Access Modifier & Package

Java Modifiers-

Role of modifier – we are providing information to the JVM about our class behaviour

•public

•private

•protected

•<default>

• final

• static

•abstract

- Class Level modifiers
- Member level modifiers

1. Class level modifiers:-

(public , default , final, abstract – applicable for outer class)

(public, private ,protected, default, final, abstract - applicable for inner class)

- **Final modifier** – this modifier is applicable for classes , methods and variables.

- final method
- final class
- final variable

- **abstract modifier** – this modifier is applicable for classes, method but not for variables

- abstract method
- Abstract class

- **static modifier** – this modifier is applicable for variables , methods but not for classes.

- static method
- static variable

final keyword in Java

- Variable: stops value change

  final variable once assigned a value can never be changed

- Methods: stops overriding

- Class: stops inheritance

  every method present inside final class is always final by default but every variable present inside final class need not to be final

Example-

final variable

- Final variable is initialize at the time of declaration .
- Inside constructor , we can initialize
- Final variable can not be initialize inside any other method

Example –

- Instance variable name and local variable name can be ________.
- Static variable name and local variable name can be ________.
- Instance variable name and Static variable name can be same or not ?

- final instance variable – must be initialize
- final static variable – we should perform initialization
- final local variable - must be initialize

The only applicable modifier for local variable is final.

```
public class Demo{
      public static void main(String[] args) {
         public int a =20;
         private int b =4;
         protected int c=67;
         static int  d =89;
      }

}
```

Error

Abstract method

- Abstract method - Declared without a body in an abstract class.
- Abstract method declaration should be ends with ; **(semicolon)**
- Syntax:

```
abstract void methodName();
```

- Must be **overridden** in a subclass. A subclass must **override all abstract methods**.
- **NOTE –** if a class contain at least one abstract method then that class must be declared as Abstract class
- Example -

Abstract class

- if we are not allowed to create object , such type of class we have to declared with abstract
- Syntax:

```
abstract class Demo{
        public static void main(String[] args) {
             Demo d = new Demo();
        }
}
```

❌

Error i.e
Compile time
error

Abstract method & Abstract class

- if a class contain at least one abstract method - that class should be abstract class
- Abstract class can contain zero number of abstract method also.
- If we extend abstract class then for each and every abstract method of that class , we should provide implementation

Example-

```
abstract class Demo{
       abstract void m1();
       abstract void m2();
}
class  Child extends Demo {
      public void m1(){
      }
}
```

Question?

- final abstract Method – Not Allowed
- final abstract Class – Not Allowed
- abstract class can contain final method ? y
- final class can contain abstract method ? N
- Example-

2.Member level modifiers:-

- **default members** – If a member declared as default then we can access that member only with in the same package (any other package can’t access that member).
- **private members** – if a member is private then we can access that member only with in the same class (from outside , we can’t access).
- **protected members** – if a member declared as protected then we can access that member anywhere with in the same package but only in child class of outside package.
- **public members** – if a member declared as public then we can access that member from anywhere but the corresponding class should be public also then only access.

Question?

- private abstract is useless for method why?

NOTE -

The most restricted access modifier is private and the most

accessible modifier is public.

[private <default<protected <public]

- For method (public)
- For data member/ variable (private)

Access Modifiers

Public

Class, methods, variables and constructors can be accessed from any other class.

Private

Methods, variables and constructors can only be accessed within the declared class.

Protected

Methods, variables and constructors are declared protected in a superclass can be accessed only by the subclasses.

Default

No modifier required. Access class, variables, method in same package but not from outside.

Visibility	private	Default	protected	public
With in the class	Yes	Yes	Yes	Yes
From child class of same package	No	Yes	Yes	Yes
From non-child class of same package	No	Yes	Yes	Yes
From child class of outside package	No	No	Yes(we use child)	Yes
From non-child class of outside package	No	No	No	Yes

Java Package

- A java package is a group of similar types of classes, interfaces and sub-packages.
- Package in java can be categorized in two form**, built-in** package and **user-defined** package.
- There are many built-in packages such as java, lang, awt, swing, io, util, sql etc.

Advantages

- Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- Java package provides access protection.
- Java package removes naming collision.
- Better organization
- Example- all classes and interface which perform input and output operation are stored in **java.io** package.

