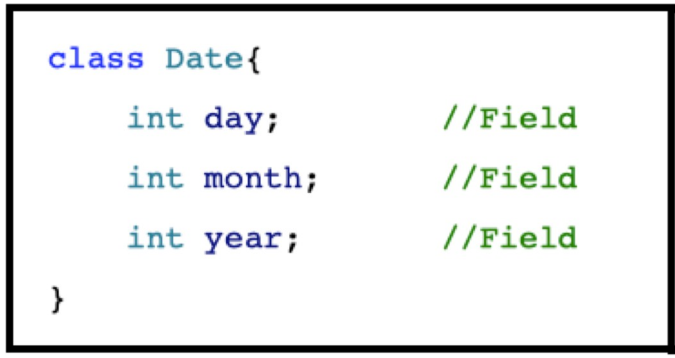
OOP using Java

Trainer: Mr. Rohan Paramane

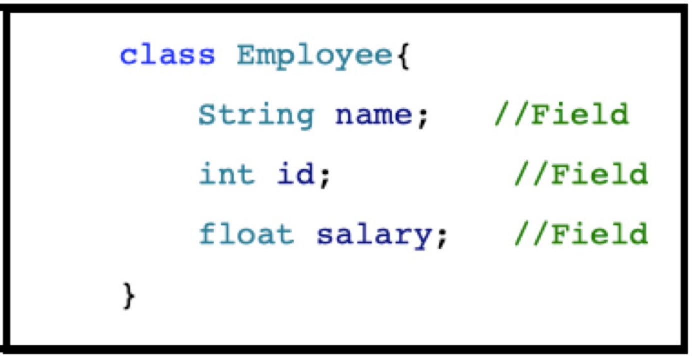
Class

- Consider following examples:
  1. day, month, year - related to – Date
  2. hour, minute, second - related to – Time
  3. red, green, blue - related to Color
  4. real, imag - related to – Complex
  5. xPosition, yPosition - related to Point
  6. number, type, balance - related to Account
  7. name, id, salary - related to Employee

- If we want to group related data elements together then we should use/define class in Java.



```
class Date{
    int day;            //Field
    int month;          //Field
    int year;           //Field
}
```



```
class Employee{
    String name;    //Field
    int id;          //Field
    float salary;    //Field
}
```

Class

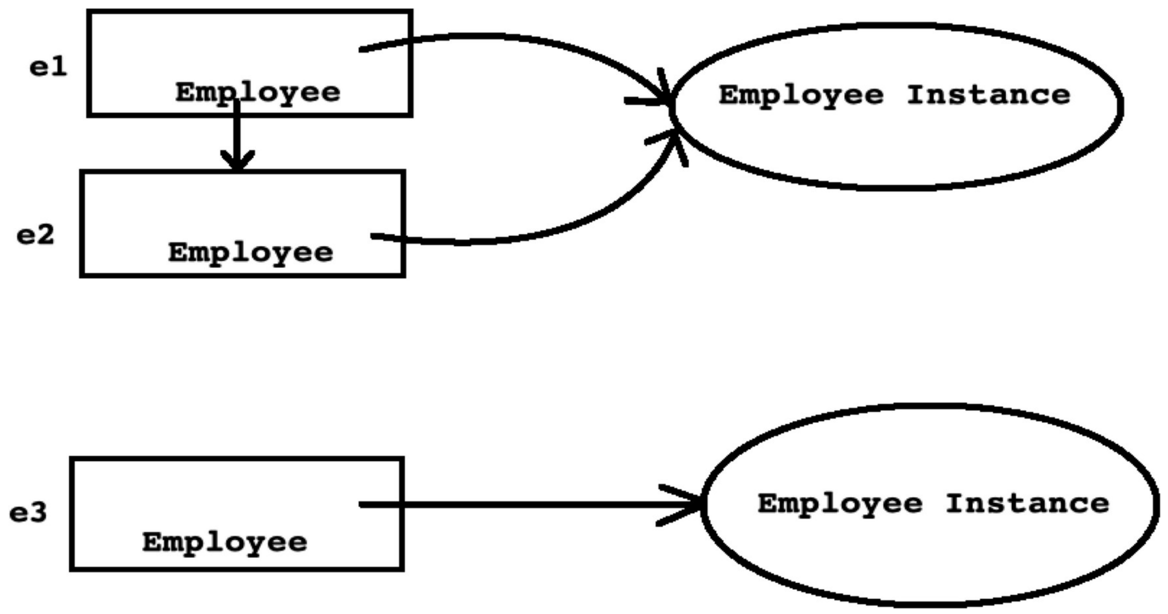
- Class is a non primitive/reference type in Java.
- A **class** is a user defined blueprint or prototype or template, from which objects are created.
- It is a logical entity
- It is a collection of fields(variables) and methods(Functions)
- Field
  - A variable declared inside class / class scope is called a field.
  - Field is also called as attribute or property.
- Method
  - A function implemented inside class/class scope is called as method.
  - Method is also called as operation, behavior or message.

