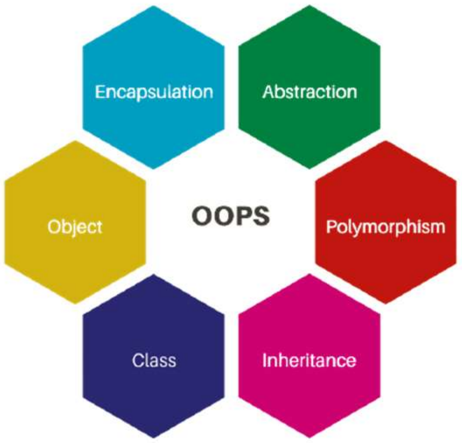
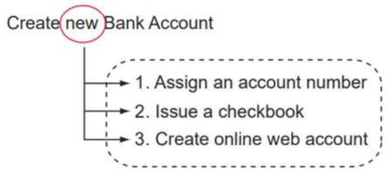


Java Constructors

Constructor

❖Constructors are special methods that create and return an object of the class in which they're defined.

❖Let's Understand Constructors –



Constructor

- Rules to remember
  - Constructors are special methods
  - Constructors have the same name as the class name
  - don't specify a return type

Syntax -

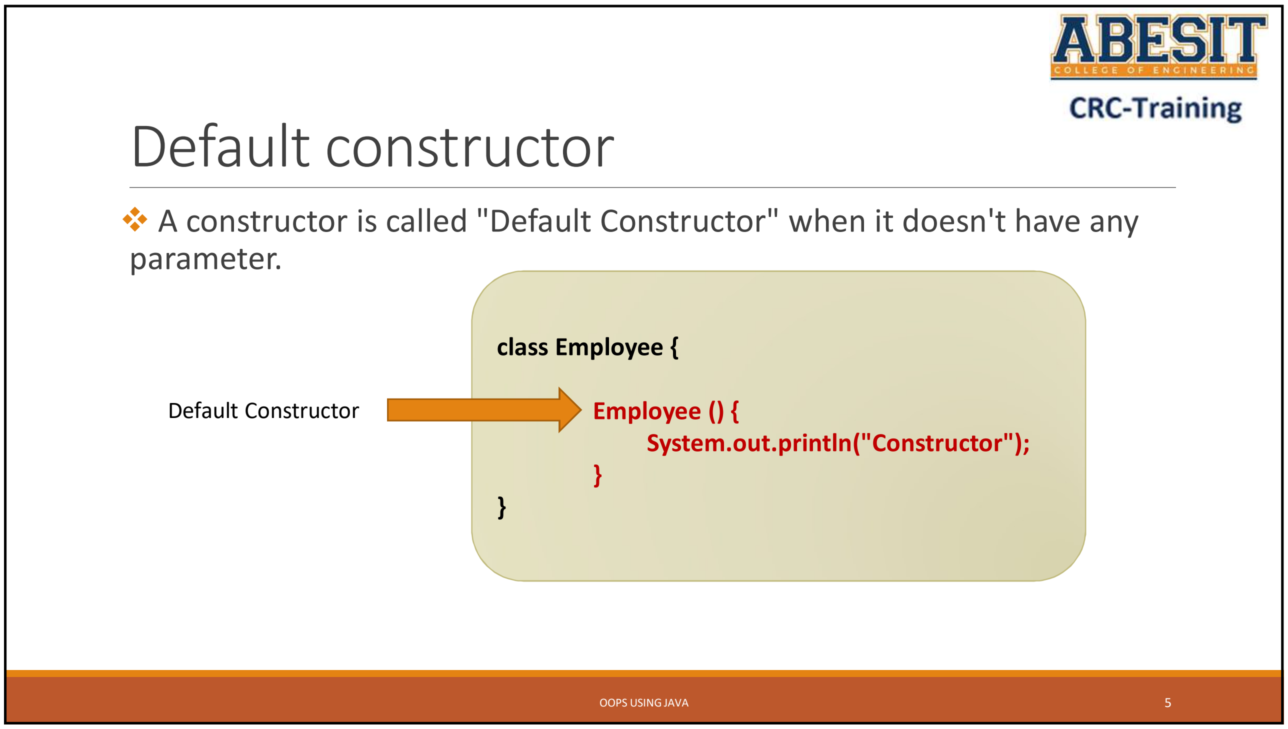
```
class_name(){
//Body of Constructors

}
```

```
class Employee {

        Employee () {
            System.out.println("Constructor");
        }
}
```

