SYNTAX TECHNOLOGIES

JAVA

Class 17

Agenda

Method in Java

Methods with Parameters and without

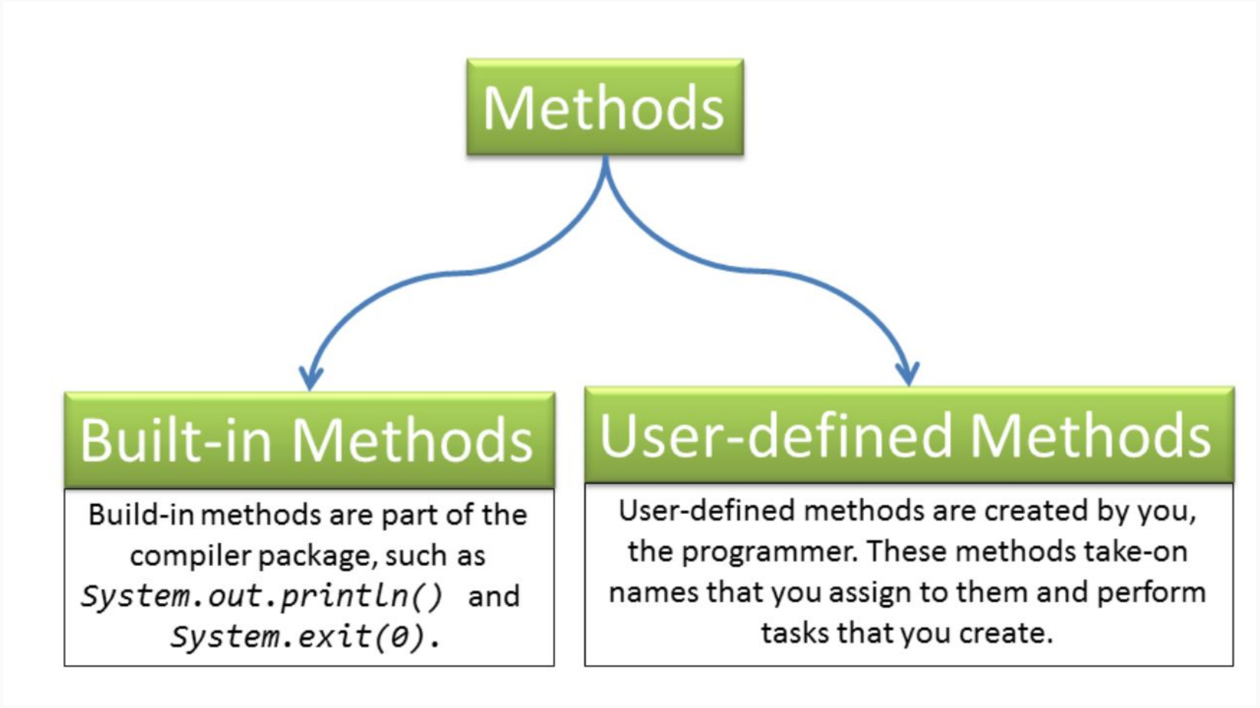
Methods with return values and without

Method in Java

- A Java method is a collection of statements that are grouped together to perform an operation
- Method describe behavior of an object
- Method means a block of code
- You can say the method is a subprogram that acts on data and often returns a value.
- **Java methods must be located inside a Java class.**