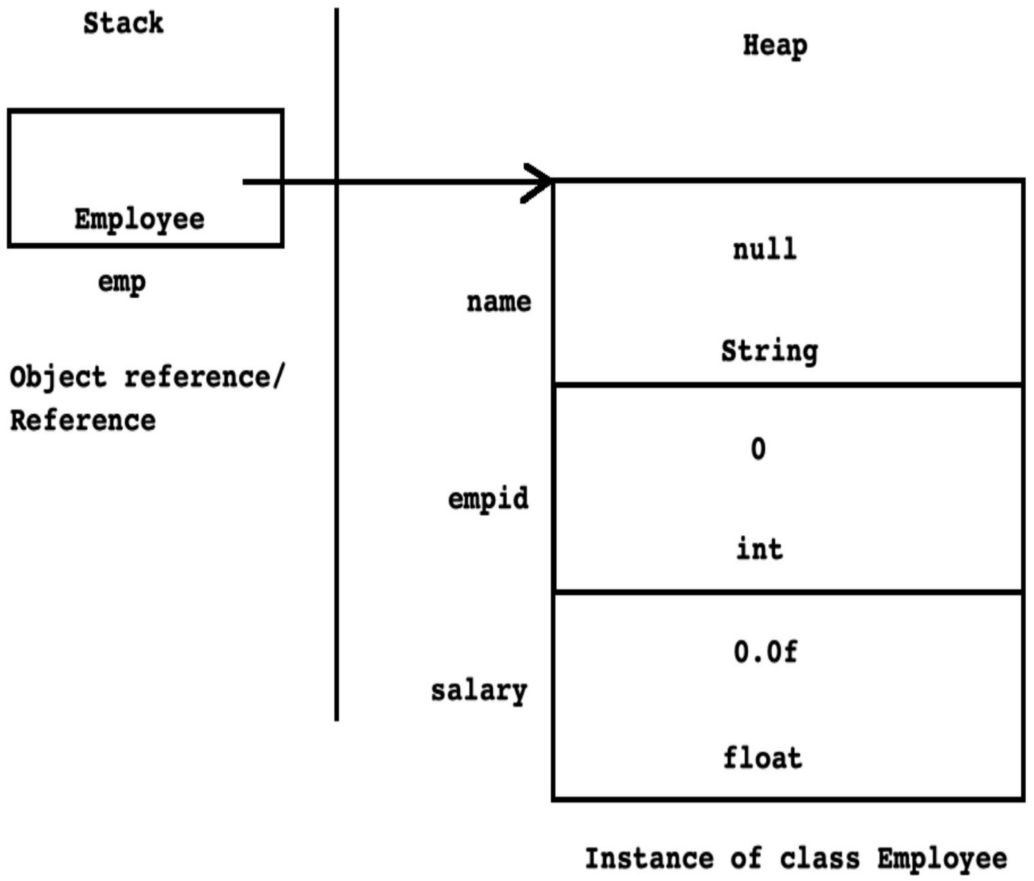
Object

- It is a basic unit of Object Oriented Programming.
- It is a physical entity
- In Java, Object is also called as instance.
- An object consists of :
  - State : It is represented by attributes of an object. (properties / instance variables(non static) / fields)
  - Behavior : It is represented by methods.
  - Identity : It gives a unique identity to an object and enables one object to interact with other objects.
    - eg : Emp id / Student PRN / Invoice No
- Creating an object
  - The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory.

