
JAVA

Step 1

What Is Java?

- Java is a popular programming language.
- Java is a high-level programming language
- It is owned by Oracle, and more than 3 billion devices run Java.
- It was originally designed for developing programs for set-top boxes and handheld devices, but later became a popular choice for creating web applications.
- The Java syntax is similar to C++, but is strictly an object-oriented programming language.

Step 2

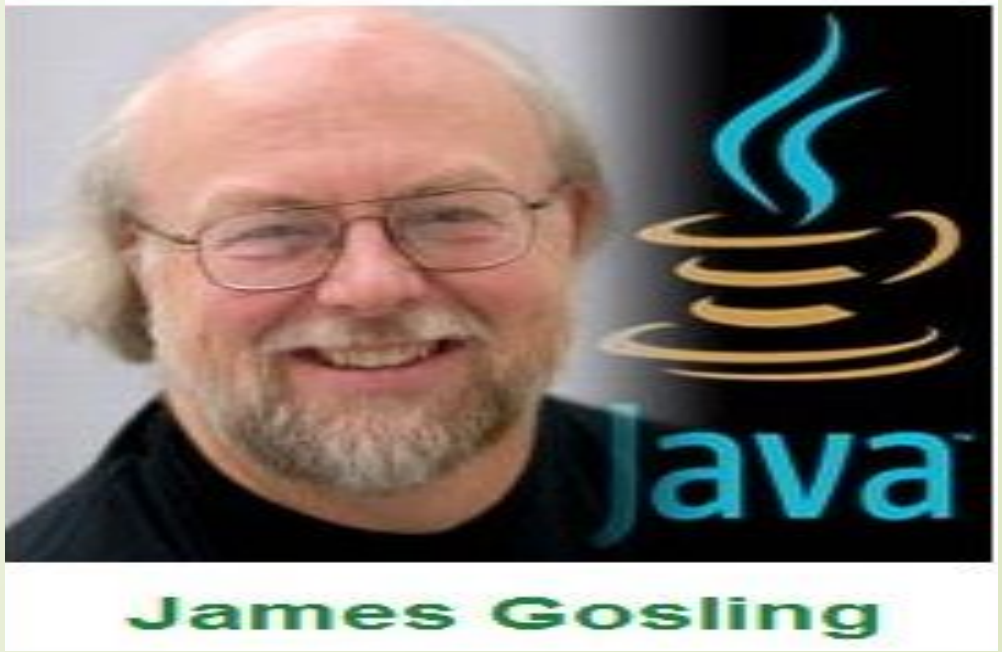
Why Use Java ?

- Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
- It is one of the most popular programming language in the world
- It is easy to learn and simple to uses
- It is open-source and free
- It is secure, fast and powerful
- Java is an object oriented language which gives a clear structure to programs and allows code to be reused
- As Java is close to C++ and C

History of Java

Step 3

- ✔ Java programming language was developed by Sun Microsystems
- ✔ It was initiated by James Gosling in May 23 1995
- ✔ Latest Java Version is Java SE 13.0.1 in Sep 17 2019



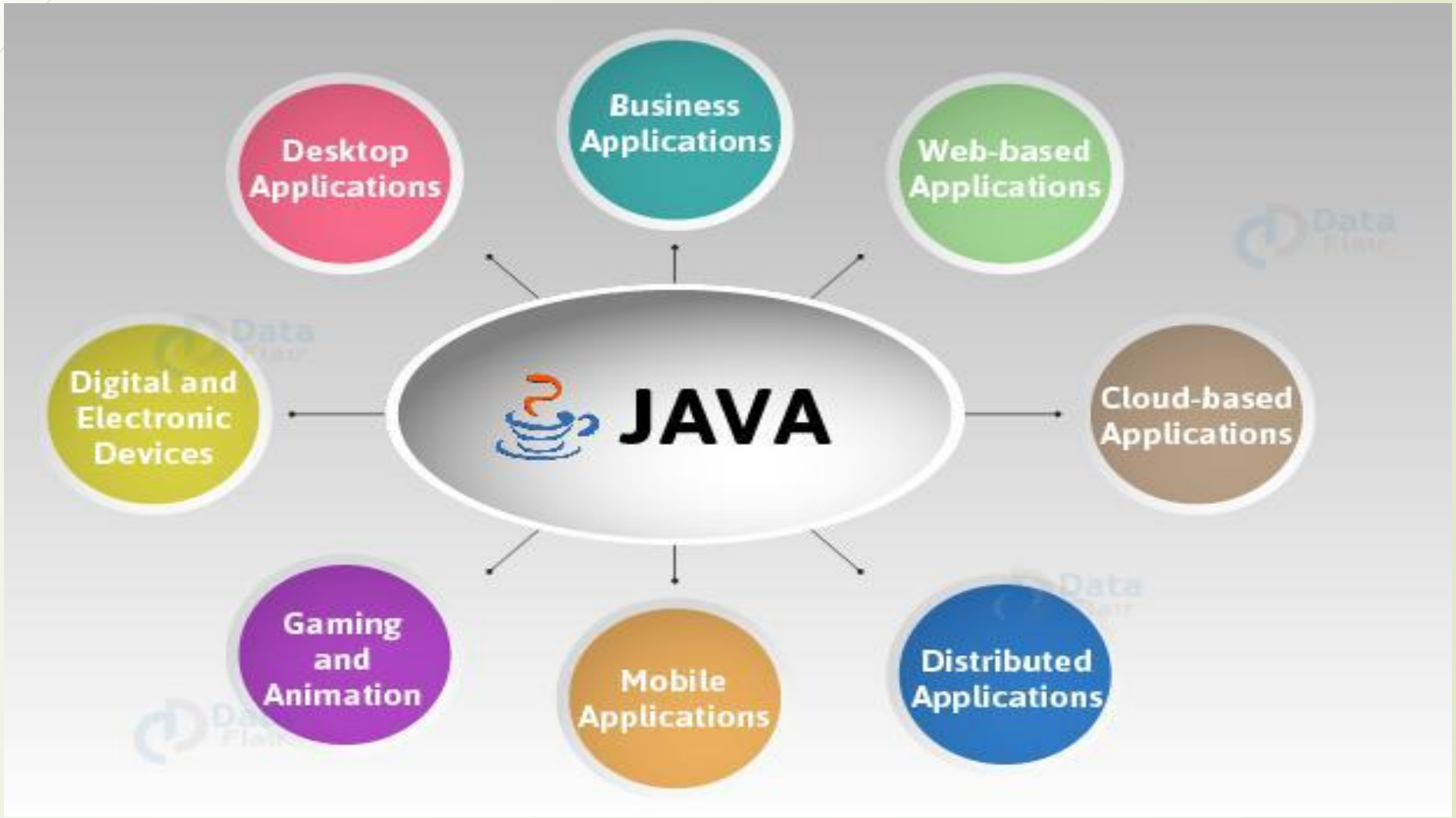
Step 4

Intro to Java

- A platform is an environment that helps to develop and run programs written in any programming language.
- Java is fast, reliable and secure.

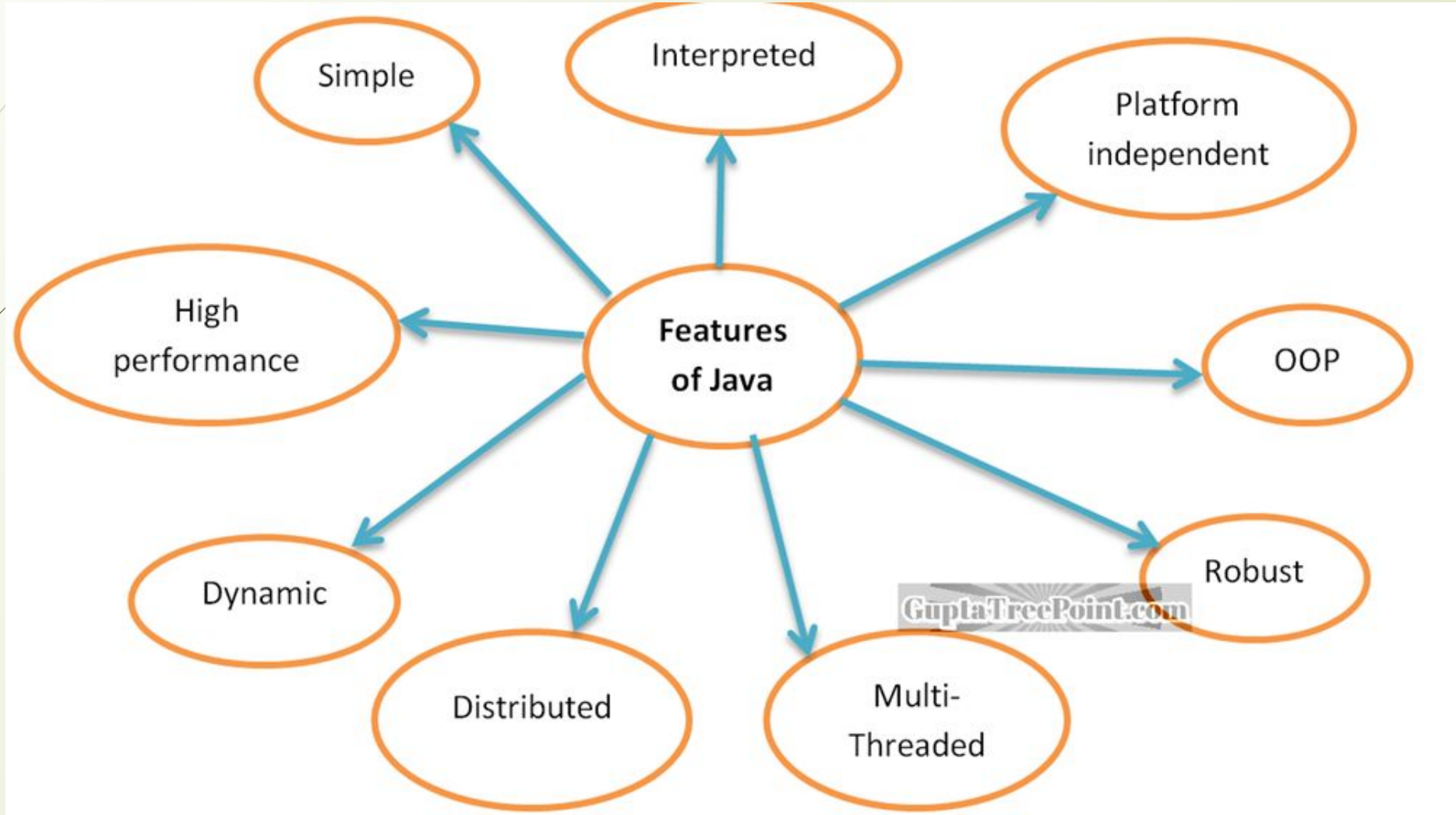
Types of Java Application

Step 5



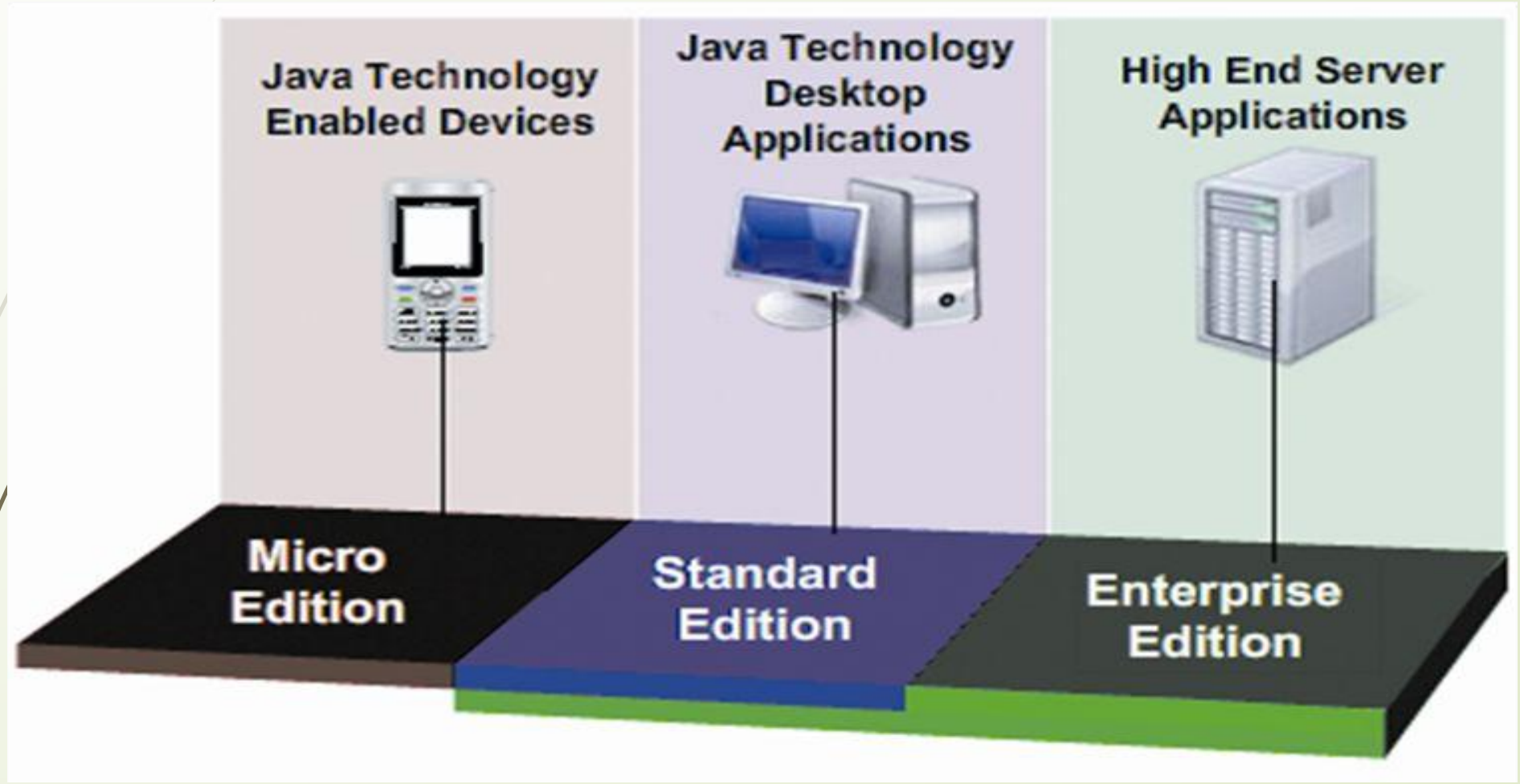
Step 6

Features of Java



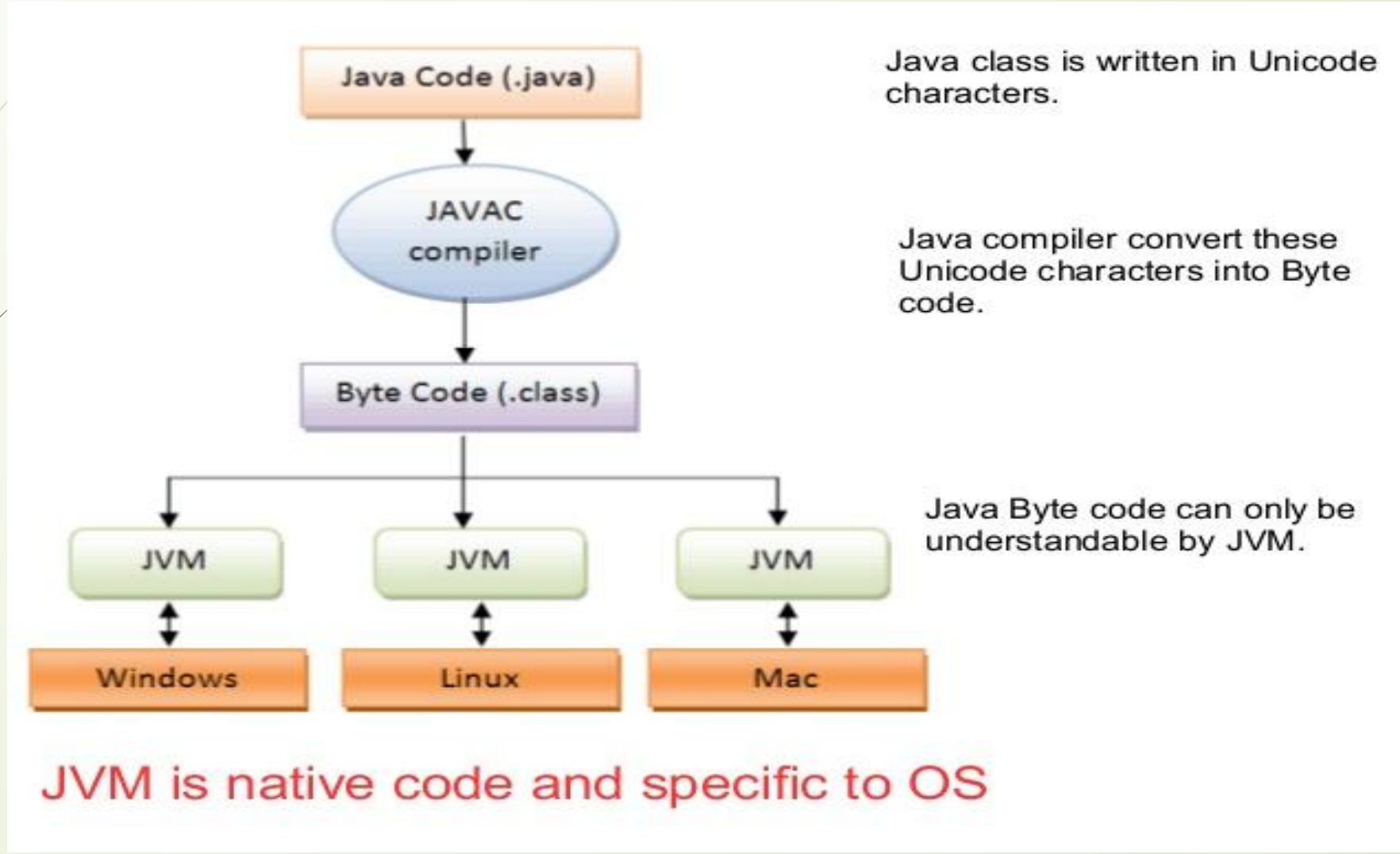
Step 7

Java Platform/Editions



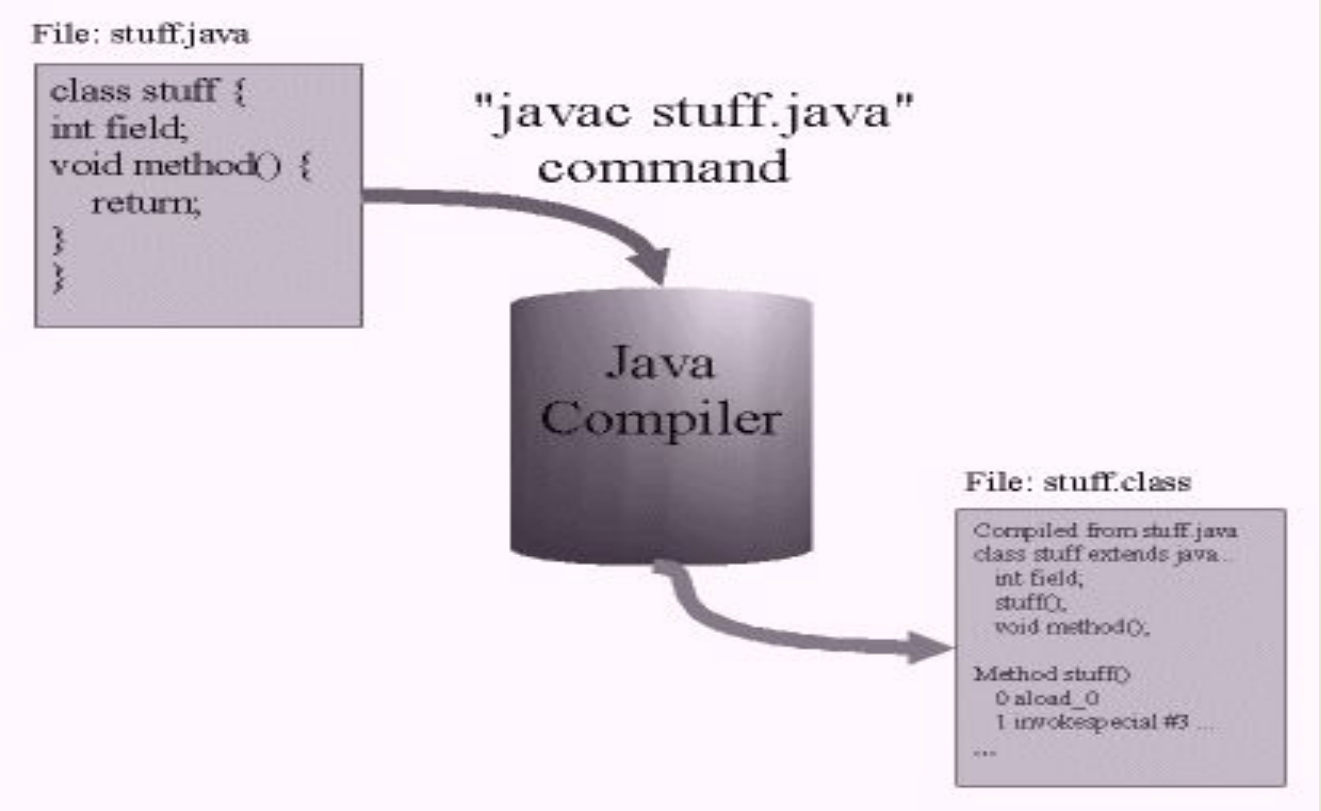
Step 8

Architecture Diagram for Java



Step 9

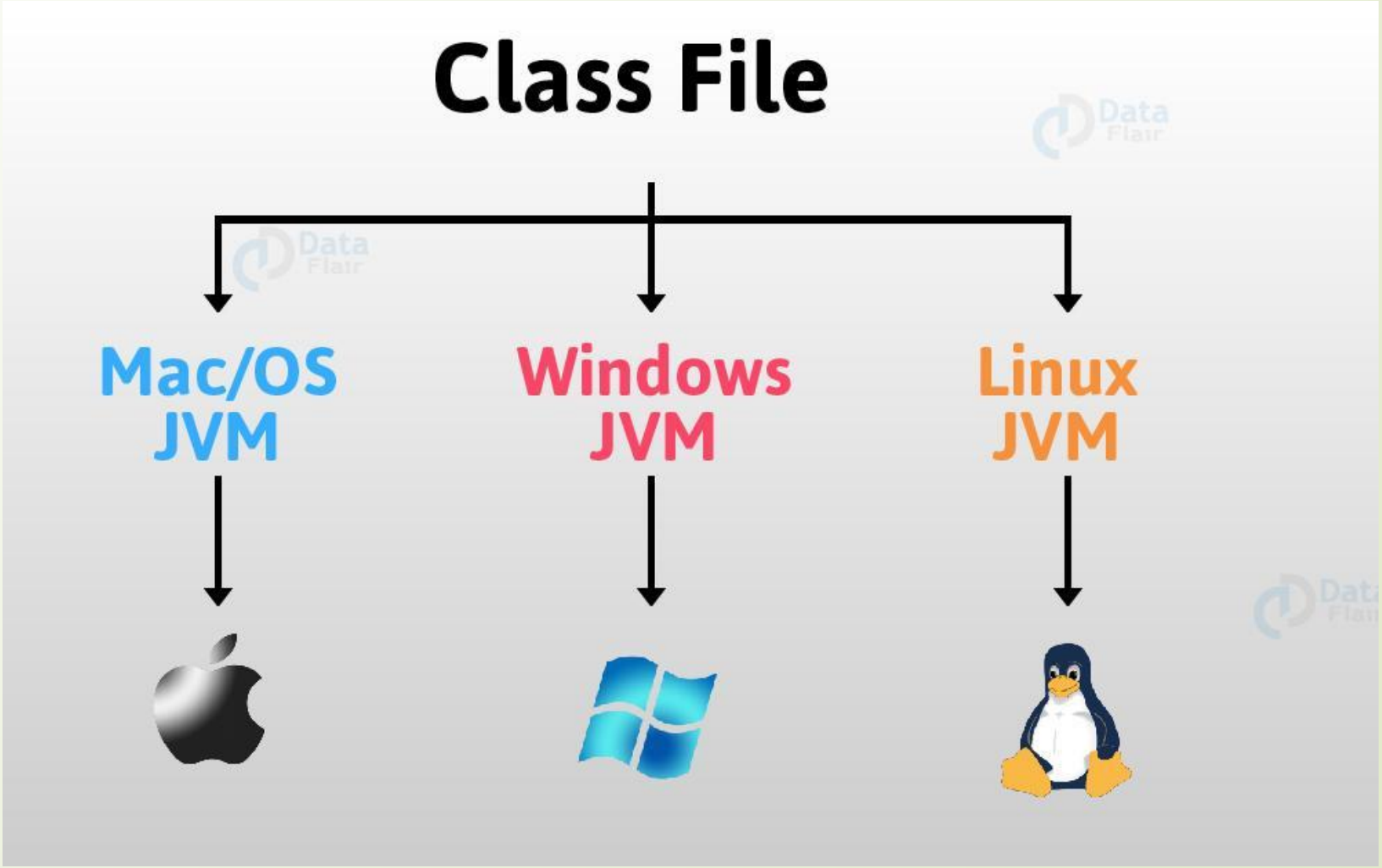
What happens at compile time?



