Private Interface Method

Java 7 and Java 8

Default methods

Static method

Interface Myinterface{

void method1();

default void method2(){

sysout(“common code”)

Sysout(“test code”)

}

default void method3(){

sysout(“common code”)

Sysout(“test code”)

}

}

10 classes implemets MyInterface

Interface MyPerCombi

Permutation –n!/(n-r)!

Combinamtion ----- n!/r!\*(n-r)!

Try -with resources in Java 9

Autoclosable ----- java7

PrintWriter pr=newPrintWriter(“kdhjkh”);

try(pr)

{

}

Rule : pr has to be final or effectively final

Effectively final ----- the variable is not declared as final, but once it is initiallised the it should not be modified.

}

Collection factory methods

Mutable and Imutable objects

Modifiable collection and nonmodifiable

List<String> list=new ArrayList<>();

List.add(“aaa”);

List.add(“bbb”);

List list1=Collections.unmodifiableList(list);

list.add(“cccc”)

other way

Myclass[] e=new MyClass[]{new MyClass(1),new MyClass(2) };

List<Myclass> list=Arrays.asList(e);

Drawbacks

Changes in array will be reflected in the list

List.add(4) ///// generate UnsupportedOperationException

Set<String> set = Collections.unmodifiableSet(new HashSet<String>() {{

add(“v1"); add(“v2"); add(“v3");

}});

Stream.of(“v1", “v2", “v3")

.collect(collectingAndThen(toSet(), Collections::unmodifiableSet));

Factory methods:

12 factory methods

List, Set and Map interface

List.of()

List.of(Ele);

List.of(ele1,ele2);

List.of(ele1,ele2,………….,ele10)

List.of(Elements….ele) ///not performance efficient

Values based object

List<String> lst1=List.of(“aaa”,”bbb”);

List<String> lst2=List.of(“aaa”,”bbb”);

MyClass ob=lst1.get(0);

Set<String> s=Set.of(“aaa”,”bbb”);

Set<String> s=Set.of(“aaa”,”bbb”,”aaa”); //duplicate object IllegalArgumentExpection

You cannot add null;

Try using Set.of

Try adding duplicate values.

Memory Efficient

Set.of(“aaa”,”vvv”)

Set<String> s=new HashSet<>(3);

Number of objects ceated- Set, hashmap,array , objects data

Collections.unModifibaleSet()

Jshell

It is a tool to try for small code snippet.

REPL

Class Helloworld{

Public static void main(String[] args)

{

}

}

MultiRelease jar file

2 to 3 year

After every 6 months

Java 8 API

MyClass{

myfunction(){

}

}

Java 9

MyClass{

myfunction(){

}

}

JPMS

rt.jar modules

module-info.jaa

module name{

requires

opens

exports

provider

use

}

Advantages

Private classes can be accessed by reflection

Runtime Noclassdef found

No encapsulation

Package p1

Public class A{}

Class B{}

Package p2

Import p1.A;

Public class MyClass{

A ob=new A();

Class c=Class.forName(“p1.B”);

Field[] farr=c.getFields();

}

Module

It is packaging multiple packages along with configuration file

module-info.java{

requires <module name>

exports <packages which are visible to other module>

opens <packages are visible to other modules and reflection>

uses <services provided by other modules>

provides <services to other module> with <service implementiions>

version <value>

}

Multiple packages in the module

Java.base --- implicitly added

Java.logging ----explicitly in the configuration file

Module name should be universely unique

Packages ---by default hidden

Circular dependency is not not allowed

Dependency verification happens at startup

Execution speed also has increased

Java --cp jar1,jar2,jar3 TestClass

A jar1

Class A---- jar2

Java –module-path jar1;

Types of modules

* Unnamed modules
* Automatic modules
* System module
* User defined modules

JPMS

Java.base java.logging java.sql

Jdk modules

Jdk.incubator

Jdk.httpserver

List of modules

Java --list-modules

Rules for module name

1. Should not end with number
2. Inbetween . and – are allowed

Configuration file

Module-info.java

module com.demo{

requires java.logging; // this is required bu current module

requires transitive mymodule // this shoulbe available to all where current module is accessible

require static moduleA /// module is required only at compile time

exports pack1;

exports pack2 to moduleB

}

Execute modular application

Java –module-path or -p

-m or –module

If we place modular jars with -cp

-cp -p

Unnamed module

Non modular jar with -p

Automic module --🡪 jar pack1,pack2,pack3

Manifest.mf

Manifest-vesrion

Automatic-Module-Name : Mymodule