Instantiation

- Process of creating instance/object from a class is called as instantiation.
- For eg –
  - Employee e1 = new Employee();
  - Employee e2 = e1;
  - Employee e3 = new Employee();

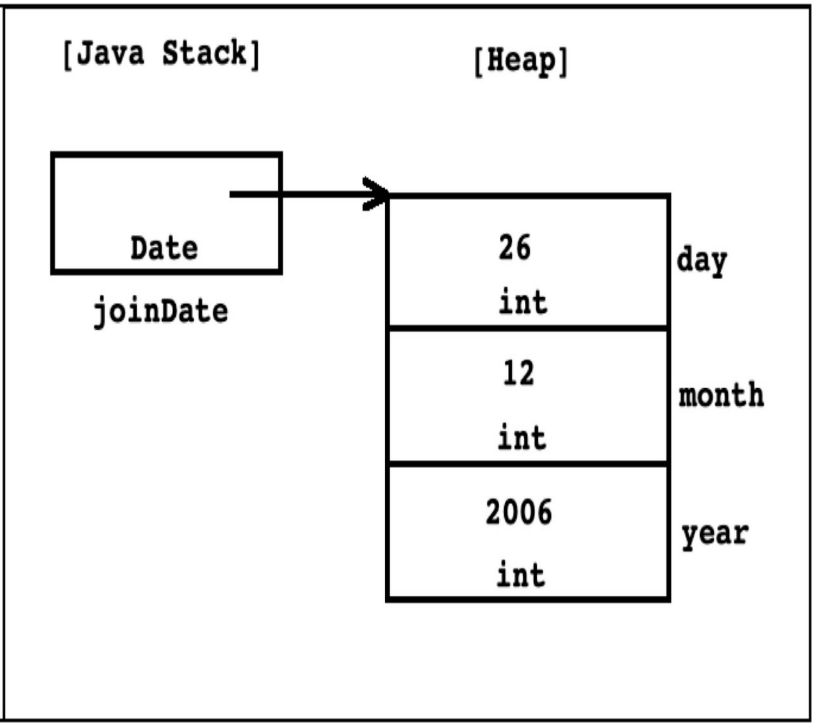




Reference

- Local reference variable get space on Java Stack.

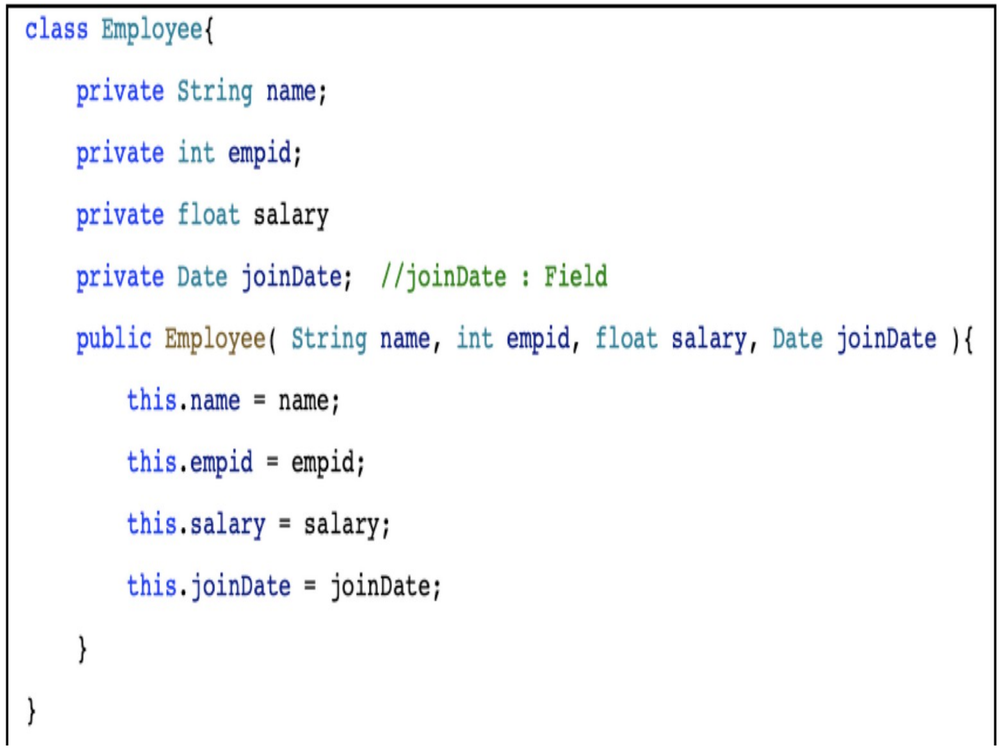
```
class Date{
    private int day;
    private int month;
    private int year;
}
class Program{
    public static void main(String[] args) {
        Date joinDate = new Date( 26,12,2006);
        //joinDate : Method Local Variable
    }
}
```



