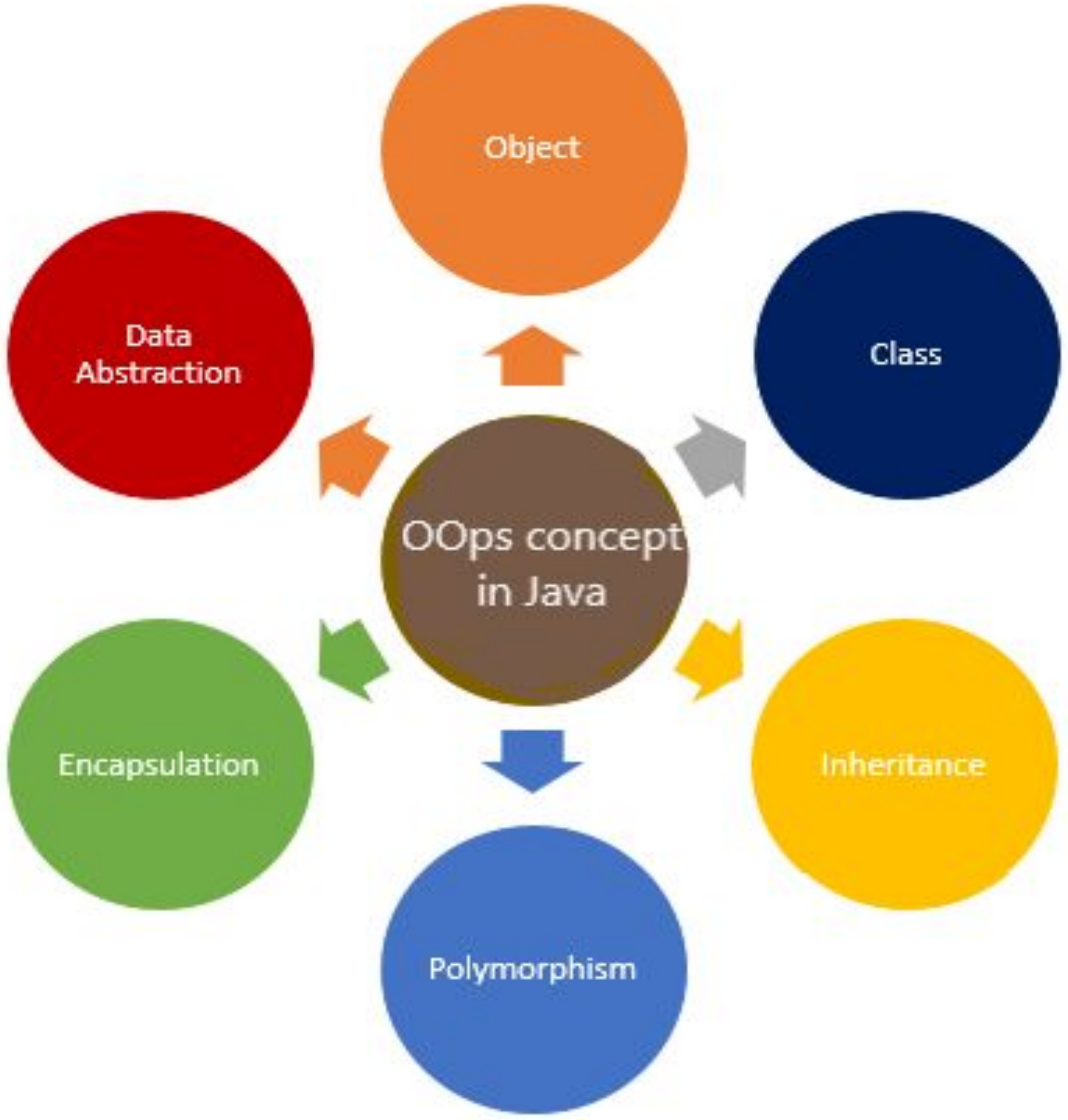


CS20004: Object Oriented Programming using Java

Lec-5

In this Discussion . . .

- Variables
  - Java variable Types: Based on Scope
    - Member Variables (Class Level Scope)
      - Java Package
      - Java Access Specifiers/Modifiers
    - Local Variables (Method Level Scope)
- Automatic conversion of compatible data types & Typecasting
- References

Java variable Types: Based on Scope

- In Java, there exists different types of variables based on scope:
    - Member Variables (Class Level Scope)
    - Local Variables (Method Level Scope)

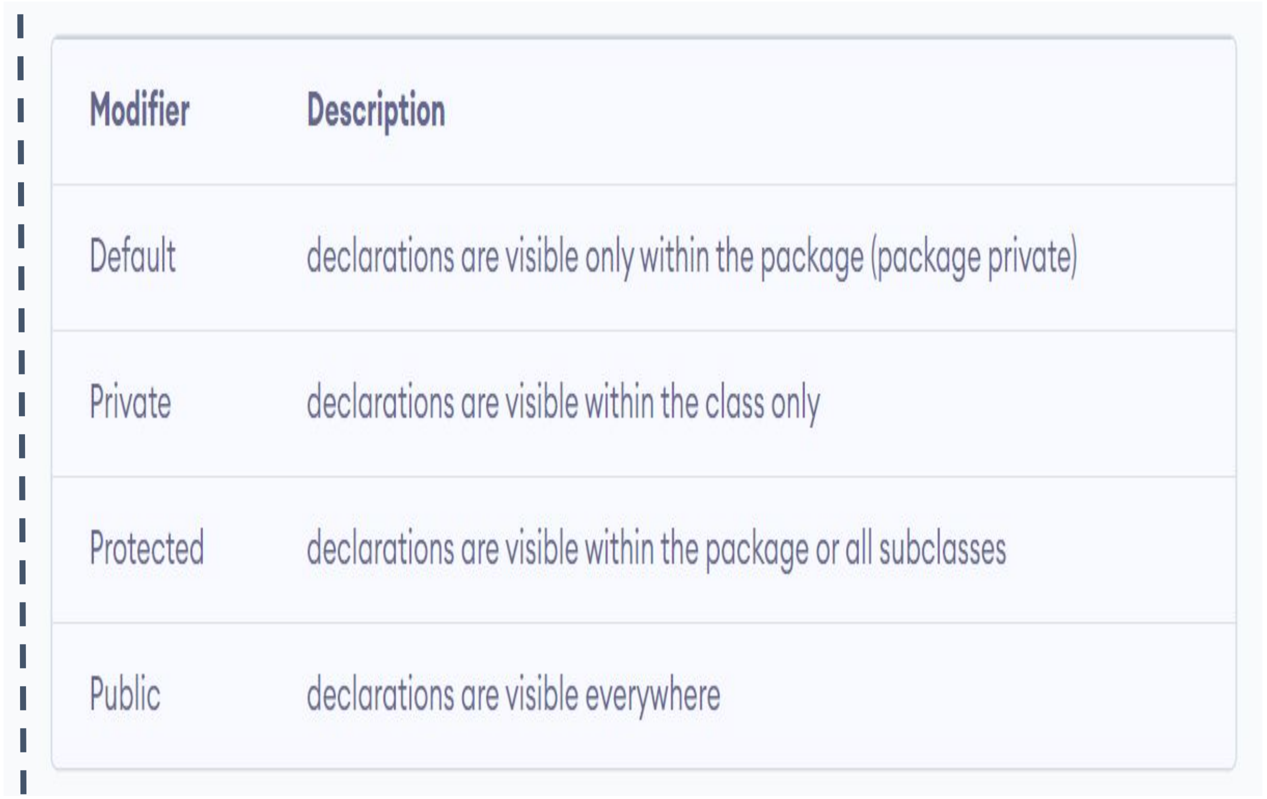
Member Variables (Class level scope)

- These are the variables that are declared inside the class but outside any function, and have class-level scope.
- We can access these variables anywhere inside the class.
- Note that the access specifier of a member variable does not affect the scope within the class.

