Strictfp

* Usually result of float varied from platform to platform.
* If we want platform independent result for floating point arithmetic the we should go for “strictfp” modifier.
* If a method declared as “strictfp” all ex.
* We can declare abcstract cobmaatjkion “strictfp”
* Comparision :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Visibility | Private | <default> | Protected | Public |
| Within the same class | Ok | Ok | Ok | Ok |
| From child class of the same package | X | Ok | Ok | Ok |
| From non-child of the same package | X | Ok | Ok | Ok |
| From child class of the outside package | X | X | Ok | Ok |
|  |  |  |  |  |

* The most restricted access modifier is private whereas the most accessible access modifier is public.
* Private<default<protected<public
* Recommended modifier for variables is “private” whereas recommended modifier for methos is “public”.

Final variables:

1. Final instance variables:

* If the value of a variable is varied from object from object such types of variables are called instance variables.
* For every object a separate copy of instance variable will be created.
* For instance variables JVM will provide default variables and we are not required to perform initialization explicitly.
* Example:
* Public class Test{
* Int x;
* Public static void main(String[] args){
* Test t=new Test();
* Sopln(t.x);//0
* }}
* If the instance variable declared as final then JVM won’t provide default value, compulsory we should perform initialization explicitly whether we are using that variables or not.
* Example:

Public class Test{

Final int x;//CE: variable x might not have been initialized.

}

* Rule: for final instance variables compulsory we should perform initialization before constructor completion that is the following are various places for this:

1. At the time of declaration

Class Test{final int x=0;}

1. Inside instance block

Class Test {

Final int x;

{ x=0;}

}

1. Inside constructor

Class Test{

Final int x;

Test(){

X=0;

}}

* These are the only possible places to perform initialization for final instance variables, if we are trying to perform initialization anywhere else we will get compile time errors.
* Example:

Class Test{

Final int x;

Public void m1(){x=0;}//CE: cannot assign a value to a final variable x

}

Final static variables:

* If the value of a variable not varied from object to object then it is not recommended to declare that variable as instance variable we have to declare such type variables at class level by using static modifier.
* In the case of instance variables for every object a separate copy will be created but in the case of static variable a single copy will be create at class level and shared by every object of that class.
* For static variables JVM will always provide default values and we are not required to perform initialization explicitly. For example:

Class Test{ static int x;

P s v m(S[] args){

Sopln(x);//0

}}

* If the static variables declared as final then JVM won’t provide default values, compulsory we should provide initialization explicitly whether we are using or not.

Example:

Class Test{ final static int x;}//variable x might not have been initialized.

* Rule : for final static variables compulsory we should perform initialization before class loading completion ie. The following are the various places for this:

1. At the time of declaration

Class Test{ final static int x=10;}

1. Inside static block

Class Test{ final static int x;

Static{ x=10;}

}

* These are the only possible places to perform initialization for final static variables. If we are trying to initialization anywhere else then we will get compile time error.

Example:

Public class Test{ final static int x;

Public static void main(String[] args){

X=10;//CE: cannot assign a value to final variable x

}}

Final local variables:

* Sometimes to meet temporary requirements of the program we can declare a variable inside method or block or constructor, such types of variables are called local variables or temporary variables or stack variables or automatic variables.
* For local variables JVM won’t provide default values, compulsory we should perform initialization explicitly before using those variables that is if we are not using the local variables then it is not required to perform initialization explicitly.

Example:

|  |  |
| --- | --- |
| Public class Test{  P s v m(String[]){  Int x;  Sopln(“hello”);}}//ok | Public class Test{  P s v m(String[]){  Int x;  Sopln(x);}}//CE variable might not have been initialized. |

* Class Test{

Public static void m1(Integer I){

Sopln(“Autoboxing”);}

Public static vdoi m1(int…I){

Sopln(“var-arg”);}

Public static void main(String[] args){  
 int x=10;

M1(x);}}  
=> auto boxing dominates var-args.

* In general var-arg methods will get least priority that is if no other method matched then only var-arg method will get chance. It is exactly same as default case inside switch.
* While resolving overloaded method compiler will always give the precedence in the following order:

1. Widening
2. Auto boxing
3. Var –args methods

Case 4:

* Widening followed by auto boxing not allowed in java whereas auto boxing followed by widening is allowed. For example:

Long l=10;//CE

Long l=10l;//auto boxing

lang l0;//widening

* Case 5:  
  => Int to Integer ---object

Case 6:

Object o=10;//int to Integer and Integer to Object

Number n=10;//int to Integer and Integer to Nuber

Object o=10l;//long to Long and Long to Object.

* Which of the following are correct?

1. Int i=10;//ok
2. Integer i=10;//auto boxing
3. Int i=10L;//CE: possible loss of preecion
4. Long l=10L;/ok
5. Long l=10;//incompatible types
6. Object o=10;//autoboxing followd by widening
7. Double d=10;
8. Double D=10;
9. Double d-10;
10. Number n=100;