Some important point

- The word which first letter is capital i.e class of program.
- The word which first letter is small and end of function symbol () ie method of program
- The class or method contain more than one word then each word is starting from capital letter
- The word which all letter are small is called keywords
- Each package in java have a unique name.
- Syntax

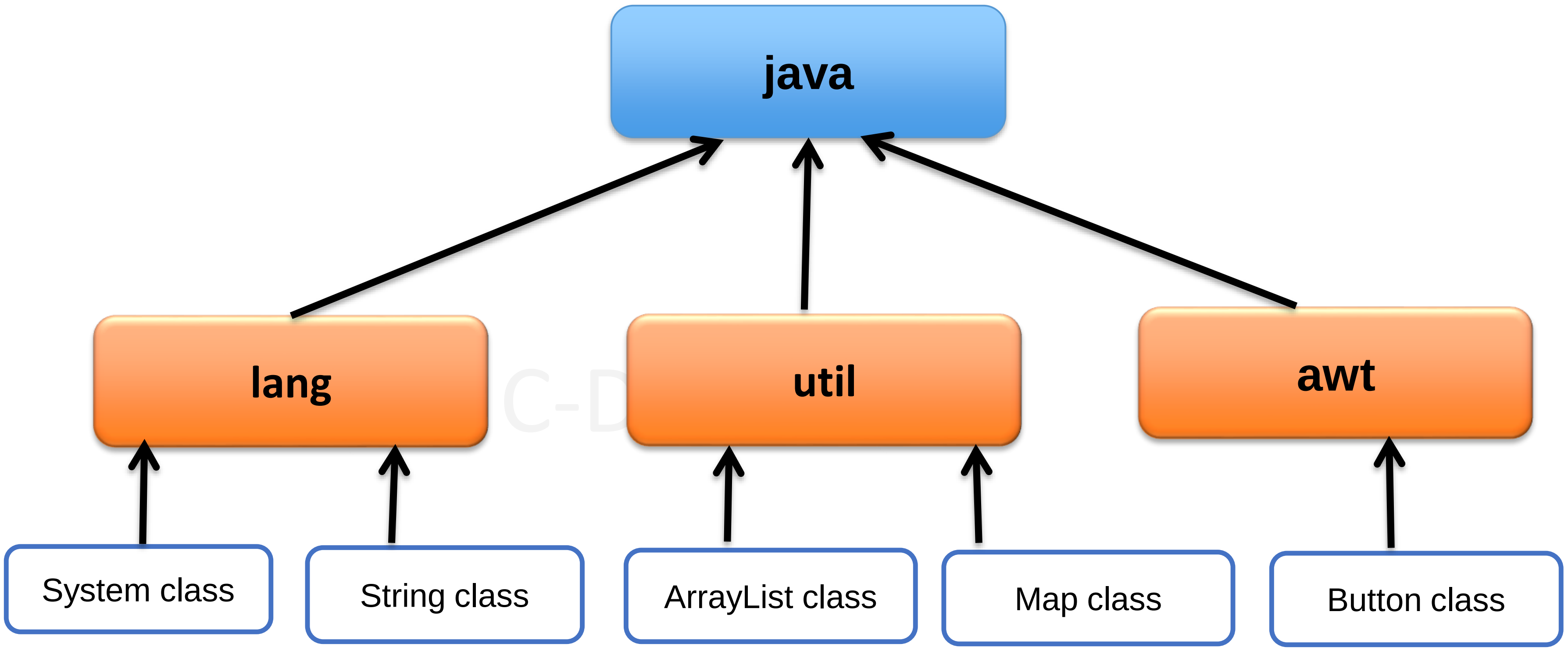
```
package  packagename;
```

- Ex- package addition;

Built-in Packages

Package	Description
java.lang	Fundamental classes (automatically imported)
java.util	Utility classes (e.g., ArrayList, Scanner)
java.io	Input-output classes
java.sql	Database access
javax.swing	GUI components

Example


java
lang
util
awt
System class
String class
ArrayList class
Map class
Button class

User defined Packages

- User can also create their own packages. They are called user defined packages.
- **package** keyword is used to create a package
- **package** keyword at the top of the Java file.

Naming convention of package

Use internet domain name

Ex – package com.test.cdac;

    package com.software.new;

    package com.loan.Amount;

- In java prg , the first non-comment statement should be package statement (if it is available) , otherwise we get error.
- There can be only one package statement i.e more than one not allowed in one java prg.

Accessing protected variable and method outside the package

- Using parent and child relationship
- Example -

Import Statement in Java

- To access classes and interfaces from other packages.
- There are two types of import statement :-

1. **Explicit class import** – It is recommended to use because it improve the readability of the code

```
import  packagename.subpackagename.classname;
```

Example- import java.util.ArrayList;

2. **Implicit class import** - reduce the readability of the code

```
import  packagename.subpackagename.*;
```

```
Ex- import java.util.*;        Date d = new Date ();
    import java.sql.*;         SOP(d.getClass().getName();
```

Ambiguity because date I present in both package

Static Import

- Allows calling static members without class name.

Type of static import

1. Explicit static import

   Syntax – import static package.classname.staticmember;

   Example- import static java.lang.Math.sqrt;

2. Implicit static import

   Syntax - import static package.classname.*;

   Example- import static java.lang. Math.*;

Example

```
import static java.lang.Math.*;

public class Test {
    public static void main(String[] args) {
        System.out.println(sqrt(16));  // instead of Math.sqrt(16)
    }
}
```

THANK YOU!!