Object Oriented Programming with Java 8
PG-DAC

Rohan Paramane

Agenda

- Constructor
- This reference
- Package
- Scanner Class

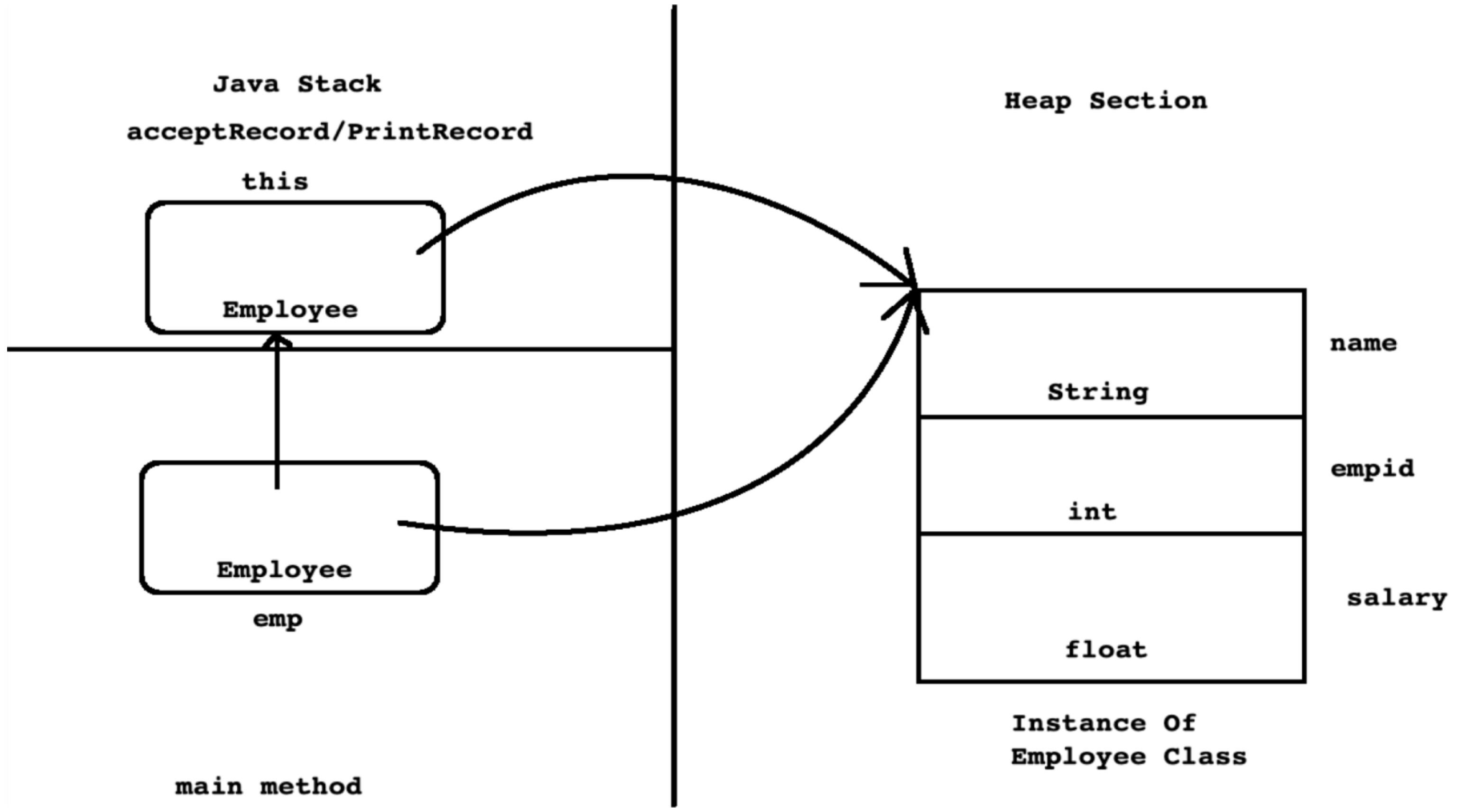
this current object reference

- 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.**
- Definition
  - **“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;