Type of Methods in Java

There is 2 types of Methods in JAVA



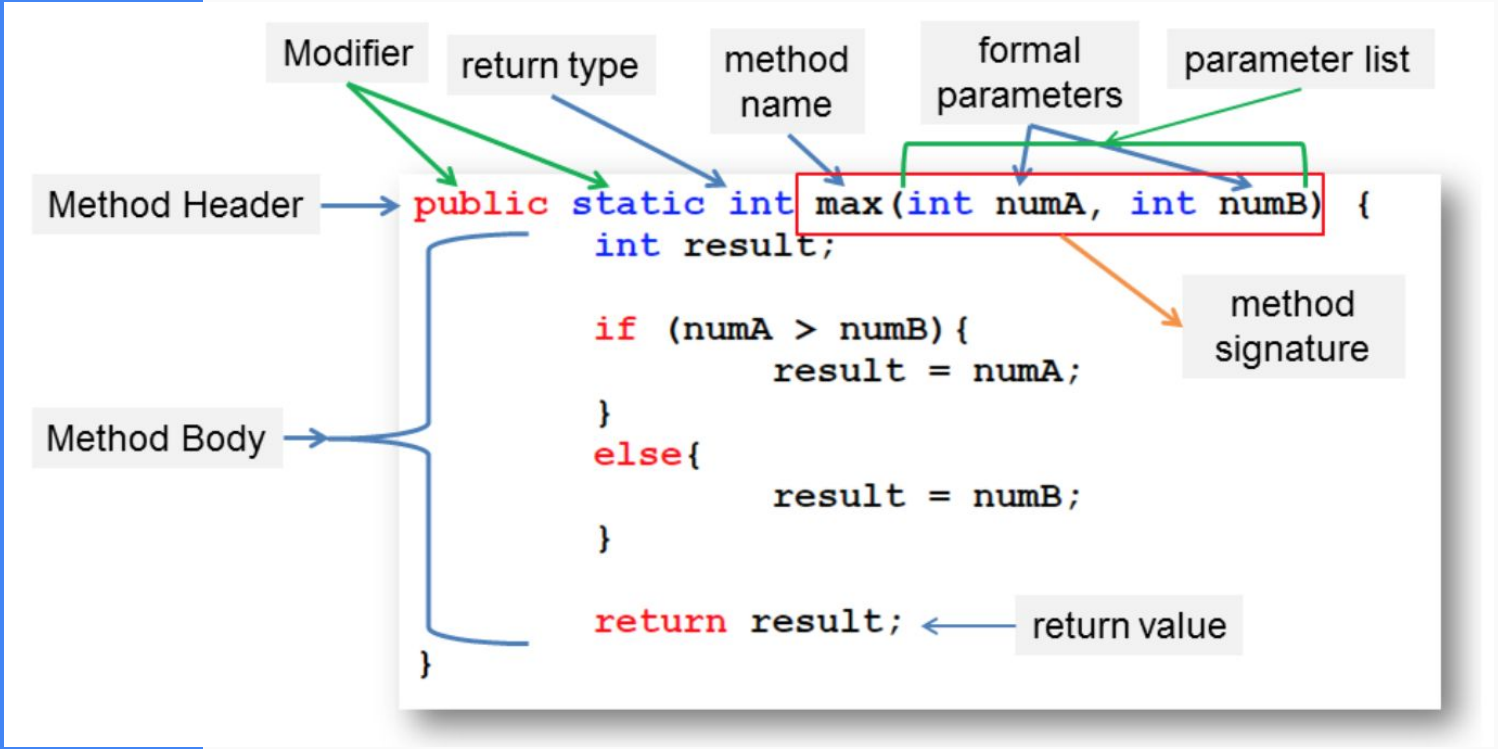
Build-in methods are part of the compiler package, such as *System.out.println()* and *System.exit(0)*.

User-defined methods are created by you, the programmer. These methods take-on names that you assign to them and perform tasks that you create.

Methods in Java

Method definitions have 2 basic parts:

- The method header
- The body of the method



```java
public static int max(int numA, int numB) {
        int result;

        if (numA > numB){
                result = numA;
        }
        else{
                result = numB;
        }

        return result;
}
```

Methods in Java

Modifiers: The modifier, which is optional, tells the compiler how to call the method.

Return Type: A method may return a value.
The return Value Type is the datatype of the value the method returns.

Method Name: This is the actual name of the method.

Parameters: This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.

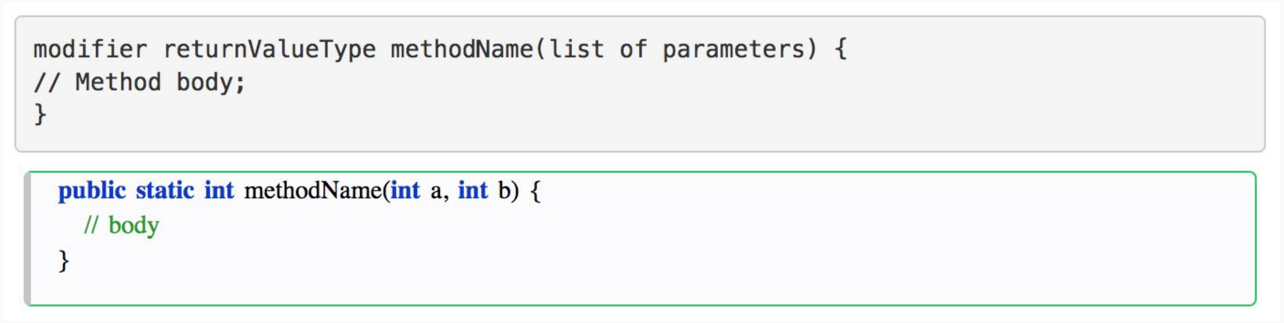
Method Body: The method body contains a collection of statements that define what the method does.