Java file is compiled by Java Compiler (It does not interact with OS) and converts the java code into byte code.

Step 10

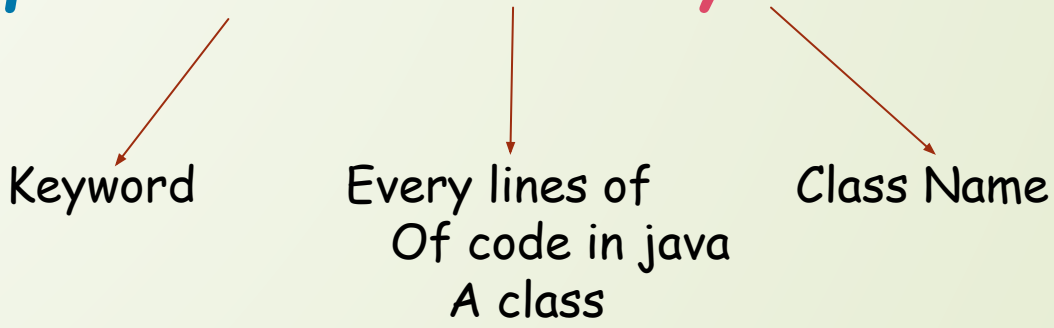
Why Platform Independent?



Step 11

Java Common Syntax

public class My Class



Step 12

The main() Method

It is common to all Java Program

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

TO PRINT

```
System.out.println("Hello World");
```

Step 13

Java Comments

Single-line comments start with two forward slashes (//)

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

Any text between // and /* the end of the line is ignored by Java (will not be executed).

Step 14

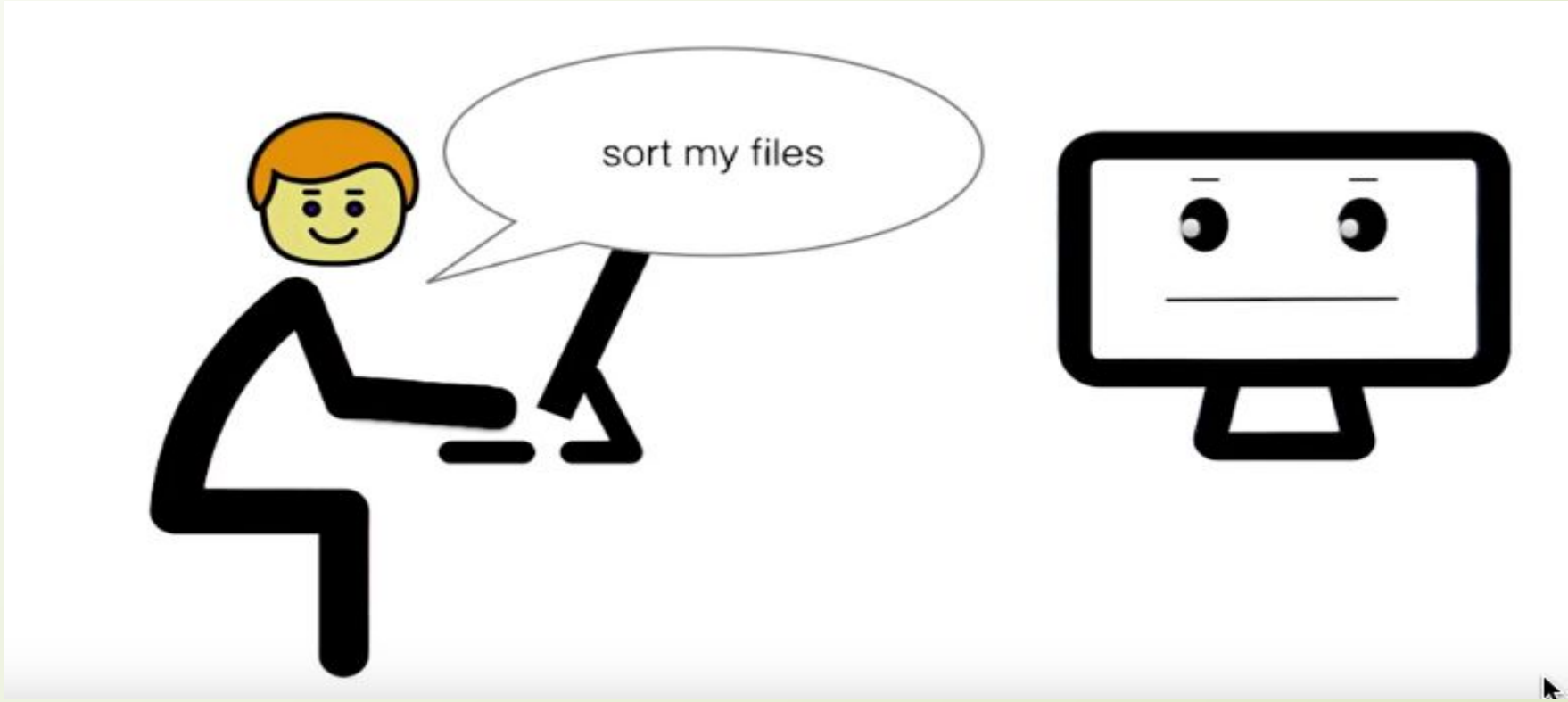
Java

Variables and Data Types

Step 15

Programming a computer

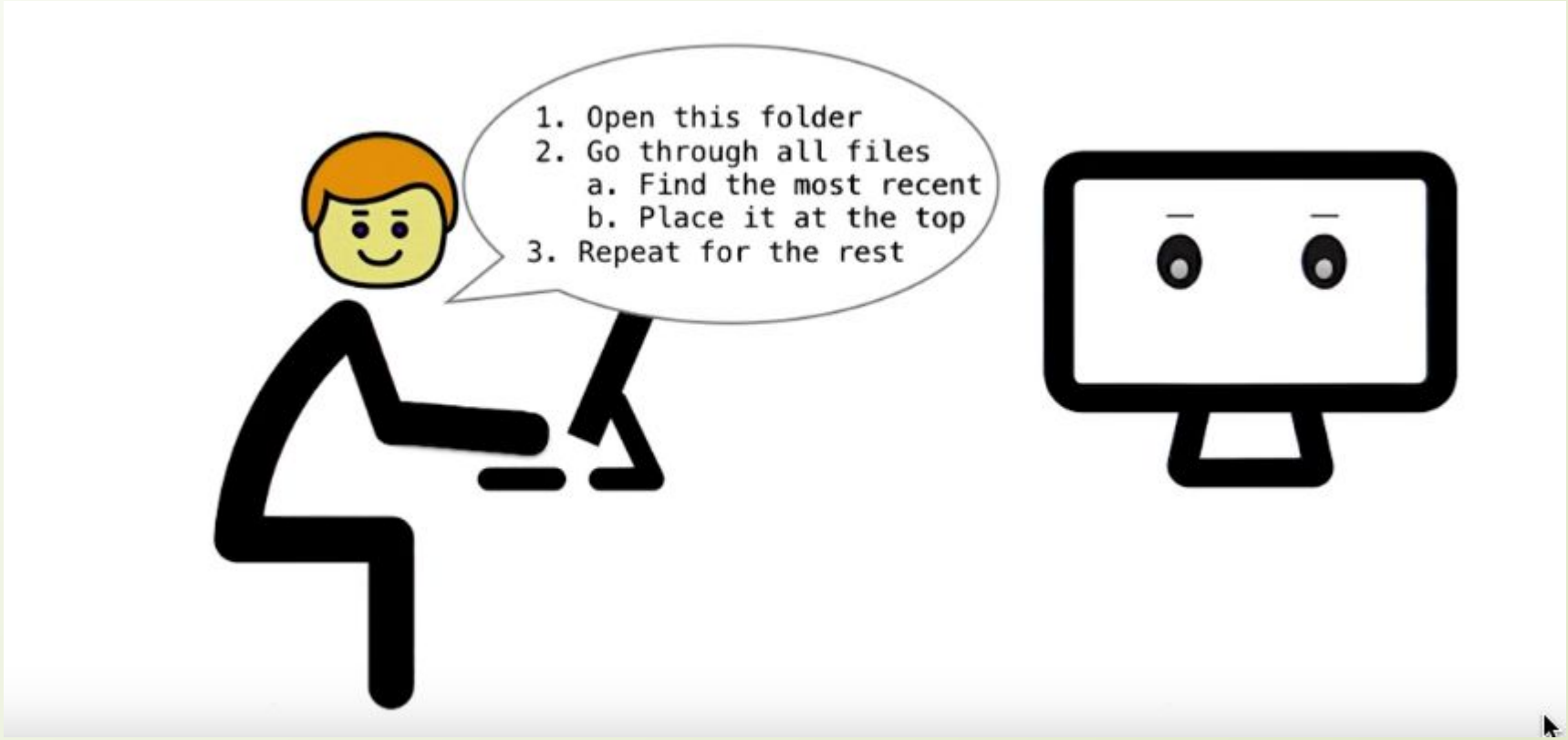
Computers are really good at following instructions, but those instructions have to be precise and detailed.



Step 16

Programming a computer

A more precise list of instructions should tell the computer step by step exactly what to do



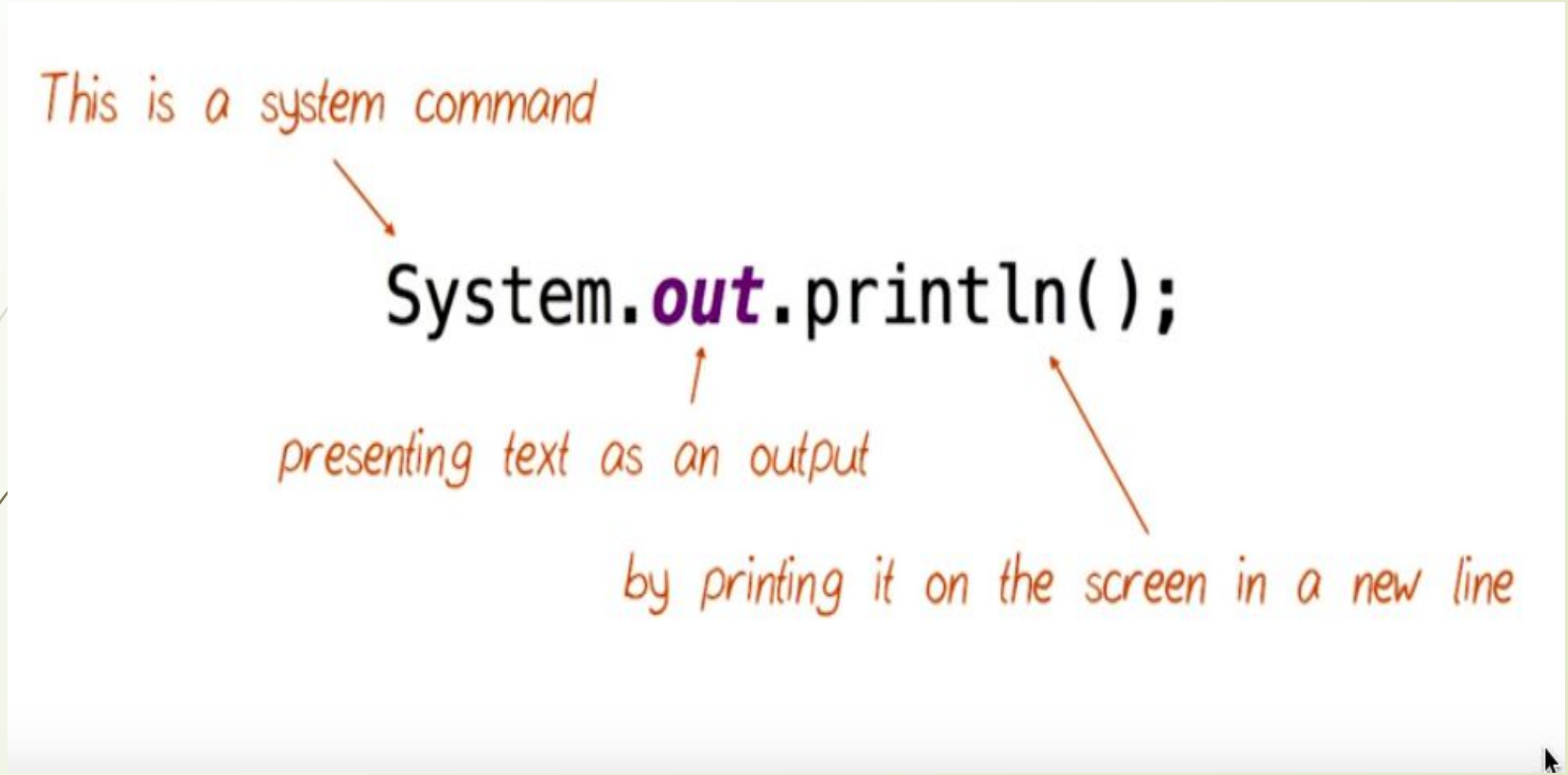
Step 17

Hello World

- Java is a computer language that uses basic English Words but in a specific way that computers can understand.
- It offers a huge library of instructions and commands that can use just straight away like

System.out.println();

Step 18



System.out.println("Welcome to Praziksana!");

Step 19

System.out.println("Welcome to Jerusalem!");

1.Java is Case Sensitive

2.Quotation mark displays the message as is

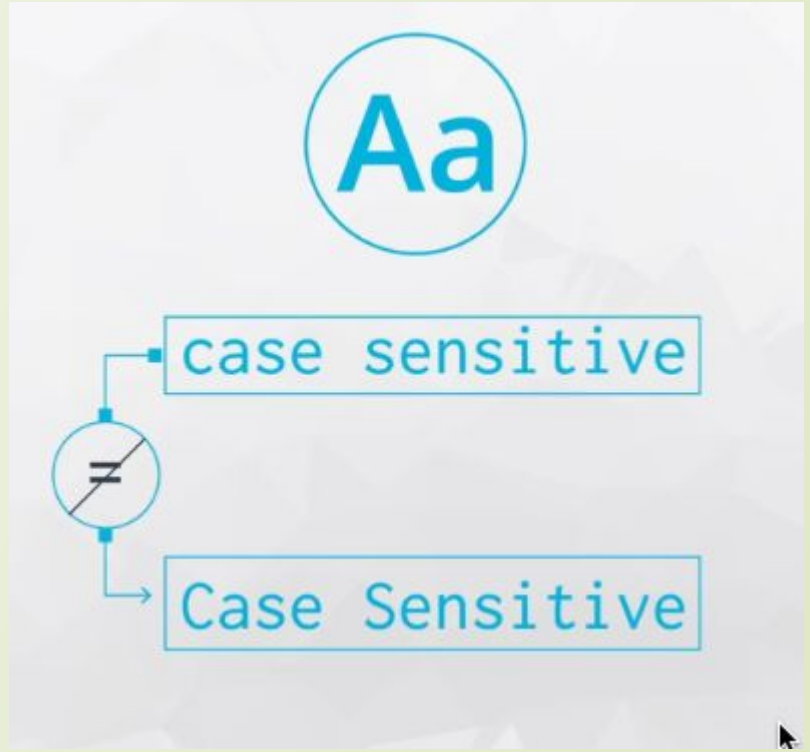
3.Semi-colon means end of statement

```
System.out.println("Hello World!");
System.out.println(2+3);
```

System.out.println("2+3") // 2+3

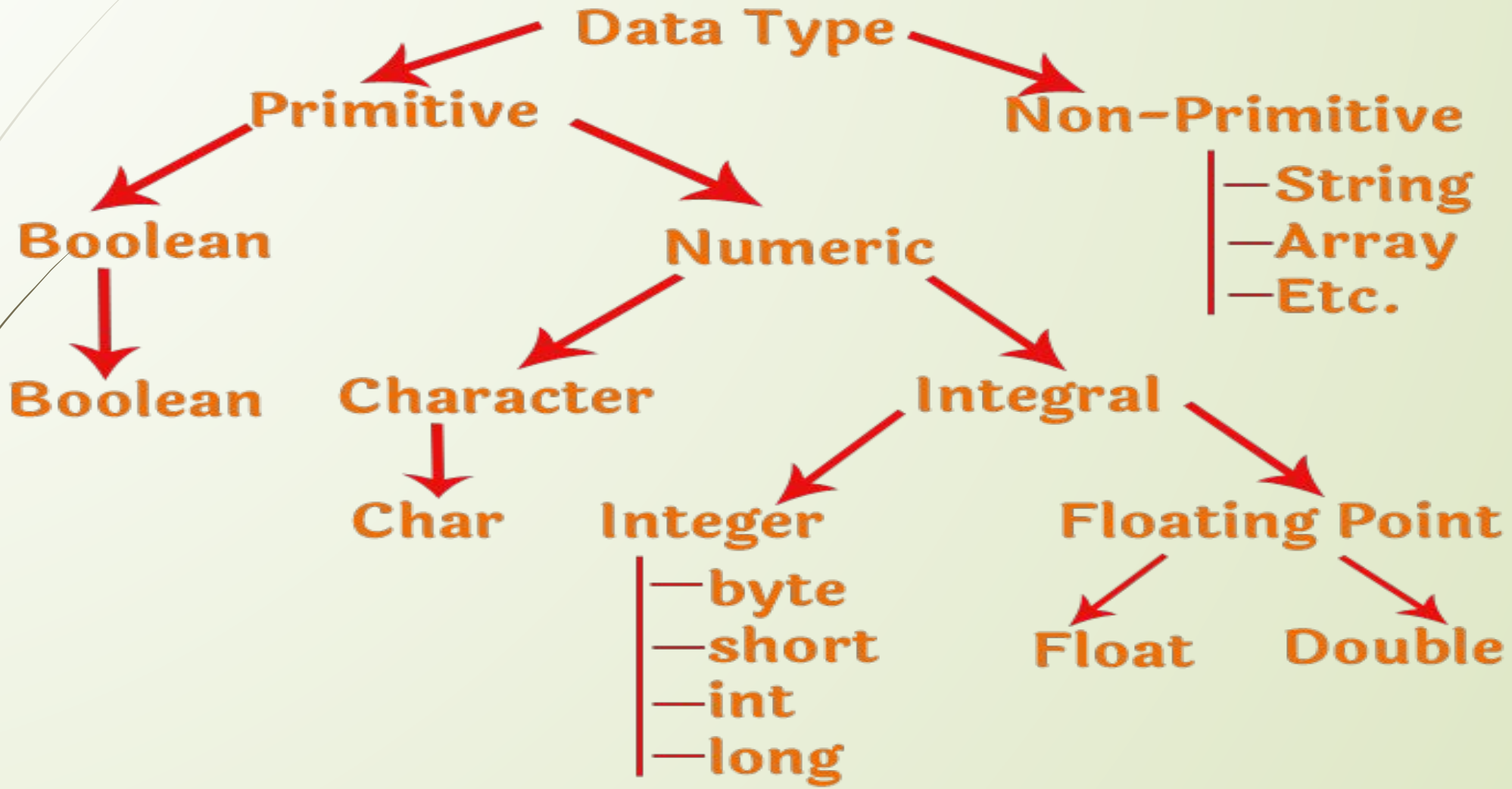
Print output:

```
Hello world!
5
```



Step 20

Data Types



Step 21

What Is Data Type?

- **Data type** specifies the size and **type** of values that can be stored in an identifier.

  The **Java** language is rich in its **data types**. ...

- **Data types in Java** are classified into two types:

- Primitive—which include Integer, Character, Boolean, and Floating Point.