Types of Java constructors

There are two types of constructors in Java:

- Default constructor (no-arg constructor)
- Parameterized constructor



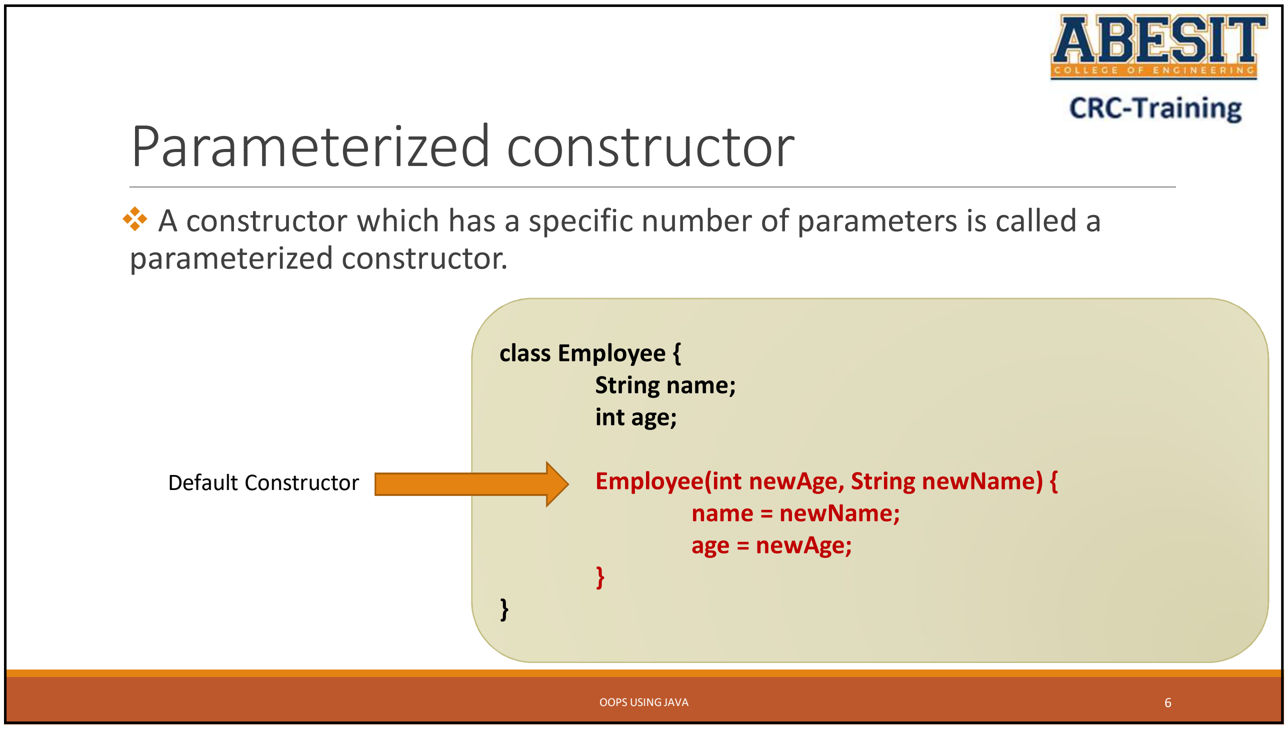
ABESIT COLLEGE OF ENGINEERING
CRC-Training

Default constructor

- A constructor is called "Default Constructor" when it doesn't have any parameter.