Member Variables (Class level scope)

- Java allows us to access member variables outside the class with the following rules:

Access Specifier	Package	Subclass
public	Yes	Yes
protected	Yes	Yes
private	No	No
default	Yes	No



Modifier	Description
Default	declarations are visible only within the package (package private)
Private	declarations are visible within the class only
Protected	declarations are visible within the package or all subclasses
Public	declarations are visible everywhere

Java Package

- A **java package** is a group of similar types of classes, interfaces and sub-packages.
- Package in java can be categorized in two form, built-in package and user-defined package.
- There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.
- However, here we will proceed to creating and using user-defined packages.

Advantages of Java Package

- Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- Java package provides access protection.
- Java package removes naming collision.

Java Package Syntax

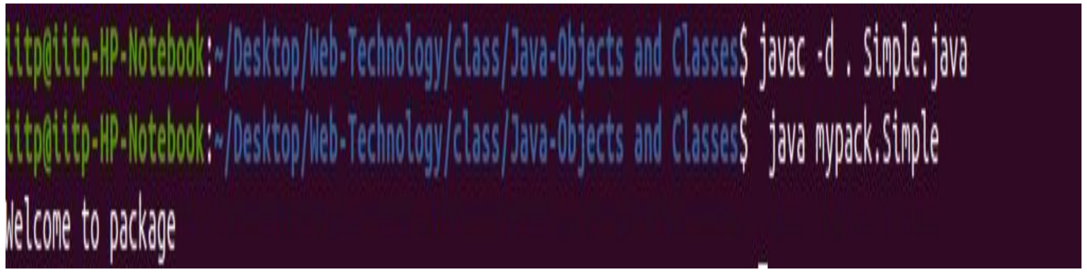
- The **package keyword** is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple
{
        public static void main(String args[])
        {
                System.out.println("Welcome to package");
        }
}
```

Java Package Syntax

- The **package keyword** is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple
{
        public static void main(String args[])
        {
                System.out.println("Welcome to package");
        }
}
```



```
itp@itp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$ javac -d . Simple.java
itp@itp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$ java mypack.Simple
Welcome to package
```

Steps to ***Compile***: **javac -d . Simple.java**

[**-d** switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), C:\abc (in case of windows) etc. If you want to keep the package within the same directory, you can use **.** (dot)]

Run: **java mypack.Simple**

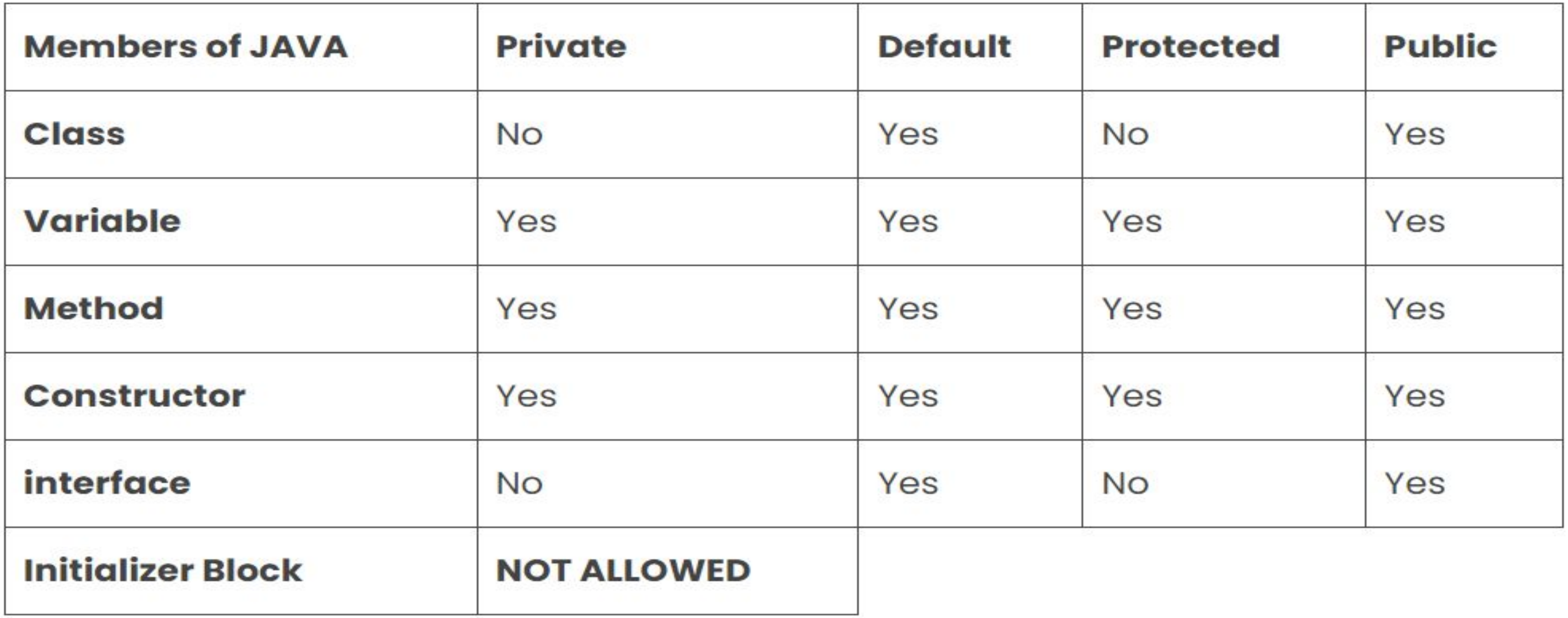
Access Modifiers in Java

- There are two types of modifiers in Java: **access modifiers** and **non-access modifiers.**
- 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.

Access Modifiers in Java

- There are **four** types of Java access modifiers:
    - **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
    - **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.
    - **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.
    - **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.
- There are many non-access modifiers, such as **static**, **abstract**, **synchronized**, **native**, **volatile**, **transient**, etc.

List of Java Members Which can be assigned with Access Modifiers



Members of JAVA	**Private**	**Default**	**Protected**	**Public**
Class	No	Yes	No	Yes
Variable	Yes	Yes	Yes	Yes
Method	Yes	Yes	Yes	Yes
Constructor	Yes	Yes	Yes	Yes
interface	No	Yes	No	Yes
Initializer Block	**NOT ALLOWED**			

Default Access Modifier

- If we do not explicitly specify any access modifier for classes, methods, variables, etc, then by default the default access modifier is considered. For example,

```
package defPack;
class Log {
    void message(){
        System.out.println("This is a
message");
    }
}
```

- Here, the `Log` class has the default access modifier.
- And the class is visible to all the classes that belong to the `defPack` package.
- However, if we try to use the `Log` class in another class outside of `defPack`, we will get a compilation error.

Private Access Modifier

- When variables and methods are declared private, they cannot be accessed outside of the class. For example,

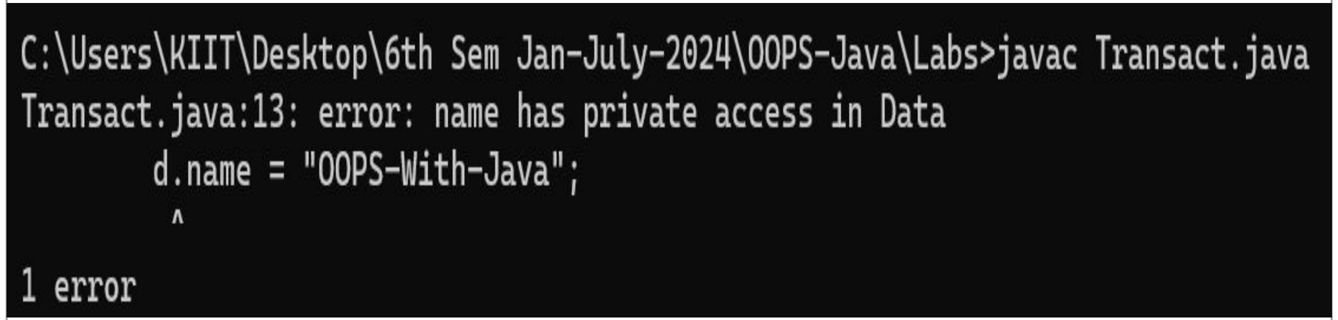
```
class Data {
    // private variable
    private String name;
}
public class Transact{
    public static void main(String[] main){

        // create an object of Data
        Data d = new Data();
        // access private variable and field
from another class
        d.name = "OOPS-With-Java";
    }
}
```

- In the program, we have declared a private variable named **name**. When we run the program, what will be the output: ??????