- Non-primitive—which include Classes, Interfaces, and Arrays types

Step 22

Primitive Data types/Non Primitive data Types

Primitive data type	Non-Primitive data types
• It is predefined in java	• It is created by programmer and not defined in java.
• It cannot call method	• It can be used to call a method to perform certain operation.
• It always have value	• It can be null
• It starts with Lowercase	• It starts with Uppercase
• Size is depends on the data types	• It have all same size

8 types of primitive data types

Step 23

Boolean Data Types: The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example: Boolean one = false

Byte Data Types: The byte data type is used to save memory in large arrays. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example: byte a = 10, byte b = -20

Short Data Types: The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example: short s = 10000, short r = -5000

Int Data Types: The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example: int a = 100000, int b = -200000

Cont.....

Step 24

Long Data types: The long data type is used when you need a range of values more than those provided by int.

Example: long a = 100000L, long b = -200000L

Float Data types: The float data type is a single-precision 32-bit. It is recommended to use a float (instead of double) to save memory in large. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example: float f1 = 234.5f

Double Data Types: The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

Example: double d1 = 12.3

Char Data Type: The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

Example: char letterA = 'A'

Simple Program for Primitive Data types

Step 25

```
class  Primitivedatatypes
{
    public static void main(String[]args)
{
     int a=25;
     Boolean b= True;
      char c='S';
     double d= 300.9900d;
      short e= 32;
      float f =78.9f;
       byte g=30;
       long h=3000000000L;
   System.out.println("Char word is" +c);
}
}
```

O/P

Char word is S

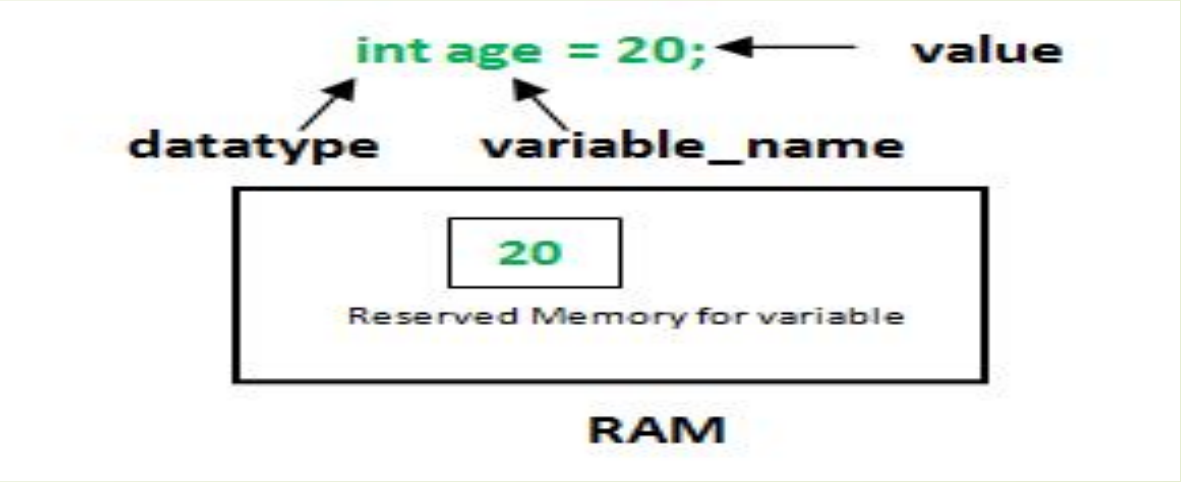
Variables

Step 26

What is java variable?

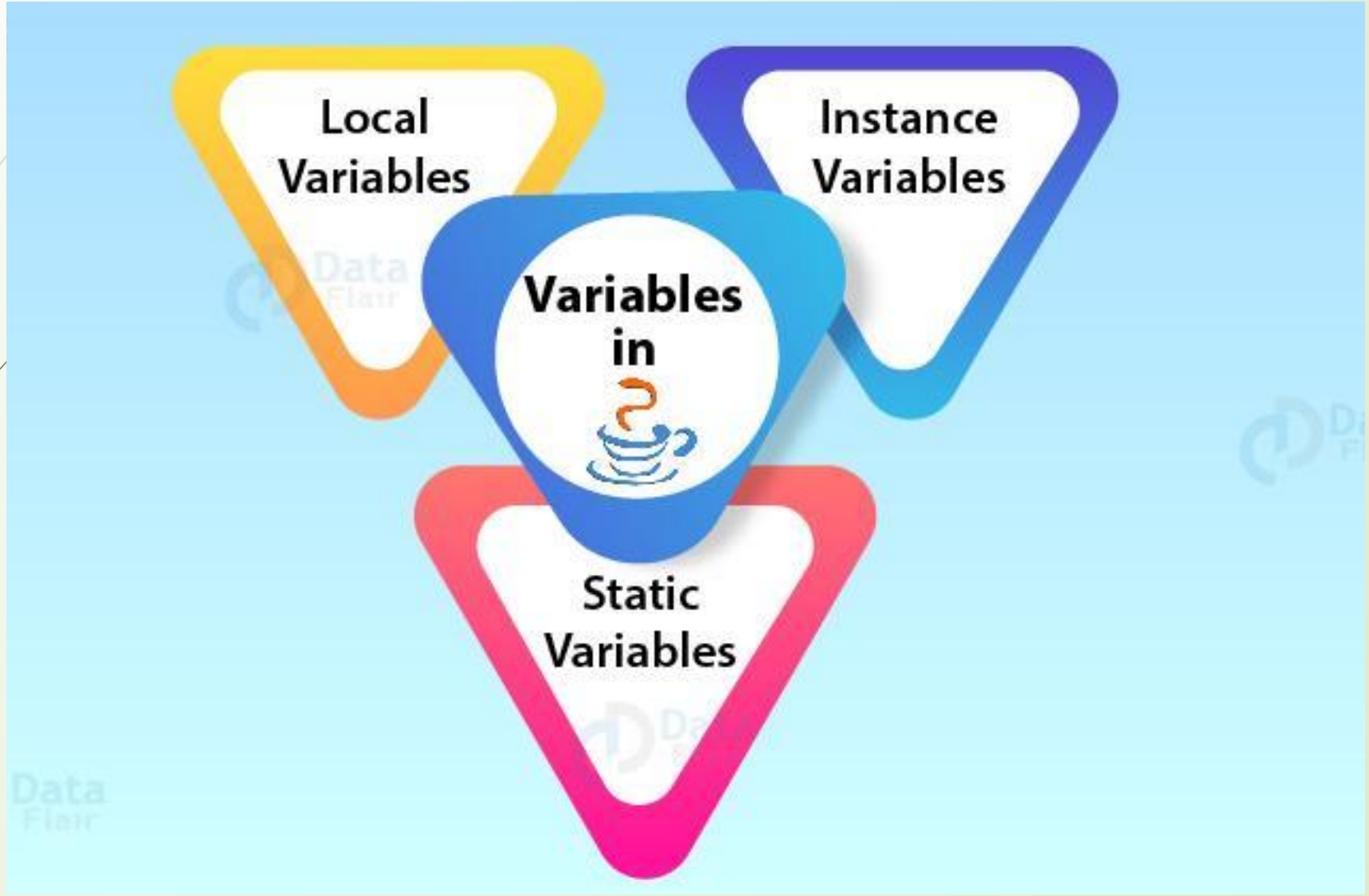
- A variable is a container which holds the value. A variable is assigned with a data type.
- Variable is a name of memory location.
- There are three types of variables in java: local, instance and static.

Variable is name of *reserved area allocated in memory.* In other words, it is a *name of memory location.* It is a combination of "vary + able" that means its value can be changed.



Types of variable

Step 27



Local Variable

Step 28

- ✔ A variable declared inside the body of the method is called local variable.
- ✔ You can use this variable only within that method

EXAMPLE:

```
class Localvariable
{
    public static void main(String args[])
{
    String s ="Hai";// local variable
 System.out.println("Display" + s);
    }
}
```

Output

Display Hai

Instances variable

Step 29

- A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.
- It is called instance variable because its value is instance specific and is not shared among instances.

EXAMPLE:

```
        class Instantvariable
{
float f =7.1f;//Instant Variable
    public static void main(String args[])
{
String s ="Hai";// local variable
 System.out.println("Display" + s);


}
}
```

Output

Display Hai

Static Variable

Step 30

- ✔ A variable which is declared as static is called static variable. It cannot be local.
- ✔ You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

EXAMPLE:

```
        class Instantvariable{
Static String s= 100//Static variable
    public static void main(String args[])
{
   String s ="Hai";// local variable
 System.out.println("Display" + s);
}
}
```

Output

Display Hai

Step 31

Updating Variables

```
int passengers;
passengers = 0;
passengers = passengers + 5;
passengers = passengers - 3;
passengers = passengers - 1 + 5;
System.out.println("Number of Passengers: " + passengers);
// Number of Passengers: 6
```

Updating vs. Setting Variables

Step 32

- When changing a variable value, you could either set it to a new value all together, or update it based on its previous value.
- For example, if we have a variable called time and we want to add 5 to it, we write:

```
time = time + 5;
```

 Updating

- This will **Add** 5 to whatever value time already had!
- But if we want to set time to 5 without caring about what the current value was, we write:

```
time = 5;
```

 Setting

Step 33

Quiz - 1

Update the variable passengers to subtract 1 and add 5

```
int passengers;
passengers = 0;
passengers = passengers + 5;
passengers = passengers - 3;
```

- `passengers = passengers - 1 + 5;`
- `passengers = 5 - 1;`
- `passengers = passengers - 5 + 1;`
- `passengers = 5;`
  `passengers = passengers - 1;`

Displaying variables

Step 34

```
int Chocolates;
Chocolates = 0;
Chocolates = Chocolates + 5;
Chocolates = Chocolates - 3;
passengers = passengers - 1 + 5;
System.out.println(Chocolates);
```

Displaying

- Java automatically assumes that Chocolates is a variable and starts looking for the value stored inside the Chocolates variable and displays it because it is a print line statement
- If we run the program, the output will be 2

Multiple variables

Step 35

- Computers can keep track of more than one variable at a time something that us humans struggle with.

for example:

```
int  choki;
 int cakepiece;
 choki=0;
 cake piece=0;
  choki=cakepiece + 10;
  cake piece=choki +20;
  choki=cakepiece -5;
  cake piece= 50+ choki;
System.out.println(choki);
```

Output

25

Quiz

Step 36

Find the output for this following program

```
int ball;
int stone;
int choki;
choki=0;
ball=0;
stone=0;
ball=choki+87;
stone=ball-10;
choki=stone+ball;
stone=choki+3;
System.out.println(stone);
```

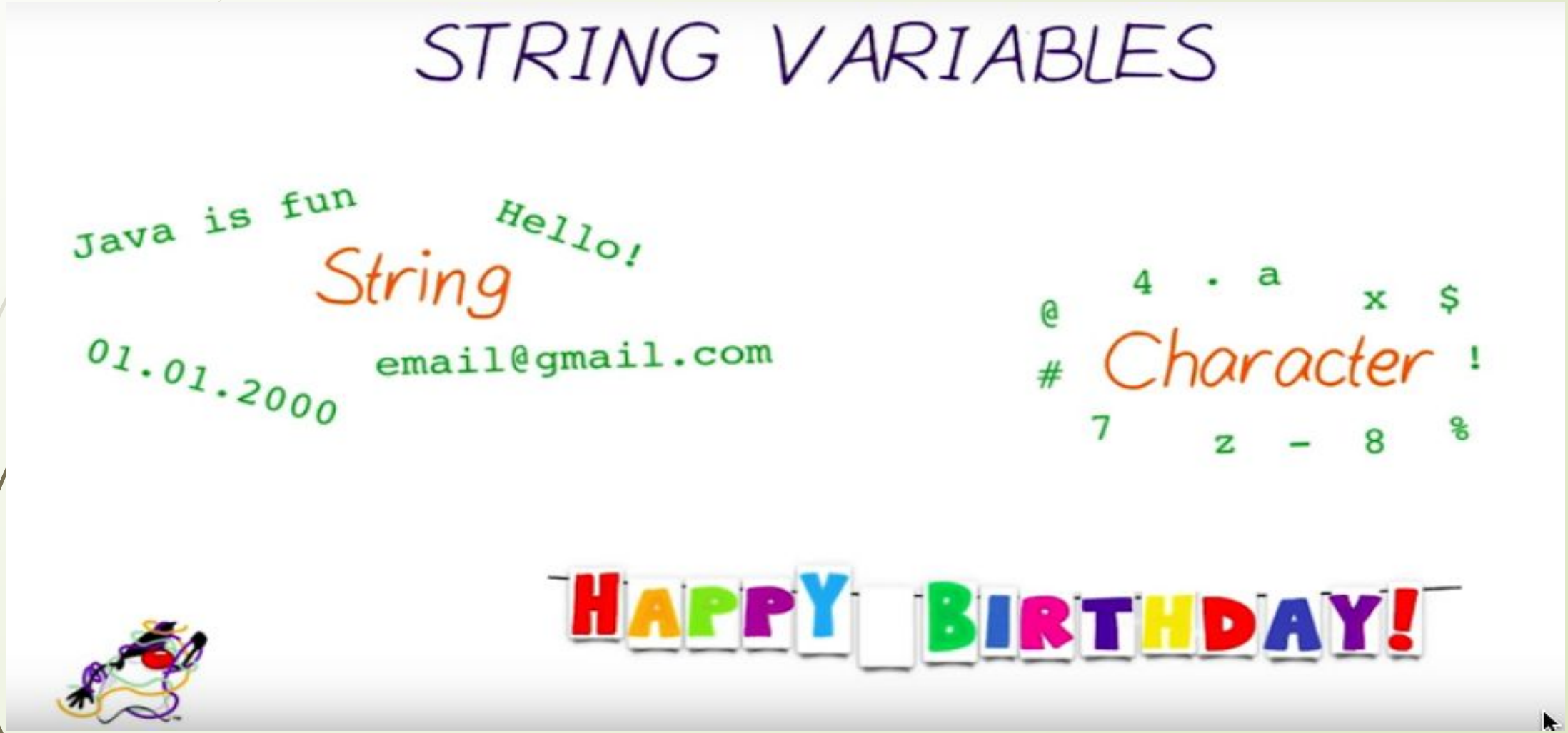
String Variable

Step 37

- A string variable is used to store names, words or even whole sentences.
- Anything that's made up of characters.
- In Java, A character is a single alphabetical letter or digit or even a symbol like the dollar sign or a hash or @ that you can use when you are creating a password for example.
- If you want to store a phrase or a word that's made up of those characters you should use a string variable
- Its like string connecting bunch of characters

String Variable

Step 38



String Variable Example

Step 39

```
class  Mybod
{
Public static void main(String[] args)
{
String date;
date= "16.08.2001";
System.out.println("My B'day Date is " +date);
}
}
```

o/p

My B'day Date Is 16.08.2001

- "16.08.2001" is a String Literal
- Here, 16.08.2001 is surrounded with quotation marks

To count the letters

Step 40

```
Variablename.length();
```

It counts the number of letters inside the variable name not the word name itself

We need to store that number in a variable since letter counts are natural numbers we can use an integer for that

```
int letters = name.length()
```

We stored the numbers in the integer letters.

To count the letters

Step 41

Example:

```
String name;

name = "Micky";

int letters = name.length();

System.out.println(letters);
```

Output: 5

Uppercase

Step 42

Example:

```
String name;

name = "micky";

name = name.toUpperCase();

System.out.println(name);
```

Variablename.toUpperCase()

Output:

MICKY

Step 43

Lower Case

Example:

```
String name;

name = "MICKY";

name = name.toLowerCase();

System.out.println(name);
```

Variablename.toLowerCase()

Output:

micky

String Concatenation

Step 44

- Another thing we can do to strings, is actually add them to Strings.
- This is called String concatenation
- It is basically joining strings together to build a longer string.

```
String a = "Micky";
String b = "Mouse";
String c = a + b;
System.out.println(c);
```

Output: MickyMouse

✔ We can add extra white space in it by using quotation

```
String c = a+ " " + b;
```

Quiz

Step 45

Program

```
String a ="Java";
String b ="Program"
```

Print output:

This is Java Simple Program

Example Program

Step 46

```
class Program
{
public static void main(String[] args)
{
String name1 ;
String name2 ;
String name3;
 name1="Malu";
 name2="Malu";
 name3=name1+" "+name2;
name3=name3.toUpperCase();
int num=name3.length();
System.out.println(num);
System.out.println(name3);
}
}
```

O/p:

9
MALU MALU

Variable Names

Step 47

String driverFirstName = "Harish";

- This is called Camel Case
- Because it resembles the humps of a camel back.
- However the first letter here seemed to be a lowercase letter. It is lower camel case
- Another variation is called the upper camel case

Variable Names

Step 48

- Lower camel case is used when creating a new variable like

  String driverFirstName = "Harish";

- We can't have spaces between the words in a variable name
- It is easy to read
- Upper camel case is

  String DriverFirstName = "Harish";

Variable Name Rules

Step 49

- Start variable names with an alphabetical letter
- Start variable name with underscore sign (_) or dollar sign($).
- Cannot have white spaces
- Can have numbers and symbols within your variable name.

Variable Arithmetic

Step 50

1.Addition	int add = 2 + 3;	5
2.Subtraction	int sub = 3 - 5;	-2
3.Multiplication	int mul = 5 * 5	25
4.Division	double div = 5/2	2.5

double div1 = 5/2.0 = 2.5

Example:

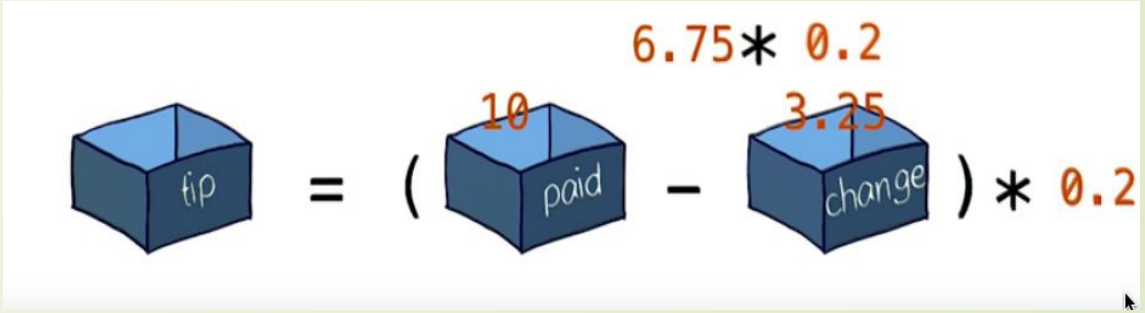
```
int x = 2 + 3;

int y = 4 - 5;

int z = x * y;
```

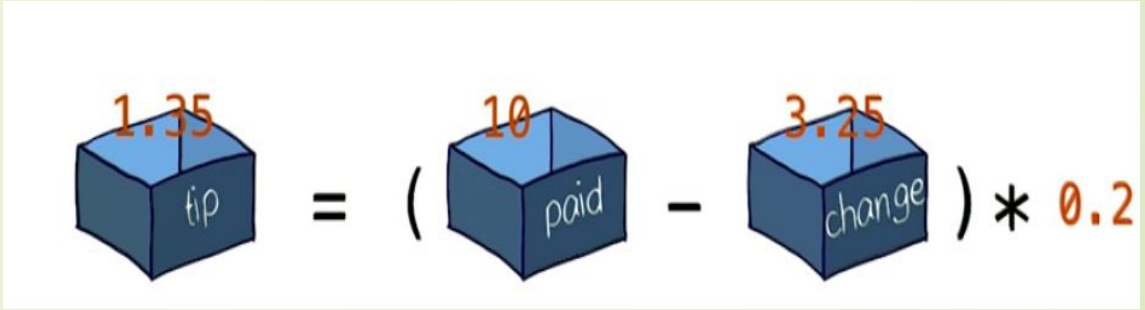
Variable Arithmetic

Step 51

Example:

```
double paid = 10;

double change = 3.25;

double tip = (paid - change) * 0.2;
```



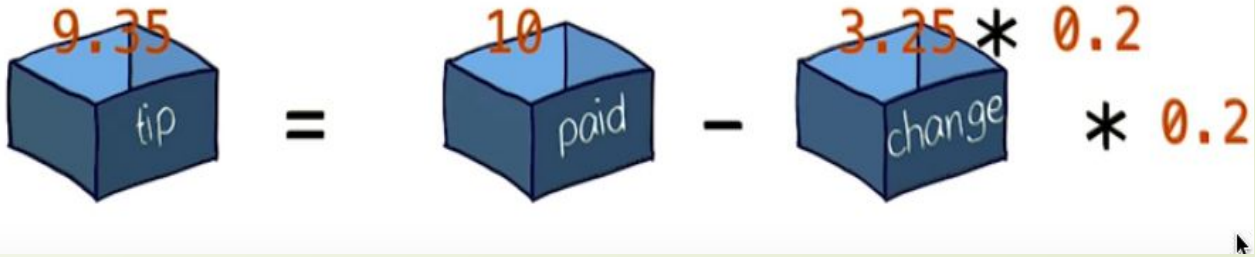


Cont…….

Step 52

Without Parenthesis:

```
double paid = 10;
double change = 3.25;
double tip =  paid-change *0.2;
```



✔ Always remember to use those set of parentheses when arithmetic operations are perform in certain order.

