

# CSN-103: Fundamentals of Object Oriented Programming

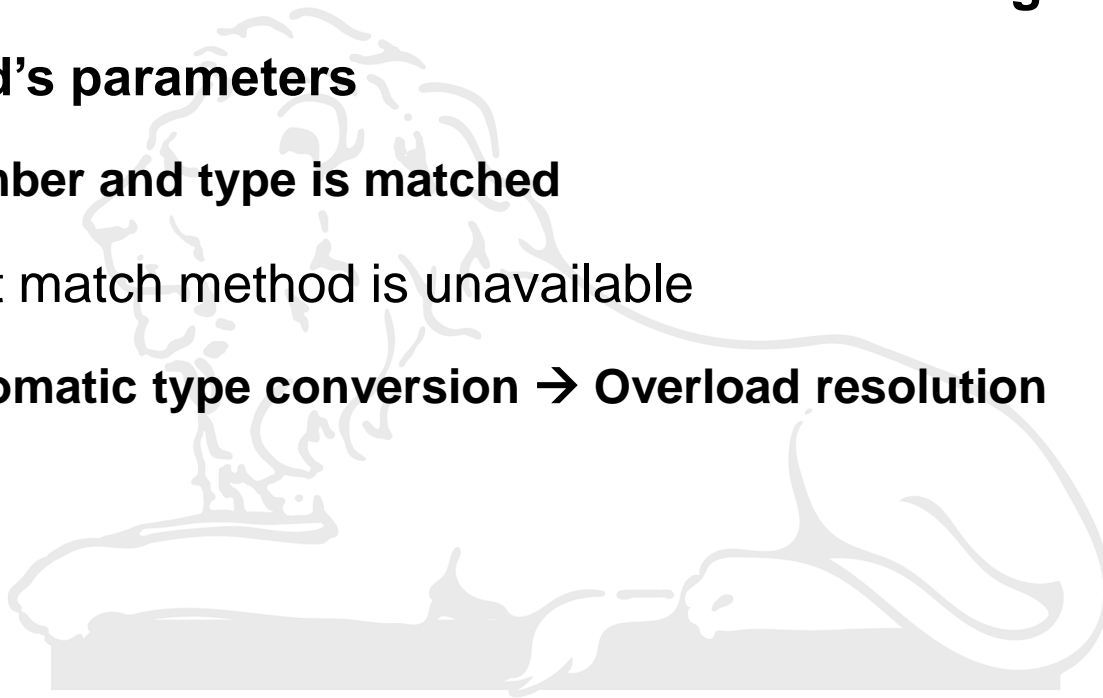


# Overloading Methods

- When an overloaded method is invoked
  - Java uses the type and/or number of arguments to determine which overloaded method to call
- Overloaded methods may have different return type
  - The return type alone is **insufficient** to distinguish two overloaded methods
  - Return type **doesn't play** a role in overload resolution

# Overloading Methods

- When an **overloaded** method is called:
  - Java **first** looks for an exact match between the **arguments** and **method's parameters**
    - Number and type is matched
  - if exact match method is unavailable
    - Automatic type conversion → Overload resolution



# Overloading Constructor

- Similar to methods, constructors can also be overloaded

```
class Box {  
    double width;  
    double height;  
    double depth;  
  
    Box(double w, double h, double d) {  
        width = w;  
        height = h;  
        depth = d;  
    }  
}
```

```
class Overload3 {  
    public static void main(String args[]) {  
        Box mybox1 = new Box();  
    }  
}
```

Overload3.java:15: error: constructor Box in class Box cannot  
be applied to given types;

Box mybox1 = new Box();