Default Constructor

```
class Employee {

          Employee () {
                System.out.println("Constructor");
          }
}
```



ABESIT COLLEGE OF ENGINEERING
CRC-Training

Parameterized constructor

- A constructor which has a specific number of parameters is called a parameterized constructor.

Default Constructor

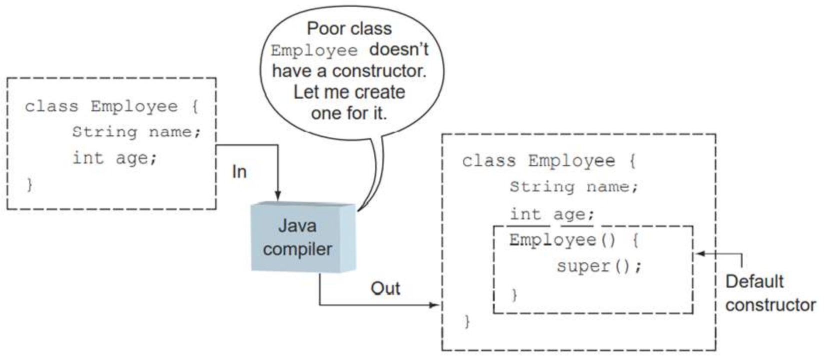
```
class Employee {
          String name;
          int age;

          Employee(int newAge, String newName) {
                    name = newName;
                    age = newAge;
          }
}
```

ABESIT COLLEGE OF ENGINEERING

CRC-Training

Constructor

What happens if you don't define any constructor in a class?



ABESIT COLLEGE OF ENGINEERING

CRC-Training

Constructor Overloading

- Constructor overloading in Java is a technique of having more than one constructor with different parameter lists.

```
class Employee {
        String name;
        int age;
        Employee() {
                name = "John";
                age = 25;
        }
        Employee(String newName) {
                name = newName;
                age = 25;
        }
        Employee(int newAge, String newName) {
                name = newName;
                age = newAge;
        }
```

No-argument constructor

Constructor with one String argument

Constructor with two arguments—int and String

CRC-Training

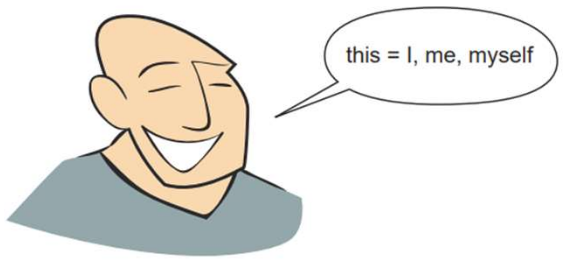
What happens if you put “return-type” before any constructor in a class?

```
class Employee {

        void Employee () {
            System.out.println("Constructor");
        }
}
```

CRC-Training

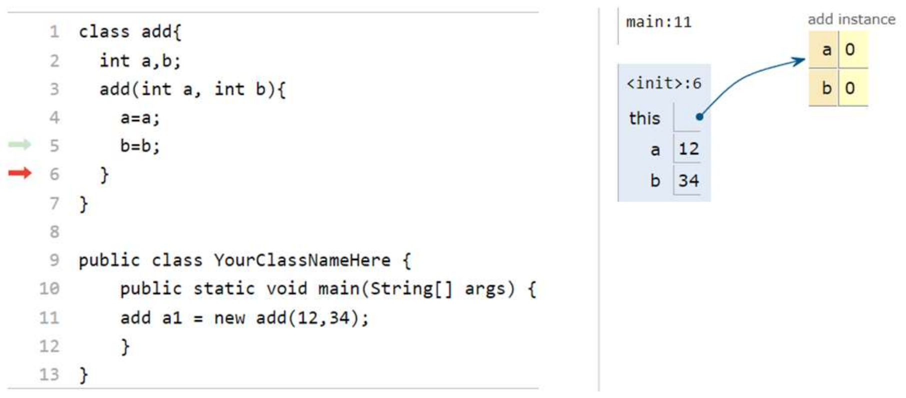
this keyword

- This is a **reference variable** that refers to the current object
- Any object can use **this reference** to refer to its own instance.



