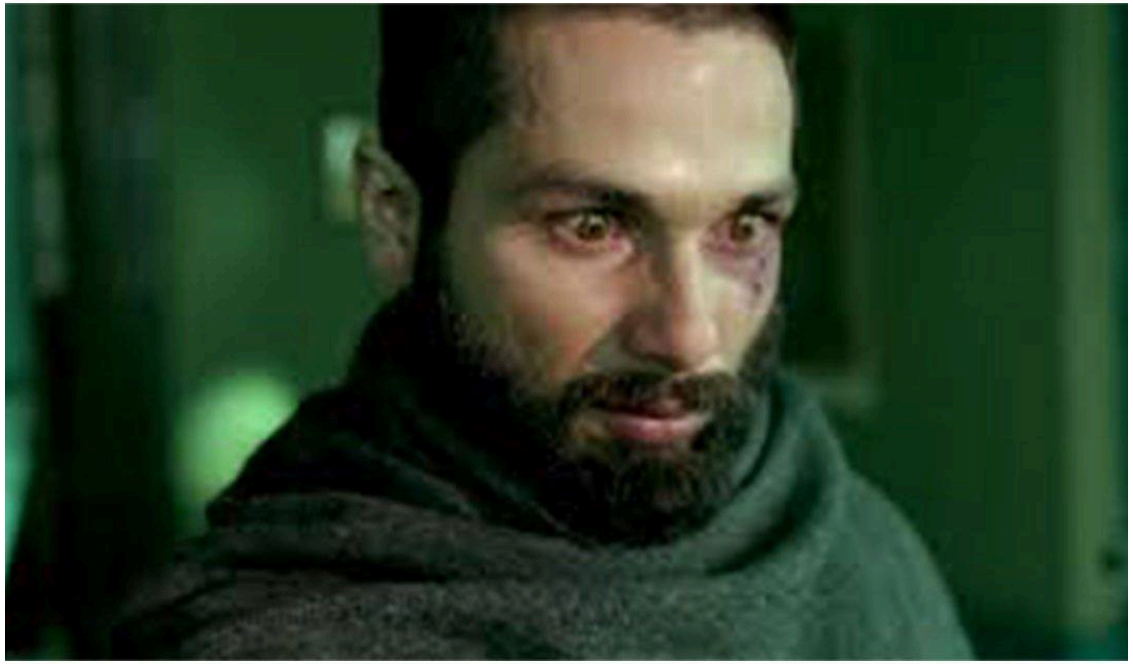
PROGRAMMING

Lecture 5

Sushil Paudel

SEMICOLON

When you debug 30 errors and 50 warnings All day long



and find it was all because of missing ;

PREVIOUS TOPICS

- Control Statements – If Else
- Control Statement – Switch
- Control Statement Loop – For, While, Do-While
- Break and Continue

TODAY'S TOPIC

- Scanner – User Input
- Constructor
- Overloading
- Variable

SCANNER – USER INPUT

- The Scanner class is a class in **java.util**, which allows the user to read values of various types.
- It can parse primitive types and strings.
- A scanning operation may block waiting for input.
- Java Scanner class must be imported at the top before class declaration.
- There will be compile time error if you use Scanner class without importing.

SCANNER - IMPORT

import is a keyword. **import** keyword is used to **import** built-in and user-defined packages into your **java** source file so that your class can refer to a class that is in another package by directly using its name. **import** should be done before creating class.

Syntax:

```
import package.name.ClassName; // To import a certain class only
import package.name.* // To import the whole package
```

Importing Scanner class

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

SCANNER - USER INPUT

Example: Reading integer number

```
Scanner scan = new Scanner(System.in);
int value = scan.nextInt();
```

Example: Reading String value

```
Scanner scan = new Scanner(System.in);
String value = scan.nextLine();
```

EXAMPLE

```java
import java.util.Scanner; // 1. Import Scanner class

public class StudentData {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in); // 2. Create object of Scanner class

        System.out.println("Enter name");
        String name = scan.nextLine(); // 3(a). Input the string value
        System.out.println("Name  is:  " + name);

        System.out.println("Enter  salary"); // 3(b). Input the double value
        double salary = scan.nextDouble();
        System.out.println("Salary  is:  " + salary);

    }
}
```

SCANNER METHODS

public String nextLine()	it moves the scanner position to the next line and returns the value as a string.
public byte nextByte()	it scans the next token as a byte.
public short nextShort()	it scans the next token as a short value.
public int nextInt()	it scans the next token as an int value.
public long nextLong()	it scans the next token as a long value.
public float nextFloat()	it scans the next token as a float value.
public double nextDouble()	it scans the next token as a double value.