^

required: double,double,double

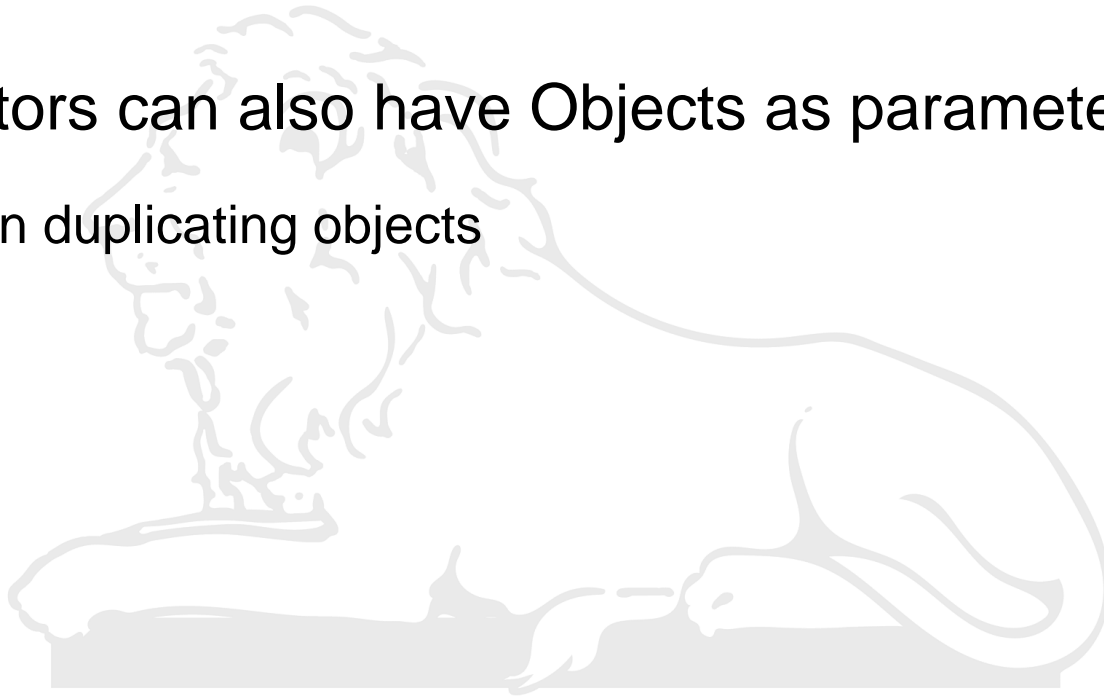
found: no arguments

reason: actual and formal argument lists differ in length

1 error

# Objects as Parameters

- Just like primitive types, objects can also be used as **parameters** to methods
- Constructors can also have Objects as parameters
  - Useful in duplicating objects

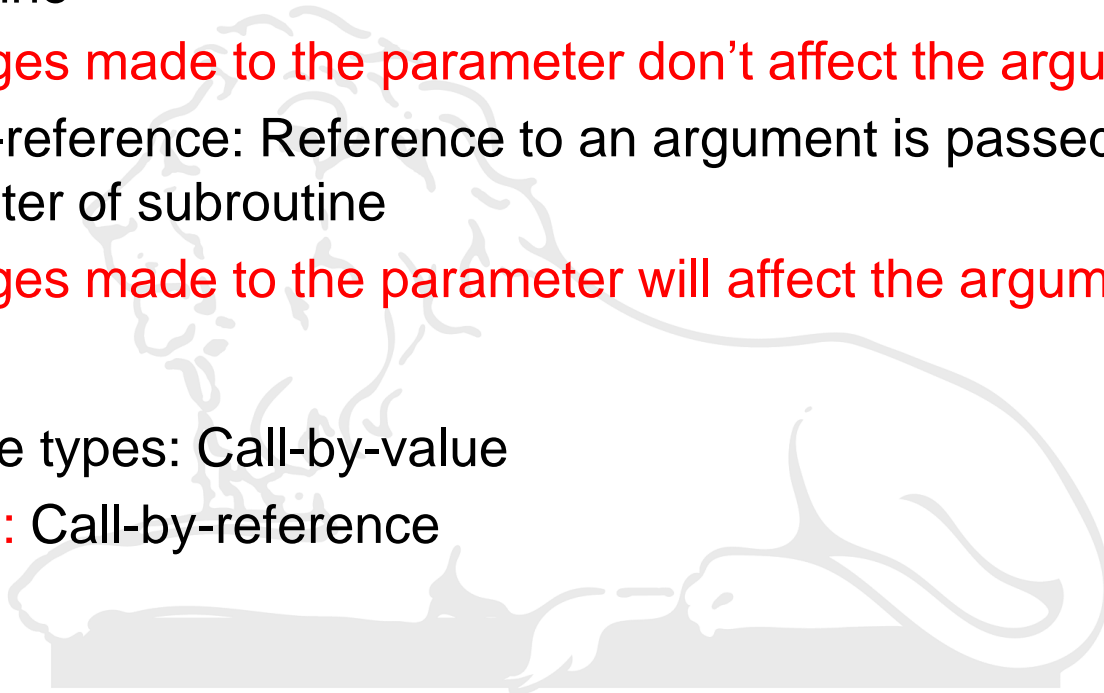


# Argument Passing

- 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

