**JAVA INTRODUCTION**

Java programming language was originally developed by Sun Microsystems which was initiated by **James Gosling** and released in 1995 as core component of Sun Microsystems'

With the advancement of Java

and its widespread popularity, multiple configurations were built to suit various types of platforms.

For example: J2EE for Enterprise Applications, J2ME for Mobile Applications.

The new J2 versions were renamed as Java SE, Java EE, and Java ME respectively.

Java is guaranteed to be Write Once, Run Anywhere(WORA).

**Java is:**

**Object Oriented:** In Java, everything is an Object. Java can be easily extended

since it is based on the Object model.

**Platform Independent:** Unlike many other programming languages including C

and C++, when Java is compiled, it is not compiled into platform specific machine,

rather into platform independent byte code.

This byte code is distributed over the

web and interpreted by the Virtual Machine (JVM) on whichever platform it is being

run on.

**Simple**: Java is designed to be easy to learn. If you understand the basic concept

of OOP Java, it would be easy to master.

**Secure:** With Java's secure feature it enables to develop virus-free, tamper-free

systems. Authentication techniques are based on public-key encryption.

**Architecture-neutral**: Java compiler generates an architecture-neutral object

file format, which makes the compiled code executable on many processors, with

the presence of Java runtime system.

**Portable:** Being architecture-neutral and having no implementation dependent

aspects of the specification makes Java portable. Compiler in Java is written in

ANSI C with a clean portability boundary, which is a POSIX subset.

**Robust**: Java makes an effort to eliminate error prone situations by emphasizing

mainly on compile time error checking and runtime checking.

**Multithreaded**: With Java's multithreaded feature it is possible to write programs

that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.

**Interpreted**: Java byte code is translated on the fly to native machine instructions and is not stored anywhere.

The development process is more rapid

and analytical since the linking is an incremental and light-weight process.

**High Performance**: With the use of Just-In-Time compilers, Java enables high

Performance.

**Distributed**: Java is designed for the distributed environment of the internet.

**Dynamic**: Java is considered to be more dynamic than C or C++ since it is

designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on runtime.

------------------------------

**History of Java**

James Gosling initiated Java language project in **June 1991** for use in one of his many settop box projects.

**The language, initially called ‘Oak’ after an oak tree that stood outside**

**Gosling's office,** also went by the name ‘Green’ and ended up later being renamed as Java, from a list of random words.

Sun released the first public implementation as Java 1.0 in 1995.

It promised Write Once, Run Anywhere (WORA), providing no-cost run-times on popular platforms.

On 13 November, 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL).

On 8 May, 2007, Sun finished the process, making all of Java's core code free and opensource, aside from a small portion of code to which Sun did not hold the copyright.

**What is a compiler?**

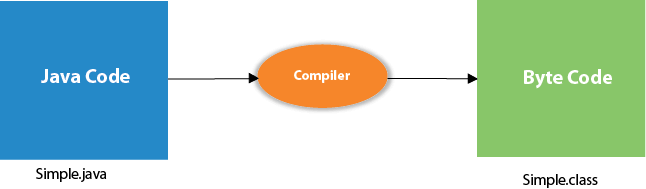
A compiler is a **special program** or TRANSLATOR that translates a programming language's source code into machine code, bytecode or another programming language.

The source code is typically written in a high-level, human-readable language such as Java or C++.

A programmer writes the source code in a code editor or an integrated development environment (IDE) that includes an editor, saving the source code to one or more text files.

A compiler that supports the source programming language reads the files, analyzes the code, and translates it into a format suitable for the target platform.

Some compilers can translate source code to bytecode instead machine code. Bytecode, which was first introduced in the Java programming language, is an intermediate language that can be executed on any system platform running a Java virtual machine (JVM) or bytecode interpreter. The JVM or interpreter converts the bytecode into instructions that can be executed by the hardware processor.



------------------------------

**JDK JRE AND JVM**

**JDK**

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.

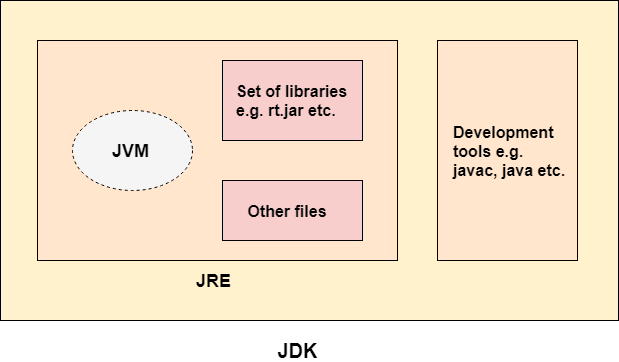
JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

**Standard Edition Java Platform**

**Enterprise Edition Java Platform**

**Micro Edition Java Platform**

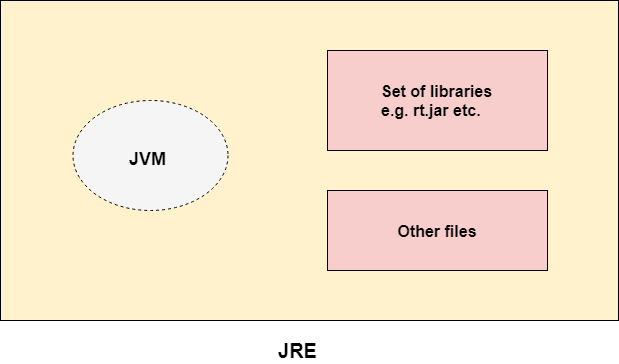
The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.



**JRE**

JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

The implementation of JVM is also actively released by other companies besides Sun Micro Systems.



**JVM**

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each OS is different from each other. However, Java is platform independent. There are three notions of the JVM: specification, implementation, and instance.

**The JVM performs the following main tasks:**

*Loads code*

*Verifies code*

*Executes code*

*Provides runtime environment*

----------------------------

**Java basic terminology**

When we consider a Java program, it can be defined as a collection of objects that

communicate via invoking each other's methods. Let us now briefly look into what do class, object, methods, and instance variables mean.

**Object** - Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behavior such as wagging their tail, barking, eating. An object is an instance of a class.

**Class** - A class can be defined as a template/blueprint that describes the

behavior/state that the object of its type supports.

**Methods** - A method is basically a behavior. A class can contain many methods.

It is in methods where the logics are written, data is manipulated and all the

actions are executed.

**Instance Variables** - Each object has its unique set of instance variables. An

object's state is created by the values assigned to these instance variables.