Note: In certain other languages, methods are referred to as procedures and functions. A method with a non void return value type is called a function; a method with a void return value type is called a procedure

Methods without parameters

```
public class Greetings {

    public static void main(String[] args) {

        Greetings obj = new Greetings();
        obj.hello();

    }

    void hello() {
        System.out.println("Hello");
    }
}
```

Methods with parameters



```
modifier returnValueType methodName(list of parameters) {
// Method body;
}
```

```
public static int methodName(int a, int b) {
  // body
}
```

```
void sum(int a, int b) {
        System.out.println(a+b);
}

void sub(int a, int b) {

        System.out.println(a-b);
}
```

Task

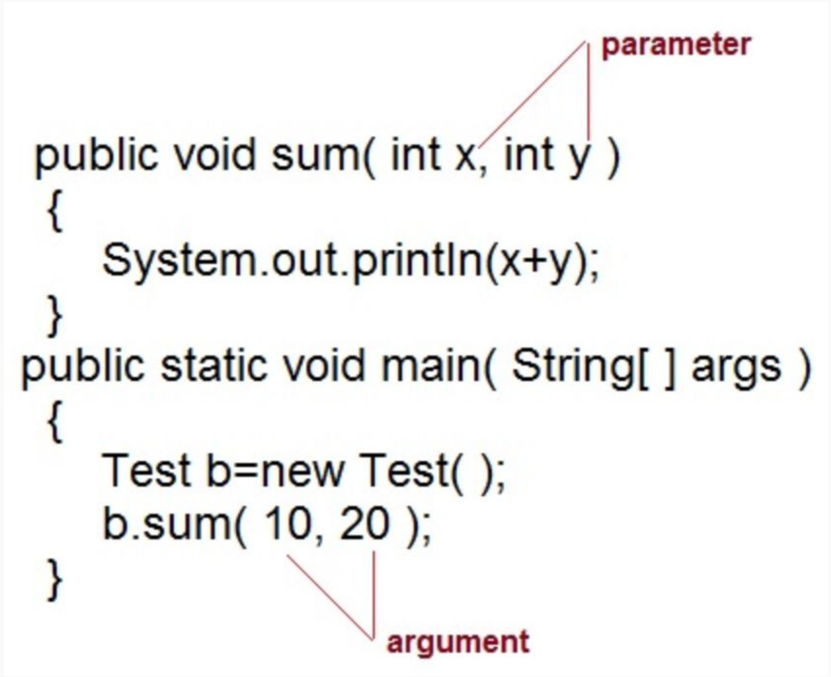
1. Create a method that will take 2 parameters as a numbers and prints which number is larger.

2. Create a method that will take a number and prints whether the number is even or odd.

3. Create a method that will print whether given String is palindrome or not.

Parameter vs. Argument

- Parameter is variable defined by a method that receives value when the method is called.
- Parameter are always local to the method they don't have scope outside the method.
- Argument is a value that is passed to a method when it is called.



```
public void sum( int x, int y )
 {
     System.out.println(x+y);
 }
public static void main( String[ ] args )
 {
     Test b=new Test( );
     b.sum( 10, 20 );
 }
```

User Defined Methods in Java

1. Method without return any value

   Uses **void** keyword

1. Method with return values.

   Uses **return** keyword

Void keyword allows to create methods which do not return a value.

