**Program 1:**  
public class Main {

public static void main(String[] args) {

int a=10, b=20;

System.out.println(a+b);

}

}

**Output:** 30. Here ‘a’ is 10 and b is 20. So when we add both ‘a’ and ‘b’, we get 10+20=30 as the output

**Program 2:**  
public class Main1 {

public static void main(String[] args) {

byte b1=10, b2=20;

byte b3 = b1+b2;

System.out.println(b3);

}

}

**Output:** Compilation Error. Reason: when we perform any arithmetic operation, by default the output is in “int” datatype. In this case byte is of 1 byte of memory whereas the int is of 4 bytes memory. So we are trying to store 4 bytes data into 1 byte of memory which not possible and if still we try to do so then there is a chance of loss of data. So to avoid this possible loss of data compiler doesn’t allow such operation.

//Range of byte: -128 to 127

//byte -> 1byte, short->2 bytes, int->4 bytes, long->8 bytes

So as a part the solution for the above program we can 2 different solutions:

1. Change the datatype of b3 to int

public class Main {

public static void main(String[] args) {

byte b1=10, b2=20;

int b3 = b1+b2;

System.out.println(b3);//output: 30

}

}

(or)

2. Convert the int output to byte first then assign to byte type variable

public class Main {

public static void main(String[] args) {

byte b1=10, b2=20;

byte b3 = (byte)(b1+b2);

System.out.println(b3);

}

}

**Note:**  
When we try to store lower datatype value into higher datatype, this is called as “Widening” and widening is implicit in nature as there’s no chance of loss of data. But when we try to store higher datatype value into lower datatype variable, this is called as “Narrowing” and narrowing is explicit in nature as there’s a chance of loss of data.

**Program 3:**  
public class Main {

int a=10;//Global variable as declared inside the class but outside of the method

public static void main(String[] args) {

int b=20;//Local variable as declared inside the method

int sum = a+b;

System.out.println(sum);

}

}

**Output:** Compilation error. Reason: Non static members can’t be accessible inside the static block. So in this case ‘a’ is the global variable which can be accessible throughout the class but is of non static type and the same we are trying to use inside the main method which is static in nature.

So as a part the solution for the above program is  
1. either change the non static member to static

public class Main {

static int a=10;//Global variable as declared inside the class but outside of the method

public static void main(String[] args) {

int b=20;//Local variable as declared inside the method

int sum = a+b;

System.out.println(sum);//30

}

}

2. create an object of the class which is holding the non static member and access the non static member using the object reference

public class Main {

int a=10;

public static void main(String[] args) {

Main m = new Main();

int b=30;

int sum = m.a+b;

System.out.println(sum);//40

}

}

**Note:**

The members which is coming with static keyword are treated as static members whereas the rest all are treated as non static members

**Concept on Static and Non static:**

**Static**: There members are getting loaded in to the system during compilation time. This is not recommended as there’s a high chance to slow down the further processes

**Non Static**: These members are getting loaded in to the system during execution time/runtime. Here it’s recommended as there’s very less chance to slow down the further processes as once the execution completes, it releases the memory.