**First Java Program**

**class MyFirstJavaProgram**

**{**

**/\* This is my first java program.**

**\* This will print 'Hello World' as the output**

**\*/**

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

**{**

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

**// prints Hello World**

**}**

**}**

**Explanation:**

**class** keyword is used to declare a class in Java.

**public** keyword is an access modifier that represents visibility. It means it is visible to all.

**static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method.

**The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.**

**void** is the return type of the method. It means it doesn't return any value.

**main** represents the starting point of the program.

**String[] args or String args[]** is used for command line argument.more details coming lectures.

**System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.

**Basic Syntax**

About Java programs, it is very important to keep in mind the following points.

**Case Sensitivity**

- Java is case sensitive, which

means identifier Hello and hello would have different meaning in Java.

**Class Names** - For all class names the first letter should be in Upper Case.

If several words are used to form a name of the class, each inner word's first letter

should be in Upper Case.

Example: class MyFirstJavaClass

**Method Names** - All method names should start with a Lower Case letter.

If several words are used to form the name of the method, then each inner word's

first letter should be in Upper Case.

Example: public void myMethodName()

**Program File Name** - Name of the program file should exactly match the class name.

When saving the file, you should save it using the class name (Remember Java is

case sensitive) and append '.java' to the end of the name (if the file name and the

class name do not match,

your program will not compile).

Example: Assume 'MyFirstJavaProgram' is the class name. Then the file should

be saved as 'MyFirstJavaProgram.java'

**public static void main(String args[])** - Java program processing starts from

the main() method which is a mandatory part of every Java program.

---------------------------------------

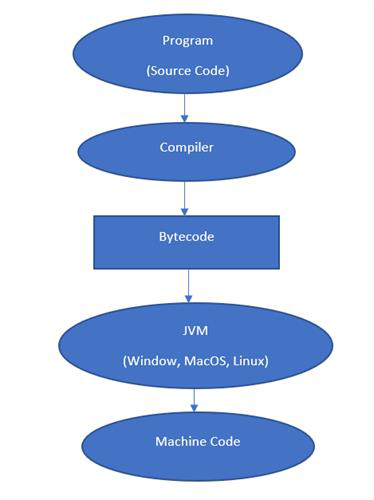
**Java bytecode**

Byte Code can be defined as an intermediate code generated by the compiler after the compilation of source code(JAVA Program).

This intermediate code makes Java a platform-independent language.

**How is Byte Code generated?**

Compiler converts the source code or the Java program into the Byte Code(or machine code), and secondly, the Interpreter executes the byte code on the system. The Interpreter can also be called JVM(Java Virtual Machine). The byte code is the common piece between the compiler(which creates it) and the Interpreter or JVM (which runs it).



Whenever we write any program, it is not written in machine code. We write it in a high-level language like JAVA, C++, Python, etc. But the computer understands only the machine code. So when we execute our program, it is first converted into machine code or Byte code by the compiler and then executed by the Interpreter.

**This intermediate code or the byte can run on any platform making, JAVA a platform-independent language.**

**----------------------------**

**Java Variables**

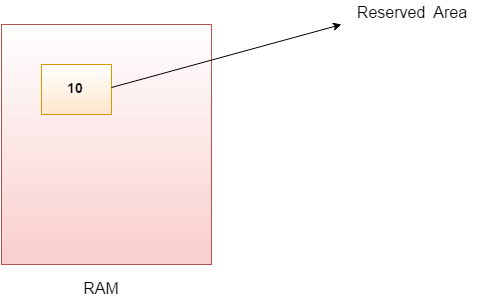
A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of data types in Java: primitive and non-primitive.

**Variable definition**

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



**Example**

**int data=50;//Here data is variable**

**Types of Variables**

There are three types of variables in Java:

local variable

instance variable

static variable

**1) Local Variable**

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

**2) Instance Variable**

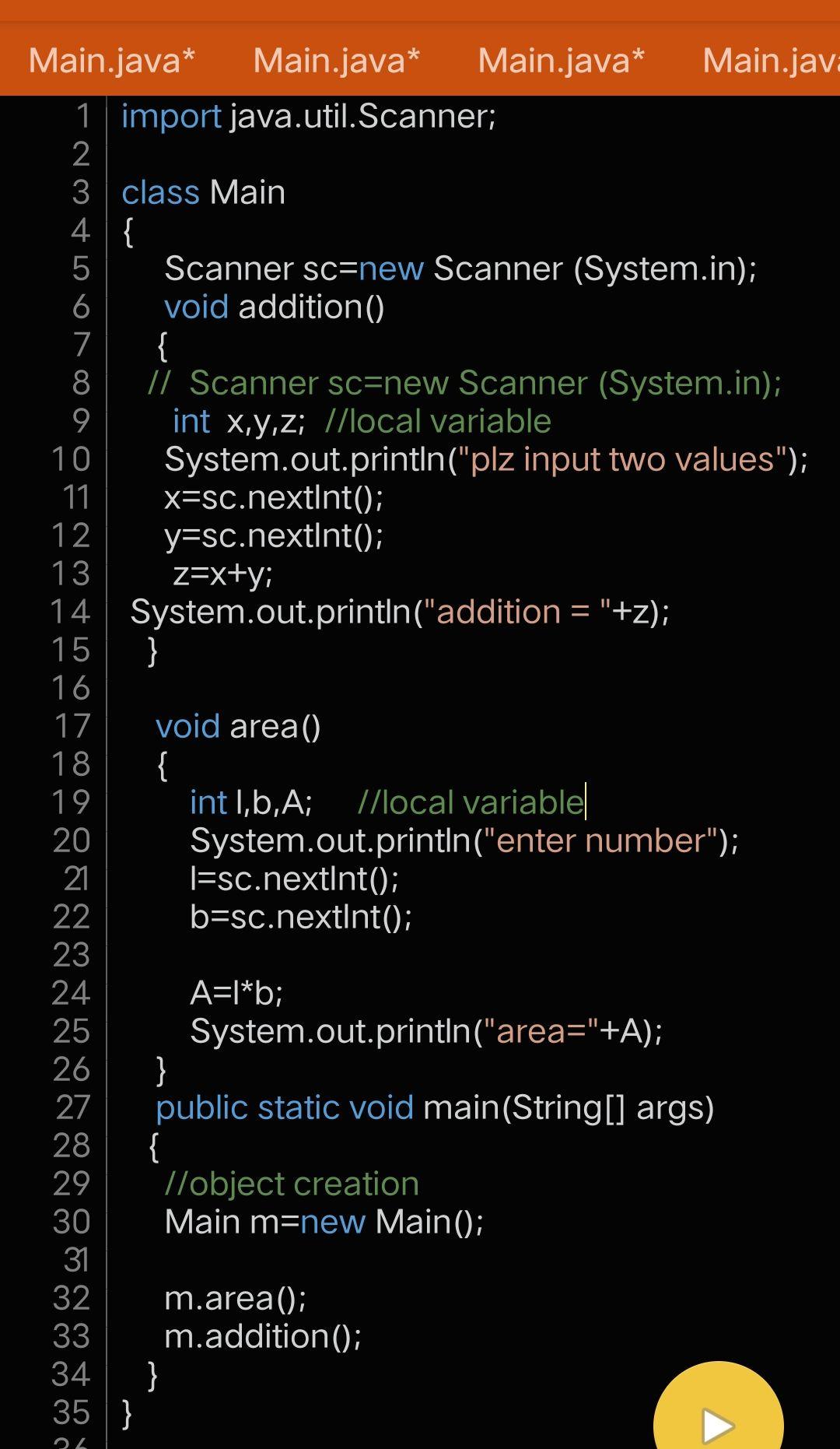
A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.