Private Access Modifier (Contd.)

- When variables and methods are declared private, they cannot be accessed outside of the class. For example,

```
class Data {
    // private variable
    private String name;
}
public class Transact{
    public static void main(String[] main){

        // create an object of Data
        Data d = new Data();
        // access private variable and field 
from another class
        d.name = "OOPS-With-Java";
    }
}
```



```
C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>javac Transact.java
Transact.java:13: error: name has private access in Data
        d.name = "OOPS-With-Java";
         ^
1 error
```

The error is generated because we are trying to access the private variable of the Data class from the Transact class.

Private Access Modifier (Contd.)

- **You** might be wondering what if we need to **access** those private variables. In this case, **we can use the getters and setters method**. For example,

```
class Data {
    private String name;

    // getter method
    public String getName() {
        return this.name;
    }
    // setter method
    public void setName(String name) {
        this.name= name;
    }
}
```

```
    public class Transact {
        public static void main(String[] main){
            Data d = new Data();

// access the private variable using the getter and
setter
            d.setName("OOPS-With-Java");
            System.out.println(d.getName());
        }
    }
```

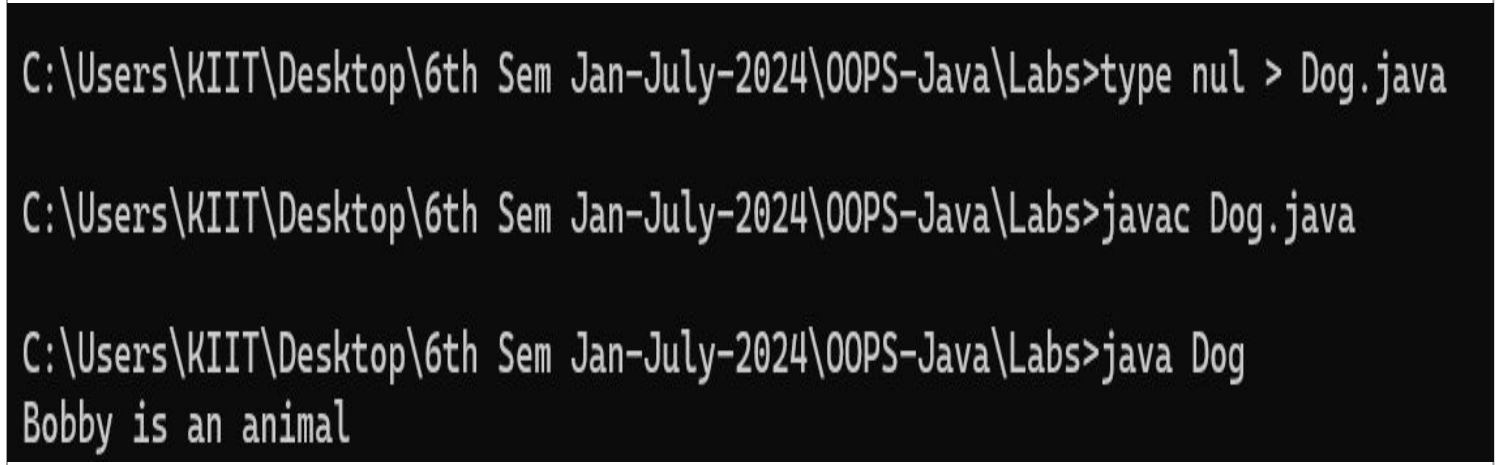
Private Access Modifier (Contd.)

- For the program in the previous slide:
    - We have a private variable named name.
        - In order to access the variable from the outer class, we have used methods: getName() and setName(). These methods are called getter and setter in Java.
    - Here, we have used the setter method (setName()) to assign value to the variable and the getter method (getName()) to access the variable.
    - We have used **this** keyword inside the setName() to refer to the variable of the class.

Protected Access Modifier

- When methods and data members are declared protected, we can access them within the same package as well as from subclasses. For example,

```
class Animal {
    // protected method
    protected void display() {
        System.out.println("Bobby is an animal");
    }
}

class Dog extends Animal {
    public static void main(String[] args) {

        // create an object of Dog class
        Dog dog = new Dog();
         // access protected method
        dog.display();
    }
}
```



```
C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>type nul > Dog.java

C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>javac Dog.java

C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>java Dog
Bobby is an animal
```

Protected Access Modifier (Contd.)

- For the example program in the previous slide,
    - In the above example, we have a protected method named display() inside the Animal class. The Animal class is inherited by the Dog class.
    - We then created an object dog of the Dog class. Using the object we tried to access the protected method of the parent class.
    - Since protected methods can be accessed from the child classes, we are able to access the method of Animal class from the Dog class.

Public Access Modifier (Contd.)

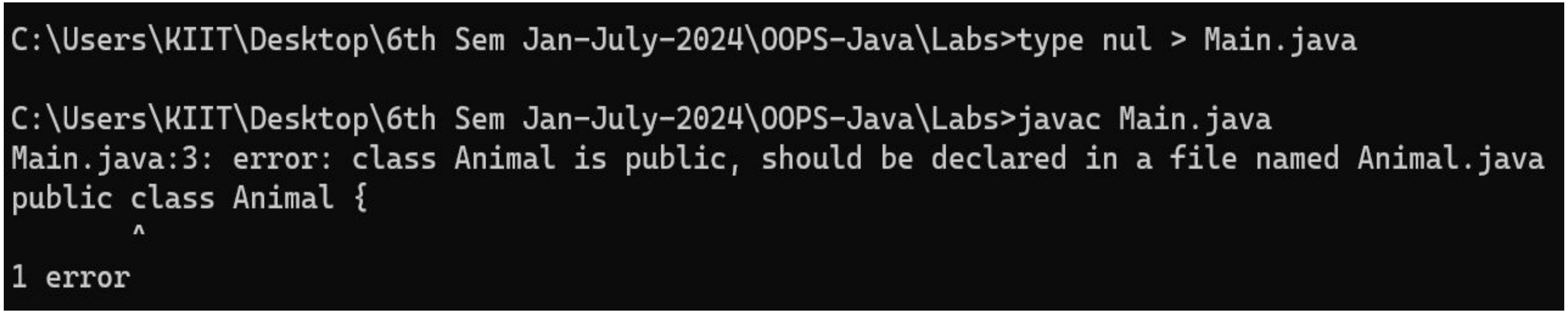
- When methods, variables, classes, and so on are declared public, then we can access them from anywhere. The public access modifier has no scope restriction. For example,

```
// public class
public class Animal {
    // public variable
    public int legCount;

    // public method
    public void display() {
        System.out.println("I am an animal.");
        System.out.println("I have " + legCount + "
legs.");
    }
}
```

```
// Main.java
public class Main {
    public static void main( String[] args ) {
        // accessing the public class
        Animal animal = new Animal();

        // accessing the public variable
        animal.legCount = 4;
        // accessing the public method
        animal.display();
    }
}
```

Public Access Modifier (Contd.)

For our example in the previous slide,



```
C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>type nul > Main.java

C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>javac Main.java
Main.java:3: error: class Animal is public, should be declared in a file named Animal.java
public class Animal {
       ^
1 error
```

- The public class Animal is accessed from the Main class.
- The public variable legCount is accessed from the Main class.
- The public method display() is accessed from the Main class.

Public Access Modifier (Contd.)