Type Casting

Step 53

Assigning a value of one type to a variable of another type is known as **Type Casting**.

```
dataType variableName = (datatype) variableToConvert
```

```
char a='X';
int b;
    b =(int) a;
```

(Type Casting)

Types of Type Casting

Step 54

1. Widening casting(implicit)
2. Narrowing casting(explicit)

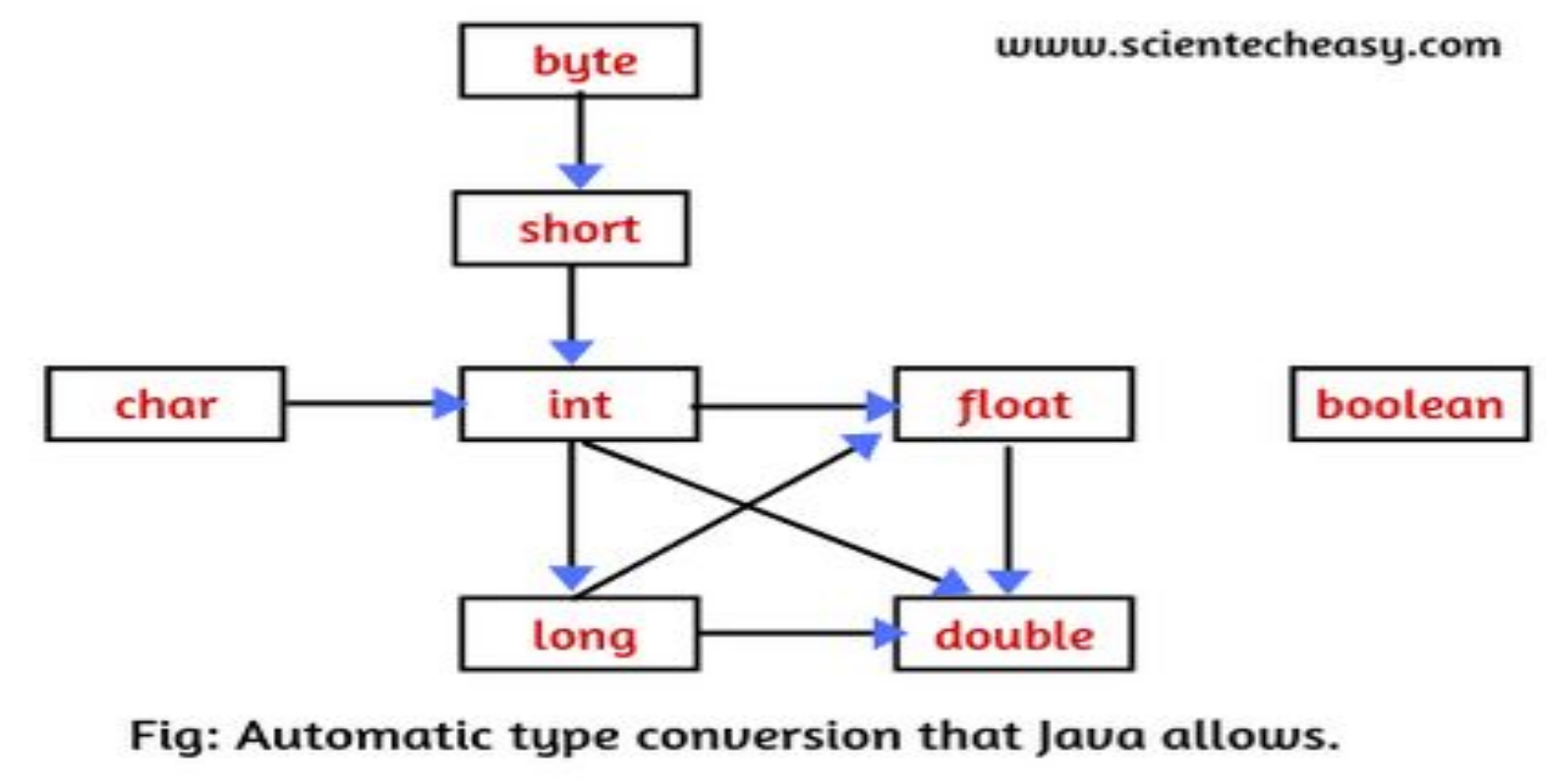


Fig: Automatic type conversion that Java allows.

Example for Type casting

Step 55

1.Widening casting(implicit):

```
class Type
{
 public static void main(String[] args)
{
int i =69;
float f;
f=(float) i;
System.out.println(f);
}
}
```

o/p

69.0

Cont……

Step 56

2.Narrowing casting(explicit)

```
class Type
{
 public static void main(String[] args)
{
int i ;
float f=69.3f;
i=(int) f;
System.out.println(i);
}
}
```

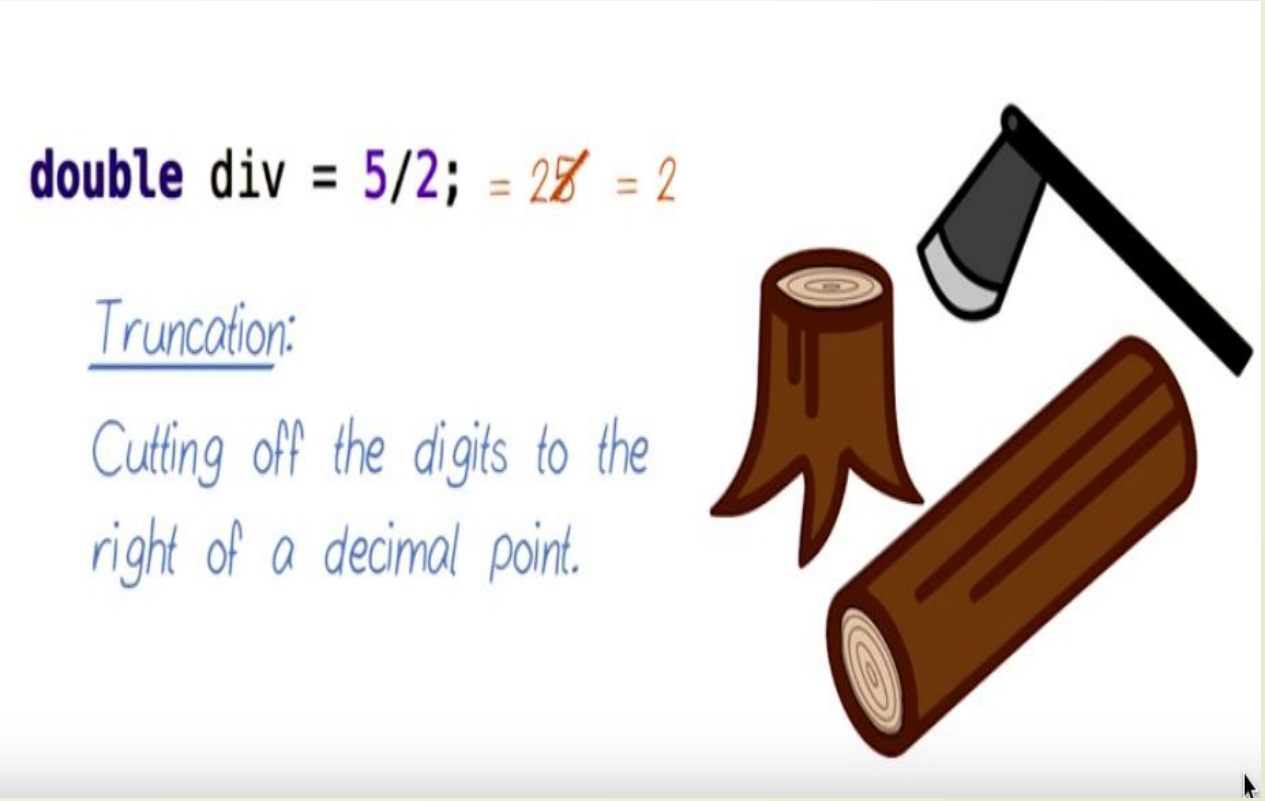
o/p

69

Truncation

Step 57

In Java, fractional part is just thrown away and only the integer part remains.



Truncation

Step 58

```
double current = 17;
double rate = 1.5;
double future = current * rate;
System.out.println(future);
```

Output:

25.5

```
double current = 17;
double rate = 1.5;
double future = current * rate;
System.out.println(future);
int approx  = (int) future;
System.out.println(approx );
```

We can convert that double variable into an integer

Truncation cont…….

Step 59

Example:

```
double current = 17;
double rate = 1.5;
double future = current * rate;
System.out.println(future);
int approx  = (int) future;
System.out.println(approx);
```

Output:

25

Java Operators

Step 60

Operator in Java is a symbol which is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in Java:

Unary Operator,

Arithmetic Operator,

Shift Operator,

Relational Operator,

Bitwise Operator,

Logical Operator,

Ternary Operator and

Assignment Operator.

Java Operator Precedence

Step 61

Operator Type	Category	Precedence
Unary	postfix	*expr++ expr--*
	prefix	*++expr --expr +expr -expr ~ !*
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Unary Operators

Step 62

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

- incrementing/decrementing a value by one
- negating an expression
- inverting the value of a boolean

Java Unary Operator Example: ++ and --

```
1.class OperatorExample{
2.public static void main(String args[]){
3.int x=201;
4.System.out.println(x++);//201(202)
5.System.out.println(++x);//203
6.System.out.println(x--);//203 (202)
7.System.out.println(--x);//201
8.}}
```

Output:

```
201
203
203
201
```

Step 63

Example 2

```
class OperatorExample
{
public static void main(String args[])
{
int a=10;
int b=10;
System.out.println(a++ + ++a);//10+12=22
System.out.println(b++ + b++);//10+11=21
  }
}
```

Java Unary Operator Example: ~ and !

Step 64

```
class OperatorExample
{
public static void main(String args[])
{
int a=17;
int b=-100;
boolean c=true;
boolean d=false;
System.out.println(~a);//-18 (minus of total positive value which starts from 0)
System.out.println(~b);//99 (positive of total minus, positive starts from 0)
System.out.println(!c);//false (opposite of boolean value)
System.out.println(!d);//true
}
}
```

Java Arithmetic Operators

Step 65

Java arithmatic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

Example:

```
        class operators
{
Public satic void main(Strings[] args ){
 int a= 19;
int b =21;
System .Out.Println(a+b);
System .Out.Println(a-b);
System .Out.Println(a*b);
System .Out.Println(a/b);
System .Out.Println(a%b);}}
```

Step 66

Java Arithmetic Operator Example: Expression

```
class Operators {

 public static void main(String[] args) {

  System.out.println(15-1*10+50+17*21/3);

   }

}
```

o/p

174

Java Logical & Bitwise AND Operator

Step 67

- ✔ The logical && operator doesn't check second condition if first condition is false. It checks second condition only if first one is true.
- ✔ The bitwise & operator always checks both conditions whether first condition is true or false.

EXAMPLE:

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a<c);//false && true = false
System.out.println(a<b&a<c);//false & true = false
}}
```

O/P

```
false
false
```

Java Logical & Bitwise || Operators

Step 68

- ✔ The logical || operator doesn't check second condition if first condition is true. It checks second condition only if first one is false.
- ✔ The bitwise | operator always checks both conditions whether first condition is true or false.

EXAMPLE:

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a>b||a<c);//true || true = true
System.out.println(a>b|a<c);//true | true = true  }}
```

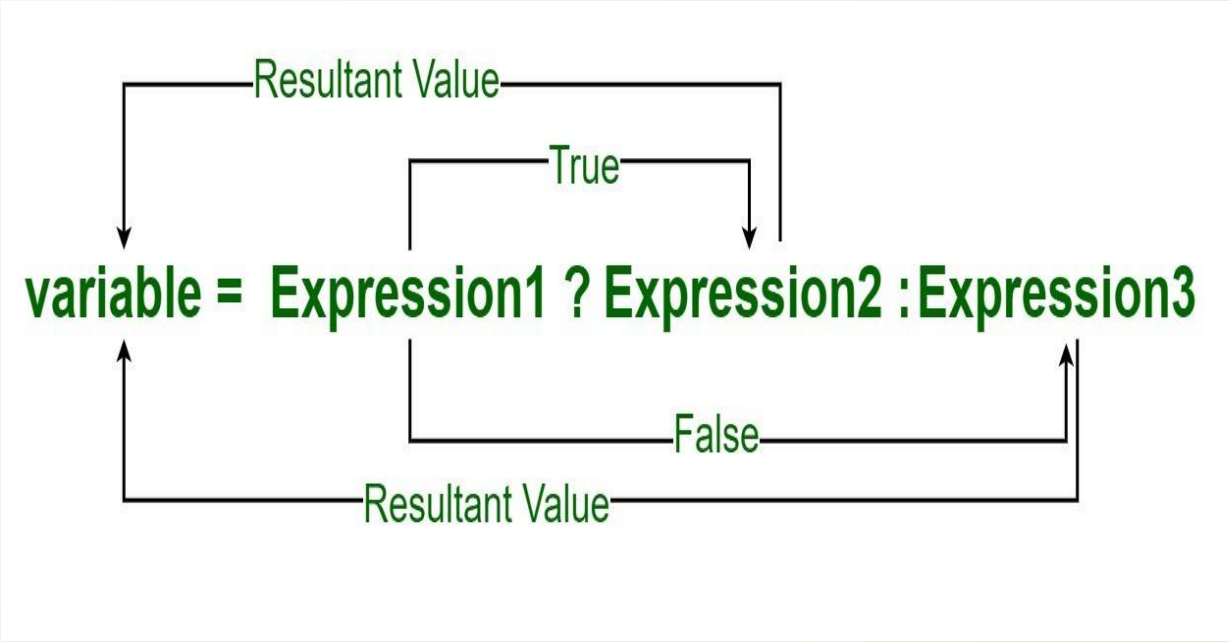
O/P

```
true
true
```

Java Ternary Operator

Step 69

Java ternary operator is the only conditional operator that takes three operands. It's a one-liner replacement for if-then-else statement and used a lot in Java programming. We can use the ternary operator in place of if-else conditions or even switch conditions using nested ternary operators



Example Program

Step 70

```
class OperatorExample{
public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}
}
```

Output:

2

Java Assignment Operator

Step 71

Java assignment operator is one of the most common operator. It is used to assign the value on its right to the operand on its left.

Example:

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=20;
a+=4;//a=a+4 (a=10+4)
b-=4;//b=b-4 (b=20-4)
System.out.println(a);
System.out.println(b);
}}
```

O/P

```
14
16
```

Another Example

Step 72

```
class OperatorExample{
public static void main(String[] args){
int a=10;
a+=3;//10+3
System.out.println(a);
a-=4;//13-4
System.out.println(a);
a*=2;//9*2
System.out.println(a);
a/=2;//18/2
System.out.println(a);
}}
```

O/P

13
9
18
9

Step 73

CONTROL FLOW

Step 74

CONTROL FLOW

- We will talk about how to build programs that make decisions and can follow different sets of instructions based on the information we give it
- This is called control flow
- We use it to build programs together to do things like make a smart coffee maker, automatically set museum admission prices, create a restaurant rating system and more
- Need a tool to build a program that follows any rules you specify.

Control Flow

Step 75

- ✔ Statements can be executed multiple times or only under a specific condition
- ✔ The if, else, and switch statements are used for testing conditions.
- ✔ while and for statements to create cycles.
- ✔ break and continue statements to alter a loop.

- ❑ If...else statement
- ❑ switch
- ❑ for loop
- ❑ for-each loop
- ❑ while Loop
- ❑ break Statement
- ❑ continue statement

Java If-else Statement

Step 76

The Java *if statement* is used to test the condition. It checks boolean condition: *true* or *false*.

There are various types of if statement in Java.

- if statement
- if-else statement
- if-else-if ladder
- nested if statement

If Statement

Step 77

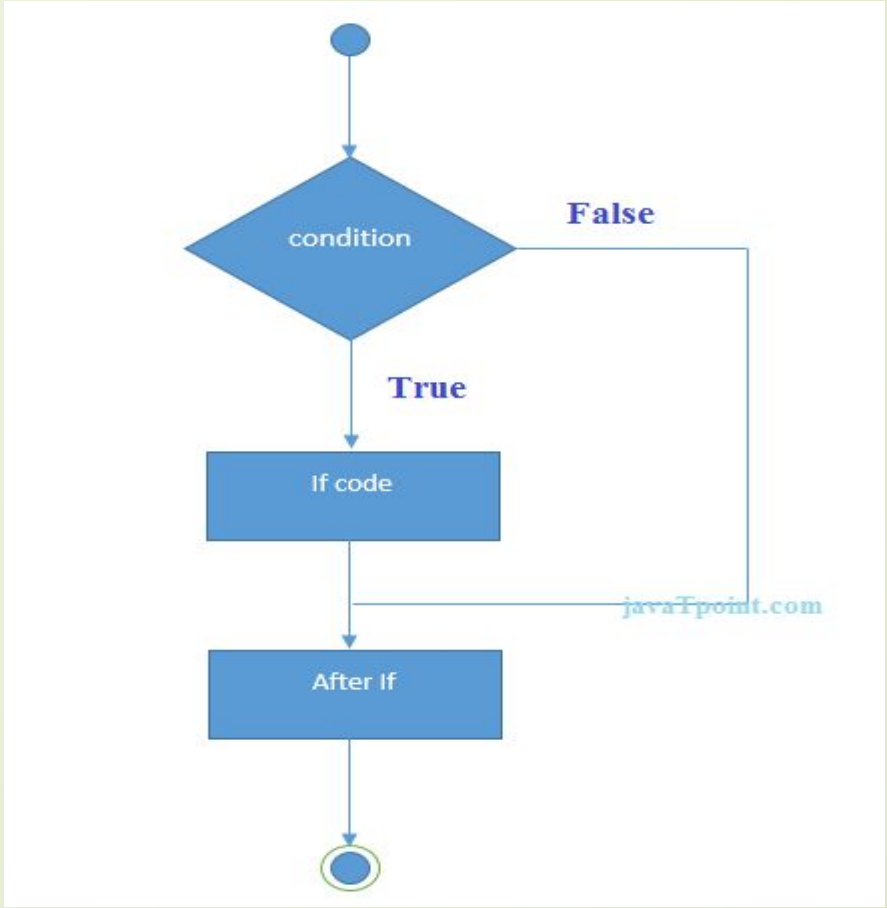
The Java if statement tests the condition. It executes the *if block* if condition is true.

SYNTAX

```
if(condition)
{
        Statement//code to be executed
}
```

EXAMPLE

```
 class IfExample {
public static void main(String[] args)
 {
     int age=20;
  if(age>=18){
       System.out.println("Age is greater than 18");

    }
}
 }
```



Self-Driving Car

Is the light green?

If yes, then drive!

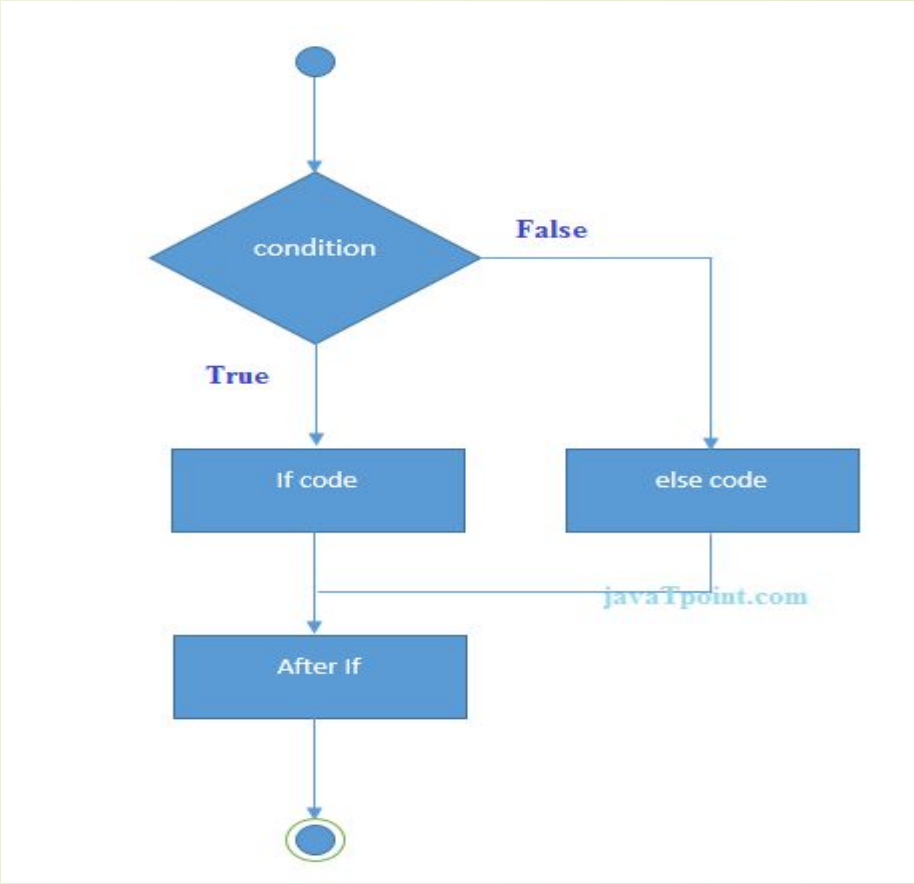
Java If-else Statement

Step 79

The Java if-else statement also tests the condition. It executes the *if block* if condition is false otherwise *else block* is executed.

SYNTAX

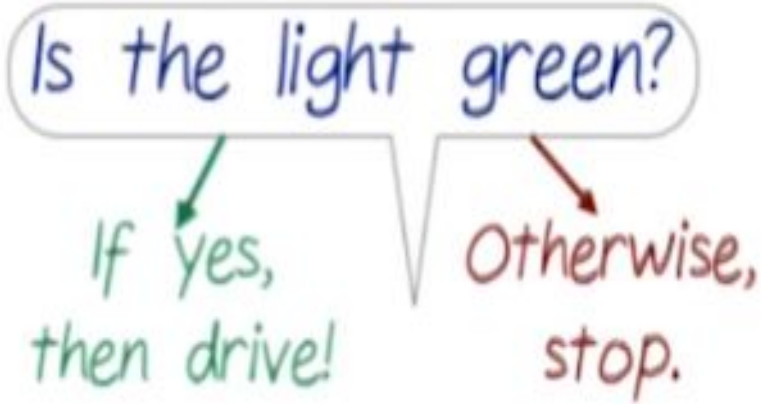
```
if(condition) {
//code if condition is true
}
else
{
//code if condition is false
}
```

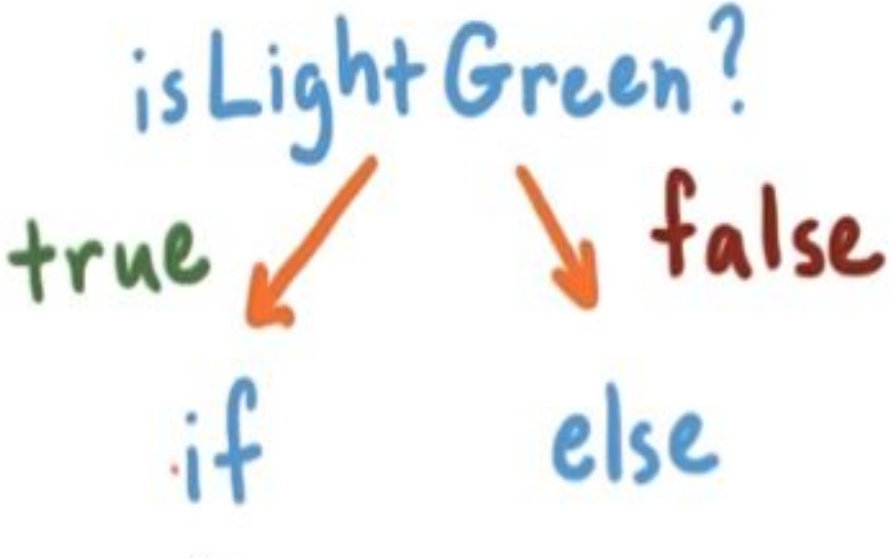


else Statement

Comes after an if statement

Provides an alternate set of instructions to follow





Step 82

EXAMPLE