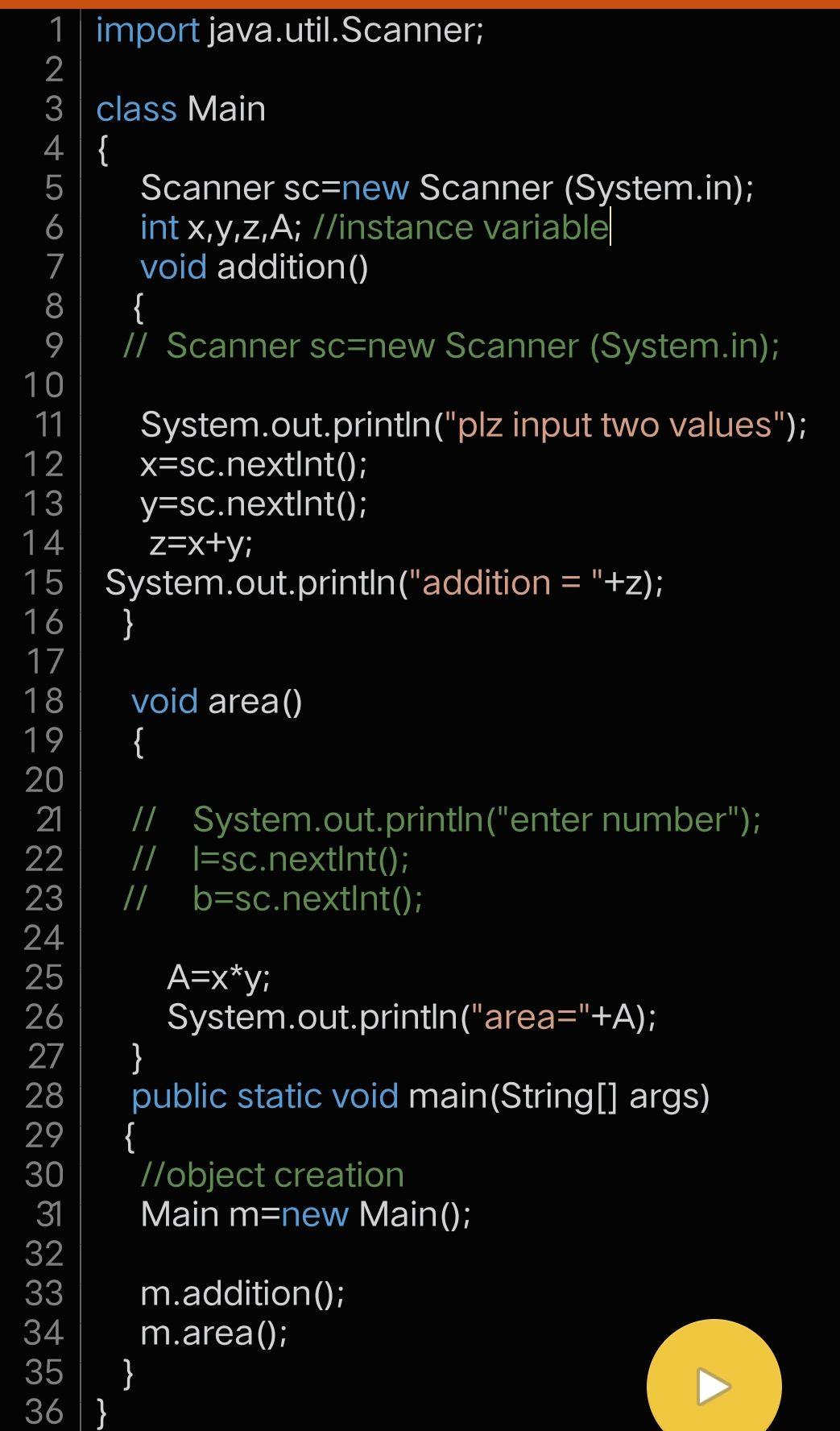
It is called an instance variable because its value is instance-specific and is not shared among instances.

**3) Static variable**

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

**EXAMPLE**

****

****

**STATIC VS INSTANCE**

Static and instance variables behave same if both are represented by same object.

But if object are different than behavior would be different.