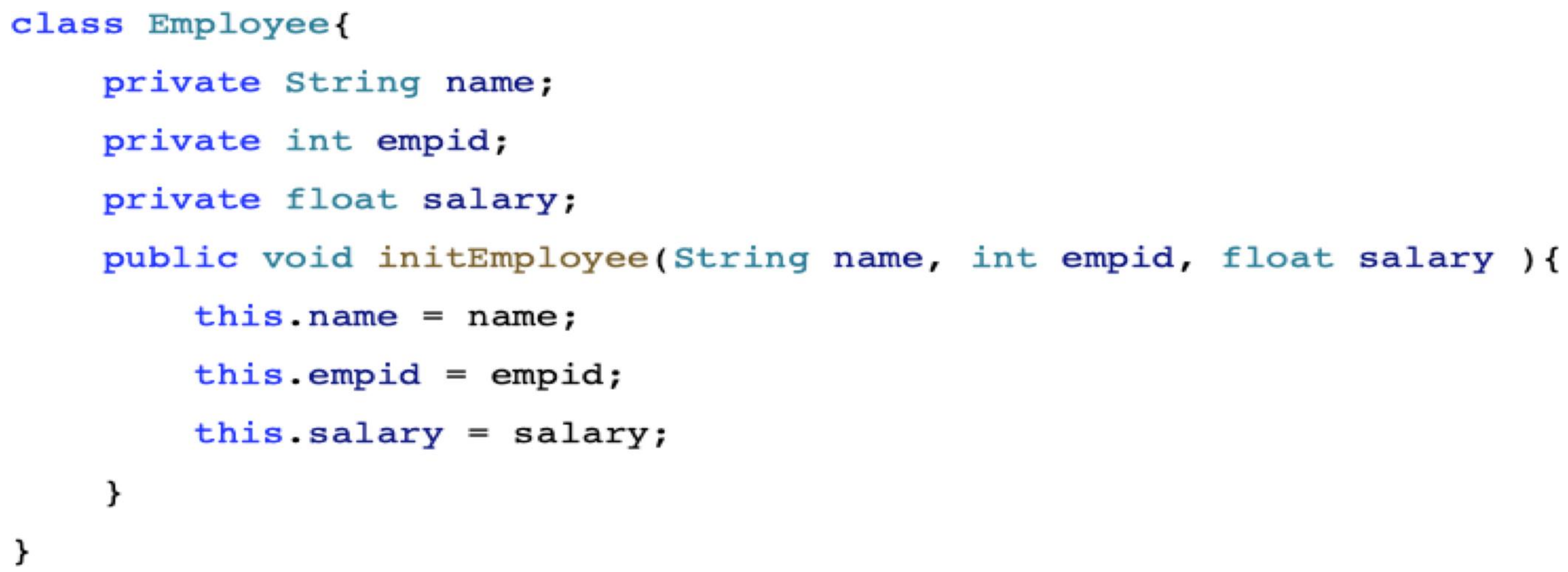
2. To invoke the constructor , from another overloaded constructor in the same class.(constructor chaining , to avoid duplication)

this reference



this reference

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



```
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;
    }
}
```

Constructor

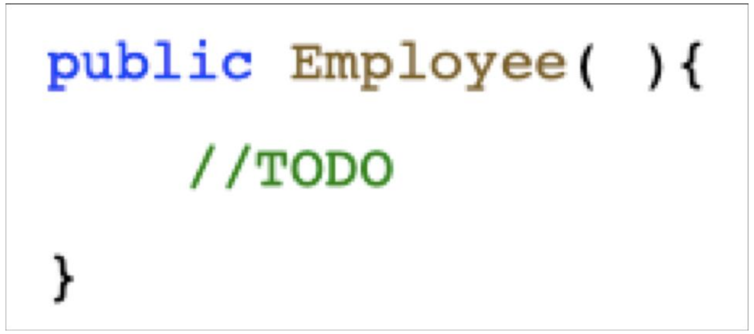
- If we want to initialize instance then we should define constructor inside class.
- Constructor look like method but it is not considered as method.
- It is special because:
  - Its name is same as class name.
  - It doesn't have any return type.
  - It is designed to be called implicitly
  - It is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

Employee emp = new Employee();
emp.Employee( ); //Not Ok

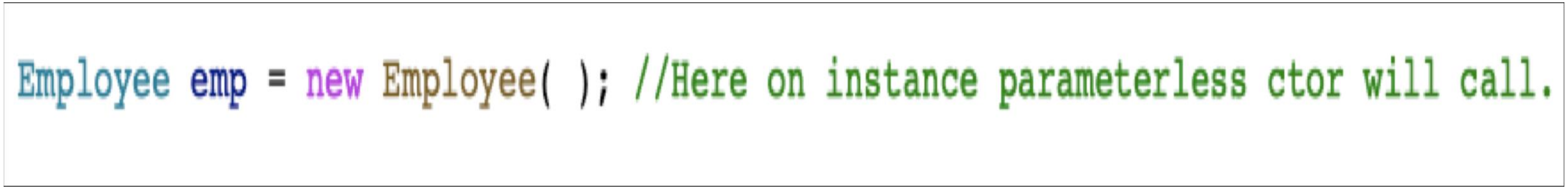
- **Types of constructor:**
  1. Parameterless constructor
  2. Parameterized constructor
  3. Default constructor .