1.Method without return type and without arguments.

```
class sample{

public void add(){

int a=40;
int b=50;
int c=a+b;
SOP(c);
}

public static void main(String args[]) {
sample obj= new sample();
obj.add();
 }
}
```

2.Method without return type and with arguments.

```
class sample{

public void add(int a, int b){

int c=a+b;
SOP(c);
}

public static void main(String args[]) {
 sample obj= new sample();
obj.add(13,24);
 }
}
```

3.Method with return type and without arguments.

```
class sample{

public int add(){
int a=40;
int b=50;
int c=a+b;
return c;
}

public static void main(String args[]) {

sample obj= new sample();
int x=obj.add();
SOP(x);
 }
}
```

4.Method with return type and with arguments.

```
class sample{

public int add(int a, int b){

int c=a+b;
return c;
}

public static void main(String args[]) {

sample obj= new sample();
int x=obj.add(1,2);
SOP(x);
 }
}
```

Rules

- The return type is the **must** in a method, you can't declare the method without it's return type.
- If the return type is **void** it means method will **not return** any value.
- The **return** statement should be the **last** statement in the method.
- You **can't** declare more than one **return** statement in one method.
- The data type of the returned value must match the type of the method's declared return value.
- You can **call** the method with its name only.
- The method **must** have the **body**.
- The method can accept the n number of **parameters**.
- If the method has n number of **parameter** then it's the **must** to pass all parameter in method body while **calling** the method in program code.
- The variables declared **inside** the method body are called the **local** variable.
- Methods access modifiers define the access level of method.

SYNTAX TECHNOLOGIES

JAVA

Class 19

Agenda

Variable types in JAVA: Local
Instance
Static

Variables

Variables are used to store information to be referenced and manipulated in a computer program.

Variable is name of reserved area allocated in memory. In other words, it is a name of memory location.

Value of the variable can change, depending on conditions or on information passed to the program.

variable="vary + able" that means its value can be changed.

type identifier [= value][, identifier [= value] ...] ;

The type is one of Java's primitive types or the name of a class or interface.

The identifier is the name of the variable. You can initialize the variable by specifying an equal sign and a value.

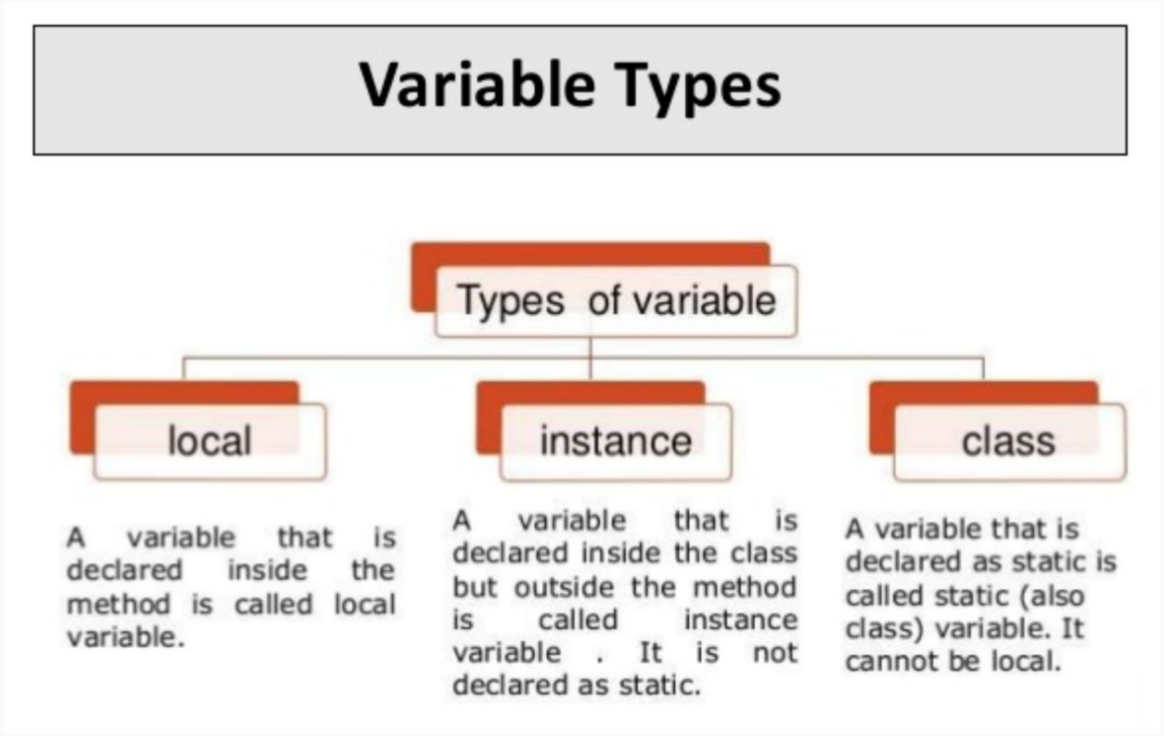
You can initialize the variable by specifying an equal sign and a value.