**We can say that**

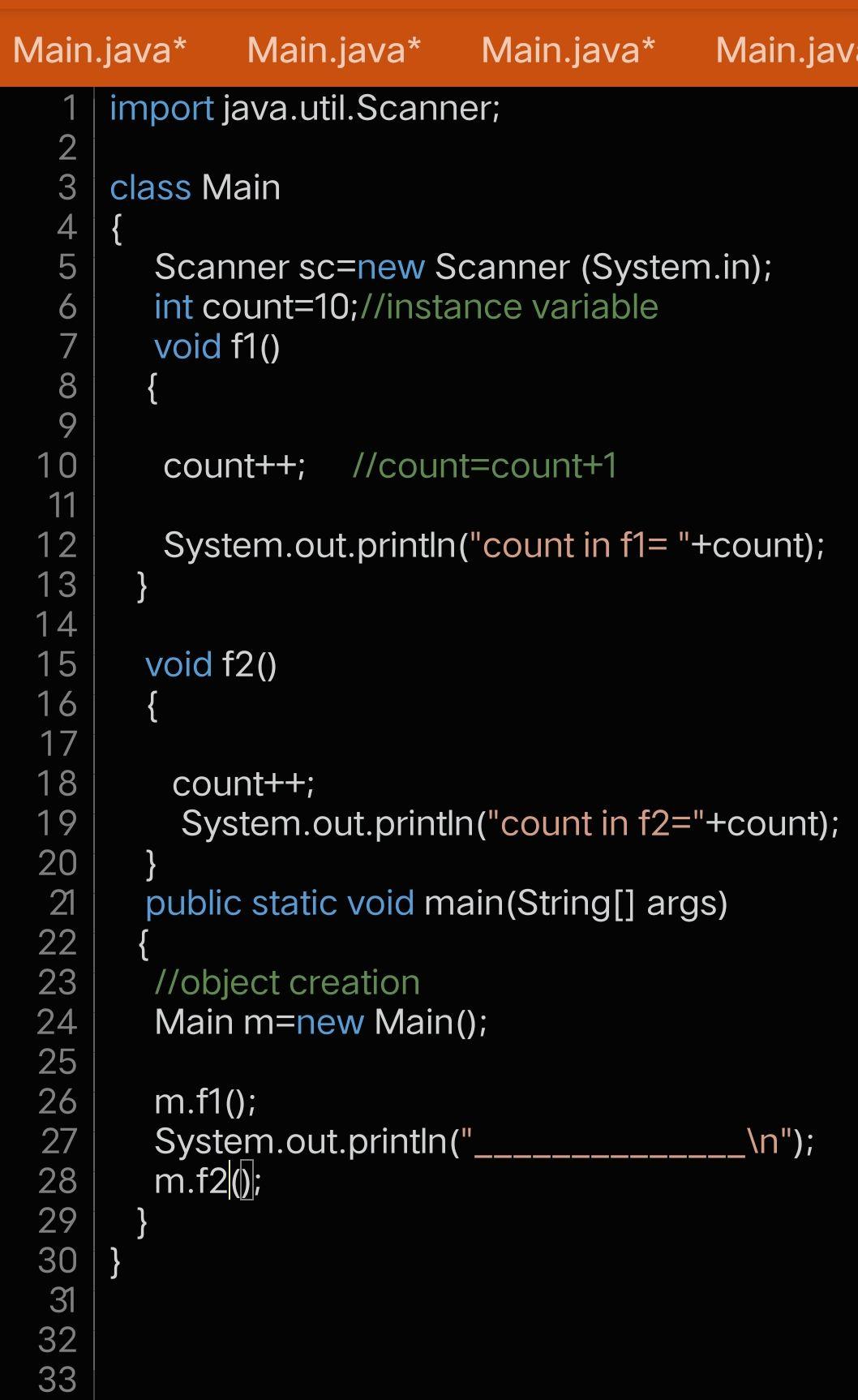
Static variables doesn't bound with objects means they are same with each and every objects of class

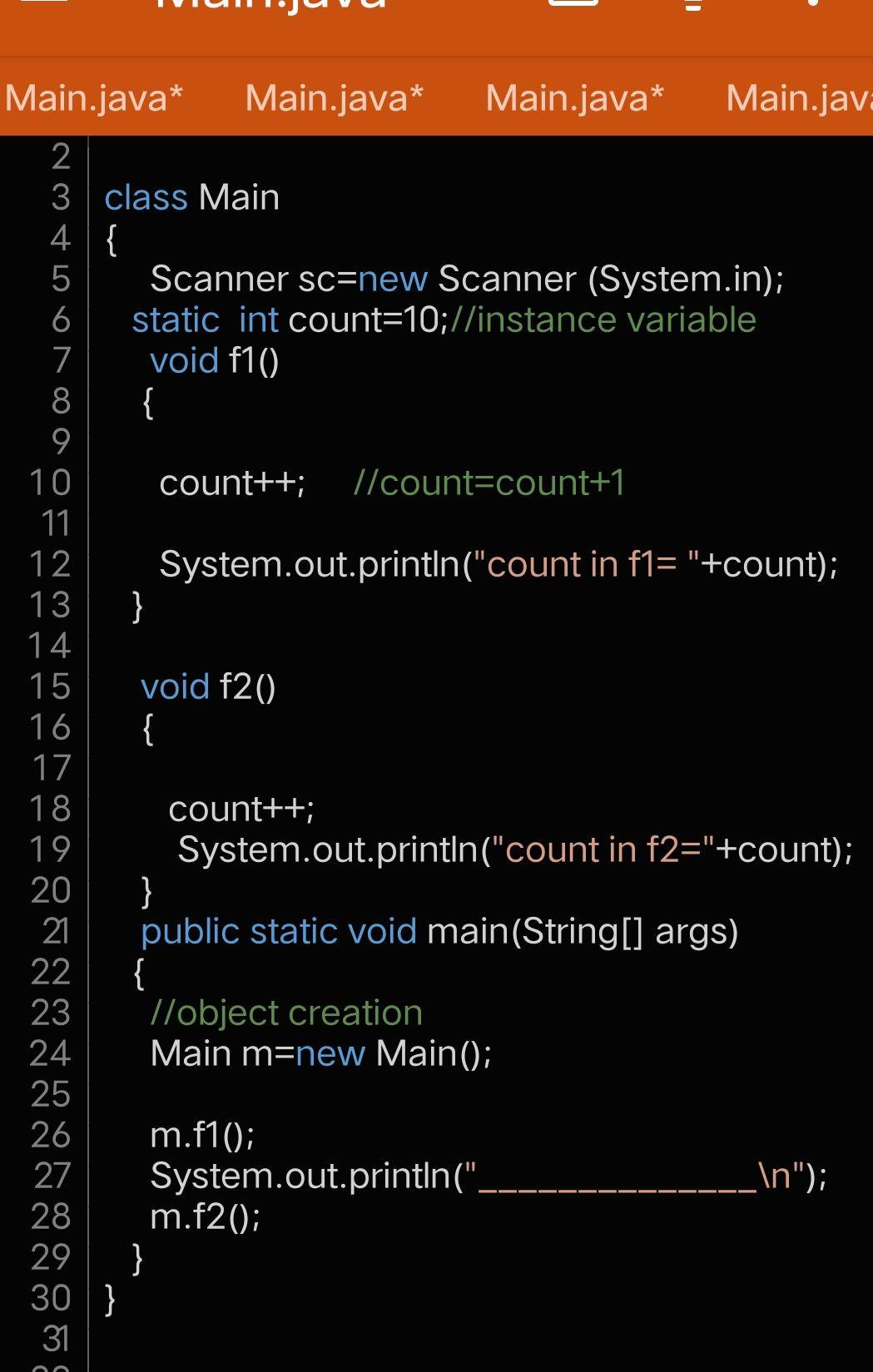
**in simple words**

**Static variable share same value with every objects whereas instance variables share separate value with every object.**