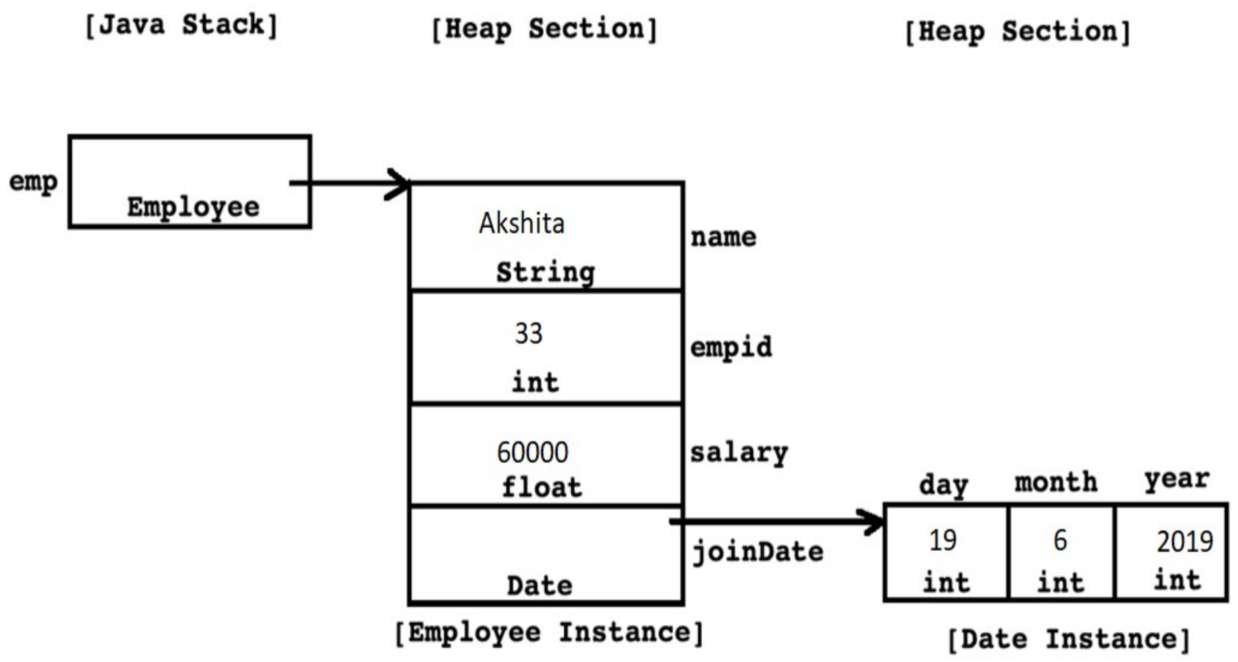
- In above code joinDate is method local reference variable hence it gets space on Java Stack.

Reference

- Class Scope reference variable get space on Java heap.



```
class Employee{

    private String name;

    private int empid;

    private float salary

    private Date joinDate;  //joinDate : Field

    public Employee( String name, int empid, float salary, Date joinDate ){

        this.name = name;

        this.empid = empid;

        this.salary = salary;

        this.joinDate = joinDate;

    }

}
```



- In above code, emp is method local reference variable hence it gets space on Java Stack. But joinDate is field of Employee class hence it will get space inside instance on Heap.

Scanner

- A class (java.util.Scanner) that represents text based parser.
- It can parse text data from any source
- Scanner is a final class declared in java.util package.
- Methods of Scanner class:
    1. public String nextLine()
    2. public int nextInt()
    3. public float nextFloat()
    4. public double nextDouble()
- How to user Scanner?

```
Scanner sc = new Scanner(System.in);
String name = sc.nextLine( );
int empid = sc.nextInt( );
float salary = sc.nextFloat( );
```