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.



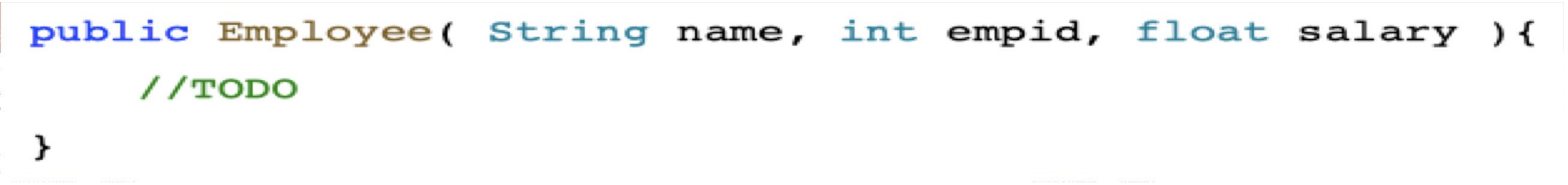
```
public Employee( ){
      //TODO
}
```



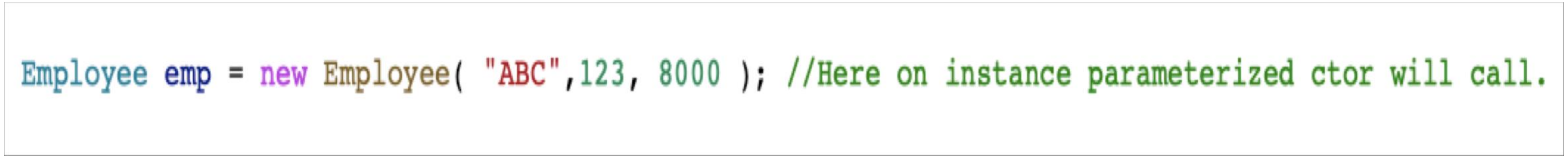
```
Employee emp = new Employee( ); //Here on instance parameterless ctor will call.
```