**EXAMPLE**

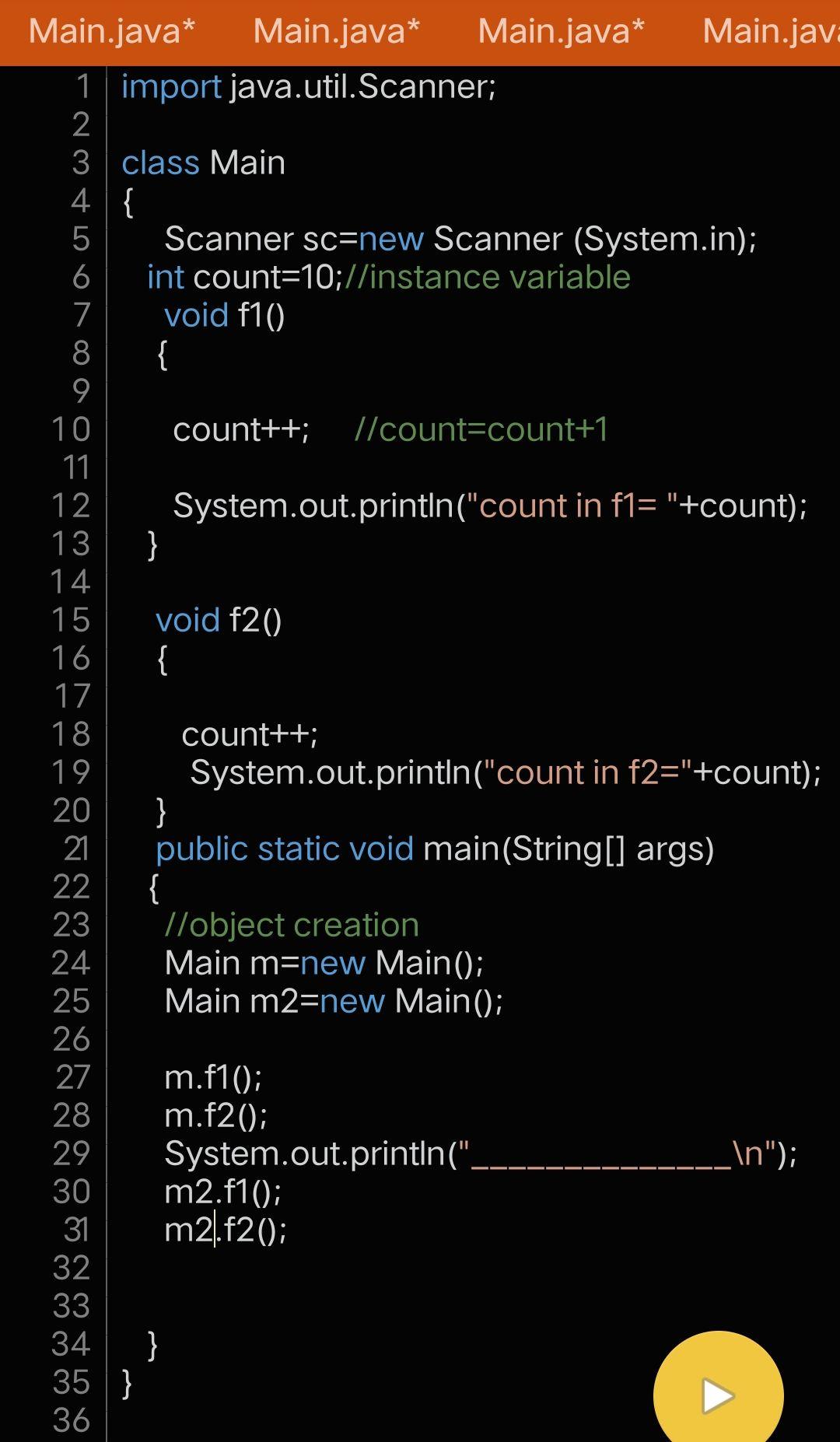
**Case 1 when object are same on just one than instance and static variable are same.**