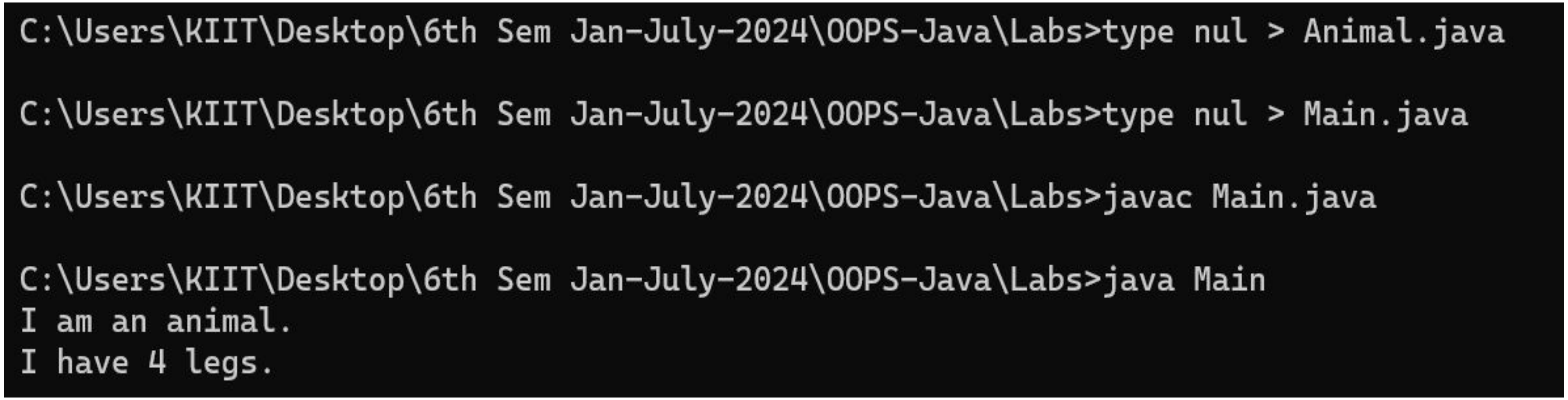
- So, for example, as a variation, having the Animal class stored in a separate file called Animal.java and then calling the object of this class in another file called Main.java, the output becomes:

```
// Animal.java file
public class Animal {
    // public variable
    public int legCount;

    // public method
    public void display() {
        System.out.println("I am an animal.");
        System.out.println("I have " + legCount + "
legs.");
    }
}
```

```
// Main.java
public class Main {
    public static void main( String[] args ) {
        // accessing the public class
        Animal animal = new Animal();

        // accessing the public variable
        animal.legCount = 4;
        // accessing the public method
        animal.display();
    }
}
```

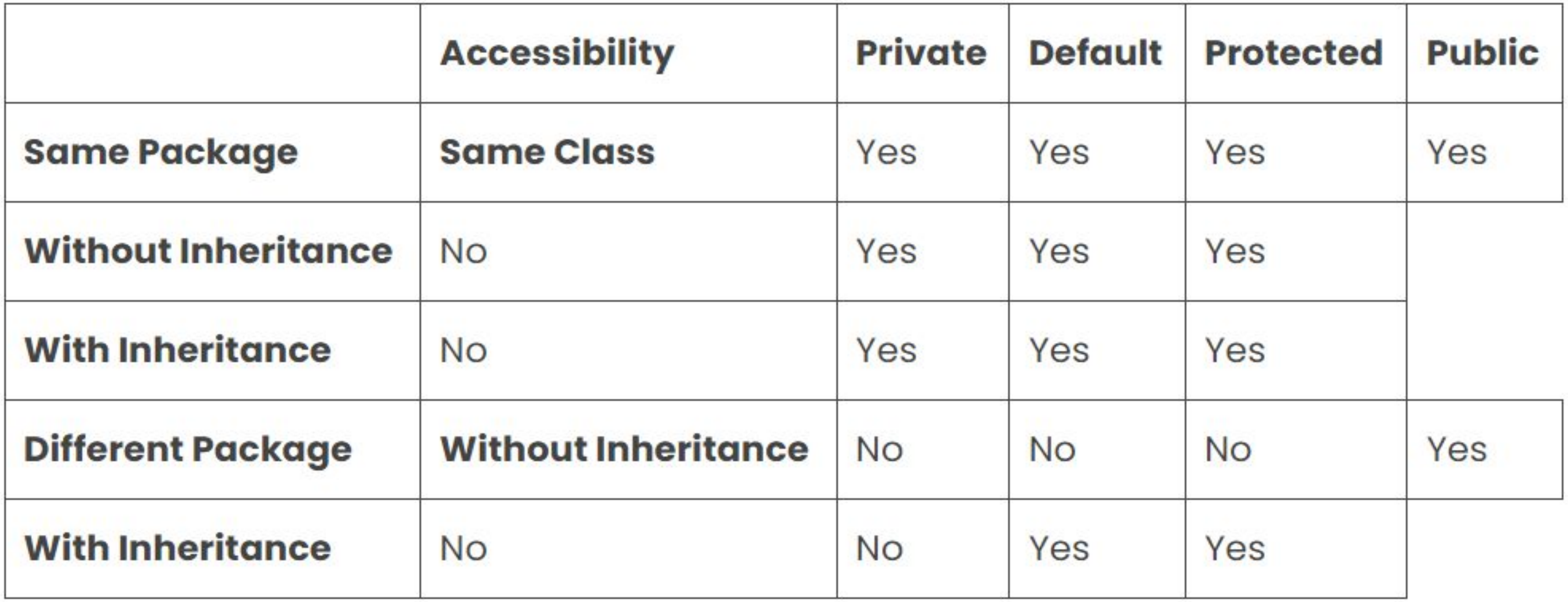
Public Access Modifier (Contd.)

For our example in the previous slide,



```
C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>type nul > Animal.java

C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>type nul > Main.java

C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>javac Main.java

C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>java Main
I am an animal.
I have 4 legs.
```

Scope of Access Modifiers



	Accessibility	Private	Default	Protected	Public
Same Package	**Same Class**	Yes	Yes	Yes	Yes
Without Inheritance	No	Yes	Yes	Yes	
With Inheritance	No	Yes	Yes	Yes	
Different Package	**Without Inheritance**	No	No	No	Yes
With Inheritance	No	No	Yes	Yes	

Member Variables (Class level scope)

```
public class WTjavavariablesdemo
{

        int age;
        private String name;

        void displayName()
        {
                //statements
        }
        int dispalyAge()
        {
                //statements
        }
        char c;
}
```

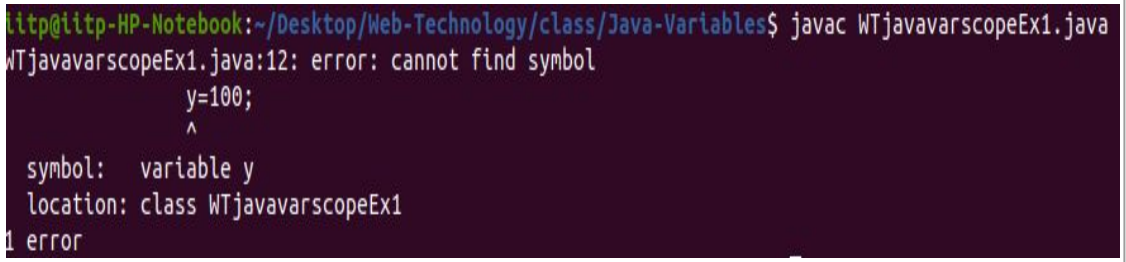
} variables declared inside the class have class level scope

}variables declared inside the class have class level scope

Member Variables (Class level scope) - Example - I

```
public class WTjavavarscopeEx1
{
        public static void main(String args[])
        {
                int x=10;
                {
                        //y has limited scope to this block only
                        int y=20;
                        System.out.println("Sum of x+y = " + (x+y));
                }
                //here y is unknown
                y=100;
                //x is still known
                x=50;
        }
}
```