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.
```

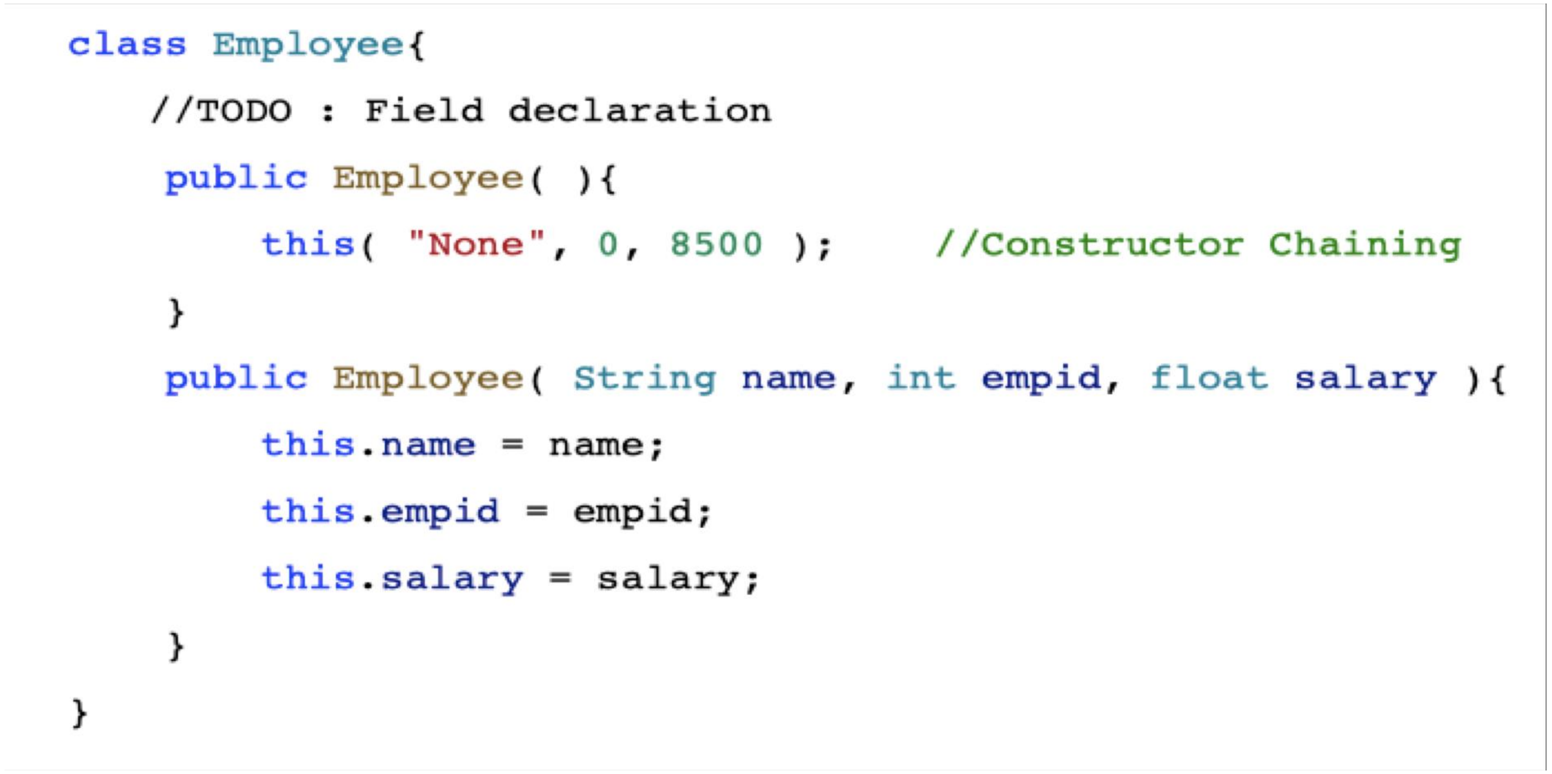
Default Constructor

- 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.
- Compiler never generate default parameterized constructor. In other words, if we want to create instance by passing arguments then we must define parameterized constructor inside class.

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;
    }
}
```

What is Scanner ?

- A class (java.util.Scanner) that represents text based parser(has inherent small ~ 1K buffer)
- It can parse text data from any source --Console input,Text file , socket, string

e.g.

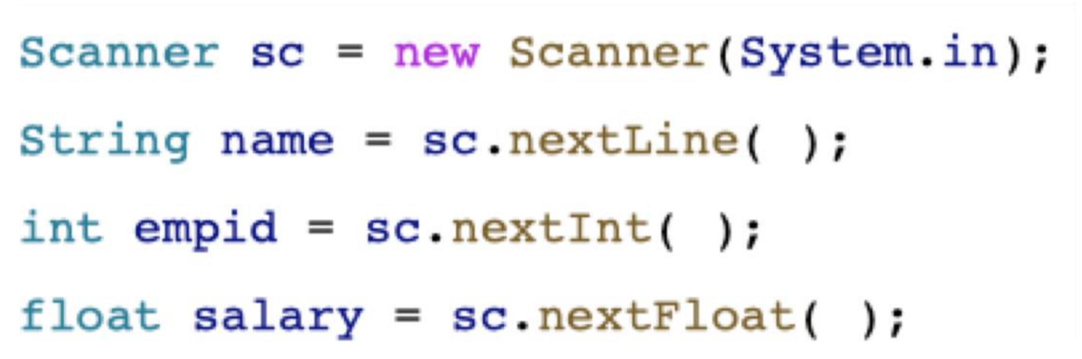
```
Scanner input = new Scanner(System.in);
System.out.print("Enter your name: ");
String name = input.next ();
System.out.println(“Your name is " + name);
input.close();
```

User Input Using Scanner class.