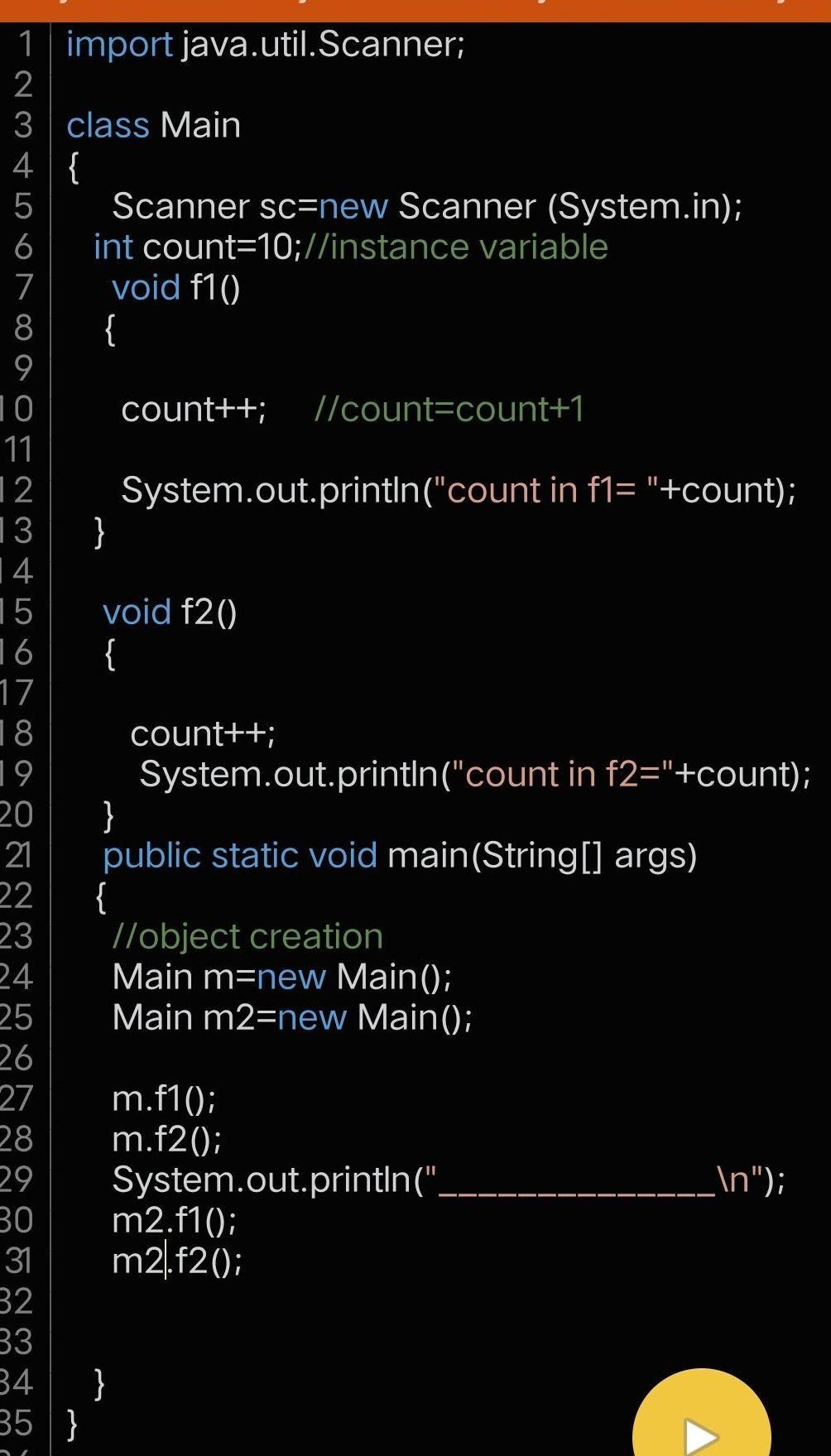
****

****

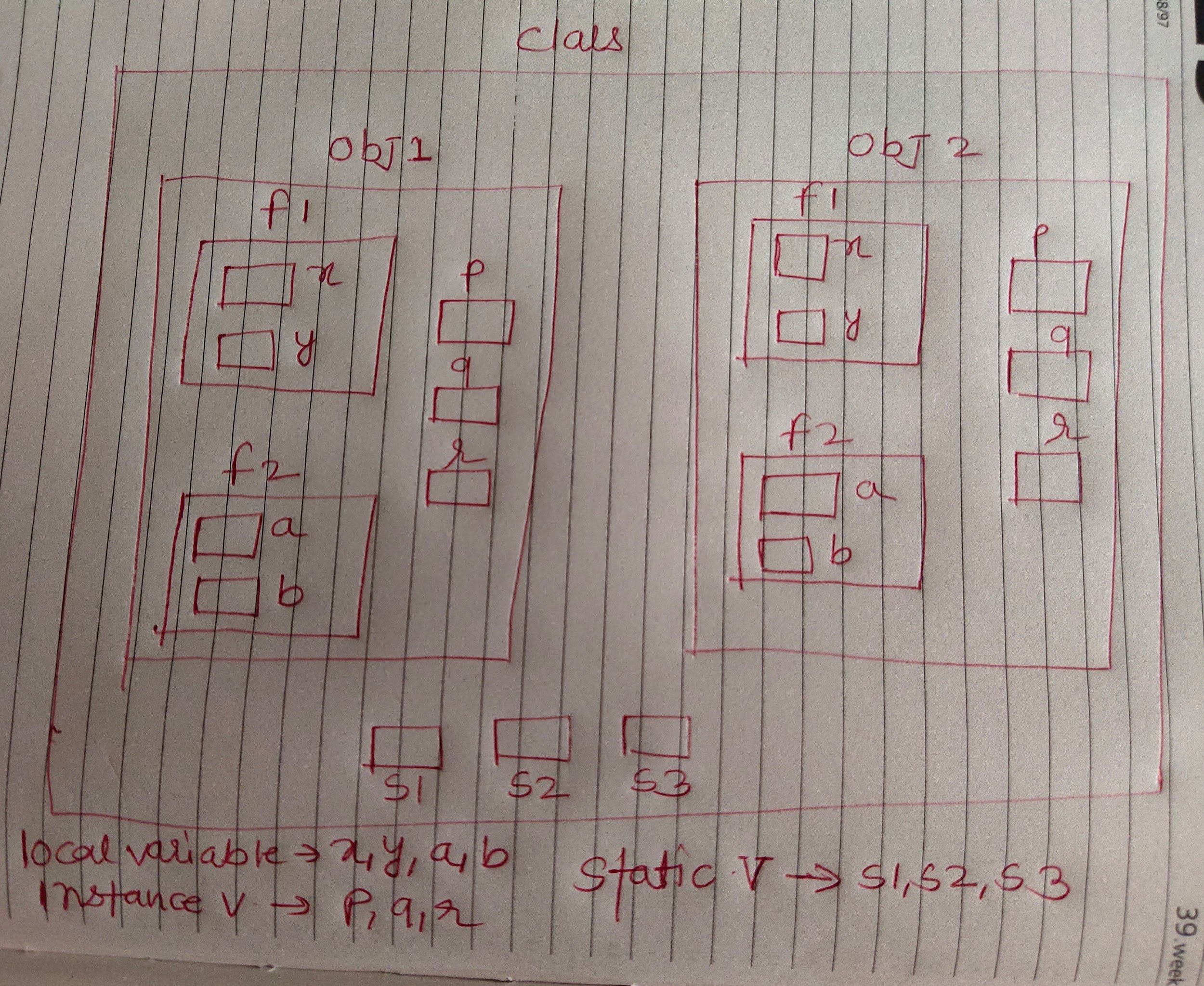
**Case 2**

**When objects are more than one than instance and static variable behave differently because instance variable bound with objects whereas static doesn't.**