On trying to compile the program, the output pops up the following error:-



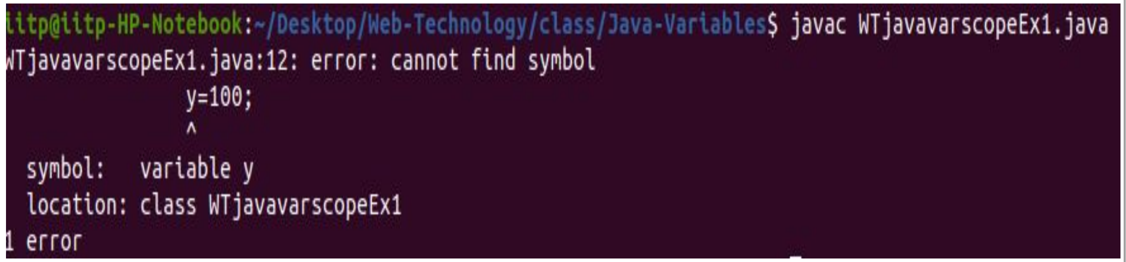
```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ javac WTjavavarscopeEx1.java
WTjavavarscopeEx1.java:12: error: cannot find symbol
                y=100;
                ^
  symbol:   variable y
  location: class WTjavavarscopeEx1
1 error
```

The error refers to: We see that **y=100 is unknown.**

Member Variables (Class level scope) - Example - I

```
public class WTjavavarscopeEx1
{
        public static void main(String args[])
        {
                int x=10;
                {
                        //y has limited scope to this block only
                        int y=20;
                        System.out.println("Sum of x+y = " + (x+y));
                }
                //here y is unknown
                y=100;
                //x is still known
                x=50;
        }
}
```

On trying to compile the program, the output pops up the following error:-



```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ javac WTjavavarscopeEx1.java
WTjavavarscopeEx1.java:12: error: cannot find symbol
                y=100;
                ^
  symbol:   variable y
  location: class WTjavavarscopeEx1
1 error
```

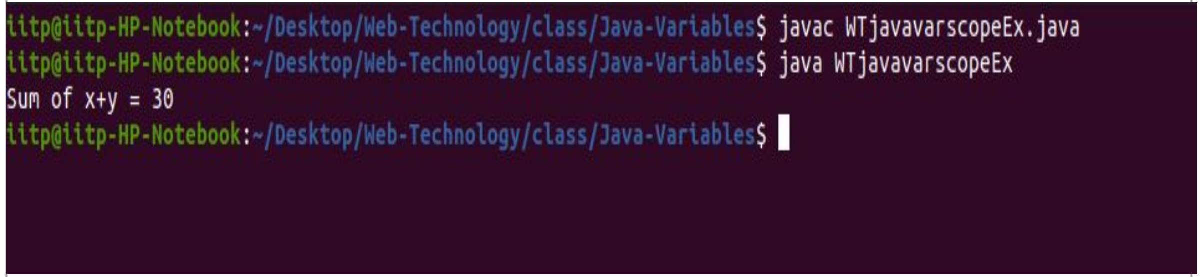
We see that y=100 is unknown.

Now, If we want to compile and run the above program: then, we need *to remove or comment the statement y=100.*

Member Variables (Class level scope) - Example - I

```
public class WTjavavarscopeEx
{
        public static void main(String args[])
        {
                int x=10;
                {
                        //y has limited scope to this block only
                        int y=20;
                        System.out.println("Sum of x+y = " + (x+y));
                }
                //here y is unknown
                //y=100;
                //x is still known
                x=50;
        }
}
```

After commenting out the variable y = 100



```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ javac WTjavavarscopeEx.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ java WTjavavarscopeEx
Sum of x+y = 30
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$
```

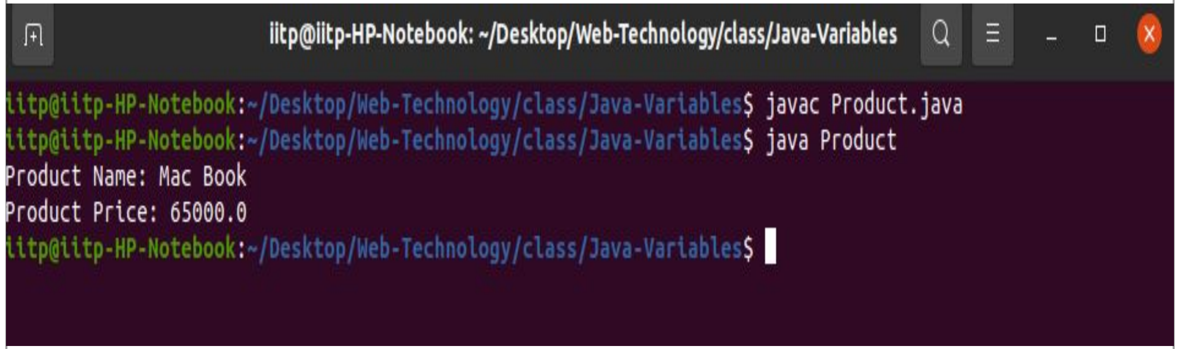
We see that now the program runs successfully, and prints the sum x and y together

Member Variables (Class level scope)

- There is another variable named an instance variable.
- These are declared inside a class but outside any method, constructor, or block.
- When an instance variable is declared using the keyword static, the instance variable is known as a static variable.
- Their scope is class level but visible to the method, constructor, or block that is defined inside the class.

Example-II

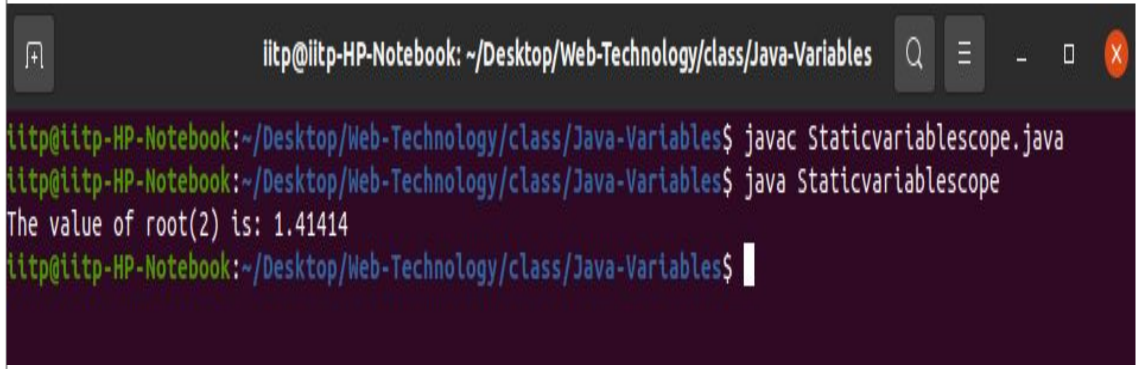
```
public class Product
{
	//variable visible to any child class
	public String pName;
	//variable visible to Product class only
	private double pPrice;
	//creating a constructor and parsed product name as a
parameter
	public Product (String pname)
	{
		pName = pname;
	}
	//function sets the product price
	public void setPrice(double pprice)
	{
		pPrice= pprice;
	}
	//method prints all product info
	public void getInfo()
	{
		System.out.println("Product Name: " +pName );
		System.out.println("Product Price: " +pPrice);
	}
	public static void main(String args[])
	{
		Product pro = new Product("Mac Book");
		pro.setPrice(65000);
		pro.getInfo();
	}
}
```


iitp@iitp-HP-Notebook: ~/Desktop/Web-Technology/class/Java-Variables

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ javac Product.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ java Product
Product Name: Mac Book
Product Price: 65000.0
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$
```

Static variables