- 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( );
```

Package

- Package is a Java language feature which helps developer to:
  1. To group functionally equivalent or related types together.
  2. To avoid naming clashing/collision/conflict/ambiguity in source code.
  3. To control the access to types.
  4. To make types easier to find(from the perspective of java docs).

- Consider following class:
  - ➢ java.lang.Object
    - o Here java is main package, lang is sub package and Object is type name.

Package

- Not necessarily but as shown below, package can contain some or types.
    1. Sub package
    2. Interface
    3. Class
    4. Enum
    5. Exception
    6. Error
    7. Annotation Type

Package Creation

- 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.
- Any type can be member of single package only.

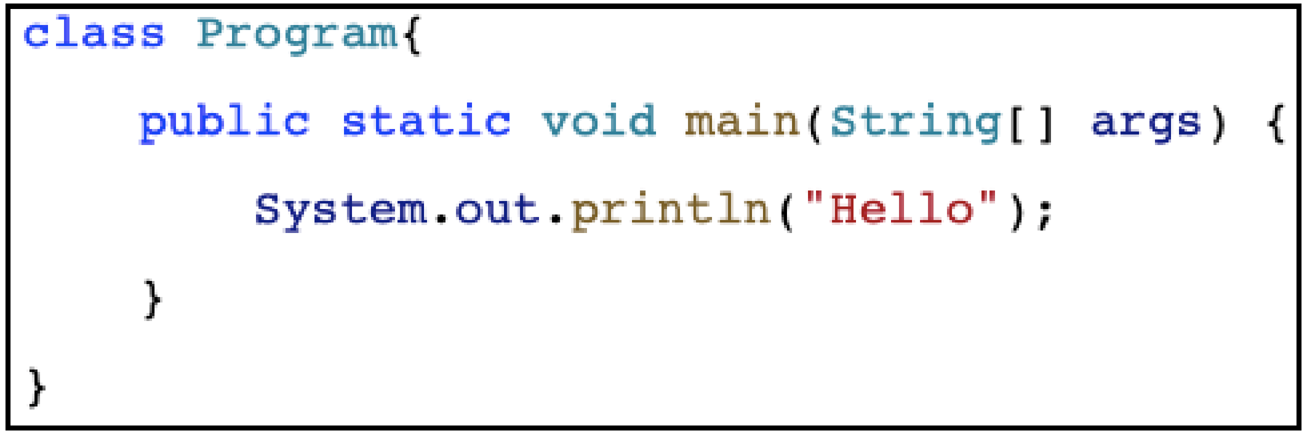
```java
package p1;   //OK
class Program{
    //TODO
}
```

```java
package p1, p2; //NOT OK
class Program{
    //TODO
}
```

```java
package p1; //OK
package p2; //NOT OK
class Program{
    //TODO
}
package p3; //Not OK
```

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 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:
    1. java.lang.reflect.Proxy
    2. oracle.jdbc.driver.OracleDriver
    3. com.mysql.jdbc.cj.Driver
    4. org.cdac.sunbeam.dac.utils.Date

How to use package members in different package?

- If we want to use types declared inside package anywhere outside the package then
    1. Either we should use fully qualified type name or
    2. import statement.
- If we are going to use any type infrequently then we should use fully qualified name.
- Let us see how to use type using package name.

```
class Program{
    public static void main(String[] args) {
        java.util.Scanner sc = new java.util.Scanner( System.in );
    }
}
```

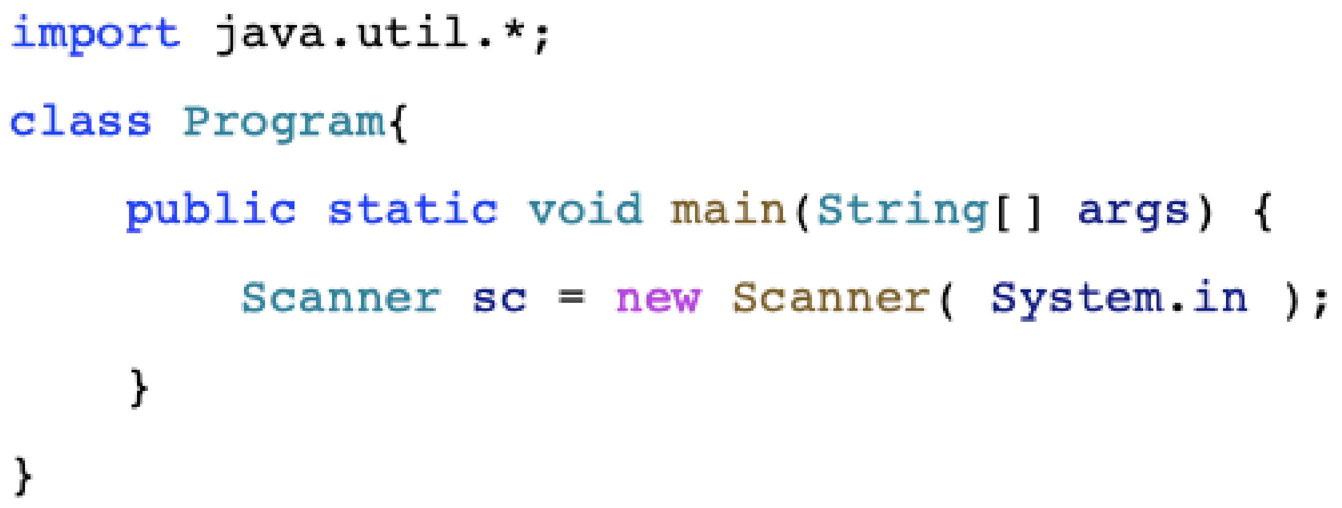
How to use package members in different package?

- If we are going to use any type frequently then we should use import statement.
- Let us see how to import Scanner.