CREATING OBJECT IN JAVA

Syntax:

```
ClassName object = new ClassName();
```

Example:

```
Student object = new Student();
```

CONSTRUCTOR

- It is a special type of method which is used to initialize the object.
- It is called when an instance of the class is created.
- Every time an object is created using the new() keyword, at least one constructor is called.
- A constructor allows you to provide initial values for class fields when you create the object.

RULES OF CONSTRUCTOR

- Constructor name must be the same as its class name
- A Constructor must have no explicit return type

EXAMPLE:

```
public class <Class_Name> {
        public <Class_Name> () {
        }
}
```

TYPES OF CONSTRUCTOR

There are three types of Constructor in java, they are:

1. Default constructor
2. No-Argument constructor:
3. Parameterized constructor

DEFAULT CONSTUCTOR

- If you do not implement any constructor in your class, Java compiler inserts a default constructor into your code on your behalf.
- This constructor is known as default constructor.
- You would not find it in your source code(the java file) as it would be inserted into the code during compilation and exists in .class file.

NO-ARGS CONSTRUCTOR

- A constructor that has no parameter is known as NO-ARGUEMENT constructor.
- If we don’t define a constructor in a class, then compiler creates **default constructor** for the class.
- If we write a constructor with arguments or no-arguments then the compiler does not create a default constructor.

NO-ARGS CONSTRUCTOR EXAMPLE

```
public class Student {

    public Student () {
        System.out.println("This is constructor");
    }

    public static void main(String[] args) {
        Student student = new Student();
    }
}
```

Output

This is constructor

PARAMETERIZED C0NSTRUCTOR

- Constructor with arguments(or you can say parameters) is known as Parameterized constructor.
- A class can have multiple parameterized constructors.

```
public class Student {

    public Student (String name) {
        System.out.println("My name is: "+name);
    }

    public static void main(String[] args) {
        Student student = new Student("Ram");
    }
}
```

OUTPUT

My name is: Ram

EXAMPLE

```
public class Addition{
    public Addition(){
        System.out.println("This is no-args constuctor");
    }

    public Addition(int a, int b){
        System.out.println(”Result from Parameterized constructor: "+(a+b));
    }

    public static void main(String[] args){
        Addition addition1 = new Addition(); // Calls no-args constructor
        Addition addition2 = new Addition(5, 8); // Calls constructor with 2 parameters
         Addition addition3 = new Addition(10, 20); // Calls constructor with 2 parameters
    }
}
```

OUTPUT

This is no-args constuctor

Result from Parameterized constructor: 13

Result from Parameterized constructor: 30

EXAMPLE

```
public class Student {

    public Student () {
        System.out.println("This is no args constructor");
    }

    public Student (String name) {
        System.out.println("My name is: "+name);
    }

    public static void main(String[] args) {
        Student ram = new Student(”Ram");
        Student someone = new Student();
    }
}
```

OUTPUT

My name is: Ram

This is no args constructor

CONSTRUCTOR OVERLOADING

- Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.
- The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

- Constructor overloading is a concept of having more than one constructor with different parameters list, in such a way so that each constructor performs a different task.

CONSTRUCTOR OVERLOADING EXAMPLE

```
public class Addition{
    public Addition(int a, int b){
        System.out.println("Sum is: "+(a+b));
    }
    public Addition(int a, int b, int c){
        System.out.println("Sum is: "+(a+b+c));
    }
    public Addition(String firstName, String lastName){
        System.out.println(firstName + " " + lastName);
    }

    public static void main(String[] args){
        Addition addition1 = new Addition(10, 8);
        Addition addition2 = new Addition(10, 20, 30);
        Addition addition3 = new Addition("Ram", "Bahadur");
    }
}
```

OUTPUT

Sum is: 18

Sum is: 60

Ram Bahadur

METHOD OVERLOADING

- Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different.
- It is similar to constructor overloading in Java, that allows a class to have more than one constructor having different argument lists.
- Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as:
- **addTwo**(int,int) for two parameters, and
- **addThree**(int a,int b,int c) for three parameters, then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

WAYS OF METHOD OVERLOADING

1. No of parameters

```
public class Addition {
    public void add(int a, int b) {
    }

    public void add(int a, int b, int c) {
    }

    public void add(int a, int b, int c, int d) {
    }
}
```

WAYS OF METHOD OVERLOADING

2. Data type of parameters

```
public class Operations {
    public void add(double a, int b) {
    }

    public void add(double a, double b) {
    }

    public void add(String a, int b, int c) {
    }
}
```

EXAMPLE

```
public class Addition{
    public void add(int a, int b){
        System.out.println("Sum is: "+(a+b));
    }
    public void add(int a, int b, int c){
        System.out.println("Sum is: "+(a+b+c));
    }
    public void test(){
        add(1,2);
        add(1,2,3);
    }
}
```

Question: What will be the output if the method test() is called?