```
public class Staticvariablescope
{
        //declaring a private static variable
        private static double rootvalue;
        //declaring a constant variable
        public static final String rootconstant = "root(2)";
        public static void main(String args[])
        {
                rootvalue = 1.41414;
                System.out.println("The value of " + rootconstant + " is:
" + rootvalue);
        }
}
```



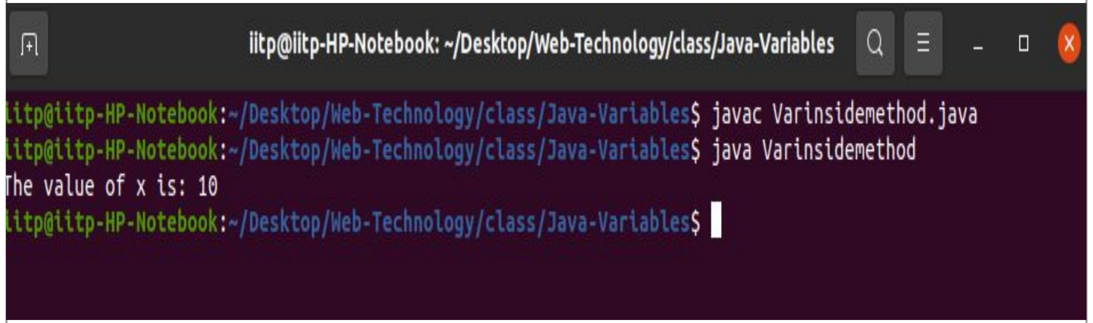
```
iitp@iitp-HP-Notebook: ~/Desktop/Web-Technology/class/Java-Variables
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ javac Staticvariablescope.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ java Staticvariablescope
The value of root(2) is: 1.41414
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ 
```

Local variables (Method Level Scope)

- These variables that are declared inside a method, constructor, or block, and have a method-level or block-level scope and cannot be accessed outside of the structure in which it is defined.
- Variables declared inside a pair of curly braces {} have block-level scope.

Local variables (Method Level Scope): Declaring variables inside a method example

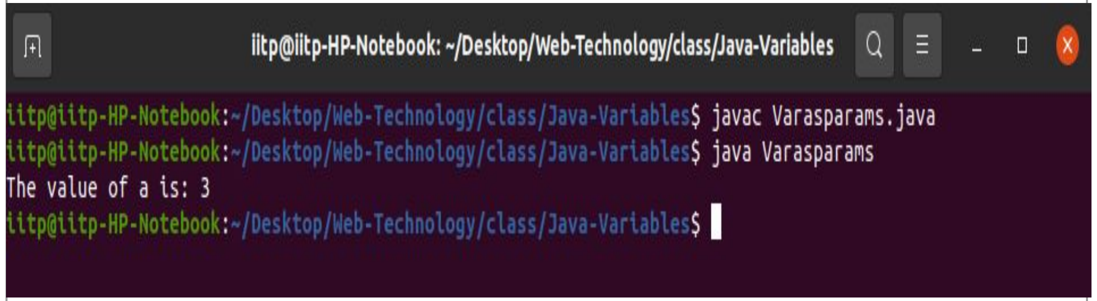
```
public class Varinsidemethod
{
        void show()
        {
                //variable declared inside a method has method
level scope
                int x=10;
                System.out.println("The value of x is: "+x);
        }
        public static void main(String args[])
        {
                Varinsidemethod dc = new Varinsidemethod();
                dc.show();
        }
}
```



```
iitp@iitp-HP-Notebook: ~/Desktop/Web-Technology/class/Java-Variables
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ javac Varinsidemethod.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ java Varinsidemethod
The value of x is: 10
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$
```

Local variables (Method Level Scope): Variables passed as parameters to a method example

```
public class Varasparams
{
        private int a;
        public void setNumber(int a)
        {
                this.a = a;
                System.out.println("The value of a is: "+a);
        }
        public static void main(String args[])
        {
                Varasparams vp = new Varasparams();
                vp.setNumber(3);
        }
}
```

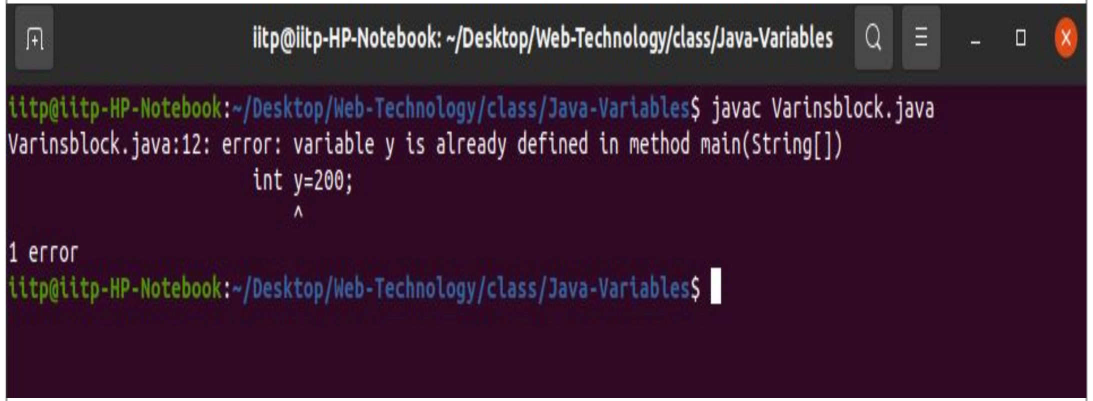


```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ javac Varasparams.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ java Varasparams
The value of a is: 3
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$
```

this keyword differentiates between the class variable and local variable.

Local variables (Method Level Scope): Declaring variables inside a block example

```
public class Varinsblock
{
	public static void main(String args[])
	{
		int x=4;
		{
			//y has limited scope to this block only
			int y=100;
			System.out.println("Sum of x+y = " + (x+y));
			y=10;
			//gives error, already defined
			int y=200;
		}
	//creates a new variable
	int y;
	}
}
```

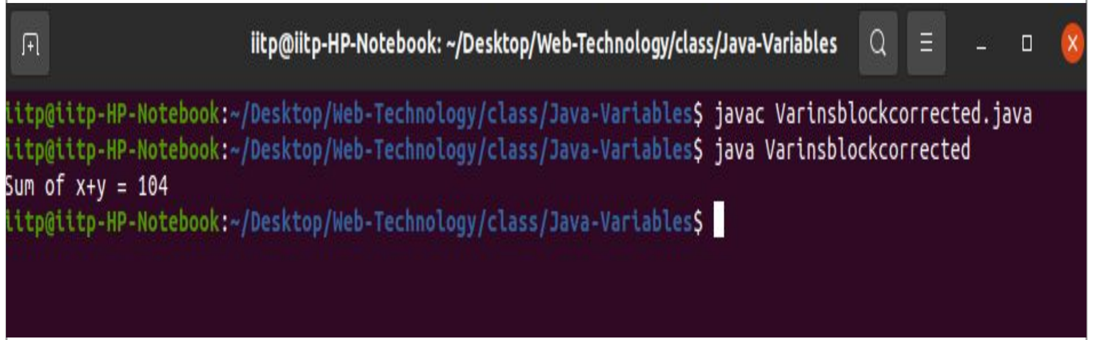


iitp@iitp-HP-Notebook: ~/Desktop/Web-Technology/class/Java-Variables

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ javac Varinsblock.java
Varinsblock.java:12: error: variable y is already defined in method main(String[])
                        int y=200;
                            ^
1 error
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ 
```