```
class signal {
public static void main(String[] args) {
    //defining a variable
        int number=13;
//Check if the number is divisible by 2 or not
    if(number%2==0){
       System.out.println("even number");
    }else{
        System.out.println("odd number");
    }
}
}
```

O/P

Odd number

Ternary Operator

Step 81

We can also use ternary operator (? :) to perform the task of if...else statement. It is a shorthand way to check the condition. If the condition is true, the result of ? is returned. But, if the condition is false, the result of : is returned

EXAMPLE:

```
public class IfElseTernaryExample {
public static void main(String[] args) {
    int number=13;
    //Using ternary operator
    String output= (number%2==0)?"even number":"odd number";
    System.out.println(output);
}
}
```

Ternary operator

O/P

odd number

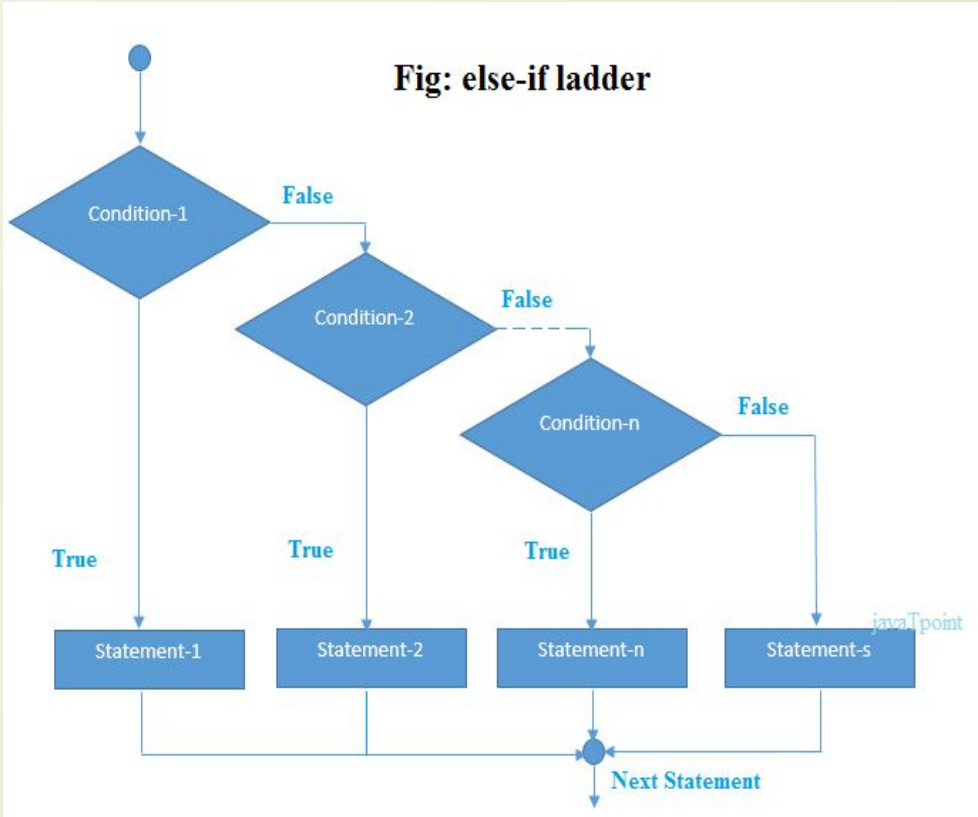
Java if-else-if ladder Statement

Step 83

The if-else-if ladder statement executes one condition from multiple statements.

SYNTAX

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```



Example Program

Step 84

```
class IfElseStatement {
  public static void main(String[] args) {
    int a,b,c,d;
    a=10;
    b=20;
    c=30;
    d=40;
    if(a>b)
    {
      System.out.println("a is true");
    }
    else if(a>c)
    {
    System.out.println("c is big");
    }
  else if(c<d)
  {
    System.out.println("Hai ");
  }
    else
    {
      System.out.println("Bye ");
    }

  }
}
```

O/P

Hai

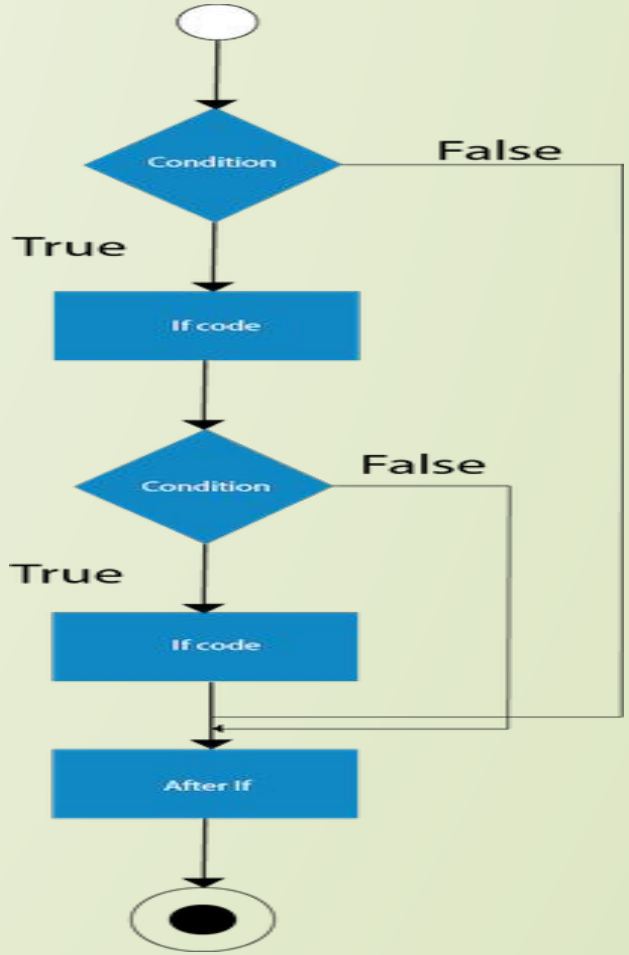
Java Nested if statement

Step 85

The nested if statement represents the *if block within another if block*. Here, the inner if block condition executes only when outer if block condition is true.

SYNTAX

```
if(condition){
 if(condition){
        //code to be executed
   }
}
```



Example Program

Step 86

```
public class JavaNestedIfExample2
 {
public static void main(String[] args) {
  int age=25;
     int weight=48;
   if(age>=18){
      if(weight>50){
  System.out.println("You are eligible to donate blood"); }
 else
{
System.out.println("You are not eligible to donate blood");
        }
    }
else
{
     System.out.println("Age must be greater than 18");
    }
} }
```

O/P

You are not eligible to donate blood

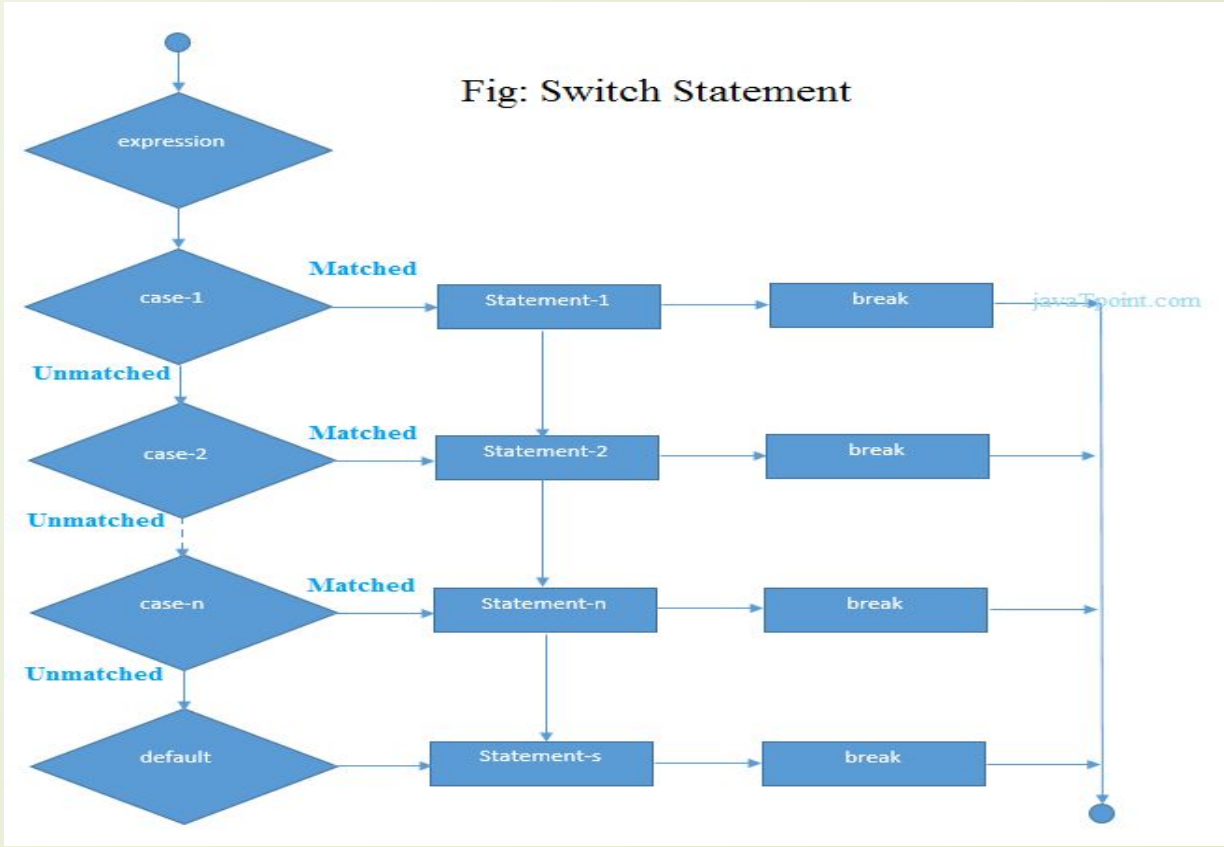
Java Switch Statement

Step 87

The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement. The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long.

SYNTAX

```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
.....

default:
 code to be executed if all cases are not matched;
}
```



Example Program

Step 88

```
class Switch {
  public static void main(String[] args) {
    int order=2;
    String icename="";
    switch(order)
    {
      case 1:icename="Vanila icecream" ;
        break;
      case 2:icename="Chocolate Shake" ;
        break;
      case 3:icename="I Bar Icecream";
        break;
      case 4:icename="Butterscotch icecream" ;
        break;
      case 5:icename="Chocolate icecream" ;
        break;
     default:System.out.println("Give Me Menu Card");
        break:
    }
    System.out.println(icename);
     }}
```

O/P

Chocolate Shake

switch vs if Statement

```
class Switch {
 public static void main(String[] args) {
  int order=2;
  String icename="";
  switch(order)
  {
    case 1:icename="Vanila icecream" ;

    break;
   case 2:icename="Chocolate Shake" ;
    break;
   case 3:icename="I Bar Icecream";
    break;
   case 4:icename="Butterscotch icecream" ;
    break;
    case 5:icename="Chocolate icecream" ;
    break;
default:System.out.println("Give Me Menu Card");
 }
 System.out.println(icename);
  }}
```

```
 class Switch{
pubic static void main(String[] args)
int order=2;
if(order!=2)
  {
 System.out.println("Vanila icecraem");
  }
else if(order==2)
 {
System.out.println("Chocolate Shake");
 }
 else
 {
System.out.println("Give Me Menu Card");
 }
```

Step 91

Loops In Java

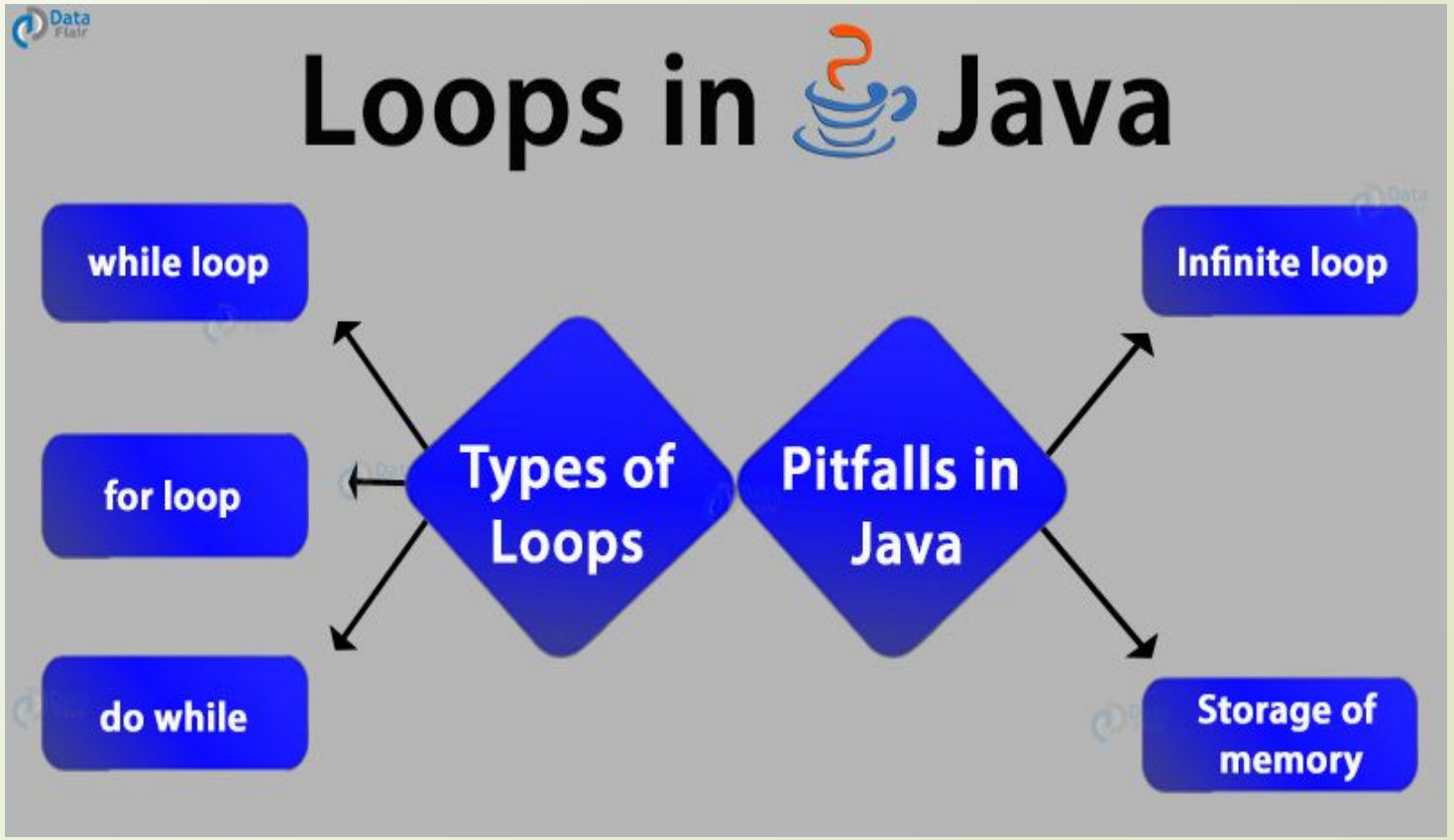
Loops

Step 92

In Programming Language,loops are used to execute set of instruction/function repeatedly when some condition became true.

There are three types of loops

1) for loops
2) While loop
3) do-While loop



Step 93

Java For Loop

Java For loop is used to iterate the program several times,If the number of iteration is fixed,it is recommended to use For loop.

There are three types of For loop

- Simple For loop
- For each or Enhanced For loop
- Labeled For Loop

Simple For Loop

A simple For loop is same as C/C++.We can initialize the variable,check condition and increment /decrement value.

Syntax

Step 94

```
for(initialization;condition;incre/decre)
{
    Statement code
}
```

INITIALIZATION -It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.

CONDITION -It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return Boolean value either true or false. It is an optional condition.

STATEMENT -The statement of the loop is executed each time until the second condition is false.

INCREMENT/DECREMENT -It increments or decrements the variable value. It is an optional condition.

Example Program

Step 95

```
class Example {

  public static void main(String[] args) {

    for(int i =90; i<=100;i++)

    {

    System.out.println(i);

    }

  }

}
```

O/P

```
90
91
92
93
94
95
96
97
98
99
100
```

Another Example

Step 96

```
class Example {

  public static void main(String[] args) {

    for(char i =97; i<=110;i++)

    {

     System.out.println(i);

    }

  }

}
```

O/P

```
a
b
c
d
e
f
g
h
i
j
k
l
m
n
```

Java Nested For Loop

Step 97

If we have a for loop inside the another loop, it is known as nested for loop. The inner loop executes completely whenever outer loop executes.

EXAMPLE:

```
class Example {
  public static void main(String[] args) {
    for(char i=97;i<=101;i++){
for(char j=97;j<=i;j++){
        System.out.print(i);
}
System.out.println();
}
}
}
```

O/P

```
a
bb
ccc
dddd
eeeee
```

Example Program

Step 99

Step 98

```
class loop
{
public static void main(String[] args)
{
char  arr[]={96,97,98,99,100};
for(char c:arr)
{
System.out.println(c);
}
}
}
```

O/P

a
b
c
d

Java Labeled For Loop

Step 100

We can have a name of each Java for loop. To do so, we use label before the for loop.

SYNTAX

```
labelname:
for(initialization;condition;incr/decr)
{
//code to be executed
}
```

Example Program

Step 101

```
class label
{
  public static void main(String[] args)
  {
    aa:
    for(int i=1;i<=3;i++)
    {
       System.out.println(i);
    }
      bb:
      for(char j=65;j<=69;j++)
{
        System.out.print(j);
      }


    System.out.println();

  }
}
```

O/P
1
2
3
ABCDE

Java While Loop

Step 102

The Java *while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

SYNTAX

```
while(condition){
//code to be executed
}
```

Example Program

Step 103

```
class  While
{
public static void main(String[] args)
{
char c=67;
while(c<=77)
{
System.out.println(c);
c++;
}
}
}
```

O/P

C
D
E
F
G
H
I
J
K
L
M

Java do-while Loop

Step 104

The Java do-while loop is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

The Java do-while loop is executed at least once because condition is checked after loop body.

SYNTAX

```
do{
//code to be executed
}while(condition);
```

Example Program

Step 105

```
class dowhile
{
public static void main(String[] args)
{
char a=97;
do
{
System.out.println(a);
a++;
}
while( a<=101);
}
}
```

O/P

a
b
c
d
e

Java Break Statement

Step 106

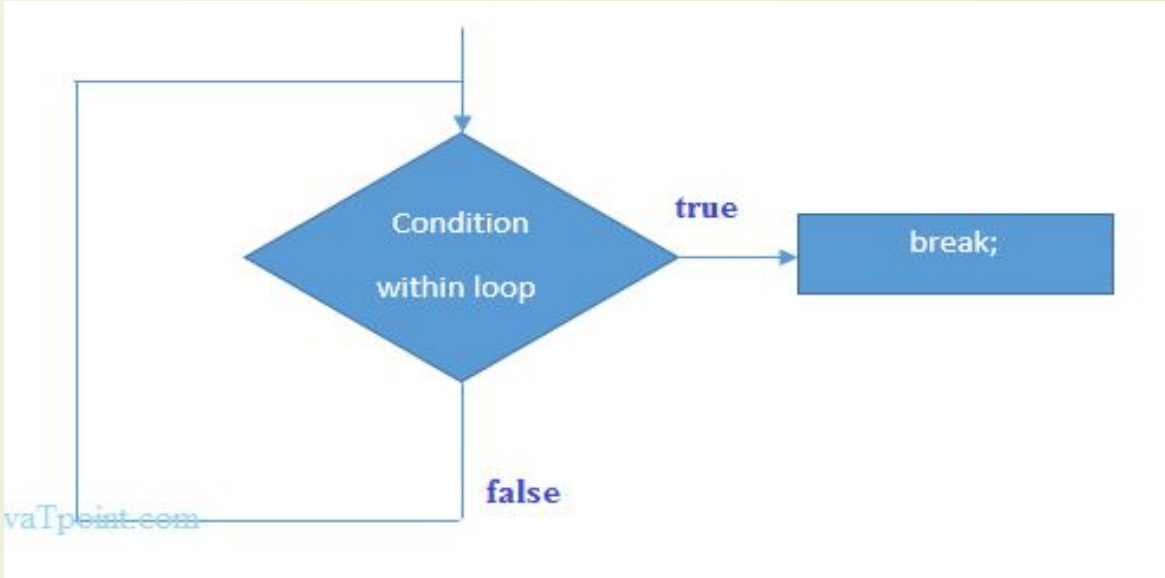
The Java break statement is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

SYNTAX

jump-statement;

break;



Example Program With Loop

Step 107

```
class Break
{
public static void main(Strings[] args)
{
char a ;
for(a=97;a<=105;a++)
{
if (a==102)
{
break;
}
System.out.println(a);
}
}
}
```

O/P

a
b
c
d
e

Java Break Statement in while loop

Step 108

```
class break
{
public static void main(String[] args)
{
char a =97;
While(a<=108)
{
a++;
if (a==102)

{
break;
}
System.out.println(a);
}
}
}
```

O/P

b
c
d
e

Java Break Statement in do-while loop

Step 109

```
class break
{
public static void main(String[] args)
{
char a=97;
do
{

if(a==102)
{
break;
}
System.out.println(a);
a++;
}
while(a<=108);
}
}
```

O/P

a
b
c
d
e

Java Continue Statement

Step 110

The Java continue statement is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only.

We can use Java continue statement in all types of loops such as for loop, while loop and do-while loop.

SYNTAX