```
import java.util.Scanner;
class Program{
    public static void main(String[] args) {
        Scanner sc = new Scanner( System.in );
    }
}
```

How to use package members in different package?

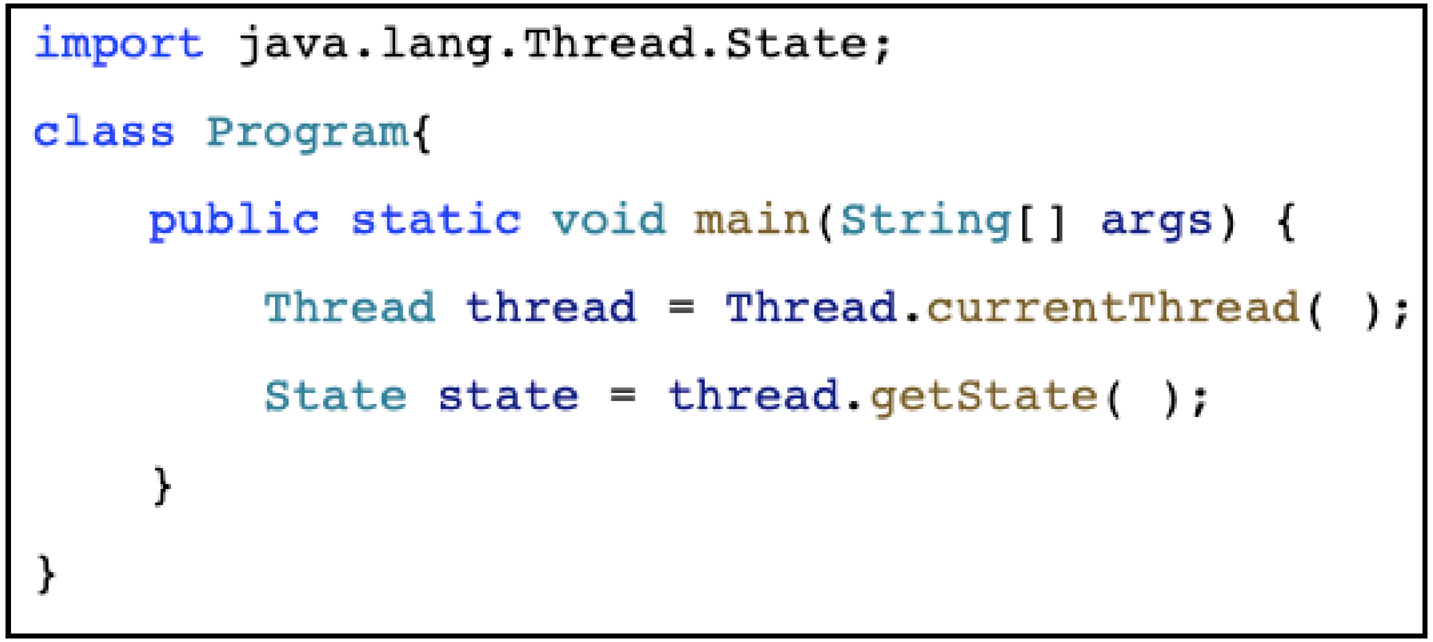
- There can be be any number of import statements after package declaration statement
- With the help of(*) we can import entire package.



```
import java.util.*;
class Program{
    public static void main(String[] args) {
        Scanner sc = new Scanner( System.in );
    }
}
```

How to use package members in different package?

- Another, less common form of import allows us to import the public nested classes of an enclosing class. Consider following code.



```
import java.lang.Thread.State;
class Program{
    public static void main(String[] args) {
        Thread thread = Thread.currentThread( );
        State state = thread.getState( );
    }
}
```

- No of core java. This
  pa ence to use type
  de ptional.

Thank you.

Rohan.paramane@sunbeaminfo.com