Local variables (Method Level Scope): Declaring variables inside a block example

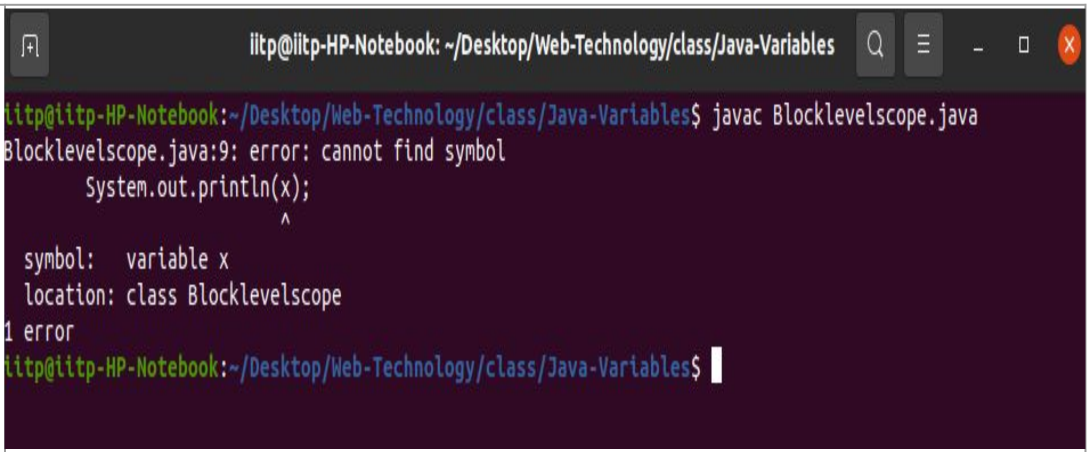
```
public class Varinsblockcorrected
{
        public static void main(String args[])
        {
                int x=4;
                {
                        //y has limited scope to this block only
                        int y=100;
                        System.out.println("Sum of x+y = " + (x+y));
                        y=10;
                        //gives error, already defined
                        //int y=200;
                }
        //creates a new variable
        int y;
        }
}
```


iitp@iitp-HP-Notebook: ~/Desktop/Web-Technology/class/Java-Variables

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ javac Varinsblockcorrected.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ java Varinsblockcorrected
Sum of x+y = 104
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$
```

Local variables (Method Level Scope): Declaring variables inside a block example -I

```
public class Blocklevelscope
{
        public static void main(String args[])
        {
                for (int x = 0; x < 10; x++)
                {
                        System.out.println(x);
                }
        System.out.println(x);
        }
}
```

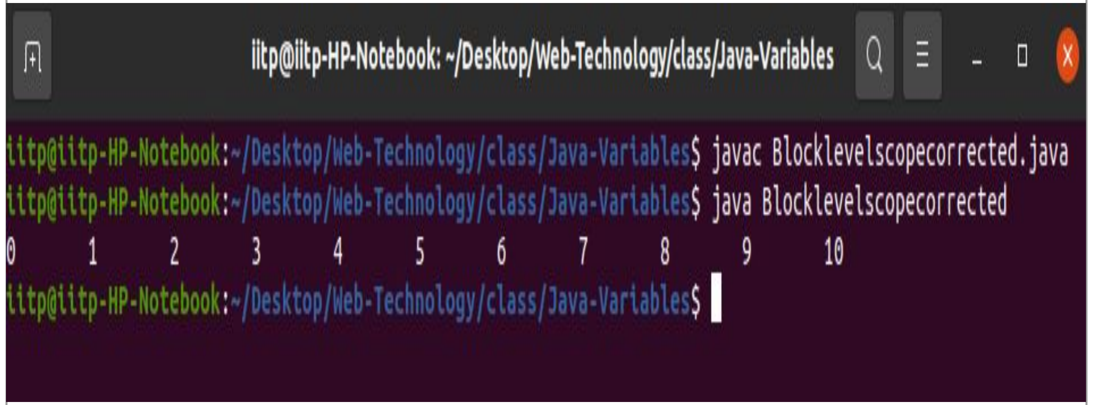

iitp@iitp-HP-Notebook: ~/Desktop/Web-Technology/class/Java-Variables

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ javac Blocklevelscope.java
Blocklevelscope.java:9: error: cannot find symbol
        System.out.println(x);
                           ^
  symbol:   variable x
  location: class Blocklevelscope
1 error
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$
```

When we run the above program, it shows an error at line 9, **cannot find symbol** because we have tried to print the variable x that is declared inside the loop.

Local variables (Method Level Scope): Declaring variables inside a block example -I

```
public class Blocklevelscopecorrected
{
	public static void main(String args[])
	{
		//To resolve the error from previous slide, we need
//to declare the variable x just before the for loop.
		int x;
		for (x = 0; x < 10; x++)
		{
			//prints 0 to 9
			System.out.print(x+"\t");
		}
		//prints 10
		System.out.println(x);
	}
}
```



iitp@iitp-HP-Notebook: ~/Desktop/Web-Technology/class/Java-Variables

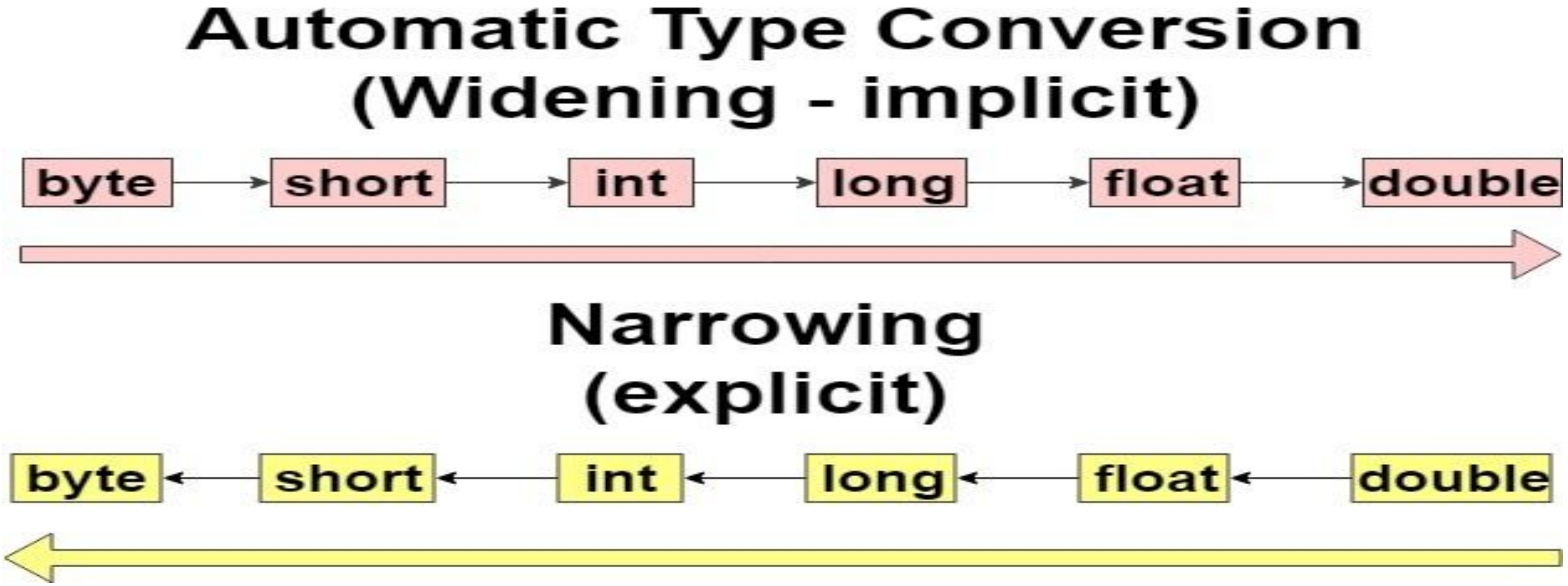
```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ javac Blocklevelscopecorrected.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$ java Blocklevelscopecorrected
0	1	2	3	4	5	6	7	8	9	10
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Variables$
```

Typecasting

- Type Casting in Java is all about assigning a value of one type to a variable of another type.
- When we assign value of one data type to another, the two types **might not be** compatible with each other.
- If the data types are compatible, then Java will perform the conversion automatically known as **Automatic Type Conversion (Widening)** and if not then they need to be casted or converted **explicitly(narrowing)**.

