OOP using Java

Trainer: Mr. Rohan Paramane

Agenda

- This reference
- Constructor
- Constructor Chaning
- Static Field
- Static method
- Package
- Access Modifiers
- Static import

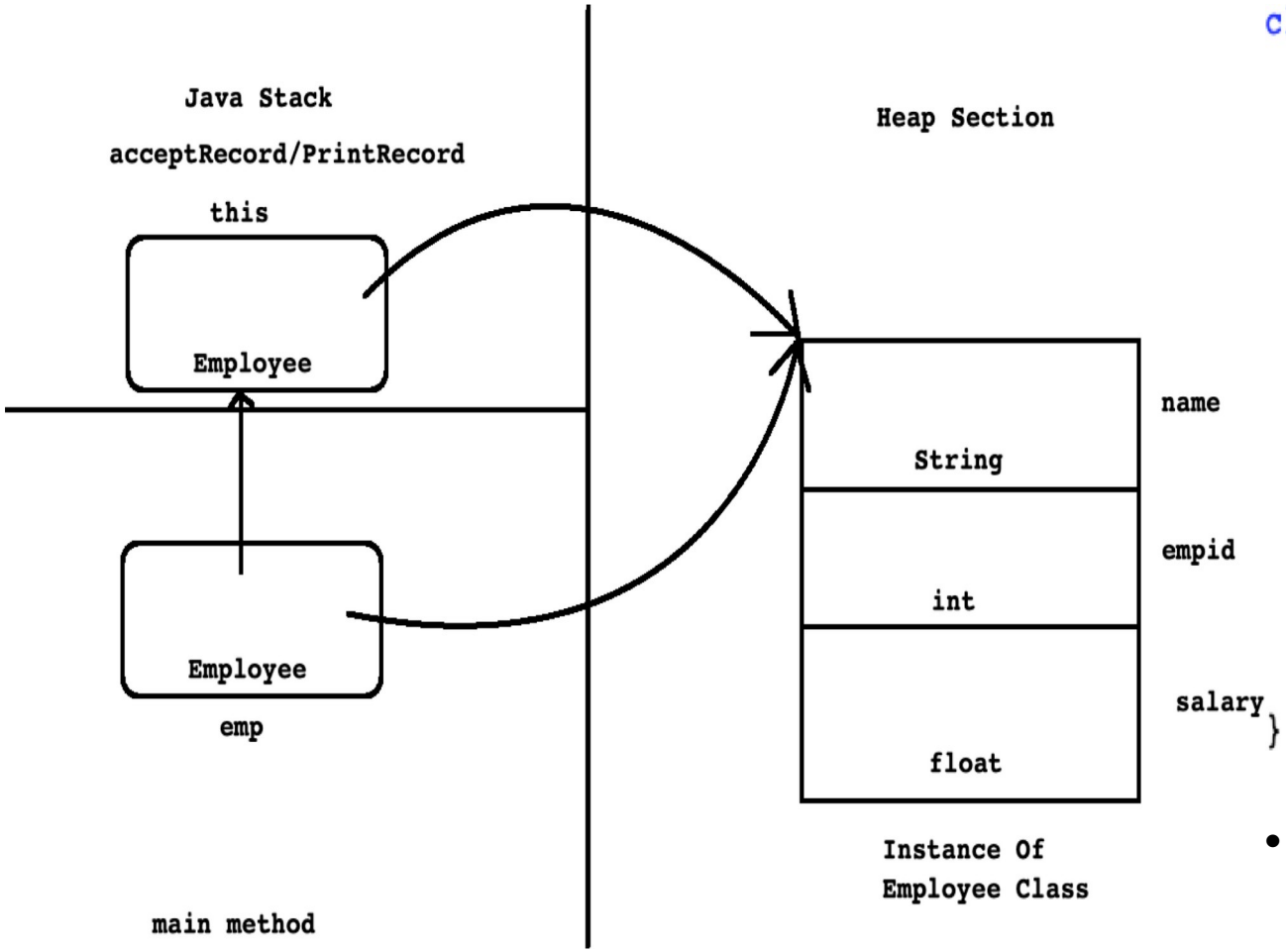
This reference

- 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;
    }
}
```

Static Field & its Initialization

- **Static Field**
  - Static field do not get space inside instance rather all the instances of same class share single copy of it.
  - Static Field is also called as class variable. It gets space once per class.
  - Static Field gets space once per class during class loading on method area .
  - Static Field can be accessed using object reference but it is designed to access using class name and dot operator.
- **Static Initializer block**
  - It is used to initialize the static fields
  - A static initialization block is a normal block of code enclosed in braces, { }, and preceded by the static keyword. Here is an example: static { // code to write }
  - A class can have any number of static initialization blocks, and they can appear anywhere in the class body.
  - The runtime system guarantees that static initialization blocks are called in the order that they appear in the source code.

Static Method

- To access non static members of the class, we should define non static method inside class.
- Non static method/instance method is designed to call on instance.
- To access static members of the class, we should define static method inside class.
- static method/class level method is designed to call on class name.
- static method do not get this reference:
- If we call, non static method on instance then method get this reference.
- Static method is designed to call on class name.
- Since static method is not designed to call on instance, it doesn't get this reference.

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.

```
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

Static Import

- If static members belonging to the different class then use of type name and dot operator is mandatory.
- There are situations where you need frequent access to static final fields (constants) and static methods from one or two classes.
- Prefixing the name of these classes over and over can result in cluttered code.
- The static import statement gives you a way to import the constants and static methods that you want to use so that you do not need to prefix the name of their class.

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

Thank you!

Rohan Paramane

rohan.paramane@sunbeaminfo.com