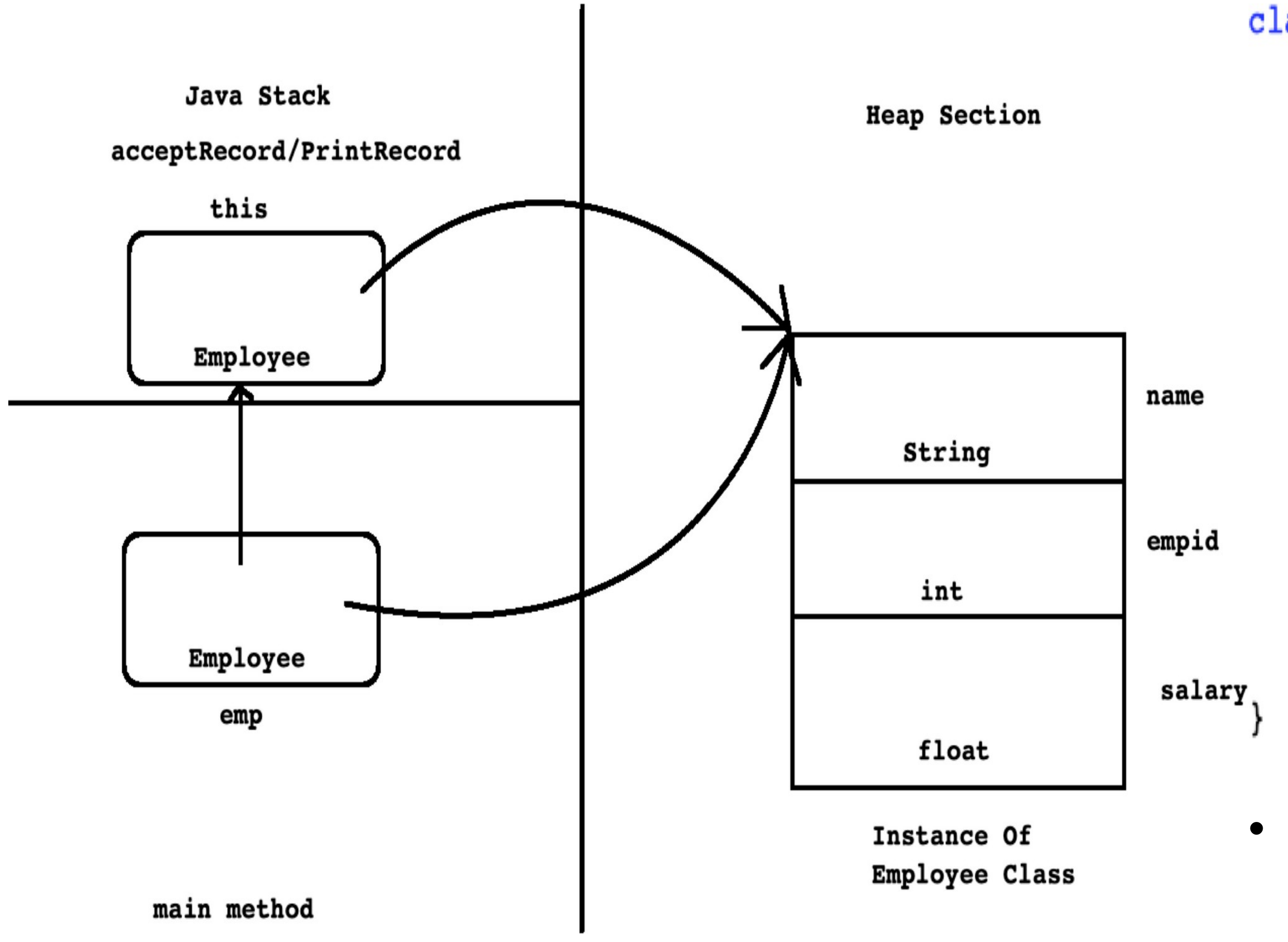
This reference

- If we call non static method on instance(actually object reference) then compiler implicitly pass, reference of current/calling instance as a argument to the method implicitly. To store reference of current/calling instance, compiler implicitly declare one reference as a parameter inside method. It is called this reference.
- Using this reference, non static fields and non static methods are communicating with each other. Hence this reference is considered as a link/connection between them.
- **“this” is implicit reference variable that is available in every non static method of class which is used to store reference of current/calling instance.**
- Inside method, to access members of same class, use this keyword is optional

- **Uses of this keyword :**
- 1. To unhide , instance variables from method local variables.(to resolve the conflict)
  - eg : this.name=name;

- 2. To invoke the constructor , from another overloaded constructor in the same class.(constructor chaining , to avoid duplication)

This reference



```
class Employee{
    private String name;
    private int empid;
    private float salary;
    public void initEmployee(String name, int empid, float salary ){
        this.name = name;
        this.empid = empid;
        this.salary = salary;
    }
}
```

- If name of local variable/parameter and name of field is same then preference is always given to the local variable.