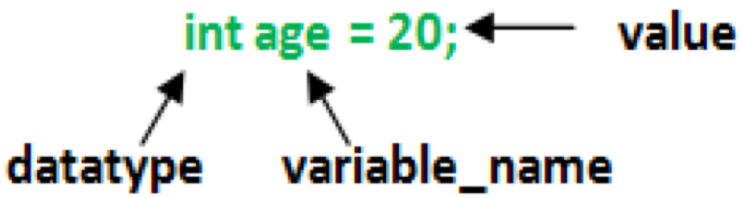
OUTPUT

Sum is: 3

Sum is: 6

VARIABLE

- Variables are containers for storing data values.
- This means that when you create a variable you reserve some space in the memory.
- The value stored in a variable can be changed during program execution.
- In Java, all the variables must be declared before use.
- Here, the value 20 is also called **literal**.



TYPES OF VARIABLES

- Local Variables
- Instance Variables
- Static Variables

LOCAL VARIABLES

- A variable defined within a block or method or constructor is called local variable.
- These variable are created when the block in entered or the function is called and destroyed after exiting from the block or when the call returns from the function.
- The scope of these variables exists only within the block in which the variable is declared. i.e. we can access these variable only within that block.

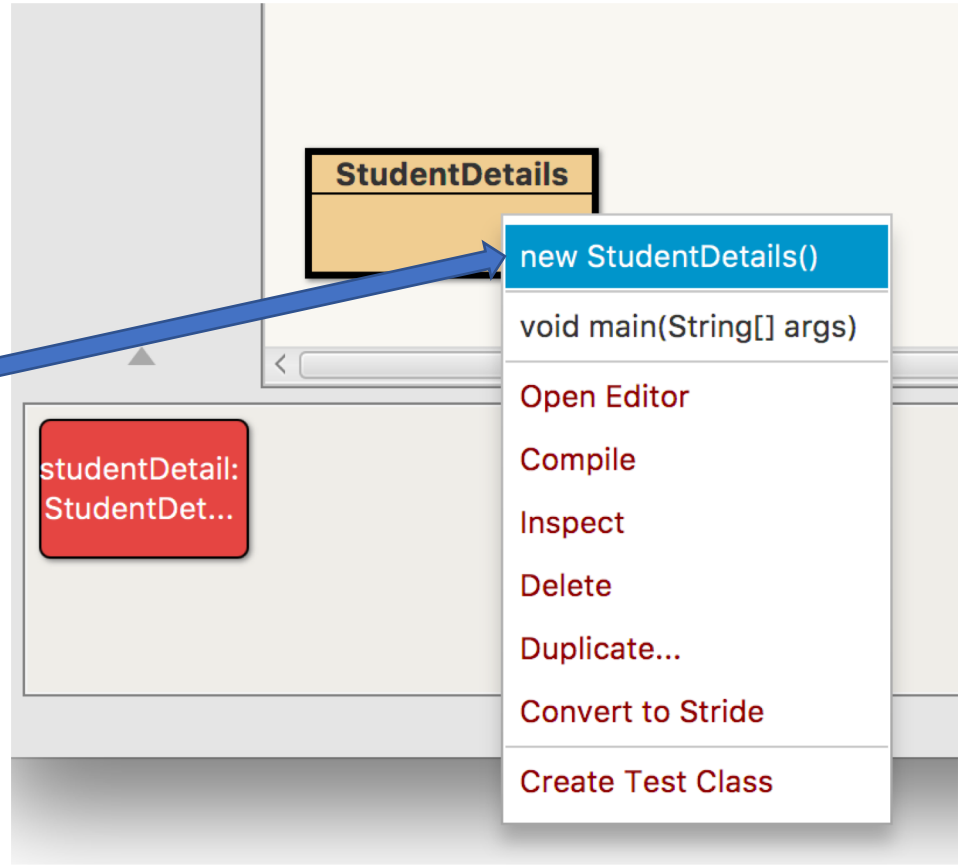
LOCAL VARIABLE EXAMPLE

```
public class StudentDetails {

    public void studentAge() {
        //local variable age
        int age = 10;
        age = age + 5;
        System.out.println("Student age is : " + age);
    }

    public static void main(String args[]) {
        StudentDetails obj = new StudentDetails();
        obj.studentAge();

        // Is the following statement correct?
       // System.out.println("Student age is : " + age);
    }
}
```