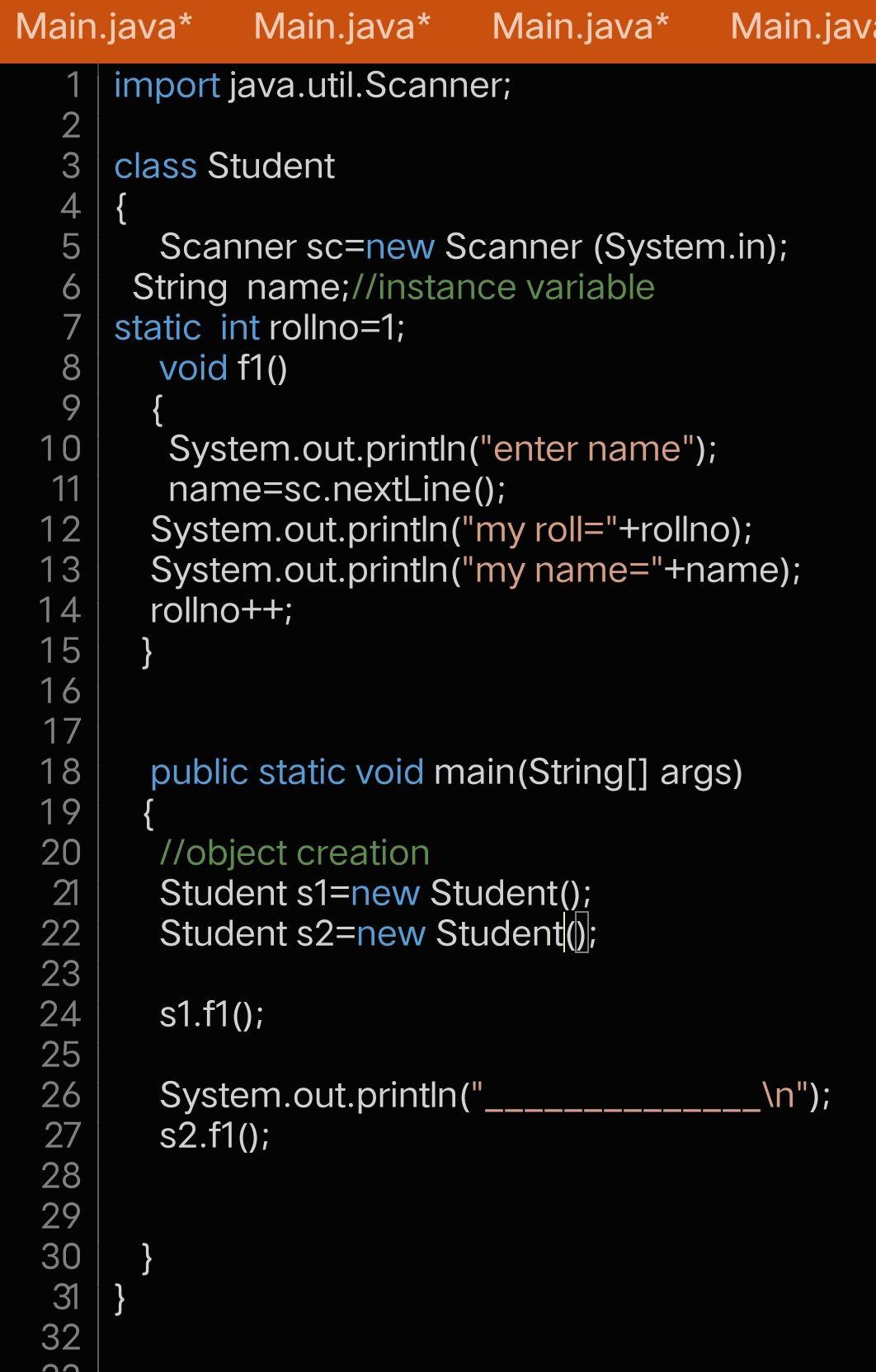
****

****

**Block diagram of all 3 variables and relationship with class and objects**

****

**A real life example**

****

**-------------------------------------**

**Java If-else Statement**

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**

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

Syntax:

if(condition)

{

//code to be executed

}

if statement in java example

**//Java Program to demonstate the use of if statement.**

**class IfExample**

**{**

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

**{**

**//defining an 'age' variable**

**int age=20;**

**//checking the age**

**if(age>18)**

**{**

**System.out.print("Age is greater than 18");**

**}**

**}**

**}**

**Java if-else Statement**

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

Syntax:

if(condition)

{

//code if condition is true

}

else

{

//code if condition is false

}

**Example**

**//A Java Program to demonstrate the use of if-else statement.**

**//It is a program of odd and even number.**

**class IfElseExample**

**{**

**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");**

**}**

**}**

**}**

**Imp example**

**Leap year**

**A year is leap, if it is divisible by 4 and 400. But, not by 100.**

**class LeapYearExample**

**{**

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

**int year=2020;**

**if(((year % 4 ==0) && (year % 100 !=0)) || (year % 400==0))**

**{**

**System.out.println("LEAP YEAR");**

**}**

**else**

**{**

**System.out.println("common year");**

**}**

**}**

**}**

**Java if-else-if ladder Statement**

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**

**//Java Program to demonstrate the use of If else-if ladder.**

**//It is a program of grading system for fail, D grade, C grade, B grade, A grade and A+.**

**class IfElseIfExample**

**{**

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

**int marks=65;**

**if(marks<50)**

**{**

**System.out.println("fail");**

**}**

**else if(marks>=50 && marks<60)**

**{**

**System.out.println("D grade");**

**}**

**else if(marks>=60 && marks<70)**

**{**

**System.out.println("C grade");**

**}**

**else if(marks>=70 && marks<80)**

**{**

**System.out.println("B grade");**

**}**

**else if(marks>=80 && marks<90)**

**{**

**System.out.println("A grade");**

**}**

**else if(marks>=90 && marks<100)**

**{**

**System.out.println("A+ grade");**

**}**

**else**

**{**

**System.out.println("Invalid!"); }**

**}**

**}**

**--------------------------------**

**Constructor**

**In Java, 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.**

**It is a special type of method which is used to initialize the object.**

**Every time an object is created using the new() keyword, at least one constructor is called.**

***It calls a default constructor if there is no constructor available in the class.***

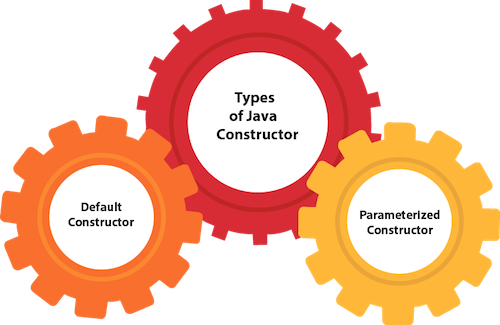
***In such case, Java compiler provides a default constructor by default.***

**There are two types of constructors in Java:**

**no-arg constructor (default)**

**And**

**parameterized constructor.**

****

**Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.**

**Rules for creating Java constructor**

**There are two rules defined for the 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***

**//Java Program to create and call a default constructor**

**class Bike1**

**{**

**//creating a default constructor**

**Bike1()**

**{**

**System.out.println("Bike is created");**

**}**

**//main method**

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

**{**

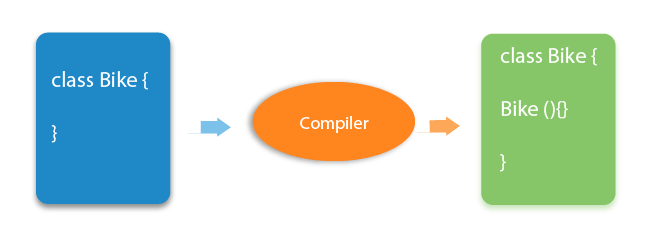
**//calling a default constructor**

**Bike1 b=new Bike1();**

**}**

**}**

**Rule: If there is no constructor in a class, compiler automatically creates a default constructor.**

****

**Imp point to remember**

**Q) What is the purpose of a default constructor?**

**ans)**

**The default constructor is used to provide the default values to the variables of objects like 0 (if int), null(if string), etc., depending on the type.**

