is rule. |

Example (where the public class name and file name is different)

Five.java

|  |
| --- |
| public class Demo  { public static void main(String args[])  { System.out.println("welcome..");  }  } |

Example Without public class

|  |
| --- |
| class Demo  {  public static void main(String args[])  {  System.out.println("welcome..");  }  }  Output: |

Back slash codes or escape sequences or escape characters

|  |
| --- |
| We can use backslash codes only within a string literal  Ex: |

Example on escape characters

|  |
| --- |
| class Demo  {  public static void main(String args[])  {  System.out.print("MadhuTechSkills\nVijayawada\nNTR District\n");  System.out.print("Madhutechtttttt\tVijayawada\tNTR District\n");  System.out.print("\nABC\b\bxyz\n");  System.out.print("\nABC\rxyzijk\n");  System.out.print("\nABC\rx\n");  System.out.println("\"Yes\" Bank");  System.out.println("'Yes' Bank");  System.out.println("\'Yes\' Bank");  System.out.println("D:\trainings\Nanda\badri\raamu");  System.out.println("D:\\trainings\\nanda\\badri\\raamu");  System.out.printf("%-25s%-25s%-25s%n","Madhutechtttttt","Vijayawada","NTR District");  System.out.format("%-25s%-25s%-25s","Madhutechtttttt","Vijayawada","NTR District");  }  }  Output:  MadhuTechSkills  Vijayawada  NTR District  Madhutechtttttt Vijayawada NTR District  Axyz  xyzijk  xBC  "Yes" Bank  'Yes' Bank  'Yes' Bank  D: rainings  aamudri  D:\trainings\nanda\badri\raamu  Madhutechtttttt Vijayawada NTR District  Madhutechtttttt Vijayawada NTR District |