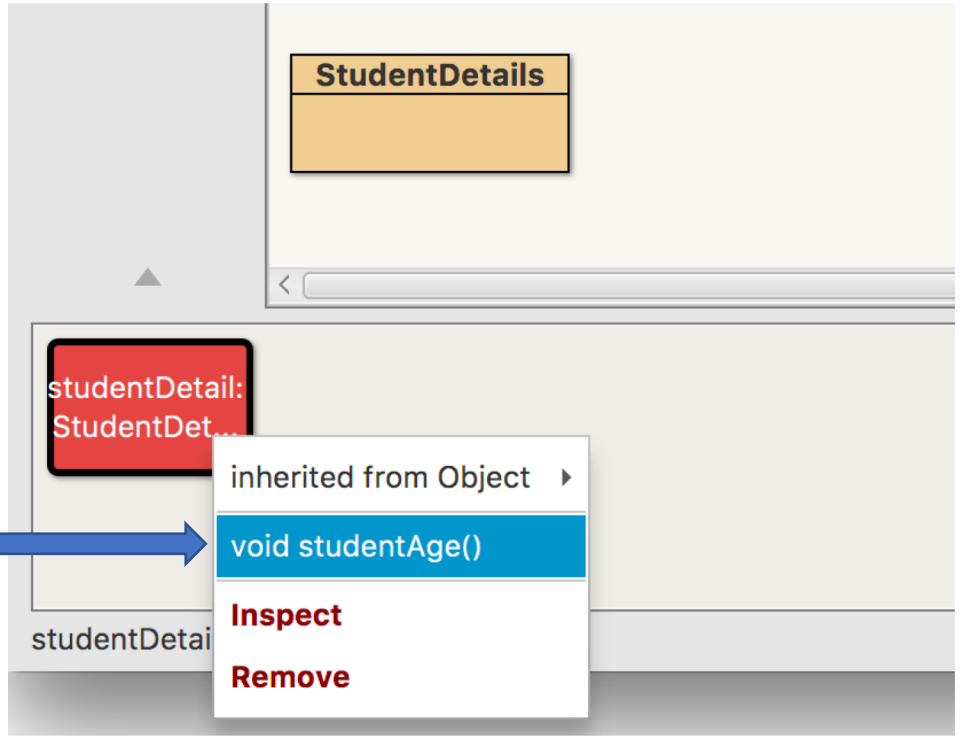
LOCAL VARIABLE EXAMPLE

```
public class StudentDetails {

    public void studentAge() {
        //local variable age
        int age = 10;
        age = age + 5;
        System.out.println("Student age is : " + age);
    }

    public static void main(String args[]) {
        StudentDetails obj = new StudentDetails();
        obj.studentAge();

        // Is the following statement correct?
       // System.out.println("Student age is : " + age);
    }
}
```



INSTANCE VARIABLE

- Instance variables are declared in a class, but outside a method, constructor or any block.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- The instance variables are visible for all methods, constructors and block in the class.

EXAMPLE

```
public class Student {

	private int id;
	public String name;

	public Student(String sname) {
		name = sname;
	}

	public void printStudentDetail() {
		System.out.println("Name: " + name);
		System.out.println("Id: " + id);
	}

	public static void main(String[] args) {
		Student student = new Student("Leo");
		student.id = 1;
		student.printStudentDetail();
	}
}
```

OUTPUT

Name: Leo

Id: 1

CLASS/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.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are created when the program starts and destroyed when the program stops.
- Static variables can be accessed by calling with the class name

  ClassName.VariableName

EXAMPLE

```
public class Student {

        // Instance variable
        public String name;
        public int rollNo;

        // Static/Class variable
        public static int total;

        // The name variable is assigned in the constructor.
        public Student(String name) {
                this.name = name;
                total++;
                rollNo++;
        }

        public void printEmp() {
                System.out.println("name  : " + name);
                System.out.println("roll No  : " + rollNo);
                System.out.println("total :" + Student.total);
        }

        public static void main(String args[]) {
                Student s1 = new Student("Ram");
                s1.printEmp();

                Student s2 = new Student("Shyam");
                s2.printEmp();
        }
}
```

OUTPUT

name : Ram

roll No : 1

total :1

name : Shyam

roll No : 1

total :2

THIS KEYWORD

- Using *this* you can refer the members of a class such as constructors, variables and methods.
- The keyword *this* is used only within instance methods or constructors.
- In general, the keyword *this* is used to differentiate the instance variables from local variables if they have same names, within a constructor or a method.

EXAMPLE

```
public class Student {

        public String name;

        public Student(String name) {
                this.name = name;
        }

        public void printEmp() {
                System.out.println("name  : " + this.name);
        }

        public static void main(String args[]) {
                Student s1 = new Student("Ram");
                s1.printEmp();
        }
}
```

OUTPUT

name : Ram

THANK YOU!

Any questions?