Constructor

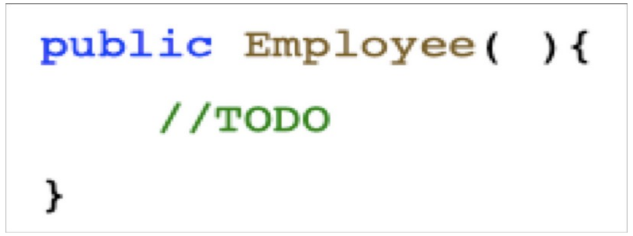
- If we want to initialize instance then we should define constructor inside class.
- Constructor look like method but it is not considered as method.
- It is special because:
  - Its name is same as class name.
  - It doesn't have any return type.
  - It is designed to be called implicitly
  - It is called once per instance.
- We can not call constructor on instance explicitly
  - Employee emp = new Employee();
  - emp.Employee( ); //Not Ok

- Types of constructor:
  1. Parameterless constructor
  2. Parameterized constructor
  3. Default constructor.

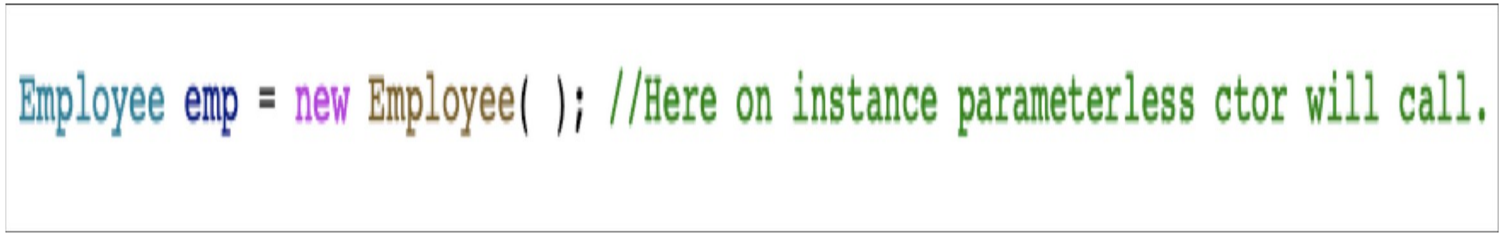
Default & Parameterless Constructor

- **Default Constructor**
- If we do not define any constructor inside class then compiler generate one constructor for the class by default. It is called default constructor.
- Compiler generated default constructor is parameterless.

- **Parameterless Constructor**
- If we define constructor without parameter then it is called as parameterless constructor.
- It is also called as zero argument / user defined default constructor.
- If we create instance without passing argument then parameterless constructor gets called.



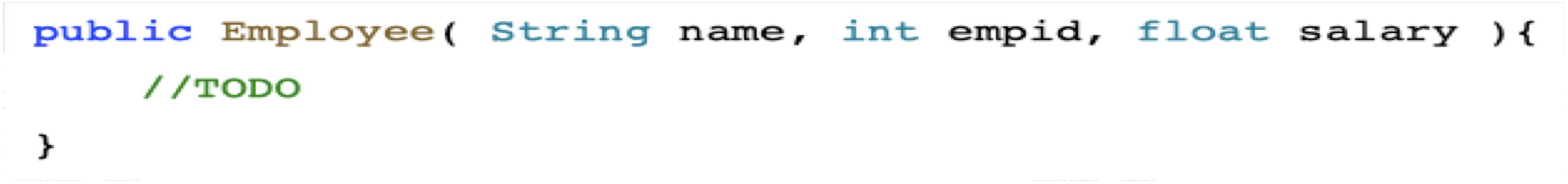
```
public Employee( ){
    //TODO
}
```



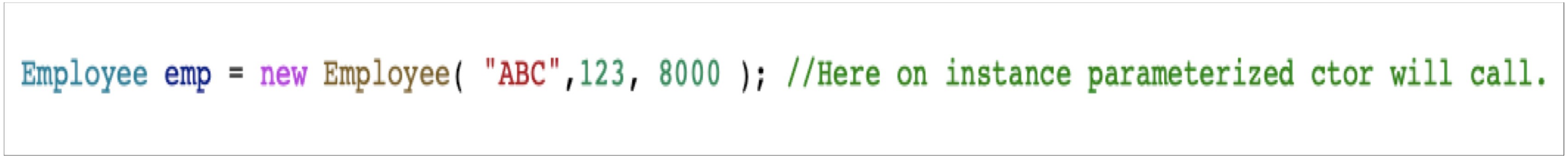
```
Employee emp = new Employee( ); //Here on instance parameterless ctor will call.
```

Parameterized Constructor

- If we define constructor with parameter then it is called as parameterized constructor.
- If we create instance by passing argument then parameterized constructor gets called.