The initialization expression must result in a value of the same (or compatible) type as that specified for the variable.

To declare more than one variable of the specified type, use a comma-separated list.

Types of variables in Java

1. **Local variables.**
2. **Instance variables.**
3. **Static variables.**



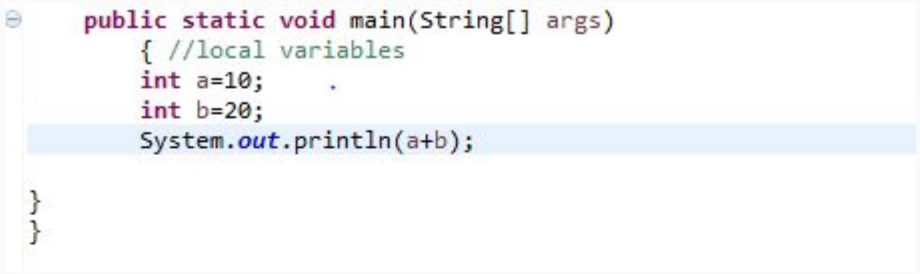
A variable that is declared inside the method is called local variable.

A variable that is declared inside the class but outside the method is called instance variable . It is not declared as static.

A variable that is declared as static is called static (also class) variable. It cannot be local.

Local Variable

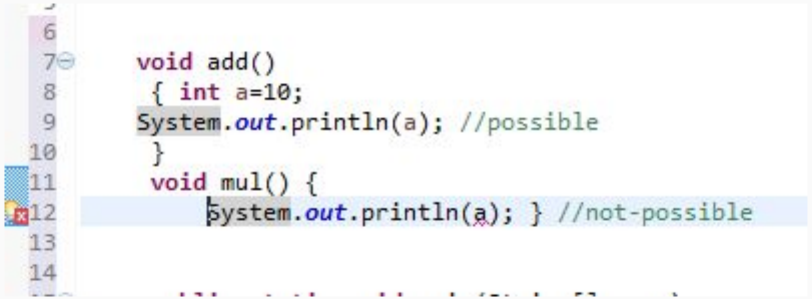
The variables which are declare inside a method or constructor or blocks those variables are called local variables.



```
    public static void main(String[] args)
        { //local variables
        int a=10;
        int b=20;
        System.out.println(a+b);

}
}
```

Local Variable

It is possible to access local variables only inside the method or constructor or blocks only, it is not possible to access outside of method or constructor or blocks.
Local variables memory allocated when method starts & memory released when method completed



```
    void add()
     { int a=10;
    System.out.println(a); //possible
     }
     void mul() {
            System.out.println(a); } //not-possible
```

Local Variable

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
- Access modifiers cannot be used for local variables.
- 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.

```
public class VariableExample {
     public String myVar="instance variable";   // instance variable
   public void myMethod(){
        // local variable
        String myVar = "Inside Method";
        System.out.println(myVar);
   }
   public static void main(String args[]){
          VariableExample obj = new VariableExample();   // Creating object

      /* We are calling the method, that changes the  value of myVar. We are displaying myVar
again after
       * the method call, to demonstrate that the local  variable scope is limited to the method
itself.
       */
      System.out.println("Calling Method");
      obj.myMethod();
      System.out.println(obj.myVar);
   }
}
```

```
Calling Method
Inside Method
instance variable
```

Instance Variable

The methods and variables defined within a class are called members of the class.

A variable which is declared inside the class but outside the method is called instance variable. It is not declared as static.

Why the name is instance: Variables defined within a class are called instance variables because each instance of the class (that is, each object of the class) contains its own copy of these variables.

Thus, the data for one object is separate and unique from the data for another.