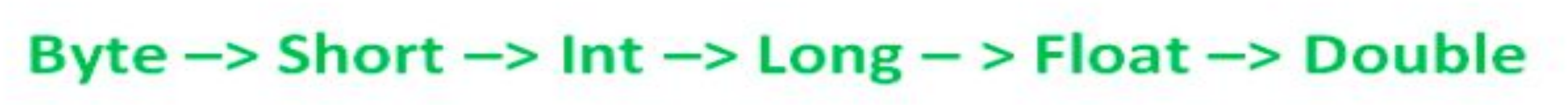
Typecasting : Two Types

1. Automatic Type Conversion (Widening – implicit)
2. Narrowing (Explicit)



Automatic / Widening / Implicit Type conversion

- Widening conversion takes place when two data types are automatically converted. This happens when:
  - The two data types are compatible.
  - When we assign value of a smaller data type to a bigger data type

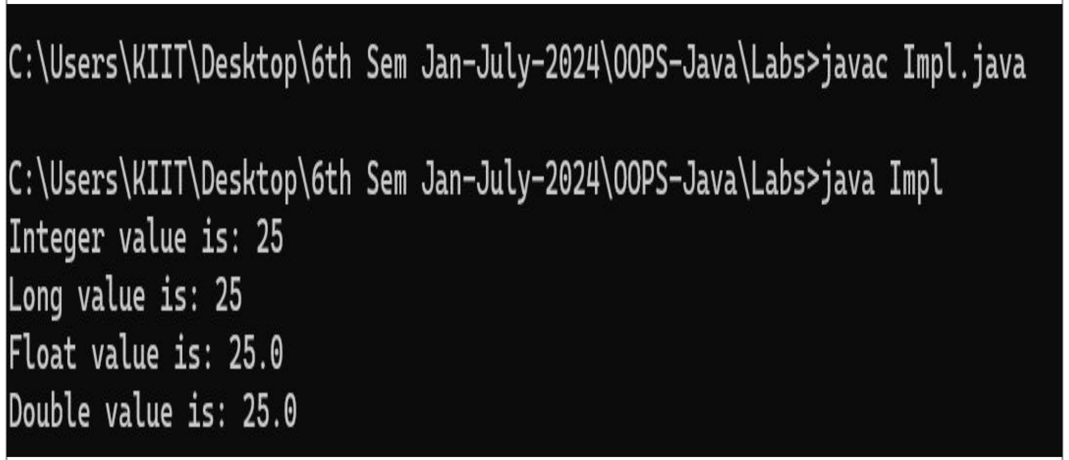


Byte –> Short –> Int –> Long – > Float –> Double

- byte ➔ short, int, long, float, double
- short ➔ int, long, float, double
- int ➔ long, float, double
- long ➔ float, double
- float ➔ double

Automatic conversion of compatible data types

```
public class Impl
{
  public static void main(String[] args)
  {
    int var = 25;
    long longVariable = var;
    float floatVariable = longVariable;
    double doubleVariable = floatVariable;
    System.out.println("Integer value is: " +var);
    System.out.println("Long value is: " +longVariable);
    System.out.println("Float value is: " +floatVariable);
    System.out.println("Double value is: "
+doubleVariable);
  }
}
```



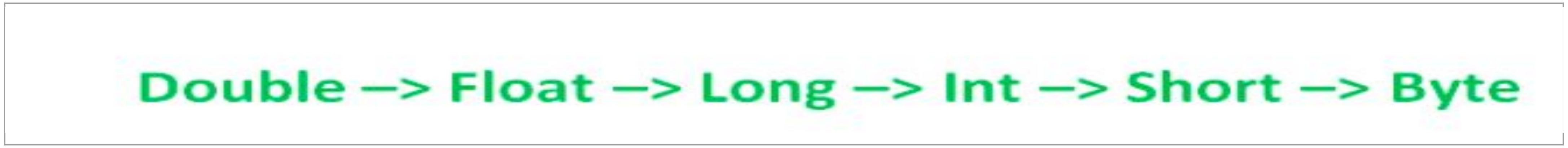
```
C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>javac Impl.java

C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>java Impl
Integer value is: 25
Long value is: 25
Float value is: 25.0
Double value is: 25.0
```

Explicit / Narrowing Type Conversion

- If we want to assign a value of larger data type to a smaller data type we perform explicit type casting or narrowing.
    - This is useful for incompatible data types where automatic conversion cannot be done.
    - Here, the target type specifies the desired type to convert the specified value to.

Explicit / Narrowing Type Conversion

- (targetType) value
- double d=12.34D;
- float f=(float) d;

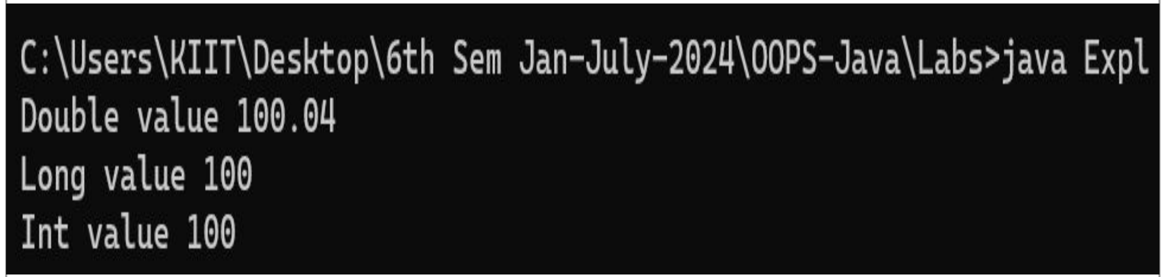


Double –> Float –> Long –> Int –> Short –> Byte

If the whole number is too large to fit into the target type, then value will be reduced modulo the target type range

Explicit / Narrowing Type Conversion

```
class Expl
{
    public static void main(String[] args)
    {
        double d = 100.04;

        //explicit type casting
        long l = (long)d;

        //explicit type casting
        int i = (int)l;
        System.out.println("Double value "+d);

        //fractional part lost
        System.out.println("Long value "+l);

        //fractional part lost
        System.out.println("Int value "+i);
    }
}
```



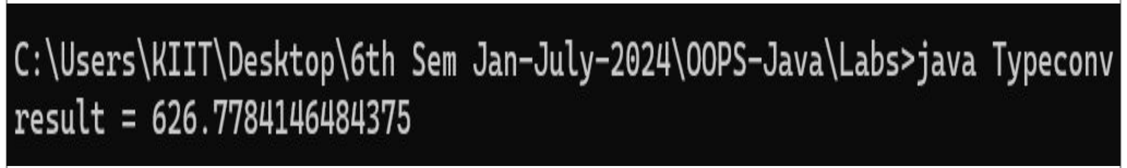
```
C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>java Expl
Double value 100.04
Long value 100
Int value 100
```

Promotion of data types in expressions

- While evaluating expressions, the intermediate value may exceed the range of operands and hence the expression value will be promoted. Some conditions for type promotion are:
- byte, short, or char are always promoted to int
- if one operand is long or float or double, the whole expression is promoted to long or float or double respectively.

Promotion of data types in expressions

```
class Typeconv
{
	public static void main(String args[])
	{
		byte b = 42;
		char c = 'a';
		short s = 1024;
		int i = 50000;
		float f = 5.67f;
		double d = .1234;

		// The Expression
		double result = (f * b) + (i / c) - (d * s);

		//Result after all the promotions are done
		System.out.println("result = " + result);
	}
}
```



```
C:\Users\KIIT\Desktop\6th Sem Jan-July-2024\OOPS-Java\Labs>java Typeconv
result = 626.7784146484375
```

References

1. https://www.javatpoint.com/difference-between-jdk-jre-and-jvm
2. https://www.oracle.com/java/technologies/architecture-neutral-portable-robust.html#:~:text=The%20Java%20environment%20itself%20is,which%20is%20essentially%20POSIX%2Dcompliant.
3. https://www.oreilly.com/library/view/the-java-language/9780133260335/ch03lev1sec6.html#:~:text=White%20space%20is%20defined%20as,terminator%20characters%20(%C2%A73.4).
4. https://www.geeksforgeeks.org/type-conversion-java-examples/
5. https://www.javatpoint.com/scope-of-variables-in-java
6. https://www.mygreatlearning.com/blog/the-access-modifiers-in-java/
7. https://simplesnippets.tech/typecasting-in-java/