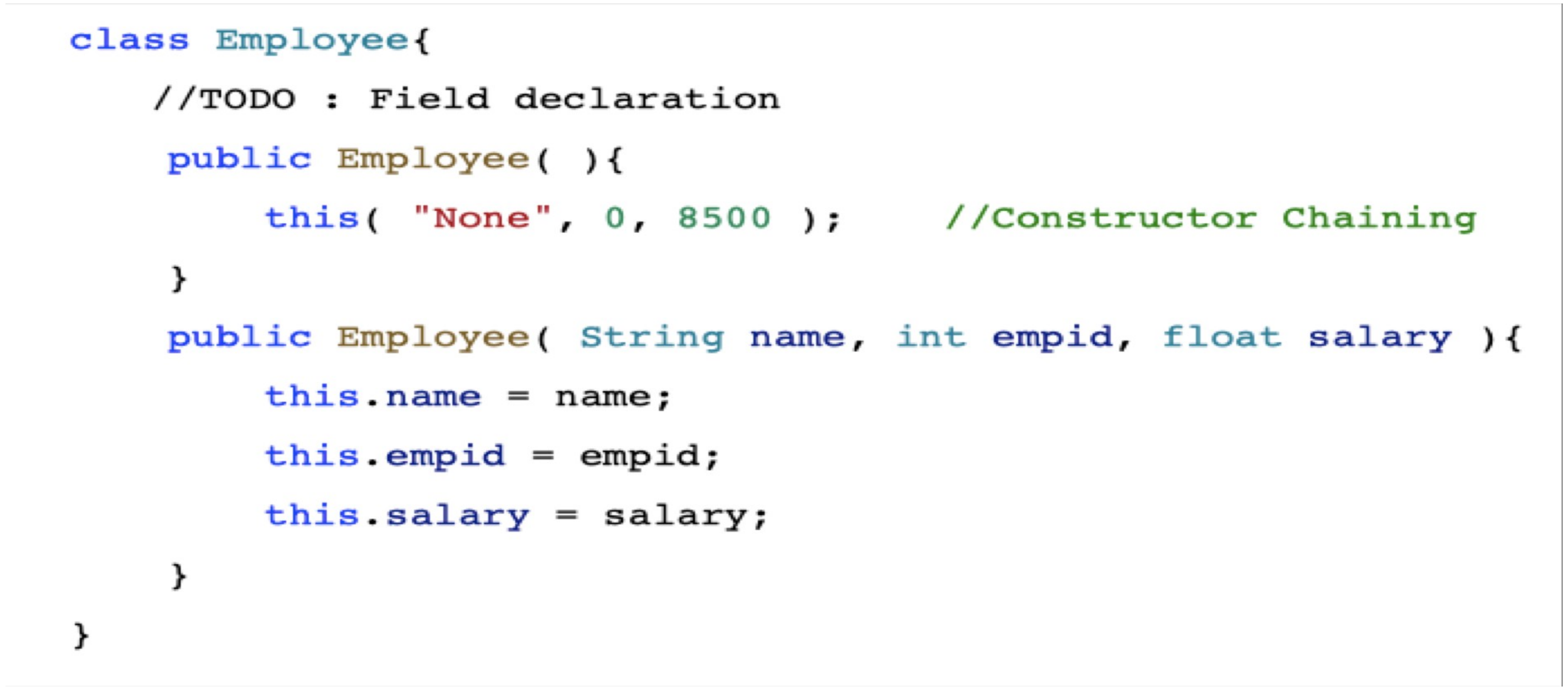
```
public Employee( String name, int empid, float salary ){
        //TODO
}
```



```
Employee emp = new Employee( "ABC",123, 8000 ); //Here on instance parameterized ctor will call.
```

Constructor Chaining

- We can call constructor from another constructor. It is called constructor chaining.
- For constructor chaining, we should use this statement.
- this statement must be first statement inside constructor body.
- Using constructor chaining, we can reduce developers effort.