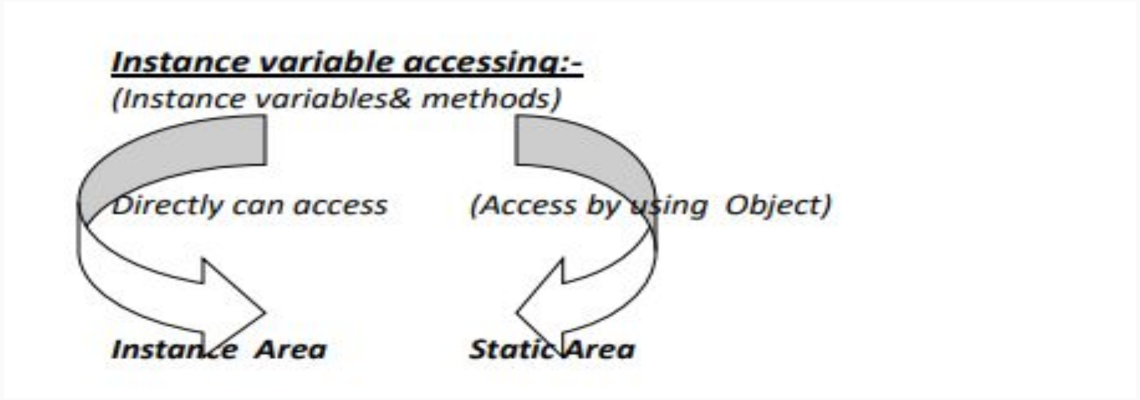
It is called instance variable because its value is instance specific and is not shared among instances.

Important Points about Instance Variable

Instance variables are used in a class, but outside a method, constructor or any block.

A slot for each instance variable value is created, when space is allocated for an object in the heap.

Instance variables can be declared in the class level before or after use.



```java
public class InstanceVarExample {
   String myInstanceVar="instance variable";

   public static void main(String args[]){
        InstanceVarExample obj = new InstanceVarExample();
        InstanceVarExample obj2 = new InstanceVarExample();

        System.out.println(obj.myInstanceVar);
        System.out.println(obj2.myInstanceVar);

        obj2.myInstanceVar = "Changed Text";

        System.out.println(obj.myInstanceVar);
        System.out.println(obj2.myInstanceVar);
   }
}
```

```
instance variable
instance variable
Changed Text
instance variable
```

```
public class InstanceVariable{

    public String name;// this instance variable

    public InstanceVariable(String consName) {
     name = consName;
     System.out.println("Your Name is: " + name);
     }

    public static void main(String args[]) {
        InstanceVariable r = new InstanceVariable("atnyla");
    }
}
```

```
Your Name is: atnyla
Press any key to continue . . .
```

Static variable

A variable which is declared as static is called static variable.

Static variables are also known as class variable because they are associated with the class and common for all the instances of class.

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.

Static variable

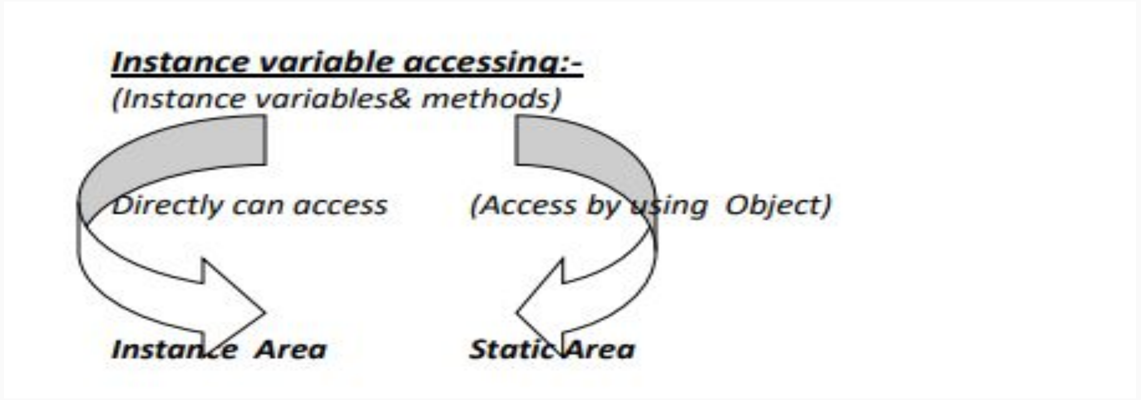
Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.