**EXAMPLE**

**class Student3**

**{**

**int id;**

**String name;**

**//method to display the value of id and name**

**void display()**

**{**

**System.out.println(id+" "+name);**

**}**

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

**{**

**//creating objects**

**Student3 s1=new Student3();**

**Student3 s2=new Student3();**

**//displaying values of the object**

**s1.display();**

**s2.display();**

**}**

**}**

**Output**

**0 null**

**0 null**

**Explanation:**

**In the above class,you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.**

**Parameterized Constructor**

**A constructor which has a specific number of parameters is called a parameterized constructor.**

**Why use the parameterized constructor?🤔**

**The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.**

**Example**

**In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.**

**//Java Program to demonstrate the use of the parameterized constructor.**

**class Student4**

**{**

**int id;**

**String name;**

**//creating a parameterized constructor**

**Student4(int i,String n)**

**{**

**id = i;**

**name = n;**

**}**

**//method to display the values**

**void display()**

**{**

**System.out.println(id+" "+name);**

**}**

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

**{**

**//creating objects and passing values**

**Student4 s1 = new Student4(1,"Karan");**

**Student4 s2 = new Student4(2,"Aryan");**

**//calling method to display the values of object**

**s1.display();**

**s2.display();**

**}**

**}**

**Output:**

**1 Karan**

**2 Aryan**

**Constructor Overloading**

**In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.**

**Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.**

**Example of Constructor Overloading**

**//Java program to overload constructors**

**class Student5**

**{**

**int id;**

**String name;**

**int age;**

**//creating two arg constructor**

**Student5(int i,String n)**

**{**

**id = i;**

**name = n;**

**}**

**//creating three arg constructor**

**Student5(int i,String n,int a)**

**{**

**id = i;**

**name = n;**

**age=a;**

**}**

**void display()**

**{**

**System.out.println(id+" "+name+" "+age);}**

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

**{**

**Student5 s1 = new Student5(1,"Karan");**

**Student5 s2 = new Student5(2,"Aryan",25);**

**s1.display();**

**s2.display();**

**}**

**}**

**Output:**

**1 Karan 0**

**2 Aryan 25**

**Difference between constructor and method in Java**

**There are many differences between constructors and methods. They are given below.**

**1)A constructor is used to initialize the state of an object.**

**A method is used to expose the behavior of an object.**

**2)A constructor must not have a return type.**

**A method must have a return type.**

**3)The constructor is invoked implicitly.**

**The method is invoked explicitly.**

**4)The Java compiler provides a default constructor if you don't have any constructor in a class.**

**The method is not provided by the compiler in any case.**

**5)The constructor name must be same as the class name.**

**The method name may or may not be same as the class name.**

**Java Copy Constructor(not imp)**

**There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.**

**//Java program to initialize the values from one object to another object.**

**class Student6**

**{**

**int id;**

**String name;**

**//constructor to initialize integer and string**

**Student6(int i,String n)**

**{**

**id = i;**

**name = n;**

**}**

**//constructor to initialize another object**

**Student6(Student6 s)**

**{**

**id = s.id;**

**name =s.name;**

**}**

**void display()**

**{**

**System.out.println(id+" "+name);**

**}**

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

**{**

**Student6 s1 = new Student6(1,"Karan");**

**Student6 s2 = new Student6(s1);**

**s1.display();**

**s2.display();**

**}**

**}**

**Output:**

**1 Karan**

**1 Karan**

**Q) Does constructor return any value**

**Yes, it is the current class instance (You cannot use return type yet it returns a value).**

**Can constructor perform other tasks instead of initialization?**

**Yes, like object creation, starting a thread, calling a method, etc. You can perform any operation in the constructor as you perform in the method.**

**Q)🤔Is there Constructor class in Java?**

**Yes (no more details should need to learn )**

**--------------------------**

**This keyword**

**this keyword in Java is a reference variable that refers to the current object of a method or a constructor.**

**The main purpose of using this keyword in Java is to remove the confusion between object(instance) variable and parameters that have same names.**

**First Understanding the problem without this keyword**

**class Student**

**{**

**int rollno;**

**String name;**

**float fee;**

**Student(int rollno,String name,float fee)**

**{**

**rollno=rollno;**

**name=name;**

**fee=fee;**

**}**

**void display()**

**{**

**System.out.println(rollno+" "+name+" "+fee);**

**}**

**}**

**class TestThis1**

**{**

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

**{**

**Student s1=new Student(1,"ankit",5000f);**

**Student s2=new Student(2,"sumit",6000f);**

**s1.display();**

**s2.display();**

**}**

**}**

**Output**

**0 null 0.0**

**0 null 0.0**

**Explanation**

**In the above example, parameters (formal arguments) and instance variables are same.**

**So, java provides this keyword to distinguish local variable and instance variable.**

**Solution of the above problem by this keyword**

**class Student**

**{**

**int rollno;**

**String name;**

**float fee;**

**Student(int rollno,String name,float fee)**

**{**

**this.rollno=rollno;**

**this.name=name;**

**this.fee=fee;**

**}**

**void display()**

**{**

**System.out.println(rollno+" "+name+" "+fee);}**

**}**

**class TestThis2**

**{**

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

**{**

**Student s1=new Student(111,"ankit",5000f);**

**Student s2=new Student(112,"sumit",6000f);**

**s1.display();**

**s2.display();**

**}**

**}**

**Output**

**111 ankit 5000.0**

**112 sumit 6000.0**

**Note✍🏻:**

**We can avoid use of this keyword by giving different names to instance variable and parameters (local variable)**

**but It is better approach to use meaningful names for variables. So we use same name for instance variables and parameters in real time, and always use this keyword.**

**This keyword is used in many ways in java.**

**Here is given the 6 usage of java this keyword.**

**1)This can be used to refer current class instance variable. (explained)**

**2)this can be used to invoke current class method (implicitly)**

**3)this() can be used to invoke current class constructor.**

**(2 and 3 Will be explain later)**

**4)this can be passed as an argument in the method call.**

**5)This can be passed as argument in the constructor call.**

**6)this can be used to return the current class instance from the method.**

**--------------------------**