```
jump-statement;
continue;
```

Example Program With Loop

Step 111

```
class Continue
{
public static void main(String[] args)
{
char a ;
for(a=97;a<=105;a++)
{
if (a==102)
{
continue;
}
System.out.println(a);
}
}
}
```

O/P

a
b
c
d
e
g
h
i

Java Continue Statement in while loop

Step 112

```
class continue
{
public static void main(String[] args)
{
char a=97;
while(a<=108)
{
a++;
if (a==102)
{
a++;
continue;
}
System.out.println(a);
}
}
}
```

O/P

b
c
d
e
g
h
i
j
k
l
m

Java Continue Statement in do-while loop

Step 113

```
class continue
{
public static void main(String[] args)
{
char a=97;
do
{

if(a==102)
{

continue;
}
System.out.println(a);
a++;
}
while(a<=108);
}
}
```

O/P

a
b
c
d
e
g
h
i
j
k
l
m

Java Methods(Functions)

Step 114

- A **method** is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a method.
- Methods are used to perform certain actions, and they are also known as **functions**.

Why use methods? To reuse code: define the code once, and use it many times.

Create A Method:

A method must be declared within a class. It is defined with the name of the method, followed by parentheses **()**. Java provides some pre-defined method such asSystem.out.println() ,but you can also create your own method to perform certain action.

How to declare an function

static void Function name()

Or public void Function name()

How to call a function

Using function name we call the function

functionname();

Example

Step 115

Create a method inside the class

```
   class  classname
{
   static void weather()
{
// statement
}
}
```

weather()- it is name of the method.

static - it means that the belongs to the classname class and not an object of the classname class.

void - it means that is return value

CALL A METHOD

Step 116

To call a method in Java, write the method's name followed by two parentheses **()** and a semicolon;

EXAMPLE

```
class classname
 {
   static void weather()
{
    System.out.println("Weather is too hot");
 }
  public static void main(String[] args)
     {
         weather();
}

}
```

O/P

Weather is too hot

Example program

Step 117

```
class example
{
  String a="Micky ";
public  static void main(String[] args)
{
 example obj =new example();
 String b="Milky";
 System.out.println(obj.a+" " +b);
name();// calling a method(Invoked)
}
 static  void  name()//new method
{
System.out.println("Harish micky");
}}
```

O/P

Micky Milky

Harish micky

Example program

```
class Function
{
  public void weather()
{
   System.out.println("Weather is too hot");
}
   public static void main(String[] args)
{

Function a= new Function();
 a.weather();
}
}
```

O/P

Weather is too hot

Java Method Parameters

Step 118

ARGUMENTS AND PARAMETERS

Information can be passed to methods as parameter. Parameters act as variables inside the method.

Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

Example program

Step 119

```
class  Example
{
   static void weather(int celsius)
{
System.out.println(celsius +"  degree celsius");
}
   public static void main(String[] args)
{
   weather(90  );
   weather(75 );
    weather(60);
}
}
```

O/P

90 degree celsius
75 degree celsius
60 degree celcius

Multiple parameters

Step 120

```
class  Example
{
   static void weather(int celsius,String name)
{
System.out.println(celsius +"  degree celsius"+" "+
name);
}
   public static void main(String[] args)
{
   weather(90,"in Thoothukudi"  );
   weather(75 ,"in Kovai");
    weather(60,"in Theni");
}
}
```

O/P

90 degree celsius in Thoothukudi
75 degree celsius in Kovai
60 degree celcius in Theni

Return value

Step 121

The void keyword indicates that the method should not return a value. If you want method to return a value ,you can use primitive data types(int,char,etc). Instead of void ,and use the return keyword inside the method.

EXAMPLE

```
class Example
{
static int num(int x,int y)
{
return x*y;
}
public static void main(String[] args)
{
System.out.println(num(56,89));
}
}
```

O/P

4984

A Method With If- Else

Step 122

```
class Example
{
static void weather(int num)
{
if (num>90)
{
System.out.println("The sun is too hot");
}
else if(num <90)
{
System.out.println("The sun is cloudy");
}
else
{
 System.out.println("The Weather is not bad");
}
}
public static void main(String[] args)
{
 weather(89);
}
}
```

O/P

The sun is cloudy

Method Overloading

Step 123

With **method overloading**, multiple methods can have the same name with different parameters

EXAMPLE

int weather(int x)	Weather-Method
float weather(float x)	intx,floatx,char- different parameters
char weather(char x)	

Example Program

Step 124

```
class Example
{
static int num(int x,int y)
{
return x*y;
}
static double num(double x,double y)
{
 return x+y;
}
public static void main(String[] args)
{
int obj1= num(20,15);
double obj2 = num(7.9 ,12.5);
  System.out.println("Integer number is"+" " +  obj1);
  System.out.println("Double number is"+ " " +  obj2);
}
}
```

O/P

Integer number is 300
Double number is 20.4

Java Array

Step 125

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with square brackets

SYNTAX

dataType[] arr; (or)

dataType []arr; (or)

dataType arr[];

1. String[] Chocolates;
2. String[] Chocolates={"Milkybar","DiaryMilk","Kitkat","Munch","5Star"};
3. For integer to create

int[] numbers={10.20,30,40,50};

Access the Elements of an Array

Step 126

You access an array element by referring to the index number.

For example

```
class Choki
   {
public static void main(String[] args)
{
 String[] Chocolates={"Milkybar","DiaryMilk","Kitkat","Munch","5Star"}
System.out.println(Chocolates[1]);
}}
```

O/P

Diary Milk

Change An Array Element

Step 127

To change the value of a specific element, refer to the index number

For example

Chocolates[1]="Temptation";

Example Program

```
class Choki
    {
public static void main(String[] args)
{
 String[] Chocolates={"Milkybar","DiaryMilk","Kitkat","Munch","5Star"};
Chocolates[1]="Temptation";
System.out.println(Chocolates[1);
}}
```

O/P

Temptation

Array Length

Step 128

To find out how many elements an array has, use the length property.

For example

```
System.out.println(Chocolates.length);
```

O/P

The Length is 5

Example Program

```
class Choki
    {
public static void main(String[] args)
{
 String[] Chocolates={"Milkybar","DiaryMilk","Kitkat","Munch","5Star"};
System.out.println( "The Length is " + Chocolates.length);
}}
```

Loop Through An Array

Step 129

You can loop through the array element with the for loop and use the length property to specify how many times the loop should run.

For example

```
String[] Chocolates={"Milkybar","DiaryMilk","Kitkat","Munch","5Star"};

        for(int i= 0;i<Chocolates.length;i++)
```

Example Program

Step 130

Example Program

```
class Choki
    {
public static void main(String[] args)
{
 String[] Chocolates={"Milkybar","DiaryMilk","Kitkat","Munch","5Star"};
for(int i=0;i<Chocolates.length;i++)
{
System.out.println( Chocolates[i]);
}
}
}
```

O/P

Milkybar
DiaryMilk
KitKat
Munch
5Star

Step 131

Loop Through an Array with For-Each

There is also a "**for-each**" loop, which is used exclusively to loop through elements in arrays.

SYNTAX

```
for (type variable : arrayname)
{
    ……..
}
```

Example Program

Step 132

Example Program

```
class Choki
    {
public static void main(String[] args)
{
 String[] Chocolates={"Milkybar","DiaryMilk","Kitkat","Munch","5Star"};
for(String s  :  Chocolates)
{
System.out.println( s);
}
}
}
```

O/P

Milkybar
DiaryMilk
KitKat
Munch
5Star

Types of Array in java

Step 133

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Instantiation of an Array in Java

```
arrayRefVar=new datatype[size];
```

Step 134

Single Dimensional Array

One-dimensional array or Single dimensional in Java programming is an **array** with a bunch of values having been declared with a **single** index.

EXAMPLE

```
class kitty{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
```

O/P

```
10
20
70
40
50
```

Multidimensional Arrays

Step 135

A multidimensional array is an array containing one or more arrays.

To create a two-dimensional array, add each array within its own set of curly braces.

For example

```
String [] [] Chocolates={ {"Milky bar", "kitkat"}, {"Diary Milk", "Munch"} };
String a= Chocolates[1][1];
```

EXAMPLE PROGRAM

Step 136

```
class MyClass
{
  public static void main(String[]  args)
  {
    String Chocolates[][]= {{"Milkybar","Eclaris"},{"DiaryMilk","KitKat"}};
    for(int i=0;i<2;i++)
    {
      for(int j=0;j<2;j++)
      {
        System.out.print(Chocolates[i][j]+"  ");
      }

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

O/P

MilkyBar Eclaris
DiaryMilk Kitkat

Matrix Program

Step 137

```
public class MatrixMultiplicationExample{
public static void main(String args[]){
//creating two matrices
int a[][]={{1,1,1},{2,2,2},{3,3,3}};
int b[][]={{1,1,1},{2,2,2},{3,3,3}};

//creating another matrix to store the multiplication of two matric
es
int c[][]=new int[3][3];  //3 rows and 3 columns

//multiplying and printing multiplication of 2 matrices
for(int i=0;i<3;i++){
for(int j=0;j<3;j++){
c[i][j]=0;
for(int k=0;k<3;k++)
{
c[i][j]+=a[i][k]*b[k][j];
}//end of k loop
System.out.print(c[i][j]+" ");  //printing matrix element
}//end of j loop
System.out.println();//new line
}
}}
```

O/P

6 6 6
12 12 12
18 18 18

Step 138

Java OOPS Concept

Step 139

What is OOP?

OOP stands for **Object-Oriented Programming**.

Procedural programming is about writing procedures or methods that perform operations on the data.

object-oriented programming is about creating objects that contain both data and methods.

Object-oriented programming has several advantages over procedural programming:

- ✔ OOPS is faster and easier to execute
- ✔ OOPS provides a clear structure for the programs
- ✔ OOPS helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- ✔ OOPS makes it possible to create full reusable applications with less code and shorter development time

Java Class And Objects

Step 140

- ✔ Classes and objects are the two main aspects of object-oriented programming.
- ✔ Everything in Java is associated with classes and objects, along with its attributes and methods.
- ✔ For example: in real life, a LactoCalamine is an object. The LactoCalamine has **attributes**, such as weight, rate and color, and **methods**, such as applying and testing
- ✔ A Class is like an object constructor, or a "blueprint" for creating objects.

Class	Object
Cosmetics	Mascara Eye Shadow Eyeliner lipstick

Cont.....

Step 141

So, a class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the variables and methods from the class.

	Class	Object
What	A data type	A variable
Where	Has its own file	Scattered around the project
Why	Defines the structure	Used to implement to logic
Naming Convention	CamelCase (starts with an upper case)	camelCase (starts with an lowercase)
Examples	Country	india
	Book	harleyQuinnAdventures
	Pokeman	pikachu

TODO

Step 142

1. Create a class with Multiple variable
2. Create an Multiple Objects and Call it

Using Multiple Classes:

You can also create an object of a class and access it in another class. This is often used for better organization of classes (one class has all the attributes and methods, while the other class holds the main() method (Code to be executed)).

Example Program

Step 143

```
class  Kitty
{
char s =99;
}
```

This is one Class

```
class  Micky
{
pubic static void main(String[] args)
{
Kitty obj=new Kitty();
System.out.println(obj.s);
}
}
```

This is Another Class

O/P

c

Java Class Attributes

Step 144

Class attributes are **attributes** which are owned by the **class** itself. They will be shared by all the instances of the **class**. Therefore they have the same value for every instance. We define **class attributes** outside of all the methods, usually they are placed at the top, right below the **class** header.

For Example

```
class  Attribute
{
   int x;
   int y;
```

x & y are attributes

Accessing Attributes

Step 145

You can access attributes by creating an object of the class , and by using the dot syntax(.)

For Example

```
        class Attributes
{
    String s ="Milky"
 public static void main(String[] args)
{
    Attributes a=new Attributes();
System.out.println(a.s);
}
}
```

O/P
Milky

Modify Attributes

Step 146

You can also modify attribute values

For Example

```
        class Attributes
{
    String s ="Milky"// given value
 public static void main(String[] args)
{
    Attributes a=new Attributes();
a.s ="Micky";// modified value
System.out.println(a.s);
}
}
```

O/P
Micky

Class Method

Step 147

The methods are declared within a class, and that they are used to perform certain actions.

For Example

```
        class Method
{
     static void newmethod()// user defined method
{
System .out.println("Welcome to Training");
}
   public static void main(String[] args)
{
     newmethod();// method call
}
}
```

O/P
Welcome to Training

Static Vs Non Static

Step 148

Java programs that have either static or public attributes and method

static method - which means that it can be accessed without creating an object of the class

public - which can only be accessed by objects

For Example

```
class Method
{
 static void newmethod()// Static method
{
System .out.println("Welcome to Training");
}
 public static void main(String[] args)
{
 newmethod();
}
}
```

(Without Creating an object to call a method for Static method)

O/P

Welcome to Training

Sample program

Step 149

```
class Method
{
     public void newmethod()// public
{
System .out.println("Welcome to Training");
}
   public static void main(String[] args)
{
Method s = new Method();//object created
    s.newmethod();
}
}
```

(Creating an object to call a method for public)

O/P
Welcome to Training

Java Constructor

Step 150

- ✔ A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created.
- ✔ In other words, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.
- ✔ Every time an object is created using the new() keyword, at least one constructor is called.

Rules for creating Java constructor

- Constructor name must be the same as its class name
- A Constructor must have no explicit return type
- A Java constructor cannot be abstract, static, final, and synchronized.

Example for constructor program

Step 151

```
class Constructor {
   String name;
   int rollnum;
   public Constructor()
   {
      name = "Harish";
      rollnum = 3033;

   }
  public static void main(String[] args) {

    Constructor obj=new Constructor();
    Constructor obj1 =new Constructor();

    System.out.println("Student Name Is" + " " +obj. name);
    System.out.println("Student Rollnum Is" +" "  + obj1.rollnum);

  }
}
```

Create a class attribute

Set the initial value for name & rollnum

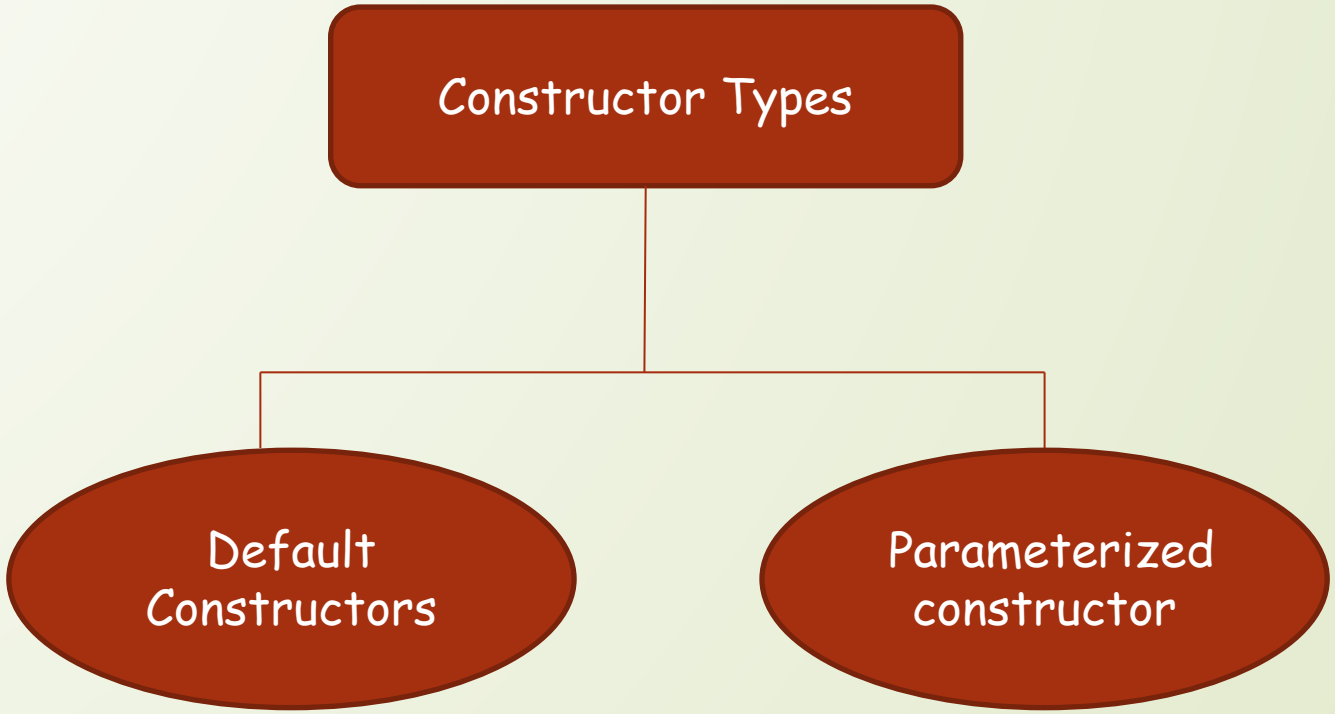
Create an Object(this will call the Constructor)

O/P

Student Name Is Milky
Student Rollnum Is 3033

Types Of Constructor

Step 151



Default Constructor

Step 152

In this constructor, it does not have any parameter. So, it is called Default constructor

Syntax of default constructor

```
<class_name>(){}
```

Example Program

```
class Constructor {
  public Constructor()// default constructor
   {
     System.out.println("I love U Amma");
   }
  public static void main(String[] args) {

    Constructor obj=new Constructor();//calling a default
}}                                                    Constructor
```

O/P

I love u Amma

Modifier

Modifiers are keywords in **Java** that are used to change the meaning of a variable or method.

In **Java**, **modifiers** are categorized into two types:

Access control **modifier** - controls the access level

Non Access **Modifier** - do not control access level, but provides other functionality

Access Control Modifier

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it. There are four types of Java access modifiers

Types of Access modifiers

1. **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected:** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Example for Private Modifier

```
class Modifier
{
private int i=20;// Private Modifier
public static void main(String[] args)
{
Modifier obj= new Modifier();//accessing only
within a class
System.out.println(obj.i);
}
}
```

O/P

20

Example for Default Modifier

```
class Modifier
{
int i=20;// default modifier
public static void main(String[] args)
{
Modifier obj= new Modifier();//accessing only
within a class
System.out.println(obj.i);
}
}
```

O/P

20

Example for Protected Modifier

```
class Modifier
{
Protected String a ="Milky";
Protected String b ="Micky";
Protected String c =  a +  b ;
}
Class  Modify extends Modifier
{
private  int age =23;
public static void main(String[] args)
{
Modify obj= new Modify();
System.out.println(obj.a);
System.out.println(obj.b);
System.out.println(obj.c);
System.out.println(obj.age);
}
}
```

O/P

Milky
Micky
MilkyMicky
23

Example for Public Modifier

```
class Modifier
{
public String a ="Milky";
public String b ="Micky";
publicString c =  a +  b ;
}
```

Save as Modifier.java

```
class  Modify
{
private  int age =23;
public static void main(String[] args)
{
Modifier obj= new Modifier();
Modify obj1= new Modify();
System.out.println(obj.a);
System.out.println(obj.b);
System.out.println(obj.c);
System.out.println(obj1.age);
}}
```

Save as Modify.java

O/P

Milky
Micky
MilkyMicky
23

Non -Access Modifier

```
abstract class Nonaccess
{
        public char  a =97;
        public char b = 98;
         public  char c= 101;
     public   abstract void alpha();// abstract function
   }

   class Mod extends Nonaccess
   {
      public String  e= "welcome";
     public void alpha() // call the abstract function
   {
System.out.println(e);
}
}
class  AccessModifier
{
   public static void main(String[] args)
{
   Mod obj=new Mod();
System.out.println(obj.a);
System.out.println(obj.b);
System.out.println(obj.c);
System.out.println(obj.e);
obj.alpha();
}}
```

Save as Nonaccess.java

Save as AccessModifier.java

O/P

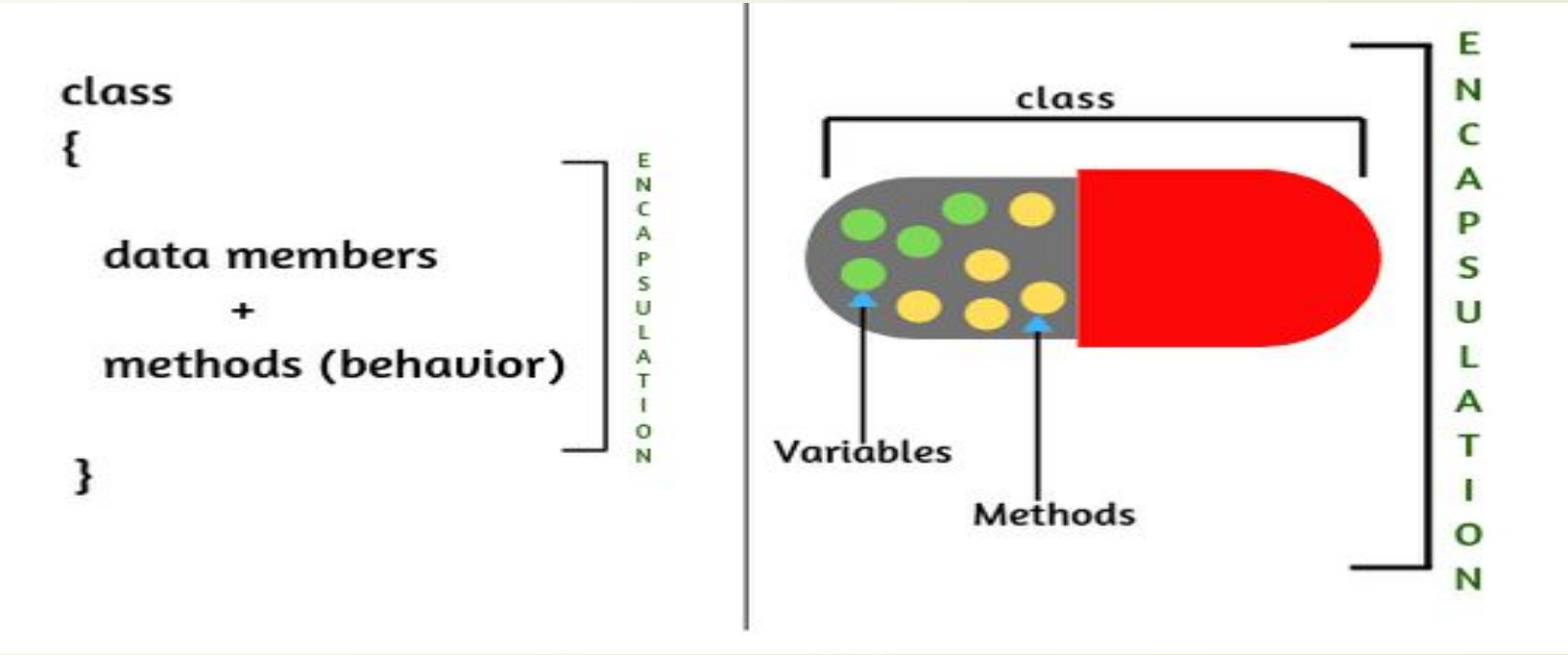
a
b
e
Welcome
Welcome

Encapsulation

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. ... Declare the variables of a class as private

1.declare class variables/attributes as private

2.provide public **get** and **set** methods to access and update the value of a private variable



Get and Set Method

- ✔ In previous concept we learnt about private variables can only be accessed within the same class . However, it is possible to access them if we provide public **get** and **set** methods.
- ✔ To understand **get and set**, it's all related to how variables are passed between different classes.

  i)The **get** method is used to obtain or **retrieve** a particular variable value from a class.

  ii)A **set** value is used to store the variables.

  iii)The whole point of the **get and set** is to **retrieve** and store the data values accordingly

Example Program

```
class Encapsulation
{
   private  int value;
  public int  getValue()// get method
{
      return value;
 }
    public void setValue(int  newValue)// set method
{
   this.value  =newValue;
}
}
   class Encap
{
    public static void main(String[] args)
{
     Encapsulation  obj = new  Encapsulation;
     obj,setValue(10);
 System.out.println(obj.getValue());
}
}
```

Save as Encapsulation.java

Save as Encap.java

O/P

10

Explanation For Program

The **get** method returns the value of the variable Value

The **set** method takes parameter (new value) and assign it to the Value variable. The **this** key word used referred the current object.