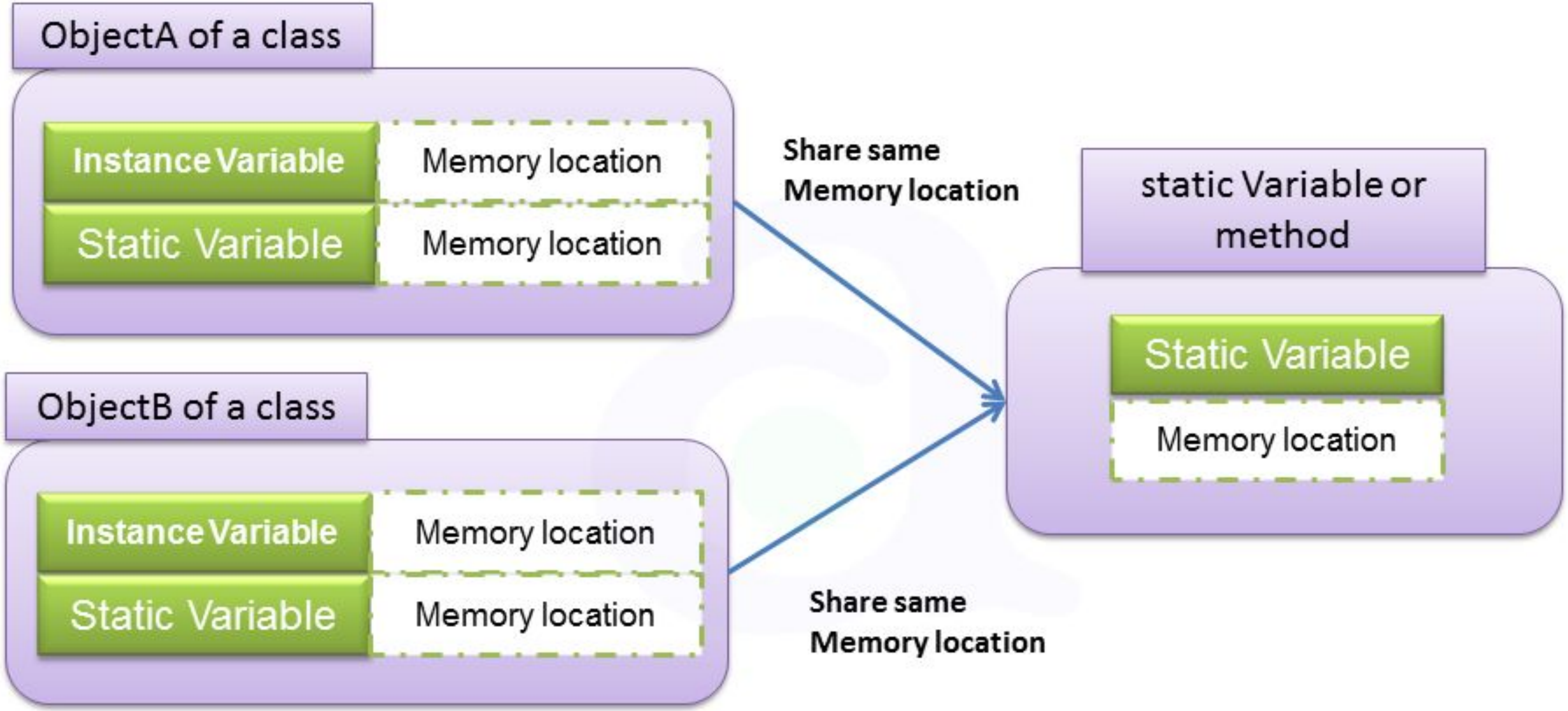
Static variables store values for the variables in a common memory location.

Because of this common location, if one object changes the value of a static variable, all objects of the same class are affected.

Important Points about Static Variable

Static variables are rarely used other than being declared as constants.
Constants are variables that are declared as public/private, final, and static.
Constant variables never change from their initial value.
Static variables are stored in the static memory.
It is rare to use static variables other than declared final and used as either public or private constants.





static variables store values for the variables and share a common memory location for all objects

```
package studytonight;
class Student{
    int a;
    static int id = 35;
    void change(){
        System.out.println(id);
    }
}
public class StudyTonight {
    public static void main(String[] args) {

        Student o1 = new Student();
        Student o2 = new Student();
        o1.change();
        Student.id = 1;
        o2.change();
    }
}
```

```
35
1
```