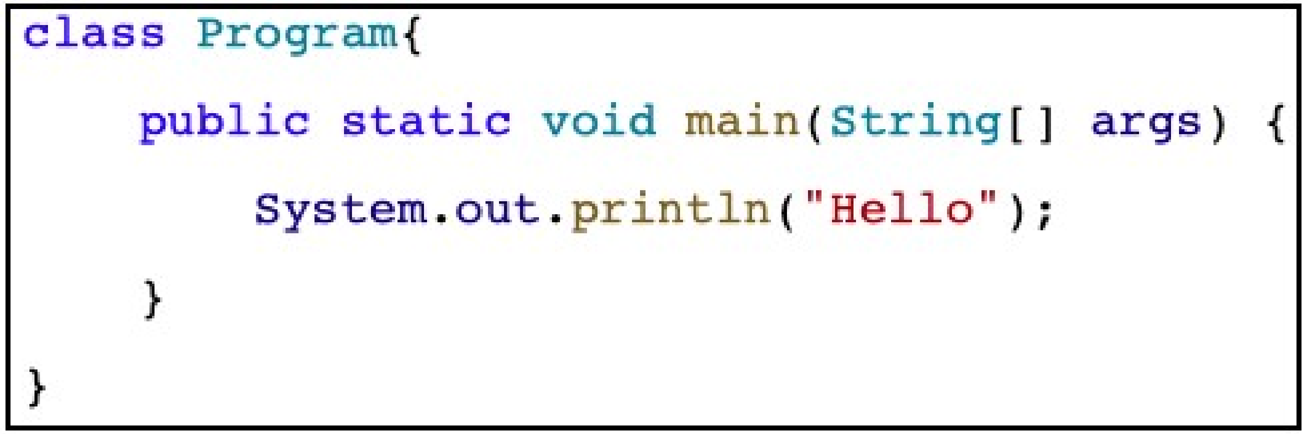
```
class Employee{
    //TODO : Field declaration
     public Employee( ){
         this( "None", 0, 8500 );    //Constructor Chaining
     }
     public Employee( String name, int empid, float salary ){
         this.name = name;
         this.empid = empid;
         this.salary = salary;
     }
}
```

Package

- Package is a Java language feature which helps developer to:
- To group functionally equivalent or related types together.
- To avoid naming clashing/collision/conflict/ambiguity in source code.
- To control the access to types.
- To make types easier to find(from the perspective of java docs).
- Consider following class:
    - java.lang.Object
    - Here java is main package, lang is sub package and Object is type name.
- package is a keyword in Java.
- To define type inside package, it is mandatory write package declaration statement inside .java file.
- Package declaration statement must be first statement inside .
- If we define any type inside package then it is called as packaged type otherwise it will be unpackaged type.

Un-named Package

- If we define any type without package then it is considered as member of unnamed/default package.
- Unnamed packages are provided by the Java SE platform principally for convenience when developing small or temporary applications or when just beginning development.
- An unnamed package cannot have sub packages.
- In following code, class Program is a part of unnamed package.



```java
class Program{
    public static void main(String[] args) {
        System.out.println("Hello");
    }
}
```

Naming convention for package

- For small programs and casual development, a package can be unnamed or have a simple name, but if code is to be widely distributed, unique package names should be chosen using qualified names.
- Generally Package names are written in all lower case to avoid conflict with the names of classes or interfaces.
- Companies use their reserved internet domain name to begin their package names. For example : com.example.mypackage
- Following examples will help you in deciding name of package:
  - java.lang.reflect.Proxy
  - oracle.jdbc.driver.OracleDriver
  - com.mysql.jdbc.cj.Driver
  - org.cdac.sunbeam.dac.utils.Date

Access Modifier

- If we want to control visibility of members of class then we should use access modifier.
- There are 4 access modifiers in Java:
  - private
  - package-level private / default
  - protected
  - public

- **Other Modifiers :**
  - abstract
  - final
  - interface
  - native
  - static
  - strict
  - synchronized
  - transient
  - volatile

Setters and Getters

- To access private members of the class outside the class public methods should be used.
- If a value of single private field needs to be changed then the public method used for it is called as setter.
- If a value of single private field needs accessed then the public method used for it to access is called as getter.
- The syntax to write setter and getter is as below.

```
public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}
```

Thank you!

Rohan Paramane

rohan.paramane@sunbeaminfo.com