However, as the Value variable is declared as private, we **cannot** access it from outside this class

Why Encapsulation?

- Better control of class attributes and methods
- Class attributes can be made **read-only** (if you only use the get method) , or **write only** (if you only use the set method)
- Flexible: the programmer can change one part of the code without affecting other parts
- Increased security of data

Java Packages

- ✔ Package is a collection of classes
- ✔ Package in java can be categorized in two form

  i) built-in package (Pre-defined package)and

  ii) user-defined package.

- ✔ built-in package- java, lang, awt, javax, swing, net, io, util, sql etc.
- ✔ user-defined package- create your own package.

Pre-defined Package

built-in package- java, lang, awt, javax, swing, net, io, util, sql etc.These are all pre-defined package in java.

For Example

```
import java.io.*;
import java.util.Scanner;
```

import → Keyword

java → Base Class(Super Class)

util → Sub Class

Scanner → Class

* → It includes all classes

Example Program

```
package  pack;
  class example
{
   public void string()
{
    String a = " Micky";
    String b =" Kitty";
    String c= a + b;
 System.out.println( "The Cartoon Cat Name Is" + "    "  +  c)
}
}
  class mainexample
{
    public static void main(String[] args)
{
    example obj =new example();
  obj.string();
}
}
```

O/P

The Cartoon Cat Name Is Micky Kitty

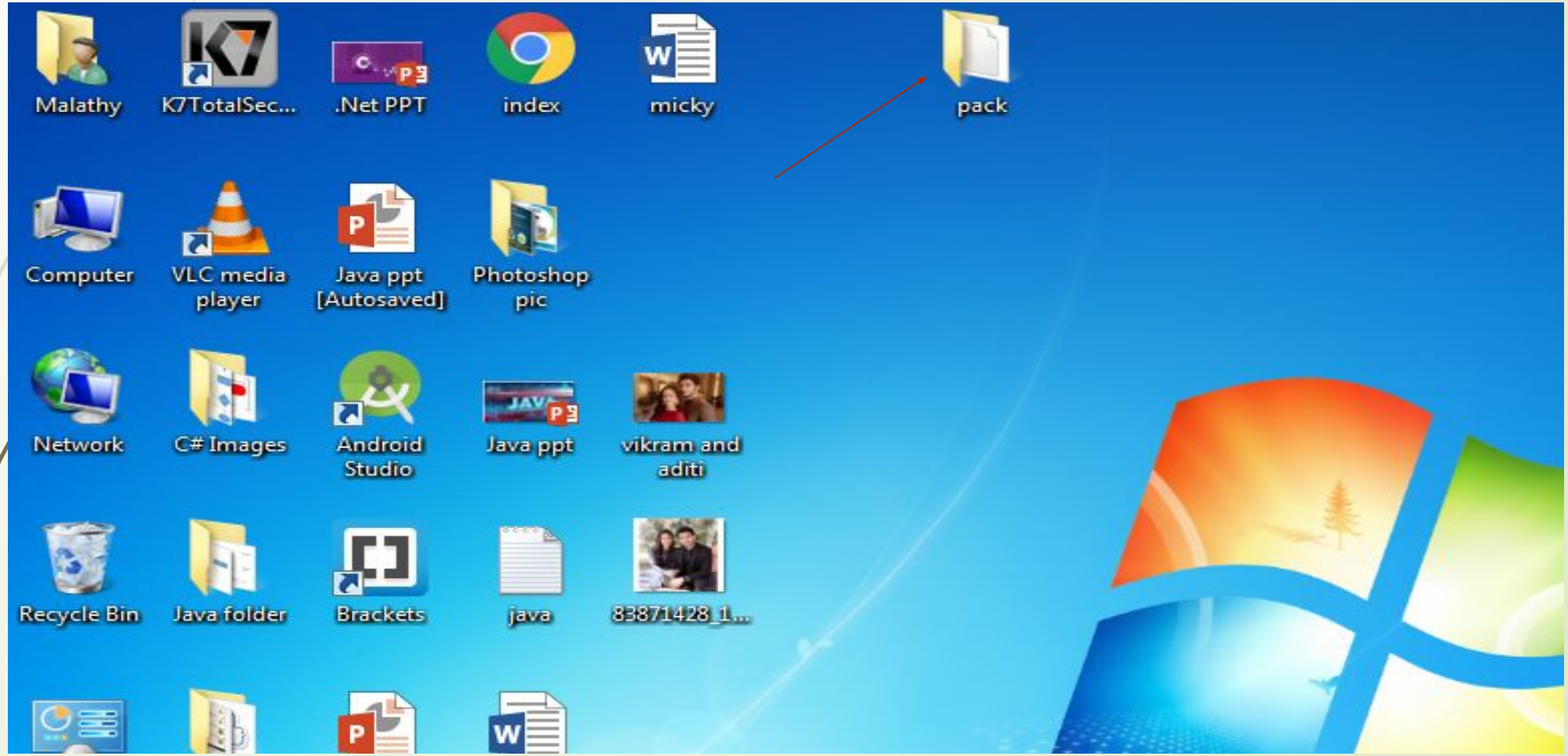
How to save and compile the package program

STEP1:type the program on your Notepad

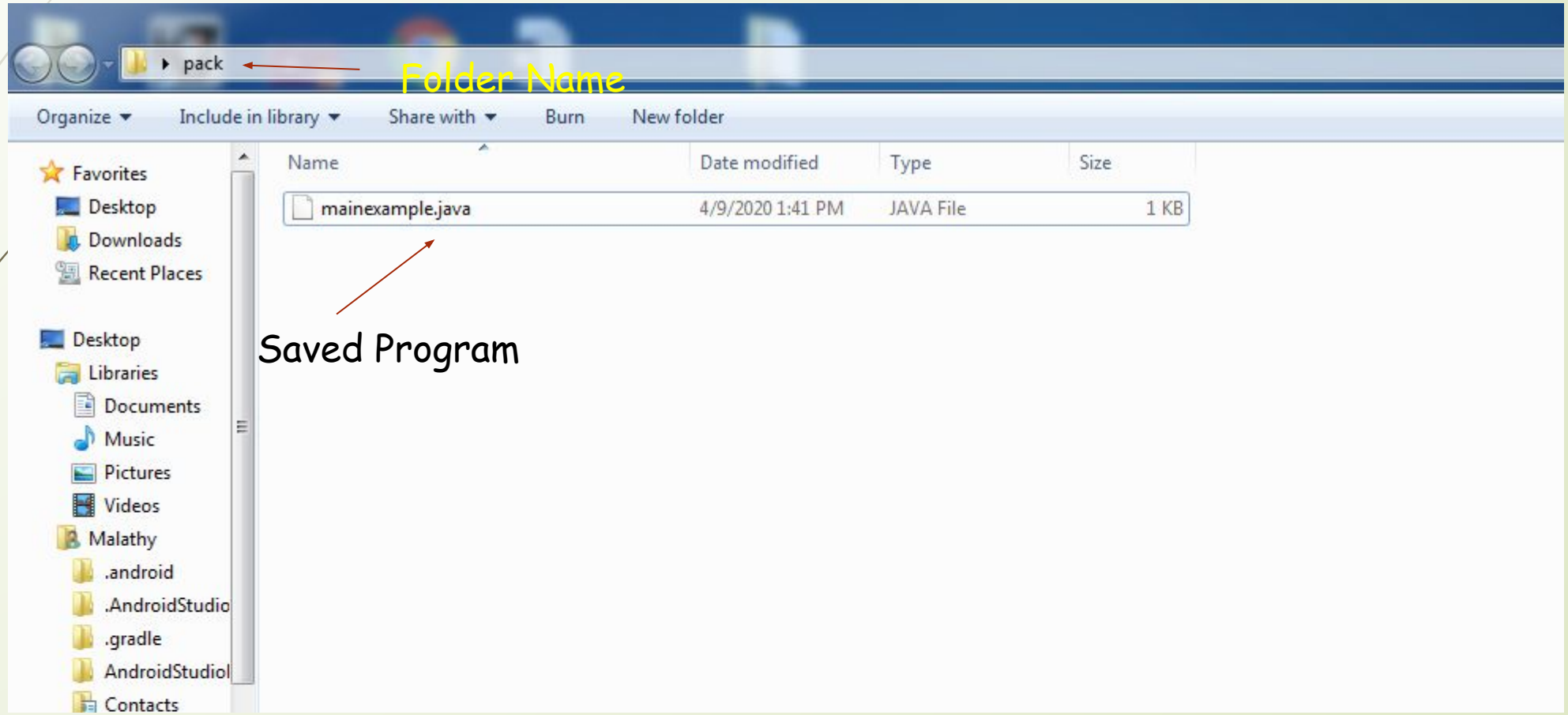
File Edit Format View Help

```
package  pack;
  class example
{
   public void string()
{
    String a = " Micky";
    String b =" Kitty ";
    String c= a + b;
 System.out.println( "The Cartoon Cat Name Is" + "    "  +  c);
}
}
  class mainexample
{
    public static void main(String[] args)
{
    example obj =new example();
  obj.string();
}
}
```

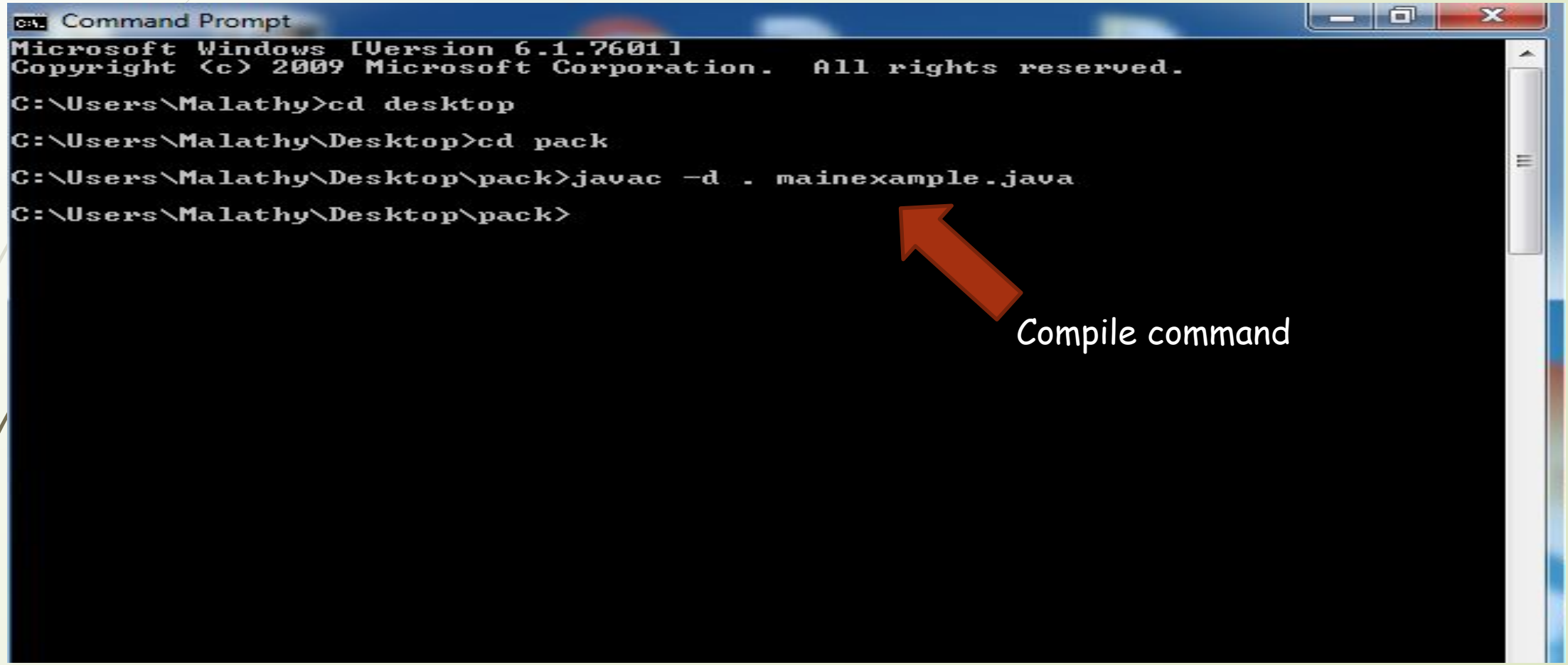
STEP2:Create a folder on your desktop and give any name to that folder.



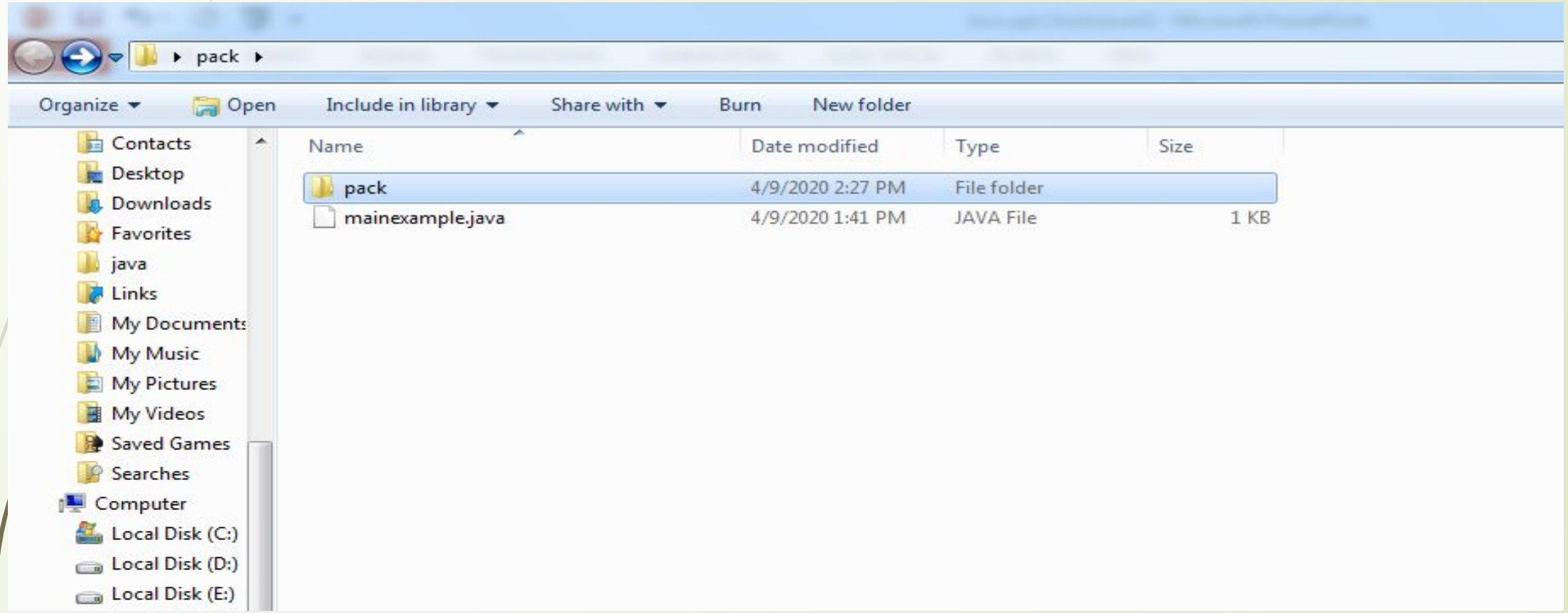
STEP3: save your program on that folder



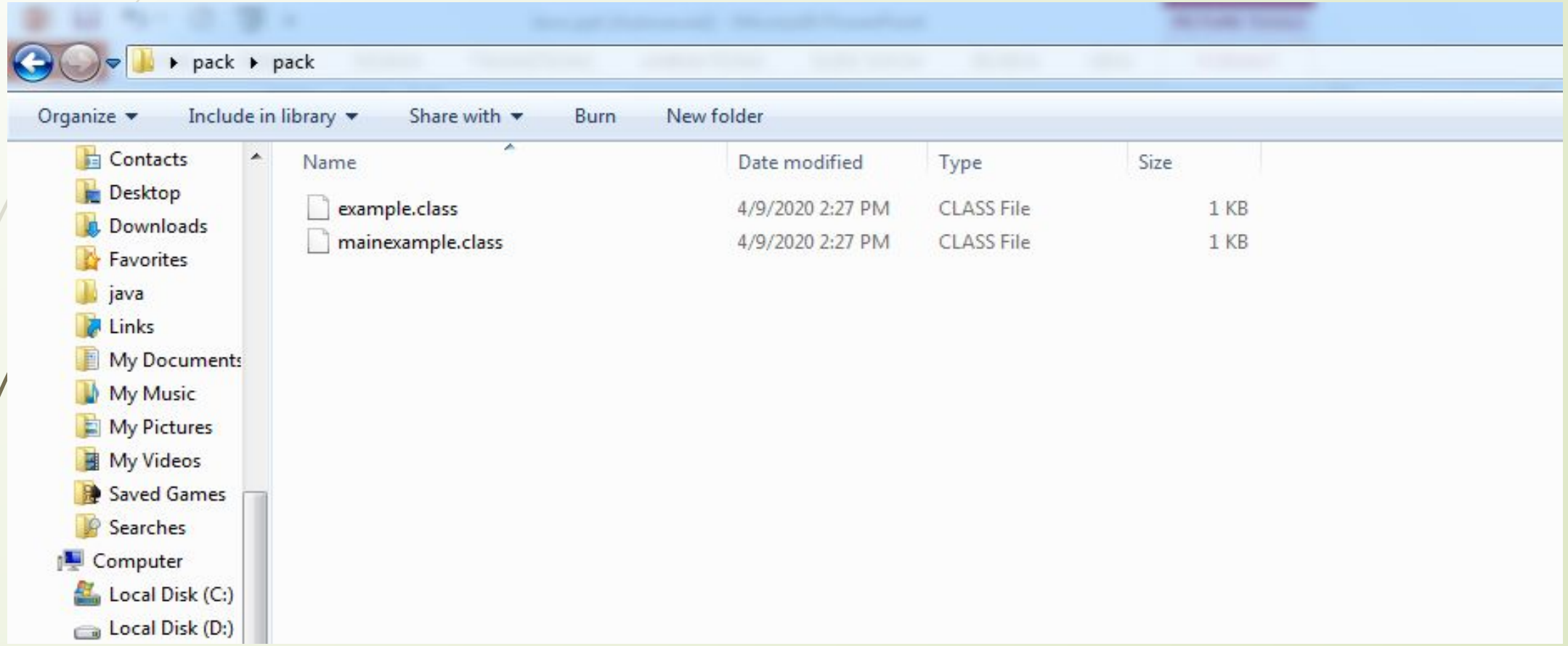
STEP4: Open the cmd and compile your program



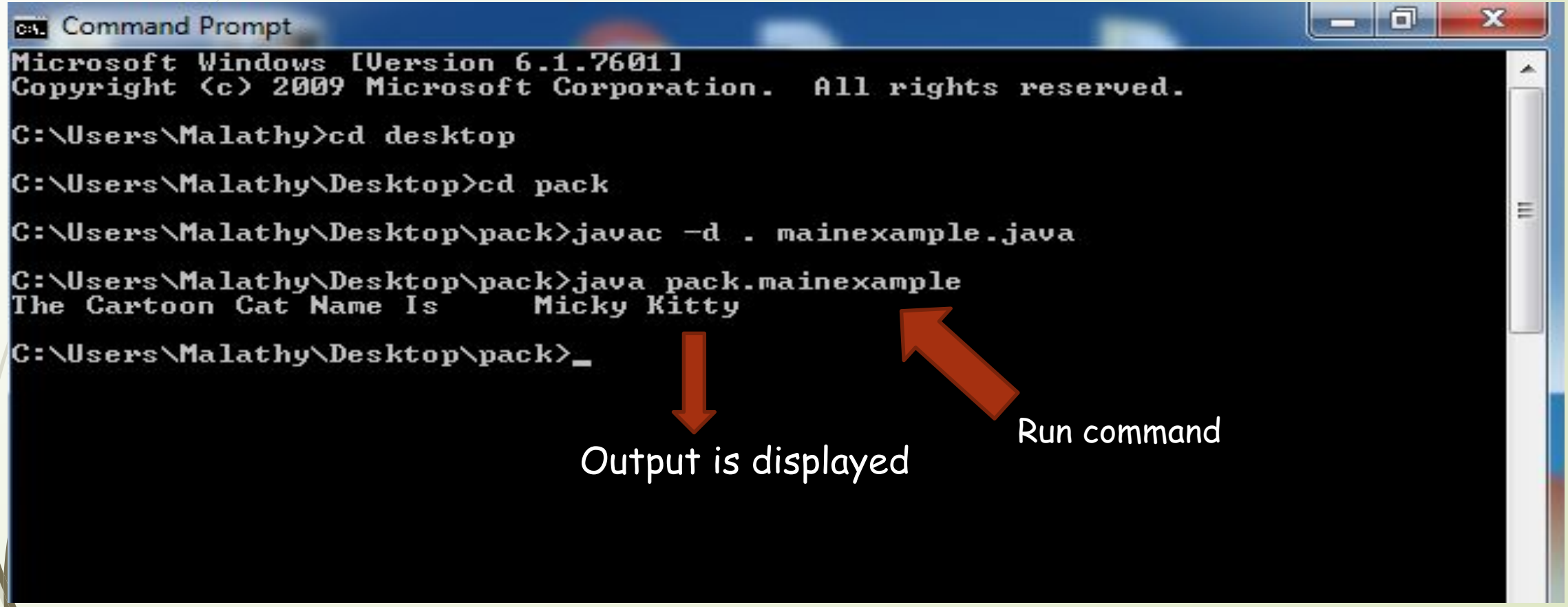
STEP5: After the compilation process see your folder one package will be created.