CRC-Training

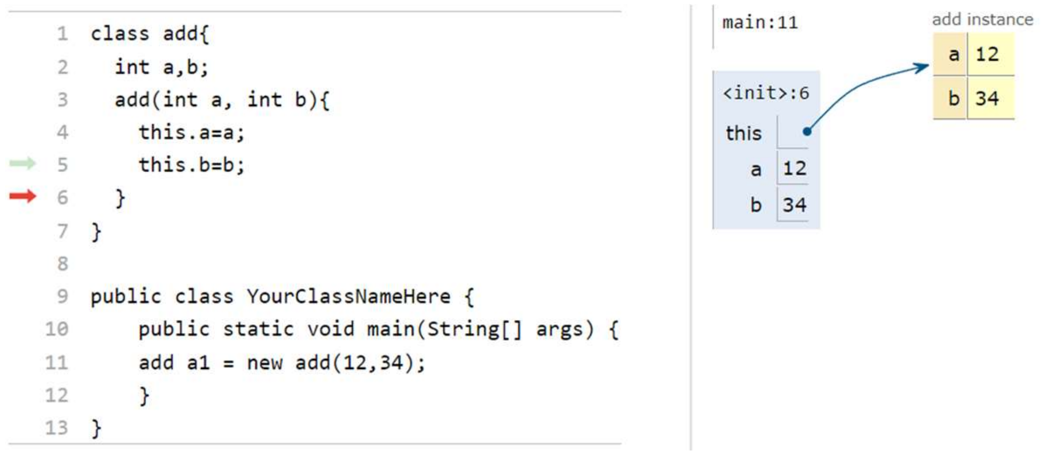
What if Instance Variable Name and local variable Name is/are the same in any Method?



```
class add{
  int a,b;
  add(int a, int b){
    a=a;
    b=b;
  }
}

public class YourClassNameHere {
    public static void main(String[] args) {
    add a1 = new add(12,34);
    }
}
```

CRC-Training

What if Instance Variable Name and local variable Name is/are the same in any Method?



```
class add{
  int a,b;
  add(int a, int b){
    this.a=a;
    this.b=b;
  }
}

public class YourClassNameHere {
    public static void main(String[] args) {
    add a1 = new add(12,34);
    }
}
```

Invoking An Overloaded Constructor From Another Constructor

- A constructor is defined using the name of its class,
- it’s a common mistake to try to invoke a constructor from another constructor using the class’s name

```
class Employee {
    String name;
    int age;
    Employee() {
        Employee(null, 0);
    }
    Employee(String newName, int newAge) {
        name = newName;
        age = newAge;
    }
}
```

Won’t compile—you can’t invoke a constructor within a class by using the class’s name.

Invoking An Overloaded Constructor From Another Constructor

```
class Employee {
    String name;
    int age;
    Employee() {
        this(null, 0);
    }
    Employee(String newName, int newAge) {
        name = newName;
        age = newAge;
    }
}
```

1 No-argument constructor

2 Invokes constructor that accepts two method arguments

3 Constructor that accepts two method arguments

Can we call two (or more) constructors within a constructor ?

- We can’t call two (or more) constructors within a constructor because the call to a constructor must be the first statement in a constructor.

```
class Employee {
    String name;
    int age;
    Employee() {
    }
    Employee(String newName, int newAge) {
        name = newName;
        age = newAge;
    }
    Employee(String newName, int newAge, boolean create) {
        this();
        this(newName, newAge);
        if (create)
            System.out.println(10);
    }
}
```

Won’t compile; can’t include calls to multiple constructors in a constructor