# CSN-103: Fundamentals of Object Oriented Programming

**Instructor: Dr. Rahul Thakur**

Assistant Professor, Computer Science and Engineering, IIT Roorkee

# Argument Passing

- Two ways:
  - Call-by-value: **Value** of argument passed to the parameter of subroutine

    Changes made to the parameter don't affect the argument
  - Call-by-reference: Reference to an argument is passed to the parameter of subroutine

    Changes made to the parameter will affect the argument
- In Java,
  - Primitive types: Call-by-value
  - Objects: Call-by-reference

# Example: Call-by-value

```
class Test {
      void meth(int i, int j) {
      i *= 2;
      j /= 2;
      }
}

class CallByValue {
      public static void main(String args[]) {
       Test ob = new Test();
       int a = 15, b = 20;
       System.out.println("a and b before call: " + a + " " + b);
       ob.meth(a, b);
       System.out.println("a and b after call: " + a + " " + b);
      }
}
```

**OUTPUT:**

a and b before call: 15 20

a and b after call: 15 20

# Example: Call-by-reference

```
class Test {
      int a, b;

      Test(int i, int j) {
       a = i;
       b = j;
       }

      void meth(Test o) {
      o.a *= 2;
      o.b /= 2;
      }
}
class CallByRef {
      public static void main(String args[]) {
        Test ob = new Test(15, 20);
        System.out.println("ob.a and ob.b before call: " + ob.a + " " + ob.b);
        ob.meth(ob);
        System.out.println("ob.a and ob.b after call: " + ob.a + " " + ob.b);
      }
}
```

**OUTPUT:**

ob.a and ob.b before call: 15 20

ob.a and ob.b after call: 30 10

# Returning Objects

- Methods can return any primitive type and **class type** you create
- Note:
  - If an object created **anywhere** inside the program **will continue to exist** as long as there is a reference to it **somewhere** in the program
  - Create a temporary object inside a function and return its reference to use it in future

# Access Control

- Encapsulation
  - Links **data** with **code** that manipulates it
  - Access control: Control what parts of a program can be access and by whom
  - Example: Allowing access to instance variables using predefined functions
  - “Black Box”: Can be used but inner working can’t be tempered

# Access Control

- Control access of a **member** by the **Access Specifier**
  - Public
  - Private
  - Protected
  - *Package-Private* (no explicit modifier): Default
- Access control can also be done at the **class** level
  - Public
  - *Package-Private* (no explicit modifier): Default

# Access Control for Class Members

- Member access specifier
  - **Public**: Member can be accessed by any other code
    - Revisiting the main() method

public Method
Called by a code (Java Runtime System) outside the program

```
class Example{
        public static void main(String args[]){
                System.out.println("This is a  simple Java program");
        }
}
```

  - **Private**: Member can be accessed by other members of **its class**
  - **Protected**: Member can only be accessed within its own package

    + *by a subclass of its class in another package* ← *(Inheritance)*

# Access Control

| Access Levels | | | | |
|---|---|---|---|---|
| Modifier | Class | Package | Subclass | World |
| public | Y | Y | Y | Y |
| protected | Y | Y | Y | N |
| *no modifier* | Y | Y | N | N |
| private | Y | N | N | N |

# static Keyword

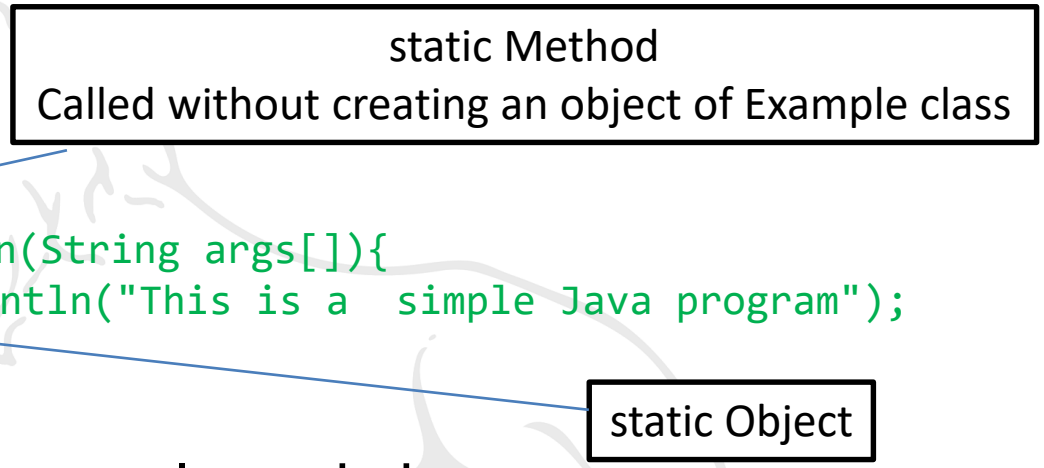
- When objects are created from the same class
  - Each have their **own distinct copies** of *instance variables*
- What if you want to have a variable that is **common** to all objects
- **static** variable
  - Preceding variable declaration with the keyword **static**
- **static** variables are associated with the class, rather than with any object

# static Keyword

- Every instance of the class shares the static variable(s)
  - Just one fixed location in memory
  - Any object can change the value of a class variable
- **static** variables can also be manipulated without creating an instance of the class

# static Keyword

- Methods can also be declared as **static**
- public **static** void **main()**
  - **main()** can be called without creating an object




- **static** methods have several restrictions
  - They can only access static data
  - They can call only other static methods
  - They can't refer to **this** or **super**(Inheritance)

# static Keyword

- If you wish to initialize **static** variables:
  - Declare a **static** block
  - **static** block executed only once when class is first loaded
- **static** variables are, essentially, global variables
  - Common to all, and used by all