If you open that new package folder the two class will be created



STEP6: Run the program



Java Inheritance

In Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:

- **subclass** (child) - the class that inherits from another class
- **superclass** (parent) - the class being inherited from

To inherit from a class, use the extends keyword

Example program

```
class Inheritance
{
Protected String a ="Milky";
Protected String b ="Micky";
Protected String c =  a +  b ;
}
Class  Inherited extends Inheritance
{
private  int age =23;
public static void main(String[] args)
{
Inheited obj= new Inherited();
System.out.println(obj.a);
System.out.println(obj.b);
System.out.println(obj.c);
System.out.println(obj.age);
}
}
```

O/P

Milky
Micky
MilkyMicky
23

Java Polymorphism

POLYMORPHISM - "Many Forms"

Poly → "Many"

Morph → "Forms"

In that ,it says about many same method and different parameters is called polymorphism

In polymorphism ,it contain two different that is

1. Compile Time Polymorphism : (Method overloading)
2. Run Time Polymorphism : (Method overriding)

Method Overloading Vs Method Overriding

```
class Animal {
        public void add( ) {
                // Your code
        }

        public void add( int a, int b) {
                // Your code
        }

        public int add( int a, int b, int c ) {
                // Your code
        }
        // Method Overloading.....
```

```
class Animal {
        public void add( ) {
                // Your Animal code
        }
}

class Dog extends Animal {
        public void add( ) {
                // Your Modified Dog code
        }
}

// Method Overriding .....
```

Small Difference Between Method overloading & Riding

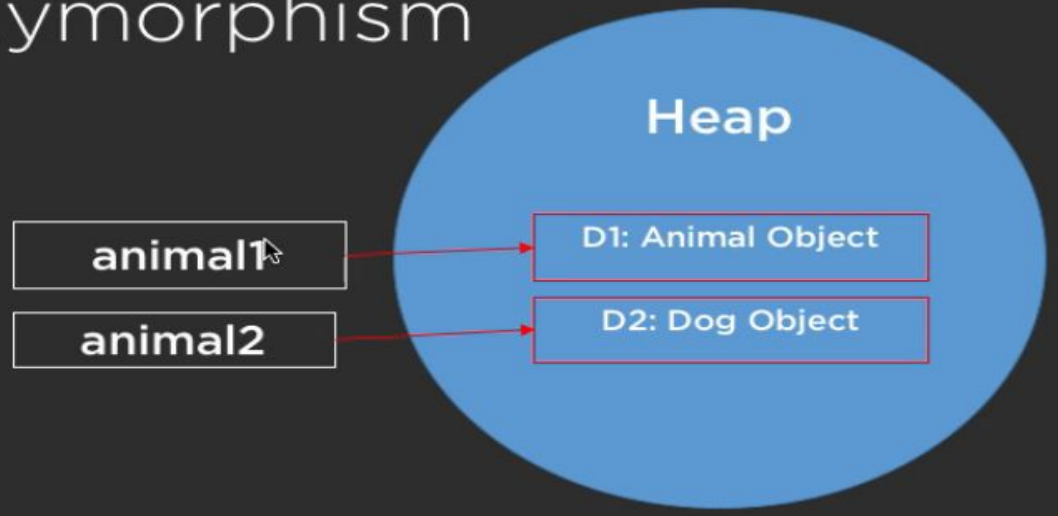
Difference between Overloading and Overriding

Overloading	Overriding
Occurs within ONE class	Occurs in TWO classes : Super class and sub class i.e. Inheritance is involved.
Name of the method is same but parameters are different	Name and Parameters both are same.
Purpose: Increases Readability of Program	Purpose: Use the method in the Child class which is already present in Parent class.
Return type can be same or different	Return type is always same.
It is an example of Compile time Polymorphism	It is an example of Run time Polymorphism.
Which method to call is decided by Compiler	Method call is decided by JVM in run time.

Run Time Polymorphism

Runtime Polymorphism

```
class Animal {
        public void eat() {
                print "Animal Eating";
        }
}
class Dog extends Animal {
        public void eat() {      // Overriding
                print "Dog Eating";
        }
}
1. Animal animal1 = new Animal();    // D1
   animal1.eat();    // "Animal Eating"
2. Animal animal2 = new Dog();   // D2  :UPCASTING
   animal2.eat();   // "Dog Eating"
3. Dog dog = new Animal();           // INVALID
```



RULES of Polymorphism

1. Reference variable → Own class Object
2. Reference variable → Child class Object // UPCASTING
3. Reference variable CANNOT point to Parent class Object

Example Program for Compile Time Polymorphism

```
class Example
{
static int num(int x,int y)
{
return x*y;
}
static double num(double x,double y)
{
 return x+y;
}
public static void main(String[] args)
{
int obj1= num(20,15);
double obj2 = num(7.9 ,12.5);
  System.out.println("Integer number is"+" " +  obj1);
  System.out.println("Double number is"+ " " +  obj2);
}
}
```

O/P

Integer number is 300
Double number is 20.4

Example Program for Run Time Polymorphism

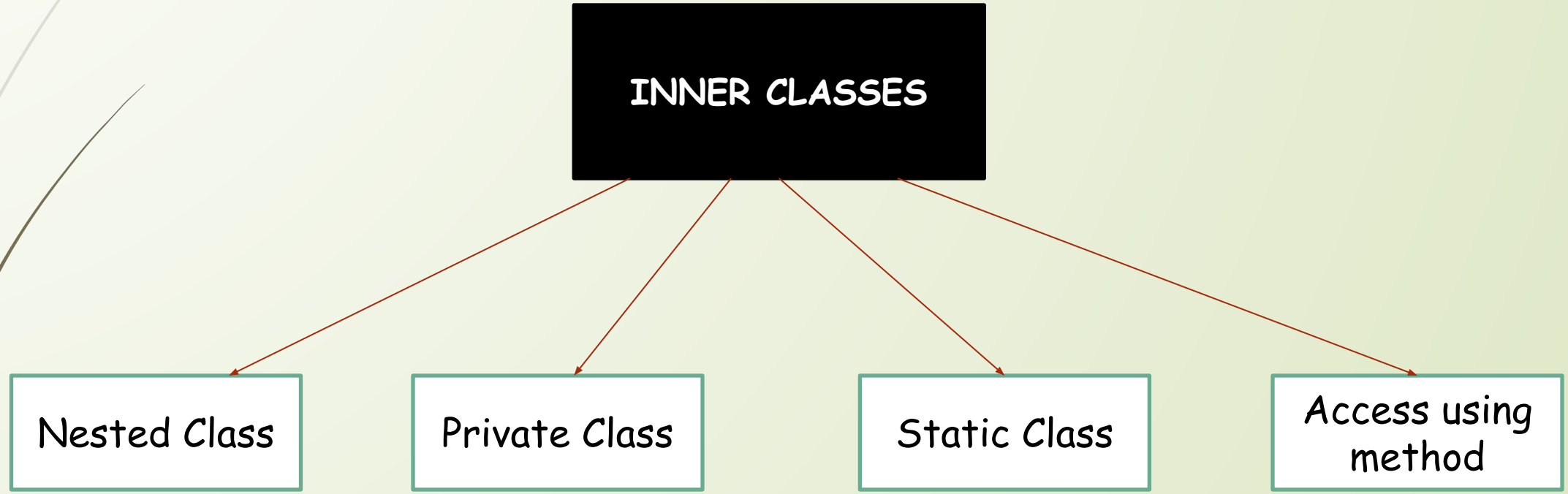
```
class Food
{
  public void menu ()
  {
  System.out.println( "  Different  types of food");
  }}
class Biriyani extends Food
  {
    public void menu()
    {
      System.out.println("Order: Chicken Biriyani");
    }}
class FriedRice extends Food
    {
      public void menu()

      {
        System .out.println(" order: Chicken FriedRice");
      }}
class mainFood
      {
        public static void main (String[] args)
        {
          Food obj1 = new Food();
          Food obj2 = new Biriyani();
          Food obj3 = new FriedRice();
          obj1.menu();
          obj2.menu();
          obj3.menu();
        }}
```

O/P

Different types of food
Order: Chicken Biriyani
Order: Chicken FriedRice

Java Inner Classes

In Java, it is also possible to nest classes (a class within a class). The purpose of nested classes is to group classes that belong together, which makes your code more readable and maintainable.



Example Program for Nested Class

```
class Chocolates// Outer class
{
   String s="Diary Milk";
   class  Choki// Inner Class
{
    String M="Milky Bar";
}
}
 class MainChoco
{
   public static void main(String[] args)
{
    Chocolates obj=new Chocolates();
Chocolates.Choki obj1= obj.new Choki();// Creating Object for inner class
System.out.println("My Favorite Chocolates Name Is :" +" " + obj.s +" "+ " & " +" "  + obj1.M);
}
}
```

O/P

My Favorite Chocolates Name Is: Diary Milk & Milky Bar

Example Program Using Access Modifier

```
class Chocolates
{
   String s="Diary Milk";
  private class  Choki // if we use private in  inner class that will be display as error
{
String M="Milky Bar";
}
}
 class MainChoco
{
   public static void main(String[] args)
{
    Chocolates obj=new Chocolates();
Chocolates.Choki obj1= obj.new Choki();
System.out.println("My Favorite Chocolates Name Is :" +" " + obj.s +" "+ " & " +" "  + obj1.M);
}
}
```

Using Static keyword

```
class Chocolates
{
    String s="Diary Milk";
   static class  Choki
{
     String M="Milky Bar";
}
}
 class MainChoco
{
   public static void main(String[] args)
{
    Chocolates.Choki obj=new Chocolates.Choki();

System.out.println("My Favourite Chocolates Name Is
:"+ obj.M);
}
}
```

O/P

My Favorite Chocolates Name Is: Milky Bar

Access Outer Class From Inner Class

```
class Chocolates// Outer class
{
   String s="Diary Milk";
   class  Choki // Inner Class
{
    public String Chocki()// creating Method for
                                    inner class
{
    return s;
}
}
}
 class MainChoco
{
   public static void main(String[] args)
{
    Chocolates obj=new Chocolates();
Chocolates.Choki obj1= obj.new Choki();// Creating
Object for inner class
System.out.println("My Favorite Chocolates Name Is :" +
obj1.Chocki() );
}}
```

O/P

My Favorite Chocolates Name Is: Diary Milk

Java Abstraction

- ✔ Data **abstraction** is the process of hiding certain details and showing only essential information to the user.
- ✔ Abstraction can be achieved with either **abstract classes** or **interfaces**
- ✔ The **abstract** keyword is a non-access modifier, used for classes and methods
- ✔ **Abstract class**: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- ✔ **Abstract method**: can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

Example Program

```
abstract class Food
{
  public abstract void menu () ;

 public void show()
{
  System.out.println( "  Different  types of food");
  }}
class Biriyani extends Food
  {
    public void menu()
    {
      System.out.println("Order: Chicken Biriyani");
    }}
class FriedRice extends Food
    {
      public void menu()

      {
        System .out.println(" order: Chicken FriedRice");
      }}
class mainFood
      {
        public static void main (String[] args)
        {

          Food obj1 = new Biriyani();
          Food obj2 = new FriedRice();
             obj1.show();
          obj1.menu();
          obj2.menu();
        }}
```

In abstract class we should create a abstract method

we must access the abstract class using the method in abstract class

O/P

Different types of food
Order: Chicken Biriyani
Order: Chicken FriedRice

Java Interface

- Another way to achieve **abstraction** in Java, is with **interfaces.**
- An **interface** is a completely "**abstract class**" that is used to group related methods with empty bodies.
- In this interface we must use **"implements"** Keyword to access the another classes

For Example

```
    interface Food //interface class
{
  public void menu () ; //no body in   interface method
public void show(); //no body in   interface method
 }
```

Example program

```
interface Food //interface class
{
  public void menu () ; //no body in   interface method
public void show(); //no body in   interface method
 }
class Biriyani implements Food
 {
   public void menu()
   {
     System.out.println("Order: Chicken Biriyani");
   }
public void show()
{
   System.out.println("Show the menu card");
}}

class  mainFood
{
   public static void main(String[] args)
{
     Biriyani obj =new Biriyani();
 obj.show();
obj.menu();
}}
```

O/P
Show the menu card
Order: Chicken Biriyani

Java Enum(Enumerations)

An enum is a special "class" that represents a group of **constants** (unchangeable variables, like final variables).

To create an enum, use the enum keyword (instead of class or interface), and separate the constants with a comma.

Enumerations → Which means "specifically Listed"

For Example

```
enum Chocolates
{
    Diary Milk,
    Kinder Joy,
    Milky Bar,
    Eclairs,
    Temptation  }
```

Separate the constant with Comma

Cont……

You can access enum constants with the **dot** syntax.

For Example

Chocolates obj=new Chocolates .Kinder joy;

Example Program

```
enum Chocolates
{
    DiaryMilk,
    KinderJoy,
    MilkyBar,
     Eclairs,
    Temptation  }
class  mainChoki
{
public static void main (String[] args){
Chocolates obj = Chocolates.KinderJoy;
System.out.println("My Fav Choki Is :" + obj);
}}
```

O/P

My Fav Choki Is : Kinder Joy

Enum Inside A Class

```
class  mainChoki
{
enum Chocolates
{
    DiaryMilk,
    KinderJoy,
    MilkyBar,
     Eclairs,
    Temptation  }
public static void main (String[] args){
 Chocolates obj = Chocolates.KinderJoy;
System.out.println("My Fav Choki Is :" + obj);
}}
```

O/P

My Fav Choki Is : Kinder Joy

Enum in a Switch Statement

Enums are often used in switch statements to check for corresponding values

```
class  mainchoki
{
enum Chocolates
{
    DiaryMilk,
    KinderJoy,
    MilkyBar
      }
public static void main (String[] args){
  Chocolates a = Chocolates.KinderJoy;
switch (a)
{
  case DiaryMilk:
     System.out.println("My Fav Chokii");
  break;
  case KinderJoy:
     System.out.println("Kids Favour Choki");
     break;
  case MilkyBar:
     System.out.println("My Besti fav Choki");
}
}

}
```

O/P

Kids Favour Choki

Java User Input

- The **Scanner** class is used to get user input, and it is found in the java.util package.
- To use the **Scanner** class, create an object of the class and use any of the available methods found in the Scanner class documentation. In our example, we will use the **nextLine()** method which is used to read Strings.

Example Program

```
import java.util.Scanner;
class Chocolates {
  public static void main(String[] args) {
    Scanner obj=new Scanner(System.in);
   String Chokiname;
   System.out.println("Enter Chocolate Name");
    Chokiname=obj.nextLine();
    System.out.println("My Fav Choki Is:"+ Chokiname);
  }
}
```

nextLine() method is only for String

O/P

Enter Chocolates Names
Kinder joy
My Fav Choki Is : Kinder joy

Other Input Types

Method	Description
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
nextLine()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

Another Example program

```
import java.util.Scanner;
class Chocolates {
  public static void main(String[] args) {
     Scanner obj=new Scanner(System.in);
    System.out.println("Enter Chocolate
Name,Rupees,expdate:");
    String Chokiname = obj.nextLine(); //String
    int Rupees = obj.nextInt();// Int
    long expdate = obj.nextLong();// Long
    System.out.println("My Fav Choki Is:"+ Chokiname);
    System.out.println("Chocolate Rs."+ Rupees);
System.out.println("Chocolate expdate is:"+ expdate);
      }}
```

O/P

Enter Chocolates Names,Rupees, expdate
Kinder joy
40
2021
My Fav Choki Is:Kinder joy
Chocolate Rs. 40
Chocolate expdate is:2021

Java Date and Time

Java does not have a built-in Date class, but we can import the java.time package to work with the date and time API. The package includes many date and time classes.

For example

Class	Description
LocalDate	Represents a date (year, month, day (yyyy-MM-dd))
LocalTime	Represents a time (hour, minute, second and milliseconds (HH-mm-ss-zzz))
LocalDateTime	Represents both a date and a time (yyyy-MM-dd-HH-mm-ss.zzz)
DateTimeFormatter	Formatter for displaying and parsing date-time objects

Display Current Date → java.time.LocalDate class , use its now() method
Display Current Time → java.time.LocalTime class ,use its now() method

Example Program Using Date & Time

```
import java.time.LocalDateTime;
class  Display
{
   public static void main(String[] args)
{
  LocalDateTime obj= LocalDateTime.now();// It Display The Current time
System.out.println("Current Date And Time Is:"+ obj);
}
}
```

Formatting Date and Time

- The "T" in the example above is used to separate the date from the time. You can use the **DateTimeFormatter** class with the ofPattern() method in the same package to format or parse date-time objects.
- The following example will remove both the "T" and milliseconds from the date-time

Example Program

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
 class Display{
public static void main(String[] args){
   LocalDateTime obj= LocalDateTime.now();
 System.out.println("Before Formatting"+ obj);
DateTimeFormatter obj1=DateTimeFormatter.ofPattern( "dd-MM-YY  HH:mm:ss");
String Format=obj.format(obj1);
System.out.println("After Formatting"+ Format);
    }}
```

It display the current Formatting

Java ArrayList

1. The ArrayList class is a resizable array, which can be found in the java.util packages.
2. The difference between Array and Array List is

   i) Array -that the size of an array cannot be modified(if you want to add or remove elements to/from an array, you have to create a new one).

   ii) ArrayList- In this ,elements can be added and removed whenever you want.
3. This is small difference between array and arraylist ,the Syntax also slightly different.

Example for creating object in arraylist

```
import java.util.ArrayList;
ArrayList<String>= new ArrayList<String>();// creating object
```

Different Methods In ArrayList

1. Add an items use add() method

2.Access an item use get() method

3.Change an item use set() method

4. Remove an item use remove() method

5. Remove all item use clear() method

6.ArrayList Size use size() method

Creating a new program by using this all method from ArrayList

Example Program By Using method

```
import java.util.ArrayList;

public class Chocolates {
  public static void main(String[] args) {
    ArrayList<String> Chokies = new ArrayList<String>();
    Chokies.add("Diary Milk");
   Chokies.add("Milky Bar");
    Chokies.add("Kinder joy");
    Chokies.add("Temptation");
    System.out.println(Chokies);
    System.out.println(Chokies.size());// size() method
              Chokies.set(3,"Eclaris");   //set() method
    System.out.println(Chokies);
    System.out.println(Chokies.get(3));// get() method
                        Chokies.remove(2);  //remove() method
    System.out.println(Chokies);
                              Chokies.clear(); // clear() method
     System.out.println(Chokies);
  }}
```

Add () method

O/P

```
[Diary Milk, Milky Bar, Kinder joy, Temptation]
4
[Diary Milk, Milky Bar, Kinder joy, Eclaris]
Eclaris
[Diary Milk, Milky Bar, Eclaris]
[]
```

Hash Map

1. In previous topic , that Arrays store items as an ordered collection, and you have to access them with an index number (int type) A HashMap however, store items in "**key/value**" pairs, and you can access them by an index of another type (e.g. a String).
2. One object is used as a key (index) to another object (value). It can store different types:String keys and Value or the same type, like: String Key and String Values.

Example for creating a object in HashMap:

```
HashMap<String,String> class name= new HashMap <String,String>();
```

Difference Between ArrayList and HashMap

ArrayList	HashMap
O(n) insertion time	O(1) insertion time
Ordered by insertion sequence	Unordered.
Can contain duplicate entries	Cannot have duplicate entry of Key
Access to element by index number	Access to element by Key
Example Use Case: Shopping Cart	Example Use Case: Phonebook (Name, Number)

Different Methods In HashMap

1. Add an items use put() method

2.Access an item use get() method

3. Remove an item use remove() method

4. Remove all item use clear() method

5.ArrayList Size use size() method

Creating a new program by using this all method from HashMap

Example Program By Using method

```
import java.util.HashMap;

public class Chocolates {
  public static void main(String[] args) {
    HashMap<String,Integer> Chokies = new HashMap<String,Integer>();
    Chokies.put("Diary Milk",60);
   Chokies.put("Milky Bar",20);
    Chokies.put("Kinder joy",40);
    Chokies.put("Temptation",80);
    System.out.println(Chokies.size());//
    System.out.println(Chokies.get(3));//
                    Chokies.remove(2);  //
    for (String s : Chokies.keySet()) {
    System.out.println("Chokies Names:"+ s +"    "+ "Rs." +
Chokies.get(s));

  }}}
```

put() method

size() method

get() method

remove() method

```
4
null
Chokies Names:Milky Bar    Rs.20
Chokies Names:Kinder joy    Rs.40
Chokies Names:Diary Milk    Rs.60
Chokies Names:Temptation    Rs.80
```

Exception Handling

Java File Handling

Java File

- File Handling is an important part of any application
- Java has several method for creating, reading, updating, and deleting files

Java File Handling

- The File class from the java.io package, allows us to work with files.
- To use the File class, create an object of the class,and specify the filename or directory name.

For Example

```
import java .io.File ;// Import the File class
File obj = new File("filename.txt");// Specify the filename
```

The File class has many useful methods for creating and getting information about files. For example:

Method	Type	Description
canRead()	Boolean	Tests whether the file is readable or not
canWrite()	Boolean	Tests whether the file is writable or not
createNewFile()	Boolean	Creates an empty file
delete()	Boolean	Deletes a file
exists()	Boolean	Tests whether the file exists
getName()	String	Returns the name of the file
getAbsolutePath()	String	Returns the absolute pathname of the file
length()	Long	Returns the size of the file in bytes
list()	String[]	Returns an array of the files in the directory
mkdir()	Boolean	Creates a directory